

JIGSAWS AND FASTER FRACTAL PICTURES

by

Lindsey Menzies

A thesis presented to the University of Glasgow Faculty of Science
for the degree of Doctor of Philosophy.

Department of Mathematics,

June 1997.

© Lindsey Menzies June 1997

ProQuest Number: 13834268

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest 13834268

Published by ProQuest LLC (2019). Copyright of the Dissertation is held by the Author.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

Table of Contents

Preface	iv
Abstract	v
Chapter 1 Introduction	1
Chapter 2 An Introduction to Iterated Function Systems	
2.1 Background	5
2.2 Metrics and The Fixed Point Theorem	7
2.3 Iterated Function Systems	12
2.3.1 Classification of Affine Maps.....	14
2.3.2 Finding the IFS Code.....	18
2.4 The Collage Algorithm	21
2.5 The Random Iteration Algorithm (RIA).....	23
2.5.1 An Addressing Scheme	26
2.5.2 Limitations of the RIA.....	29
2.6 The Adaptive Cut Method (ACM).....	31
Chapter 3 The Graphical Algorithm	
3.1 Definition of the Algorithm	34
3.2 The $N : 1$ Property	41
3.3 Introduction to Jigsaws.....	43

Chapter 4 Properties of Jigsaw Pieces

4.1 Holes in Jigsaw Pieces.....	49
4.2 Rectangular Jigsaw Pieces	52
4.3 Bridges in Jigsaw Pieces	58
4.4 Gaps in Jigsaw Pieces	64

Chapter 5 Improving the Graphical Algorithm

5.1 Finding the Jigsaws	75
5.1.1 Simple Jigsaws	76
5.1.2 More Complicated Jigsaw Pieces	81
5.2 Using the Jigsaws	83
5.2.1 Recording the Marks	84
5.2.2 Examples	85
5.3 Reduced Jigsaws	91
5.4 Using Less Than N Jigsaws	97
5.5 To Sort or Not To Sort.....	101

Chapter 6 Maps with Semi-Regular Jigsaws

6.1 Translation Symmetries	105
6.2 Using Semi-Regular Jigsaws	116
6.3 Finding a Building Block of Least Dimensions	124

Chapter 7 Irregular Jigsaws

7.1 Scanning Around the Starting Point.....	135
7.2 Scanning Along Rows.....	143
7.3 Reflecting Maps in a Horizontal or Vertical Line	155

Table of Contents

Chapter 8 Approximating Irregular Jigsaws

8.1 Approximation by Nearest "Small" Rational..... 158
8.2 Rational Approximation..... 161
8.3 Limitations of Approximation..... 172

Chapter 9 Discussion

9.1 Dealing with Bridges and Gaps..... 184
9.2 Summary and Further Research..... 186

Appendix 190

References 192

Preface

This thesis was submitted to the University of Glasgow in accordance with the requirements for the degree of Doctor of Philosophy.

There are a number of people that I wish to thank for their encouragement and support throughout my years at University. In the Mathematics department, I would like to express my gratitude to my supervisor, Dr Stuart G. Hoggar, not only for his encouragement but also for his invaluable advice and assistance, especially when problems arose. I would also like to thank Professor R. W. Ogden and Professor K. Brown, both of whom held the position of Head of Department over the last four years, for allowing me to study within this department.

Outside of University, I am extremely grateful to my parents and boyfriend for their love and support. Without them, I doubt if I would have completed this thesis.

Finally, I would like to thank the Engineering and Physical Sciences Research Council for their financial support from October 1993 to September 1996.

Abstract

Today it is possible to express a wide range of images as attractors of Iterated Function Systems in the plane. An *Iterated Function System*, or IFS for short, is a finite collection of contractive affine transformations w_1, w_2, \dots, w_N , while the *attractor* \mathcal{A} is the limit under repeated application of the associated collage map $W(E) = \cup_i w_i(E)$ for any compact set E - that is, $W^n(E) \rightarrow \mathcal{A}$ [2, 12, 25, 26]. Since only $6N$ real numbers, known as the IFS *code*, are required to store the image, IFS's are now being widely used as a method of *image compression* [3, 4, 9] - leading to the need for algorithms which produce the attractor \mathcal{A} quickly on a computer screen.

In this thesis, we study one such algorithm - the *Graphical Algorithm*, or GA [5] - whose main characteristic is that maps are applied to points only after their coordinates have been rounded to integers. By introducing the concept of *jigsaws* of maps - where the *jigsaw piece* of an integer point (u, v) is the set of all integer points (x, y) such that $w(x, y)$ rounds to (u, v) - and studying the properties of such sets, we reduce the time taken to produce an approximation to the attractor by up to a factor of N , making the GA as accurate as the *Adaptive Cut Method* (ACM) [11, 25, 26], and a frequently faster option than the popular *Random Iteration Algorithm* (RIA) [1, 2].

Chapter 1

Introduction

There are many different ways of obtaining fractal images on computer screen; in this thesis, we concentrate on one such method - the *Graphical Algorithm* [5] - and look at ways of improving its performance. We will consider fifty distinct images, obtained from references [2, 5, 12, 19, 26, 28]. These are shown in Figure 1.1.

In Chapter 2 we give a brief introduction to *Iterated Function Systems* (IFS's), beginning with a number of definitions relating to metric spaces and a classification of affine maps. We define the *code* and *attractor* of an IFS, and consider three ways of screening the attractor, namely the *Collage Algorithm*, *Random Iteration Algorithm* (RIA) and *Adaptive Cut Method* (ACM). During the chapter, we will show that each of these methods has limitations - and conclude that, ideally, we would like to have an algorithm which combines the speed of the RIA with the accuracy of the ACM.

We describe an algorithm in Chapter 3 - the aforementioned Graphical Algorithm (GA) - which, as presently defined, is comparable with the ACM, in that it is slower but more accurate than the RIA. From a particular starting point z_0 , the GA produces the attractor by applying each of the N maps of the IFS to z_j and plotting the nearest integer point. These resultant integer points are placed in a store - from which the next point z_{j+1} is chosen - with the process terminating when the store is empty. Examples 3.4 and 3.5 show that the ratio of calculated points to unique points is always $N : 1$ for this algorithm. Investigating why this is the case leads us to define *jigsaws of IFS maps*, where two points are in the same map i jigsaw piece if and only if they are sent to the same point under map i . It is these jigsaws which

Chapter 1

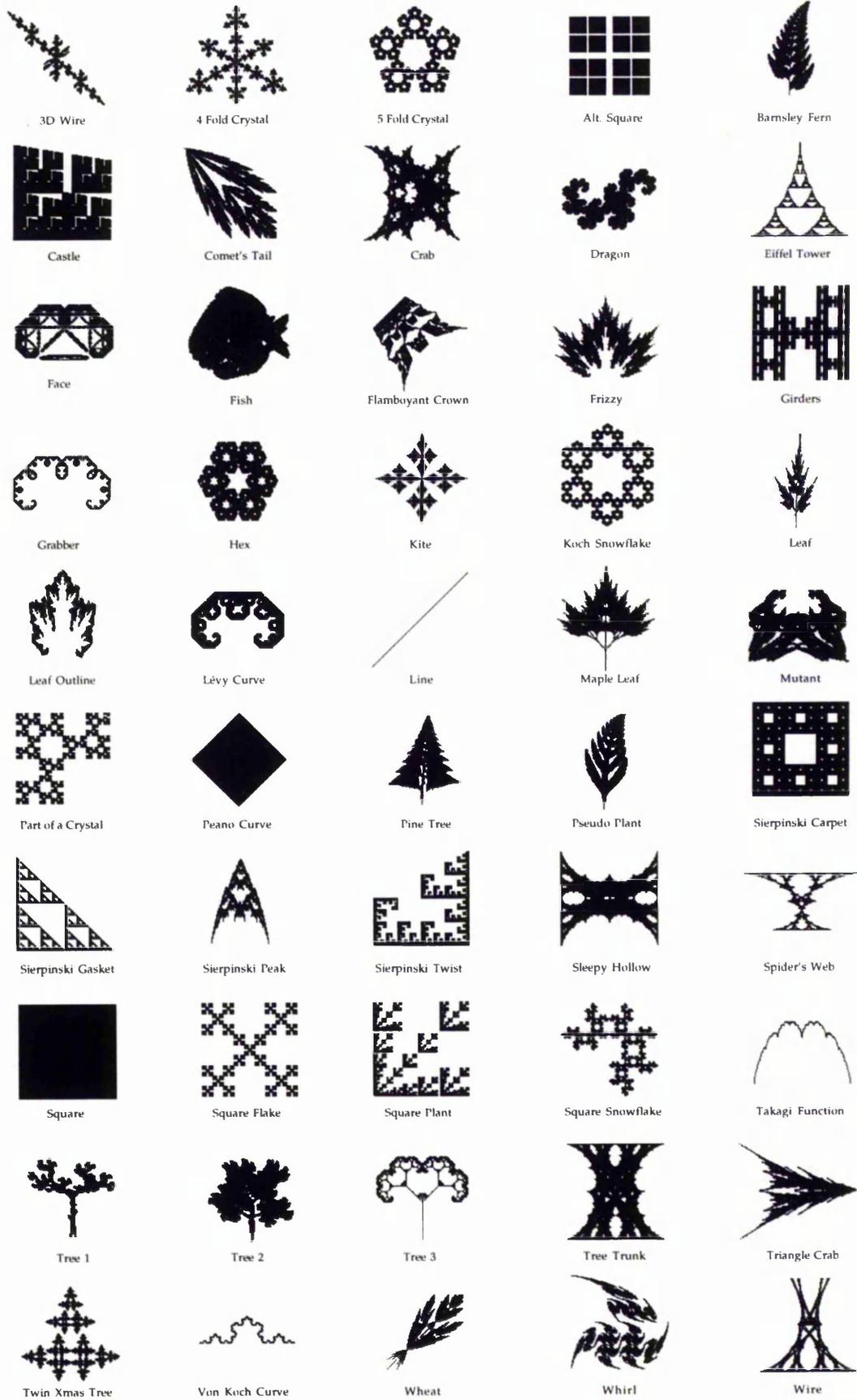


Figure 1.1 The Attractors of our Fifty IFS's.

enable us to develop methods to improve the speed of the GA so that it compares favourably with that of the RIA.

Chapter 4 is concerned with the various properties which the pieces of these jigsaws exhibit. We show that the jigsaw pieces cannot contain holes and that, for the majority of the maps of our fifty IFS's, a given jigsaw piece cannot contain two pixels which are either not joined, or are joined only at a vertex. We also show that, in many cases, all of the jigsaw pieces are rectangular. For each of the properties discussed, we give conditions on the IFS code entries which allow us to infer that the property holds.

In Chapter 5 we consider simple IFS maps, whose jigsaw pieces are all rectangular and all the same size. We show that in such cases it is an easy process to determine from a given starting pixel which other pixels are in the same jigsaw piece, enabling us to ensure that the map is only applied to one pixel of the jigsaw piece. Doing this means that duplication of attractor points can be reduced (indeed, in some cases, completely removed) and the time taken to produce the attractor can also be reduced. We then move on to show that further improvements are possible both by using what we shall call *reduced jigsaws*, and by using less than N jigsaws (the same jigsaw acting for several maps).

Chapter 6 continues work begun in the previous chapter, but looks at more complicated maps, where the pieces are not all of the same size and shape. We consider cases where the jigsaws are built from small sub-jigsaws (*building blocks*) and note that, provided these blocks are small enough, we can use them to help us to improve the GA for such maps. In Section 6.2 we give a general scheme for finding the reduced jigsaw in cases where the building block contains only whole jigsaw pieces, while in Section 6.3 we give a method for finding the dimensions of the smallest possible building block.

Chapter 7 is concerned with the problems which we encounter when we try to make use of jigsaws for which the building blocks are too large - which we call *irregular jigsaws*. We consider several methods for finding the pieces of such jigsaws, using a number of the Theorems of Chapter 4 in order to optimise their performance. In Section 7.2 we define a procedure which enables us to improve the performance of the GA for a specific class of IFS - where the majority of jigsaw pieces contain a relatively large number of attractor points. By considering a number of examples we show that, in addition to the irregular cases, this method enables us to achieve even better results for several of the IFS's already considered in earlier chapters. We end the chapter by noting that, since this procedure cannot be applied to the majority of IFS's, we must search for another method - and conclude that in order to improve the GA for such maps, we must alter the maps slightly.

Such alterations are the subject of Chapter 8, where we consider approximating the code entries by 'small' rational numbers, so that we may find small building blocks for the jigsaws. We consider performing this approximation by inspection, but note that this does not always lead to the best choice, leading us to use *rational approximation*, which we define in Section 8.2. We notice that, in many cases, using approximation by small rational makes very little difference to the appearance of the attractor, and explain why this is the case. In Section 8.3 we consider IFS maps which contain a rotation, and show that unless the angle of rotation is a multiple of $\pi/4$, approximation easily causes fundamental changes in the appearance of the attractor. We conclude that, while approximation is not feasible in some cases, it may be used, together with the methods of Chapters 5 and 6, to make at least partial improvements to the GA for every one of our fifty IFS's.

Chapter 9 contains a summary of the results of this thesis, together with comments about the methods used and a look to the future.

Chapter 2

An Introduction to Iterated Function Systems

In this chapter, we give a number of definitions which provide the necessary background to Iterated Function Systems. We then move on to describe several previously established methods of producing fractal pictures, and discuss their advantages and failings.

2.1 Background

One of the most important ideas in the study of Iterated Function Systems is the concept of 'distance between sets' or, more specifically, 'distance between pictures', where we may think of a picture as a set of pixels in the plane. In this section, we will define the axioms which any distance function must satisfy, and then define the *Hausdorff distance* between sets together with the space in which such distance calculations will be carried out. As we shall see in Section 2.3, the Hausdorff distance can also be used to show sets, and therefore pictures, tending towards a limit - and in fact it is this idea which is the key to producing fractal pictures by way of Iterated Function Systems. References which are relevant both to this section, and Section 2.2, include [2, 7, 8, 12, 25, 26].

Definition 2.1 A *metric space* is a pair (X, d) , where X is a nonempty set and d is a real function on pairs of points, called *distance*, or a *metric*, which satisfies the following axioms:

- (i) *Positivity*: $d(z_1, z_2) \geq 0 \forall z_1, z_2 \in X$, with equality if and only if $z_1 = z_2$.
- (ii) *Symmetry*: $d(z_1, z_2) = d(z_2, z_1) \forall z_1, z_2 \in X$.
- (iii) *Triangle Inequality*: $d(z_1, z_3) \leq d(z_1, z_2) + d(z_2, z_3) \forall z_1, z_2, z_3 \in X$.

Definition 2.2 A sequence $\{z_n\}$ in (X, d) is *Cauchy* if $\forall \varepsilon > 0, \exists k \in \mathbf{N}$ such that $p, q > k \Rightarrow d(z_p, z_q) < \varepsilon$. Then (X, d) is *complete* if every Cauchy sequence in X is convergent. Further, a subset $Y \subset X$ is *compact* if every sequence in Y has a convergent subsequence. Note that in the plane, Y is compact if and only if Y is bounded and closed.

At this junction, it is important to note that while compactness is a stronger condition, and much of our work will be carried out in compact spaces, the *Fixed Point Theorem* (Theorem 2.10) - which is the key result of this chapter - requires only completeness, and therefore the idea of Cauchy sequences is important.

Notation 2.3 Let (X, d) be a complete metric space. Then we denote by $\mathcal{H}(X)$ the collection of all non-empty compact subsets of X .

Definition 2.4 Let $z \in X$ and $A, B \in \mathcal{H}(X)$. The distance from the point z to the set B , $d(z, B)$, is given by $d(z, B) = \min\{d(z, b) : b \in B\}$, while the distance $d(A, B)$ from the set A to the set B is given by $d(A, B) = \max\{d(z, B) : z \in A\}$. From these, we define the *Hausdorff distance* between A and B by

$$h(A, B) = \max\{d(A, B), d(B, A)\}. \quad (2.1)$$

It is easy to verify that this function satisfies the axioms of Definition 2.1 and is therefore a metric. Note, in particular, that the Hausdorff distance, $h(A, B)$, is zero if and only if A and B are identical.

We will see shortly how the Hausdorff distance is used in practice to find the distance between a 'perfect' image and various approximations to it, but it is clear from (2.1) that the actual calculation, and indeed the specific value, of $h(A, B)$

depends on the metric d . It is therefore appropriate at this stage to briefly discuss the choice of metrics.

2.2 Metrics and The Fixed Point Theorem

In this section we discuss the choice of metric over one particular space - Euclidean 2-space, or \mathbf{R}^2 . We will shortly introduce *contractive transformations*, $f(z)$ - where, for any two points P and Q , the distance between $f(P)$ and $f(Q)$ is strictly less than the distance between P and Q - and we will show in Examples 2.7 and 2.8 that a transformation being non-contractive with respect to one metric does not mean that it will be non-contractive with respect to every metric. If, however, we can find some metric d which makes a transformation contractive, then the famous *Fixed Point Theorem* of Banach (Theorem 2.10) tells us that the sequence of points $\{f^n(z_0)\}$ tends to a limit in (\mathbf{R}^2, d) . The importance of this will be highlighted in Section 2.3, where we will see that it can be applied to pictures tending to limits, when we consider a picture to be a point in $\mathcal{H}(X)$.

Over \mathbf{R}^2 there are many metrics which can be chosen to measure distance. Let $P = (x, y)$ and $Q = (u, v)$ be two points of the plane. Below, we describe the best known metrics on \mathbf{R}^2 , and Figure 2.1 shows the unit discs $\{z \in \mathbf{R}^2: d(0, z) < 1\}$ for each.

(i) *The lattice metric:* $d_1(P, Q) = |x - u| + |y - v|$.

(ii) *The Euclidean metric:* $d_2(P, Q) = \sqrt{(x - u)^2 + (y - v)^2}$.

(iii) *The maximum metric:* $d_\infty(P, Q) = \max\{|x - u|, |y - v|\}$.

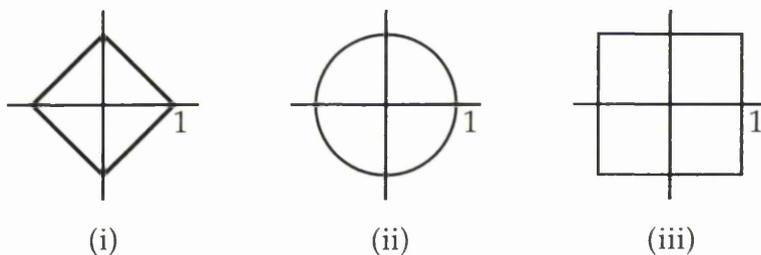


Figure 2.1 The unit discs for the three metrics described above.

Note 2.5 The lattice metric is often used for Hausdorff distance calculations in practice, as it has no need for calculation of square roots, and is therefore much simpler than the Euclidean metric.

Definition 2.6 A transformation $f: X \rightarrow X$ is said to be *contractive* if there exists some number s , $0 \leq s < 1$, such that

$$d(f(z_1), f(z_2)) \leq s d(z_1, z_2)$$

for any two points $z_1, z_2 \in X$. The smallest such number s is known as the *contractivity factor*, *Lipschitz constant*, or simply the *ratio*, of f .

As noted earlier, the contractivity of a transformation often depends on the choice of metric i.e. a transformation can be contractive in one metric but not in another, as the following example illustrates.

Example 2.7 Consider the map $w(x, y) = 0.65(x - y, x + y)$, which consists of scaling by $0.65\sqrt{2}$ followed by rotation by $\pi/4$. Let $P = (0, 0)$, $Q_1 = (1, 0)$ and $Q_2 = (1, 1)$. Then $w(P) = (0, 0)$, $w(Q_1) = (0.65, 0.65)$ and $w(Q_2) = (0, 1.3)$. Then

$$d_1(w(P), w(Q_1)) = 0.65 + 0.65 = 1.3 > 1 = d_1(P, Q)$$

$$d_\infty(w(P), w(Q_2)) = \max\{0, 1.3\} = 1.3 > 1 = d_\infty(P, Q)$$

and therefore w is not a contraction with respect to either d_1 or d_∞ . However, w is a contraction with respect to d_2 : let $P = (x, y)$ and $Q = (u, v)$ so that $w(P) = 0.65(x - y, x + y)$ and $w(Q) = 0.65(u - v, u + v)$. Then

$$\begin{aligned}
& d_2(w(P), w(Q)) \\
&= 0.65\sqrt{((x-y)-(u-v))^2 + ((x+y)-(u+v))^2} \\
&= 0.65\sqrt{((x-u)-(y-v))^2 + ((x-u)+(y-v))^2} \\
&= 0.65\sqrt{2((x-u)^2 + (y-v)^2)} \\
&= 0.65\sqrt{2}d_2(P, Q)
\end{aligned}$$

Thus w is a contraction with respect to the Euclidean metric, with contraction ratio $0.65\sqrt{2} \approx 0.92$. Note that replacing 0.65 by any α in the range $(0.5, 1/\sqrt{2})$ will also give a map which is not contractive with respect to d_1 or d_∞ , but is contractive with respect to d_2 , since the contraction ratio will be $\alpha\sqrt{2}$.

From this example, it may appear that the Euclidean metric is always the best choice of metric as it highlights the contractivity of transformations. However, this is not the case, as the example below illustrates.

Example 2.8 Consider the transformation which scales the vertical coordinate by 0.6 and then rotates the result by 270° .

$$w(x, y) = (0.6y, -x)$$

Let $P = (0, 0)$, $Q = (1, 0)$ so $w(P) = (0, 0)$, $w(Q) = (0, -1)$. Then for any of the three given metrics, $d(w(P), w(Q)) = d(P, Q)$ and therefore w is not contractive with respect to any of these metrics. It is possible, however, to find metrics which make w contractive. Let $P = (x, y)$ and $Q = (u, v)$. Then one such metric is $d(P, Q) = \max\{1.2|x-u|, |y-v|\}$. For since $w(P) = (0.6y, -x)$ and $w(Q) = (0.6v, -u)$, we have

$$\begin{aligned}
d(w(P), w(Q)) &= \max\{1.2|0.6y - 0.6v|, |x - u|\} \\
&= \max\{0.72|y - v|, |x - u|\} \\
&\leq 5/6 \max\{1.2|x - u|, |y - v|\} \\
&= 5/6 d(P, Q).
\end{aligned}$$

Thus w is contractive with respect to this new metric.

Many pictures can be described by a set of plane transformations $\{w_1, \dots, w_N\}$, with the whole picture being covered by the union of the images of the picture. If each of the w_i 's are contractive, then it can be shown that the transformation $W = w_1 \cup \dots \cup w_N$ is also contractive. Then, by the famous Fixed Point Theorem of Banach, which we state and prove below, the sequence $W^n(A_0)$ - defined below - tends towards the fixed point of W - which is the picture itself. In practice this means that, if we can find the w_i 's for a picture, and each of the w_i 's is contractive with respect to some metric d , then we can reproduce the picture. We will shortly define each of these ideas formally, but in the meantime we note that it is this concept which makes the ability to find suitable d of such great importance. For our purposes, d will be the Euclidean metric, unless otherwise stated.

Definition 2.9 Let $f: X \rightarrow X$ be a transformation on (X, d) . Then $f^n: X \rightarrow X$, for $n \geq 0$, is defined by $f^0(z) = z$, $f^1(z) = f(z)$, ..., $f^{(n+1)}(z) = f(f^n(z))$.

Theorem 2.10 (Banach's Fixed Point Theorem) Let $f: X \rightarrow X$ be a contraction mapping on the complete metric space (X, d) with ratio s , $0 \leq s < 1$. Then

- (i) f has unique fixed point c , i.e. there exists exactly one point $c \in X$ such that $f(c) = c$.
- (ii) For any $z_0 \in X$, the sequence $\{f^n(z_0)\}$ converges to the fixed point.
- (iii) After n iterations, we have the following estimates for the distance from the fixed point.

$$d(z_n, c) \leq \frac{s}{1-s} d(z_{n-1}, z_n) \quad (n \geq 1) \quad (2.2)$$

$$d(z_n, c) \leq \frac{s^n}{1-s} d(z_0, z_1) \quad (n \geq 0) \quad (2.3)$$

Proof Let $z_0 \in X$, and let $z_n = f^n(z_0)$. Then for integers $m > n \geq 0$ we have

$$\begin{aligned}
 d(z_n, z_m) &= d(f^n(z_0), f^m(z_0)) \\
 &= d(f^n(z_0), f^n(f^{m-n}(z_0))) \\
 &\leq s^n d(z_0, f^{m-n}(z_0)) && \text{(definition of ratios)} \\
 &= s^n d(z_0, z_{m-n}) \\
 &\leq s^n \{d(z_0, z_1) + d(z_1, z_2) + \dots + d(z_{m-n-1}, z_{m-n})\} \\
 &&& \text{(triangle inequality)} \\
 &\leq s^n d(z_0, z_1) \{1 + s + s^2 + \dots + s^{m-n-1}\} \\
 &\leq s^n d(z_0, z_1) \{1 + s + s^2 + \dots\} && \text{(infinite series)} \\
 &= s^n d(z_0, z_1) / (1 - s) && \text{(sum of geometric series)}
 \end{aligned}$$

This final expression can be made arbitrarily small by taking m, n sufficiently large. Hence the sequence $\{z_n\}$ is Cauchy and, since X is complete, it has a limit $c \in X$ (that is $z_n \rightarrow c$ as $n \rightarrow \infty$). Since f is contractive, and therefore continuous, we have

$$f(c) = f(\lim_{n \rightarrow \infty} f^n(z_0)) = \lim_{n \rightarrow \infty} (f^{n+1}(z_0)) = c$$

showing that c is a fixed point. To show that this point is unique, suppose that there were two fixed points c and c' . Then $c = f(c)$ and $c' = f(c')$ so

$$d(c, c') = d(f(c), f(c')) \leq s d(c, c')$$

From this, $(1 - s)d(c, c') = 0$, with $(1 - s) \neq 0$, and so $d(c, c') = 0$ implying $c = c'$. Hence c is the unique fixed point of f .

To prove (2.2) we use similar arguments to the above, with $m > n \geq 1$.

$$\begin{aligned}
 d(z_n, z_m) &\leq d(z_n, z_{n+1}) + \dots + d(z_{m-1}, z_m) \\
 &\leq d(z_{n-1}, z_n) (s + s^2 + \dots + s^{m-n}) \\
 &\leq d(z_{n-1}, z_n) s / (1 - s)
 \end{aligned}$$

Now let $m \rightarrow \infty$ so that $z_m \rightarrow c$. Then, since d is continuous in each variable, $d(z_n, z_m) \rightarrow d(z_n, c)$. Since the right hand side of the inequality remains constant, we have proved (2.2), and by noting that

$$d(z_{n-1}, z_n) \leq s d(z_{n-2}, z_{n-1}) \leq \dots \leq s^{n-1} d(z_0, z_1)$$

we obtain (2.3) from (2.2). This completes the proof.

2.3 Iterated Function Systems

We now move on to define Iterated Function Systems and their associated attractors. Definitions 2.11 and 2.13 are key definitions, which will be referred to throughout.

Definition 2.11 An *iterated function system*, or *IFS* for short, is a collection of a complete metric space (X, d) , together with a finite set of contractive mappings $w_i: X \rightarrow X$ with respective contractivity factors s_i ($1 \leq i \leq N$). Such an IFS is often denoted $\{X: w_1, w_2, \dots, w_N\}$, or simply $\{X: w_{1-N}\}$. If all s_i are equal then the IFS is said to be *uniform*.

Definition 2.12 The *collage map* $W: \mathcal{H}(X) \rightarrow \mathcal{H}(X)$ associated with an IFS $\{X: w_{1-N}\}$ is given by

$$W(E) = w_1(E) \cup w_2(E) \cup \dots \cup w_N(E)$$

for some set $E \in \mathcal{H}(X)$. Then W is a contractive mapping with contractivity factor $s = \max\{s_1, \dots, s_N\}$. The map W is called the *collage map* since it is formed as a union, or *collage*, of transformed copies of E . However, it is often referred to as the *Hutchison operator*, after J. E. Hutchison, who proved that W is contractive with respect to the Hausdorff distance [17].

Definition 2.13 The *attractor* of an IFS is the unique set \mathcal{A} , given by the Fixed Point Theorem (Theorem 2.10), for which $W^n(E) \rightarrow \mathcal{A}$ for every starting set $E \in \mathcal{H}(X)$. The term attractor is chosen to suggest the movement of E towards \mathcal{A} under successive applications of W . By Theorem 2.10, \mathcal{A} is also the unique set in $\mathcal{H}(X)$ which is left unchanged by W , $W(\mathcal{A}) = \mathcal{A}$, and is therefore often referred to as the *invariant set* of the IFS.

Throughout this thesis, X will be either the plane \mathbf{R}^2 or some compact subset of \mathbf{R}^2 and we will often refer to an IFS simply as $\{w_{1-N}\}$.

Definition 2.14 An *affine transformation* $w: \mathbf{R}^2 \rightarrow \mathbf{R}^2$ is a composition of a linear map and a translation. It may be represented as

$$w(z) = Az + t$$

or

$$w(z) = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}$$

where $a, b, c, d, e, f \in \mathbf{R}$. The 6-tuple (a, b, c, d, e, f) is known as the *code* of w , while the *code of an IFS* is a table whose rows are the respective codes of w_1, \dots, w_N .

Note The affine transformation w with code (a, b, c, d, e, f) has unique fixed point (x, y) given by

$$x = \frac{-e(d-1) + bf}{(a-1)(d-1) - bc}, \quad y = \frac{-f(a-1) + ce}{(a-1)(d-1) - bc}, \quad (2.4)$$

and ratio s (with respect to the Euclidean metric) given by

$$s = \sqrt{\frac{p + \sqrt{p^2 - 4q}}{2}} \quad \text{where } p = a^2 + b^2 + c^2 + d^2 \text{ and } q = (ad - bc)^2. \quad (2.5)$$

Note that (2.5) is the 2-dimensional case of the general eigenvalue formula for s ,

$$s = \sqrt{\max\{|\lambda_i| : \lambda_i \text{ eigenvalue of } A^T A\}}.$$

For more information, see [25, 26].

2.3.1 Classification of Affine Maps

Before considering methods for producing the attractors of iterated function systems, it is important to consider what a map specified by a row of the IFS code table can do to a point (x, y) (or, indeed, to a set of points).

Recall from basic geometry that every distance preserving map (or *isometry*) may be written as a composition of one or more of the following three basic types:

- (i) Translation $T_a : z \rightarrow z + a$
- (ii) Rotation $R_0(\theta)$ about the origin, through signed angle θ , with anticlockwise as positive direction, and
- (iii) Reflection R_m in a line m which is either of the coordinate axes.

The matrices for rotation about the origin by angle θ , and reflection in the x -axis are given below.

$$R_0(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \quad R_{x\text{-axis}} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Now, although our definitions are of rotation about the origin, and of reflection in one of the coordinate axes, we may also need to consider rotation about some point P , or reflection in some general line. Informally speaking, to find the

matrices for such isometries, we map everything to the standard axes and origin, perform the rotation or reflection, and then map the transformed object back to the required axes and origin. For example, if we want to perform rotation about the point $(3, 4)$ by angle θ , the transformation is given by

$$T_{(3,4)}R_0(\theta)T_{-(3,4)}(x, y) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x-3 \\ y-4 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \end{bmatrix}.$$

In general, however, we will only need to consider rotation about the origin and reflection in one of the coordinate axes.

Now, as we stated in Definition 2.11, the maps of an Iterated Function System must be contractive - and therefore no isometry g can be an IFS map on its own. However if, before applying g , we first scale x and y by values r_1 and r_2 respectively, with $0 \leq r_1, r_2 < 1$, then the composition of this scaling with g will indeed be a contractive map. Such a scaling - which we call a *coordinate scaling* - has matrix

$$A = \begin{bmatrix} r_1 & 0 \\ 0 & r_2 \end{bmatrix}.$$

Note that if $r_1 = r_2$ then the scaling is said to be *uniform*. We will see shortly that it is possible to give a general form for the matrix of an IFS map, but first we define one final affine transformation, which is not an isometry.

Definition 2.16 A *shear* along the x -axis with parameter α is the transformation $(x, y) \rightarrow (x + \alpha y, y)$, in which points are moved parallel to the x -axis in proportion to their signed distance above it, α being the constant of proportionality. Note that, as with isometries, a shear must be combined with scaling before it may be considered as an IFS map as it is not in itself contractive. Figure 2.2 shows the

effect that uniform scaling with scale factor 0.5 followed by shear with parameter 1 along the x -axis has on a simple rectangle.

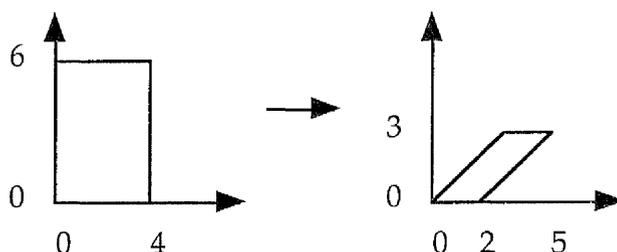


Figure 2.2 The effect of uniform scaling by 0.5 followed by shear with parameter 1.

Note As with the isometries, we may need to consider a shear with respect to some other axis. We can find the matrix of such a map in much the same way as we did for the rotation through a point other than the origin. Suppose, for example, that we wish to perform a shear along a line which makes angle θ with the x -axis. Then the matrix is given by

$$\begin{aligned} R_{-\theta}S_{\alpha}R_{\theta} &= \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & \alpha \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \\ &= \begin{bmatrix} 1 + \alpha \sin\theta \cos\theta & \alpha \cos^2\theta \\ -\alpha \sin^2\theta & 1 - \alpha \sin\theta \cos\theta \end{bmatrix} \end{aligned}$$

Figure 2.3 gives an example of a shear along a line which makes angle 30° with the x -axis, with the bicycle on the left being the original. Further details may be found in [12].

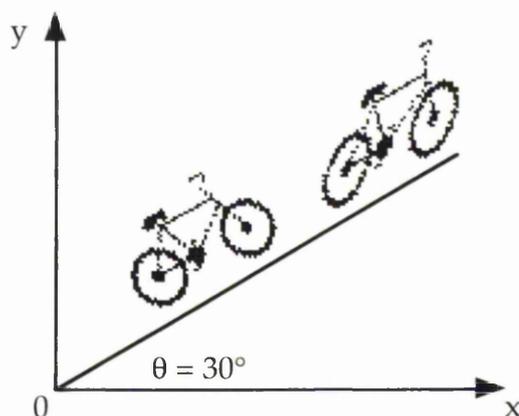


Figure 2.3 Shear along a line making angle 30° with the x-axis.

We are now in a position to give a general statement about the matrices of IFS maps, and to give without proof the general form of such matrices.

Theorem 2.17 *Let $w(z) = Az + t$ be an affine transformation with $A \neq 0$. Then A is the product of a coordinate scale, shear and reflection or rotation, and may be written*

$$A = \begin{bmatrix} r_1 \cos \theta_1 & -r_2 \sin \theta_2 \\ r_1 \sin \theta_1 & r_2 \cos \theta_2 \end{bmatrix} = \begin{bmatrix} \cos \theta_1 & -\sin \theta_2 \\ \sin \theta_1 & \cos \theta_2 \end{bmatrix} \begin{bmatrix} r_1 & 0 \\ 0 & r_2 \end{bmatrix} \quad (2.6)$$

where r_1, θ_1 and $r_2, \theta_2 + \pi/2$ are polar coordinates for column vectors 1, 2 of A . Thus θ_1 and θ_2 are the angles between unit vectors e_1 and e_2 along the positive x and y axes and their respective images Ae_1 and Ae_2 . Further, for a given IFS code, we can find the parameters in A above using

$$\begin{aligned} r_1 &= \sqrt{a^2 + c^2} & \theta_1 &= \arccos \frac{a}{r_1} \\ r_2 &= \sqrt{b^2 + d^2} & \theta_2 &= \arccos \frac{d}{r_2} \end{aligned} \quad (2.7)$$

Note that $\theta_1 = \theta_2$ in (2.6) implies that A is scaling followed by rotation (about the origin), while $\theta_1 = \theta_2 + \pi$ implies that A is scaling followed by reflection (in a line through the origin). We shall refer to (2.6) and (2.7) throughout this thesis.

2.3.2 Finding the IFS Code

In Section 2.2 we mentioned that if we can find the w_i 's for a given picture then we can reproduce the picture. We will shortly consider several algorithms which produce the attractors of iterated function systems on screen, but in the meantime we give an example which illustrates how to find the maps for a picture.

Theorem 2.18 *An affine transformation $w(z) = Az + t$ is uniquely determined by its effect on the vertices of any triangle. Conversely, for any two triangles BCD, EFG , there is a unique affine transformation sending B, C, D to E, F, G respectively. Let b be the position vector of B with respect to fixed perpendicular axes, and similarly for c, d, e, f and g . Then this transformation is given by*

$$A = [y_1 \ y_2][x_1 \ x_2]^{-1}, \text{ with } t = e - Ab, \quad (2.8)$$

where $x_1 = b - d, x_2 = c - d, y_1 = e - g$ and $y_2 = f - g$.

Example 2.19 Consider the face shown in Figure 2.4(a).

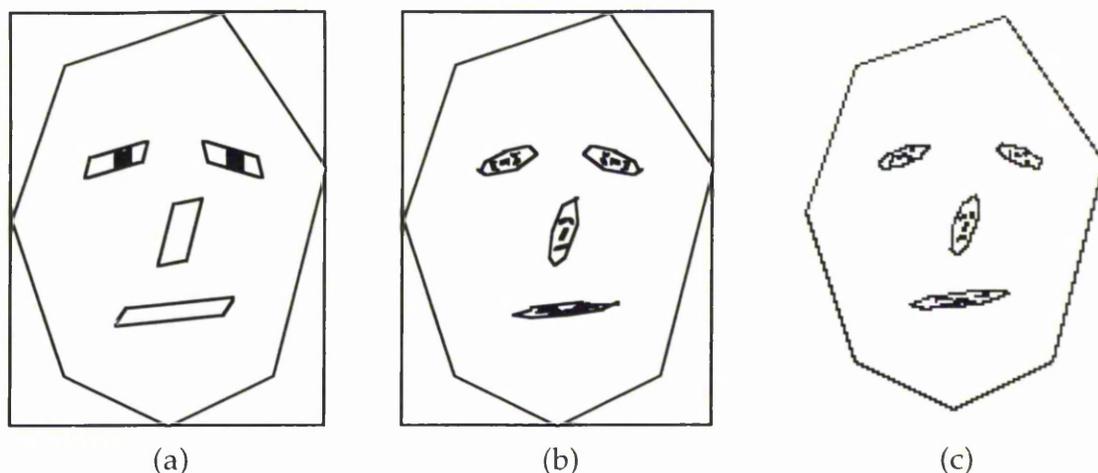


Figure 2.4 Finding an IFS whose attractor roughly approximates a given image.

It is roughly true that

- (i) the outline of the face is made up of 7 straight lines, and
- (ii) each feature is a scaled copy of the whole face.

We begin by drawing an invisible frame round the whole face, with horizontal and vertical sides just touching. Each of the 11 maps is specified by its effect on this frame - or rather, by Theorem 2.18, its effect on any three of the four vertices - and its code is calculated by (2.8). For example, suppose that the vertices of the frame are at $(0, 0)$, $(60, 0)$, $(0, 80)$ and $(60, 80)$ and that we use the first three of these as B , C , D respectively. Suppose further that map 1 sends B to $(0, 40)$, C to $(0, 40)$ and D to $(10, 70)$. Then $x_1 = (0, -80)$, $x_2 = (60, -80)$, $y_1 = (-10, -30)$ and $y_2 = (-10, -30)$ and by (2.8), $t = (0, 40)$. Finally, also by (2.8),

$$A = \begin{bmatrix} -10 & -10 \\ -30 & -30 \end{bmatrix} \begin{bmatrix} 0 & 60 \\ -80 & -80 \end{bmatrix}^{-1} = \frac{1}{4800} \begin{bmatrix} -10 & -10 \\ -30 & -30 \end{bmatrix} \begin{bmatrix} -80 & -60 \\ 80 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0.125 \\ 0 & 0.375 \end{bmatrix}$$

Thus w_1 has code $(0, 0.125, 0, 0.375, 0, 40)$. We can calculate the codes of the other ten transformations in exactly the same way, noting that for each feature we map the frame into some parallelogram. The resulting collage is shown in Figure 2.4(b),

while the IFS code is given in Table 2.1. Figure 2.4(c) shows the approximation to the attractor obtained using the Graphical Algorithm [5], which we shall introduce in Chapter 3. Notice that, in (b) and (c), the features are not simply parallelograms since the original face outline is not - which makes the face 'more human' - and also that mapping the whole face to each of the features results in the features being 'filled in' - which is appropriate for the eyes in particular. We will return to this IFS in Chapter 8.

	a	b	c	d	e	f
w ₁	0	1/8	0	3/8	0	40
w ₂	0	3/8	0	1/8	10	70
w ₃	0	1/4	0	-3/8	40	80
w ₄	0	-1/8	0	-1/2	60	50
w ₅	0	-1/4	0	-1/8	50	10
w ₆	0	-1/4	0	1/8	30	0
w ₇	0	-1/8	0	3/8	10	10
w ₈	1/60	1/8	1/15	3/80	14	48
w ₉	-1/60	9/80	1/15	-3/80	39	51
w ₁₀	1/12	3/80	1/60	3/20	28	31
w ₁₁	1/3	1/40	1/30	3/80	20	20

Table 2.1 The IFS code for the face of Figure 2.4.

Note We can use (2.6) and (2.7) to investigate how each map is made up. For example, map 1 has

$$\begin{bmatrix} r_1 \cos \theta_1 & -r_2 \sin \theta_2 \\ r_1 \sin \theta_1 & r_2 \cos \theta_2 \end{bmatrix} = \begin{bmatrix} 0 & 0.125 \\ 0 & 0.375 \end{bmatrix}$$

and so (2.7) says that $r_1 = 0$, $r_2 = \sqrt{0.125^2 + 0.375^2} \approx 0.395$ and $\theta_2 \approx -18.43^\circ$. Thus map 1 consists of scaling by 0 horizontally and 0.395 vertically, followed by rotation by -18.43° .

Once we have the IFS code, we require a method to reproduce the picture - and we now introduce several algorithms which do just that. Each of these algorithms is well documented elsewhere - general references include [2, 12, 25, 26]. For the remainder of this thesis we will simply state the codes of IFS's - but note that any of them can be calculated using (2.8).

2.4 The Collage Algorithm

The collage algorithm is a deterministic algorithm for producing the attractor of an IFS which follows from the definition of the collage map (Definition 2.12). If $\{w_{1-N}\}$ is an IFS of the plane, with ratio s , then the collage algorithm is given by

- (i) Choose any set $A_0 \in \mathbb{R}^2$
- (ii) Generate a sequence of collages $\{A_0, A_1, \dots\}$ where

$$A_{n+1} = W(A_n),$$

or equivalently

$$A_n = W^n(A_0).$$

By the Fixed Point Theorem 2.10, since W is contractive, this process generates a converging sequence of sets which tend toward the fixed point \mathcal{A} of W . In particular, (2.3) enables us to find a value of n which results in A_n being within a prescribed distance of \mathcal{A} , since

$$d(A_n, \mathcal{A}) \leq \frac{s^n}{1-s} d(A_0, A_1). \quad (2.9)$$

Example 2.20 We consider the attractor known as the *Sierpinski Gasket*. In spite of the fact that any starting set A_0 can be used - a consequence of Theorem 2.10 - it is illuminating to start with $A_0 = \Delta$, a filled in triangle. The vertices of Δ can be any three non-collinear points, a_1, a_2 and a_3 , but we shall consider a right angled triangle. The IFS is given by $\{w_1, w_2, w_3\}$, where $w_i(z) = 1/2(z + a_i)$ maps any point z to the halfway point on the line segment from z to a_i . Thus $w_1(\Delta)$ is a triangle whose vertices are a_1 and the midpoints of the sides adjacent to a_1 . Figure 2.5 shows the starting set $A_0 = \Delta$, and the first two collages A_1 and A_2 .

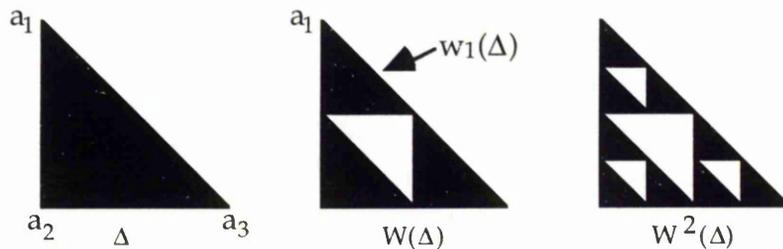


Figure 2.5 The first three approximations to the Sierpinski Gasket.

Thus $W(\Delta)$ may be regarded as Δ with a 'hole'. Observe further that applying w_1 to $W(\Delta)$ gives a small triangle with a hole. Thus applying W has the effect of scooping a hole out of every triangle at the previous stage, and the attractor, the Sierpinski Gasket, is as represented in Figure 2.6.

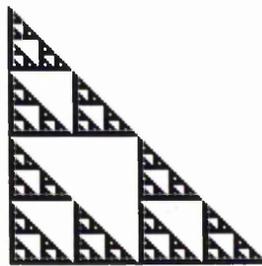


Figure 2.6 The Sierpinski Gasket Attractor.

However, whilst this algorithm will eventually give an approximation to the attractor to any desired accuracy, it is clearly an extremely inefficient way of doing

so. If we calculate N to be the minimum value of n which will give the desired accuracy, then in order to produce the approximation to the attractor, the computer must calculate, and then draw,

$$1 + 3 + 3^2 + \dots + 3^N = \frac{3^{N+1} - 1}{2}$$

triangles. Even for a relatively small value of N this number is very large (approximately 5.2×10^9 for $N = 20$) and consequently this algorithm is very slow. Since the aim is to produce the attractor quickly as well as accurately, the Collage Algorithm has very little practical use, but fortunately there are several algorithms which are faster. Shortly, we will introduce the algorithm which is the basis for this thesis - the *Graphical Algorithm* [5] - but first we describe two others which contrast.

2.5 The Random Iteration Algorithm (RIA)

In contrast to the collage algorithm, where at stage n we needed to calculate the collage of images $w_1(A_n) \cup \dots \cup w_N(A_n)$, in the random iteration algorithm we calculate just one point z_{n+1} , from its predecessor z_n , by $z_{n+1} = w_i(z_n)$ with i chosen at random from $1, 2, \dots, N$. The algorithm which is used to make this choice will be called the *driver* of the RIA. Traditionally, each map w_i occurs with preassigned probability p_i - determined by a method such as the one described below - and a random number generator is used to drive the RIA. Formally, the random iteration algorithm is given by

- (i) Choose $z_0 \in \mathcal{A}$ (typically z_0 is the fixed point of w_i for some i , $1 \leq i \leq N$)
- (ii) Plot points z_0, z_1, \dots , where $z_{n+1} = w_i(z_n)$ and i is chosen at random from $1 \dots N$, subject to preassigned probabilities p_i .

In order to set the probabilities we use the following method, popularised by Barnsley [1, 2]. As usual, let \mathcal{A} denote the attractor of the IFS $\{w_{1-N}\}$. Then the N sets $w_1(\mathcal{A}), \dots, w_N(\mathcal{A})$ form a covering of \mathcal{A} , that is, every point of \mathcal{A} is in at least one of the sets $w_i(\mathcal{A})$, called *attractorlets*. To achieve uniform distribution, assuming no significant overlap between attractorlets, the number of points in each attractorlet should be proportional to the area of that attractorlet. Thus since an affine map, $Az + t$, scales all areas by factor $|\det A|$, we set

$$p_i = \frac{|\det A_i|}{\sum |\det A_i|} \text{ where } w_i(z) = A_i z + t_i. \quad (2.10)$$

Thus the transformations which cover the largest areas are allowed the most points, and therefore this formula usually gives a good estimate of probabilities. There are, however, two situations where this formula is less effective, namely when there are large areas of overlap between attractorlets, or when w_i maps everything onto a line. In this second case, $|\det A_i| = 0$ so $p_i = 0$, meaning that w_i would never be chosen by the random number generator. To counter this, we set p_i to some small value, such as $p_i = 0.01$, ensuring that w_i will be chosen.

Note A better method for defining probabilities is given at the end of Section 2.6.

Example 2.21 Once again we consider the Sierpinski Gasket. Applying (2.10) to each w_i gives $p_i = 1/3$ for each i , $1 \leq i \leq 3$. The images produced at various stages of the RIA are shown in Figure 2.7. Note that even after only 500 iterations, the structure of the Sierpinski Gasket is clearly visible, although the approximation is clearly not particularly close to the attractor of Figure 2.6.

As discussed in Section 2.1, we can measure the 'closeness' of the approximations to the attractor using the Hausdorff distance and the results of doing this are given

in Table 2.2. For these calculations, the metric used was the lattice, or d_1 , metric - as discussed in Note 2.5 - and we calculated the Hausdorff distance using an algorithm described by Shonkwiler [27], with the attractor of Figure 2.6 being used as the 'perfect' image.

The table shows that the RIA approximations do indeed tend towards the attractor - however, even after 50,000 iterations, the approximation is still at Hausdorff distance of 2 from the perfect image, indicating that the RIA has limitations. Notice also that between 500 and 1,000 iterations there is no improvement in terms of distance, but the 1,000 iteration approximation clearly looks more like the attractor than the 500 iteration approximation. The same is true between 5,000 and 10,000 iterations - indicating that the Hausdorff distance also has its limitations as a measure of "closeness". However, we will not dwell on this problem here.

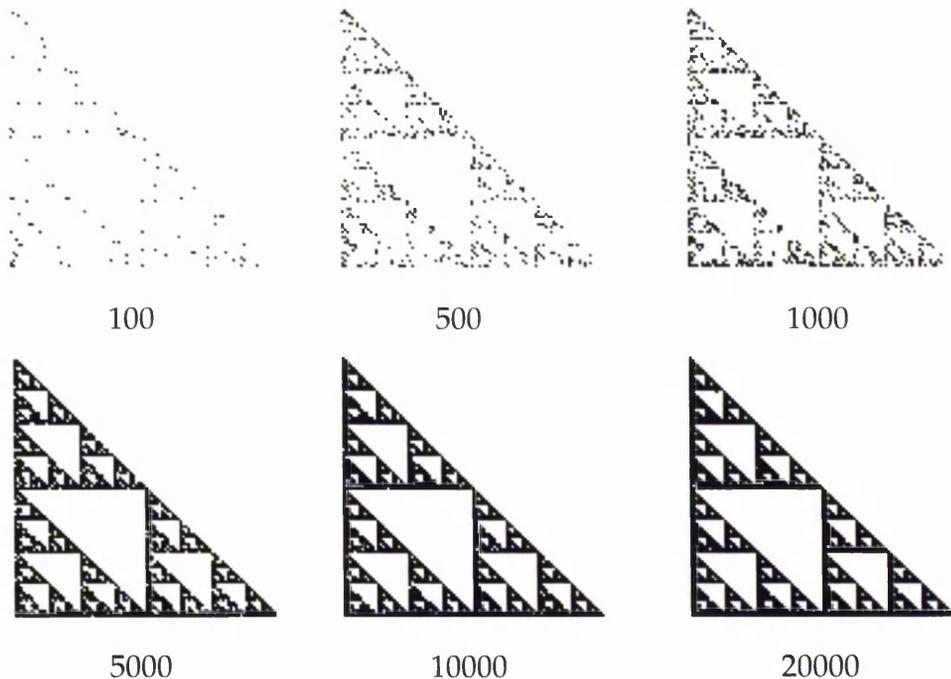


Figure 2.7 The Sierpinski Gasket after various stages of the RIA.

Number of Iterations	Hausdorff Distance
1	200
100	13
500	6
1000	6
5000	3
10000	3
20000	2
50000	2

Table 2.2 Hausdorff distances between the 'perfect' Sierpinski Gasket and various RIA approximations.

We have already hinted that the Random Iteration Algorithm has limitations, but before discussing what these limitations are, we must first introduce the idea of *addressing* for IFS's.

2.5.1 An Addressing Scheme

An addressing scheme is a way of identifying any given point of an IFS attractor by a sequence of symbols chosen from some alphabet. Below, we describe a general scheme for an IFS $\{w_{1-N}\}$, but first we give a simple example.

Example 2.22 Recall from Example 2.20 that stage n of the Collage Algorithm construction of the Sierpinski Gasket attractor involves dividing each triangle of the stage $(n - 1)$ approximation into four equal parts and then deleting the middle one. Suppose that each time this division is performed we identify the lower left triangle by 1, the lower right by 2, and the upper triangle by 3. Then after one stage it is easy to see, in Figure 2.8 below, that the point z lies in the subtriangle with label 3 - which we will call Δ_3 . At the second stage of the subdivision we see that z lies in subtriangle 1 of Δ_3 , or Δ_{31} , while a third subdivision reveals that z lies

in Δ_{312} - and thus the address of z begins 312. Finally, to determine the address of z to greater accuracy, we simply continue to divide and identify subtriangles.

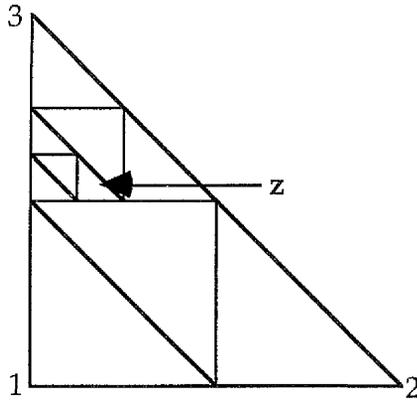


Figure 2.8 An addressing scheme for the Sierpinski Gasket.

Note The addressing scheme is closely related to the applications of the maps w_i , $1 \leq i \leq 3$, since $\Delta_i = w_i(\Delta)$ and therefore $\Delta_{ijk} = w_i w_j w_k(\Delta)$, where the maps are applied from left to right ($1 \leq i, j, k \leq 3$).

We now extend the notation of the above example to the more general IFS $\{w_{1-N}\}$ with attractor \mathcal{A} .

Notation 2.23 The subset of \mathcal{A} with address beginning $\sigma = \sigma_1 \sigma_2 \dots \sigma_k$ where k is finite, is given by

$$\mathcal{A}_\sigma = \mathcal{A}_{\sigma_1 \sigma_2 \dots \sigma_k} = w_{\sigma_1} w_{\sigma_2} \dots w_{\sigma_k}(\mathcal{A}).$$

Then $\mathcal{A} = \cup \mathcal{A}_{\sigma_1 \sigma_2 \dots \sigma_k}$, $1 \leq \sigma_i \leq N$, and further, any point $z \in \mathcal{A}$ is contained in some subset $\mathcal{A}_{\sigma_1 \sigma_2 \dots \sigma_k}$ and therefore has address beginning $\sigma_1 \sigma_2 \dots \sigma_k$.

Theorem 2.24 [26] *The ratio of the composite map $w_{\sigma_i} w_{\sigma_j}$ does not exceed the product of the ratios of w_{σ_i} and w_{σ_j} . If we denote the ratio of the map w_σ by $\rho(\sigma)$, then we have*

$$\rho(\sigma_i \sigma_j) \leq \rho(\sigma_i) \rho(\sigma_j)$$

Definition 2.25 The *diameter* of the attractor \mathcal{A} of an IFS is defined to be the maximum distance between any two points of \mathcal{A} . Thus

$$\text{diam}(\mathcal{A}) = \max\{d(z_1, z_2) : z_1, z_2 \in \mathcal{A}\}.$$

For further details and calculations see, for example, [6].

Definition 2.26 Let \mathcal{A} be the attractor of the IFS $\{w_{1-N}\}$ and let $\varepsilon > 0$. Then the associated list of *addresses* consists of all sequences $\sigma = \sigma_1\sigma_2\dots\sigma_k$ over $\{1, \dots, N\}$ with $\rho(\sigma_1\sigma_2\dots\sigma_k) \leq \varepsilon$ and $\rho(\sigma_1\sigma_2\dots\sigma_{k-1}) > \varepsilon$. Thus the addresses divide \mathcal{A} into subsets $\mathcal{A}_\sigma = w_{\sigma_1}w_{\sigma_2}\dots w_{\sigma_k}(\mathcal{A})$, each of diameter not exceeding $\varepsilon \times \text{diam}(\mathcal{A})$.

Example 2.27 We return to our previous example, and take $\varepsilon = 1/8$. Then since the ratio of the Sierpinski Gasket is 0.5, we require the addresses to have length 3. Thus the complete list of addresses is $\{\sigma_1\sigma_2\sigma_3 : \sigma_i \in \{1, 2, 3\}, 1 \leq i \leq 3\}$, and the subsets associated with these addresses partition the attractor as shown in Figure 2.9 below.

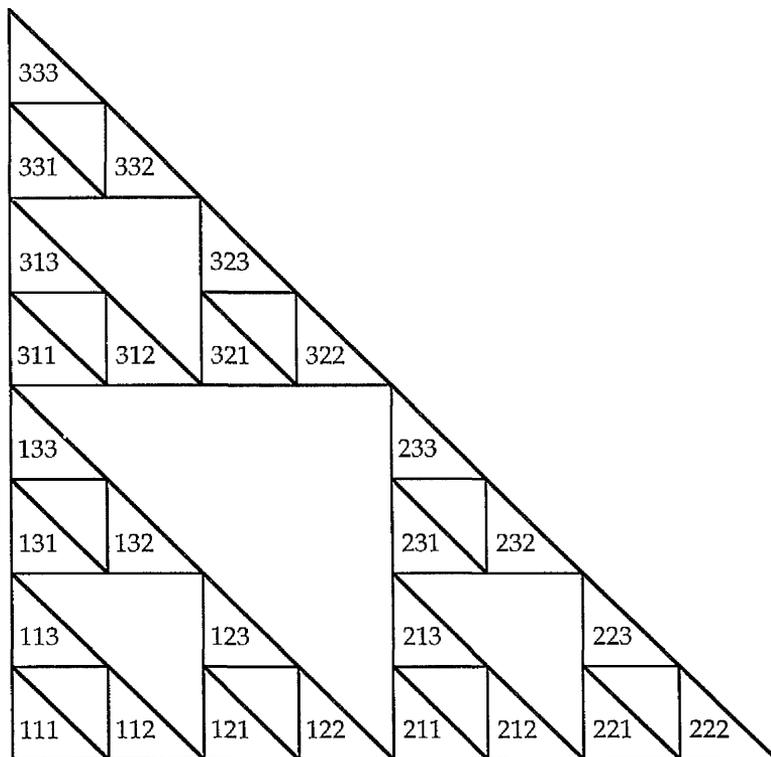


Figure 2.9 The subsets Δ_σ for the Sierpinski Gasket, where $\sigma = \sigma_1\sigma_2\sigma_3$ is an address.

Thus we have seen that in order to produce the attractor of an IFS $\{w_{1-N}\}$ to a guaranteed accuracy by our chosen method, we need to plot at least one point in each subset \mathcal{A}_σ - and it is here that the limitations of the RIA are exposed.

2.5.2 Limitations of the RIA

As we saw in the previous section, in order to produce the attractor of an IFS to a given accuracy, we shall plot at least one point in each subset \mathcal{A}_σ , where σ is an address. For a particular address $\sigma = \sigma_1\sigma_2\dots\sigma_k$, this means that we must plot $w_{\sigma_1}w_{\sigma_2}\dots w_{\sigma_k}(z)$, for some $z \in \mathcal{A}$. But what does this mean in terms of the sequence of maps produced by the RIA driver? Recall that once the driver - in our case a random number generator - generates a number i , w_i is immediately applied to the last point found, and therefore, in order to plot $w_{\sigma_1}w_{\sigma_2}\dots w_{\sigma_k}(z)$ we need to generate the maps in the order $\sigma_k\sigma_{k-1}\dots\sigma_1$. Thus the sequence of maps produced by the random number generator (or more generally the RIA driver) must ultimately contain the reverse of every address given by Definition 2.26 - which is not true of every RNG (for further discussion see [10, 11, 18]).

If, however, the random number generator is capable of producing a sequence of numbers which ultimately includes the reverse of every address, then the RIA will eventually produce the attractor to the desired accuracy - but even in these circumstances, the RIA still has limitations. Before elaborating, we require two Theorems - their proofs are given in [12].

Theorem 2.28 *Consider the IFS $\{w_{1-N}\}$ with ratio $s = \max(s_1, \dots, s_N)$ and attractor \mathcal{A} . Suppose the finite sequence $\sigma = \sigma_1\sigma_2\dots\sigma_m$ contains all k -digit combinations over $\{1, 2, \dots, N\}$ and is used as the driver in the RIA to produce a sequence of points $\{z_n\}$ where $z_0 \in \mathcal{A}$ and $z_{n+1} = w_{\sigma_n}(z_n)$. Then for every point $z \in \mathcal{A}$, there is a point z_n within distance ε , where ε is given by*

$$\varepsilon \leq s^k \text{diam}(\mathcal{A}). \quad (2.11)$$

Theorem 2.29 *If a sequence of digits from $\{1, 2, \dots, N\}$ contains every sequence of length k , then it has size at least $N^k + k - 1$.*

With the help of Theorems 2.28 and 2.29, we can now show that, even if our chosen random number generator is capable of producing a sequence of numbers which includes all length k sequences, or addresses, the RIA still has limitations. This is best shown by an example.

Example 2.30 Consider the Sierpinski Gasket once more, and let the attractor lie within a square of 100×100 pixels on a computer screen. Then if we take one pixel as one unit of length, we have $\text{diam}(\mathcal{A}) \leq 100\sqrt{2}$, say 142. Suppose that we want to ensure that every point of \mathcal{A} has a sequence point within one pixel. Then by (2.11) we require $(1/2)^k 142 < 1$ and so $k > 7$, although in practice $k = 7$ will probably suffice. Now, by Theorem 2.29, the RIA sequence must have length at least $3^7 + 7 - 1 = 2193$. However, it is highly unlikely that any sequence of random numbers of this minimum length would be constructed in such a way that it would include all 7-digit sequences over $\{1, 2, 3\}$ - and therefore the RIA is likely to require many more iterations than the optimal 2,193. In fact, running the RIA shows that even after 15,000 iterations there are still two 7-digit sequences which have not been produced, and it takes 15,887 iterations to produce all 7-digit sequences.

Thus we have shown that the RIA driven by a random number generator has severe limitations, even when all addresses are produced. It is clear from Example 2.30 that sequences produced by a random number generator are far from optimal. However, for a restricted type of case, it is possible to improve the RIA

by using Optimal Sequences - sequences which contain all addresses given by Definition 2.26 - to drive the RIA. For more details, see references [13, 14, 21, 22].

We now introduce another method [25, 26] which takes a slightly different approach, terminating only when the required accuracy has been reached.

2.6 The Adaptive Cut Method (ACM)

In Sections 2.4 and 2.5 we saw two methods for producing approximations to the attractor \mathcal{A} of an Iterated Function System $\{w_{1-N}\}$ - the Collage Algorithm and the Random Iteration Algorithm. Whilst both approaches can produce reasonable approximations we noted that they both have limitations. In particular, we noted that the performance of the RIA depends strongly on the choice of probabilities, and also that in order to reach the desired accuracy the RIA may require many more than the optimal number of iterations. We now briefly describe another method - the *Adaptive Cut Method* - which does not require probabilities, but is guaranteed to plot one point in each subset \mathcal{A}_σ for a given $\varepsilon > 0$. This means that for every point, z , of the attractor, there is a point z' in some \mathcal{A}_σ such that $d(z, z') < \varepsilon \times \text{diam}(\mathcal{A})$. Let $z_0 \in \mathcal{A}$.

At the first step, subdivide the attractor \mathcal{A} according to $\mathcal{A} = w_1(\mathcal{A}) \cup \dots \cup w_N(\mathcal{A})$. Thus, after this step, the addresses we have are just 1, 2, ..., N. Then step s is performed according to the following

- (i) Each subset of \mathcal{A} with address $\sigma_1 \dots \sigma_{s-1}$ where $\rho(\sigma_1 \dots \sigma_{s-1}) > \varepsilon$ is further subdivided into those with addresses $\sigma_1 \dots \sigma_{s-1}1, \sigma_1 \dots \sigma_{s-1}2, \dots, \sigma_1 \dots \sigma_{s-1}N$.
- (ii) For each subset of \mathcal{A} with address $\sigma_1 \dots \sigma_{s-1}$ where $\rho(\sigma_1 \dots \sigma_{s-1}) \leq \varepsilon$, plot the point $w_{\sigma_1} w_{\sigma_2} \dots w_{\sigma_{s-1}}(z_0)$.

We can implement the ACM by a simple recursive procedure, such as that described by the following pseudocode, reproduced from [20].

```

procedure AdCut (w : affine map)
  if  $\rho(w) \leq \epsilon$  then plot  $w(z_0)$ 
  else for  $i = 1$  to  $N$  do AdCut( $w.w_i$ )
  
```

Then calling $\text{AdCut}(\text{identity})$ will eventually result in one point being plotted in each attractorlet which is less than $\epsilon \times \text{diam}(\mathcal{A})$ in diameter - and thus we will have an approximation to \mathcal{A} which is of the desired accuracy, as discussed above.

Note The ACM can be used to give a better way of assigning probabilities for the RIA. We subdivide the points plotted by the ACM into N subsets, each one containing the points plotted in $w_k(\mathcal{A})$. Then the relative number of points in each subset determines the corresponding probability. When used as probabilities in the RIA, these values give an approximation which has more evenly distributed points than the one obtained using (2.10) to assign probabilities. An example is given in [25, 26].

Example 2.31 Recall that the ratio of the Sierpinski Gasket is 0.5. Taking $\epsilon = 0.5^n$ and running the ACM for various values of n gives the results shown in Figure 2.10. Note that the number of iterations is given by 3^n .

It is clear from Figure 2.10 that the ACM is more accurate than the RIA since even after only 81 iterations ($n = 4$) the structure of the Sierpinski Gasket is clearly visible. However, it has been noted [20] that the performance of the ACM appears sometimes to depend on the choice of starting point z_0 . As Figure 2.11 illustrates, this choice can affect the way the approximation to the attractor looks, and indeed can give misleading information about what the attractor looks like.

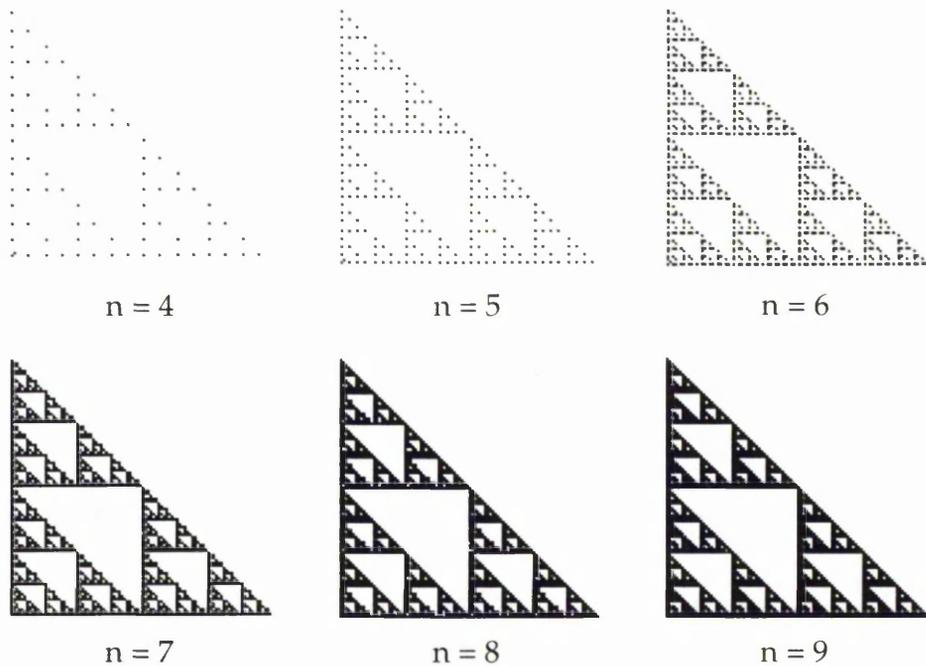


Figure 2.10 The Sierpinski Gasket after various stages of the ACM.

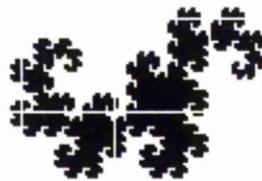


Figure 2.11 The Dragon Curve produced by the ACM. The white lines are not a feature of the attractor, but are instead a consequence of choice of z_0 .

In addition to the problem illustrated, the ACM is generally slower than the RIA, particularly for larger values of n . Ideally, therefore, we would like an algorithm which has the speed of the RIA, coupled with the accuracy of the ACM. For the remainder of this thesis we will concentrate on the *Graphical Algorithm* [5] which appears to have these properties, certainly after the improvements we shall study.

Chapter 3

The Graphical Algorithm

In Chapter 2 we described a number of algorithms which can be used to produce on-screen approximations to the attractor \mathcal{A} of an IFS $\{w_{1-N}\}$, and we noted that each has limitations, leading us to seek a more efficient algorithm. In this chapter, we describe another algorithm which not only appears to be quicker in many cases, but also to give a better approximation to \mathcal{A} . However, as we shall see shortly, it is not these initial observations alone which make the Graphical Algorithm of such great interest - rather it is the potential for improvement which we discuss in Section 3.2, and the subsequent improvements which we shall make in Chapters 5 to 8, which make this algorithm far better than any other discussed thus far, at least for a wide range of attractors.

3.1 Definition of the Algorithm

As with the other algorithms which we have discussed, we expect that the GA will give a series of approximations which tend towards the attractor of the IFS. We will show in Theorem 3.2 that this is indeed the case, but in order to be able to prove this theorem, we must begin by giving the formal definition of the Graphical Algorithm (GA), which is reproduced from [5]. However, whilst it is important to understand the algorithm in general, it can be stated much more simply as we shall see in Figure 3.1.

Definition 3.1 Let X be a metric space and δ be a positive real number. A *mesh* $M(\delta)$ is a subset of X such that

- (i) for any $z \in X$, $d(z, M(\delta)) < \delta$, and
- (ii) any ball of X contains only a finite number of points of $M(\delta)$.

Now let $X = \mathbf{R}^2$, and consider an IFS $\{w_{1-N}\}$. We recursively define two sequences of subsets of \mathbf{R}^2 , B_n and C_n for $n \geq 0$, until an empty set C_{n+1} is produced. Let z^* be the fixed point of some w_i , $1 \leq i \leq N$, and let $z_0 \in M(\delta)$ be such that $d(z_0, z^*) < \delta$. Let $B_0 = \{z_0\}$ and $C_0 = B_0$. If B_n and C_n have been computed, and C_n is non-empty, then B_{n+1} and C_{n+1} are found by the following rules

(1) Choose $z_n \in C_n$.

(2) Let $T = \emptyset$ where T is a temporary set. For each map w_i of the IFS with $d(w_i(z_n), B_n \cap T) \geq \delta$, choose $z' \in M(\delta)$ with $d(w_i(z_n), z') < \delta$, noting that such a z' exists by (1) of Definition 3.1. Then $T := T \cup \{z'\}$.

(3) $B_{n+1} := B_n \cup T$, $C_{n+1} := C_n \cup T - \{z_n\}$.

Note Whenever $B_n \cap T = \emptyset$ at step (2) above, we give $d(w_i(z_n), B_n \cap T)$ any value greater than δ and continue with the algorithm as before. This decision will be justified shortly.

The choice of z_n at step (1) will depend on z_{n-1} . In order to specify this choice we consider the following ordering of C_n :

$$\text{If } P = (a, b) \text{ and } Q = (c, d) \text{ then } P < Q \Leftrightarrow b > d \text{ or } (b = d \text{ and } a < c) \quad (3.1)$$

Then, in choosing $z_n \in C_n$, we choose the smallest point of C_n which is strictly greater than $z_{n-1} \in C_{n-1}$. If no such point exists, we choose z_n to be the smallest point of C_n .

Note In practice, we set $\delta = 1$ pixel so that the mesh consists of all integer points. Then, at step (2), the point added to T is simply the nearest integer point to the calculated point. Refining the mesh corresponds to scaling up the translation parts of the IFS maps.

Thus the GA can be summarised as follows, where the *store* is simply C_n . Note that at step 3, we plot and store the nearest integral point to $w_i(z_n)$ rather than $w_i(z_n)$ itself.

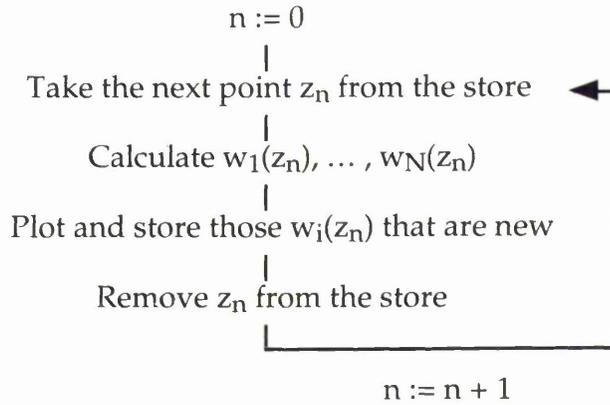


Figure 3.1 The Graphical Algorithm.

Note that at step (2) of the Graphical Algorithm, we add a point to T if and only if $d(w_i(z_n), B_n \cap T) \geq \delta$. In practice, this means that the only time we *do not* add a point to T is when the nearest integral point to $w_i(z_n)$ has already been lit in the approximation to the attractor *and* has been calculated previously *at this iteration*. - which is clearly not the case if $B_n \cap T = \emptyset$, justifying our earlier decision to add a point to T in such situations. Since a point z is in $B_n \cap T$ if and only if it is in both B_n and T , a point which has not been previously calculated at the current iteration will be added to T whether it has already been lit in the attractor approximation or not - and therefore, while this process enables us to avoid plotting the same point twice at the *same iteration*, it may not avoid duplication of points altogether.

In order to avoid plotting points more than once, it would seem logical to use $B_n \cup T$ rather than $B_n \cap T$. However, as n increases, so too does $|B_n|$, and so calculating $d(w_i(z_n), B_n \cup T)$ would take considerably more work than would be needed to simply plot any duplicated points. Note further that, although we may be able to avoid plotting any duplicated points, we cannot avoid calculating them

by this process. As we will see shortly, the GA does indeed result in many points being duplicated, and while we cannot avoid this within the framework of the algorithm, we will see that the duplication can be considerably reduced, and indeed removed completely in many cases, by an improved implementation of the GA which uses *jigsaws*. In the meantime, however, we will concentrate on showing why the Graphical Algorithm works.

As noted earlier, we expect that the GA will give a series of approximations which tend towards \mathcal{A} . Theorem 3.2 shows that this is indeed the case, as Example 3.4 will illustrate. The theorem, and the lemma that follows, are stated in slightly different form in [5], but the proofs given are expanded considerably here. Note that, while we use $\delta = 1$ in practice, both the theorem and the lemma hold for any value of δ .

Theorem 3.2 *If X is compact then for every positive δ there is an integral value of N for which C_N is empty, and in that case the Hausdorff distance between B_N and \mathcal{A} is smaller than $\delta/(1 - s)$ where s is the ratio of the IFS.*

In order to prove Theorem 3.2, we require the following Lemma.

Lemma 3.3 *If \mathcal{A} is the attractor of an IFS $\{w_{1-N}\}$ then, for every n ,*

$$d(B_n, \mathcal{A}) < \delta/(1 - s).$$

Proof Recall that $d(B, A) = \max\{d(b, A) : b \in B\}$. Then $d(B_0, \mathcal{A}) = d(z_0, \mathcal{A})$ since $B_0 = \{z_0\}$, and since z_0 is at distance at most δ from a mesh point we have $d(B_0, \mathcal{A}) < \delta$. Thus $d(B_0, \mathcal{A}) < \delta/(1 - s)$ and the hypothesis is true for $n = 0$.

Suppose that it holds for n so that $d(B_n, \mathcal{A}) < \delta / (1 - s)$. Then $d(z, \mathcal{A}) < \delta / (1 - s) \forall z \in B_n$. Now suppose that $z' \in B_{n+1} - B_n$. Then by definition of the algorithm, $\exists z \in B_n$ and some $i, 1 \leq i \leq N$, such that $d(w_i(z), z') < \delta$. Then

$$\begin{aligned}
 d(z', \mathcal{A}) &\leq d(z', w_i(z)) + d(w_i(z), \mathcal{A}) \\
 &\leq d(z', w_i(z)) + d(w_i(z), w_i(\mathcal{A})) \text{ since } w_i(\mathcal{A}) \subseteq \mathcal{A} \\
 &< \delta + d(w_i(z), w_i(\mathcal{A})) \text{ by above statement} \\
 &< \delta + sd(z, \mathcal{A}) \text{ by definition of } s \\
 &< \delta + s\delta / (1-s) \text{ by the inductive hypothesis} \\
 &= \delta / (1-s)
 \end{aligned}$$

Thus $d(z, \mathcal{A}) < \delta / (1-s) \forall z \in B_{n+1} - B_n$, and $d(z, \mathcal{A}) < \delta / (1-s) \forall z \in B_n$, so that $d(z, \mathcal{A}) < \delta / (1-s) \forall z \in B_{n+1}$. But then $d(\mathcal{A}, B_{n+1}) < \delta / (1-s)$, and we have proved the Lemma.

We are now in a position to prove Theorem 3.2, and thus to prove that the approximations obtained by the GA do indeed tend towards \mathcal{A} .

Proof The sequence B_n is non decreasing, bounded and contained in $M(\delta)$. It follows that the cardinality of B_n is a bounded monotone sequence and so there is an integer N_0 for which B_n is a constant set for $n > N_0$. The sequence C_n is strictly decreasing for $n > N_0$ - and consequently there is an integer N for which C_N is empty. Since C_N is empty, $h(W(B_N), B_N) < \delta$, where h is the Hausdorff distance. If $z \in B_N$ then $d(w_i(z), B_N) < \delta$ so that

$$d(W(B_N), B_N) = \max\{d(b, B_N) : b \in W(B_N)\} < \delta.$$

Now

$$\begin{aligned}
 d(\mathcal{A}, B_N) &\leq d(\mathcal{A}, W(B_N)) + d(W(B_N), B_N) \\
 &\leq d(W(\mathcal{A}), W(B_N)) + d(W(B_N), B_N) \\
 &< d(W(\mathcal{A}), W(B_N)) + \delta \\
 &< sd(\mathcal{A}, B_N) + \delta
 \end{aligned}$$

and therefore $d(\mathcal{A}, B_N) < \delta/(1 - s)$. Finally, we have $h(\mathcal{A}, B_N) = \max\{d(B_N, \mathcal{A}), d(\mathcal{A}, B_N)\}$ and so by the above and Lemma 3.3, $h(\mathcal{A}, B_N) < \delta/(1 - s)$ as required.

Thus the approximations B_n do indeed tend towards \mathcal{A} , and therefore the Graphical Algorithm will lead to a good approximation to the attractor. We now give an example of an attractor being produced by the GA.

Note By Theorem 3.2, we know that the final approximation to the attractor produced by the Graphical Algorithm will be very close to the true attractor. Thus, throughout this thesis, we shall refer to this approximation simply as *the attractor*.

Example 3.4 The *Dragon Curve* consists of two transformations, given at scale 64 by

$$\begin{aligned}
 w_1(x, y) &= (0.5x + 0.5y, -0.5x + 0.5y) \\
 w_2(x, y) &= (-0.5x + 0.5y + 64, -0.5x - 0.5y)
 \end{aligned}$$

We apply the GA, starting with $B_0 = C_0 = \{(0, 0)\}$. Figure 3.2 shows the sets B_n and C_n for n divisible by 1000, and for $n = 6346$, for which C_n is empty and the algorithm terminates.

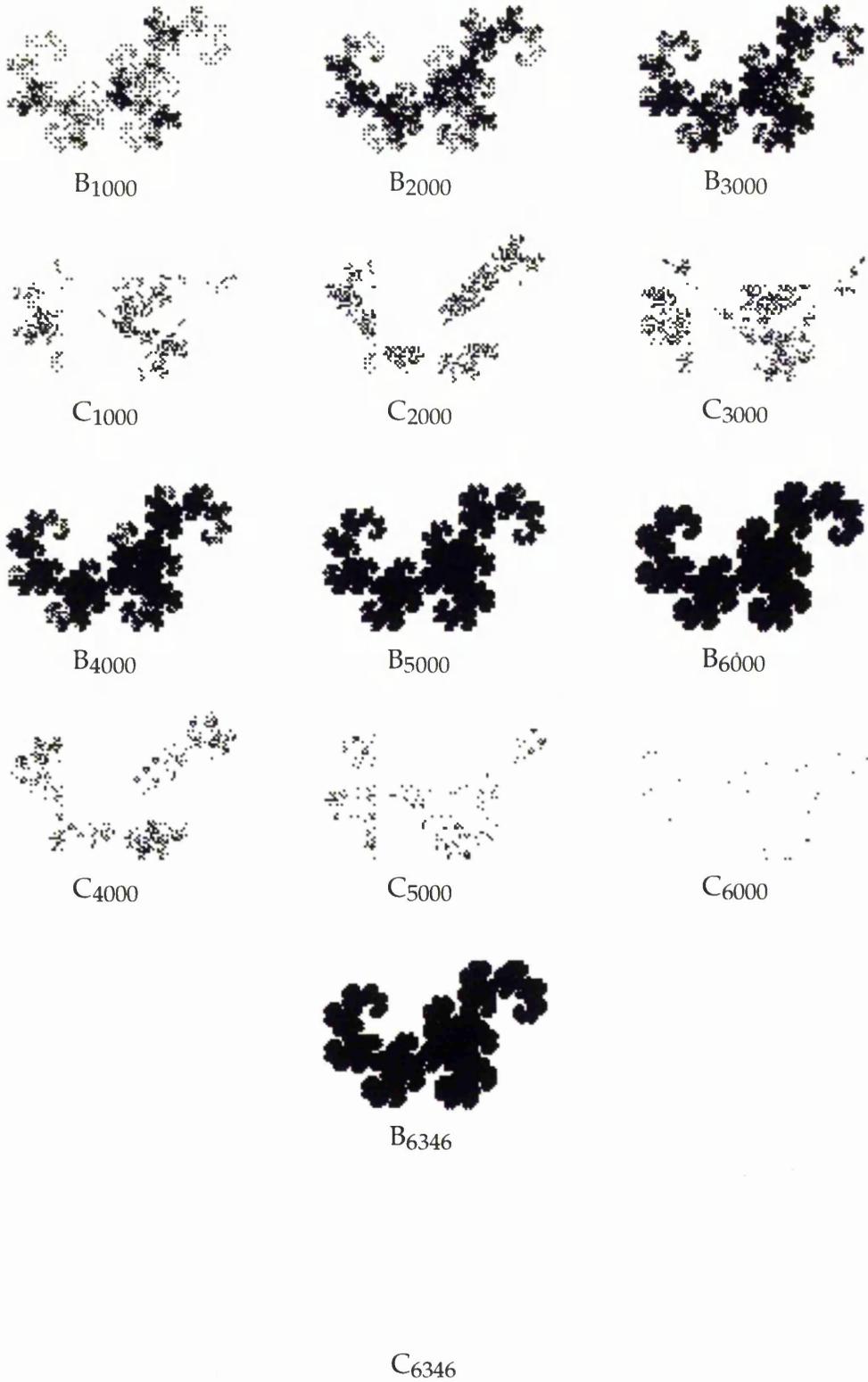


Figure 3.2 The Graphic Algorithm applied to the Dragon Curve.

3.2 The N : 1 Property

Table 1 of [5] exhibits the property that the ratio of calculated points to plotted points is always N : 1 for the GA (Algorithm E of [5]). This property is a simple consequence of the construction of the algorithm, since for each new point z_n found, we calculate $w_i(z_n)$ for $1 \leq i \leq N$. Example 3.5 illustrates this property for an IFS with two maps - the Takagi Function - while Example 3.6 shows that the N : 1 ratio is a global property which need not apply to every point individually.

Note From now on, we will refer to plotted points as *unique points* since no point is plotted more than once.

Example 3.5 The *Takagi Function* consists of two maps, given at scale 100 by

$$w_1(x, y) = 1/2(x, x + y)$$

$$w_2(x, y) = 1/2(x + 100, y - x + 100).$$

Applying the GA to this IFS results in 574 points being calculated, and 287 unique points being plotted, as predicted above. The final approximation to the attractor is shown in Figure 3.3 below.



Figure 3.3 The Takagi Function attractor produced by the Graphical Algorithm.

We have seen that when we run the GA, we calculate N times as many points as we actually need in order to produce the attractor, and consequently producing the attractor takes much longer than it ideally should. Thus, if we can cut down on the duplication of points - or even remove the duplication altogether - by a

method which is quicker than simply applying all N maps to every unique point, then we would expect that the time taken to produce the attractor would be improved. We will shortly introduce a method which does just this, and we will see that it can dramatically improve the run times of the GA. However, before considering how to improve the GA, we must focus on what causes a particular point to be hit more than once - for while it is the case that the number of points calculated is N times the number of unique points, it is not true that each unique point is hit N times. The following example illustrates a point being hit more than N times by the GA.

Example 3.6 When we apply the GA to the *Takagi Function*, we find that the four points $(47, 61)$, $(47, 60)$, $(48, 60)$ and $(48, 59)$ are all in the attractor. By definition, each of these points will be added to the store at some stage, and consequently each will also be chosen as z_n for some n , and the two maps of the IFS will be applied to each point. Now

$$w_1(47, 61) = 1/2(47, 47 + 61) = (23.5, 54),$$

$$w_1(47, 60) = 1/2(47, 47 + 60) = (23.5, 53.5),$$

$$w_1(48, 60) = 1/2(48, 48 + 60) = (24, 54),$$

$$w_1(48, 59) = 1/2(48, 48 + 59) = (24, 53.5).$$

Clearly, each of these values has closest integral point $(24, 54)$, and thus $(24, 54)$ is hit 4 times by the algorithm. Figure 3.4 illustrates this, where the shaded pixel is $(48, 60)$ and the pixel on the right hand side of the diagram is $(24, 54)$.

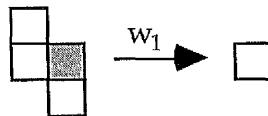


Figure 3.4 The four points (x, y) with $w_1(x, y) = (24, 54)$ as a set of grid pixels.

Thus it is possible for individual points to be hit more than N times, and since the number of points calculated is N times the number of unique points, it must also

be possible for points to be hit less than N times. We will shortly illustrate how this can happen, but first we must introduce the concept of *jigsaws*.

3.3 Introduction to Jigsaws

Example 3.6 gives a simple illustration of a concept which we will shortly describe in detail. As we will see in Chapter 5, every point in the attractor of the Takagi Function is the image of four distinct points under either w_1 or w_2 , although up to three of these points may lie outside the attractor (see Figure 3.6). Further, if the point is the image of points under map 1 then, when shown as a set of grid pixels, the four points will always lie in the formation shown in Figure 3.4. These ideas will be defined formally in Chapter 4, but for now we give a basic description of jigsaws.

A *jigsaw piece* for an IFS map i is the set of pixels $S = \{(x, y)\}$ such that when $w_i(x, y)$ is rounded to the nearest integer point, the result is the same for all $(x, y) \in S$. In other words S is the set of pixels which are sent to the same integer point under w_i . The set of all such jigsaw pieces for map i will be referred to as the *jigsaw covering*, or simply the *jigsaw* for the map.

Note In fact, when we use the term *jigsaw*, or *jigsaw covering*, we normally restrict our attention to all jigsaw pieces which lie within a rectangular set of points, given by $\{(x, y) : x_{\min} \leq x \leq x_{\max}, y_{\min} \leq y \leq y_{\max}\}$, where $x_{\min} = \min\{x : (x, y) \in \mathcal{A}\}$, and similarly for x_{\max} , y_{\min} and y_{\max} . We will often refer to this set as the *grid*, with the points of the set being referred to as *grid points*.

As noted above, the jigsaw pieces for map 1 of the Takagi Function all look like the piece of Figure 3.4. A small section of the jigsaw covering for this map is shown in Figure 3.5.

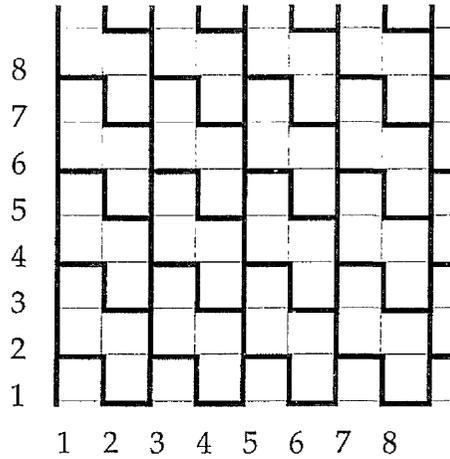


Figure 3.5 A section of the jigsaw covering for map 1 of the Takagi Function.

The jigsaw shown in Figure 3.5 is relatively simple since all of the pieces are the same shape and fit together in a regular pattern. The same properties hold for a large number of jigsaws arising from IFS maps, and indeed, as we shall see in Chapter 5, many maps have jigsaws which are even simpler than that of the Takagi Function. However, not all jigsaw coverings satisfy these properties, and in later chapters we will look at a number of IFS maps whose coverings are much more complicated. In the meantime, however, it is necessary to explain how jigsaws can be useful - and to note that their use is the same whether the jigsaws are simple or not.

In Example 3.6, we saw a situation where the GA results in four distinct attractor points being mapped to the same integral point under a particular transformation. In the example - the Takagi Function - the resultant point, $(24, 54)$, was hit four times, when in fact once would suffice. Since we have already noted that the number of points calculated is N times the number of unique points, it is clear that $(24, 54)$ is not the only point which is hit more than once. In fact, running the GA for the Takagi Function and keeping a check on how many times each unique point is hit reveals the results of Table 3.1.

	Unique Points	Calculated Points
Points hit once	87	$87 \times 1 = 87$
Points hit twice	134	$134 \times 2 = 268$
Points hit 3 times	45	$45 \times 3 = 135$
Points hit 4 times	21	$21 \times 4 = 84$
Total	287	574

Table 3.1 Results of counting how many times points are hit for the Takagi Function.

Thus the number of times points of the Takagi Function attractor are hit ranges from one to four - which is a simple consequence of the fact that each of the jigsaw pieces which cover the attractor consist of four pixels, as we saw in Figure 3.4. Figure 3.6 illustrates this point further, showing a small section of the jigsaw covering for map 1 of the Takagi Function with attractor points shaded.

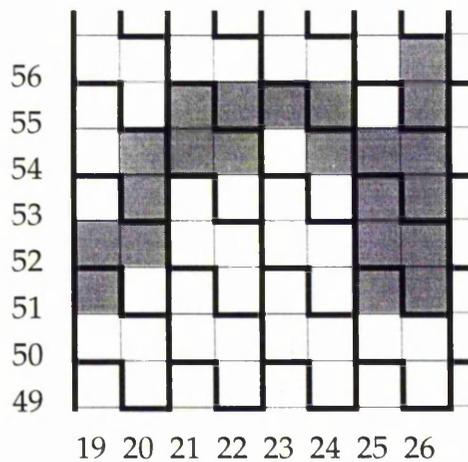


Figure 3.6 A section of the jigsaw covering with attractor points shaded.

Clearly if a jigsaw piece contains n attractor points, one of which is given by (x, y) , then the resultant point $w_1(x, y)$ will be hit n times, since all attractor points have w_1 applied to them at some stage of the algorithm. Therefore, if we can somehow ensure that we only apply w_1 to one of the points in each jigsaw piece which contains attractor points - and do likewise for w_2 - then the duplication will be

removed and the number of points calculated will equal the number of unique points.

Note As we will see later, the attractors of most Iterated Function Systems contain points which can be hit by more than one map. In these cases, it is not possible to remove duplication completely simply by considering the jigsaws. However, in the Takagi Function, and also in several other IFS's which we will consider, each attractor point can only be hit by one map and therefore duplication can be completely removed.

In the chapters that follow, we will see that in many cases it is possible to identify all points of a jigsaw piece given a particular one. In these cases, we will describe how we maintain a record of which jigsaw pieces have had a map applied to them once - and how this enables us to ensure that the remaining points within the piece will not have that map applied to them. Note, however, that since our aim is to improve the time taken to produce the attractor, it is extremely important that it does not take longer to check whether applying a map to a particular point would result in duplication or not, than it would simply to apply the map. As we will see, the method that we employ to check for duplication does not violate this condition - and as a consequence we are able to improve considerably the run time of the Graphical Algorithm for a large number of Iterated Function Systems. We begin, in Chapter 4, by investigating the properties of jigsaw pieces.

Chapter 4

Properties of Jigsaw Pieces

In the last chapter, we introduced the concept of *jigsaws*, and explained in general terms how we can make use of them to improve the performance of the Graphical Algorithm. In Chapter 5, we will describe the improvement process in much greater detail, and see many examples which highlight how well the improvements work. In this chapter, however, we will concern ourselves with the various properties which jigsaw pieces have, exploring both those which hold for any IFS map, and those which depend on the map itself - details are also given in [16]. Before we consider the first property, we must formalise the ideas which we outlined in Section 3.3 - we do so in Definition 4.2.

Notation 4.1 We use the symbol $\sim>$ to denote 'rounds to', while the symbol $\sim/>$ denotes 'does not round to'. Either symbol may be preceded by a list of expressions to which it applies. Denoting the horizontal coordinate of $w(x, y)$ by $w_h(x, y)$, and the vertical by $w_v(x, y)$, we may write

$$w(x, y) \sim> (u, v) \Leftrightarrow \begin{cases} \alpha \leq w_h(x, y) < \alpha + 1 \\ \beta \leq w_v(x, y) < \beta + 1 \end{cases} \quad (4.1)$$

where $\alpha = u - 1/2$ and $\beta = v - 1/2$. From the definition of an affine map (Definition 2.14), we have $w_h(x, y) = ax + by + e$ and $w_v(x, y) = cx + dy + f$, yielding the following useful relations

$$w_h(x + i, y + j) = w_h(x, y) + ai + bj \quad (4.2a)$$

$$w_v(x + i, y + j) = w_v(x, y) + ci + dj \quad (4.2b)$$

Finally, we use the expression $\underline{w}(x, y)$ to denote the result of rounding $w(x, y)$ to the nearest integer point. Thus $\underline{w}_h(x, y) = u$ if and only if $\alpha \leq w_h(x, y) < \alpha + 1$, and similarly for $\underline{w}_v(x, y) = v$.

Definition 4.2 The *jigsaw piece* $P_{u,v}$ of w consists of all integer points (x, y) such that $\underline{w}(x, y) = (u, v)$. In all diagrams of jigsaw pieces, a pixel shaded \blacksquare will denote one which is in the jigsaw piece, while one shaded \square will denote a pixel which may or may not be in the jigsaw piece. Finally, a sequence ... (n) ... between two pixels of the same type will indicate a line of n such pixels.

We begin our discussion of the properties of jigsaw pieces by stating and proving a simple Theorem, which explains why we use the term *jigsaw*.

Theorem 4.3 *The jigsaw pieces of an IFS map tile the plane; that is they have the following properties*

- (1) *they cover the whole plane, and*
- (2) *they do not overlap.*

Proof (1) The covering property holds because any point (x, y) is sent by w_i to another point (u, v) which, after rounding, not only has integer coordinates, but lies in the plane - and therefore (x, y) lies in a jigsaw piece, namely $w^{-1}(u, v)$. (2) For two jigsaw pieces to overlap requires one point being sent, by the same map, to two others - which is impossible by definition of *transformation*.

We now move on to consider properties of the pieces of the jigsaw, beginning with a property which applies to every IFS map, regardless of the values of its code entries.

4.1 Holes in Jigsaw Pieces

The first question we wish to ask about jigsaw pieces is whether or not they can contain 'holes' - that is, is there any set of pixels which are not in the piece, but are at least partially surrounded by piece pixels? In order to answer this question, we must give a simple formal definition of a hole.

Definition 4.4 We say that a pixel $P = (x, y)$ is a *hole* in a given jigsaw piece if P itself is not in the piece but there are pixels of the piece to the left and right of P at the same height as P , or if the analogous vertical condition holds (or, indeed, if both hold).

Figure 4.1 gives two examples of jigsaw pieces which contain holes. Note that since the horizontal and vertical conditions need not be satisfied simultaneously, a hole need not be closed off on all sides by pixels of the piece, but may simply be a gap in one of the edges of the piece, as in the second example below.

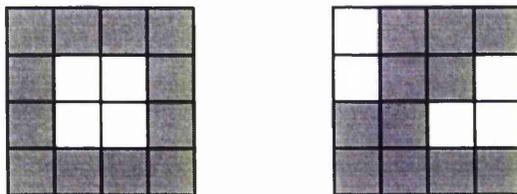


Figure 4.1 Two examples of jigsaw pieces which contain holes.

In fact, neither of the situations illustrated in Figure 4.1 can happen in a jigsaw piece arising from an IFS map, as we state in Theorem 4.5. This Theorem is a simple consequence of a much stronger result - Theorem 4.6 - which will prove to be useful not just for showing that jigsaw pieces do not contain holes, but also in the properties which we will discuss in Section 4.2.

Theorem 4.5 *The jigsaw pieces of an affine map cannot contain holes.*

Theorem 4.6 (Convexity of pieces) *Let p, q, M be integers, with M positive. If $w(x, y), w(x + Mp, y + Mq) \rightsquigarrow (u, v)$ then $w(x + kp, y + kq) \rightsquigarrow (u, v)$ for all $0 \leq k \leq M$.*

Proof We have $w_h(x + kp, y + kq) \rightsquigarrow u$ for $k = 0, M$. Then by (4.1) and (4.2) we have $\alpha \leq w_h(x, y) + k(pa + qb) < \alpha + 1$ for $k = 0, M$. But for $0 < k < M$, and whatever sign $pa + qb$ takes, the value of $k(pa + qb)$ lies *between* those of $0(pa + qb)$ and $M(pa + qb)$, and hence $\alpha \leq w_h(x, y) + k(pa + qb) < \alpha + 1$ holds for all values $0 \leq k \leq M$. Using (4.1) and (4.2) again, we have $w_h(x + kp, y + kq) \rightsquigarrow u$ for all $0 \leq k \leq M$. The argument for $w_v(x, y)$ is similar and so we have that $w(x + kp, y + kq) \rightsquigarrow (u, v)$ for all $0 \leq k \leq M$, as required.

Some of the consequences of Theorem 4.6 are illustrated in Figure 4.2 below.

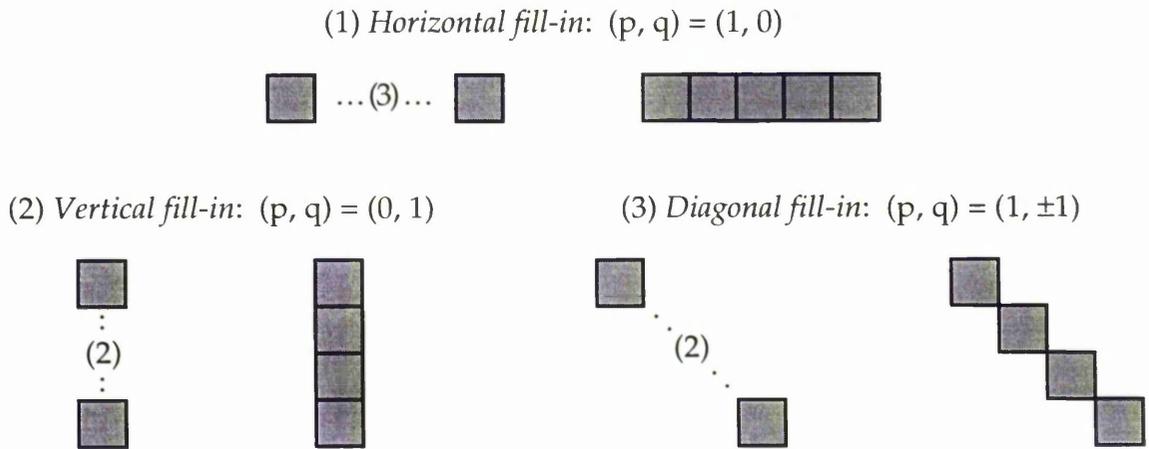


Figure 4.2 Some of the consequences of Theorem 4.6. (1) has $(p, q) = (1, 0)$, (2) has $(p, q) = (0, 1)$ and (3) has $(p, q) = (1, -1)$.

Now, suppose that two integers i and j have highest common factor M so that $(i, j) = M(p, q)$ for some integers p, q . If $P = (x, y)$ and $Q = (x + i, y + j)$ are two pixels of the same piece then, since $Q = (x + Mp, y + Mq)$, Theorem 4.6 tells us that

$(x + kp, y + kq)$ is a pixel of the piece for all $0 \leq k \leq M$. But these are precisely the pixels which lie on the straight line from P to Q , and so we can state the following consequence of Theorem 4.6.

Corollary 4.7 (General fill-in) *If P, Q are pixels of a jigsaw piece, then so are all pixels in a straight line from P to Q .*

Remark 4.8 The number of pixels in a straight line from $P = (x, y)$ to $Q = (x', y')$, excluding P and Q , is $M - 1$, where M is the greatest common factor of $x - x'$ and $y - y'$. Thus if $x - x'$ and $y - y'$ are coprime (have no common factor other than 1) then there are no such pixels.

Example 4.9 Suppose we know that the shaded pixels in Figure 4.3(a) below are in the same jigsaw piece. Then Theorem 4.6 (Diagonal fill-in, as in Figure 4.2(3)) yields the pixels represented by asterisks in Figure 4.3(b). One of these pixels - denoted by P in (b) - then tells us that the pixel denoted by R in (c) is also in the piece, since R is a pixel on the straight line from P to Q (Corollary 4.7). This is General fill-in with $(p, q) = (-2, 1)$ and $M = 2$.

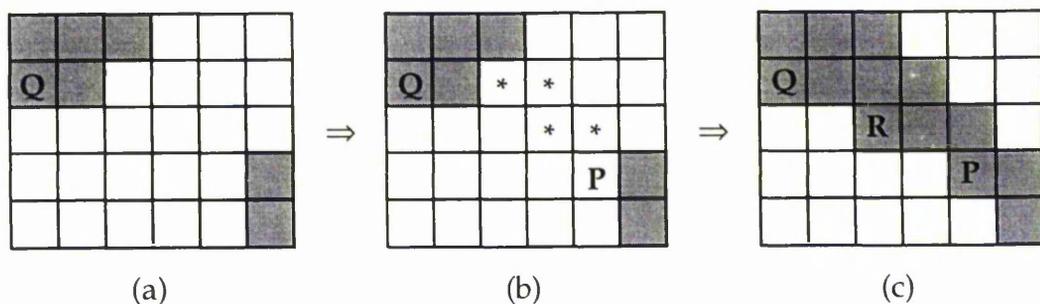


Figure 4.3 The pixels of (a) being in a piece imply those of (b) being in, and then those of (c) being in, by Theorem 4.6.

Remark 4.10 Theorem 4.6 applies to the horizontal and vertical components separately, as we saw in the proof. Thus, for example, if $w_h(x, y)$ and $w_h(x + Mp, y + Mq) \rightsquigarrow u$, then $w_h(x + kp, y + kq) \rightsquigarrow u$ for all $0 \leq k \leq M$. Equivalently, this means that there can be no break in a line of pixels (x, y) satisfying $\underline{w}_h(x, y) = u$, for given u .

The importance of knowing that jigsaw pieces cannot contain holes will be highlighted in later chapters, as we consider particular IFS maps and the jigsaws arising from them. In the meantime, however, it suffices to say that the main consequence of Theorem 4.6 is that, once we know what the boundary of a jigsaw piece is like, we know what the whole piece is like. Further, we can rule out certain configurations of pixels within jigsaw pieces - which will be especially useful when we come to look at maps with complicated jigsaws.

We now consider a property which many jigsaws arising from IFS maps have, but, unlike the property discussed in this section, our second property depends on the entries of the IFS code.

4.2 Rectangular Jigsaw Pieces

In the course of this thesis we will see a large number of examples of jigsaws arising from IFS maps, ranging from simple (or regular) jigsaws where every piece is the same size and shape (Chapter 5) to irregular jigsaws made up from a wide range of sizes and shapes of piece (Chapters 7 and 8). Often, it is impossible to tell what the pieces will look like, or even whether they will all be the same, simply by looking at the code of the IFS map - which can make using the jigsaws complicated. Fortunately, however, there are many cases where we can use the IFS code to predict the size and shape of the pieces. In this section, we will describe conditions on a, b, c and d which enable us to state that the jigsaw pieces

will all be rectangular. We begin with a general Theorem stating a condition which must hold if two pixels (x, y) and $(x + i, y + j)$ are to be in the same jigsaw piece.

Theorem 4.11 *Suppose (x, y) and $(x + i, y + j)$ are in the same jigsaw piece, for some integers i, j . Then*

$$|ai + bj| < 1 \text{ and } |ci + dj| < 1. \quad (4.3)$$

Proof Let $w(x, y)$ and $w(x + i, y + j)$ round to (u, v) , and consider first the horizontal components. By (4.1) and (4.2a), we have that $w_h(x, y)$ and $w_h(x, y) + ai + bj$ both lie in the interval $\alpha \leq z < \alpha + 1$, and therefore the absolute value of their difference, $ai + bj$, cannot exceed the interval width. Indeed, because we must have $z < \alpha + 1$, it follows that $|ai + bj| < 1$. Similarly, we find that $|ci + dj| < 1$, and the result is proven.

The simplest type of rectangular piece which we could encounter would clearly be one which is simply a line of pixels either horizontally or vertically i.e. one for which (x, y) and $(x + i, y + j)$ are in the same piece if and only if either $i = 0$ or $j = 0$. Such pieces do not occur very often, but Corollary 4.13 gives two conditions which, if they occur together, lead to exactly this scenario. First, we must give some new definitions which we will use throughout the remainder of this chapter, and beyond.

Definition 4.12 (i) A sequence of pixels, each to the right of its predecessor, is said to be *rising* (*descending*) if each is above (below) its predecessor. (ii) Two pixels P and Q are said to be *adjacent* if they have a common vertex.

Corollary 4.13 For the jigsaw pieces of an IFS map, diagonally adjacent pixels are ruled out as follows:

- (i) $|a + b| \geq 1$ or $|c + d| \geq 1$ rules out  (rising diagonal), while
(ii) $|a - b| \geq 1$ or $|c - d| \geq 1$ rules out  (descending diagonal).

If either condition holds, then no piece can contain a 2×2 square of pixels, while if both hold, then each piece is *flat*, that is, a horizontal or vertical line of pixels.

Proof (i) Any two pixels forming a rising diagonal may be written as (x, y) and $(x + 1, y + 1)$. But (x, y) and $(x + 1, y + 1)$ can only be in the same piece if $|a + b| < 1$ and $|c + d| < 1$, by (4.3) with $(i, j) = (1, 1)$. Thus if $|a + b| \geq 1$ or $|c + d| \geq 1$, (x, y) and $(x + 1, y + 1)$ cannot be in the same jigsaw piece, and therefore no rising diagonal can occur. (ii) is similar.

As we noted above, flat jigsaw pieces do not occur very often, since the code entries of most IFS maps violate at least one of the conditions of Corollary 4.13. However, as the following example illustrates, it is possible to find maps with flat jigsaw pieces.

Example 4.14 Map 1 of the *Takagi Function* satisfies only condition (i) of Corollary 4.13, and therefore the pieces have no rising diagonal, but can have descending diagonals (cf. Figure 3.4). However, both conditions of Corollary 4.13 are satisfied by map 2 of the *Dragon Curve*,

$$w(x, y) = \begin{bmatrix} -0.5 & 0.5 \\ -0.5 & -0.5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix},$$

and thus the pieces are flat. In fact, we find that the pieces have the form . Both conditions are also satisfied by map 1 of the Dragon Curve, and in this case the pieces are those for map 2 rotated through 90° .

Note Although the pieces of the Dragon Curve jigsaws are flat, and consist of just two pixels, the way in which they fit together to form the jigsaw makes them as complicated to deal with as many larger pieces. We will return to this in Chapter 6.

We now consider what conditions imply that the jigsaw pieces are all rectangular - these conditions are given in the statement of Corollary 4.18. Essentially, proving that a piece is rectangular requires us to show that if two corners of a rectangle are in the piece, then so is the whole piece. We begin by stating and proving this in a theorem, and then show that certain conditions on a , b , c and d ensure that both conditions of this theorem hold, and so the pieces are rectangular.

Definition 4.15 (i) We say that $A, B \in \mathbf{R}$ have the same sign if $A, B \geq 0$ or $A, B \leq 0$. This is equivalent to $AB \geq 0$, and holds in particular if either A or B is itself zero. (ii) If P and Q are two pixels, then $Rect(P, Q)$ denotes the rectangle which has P and Q as opposite corners and sides parallel to the axes.

Theorem 4.16 (Rectangle fill-in) Let $P = (x, y)$ and $Q = (x + i, y + j)$ be in the same jigsaw piece $P_{u,v}$. Then

if (i) $abij \geq 0$, then $w_h(r, s) \rightsquigarrow u$ for all (r, s) in $Rect(P, Q)$, while

if (ii) $cdij \geq 0$, then $w_v(r, s) \rightsquigarrow v$ for all (r, s) in $Rect(P, Q)$.

Thus if both (i) and (ii) hold, the whole of $Rect(P, Q)$ is in $P_{u,v}$.

Proof (i) Since $w_h(x, y), w_h(x + i, y + j) \rightsquigarrow u$, (4.1) and (4.2) tell us that both $w_h(x, y)$ and $w_h(x, y) + ai + bj$ lie in the interval $[\alpha, \alpha + 1) = \{z: \alpha \leq z < \alpha + 1\}$. An arbitrary pixel of $Rect(P, Q)$ may be written as $(x + p, y + q)$ where p, q have the same sign as i, j respectively, and $|p| \leq |i|, |q| \leq |j|$. Since ai and bj have the same sign, $ap + bq$ must share this sign and, further, $|ap + bq| \leq |ai + bj|$ so that $w_h(x, y) + ap + bq$ lies in the interval $[\alpha, \alpha + 1)$. Thus $w_h(x + p, y + q) \rightsquigarrow u$ and so

$w_h(r, s) \sim u$ for all (r, s) in $\text{Rect}(P, Q)$ - proving (i). The proof of (ii) is entirely similar, and clearly if both conditions hold then the whole of $\text{Rect}(P, Q)$ is in $P_{u,v}$, as illustrated in Figure 4.4.

Notice that, when $i > 0$, PQ is rising when $j > 0$, and descending when $j < 0$. Similarly, when $i < 0$, PQ is rising when $j < 0$, and descending when $j > 0$. Thus $ij > 0$ implies that PQ is rising and $ij < 0$ implies that PQ is descending - and we can state the following Corollary to Theorem 4.16.

Corollary 4.17 *If pixels $P = (x, y)$ and $Q = (x + i, y + j)$ are in a piece $P_{u,v}$, then so is $\text{Rect}(P, Q)$ if either*

- (i) $ab, cd \geq 0$ and PQ is rising, or
- (ii) $ab, cd \leq 0$ and PQ is descending.



Figure 4.4 The implications of Corollary 4.17 (i) and (ii).

Now we are in a position to state conditions which ensure that all jigsaw pieces will be rectangular. Note, however, that this Corollary should only be applied if we know that the pieces are not flat - implying that at least one of the conditions of Corollary 4.13 is violated.

Corollary 4.18 *If one of a, b is equal to zero, and one of c, d is equal to zero, then every piece of the jigsaw is rectangular.*

Proof One of a, b equal to zero, and one of c, d equal to zero means that both conditions of Theorem 4.16 are satisfied, and hence every pixel of $\text{Rect}(P, Q)$ is in

the jigsaw piece, for every pair P, Q of pixels in the jigsaw, making the pieces rectangular.

Example 4.19 Corollary 4.18 applies trivially to rotation by $\pi/2$, where

$$w(x, y) = (-y, x).$$

In this case the jigsaw pieces consist of single pixels - i.e. $P_{x, y} = (y, -x)$. Note that, while this map is not itself contractive, composing it with coordinate scaling will make it so.

Example 4.20 As we will see in Chapter 5, Corollary 4.18 applies to all maps of the Sierpinski Gasket, Sierpinski Carpet, Peano Curve and Flamboyant Crown, and thus all pieces of their jigsaws are rectangular.

Throughout this section, we have been looking at cases where (x, y) and $(x + i, y + j)$ being in a particular jigsaw piece imply that $(x + r, y + s)$ is also in the piece for all $0 \leq r \leq i$ and $0 \leq s \leq j$. In particular, when $|i| = |j| = 1$, this means that the implications shown in Figure 4.5 hold.



Figure 4.5 The implications of Corollary 4.17 when $|i| = |j| = 1$.

Further, we saw in Corollary 4.18 that whenever one of a, b is equal to zero, and one of c, d is equal to zero, both conditions of Corollary 4.17 hold, and the pieces are rectangular, and we noted that there are many IFS maps for which this is true. However, there are also many IFS maps for which at least one of the conditions of Corollary 4.17 are violated - we now consider what we can say about the jigsaw pieces of such maps.

4.3 Bridges in Jigsaw Pieces

In many IFS maps at least one of the conditions of Corollary 4.13 is violated, and at most one of the conditions of Corollary 4.17 holds. In such cases, we cannot say either that the jigsaw pieces will all be flat, or that the pieces will all be rectangular. In particular, the implications illustrated in Figure 4.5 will not both hold, and indeed, in some cases, neither will hold. In this section, we will consider IFS maps whose jigsaw pieces can contain some of the configurations illustrated in Figure 4.6, where in these cases, the pixels which are unshaded *are not in the jigsaw piece*.

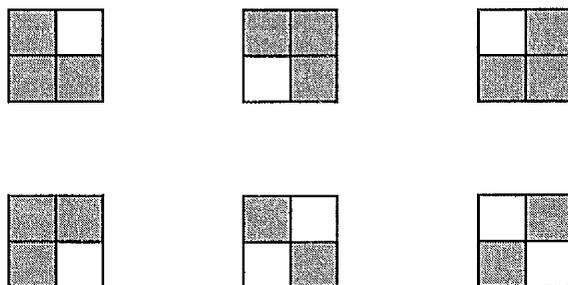


Figure 4.6 Possible configurations of pixels within jigsaw pieces which are not rectangular. Note that the unshaded pixels are not in the piece.

In each of the following definitions, we assume that all pixels P_i are in the same jigsaw piece including, of course, $P_0 = P$ and $P_r = Q$.

Definition 4.21 (i) A P - Q path is a sequence of pixels $P = P_0, P_1, \dots, P_r = Q$ such that P_i, P_{i+1} are adjacent for $0 \leq i \leq r - 1$. (ii) We say that two pixels P and Q are *joined* (or *connected*) if there is a P - Q path between them. (iii) A set of pixels is said to be *connected* if every two of its members are connected, otherwise it is *disconnected*. (iv) Finally, a *component* of a set of pixels is a maximal connected subset.

In light of these definitions, it is easy to see that in the jigsaw piece of Figure 4.7, P and R are not connected, whereas P and Q are, a shortest path being PCDQ. Further, the piece has two components, one being R alone, and the other consisting of all pixels joined to P (or equivalently Q), namely A, B, P, C, D and Q. Recall that unshaded pixels are not in the piece.

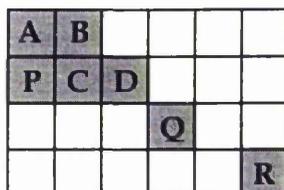


Figure 4.7 A jigsaw piece consisting of two components.

It is often useful to have a stronger version of each of the above definitions, and also of Definition 4.12(ii), prefixing each by 'edge'. Thus the adjacency of pixels P and Q is strengthened to *edge-adjacency* if P and Q have a whole edge in common rather than just one vertex. In Figure 4.7 above, A and D are joined by an edge path (APCD), but A and Q are not, since D and Q are only vertex adjacent. It follows that a component which is not itself edge connected subdivides into a number of edge connected components, which we will also refer to as *strong* components. Thus in Figure 4.7, the component containing Q splits into two strong components, namely Q and A, B, P, C, D.

Definition 4.22 If pixels P and Q from two different strong components are adjacent, we say that they form a *bridge* between (or joining) those components. In this case, P, Q have one of the positions along a diagonal line as shown in Figure 4.8, since they are, by definition, adjacent but not edge adjacent, having only a vertex in common.



Figure 4.8 The two ways in which a pair of pixels in the same piece may form a bridge.

Note There cannot be more than one bridge between two edge components, for otherwise the jigsaw piece would contain a hole.

Our first result is a criterion, already implied by Theorem 4.16, for the absence of one or other type of bridge from all pieces of a particular jigsaw. It is geared to ruling out one type of bridge, even if the other type may be present.

Corollary 4.23 (from Theorem 4.16) *The jigsaw of a map contains no rising bridge if $ab, cd \geq 0$, and no descending bridge if $ab, cd \leq 0$.*

Clearly, in order for Corollary 4.23 to rule out both types of bridge, we must have $ab = cd = 0$, i.e. one of a, b must equal zero, as must one of c, d . However, this is exactly the hypothesis of Corollary 4.18, and therefore each piece will not only not contain bridges, but will in fact be rectangular. This suggests that Corollary 4.23 is not the strongest result which we can obtain for pieces not containing bridges - we now derive a test which, without requiring pieces to be rectangular, rules out both types of bridge. It guarantees that every component of a jigsaw piece is strong, and therefore simplifies the situation, especially for connected pieces. Before stating the theorem, we need a lemma.

Lemma 4.24 *Suppose that $H, \alpha, \beta, \delta \in \mathbf{R}$, and that H and $H + \alpha + \beta$ lie in the interval $[\delta, \delta + 1)$. If $|\alpha - \beta| \leq 1$ then at least one of $H + \alpha$ and $H + \beta$ also lies in the interval $[\delta, \delta + 1)$.*

Proof If α, β have the same sign the conclusion of the theorem is trivial and does not require the condition $|\alpha - \beta| \leq 1$. By the symmetry of α, β in this result, we need only consider the case $\alpha \geq 0, \beta \leq 0$. Then $\delta \leq H \leq H + \alpha$ and $H + \beta \leq H < \delta + 1$. If $H + \alpha < \delta + 1$ then $H + \alpha$ lies in $[\delta, \delta + 1]$ and the lemma is proven. If this is not the case, then $H + \beta = H + \alpha + (\beta - \alpha) \geq \delta + 1 - 1 = \delta$, since $H + \alpha \geq \delta + 1$ and $\beta - \alpha \geq -1$. Hence $H + \beta \in [\delta, \delta + 1)$, and the lemma is proven.

Theorem 4.25 *Let (x, y) and $(x + i, y + j)$ be in the same piece $P_{u,v}$. If either*

(i) $|ai - bj| \leq 1$ and $cdij \geq 0$, or

(ii) $|ci - dj| \leq 1$ and $abij \geq 0$

then at least one of $(x + i, y)$ and $(x, y + j)$ is in $P_{u,v}$.

Proof We prove (i) only, since (ii) simply interchanges the roles of a, b and c, d . Thus we suppose that $|ai - bj| \leq 1$ and $cdij \geq 0$, which means that ci and dj have the same sign. If ai and bj have the same sign then the result follows from Theorem 4.16, without assuming that $|ai - bj| \leq 1$, and so we assume that ai, bj have opposite signs. We apply Lemma 4.24 with $H = w_h(x, y)$, $\alpha = ai$, $\beta = bj$ and $\delta = u - 1/2$. Since it is given that $w_h(x, y)$ and $w_h(x + i, y + j)$ round to u , we have that H and $H + \alpha + \beta$ lie in the interval $[\delta, \delta + 1)$. Also, $|\alpha - \beta| \leq 1$. Thus the hypotheses of the lemma hold, and we may conclude that at least one of $w_h(x + i, y)$ and $w_h(x, y + j)$ round to u . Finally, since ci and dj are assumed to have the same sign, Theorem 4.16 implies that both $w_v(x + i, y)$ and $w_v(x, y + j)$ round to v , completing the proof.

Applying the result of Theorem 4.25 in the cases $(i, j) = (1, 1)$ and $(1, -1)$ gives the desired test, as in Corollary 4.26.

Corollary 4.26 (Ruling out bridges) *Suppose that (i) ($|a - b| \leq 1$ and $cd \geq 0$) or ($|c - d| \leq 1$ and $ab \geq 0$), and (ii) ($|a + b| \leq 1$ and $cd \leq 0$) or ($|c + d| \leq 1$ and $ab \leq 0$) for some IFS map w . Then no jigsaw piece of w contains a bridge.*

Example 4.27 Map 3 of the Von Koch Curve is given by

$$w(x, y) = (1/6x - \sqrt{3}/6y + 99/2, \sqrt{3}/6x + 1/6y + 33\sqrt{3}/2)$$

and so we have $a = 1/6$, $b = -\sqrt{3}/6$, $c = \sqrt{3}/6$ and $d = 1/6$. Then condition (i) of Corollary 4.26 is satisfied by $|a - b| < 1$ and $cd > 0$, while (ii) is satisfied by $|c + d| < 1$ and $ab < 0$. Thus the jigsaw contains no bridges.

The result exhibited in Example 4.27 is just one example of a general result for maps such as map 3 of the Von Koch Curve, which consist of uniform scaling, followed by rotation.

Theorem 4.28 *The jigsaws of IFS maps which consist of uniform scaling, followed by rotation about the origin contain no bridges.*

Proof Recall from Section 2.3.1 that such maps have matrix given by

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} r\cos\theta & -r\sin\theta \\ r\sin\theta & r\cos\theta \end{bmatrix}$$

where $r \in (0, 1)$ is the scale factor, and θ is the angle of rotation. Then $|a - b| = |r| |\cos\theta + \sin\theta| = |c + d|$ and $|a + b| = |r| |\cos\theta - \sin\theta| = |c - d|$. Hence if $|a - b|$ or $|a + b|$ exceeds 1 then every piece is flat, by Corollary 4.13, and certainly cannot contain a bridge. This reduces conditions (i) and (ii) of Corollary 4.26 to requiring that one of the products ab , cd is non negative, while the other is non positive - a requirement which is satisfied because $ab = -cd$.

In fact, Corollary 4.26 enables us to show that the vast majority of the maps of our 50 IFS's cannot have jigsaws which contain bridges. Out of the 202 maps which make up the 50 IFS's, only 7 have jigsaws which may contain bridges. In Table 4.1, we list those maps which violate one of the conditions of Corollary 4.26, together with which condition they violate - which is easily checked from the given values of a, b, c and d . In addition to this, we give an example of a grid pixel (x, y) which starts a bridge, noting that if condition (i) is violated then (x, y) starts a rising bridge, while if condition (ii) is violated, (x, y) starts a descending bridge. It is easy to check that these values of (x, y) do indeed start bridges, recalling that the map is given by

$$w(x, y) = (ax + by + e \times \text{scale}, cx + dy + f \times \text{scale})$$

where scale is as given in the table, and that for rising bridges we require $\underline{w}(x, y) = \underline{w}(x + 1, y + 1)$ but $\underline{w}(x + 1, y), \underline{w}(x, y + 1) \neq \underline{w}(x, y)$, while for descending bridges we require $\underline{w}(x, y) = \underline{w}(x + 1, y - 1)$ but $\underline{w}(x + 1, y), \underline{w}(x, y - 1) \neq \underline{w}(x, y)$.

IFS Map	Code	Scale	Condition Violated	(x, y)
Comet's Tail 2	(0.5, -0.506, 0, 0.5, 0.493, 0.554)	80	(i)	(1, 73)
Leaf Outline 2	(0.176, 0.25, -0.7, 0.433, 0.547, 0.5)	100	(i)	(13, 12)
Leaf Outline 3	(0.176, -0.25, 0.7, 0.433, 0.248, -0.187)	100	(ii)	(12, 19)
Sierpinski Peak 1	(0.5, 0, 0.506, 0.5, -0.048, -0.306)	50	(ii)	(-34, -99)
Sierpinski Peak 2	(0.5, 0, -0.506, 0.5, 0.559, 0.2)	50	(i)	(0, 16)
Triangle Crab 1	(0.7, 0.2, 0.2, 0.5, 0, 0)	80	(ii)	(2, 6)
Triangle Crab 3	(0.7, 0.2, -0.2, -0.5, 0, 1)	80	(ii)	(0, 3)

Table 4.1 IFS maps whose jigsaws may contain bridges.

Note We see from Table 4.1 that $(x, y) = (-34, -99)$ starts a descending bridge for map 1 of the Sierpinski Peak. However, at the given scale, (x, y) does not lie in the grid of the IFS - and in fact we cannot find any pixel within the grid which starts a bridge. Thus, for our purposes, we may regard map 1 of the Sierpinski Peak as having no bridges in its jigsaw.

In IFS maps whose jigsaws contain bridges, it is often difficult to locate the bridges - and harder still to find the jigsaw pieces - and so we would clearly prefer that our jigsaws did not contain bridges. In fact we will see in later chapters that, for most of the maps listed in Table 4.1, it is not only the fact that the jigsaws contain bridges which makes them difficult to deal with. However, in introducing the method which we use to improve the GA, we will consider only maps whose jigsaws do not contain bridges.

In the final section of this chapter, we will describe a property which seems to occur even less frequently than bridges, but cannot be discounted since it does apply to a few of the maps of our 50 IFS's - that of gaps.

4.4 Gaps in Jigsaw Pieces

Recall that a jigsaw piece is disconnected if there exist pixels of the piece which are not linked by a path. That is, the piece may be divided into distinct sets S and T with no path from any pixel of S to any pixel of T - where S and T themselves may consist of one or more connected components. Measuring the distance between pixels $P = (x, y)$ and $Q = (x', y')$ by $|x - x'| + |y - y'|$, we define the distance between S and T by

$$d_{\min} = \text{Min}\{d(P, Q) : P \in S, Q \in T\}. \quad (4.4)$$

Then we have the following definitions.

Definition 4.29 Any two pixels P and Q attaining the minimum (4.4) will be called a *gap* between S and T . Further, we may view P, Q as an ordered pair, and say that the gap P, Q is of *type* (m, n) , where $m = x' - x$ and $n = y' - y$ - so that

$$d_{\min} = |m| + |n|.$$

Whenever we are discussing gaps, we will label so that P is to the left of Q (i.e. $m > 0$). Then the gap is *rising* if $n > 0$ and *descending* if $n < 0$ - Figure 4.9 illustrates a rising $(3, 1)$ gap.

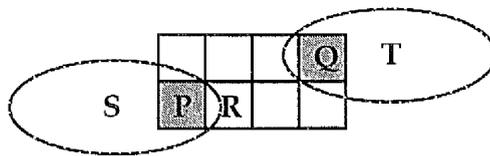


Figure 4.9 The pair P, Q form a $(3, 1)$ gap between S and T . Pixel R must be outside the piece, as must the rest of $\text{Rect}(P, Q)$ - see Remarks 4.30 (i) below.

Note that if no gap exists in a jigsaw piece, then the piece is connected, and also that a gap of type $(m, 0)$ cannot occur, since this would imply either that S and T are connected ($m = 1$), or that a hole occurs ($m > 1$), and similarly no gap of type $(0, n)$ can occur.

Remarks 4.30 (i) If P, Q is a gap then every other pixel R in the rectangle $\text{Rect}(P, Q)$ with diagonal P, Q lies outside the piece. For example, if R is in S (and therefore not in T) then $d(R, Q) < d(P, Q)$ - a contradiction.

(ii) d_{\min} is not less than 3. For if $d_{\min} = 1$ then P and Q must be edge adjacent, whilst $d_{\min} = 2$ implies either a hole (e.g. $m = 2, n = 0$), or that P and Q are diagonally adjacent (e.g. $m = n = 1$).

(iii) The integers m, n are non-zero, but with no common factor. Such a common factor implies a pixel of the piece lying in $\text{Rect}(P, Q)$, by Corollary 4.6.

(iv) It suffices to consider only (m, n) gaps from the origin. For if (α, β) , $(\alpha + m, \beta + n)$ is a gap, we may define a new map $w'(x, y) = w(x, y) - (\alpha, \beta)$ which has a gap from $(0, 0)$ to (m, n) since $\underline{w}'(x, y) = \underline{w}(x, y) - (\alpha, \beta)$ (due to the fact that α and β are integers).

We now give the main result of this section - which will immediately allow us to state that all but 5 of the maps of our 50 IFS's are connected.

Theorem 4.31 *If the jigsaw of a map w has a rising gap, then $ab, cd < 0$, while if it has a descending gap then $ab, cd > 0$. Hence if $abcd \leq 0$, the jigsaw pieces of w are connected.*

Corollary 4.32 *If w is rotation combined with (possibly non-uniform) scaling, then the jigsaw pieces of w are connected.*

Proof In the code of such an IFS map, one of a, b, c, d has different sign from the other three - and thus $abcd \leq 0$.

Corollary 4.33 *If w consists of coordinate scaling followed by shear with parameter α , along a line making angle θ with the x -axis, then its jigsaw pieces are connected if either the shear is along one of the coordinate axes, or $\alpha \leq 2/|\sin 2\theta|$.*

Proof Recall from Section 2.3.1 that the matrix of a shear with parameter α along a line making angle θ with the x -axis is given by

$$\begin{bmatrix} 1 + \alpha \sin \theta \cos \theta & \alpha \cos^2 \theta \\ -\alpha \sin^2 \theta & 1 - \alpha \sin \theta \cos \theta \end{bmatrix}$$

If we perform coordinate scaling first, then we multiply each entry of the first column by r_1 and each entry of the second column by r_2 . Then clearly

$$\begin{aligned}abcd &= -r_1^2 r_2^2 \alpha^2 \cos^2 \theta \sin^2 \theta (1 - \alpha \sin \theta \cos \theta)(1 + \alpha \sin \theta \cos \theta) \\ &= -r_1^2 r_2^2 \alpha^2 \cos^2 \theta \sin^2 \theta (1 - \alpha^2 \cos^2 \theta \sin^2 \theta)\end{aligned}$$

Thus for $abcd \leq 0$, we need either $\cos \theta = 0$ or $\sin \theta = 0$ - implying that θ is a multiple of $\pi/2$ and the shear is along one of the coordinate axes - or $(1 - \alpha^2 \cos^2 \theta \sin^2 \theta) \geq 0$. Now

$$\begin{aligned} &1 - \alpha^2 \cos^2 \theta \sin^2 \theta \geq 0 \\ \Rightarrow &\alpha^2 \cos^2 \theta \sin^2 \theta \leq 1 \\ \Rightarrow &\frac{1}{4} \alpha^2 \sin^2 2\theta \leq 1 \\ \Rightarrow &\alpha^2 \leq \frac{4}{\sin^2 2\theta} \\ \Rightarrow &\alpha \leq \frac{2}{|\sin 2\theta|}\end{aligned}$$

and the result is proved.

For the remainder of this section, we shall minimise the use of fractions by writing

$$A = \frac{1}{2} \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \quad w(x, y) = \frac{1}{2}(ax + by + e, cx + dy + f) \quad (4.5)$$

This implies that a, b, c, d lie in the open interval $(-2, 2)$. Further, $\underline{w}(p, q) = 0$ if and only if both $ap + bq + e$ and $cp + dq + f$ lie in the half open interval $[-1, 1)$. That is, (a, b) as a point in the xy -plane lies between parallel lines $px + qy + e = \pm 1$, but not on $px + qy + e = 1$, and similarly, (c, d) lies between lines $px + qy + f = \pm 1$. In particular, $\underline{w}(0, 0) = (0, 0) = \underline{w}(m, n)$ dictates that e and f lie in the interval $[-1, 1)$ and, as illustrated in Figure 4.10,

$$\begin{aligned} (a,b) \text{ lies between } mx + ny + e = \pm 1, \text{ and} \\ (c,d) \text{ lies between } mx + ny + f = \pm 1 \end{aligned} \tag{4.6}$$

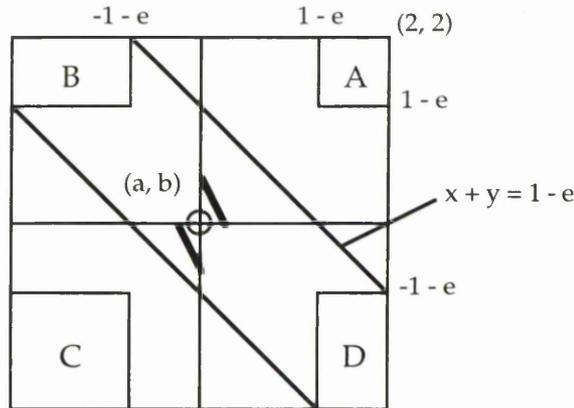


Figure 4.10 If a gap exists from the origin O to (m, n) then (a, b) must lie between parallel lines $mx + ny + e = \pm 1$, shown in bold. See discussion above.

We are now in a position to prove Theorem 4.31.

Proof of Theorem 4.31 Let the jigsaw of w have a gap from $P = (0, 0)$ to $Q = (m, n)$, where $m > 0$. Suppose that this gap is rising, but that $cd \geq 0$ - we shall obtain a contradiction which will imply $cd < 0$. By Theorem 4.16(ii), the pixels in $\text{Rect}(P, Q)$ all have $\underline{w}_v = 0$, and so every one of these pixels (x, y) , except P and Q , must satisfy $\underline{w}_h \neq 0$, since they do not lie in $P_{0,0}$. This holds in particular for $(1, 0)$, $(0, 1)$ and $(1, 1)$ and hence (a, b) must lie in the region of the xy -plane defined by $x, y, x + y \notin [-1 - e, 1 - e]$. The conditions on x and y individually are represented by regions A to D in Figure 4.10, and since (a, b) must not lie between parallel lines $x + y = -1 - e$ and $x + y = 1 - e$ (or, indeed, on the first), we have that (a, b) must lie in either region A or region C.

Now, the line $mx + ny + e = 1$ cuts the coordinate axes at $x = (1 - e)/m$ and $y = (1 - e)/n$, which are positive and do not exceed the corresponding intercepts of $x + y = 1 - e$ because $e < 1$ and $m, n \geq 1$. Thus the line $mx + ny + e = 1$ lies strictly

between the origin and region A. Similarly, $mx + ny + e = -1$ lies strictly between the origin and region C, the only exception occurring in the case $e = -1$ when the origin is actually on this line - however, since the origin is not in region C this causes no problem. Therefore no point of regions A or C can lie between or on the lines $mx + ny + e = \pm 1$ (shown bold in Figure 4.10) - which contradicts (4.6), enabling us to conclude that $cd < 0$. Similarly, if we replace e by f in Figure 4.10, we may conclude that $ab < 0$.

Finally, suppose that the gap at P is descending. Consider the jigsaw of w' defined by $w'(x, y) = w(x, -y)$. Since $\underline{w}'(x, y) = \underline{w}(x, -y)$, the w' jigsaw has an $(m, -n)$ gap, which is rising because $-n > 0$. Hence the first case applies to the matrix of w' , which is obtained from A by changing signs in the second column. Thus $a(-b)$, $c(-d) < 0$, or $ab, cd > 0$, as required. This completes the proof.

Corollary 4.34 *If any element of the matrix of w is zero then the jigsaw pieces of w are connected.*

Theorem 4.31 (noting Corollaries 4.32 to 4.34) establishes connectedness for every map in our list of 50 IFS's, except for Fish 9, Leaf 2, Triangle Crab 1 and 3, and Wheat 4. In each case, Corollary 4.22 rules out a rising bridge. In fact, neither of the Triangle Crab maps have gaps in their jigsaws.

Example 4.35 A computer search for a map with coefficients that are 1-digit decimals, with zero translation components (which is, as we shall see later, is equivalent to *integral* translation components), and whose jigsaw has a gap from the origin to $(2, 1)$ yields many cases, but only one which has the necessary property of being contractive - that is $(a, b, c, d) = 1/10(-1, 6, 5, -6)$. In agreement with Theorem 4.31, we have $abcd$ positive, and descending bridges are ruled out

by Corollary 4.22. However, rising bridges are present along with gaps, as Figure 4.11 shows.

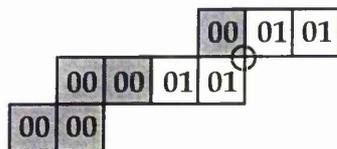


Figure 4.11 Part of the jigsaw of Example 4.35, showing a gap in piece $P_{0,0}$ and a bridge in piece $P_{0,1}$, the presence of which is indicated by a circle.

To end this section, we give a result suggested by Example 4.35.

Theorem 4.36 *Let the translation components e, f of the map w be integers. If the jigsaw of w has an (m, n) gap then $|m| + |n| = 3$. Further, the contraction ratio of w exceeds $(1 + \sqrt{5})/4$.*

Proof Since the translation components are integral, they can be replaced by any integral values, in particular by $e = f = 0$. Let $m > n \geq 1$. With the same notation as before, we have that $a, b, c, d \in [-2, 2]$ and $\underline{w}(0, 0) = (0, 0)$.

Since $w(m, n) = 1/2(am + bn, cm + dn) \sim (0, 0)$ we have that $am + bn, cm + dn \in [-1, 1)$. Thus the points $(a, b), (c, d)$ lie in the xy -plane between lines $mx + ny = \pm 1$, but not on $mx + ny = 1$, and in the square region $-2 \leq x, y \leq 2$, as shown in Figure 4.12 below.

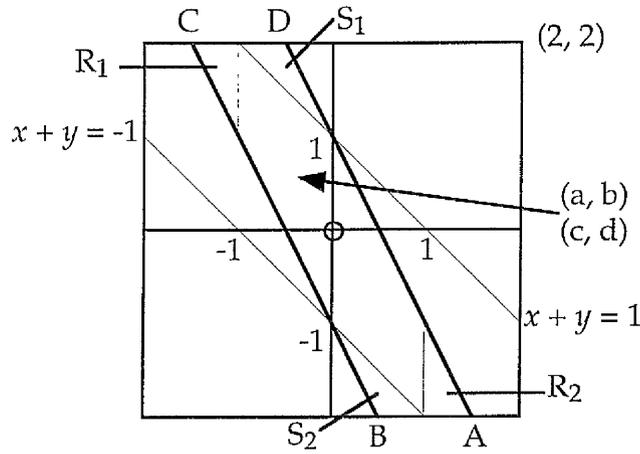


Figure 4.12 If a gap exists from the origin 0 to the point (m, n) then (a, b) and (c, d) must lie in the area ABCD.

As we see from Figure 4.12, the lines $mx + ny = \pm 1$ have four points of intersection with $y = \pm 2$, given by

$$x_A = \frac{2n+1}{m} = -x_C, \quad x_B = \frac{2n-1}{m} = -x_D.$$

These values must lie in $[-2, 2]$ as $m > n \Rightarrow 2m \geq 2n \pm 1 \Rightarrow 2 \geq 2n \pm 1/m$. On the other hand, $x_A \leq 1 \Rightarrow -1 \leq a, c < 1$ (with the strict inequality arising from the fact that $(a, b), (c, d)$ do not lie on $mx + ny = 1$), implying in turn that $w(1, 0) = 1/2(a, c) \sim (0, 0)$, a contradiction. Hence $x_A > 1$, and so $m < 2n + 1$ i.e. $m \leq 2n$.

Suppose that $(m, n) \neq (2, 1)$. Then $m \leq 2n - 1$ since m, n are coprime. Now, because the x -intercept $1/m$ of AD is less than its y -intercept $1/n$, the extremal values of $x + y$ in ABCD occur at B and D. At B, $x + y = (2n - 1)/m - 2 \geq 1 - 2 = -1$ while, at D, $x + y = 1 - 2n/m + 2 \leq -1 + 2 = 1$. Thus $-1 \leq x + y \leq 1$ on region ABCD. However, since $(a, b), (c, d)$ lie in ABCD but not on AD we have

$$a + b, c + d \in [-1, 1).$$

Hence $w(1, 1) = 1/2(a + b, c + d) \sim (0, 0)$, which is a contradiction. Thus our assumption that $(m, n) \neq (2, 1)$ cannot hold and we may conclude that $m = 2, n = 1$, giving $|m| + |n| = 3$. The remaining cases can be derived from this, as in the general gap theorem (Theorem 4.31).

With $m = 2, n = 1$, we have $x_B = 1/2$ and $x_A = 3/2$, as shown in Figure 4.12. Let $I = [-1, 1)$, noting that $x/2 \sim 0$ if and only if $x \in I$.

Claim Let (x, y) lie in area ABCD (excluding AD). Then

$$(i) x \notin I \Rightarrow y \notin I, x + y \in I,$$

$$(ii) x + y \notin I \Rightarrow |y| \geq 1.$$

Proof Consider Figure 4.12. (i) $x < -1 \Rightarrow y > 1, 0 \leq x + y < 1$ (region R_1), while $x \geq 1 \Rightarrow y < -1, -1 \leq x + y < 0$ (region R_2). (ii) $x + y < -1 \Rightarrow y \leq -1$ (region S_2), while $x + y \geq 1 \Rightarrow y \geq 1$ (region S_1).

Returning to the proof of Theorem 4.36 we note that (a, b) and (c, d) lie in the region given in the Claim. Also

$$w(1, 0) = 1/2(a, c) \sim (0, 0) \text{ implies that either } a \notin I \text{ or } c \notin I \quad (1)$$

$$\text{while } w(1, 1) = 1/2(a + b, c + d) \sim (0, 0) \text{ implies } a + b \notin I \text{ or } c + d \notin I \quad (2)$$

By symmetry we may suppose from (1) that $a \notin I$. Then $b \notin I$ and $a + b \in I$ by Claim(i). This shows that

$$|a|, |b| \geq 1 \quad (3)$$

By (2), $c + d \notin I$, and Claim(ii) says that

$$|d| \geq 1 \quad (4)$$

Remark The contraction ratio of a matrix A is unaffected by (i) sign change of the elements of a row, column or diagonal, or (ii) interchanging rows or columns. Thus if A has an even number of non-positive coefficients, the same ratio is obtained if we replace each entry by its modulus. Further, the ratio of the map is given by $\max \|Az\|$ for $\|z\| = 1$. Taking $z = (x, y)$, with $x^2 + y^2 = 1$, we have $\|Az\|^2 = (ax + by)^2 + (cx + dy)^2$ - which is maximised for x, y non-negative, and clearly increases as a, b, c or d increase.

Now, by (3), (4) and the Remark, the contraction ratio is bounded below by that of

$$A = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix},$$

noting that $|c|$ must be strictly greater than zero, as otherwise $abcd = 0$ and no gap occurs by Theorem 4.31.

Recall from (2.5) that the ratio is given by the formula

$$s = \sqrt{\frac{p + \sqrt{p^2 - 4q}}{2}}$$

where $p = a^2 + b^2 + c^2 + d^2$ and $q = (ad - bc)^2$. Here, $a = b = d = 0.5$ and $c = 0$ so that $p = 3/4$, $q = 1/16$. Then $s^2 = (3 + \sqrt{5})/8$, which is a perfect square with solution $s = (1 + \sqrt{5})/4$. Thus, since $|c| > 0$, we have $s > (1 + \sqrt{5})/4$, and the Theorem is proved.

We have seen in this section that, while we cannot completely rule out gaps, there is evidence that 'most' IFS maps do not have gaps in their jigsaws. For those with integral translation components which do have gaps, the gap will be of type $(\pm 2, \pm 1)$ or $(\pm 1, \pm 2)$ - giving a distance between components of 3. We will give further consideration to maps whose jigsaws have gaps in Chapter 9. However, we are now in a position to consider particular jigsaws and to describe how they can be used to improve the performance of the Graphical Algorithm. In Chapters

5 - 8, the jigsaws which we consider will not contain bridges or gaps - and indeed many will satisfy the conditions for rectangular pieces discussed in Section 4.2.

Chapter 5

Improving the Graphic Algorithm

In Chapter 3 we introduced the concept of *jigsaws* for IFS maps, and hinted that they could be used to improve considerably the performance of the Graphical Algorithm. In this chapter, and also in Chapter 6, we will explain exactly how we use jigsaws to improve the GA, and give several examples which will show just how effective these improvements can be - further examples are given in [15]. We begin by showing how to find the jigsaw pieces for a number of simple IFS maps.

5.1 Finding the Jigsaws

Recall from Example 3.6 that there are four distinct points which, when map 1 of the Takagi Function is applied to any of them, result in the point (24, 54) being hit, and that these points, when represented as a set of screen pixels, lie in the configuration shown in Figure 3.4 - which we called the *jigsaw piece* for (24, 54). In Example 5.6 we will show that this configuration is, in fact, the unique one for map 1 of the Takagi Function, and we will see that the jigsaw pieces for map 2 are similar. However, as we mentioned in Section 3.3, there are many IFS maps whose jigsaw pieces are simpler than those of the Takagi Function - and it is with such maps that we begin our discussion on finding the jigsaws.

Notation 5.1 Recall that we use the symbol $\sim>$ to mean 'rounds to', and that $\underline{w}(x, y)$ denotes the result of rounding $w(x, y)$ to the nearest integer point.

5.1.1 Simple Jigsaws

As we showed in Section 4.2, the simplest shape of jigsaw piece arising from an IFS map is a rectangular piece where, in the resulting jigsaw, every piece has four others attached along its edges, as in Figure 5.1. Note that each of these jigsaw pieces consists of a rectangular set of pixels.

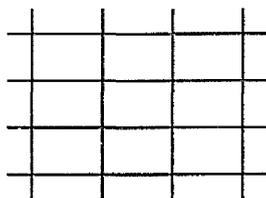


Figure 5.1 Part of the simplest type of jigsaw.

The following examples all have this type of jigsaw, with differing sizes of jigsaw piece.

Note Throughout this thesis, we will use $w(x, y) = (ax + by + e^*, cx + dy + f^*)$ to define an IFS map, where we use $e^* = e \times \text{scale}$ and $f^* = f \times \text{scale}$, $\text{scale} \in \mathbf{Z}$, in order to produce an attractor of a particular size.

Example 5.2 Recall that the three maps of the *Sierpinski Gasket* at scale 100 are given by $w_i(x, y) = (0.5x, 0.5y) + t_i$, where $t_1 = (0, 0)$, $t_2 = (50, 0)$ and $t_3 = (0, 50)$. Suppose that a point (u, v) is hit by map 1, so we have $0.5x \rightsquigarrow u$ and $0.5y \rightsquigarrow v$. Now

$$0.5x \rightsquigarrow u \Leftrightarrow 0.5x = u \text{ or } u - 1/2 \Leftrightarrow x = 2u \text{ or } 2u - 1$$

and similarly,

$$0.5y \rightsquigarrow v \Leftrightarrow y = 2v \text{ or } 2v - 1.$$

Thus we have that $w_1(x, y) \rightsquigarrow (u, v)$ if and only if $(x, y) = (2u, 2v)$ or $(2u, 2v - 1)$ or $(2u - 1, 2v)$ or $(2u - 1, 2v - 1)$, and hence the jigsaw pieces for map 1 of the

Sierpinski Gasket are as shown in Figure 5.2, where the shaded pixel has both coordinates even.

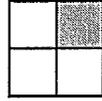


Figure 5.2 A jigsaw piece for map 1 of the Sierpinski Gasket.

Since the top right hand pixel of the piece must have both coordinates even, it is clear that there will be no overlap between pieces, and also that the pieces will fit together to cover the whole attractor - as we showed in general in Theorem 4.3.

Now suppose that a point (u, v) is hit by map 2. Then we require $0.5x + 50 \rightsquigarrow u$, i.e. $x = 2u - 100$ or $2u - 101$, while the vertical coordinate is the same as map 1. Thus the jigsaw pieces for map 2 are also 2×2 squares, and the pixel with both coordinates even is once again in the top right corner - i.e. the jigsaw pieces for map 2 are identical to those for map 1. Finally, it is easy to see that the same is true for the jigsaw pieces for map 3, and therefore the Sierpinski Gasket has the special property that all N maps have the same jigsaw - this jigsaw is shown in Figure 5.3.

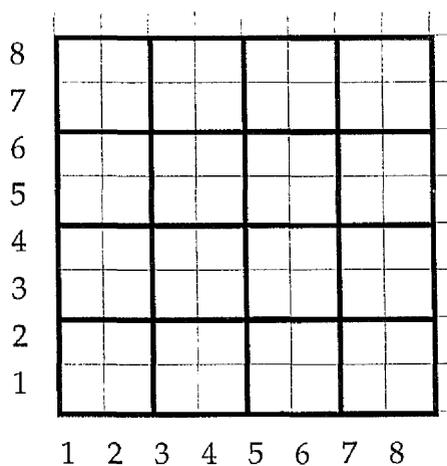


Figure 5.3 A small section of the jigsaw for each map of the Sierpinski Gasket.

As we can see from Figure 5.2, the jigsaw pieces for the maps of the Sierpinski Gasket are extremely simple, being just 2×2 squares of pixels, and consequently the jigsaw is simple, as we saw in Figure 5.3. In fact, these are amongst the simplest pieces which we will encounter and, as we will see shortly, this simplicity will be an important factor in the improvements which we shall make, as will the fact that all N maps have the same jigsaw. The next example also has this property, but, as we will see later, it offers even more scope for improvement than the Sierpinski Gasket as it has more maps.

Note 5.3 In some Iterated Function Systems, two or more maps have the same matrix, so that the maps differ only in their translation parts. If, in addition, the translation coordinates differ by integers, then the jigsaws of the maps will be the same - as we saw in the Sierpinski Gasket. Throughout this thesis, we will refer to the jigsaws of such maps as *equivalent*, or *identical*.

Example 5.4 The *Sierpinski Carpet* has eight maps, each of which consists of uniform scaling by $1/3$ and a translation. At scale 99 the maps are given by

$$w_i(x, y) = (1/3x + 33j, 1/3y + 33k), \text{ where } 0 \leq j, k \leq 2, (j, k) \neq (1, 1),$$

and the attractor is shown in Figure 5.5(a). Suppose that a point (u, v) is hit by map i . Then

$$1/3x + 33j \sim> u \Leftrightarrow x = 3u - 1 - 99j, 3u - 99j \text{ or } 3u + 1 - 99j \quad (5.1a)$$

and similarly,

$$1/3y + 33k \sim> v \Leftrightarrow y = 3v - 1 - 99k, 3v - 99k \text{ or } 3v + 1 - 99k \quad (5.1b)$$

Thus the jigsaw piece for map i consists of the nine points (x, y) where x is given by (5.1a) and y is given by (5.1b) - and clearly the eight jigsaws are equivalent. Note that for any valid pair (j, k) , the point $(3u - 99j, 3v - 99k)$ is the unique pixel of

the jigsaw piece with both coordinates divisible by 3. The jigsaw piece for the Sierpinski Carpet is shown in Figure 5.4, where the shaded pixel is $(3u - 99j, 3v - 99k)$.

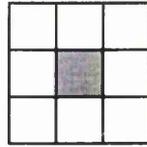


Figure 5.4 A jigsaw piece for the maps of the Sierpinski Carpet.

Thus the pieces are rectangular - just as we expected, by Corollary 4.18 - and they fit together in much the same way as in the Sierpinski Gasket, forming a simple jigsaw.

Note The results of Example 5.4 also apply to another well known IFS - the *Peano Curve* - which has nine maps, each of which consists of either scaling by $1/3$, or rotation by a multiple of $\pi/2$ followed by scaling by $1/3$. The attractor is shown in Figure 5.5(b).

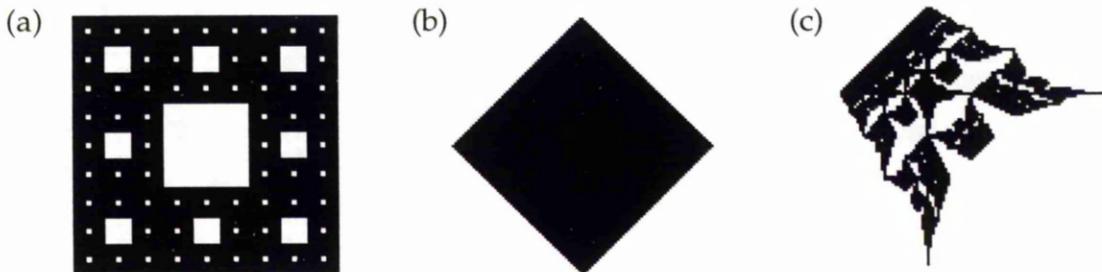


Figure 5.5 The attractors of (a) the Sierpinski Carpet, (b) the Peano Curve, and (c) the Flamboyant Crown.

In Examples 5.2 and 5.4, we considered two IFS's whose maps all have the same jigsaw. However, this is generally not the case - the following example shows an IFS whose five maps have five different jigsaws.

Example 5.5 The *Flamboyant Crown* has five maps, given at scale 66 by

$$w_1(x, y) = (-0.25x, 0.5y)$$

$$w_2(x, y) = (0.5x - 16.5, 0.5y + 33)$$

$$w_3(x, y) = (-0.25x + 16.5, -0.25y + 66)$$

$$w_4(x, y) = (0.5x, 0.5y + 49.5)$$

$$w_5(x, y) = (0.5x + 33, -0.25y + 82.5),$$

and the attractor is shown in Figure 5.5(c).

We can find the jigsaw pieces in exactly the same way as we have in the previous examples - Figure 5.6 shows the jigsaw pieces for each of the maps, where the shaded pixels represent the unique pixel whose horizontal coordinate is divisible by the number of horizontal pixels in the piece, and whose vertical coordinate is divisible by the number of vertical pixels in the piece.

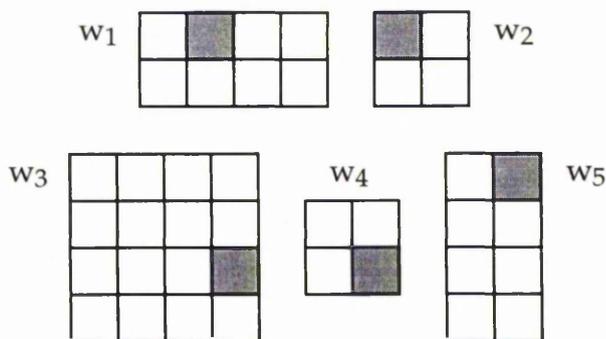


Figure 5.6 The jigsaw pieces for the maps of the Flamboyant Crown.

Note We stated at the start of this example that the five maps have five different jigsaws. Although the jigsaw pieces for maps 2 and 4 are both 2×2 squares of pixels, the jigsaws of these maps are different because the shaded pixel is not in the same position in the two pieces. This difference is illustrated in Figure 5.7 below.

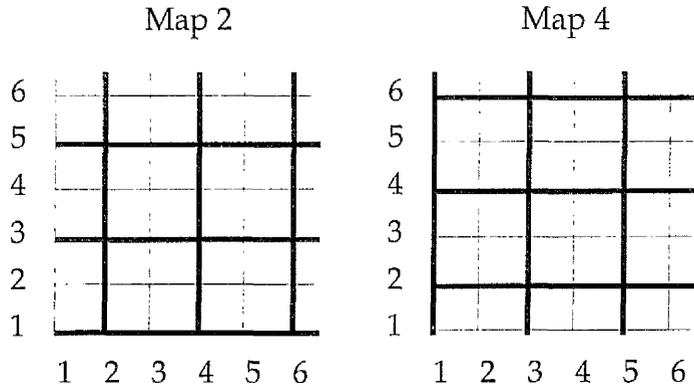


Figure 5.7 Part of the jigsaws for maps 2 and 4 of the Flamboyant Crown.

In Examples 5.2, 5.4 and 5.5 we saw three different IFS's, all of whose maps were shown to have rectangular jigsaw pieces. In fact, we could have immediately stated this, by virtue of Corollary 4.18, as in each of these examples we had $ab = cd = 0$. In such cases, the jigsaw pieces can be found from the following simple observation - if z , n and c are integers, and z/n rounds up to c then we have

$$\begin{aligned} nc - \left\lceil \frac{n}{2} \right\rceil &\leq z < nc + \left\lceil \frac{n}{2} \right\rceil \text{ for } n \text{ even} \\ nc - \left\lceil \frac{n}{2} \right\rceil &\leq z \leq nc + \left\lceil \frac{n}{2} \right\rceil \text{ for } n \text{ odd.} \end{aligned} \quad (5.2)$$

Using this for both horizontal and vertical coordinates would lead to the same results as we obtained. However, it is important to note that not all IFS maps have such simple jigsaws, as we saw with the Takagi Function - we now consider maps with more complicated jigsaws.

5.1.2 More Complicated Jigsaw Pieces

As we saw in Example 3.6, map 1 of the Takagi Function has at least one jigsaw piece which is more complicated than the rectangular pieces of the last section, and we noted that this piece was in fact the only shape of piece in the jigsaw. We

now prove that this is the case, and show that, although the pieces are not as simple as those of the last section, there is still regularity in the jigsaw.

Example 5.6 Recall that the *Takagi Function* has two maps, given at scale 100 by

$$w_1(x, y) = (0.5x, 0.5x + 0.5y)$$

$$w_2(x, y) = (0.5x + 50, -0.5x + 0.5y + 50)$$

Suppose that a point (u, v) is hit by map 1. Then $0.5x \rightsquigarrow u$ and $0.5(x + y) \rightsquigarrow v$. As in Example 5.2, $0.5x \rightsquigarrow u \Leftrightarrow x = 2u$ or $2u - 1$, and similarly,

$$0.5(x + y) \rightsquigarrow v \Leftrightarrow x + y = 2v \text{ or } 2v - 1,$$

leading to the four possibilities for (x, y) shown in Table 5.1.

$x + y$	x	y
$2v$	$2u$	$2(v - u)$
$2v$	$2u - 1$	$2(v - u) + 1$
$2v - 1$	$2u$	$2(v - u) - 1$
$2v - 1$	$2u - 1$	$2(v - u)$

Table 5.1 The four possible (x, y) with $w_1(x, y) = (u, v)$.

We can find the four possible values of (x, y) for map 2 in much the same way - these are shown in Table 5.2.

$y - x$	x	y
$2v - 100$	$2u - 100$	$2(u + v) - 200$
$2v - 100$	$2u - 101$	$2(u + v) - 201$
$2v - 101$	$2u - 100$	$2(u + v) - 201$
$2v - 101$	$2u - 101$	$2(u + v) - 202$

Table 5.2 The four possible (x, y) with $w_2(x, y) = (u, v)$.

Note that in Table 5.1 the top and bottom rows give the same value of y , whilst in Table 5.2, it is the second and third rows which give equivalent values of y . This implies that the jigsaw pieces for map 1 will be different from those of map 2 - as Figure 5.8 shows.



Figure 5.8 The jigsaw pieces for (a) Takagi map 1, and (b) Takagi map 2.

The shaded pixels represent those with both coordinates even.

Figure 3.5 showed how the jigsaw pieces for map 1 fit together to form the jigsaw covering - and it is easy to see that the pieces for map 2 will fit together in much the same way. Note that the pieces for map 1 have no rising diagonal, as noted in Example 4.14. Similarly, the pieces for map 2 can have no descending diagonal, by Corollary 4.13.

5.2 Using the Jigsaws

We saw in the previous section that for many IFS maps we can find the jigsaw pieces easily by considering $w_i(x, y) \leadsto (u, v)$ for some general point (u, v) . In these cases, we saw that we can then find the jigsaw for the map, given that the pieces cover the whole attractor, and that they do not overlap. Now, as we stated in Section 3.3, if we can find some way of applying w_i to only one pixel of each w_i jigsaw piece, and we do this for each i , $1 \leq i \leq N$, we expect that the duplication will be at least reduced, and possibly even completely removed. But how can we do this?

At stage n of the Graphical Algorithm we choose a point $z_n = (x_n, y_n)$ - which by definition is contained in some jigsaw piece J - from the store. Either z_n is the first point of J to be chosen from the store, or, for some $m < n$, $z_m \in J$. Suppose that z_n is the first point of J to be chosen from the store so that applying w_i to z_n will result in a new point being hit in the attractor. If z_n is not the only attractor point contained in J , then, at some future stage of the Algorithm, another point of J will be chosen from the store and this newly hit point will be hit again. Thus, when we choose z_n from the store, we want to *mark* the other points of J in some way - where the mark indicates that applying w_i to this point will not result in a new point, and therefore w_i should not be applied. Then, when we choose a point from the store, checking for the presence of this mark will allow us to predict whether or not a new point will result, and thus save us calculating points which will be duplicates.

Thus we must find a way of recording the marks for each jigsaw piece. Clearly, however, whatever method we use must be able to deal with N different marks since the jigsaws for each of the N maps may be different, although as we shall see later, in cases such as the Sierpinski Gasket where the N jigsaws are the same, we only need to record one mark, which then applies to all N maps. Initially, however, we will record N marks in all cases.

5.2.1 Recording the Marks

Since we need to be able to record marks for each jigsaw piece in each of N jigsaws, it is logical to represent each of the N jigsaws by a Boolean array, where each array is of the same dimensions as the grid. In this way, it is easy to identify a grid point (or attractor point) with the corresponding entry in each array, and consequently it is easy to check whether or not a particular z_n will result in a new point being lit on application of w_i .

Initially, we set all entries of the arrays to false. We will say that z_m lands on a piece if z_m has the lower left coordinates of a grid point of that piece. Then we modify the Graphical Algorithm to include the following checking and updating procedure for each map i .

If z_m lands on a piece, set the array values
corresponding to pixels of that piece to true.
If a later z_n lands on this piece, do not calculate $w_i(z_n)$.

Using this procedure guarantees that w_i will only be applied to one point of each jigsaw piece, and clearly this is true for all i , $1 \leq i \leq N$. In this way, no attractor point will be hit more than once *by the same map* - however, as we noted earlier, some points can be hit by more than one map, and this procedure cannot stop duplication of such points.

Note While it is true that each w_i is only applied to one point of each jigsaw piece, it is not necessarily true that the point to which w_i is applied is the same as that to which w_j is applied, where $i \neq j$.

We will now continue our examples from the previous section, illustrating how we find all of the pixels of a jigsaw piece given one pixel, and showing the results obtained by applying the improvements to the Graphical Algorithm.

5.2.2 Examples

Example 5.7 We saw in Example 5.2 that the jigsaw pieces of the *Sierpinski Gasket* maps are 2×2 squares of pixels, with the top left pixel having both coordinates even. This means that when we choose a point from the store, we must check to see if the coordinates are even or odd, and set the four entries of the array to true

accordingly. If the point chosen from the store is (x, y) then we have the four cases shown in Table 5.3.

x	y	Points to be set to true
even	even	$(x, y), (x - 1, y), (x, y - 1), (x - 1, y - 1)$
even	odd	$(x, y), (x - 1, y), (x, y + 1), (x - 1, y + 1)$
odd	even	$(x, y), (x + 1, y), (x, y - 1), (x + 1, y - 1)$
odd	odd	$(x, y), (x + 1, y), (x, y + 1), (x + 1, y + 1)$

Table 5.3 Points of a jigsaw piece for the Sierpinski Gasket maps.

However, rather than treating each as a separate case, we simply transform x and y to the point of the piece with both coordinates even, and then use the first row of Table 5.3 to set the four points. We use the transformation

$$x \rightarrow x + \text{odd}(x), \quad y \rightarrow y + \text{odd}(y)$$

where $\text{odd}(z)$ is the function that returns 1 if z is odd and 0 if z is even. Applying these improvements to the GA gives the results shown in Table 5.4.

	Original GA	Improved GA
Pts. Calculated	6237	2079
Unique Points	2079	2079
Time (seconds)	17.17	11.62

Table 5.4 Results of improving the GA for the Sierpinski Gasket.

Thus we see that using the jigsaw pieces has indeed improved the performance of the Graphical Algorithm. In this case, we see that the number of points calculated is equal to the number of unique points after improvement - i.e. duplication has been removed completely - and most importantly, the time taken to produce the attractor of the Sierpinski Gasket has been reduced by approximately 5.5 seconds.

We will see shortly that it is possible to improve the performance of the GA still further for the Sierpinski Gasket - but in the meantime, we continue Examples 5.3 and 5.5.

Example 5.8 We know from Example 5.4 that the jigsaw pieces of the *Sierpinski Carpet* are 3×3 squares of pixels, with the centre pixel having both coordinates divisible by 3. As in the previous example, in order to record marks for each pixel of the jigsaw piece, we must first transform x and y to the centre pixel of the piece. However, in this case, we have the added complication that the array references run from 1 to 100, while the values of x and y run from 0 to 99 - and therefore after transforming x and y we add 3 to both coordinates to ensure that no illegal array references are accessed. This means that entries $[i, j]$ of the array, where i or j is equal to 1 or 2, will never be used - however, as we will see shortly, our next improvement step will make this irrelevant. Thus the transformation used is

$$x \rightarrow x + 4 \text{ if } x \equiv 2 \pmod{3}, x \rightarrow x - (x \bmod 3) + 3 \text{ otherwise, and}$$

$$y \rightarrow y + 4 \text{ if } y \equiv 2 \pmod{3}, y \rightarrow y - (y \bmod 3) + 3 \text{ otherwise.}$$

Note We could have simply added 1 to each transformed coordinate to ensure that no illegal array references are accessed - however, the reason for adding 3 will become clear in the next section.

The results of applying these improvements are shown in Table 5.5.

	Original GA	Improved GA
Pts. Calculated	63360	8192
Unique Points	7920	7920
Time (seconds)	247.75	149.93

Table 5.5 Results of improving the GA for the Sierpinski Carpet.

Once again, we see that making use of the jigsaws improves the performance of the Graphical Algorithm. However, as with the Sierpinski Gasket, the time taken to produce the attractor of the Sierpinski Carpet can be improved further by virtue of the fact that all N maps have the same jigsaw - as we will see in Section 5.5. The Peano Curve can be improved in a similar way, with the number of points calculated being reduced from 45000 to 5202 (5000 unique points), and the time taken being reduced from 159.22 seconds to 83.6 seconds.

Note Unlike the Sierpinski Gasket, using the jigsaws to improve the time taken to produce the attractor of the Sierpinski Carpet does not completely remove the duplication of points. In Figure 5.9 we show the *IFS blueprint* - which shows the areas which the whole is sent to under each map - for the Sierpinski Carpet. Points which lie on the boundary lines between maps can clearly be hit by more than one map, and it is these points which can still be duplicated, even after our improvements. Similar situations arise in most Iterated Function Systems, and where they do, it is impossible to completely remove duplication of points by the jigsaws.

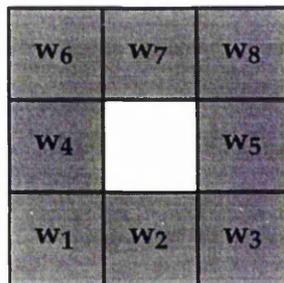


Figure 5.9 The IFS blueprint for the Sierpinski Carpet.

Example 5.9 For the *Flamboyant Crown*, we showed in Example 5.5 that the five maps each have separate jigsaws, which means that we need to consider each map separately when transforming x and y . As in the Sierpinski Carpet, we need to shift x and y so that we do not attempt to access illegal array elements, since x can

take values as low as -32 in this example. Once we have shifted x and y by suitable values, the transformations which we need to perform are as follows.

$$\mathbf{w}_1: x \rightarrow x + 1 \text{ if } x \equiv 3 \pmod{4} \text{ otherwise } x \rightarrow x - (x \pmod{4}), y \rightarrow y + \text{odd}(y)$$

$$\mathbf{w}_2: x \rightarrow x + \text{odd}(x), y \rightarrow y + \text{odd}(y)$$

$$\mathbf{w}_3: x \rightarrow x + (4 - (x \pmod{4})) \text{ if } x \not\equiv 0 \pmod{4} \text{ otherwise } x \rightarrow x,$$

$$y \rightarrow y + 1 \text{ if } y \equiv 3 \pmod{4} \text{ otherwise } y \rightarrow y - (y \pmod{4})$$

$$\mathbf{w}_4: x \rightarrow x + \text{odd}(x), y \rightarrow y - \text{odd}(y)$$

$$\mathbf{w}_5: x \rightarrow x + \text{odd}(x), y \rightarrow y + (4 - (y \pmod{4})) \text{ if } y \not\equiv 0 \pmod{4}$$

$$\text{otherwise } y \rightarrow y.$$

The results of applying these improvements are shown in Table 5.6.

	Original GA	Improved GA
Pts. Calculated	12385	2623
Unique Points	2477	2477
Time (seconds)	36.77	22.33

Table 5.6 Results of improving the GA for the Flamboyant Crown.

As with the Sierpinski Carpet, the duplication has not been completely removed for this IFS, but the performance of the Graphical Algorithm has still been improved considerably. Shortly, we will show how we can cut down on the work required to mark the arrays, but before we do we give one final example of the jigsaws in action.

Example 5.10 We can find all the pixels of the jigsaw pieces of the maps of the *Takagi Function* in much the same way as we did for the Sierpinski Gasket. Recall that the jigsaw pieces for the Takagi Function maps 1 and 2 are as shown in Figure 5.8. Once more, if (x, y) is the point chosen from the store, we transform it to the

unique jigsaw piece pixel with both coordinates even, and then set the four jigsaw piece pixels to true in the array. This time, however, the transformation is slightly more complicated. For map 1, the transformation is given by

$$x \rightarrow x + \text{odd}(x), y \rightarrow y + \text{odd}(y) \text{ if } x \text{ is even, } y \rightarrow y - \text{odd}(y) \text{ if } x \text{ is odd,}$$

while for map 2 the transformation is given by

$$x \rightarrow x + \text{odd}(x), y \rightarrow y + \text{odd}(y) \text{ if } x \text{ is even, } y \rightarrow y + 2 - \text{odd}(y) \text{ if } x \text{ is odd,}$$

and it is easy to check that these transformations do indeed send any point of the jigsaw piece to the unique one with both coordinates even. The results of applying these improvements are shown in Table 5.7.

	Original GA	Improved GA
Pts. Calculated	574	287
Unique Points	287	287
Time (seconds)	1.55	1.25

Table 5.7 Results of improving the GA for the Takagi Function.

Once again, we see that making use of the jigsaw pieces has completely removed the duplication of points, and has also improved the time taken to produce the attractor of the Takagi Function. Clearly, however, the improvement in this case is not as marked as that of our previous examples in this section - this is due to the fact that the Takagi Function maps have more complex jigsaw pieces, and consequently the transformations which shift any point of the piece to the unique point with both coordinates even are more complicated, and take longer to apply.

We saw in this section that it is possible to reduce the amount of point duplication which occurs during the Graphical Algorithm, and consequently reduce the time taken to produce an approximation to the attractor of an IFS, simply by marking jigsaw pieces when one of their points has been chosen as a z_n . However, we also saw that while we now calculate roughly $1/N^{\text{th}}$ of the number of points calculated

in the original algorithm, the time taken to produce the attractor is still considerably more than $1/N$ th of the original time taken. Thus the improvements so far are not as good as we had hoped for - and there is clearly room for further improvement. We now move on to describe the next stage in our improvements.

5.3 Reduced Jigsaws

Recall from the previous section that marking a jigsaw piece involves transforming z_n to a particular point of the piece, and then setting all points of the piece to true. Thus in an IFS map such as map 3 of the Flamboyant Crown we must set 16 points to true every time we encounter a z_n which lies in a previously unmarked piece. But every square in the piece has the same truth value, and so we really only need to mark one point of each jigsaw piece and use this point as a reference for the whole piece. In fact, what we do is to shrink the jigsaw so that each piece is transformed to a single pixel in a smaller array, as illustrated in Figure 5.10.

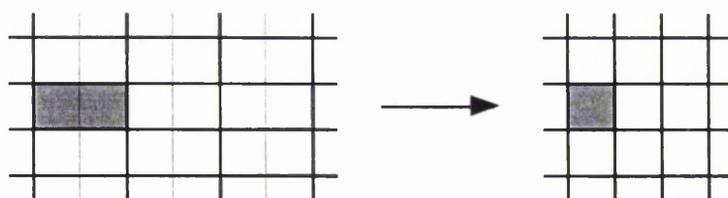


Figure 5.10 An example of jigsaws being shrunk. Each of the rectangles in the jigsaw on the left hand side contains 2 pixels, and is associated with a single pixel on the right hand side. In particular, the shaded rectangle on the LHS is sent to the shaded point on the RHS.

This method can be applied to each of the examples from the previous section, as we now illustrate.

Example 5.11 The bottom left piece of the *Sierpinski Gasket* jigsaw consists of the points $(1, 1)$, $(1, 2)$, $(2, 1)$ and $(2, 2)$. We want to send each of these points to the bottom left entry of a small array - i.e. entry $[1, 1]$. Clearly, in order to shrink a 2×2 piece to a 1×1 pixel, we must divide each coordinate by 2 - which means using the computer *div* operation. Note, however, that $(1 \text{ div } 2) = 0$, while $(2 \text{ div } 2) = 1$, and so simply using $x \text{ div } 2$ does not result in points (x, y) with $1 \leq x \leq 2$ having the same horizontal reference in the reduced jigsaw. In fact we must use the transformation

$$x \rightarrow (x + 1) \text{ div } 2, \quad y \rightarrow (y + 1) \text{ div } 2.$$

Using this transformation on any other jigsaw piece sends the piece to the correct position in the small array as Figure 5.11 shows, where the shaded jigsaw piece on the left hand side is sent to the shaded array entry on the right hand side.

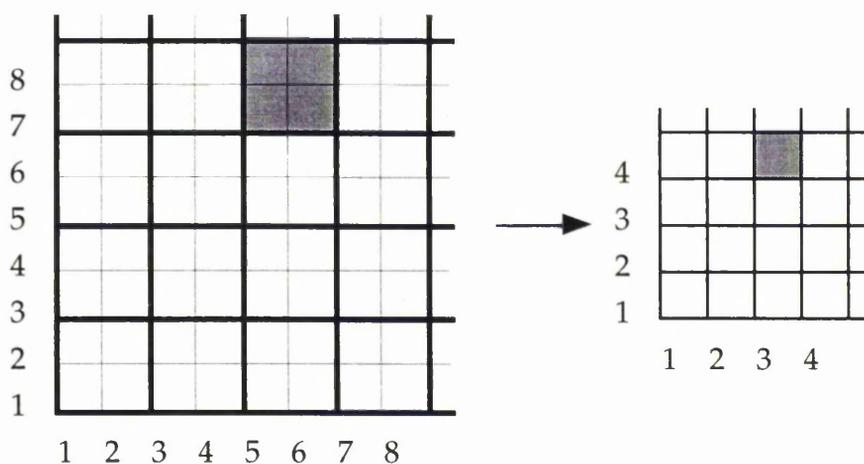


Figure 5.11 The Sierpinski Gasket jigsaw on grid $\{(x, y): 1 \leq x, y \leq 100\}$. Each of the three jigsaws is sent to its reduced companion by $(x, y) \rightarrow ((x + 1) \text{ div } 2, (y + 1) \text{ div } 2)$.

Applying this shrinking procedure means that instead of setting four points in each jigsaw piece hit, we only need to set one - and consequently we will save a small amount of time in the updating of the arrays. Unfortunately, this saving is

partially offset by the need to perform divisions on x and y , and while we do still save some time, the overall saving is relatively small, as we see in Table 5.8.

	Original GA	Improved GA	Reduced Jigsaws
Pts. Calculated	6237	2079	2079
Unique Points	2079	2079	2079
Time (seconds)	17.17	11.62	11.32

Table 5.8 Results of using reduced jigsaws for the Sierpinski Gasket.

Although the results do not show much of an improvement, shrinking the arrays means that we require less storage space - which is important, especially when we are using the Graphical Algorithm on IFS's with a large number of maps. Further, when the jigsaw pieces are larger, as in the Sierpinski Carpet or Peano Curve, the saving will be greater since we avoid a larger number of assignment statements - as the next example shows.

Example 5.12 The jigsaw pieces of the *Sierpinski Carpet*, *Peano Curve* and *Flamboyant Crown* can be shrunk in much the same way as those for the Sierpinski Gasket. Figures 5.12 and 5.13 show sections of the jigsaws for the jigsaws of the Sierpinski Carpet and Peano Curve, and their reduced companions.

Note In both the Sierpinski Carpet and Peano Curve we use $x \rightarrow (x + 4) \text{ div } 3$ to find the horizontal reference in the reduced jigsaw. This is due to the fact that the lower left piece of the jigsaw has -1 as its minimum horizontal coordinate, and this must be sent to 3 before the div operation is applied in order to get minimum reference 1. In general, if the lower left piece of a jigsaw with rectangular pieces has minimum horizontal coordinate M , and the piece has H pixels horizontally, then the reference is found by

$$x \rightarrow (x - M + H) \text{ div } H \quad (5.3)$$

with the vertical reference being found similarly. Note that this explains the given transformation for the vertical coordinate in the Peano Curve.

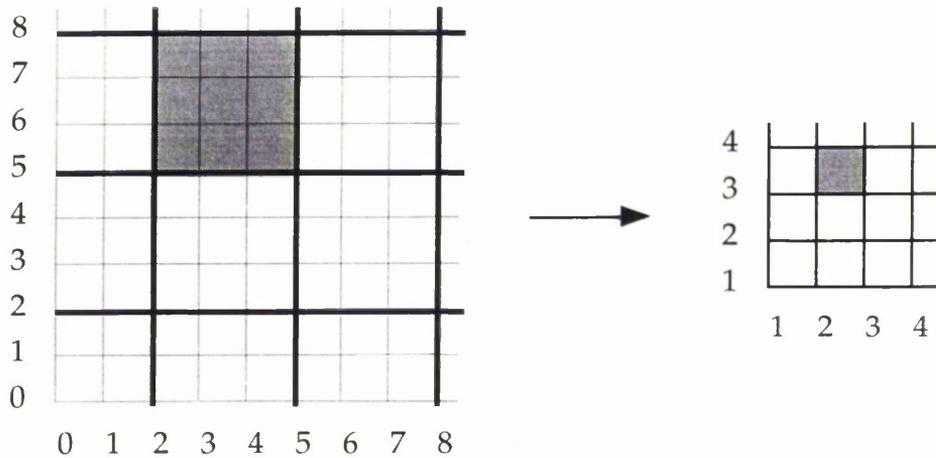


Figure 5.12 The Sierpinski Carpet jigsaw on grid $\{(x, y) : 0 \leq x, y \leq 99\}$. Each of the eight jigsaws is sent to its reduced companion by $(x, y) \rightarrow ((x + 4) \text{ div } 3, (y + 4) \text{ div } 3)$.

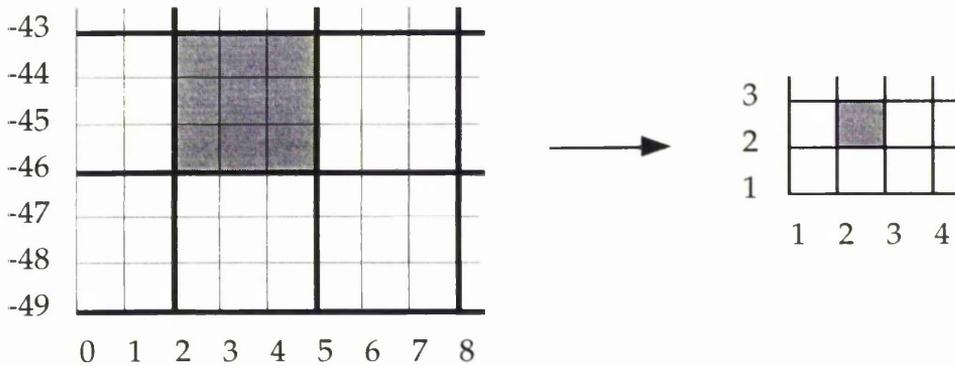


Figure 5.13 The Peano Curve jigsaw on grid $\{(x, y) : 0 \leq x \leq 99, -49 \leq y \leq 49\}$. Each of the nine jigsaws is sent to its reduced companion by $(x, y) \rightarrow ((x + 4) \text{ div } 3, (y + 52) \text{ div } 3)$.

Unlike the Sierpinski Carpet and Peano Curve, each map of the Flamboyant Crown has a distinct jigsaw, and therefore each map requires a different transformation to shrink its jigsaw. Figure 5.14 shows the procedure for map 1 - the other four maps are similar, where in each case, the required transformation is found using (5.3).

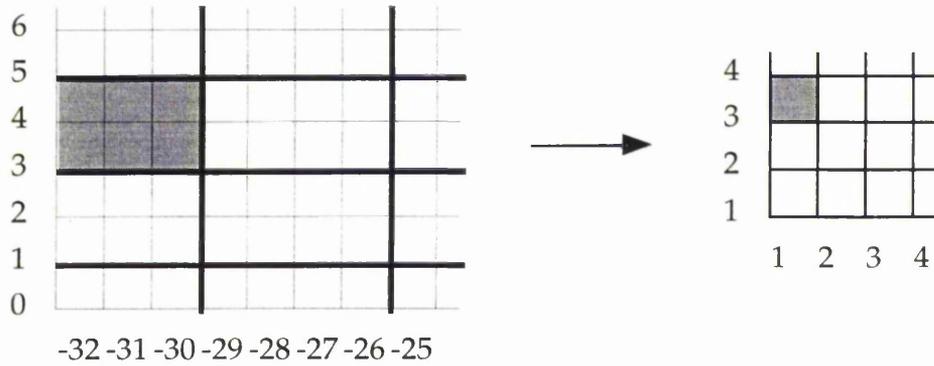


Figure 5.14 The Flamboyant Crown map 1 jigsaw on grid $\{(x, y) : -32 \leq x \leq 32, 0 \leq y \leq 100\}$.

The jigsaw is sent to its reduced companion by $(x, y) \rightarrow ((x + 37) \text{ div } 4, (y + 3) \text{ div } 2)$.

Once we have the transformations to shrink the jigsaws, we can apply them within the Graphical Algorithm in order to further improve its performance. The results of doing this for the three IFS's discussed are given in Table 5.9. Note that the number of points calculated is not altered by shrinking the jigsaws and therefore we have omitted this information from the table.

	Time (seconds)	
	Before Shrinking	After Shrinking
Sierpinski Carpet	149.93	148.88
Peano Curve	83.6	82.9
Flamboyant Crown	22.33	21.55

Table 5.9 Results of using reduced jigsaws for the IFS's of Example 5.11.

Thus we can see that using reduced jigsaws does indeed improve the GA still further, albeit only by a small amount. As expected, the improvement is most striking in the Sierpinski Carpet and Peano Curve, as these have larger jigsaw pieces. We end this section by considering, once more, the Takagi Function, in order to show that more complicated jigsaw pieces can also be reduced.

Example 5.13 (Takagi Function) As with the Sierpinski Gasket, the jigsaw pieces are shrunk horizontally by $(x + 1) \text{ div } 2$, and this is true for both maps. For the vertical reduction, as before we must consider whether x is even or odd. Figures 5.15 and 5.16 show how we reduce the pieces for each map, with the transformation being given below each diagram. Using the reduced jigsaws improves the time taken by 0.3 seconds, taking it from 1.25 seconds down to 0.95 seconds.

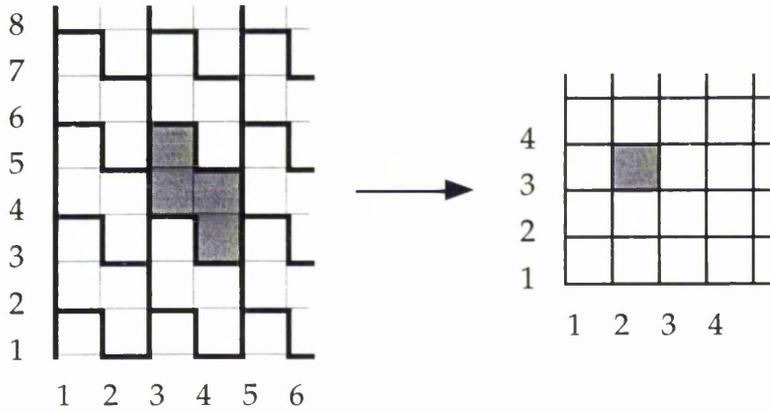


Figure 5.15 The Takagi Function map 1 jigsaw on grid $\{(x, y) : 1 \leq x, y \leq 100\}$. The jigsaw is sent to its reduced companion by $(x, y) \rightarrow ((x + 1) \text{ div } 2, (y + 3 - \text{odd}(x)) \text{ div } 2)$.

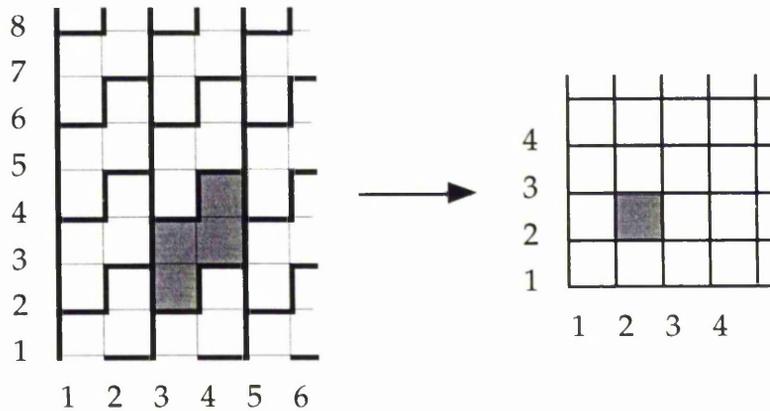


Figure 5.16 The Takagi Function map 2 jigsaw on grid $\{(x, y) : 1 \leq x, y \leq 100\}$. The jigsaw is sent to its reduced companion by $(x, y) \rightarrow ((x + 1) \text{ div } 2, (y + 1 + \text{odd}(x)) \text{ div } 2)$.

Using reduced jigsaws in the Takagi Function has reduced the time taken by the GA from its original value of 1.55 seconds down to 0.95 seconds - a reduction by ratio 1.6 : 1, which is not too far away from our target ratio of 2 : 1. In Section 5.5 we will see that it is possible to reach the target ratio, although not without altering the definition of the Graphical Algorithm, but first we consider once more our 'special cases' - those IFS's whose N jigsaws are identical.

5.4 Using Less Than N Jigsaws

In earlier sections of this chapter, we noted that the Sierpinski Gasket, Sierpinski Carpet and Peano Curve are examples of a special type of IFS, where all N maps have the same jigsaw, and we noted that this property would enable us to make significant further improvements to the performance of the GA. In this section, we introduce the possibility of using less than N arrays to store truth values, and exhibit the effect that this has on the performance of the GA, particularly when the IFS's are of the above type.

Suppose that, in a given IFS, two maps w_i and w_j have identical jigsaws, and recall that a jigsaw piece has its values set to true in the w_i jigsaw (or, more accurately, its reference entry set to true in the reduced w_i jigsaw) if and only if it has been hit by some z_m . Since the w_i and w_j jigsaw are identical, this is equivalent to the same piece having been hit, and having its reference entry set to true in the reduced w_j jigsaw. Therefore, not only are the reduced jigsaws identical, but their corresponding entries have the same truth values - and thus we can use one reduced jigsaw to record truth values for w_i and w_j simultaneously.

Similarly, if a given IFS map has three or more maps whose jigsaws are the same, one reduced jigsaw may be used to record truth values. This is particularly effective if *all* N jigsaws are the same, as for the Sierpinski Gasket and Carpet, and

the Peano Curve. In these cases, a single array contains all necessary truth values - we call this array the *Universal Reduced Jigsaw*. But what effect does using this single reduced jigsaw have on the way the Graphical Algorithm works?

Firstly, it is obvious that whenever z_m lands on a previously unhit jigsaw piece we need only assign one truth value where previously we required N assignments. The resulting saving is especially striking in the case of the Peano Curve, where we assign one value for every $9 \times 9 = 81$ required in the first improvement, and every 9 required in the second improvement. Further, recall that before applying map i to z_m we must check the truth value of z_m in the w_i jigsaw to see if a new point will result. In these special cases, this means that we check the same truth value in the Universal Reduced Jigsaw each time, and consequently the decision on whether or not to apply the map is the same each time, meaning that at each iteration we apply either 0 or N maps. Thus, in these cases, we alter the improved algorithm so that we check the truth value once, at the start of each iteration, and then *apply either all N maps, or no maps*.

However, while both of these operations have some effect on the GA, it is another far less obvious change to the algorithm which makes Universal Reduced Jigsaws so effective.

As we noted above, in cases where all N jigsaws are the same and we use a Universal Reduced Jigsaw, we often choose a z_m from the store only to immediately discard it when we find that it lies in a previously hit jigsaw piece. Thus z_m is not used to find any new points, and therefore adding it to the store originally was needless. Fortunately, we can now avoid adding such points to the store, simply by checking each new point calculated by the algorithm, and only putting in the store those points which lie in previously unhit jigsaw pieces. Since adding new points to the store involves a considerable amount of work to find the

correct position for the new point, avoiding adding needless points cuts down much of the work of the GA. In addition to this, adding fewer points to the store means that the store remains smaller - which has the knock on effect of making it easier to find useful new points. The checking and updating for such cases can be summarised as follows.

Choose z_m from the store. If it lies in a previously hit piece then discard it and choose another, otherwise apply each map to z_m , only adding to the store those $w_i(z_m)$ which lie in previously unhit jigsaw pieces.

Note Although when z_m was added to the store it lay in a previously unhit piece, it is possible that another point of z_m 's jigsaw piece is also in the store. Since this other point may have been chosen as some z_k ($k < m$), by the time z_m is chosen it may lie in a previously hit piece. Thus it is still necessary to check z_m when it is chosen from the store.

We have seen that, in theory, using a single Universal Reduced Jigsaw to store truth values, and altering the way we check and update this array, will lead to considerable improvements in the performance of the GA. Note, also, that while we have concentrated on those IFS's for which Universal Reduced Jigsaws may be used, we should not forget that some further improvement may still be made to those IFS's which have n identical jigsaws ($n < N$), by using one array for these n maps. However, it is only those IFS's with $n = N$ where we can avoid adding superfluous points to the store - since in these cases if duplication occurs in one map, then it occurs in them all.

Table 5.10 shows the results of applying Universal Reduced Jigsaws (URJ's) to the three special cases of our earlier examples. Once again, since these improvements

have no effect on the number of points calculated, we have omitted this information from the table.

	Time (seconds)			Ratio
	Original GA	Reduced Jigs.	URJ's	
Sierpinski Gasket	17.17	11.32	7	2.45
Sierpinski Carpet	247.75	148.88	66.28	3.74
Peano Curve	159.22	82.9	40.22	3.96

Table 5.10 Results of using Universal Reduced Jigsaws (URJ's) on the Sierpinski Gasket and Carpet, and the Peano Curve.

It is clear from the table that using Universal Reduced Jigsaws does indeed lead to notable further improvements in the time taken to produce an approximation to the attractor of an IFS by the Graphical Algorithm, as we had hoped. In particular, it has helped us to get much closer to achieving our aim of reducing the time taken in the ratio $N : 1$, as we see from the final column of Table 5.10.

In Chapter 7 we will introduce a different method of using jigsaws which enables us to get much closer to the target ratio in certain cases. However, as we will see, the method is not applicable to those IFS's whose jigsaw pieces contain relatively few attractor points compared with their individual sizes - as with the Takagi Function, Flamboyant Crown and Sierpinski Gasket. In fact, it is possible to achieve - and indeed better - the $N : 1$ ratio in these cases, and also in many other IFS's, by changing the definition of the Graphical Algorithm slightly. We end this chapter by discussing how to achieve the target ratio, and giving results for our five example IFS's.

5.5 To Sort or Not To Sort

In the course of our earlier discussion on using Universal Reduced Jigsaws, we noted that the process of sorting the store according to the ordering (3.1) takes a significant amount of time and work, particularly when the store is large. However, if points of the store were kept simply in order of arrival - on a 'last in first out' basis - then the original algorithm will be faster, as we no longer have to determine the correct position for a new point being added. Further, since the same points would still be added to the store, albeit in a different order, the only effect that this has on the output of the GA is to change the order in which attractor points are hit. This fact is also noted in [23].

In practice, simplifying the ordering within the store not only speeds up the original GA, but the *jigsaws reduce the times by a greater ratio than before*. In Table 5.11 we give times for the original GA and the GA using jigsaws, both altered so that the store is not sorted, for our five examples. We also give the ratios obtained for both the sorted and unsorted versions of the algorithm, for comparison.

	Time (seconds)		Ratio	
	Original	Improved	Unsorted	Sorted
Sierpinski Gasket	11.46	3.48	3.29	2.45
Sierpinski Carpet	111.27	13.67	8.14	3.74
Peano Curve	80.76	8.75	9.23	3.96
Takagi Function	1.52	0.62	2.45	1.63
Flamboyant Crown	21.45	5.95	3.61	1.71

Table 5.11 Results of removing ordering from the store.

As predicted, the ratio is better than N : 1 in four of our five examples after we remove ordering from the store, and in light of these results we can see that the

improvement achieved by using jigsaws is significantly better than might be inferred by earlier results. However, since our original aim was to improve the performance of the Graphical Algorithm without altering the definition of the algorithm, we will not dwell on these results.

Throughout this chapter, we have concentrated on 'simple' IFS's where the jigsaws are *regular* - that is, they have only one shape and size of jigsaw piece, which is repeated over the whole grid, and indeed over the whole plane. However, in the majority of IFS's, at least one map has a jigsaw which does not satisfy the conditions of regularity. Such jigsaws fall into two distinct categories:

(1) *Semi-regular jigsaws* - where the pieces are of more than one shape or size, but there exists a set of pieces which is repeated over the grid to form the whole jigsaw, and

(2) *Irregular jigsaws* - where the pieces are of more than one shape, and any set of pieces which is repeated is larger than the jigsaw itself i.e. there is no repetition within the jigsaw.

Figure 5.17 gives an example of an irregular jigsaw - that of map 3 of the *Von Koch Curve*, given by $w(x, y) = (x/6 - \sqrt{3}y/6 + 99, \sqrt{3}x/6 + y/6)$. In fact, this is one of the most complicated jigsaws which we shall encounter, and it highlights the extent of irregularity which jigsaws may possess.

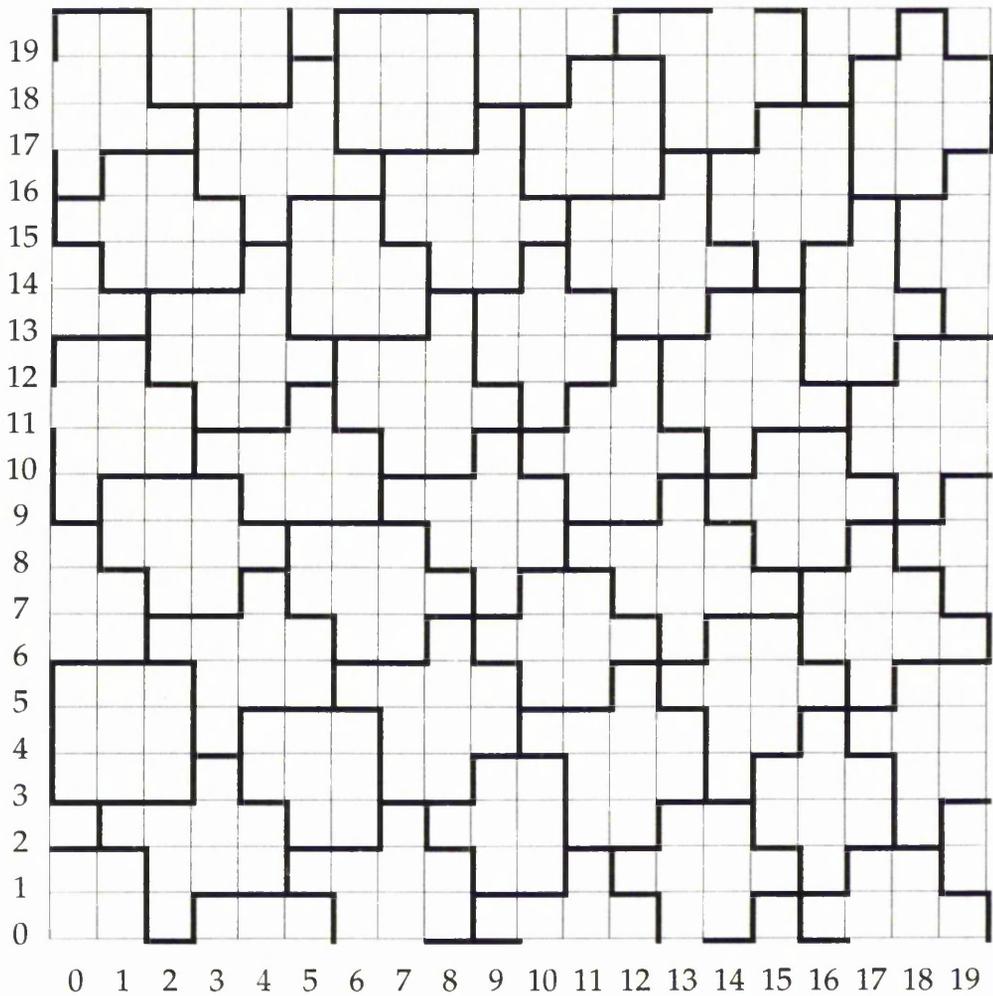


Figure 5.17 An example of an irregular jigsaw - map 3 of the Von Koch Curve.

In later chapters we will consider irregular jigsaws in some detail, and argue that the best way to improve the GA for the majority of IFS's with such jigsaws is to attempt to introduce some regularity into them in such a way that either the attractor is unaltered, or any change to the attractor is 'acceptable'. In the next chapter, however, we will consider semi-regular jigsaws, and we will see that, while the non-regularity of the jigsaws means that improvements cannot be as striking as those presented in this chapter, it is still possible to shorten significantly the run time of the GA for IFS's consisting of such maps.

Figure 5.18 gives an example of a semi-regular jigsaw - that of map 1 of the *Square Plant*, $w(x, y) = (2^x/3, 2^y/3)$ - where the set of shaded pixels is repeated to form the whole jigsaw.

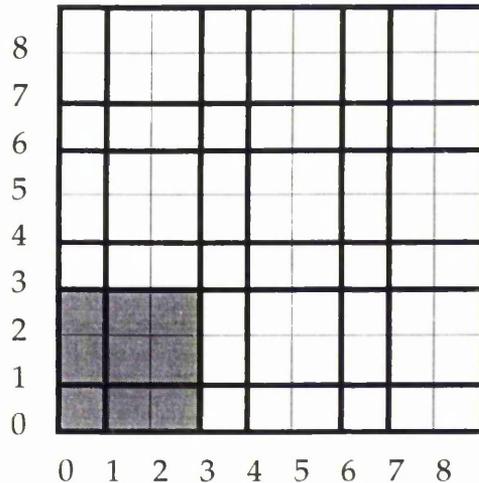


Figure 5.18 An example of a semi-regular jigsaw - map 1 of the Square Plant.

The shaded set of pixels repeats to form the whole jigsaw.

Note 5.14 Throughout this chapter, the IFS maps considered have integral translations only - that is, both of e^* and f^* are integers, where $e^* = e \times \text{scale}$, and $f^* = f \times \text{scale}$. As we have seen, in such cases the values of e^* and f^* have no effect on the appearance of the jigsaw, and are therefore not significant when finding the pieces. However, since the value of scale must apply to all N maps of the IFS, it is often impossible to ensure that every map has an integral translation, while still ensuring that the attractor is produced to the required scale. If two maps w_i and w_j are such that $A_i = A_j$, but $e_i^*, f_i^* \in \mathbb{Z}$ while at least one of $e_j^*, f_j^* \notin \mathbb{Z}$, then the jigsaws for w_i and w_j may be different - and consequently we must consider non-integral translations when we calculate the jigsaw noting, however, that we may ignore the integral part of the value. In future chapters, we will see examples of IFS maps with non-integral translations.

Chapter 6

Maps with Semi-Regular Jigsaws

In the last chapter, we explained how we use jigsaws to improve the Graphical Algorithm, and gave several examples of these improvements, showing that the time could be improved by factors of up to 4. In each of our examples the maps had regular jigsaws, where the pieces were all the same shape and size, and indeed in three of our five examples all N jigsaws of the IFS were identical, meaning that our improvements were maximal. However, as we noted at the end of Chapter 5, many jigsaws arising from IFS maps are not regular, and therefore we must attempt to extend our methods to encompass such non-regular jigsaws. In this chapter we consider IFS's whose maps have *semi-regular* jigsaws. We begin by introducing the concept of *translation symmetries*, which will lead to a formal definition of semi-regularity in Definition 6.6.

6.1 Translation Symmetries

In order for jigsaw pieces which are not necessarily all the same shape to fit together in some regular pattern, there must be some maximal set of pieces which is repeated to form the whole jigsaw. If we can find this maximal set then we can produce the jigsaw, and therefore the only information we require about the jigsaw in order to be able to use it to improve the GA is what the maximal set is. Theorem 6.3 shows how to find such a maximal set, but first we introduce *translation vectors* for a jigsaw.

Definition 6.1 An *isometry* of the plane is a transformation g of the plane which preserves distances. That is, if P and Q are any two points of the plane and d is the metric, then

$$d(g(P), g(Q)) = d(P, Q).$$

Definition 6.2 If we allow a jigsaw to extend over the whole plane, then an isometry of the plane is said to be a *symmetry* of the jigsaw if it maps every piece to another. Most useful of these is a *translation* symmetry, T , given by $T(x, y) = (x + j, y + k)$ for some $j, k \in \mathbf{Z}$. We call such an (j, k) a *translation vector* for the jigsaw. Figure 6.1 shows part of the jigsaw for map 1 of the Takagi Function - it is easy to see from this that any integer multiple of $(2, 0)$ maps every piece of the jigsaw to another, as does any integer multiple of $(0, 2)$, and therefore $(2, 0)$ and $(0, 2)$ are translation vectors for this jigsaw.

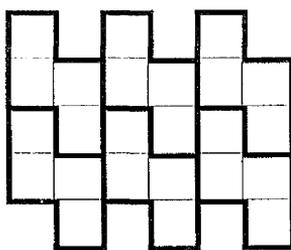


Figure 6.1 Part of a jigsaw which has translation vectors $(2, 0)$ and $(0, 2)$.

Clearly, any integer sum of $(2, 0)$ and $(0, 2)$ will also give a translation vector for the jigsaw of map 1 of the Takagi Function. However, as we will see shortly, when we consider translation vectors we will only be interested in finding the smallest such vectors - in this case $(2, 0)$ and $(0, 2)$ - since the parallelograms formed from these vectors will be used to build the jigsaws. In Theorem 6.3 we give a formula which will enable us to do just this.

Theorem 6.3 Suppose that, for some $j, k, p, q \in \mathbb{Z}$, the matrix coefficients of an IFS map w satisfy

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} j \\ k \end{bmatrix} = \begin{bmatrix} p \\ q \end{bmatrix}. \quad (6.1)$$

Then (j, k) is a translation vector for the jigsaw of w , and

$$\underline{w}(x + j, y + k) = \underline{w}(x, y) + (p, q). \quad (6.2)$$

Proof We have that

$$\begin{aligned} w(x + j, y + k) &= (a(x + j) + b(y + k), c(x + j) + d(y + k)) \\ &= (ax + by, cx + dy) + (aj + bk, cj + dk) \\ &= w(x, y) + (aj + bk, cj + dk) \\ &= w(x, y) + (p, q) \quad \text{by (6.1)}. \end{aligned}$$

But then, since p and q are integers,

$$\underline{w}(x + j, y + k) = \underline{w}(x, y) + (p, q)$$

as required by (6.2). Similarly, for another arbitrary pixel (x', y') ,

$$\underline{w}(x' + j, y' + k) = \underline{w}(x', y') + (p, q).$$

Hence $\underline{w}(x + j, y + k) - \underline{w}(x' + j, y' + k) = \underline{w}(x, y) - \underline{w}(x', y')$, with the vector (p, q) of (6.2) disappearing because of the subtractions. Thus $w(T(x, y)) = w(T(x', y'))$ if and only if $w(x, y) = w(x', y')$ - and so (j, k) is indeed a translation vector for the jigsaw of w , as was to be proved.

Thus, if a jigsaw has a translation vector (j, k) , essentially we need only establish what the pieces are within a band of width $|j, k|$, and then use (6.2) to determine the rest. This is especially useful when the translation vector is parallel to one of the axes - since then either $j = 0$ or $k = 0$, and the width of the band simplifies to $|k|$ or $|j|$. However, whilst finding a single translation vector for a jigsaw will simplify the process of finding the whole jigsaw, in many cases we can do much

better than this - as we will see once we have given the following important definition.

Definition 6.4 Two vectors z_1 and z_2 are said to be *independent* if they have neither the same direction nor the opposite direction - i.e. *they are not parallel*. Whenever z_1 and z_2 are independent, the parallelogram with adjacent edges defined by z_1 and z_2 will have positive area - as illustrated in Figure 6.2.

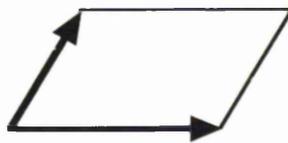


Figure 6.2 The parallelogram formed from two independent vectors.

Now, if we can find another translation vector for the jigsaw which is independent of the first, then the parallelogram with adjacent edges defined by the two vectors will give us all the information we require to form the jigsaw - simply by using (6.2) in the two independent directions. In particular, if we can find integers m , n , p , q , r and s such that

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} m \\ 0 \end{bmatrix} = \begin{bmatrix} p \\ q \end{bmatrix}, \text{ and } \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 0 \\ n \end{bmatrix} = \begin{bmatrix} r \\ s \end{bmatrix} \quad (6.3)$$

then for all integers j , k ,

$$\underline{w}(x + jm, y + kn) = \underline{w}(x, y) + j(p, q) + k(r, s) \quad (6.4)$$

and the parallelogram is a rectangle lying parallel to the axes. In this case, we need only calculate the jigsaw on this m by n rectangle and use (6.4) to find the whole jigsaw. From now on, we will concentrate solely on those maps for which translation vectors parallel to the axes may be found.

Definition 6.5 If a jigsaw has two independent translation vectors - which we take to be vectors of smallest possible magnitude satisfying (6.1) - then we call the parallelogram whose adjacent edges are defined by the vectors a *building block* for the jigsaw, since it can be used to build the jigsaw.

In fact, when we use building blocks to construct the jigsaws, we actually consider only those blocks with edges parallel to the axes as any non-rectangular building block will contain part-pixels. Thus, if the vectors of smallest possible magnitude are not parallel to the axes, we will instead consider the smallest vectors which *are* parallel to the axes as the edges of our building block - i.e. we consider those vectors satisfying (6.3) which have m, n as small as possible. This can be likened to the concept of *centred tilings* in the study of plane tilings, where it is easier to consider rectangles than rhombuses [5].

Definition 6.6 If we can find (smallest possible) integers m and n satisfying (6.3) then we say that the jigsaw is (m, n) -*semi-regular*, and it is formed from the rectangular building block with dimensions $m \times n$. When discussing such jigsaws in general, we will often refer to them simply as *semi-regular*. In theory m and n can take any value - however, as we will see in Section 6.2, the way that we use semi-regular jigsaws to improve the Graphical Algorithm requires us to place restrictions on the size of m and n .

Note Translation vectors provide us with another way of showing that the jigsaw pieces are all the same for an appropriate IFS map. For example, each of the jigsaws of the Sierpinski Gasket has two translation vectors - $(2, 0)$ and $(0, 2)$ - telling us that we need only calculate the pieces on a 2×2 square. But then, calculating one piece shows that it is a 2×2 square - and so all pieces must be the same.

We now consider three examples and show how the building blocks obtained using (6.3) are used to produce the whole jigsaw. The attractors for the IFS's of these examples are shown in Figure 6.3.

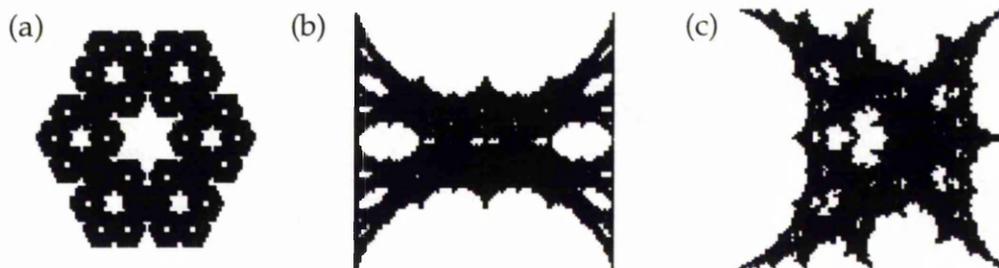


Figure 6.3 The attractors of (a) the Hex, (b) the Sleepy Hollow, and (c) the Crab.

Example 6.7 The *Hex* consists of six maps, given at scale 100 by

$$w_i(x, y) = (0.4x + e_i^*, 0.4y + f_i^*), \quad 1 \leq i \leq 6,$$

where $(e_i^*, f_i^*) = (0, 30), (52, 0), (52, 60), (69.3, 30), (17.3, 0), (17.3, 60)$. Recall from Notes 5.3 and 5.14 that since the first three maps have integral translations, their jigsaws will all be the same as that for $w(x, y) = (0.4x, 0.4y)$. Similarly, the jigsaws for maps 4, 5 and 6 will be the same as that for $(0.4x + 0.3, 0.4y)$. This means that when we use jigsaws to improve the Hex we require only two reduced jigsaws - one for maps 1, 2 and 3, and one for maps 4, 5 and 6. For each of the six maps we have

$$\begin{bmatrix} 0.4 & 0 \\ 0 & 0.4 \end{bmatrix} \begin{bmatrix} 5 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix} \text{ and } \begin{bmatrix} 0.4 & 0 \\ 0 & 0.4 \end{bmatrix} \begin{bmatrix} 0 \\ 5 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$

so the jigsaw is (5, 5)-semi-regular, and we need only calculate the jigsaw on a 5×5 square - for simplicity we use the square $\{(x, y) : 0 \leq x, y \leq 4\}$. Then we have

$$\underline{0.4x} = \begin{cases} 0 & \text{if } x = 0,1 \\ 1 & \text{if } x = 2,3, \\ 2 & \text{if } x = 4 \end{cases}, \quad \underline{0.4x+0.3} = \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{if } x = 1,2 \\ 2 & \text{if } x = 3,4 \end{cases} \quad \text{and} \quad \underline{0.4y} = \begin{cases} 0 & \text{if } y = 0,1 \\ 1 & \text{if } y = 2,3 \\ 2 & \text{if } y = 4 \end{cases}$$

and the 5 x 5 building blocks for the jigsaws are as shown in Figure 6.4.

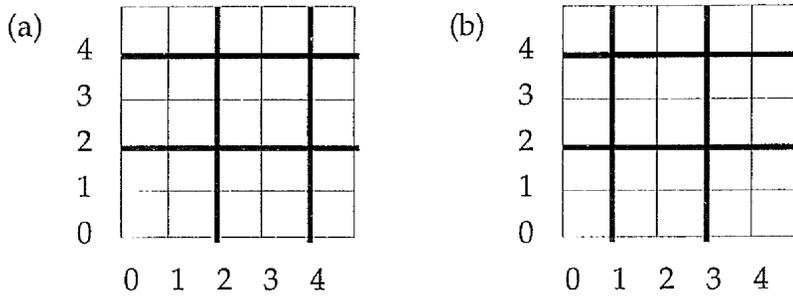


Figure 6.4 Building blocks for the jigsaws of (a) maps 1, 2 and 3, and (b) maps 4, 5 and 6, of the Hex IFS.

Now, (6.4) tells us that $\underline{w}_i(x + 5j, y + 5k) = \underline{w}_i(x, y) + (5j, 5k)$, and therefore the jigsaw for maps 1, 2 and 3 is as shown in Figure 6.5, with that for maps 4, 5 and 6 being found in the same way.

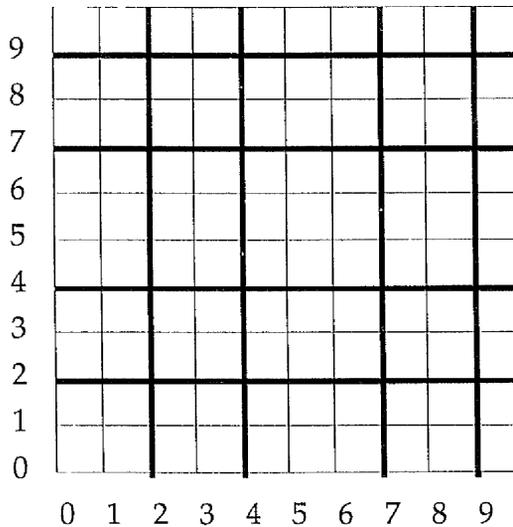


Figure 6.5 Part of the jigsaw for maps 1, 2 and 3 of the Hex.

The jigsaws for the maps of the Hex are still relatively simple, since the pieces are all rectangular - as we could have predicted using Corollary 4.18. However, rectangular building blocks do not necessarily imply that the jigsaw pieces will all be rectangular, as the next example illustrates.

Example 6.8 The *Sleepy Hollow* consists of four maps, given at scale 100 by

$$\begin{aligned}w_1(x, y) &= (0.5x, 0.25x + 0.5y) \\w_2(x, y) &= (0.5x + 50, 0.25x + 0.5y + 25) \\w_3(x, y) &= (0.5x, -0.25x + 0.5y + 50) \\w_4(x, y) &= (0.5x + 50, -0.25x + 0.5y + 25)\end{aligned}$$

Then we have

$$\begin{bmatrix} 0.5 & 0 \\ \pm 0.25 & 0.5 \end{bmatrix} \begin{bmatrix} 4 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ \pm 1 \end{bmatrix} \text{ and } \begin{bmatrix} 0.5 & 0 \\ \pm 0.25 & 0.5 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

and so, for each map of the IFS, $(4, 0)$ and $(0, 2)$ are translation vectors, meaning that the jigsaws are $(4, 2)$ -semi-regular, and we need only calculate the jigsaw on a 4×2 rectangle. The building blocks for the jigsaws are shown in Figure 6.6.

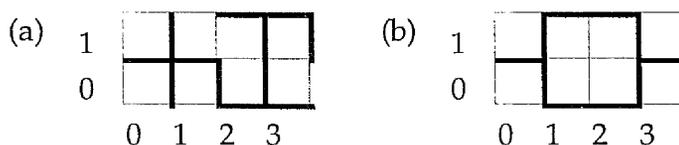


Figure 6.6 The building blocks for the jigsaws of (a) maps 1 and 2, and (b) maps 3 and 4, of the *Sleepy Hollow* IFS.

Now, we can use (6.4) to find the jigsaws, since

$$\begin{aligned}\underline{w}_i(x + 4j, y + 2k) &= \underline{w}_i(x, y) + (2j, j + k) \text{ if } i = 1 \text{ or } 2, \text{ and} \\ \underline{w}_i(x + 4j, y + 2k) &= \underline{w}_i(x, y) + (2j, k - j) \text{ if } i = 3 \text{ or } 4.\end{aligned}$$

The jigsaws for the two maps are shown in Figure 6.7.

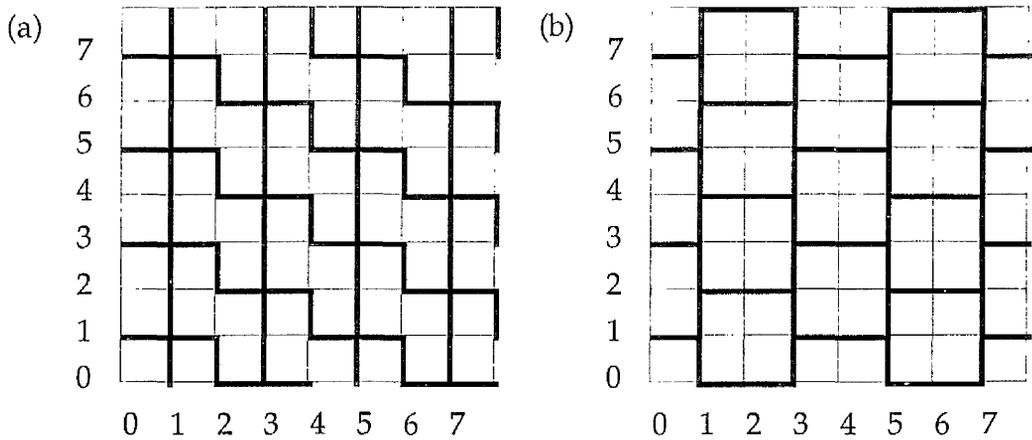


Figure 6.7 Part of the jigsaws for (a) maps 1 and 2, and
(b) maps 3 and 4, of the Sleepy Hollow IFS.

In fact, the jigsaws of Example 6.9 may be considered to be regular since the pieces are all the same shape. However, we classify them as semi-regular due to the fact that, although each piece contains a pixel with both coordinates even, this pixel can lie in more than one position within the piece. For example, the piece containing $(1, 0)$ in the map 3 jigsaw has the pixel with both coordinates even lying in the bottom right corner, while the piece containing $(3, 1)$ has this pixel lying in the top right corner.

Notice that, while the pieces in the jigsaws of maps 1 and 2 are the same shape as those of map 1 of the Takagi Function, they fit together differently and therefore the building blocks are not the same.

Our final example combines the irregularities of the previous two examples - with the jigsaw pieces being neither all rectangular, nor all the same shape.

Example 6.9 The *Crab* consists of four maps, given at scale 100 by

$$\begin{aligned}w_1(x, y) &= (0.5x + 0.2y, 0.5y) \\w_2(x, y) &= (0.5x + 0.2y, -0.5y + 100) \\w_3(x, y) &= (-0.5x + 100, 0.2x + 0.5y) \\w_4(x, y) &= (-0.5x + 100, -0.2x - 0.5y + 100)\end{aligned}$$

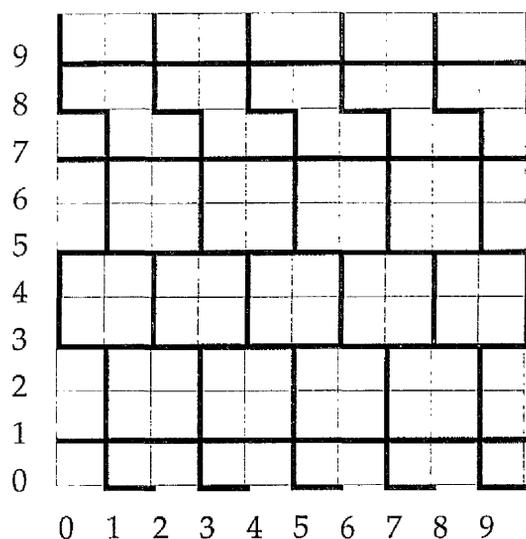
For maps 1 and 2, we find that the jigsaws have translation vectors $(2, 0)$ and $(0, 10)$, so they are $(2, 10)$ -semi-regular, while for maps 3 and 4 the translation vectors are $(10, 0)$ and $(0, 2)$, so the jigsaws are $(10, 2)$ -semi-regular. Further, since

$$\begin{aligned}\begin{bmatrix} 0.5 & 0.2 \\ 0 & \pm 0.5 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \end{bmatrix} &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } \begin{bmatrix} 0.5 & 0.2 \\ 0 & \pm 0.5 \end{bmatrix} \begin{bmatrix} 0 \\ 10 \end{bmatrix} = \begin{bmatrix} 2 \\ \pm 5 \end{bmatrix} \\ \begin{bmatrix} -0.5 & 0 \\ \pm 0.2 & \pm 0.5 \end{bmatrix} \begin{bmatrix} 10 \\ 0 \end{bmatrix} &= \begin{bmatrix} -5 \\ \pm 2 \end{bmatrix} \text{ and } \begin{bmatrix} -0.5 & 0 \\ \pm 0.2 & \pm 0.5 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ \pm 1 \end{bmatrix},\end{aligned}$$

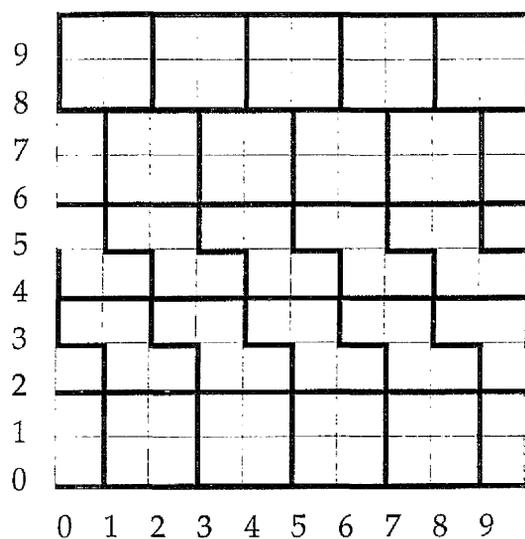
we have

$$\begin{aligned}\underline{w}_1(x + 2j, y + 10k) &= \underline{w}_1(x, y) + (j + 2k, 5k) \\ \underline{w}_2(x + 2j, y + 10k) &= \underline{w}_2(x, y) + (j + 2k, -5k) \\ \underline{w}_3(x + 10j, y + 2k) &= \underline{w}_3(x, y) + (-5j, 2j + k) \\ \underline{w}_4(x + 10j, y + 2k) &= \underline{w}_4(x, y) + (-5j, -2j - k).\end{aligned}$$

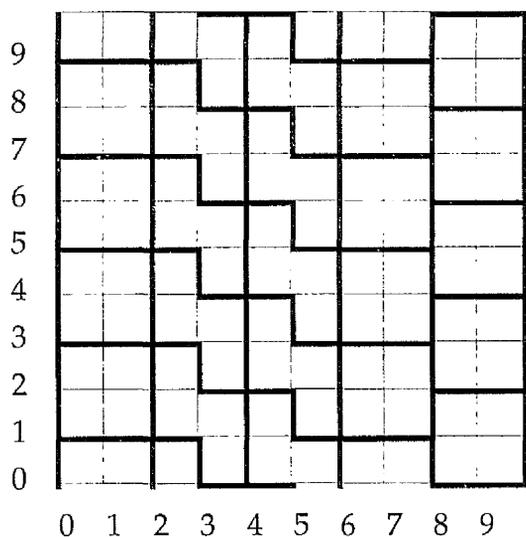
The jigsaws of the four maps, found using the above equations, are shown in Figure 6.8. Note that the jigsaws for maps 1 and 2 are similar, as are those for maps 3 and 4. This is as we would expect since the horizontal coordinates of w_1 and w_2 are equal, while the vertical coordinate of w_2 is simply the negative of that of w_1 plus a translation. A similar situation exists between w_3 and w_4 .



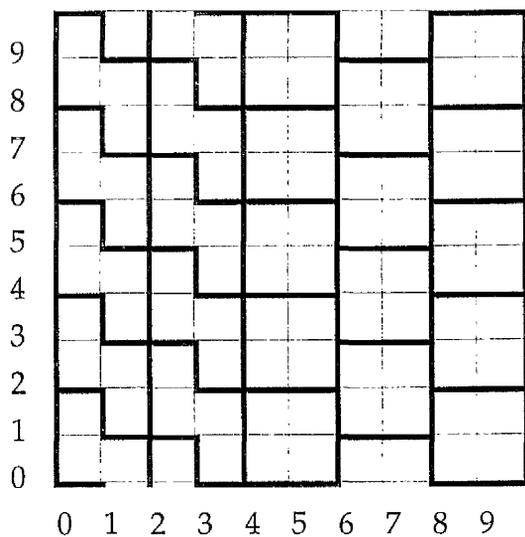
Map 1



Map 2



Map 3



Map 4

Figure 6.8 Part of the jigsaws for the four maps of the Crab IFS.

We have seen in this section that semi-regular jigsaws may be found by finding two independent vectors satisfying (6.1), calculating the jigsaw on the parallelogram defined by the two vectors, and then using (6.2) to produce the whole jigsaw. In particular, when the two vectors lie parallel to the axes, the building block is a rectangle and producing the jigsaw is made much easier. However, as with simple jigsaws, it is what we can do with the jigsaws, rather

than whether or not we can produce them, that is important to the Graphical Algorithm, and thus we must now consider how we can make use of semi-regular jigsaws.

6.2 Using Semi-Regular Jigsaws

In the last section we explained how to produce semi-regular jigsaws from building blocks, using translation symmetries. As in Chapter 5, the next step is to put the jigsaws to use within the Graphical Algorithm and, once more, this requires us to find reduced jigsaws. Since the whole jigsaw is defined by its building block, we need only consider how to reduce the building block in order to find a scheme which will reduce the whole jigsaw. Such a scheme will depend on the size of the building block and on the nature of the pieces within it.

Clearly, reducing an $m \times n$ building block requires us to consider each value of x modulo m , and each value of y modulo n . In the most simple cases, where the pieces are all rectangular, the horizontal and vertical reductions may be performed independently of one another - as with the regular jigsaws of Examples 5.11 and 5.12 - and therefore we must consider a total of $m + n$ combinations of x and y . However, as we have seen, it is common for the jigsaws to have non-rectangular pieces, and in these cases - just as with the Takagi Function of Example 5.13 - the horizontal (vertical) reduction may depend on the value of y (x), resulting in considerably more than $m + n$ combinations of x and y , up to a maximum of $m \times n$. If m and n are large, then finding the reduction scheme in such cases will be extremely complicated, and performing the reduction within the GA will take much longer than desired - even to the extent of exceeding the time taken simply to apply each of the N maps to z_n .

In general, when the code of an IFS is stated, each of its entries is given to, at most, 3 decimal places, meaning that each can be expressed as a rational number with denominator 10^k , $0 \leq k \leq 3$. As a consequence, the largest building block which we may encounter has dimensions 1000×1000 - and in some cases, this will be the only building block which the map has. However, it is clearly not feasible to use such a block to build the jigsaw since, by the above discussion, we would need to consider a minimum of 2000 combinations of x and y in order to find the reduction scheme, and therefore the scheme would be extremely large. In addition to this, a block of dimensions 1000×1000 would generally be much larger than the grid which covers the attractor, and therefore would be much larger than the part of the jigsaw which interests us. We will deal with such cases - where we say that the jigsaw is *irregular* - in Chapters 7 and 8.

In many cases, however, it is possible to find smaller building blocks for the jigsaw - as we will see in Section 6.3. How small the blocks need to be to make the reduction scheme feasible depends very much on how complicated the pieces are within the block, since this dictates how many combinations of x and y we need to consider, and also depends, to a lesser extent, on the scale of the attractor - since for an attractor of size 100×100 , a building block of size 80×80 pixels will not be practical, whilst if the attractor is of size 500×500 pixels, such a block may well be useful. For our purposes, the attractors are normally scaled so that they may be covered by a grid of dimensions 100×100 pixels - and we will usually attempt to find building blocks which have dimensions at most 20×20 pixels, both for the reason of the building block being considerably smaller than the jigsaw, and from the point of view of finding a suitable reduction scheme. In Example 6.17, we will see that such building blocks can still give considerable improvements in the Graphical Algorithm. In the meantime, however, we consider again the small building blocks which we found in the examples of the previous section, after we describe a general reduction scheme for building blocks such as those of the Hex.

Note 6.10 Although we generally consider attractors which have dimensions of at most 100×100 pixels, all of the methods described thus far, and those in the remainder of this thesis, apply to every scale of attractor. However, even when we use a larger scale, it is still preferable to have building blocks which are at most 20×20 pixels since otherwise the reduction schemes may become very complicated.

If, for a particular IFS map, a building block can be found which consists of vertical *bands of pieces*, as in Figure 6.9 where each band may contain a number of different sizes and shape of piece, then a general scheme similar to (5.3) may be used to reduce the jigsaw. Suppose the building block consists of H pixels horizontally, that these pixels are divided into P pieces, and that the lower left piece of the whole jigsaw has minimum horizontal coordinate M . Note that no jigsaw piece may be divided by a bold line in Figure 6.9, but that they may be divided by one of the other lines.

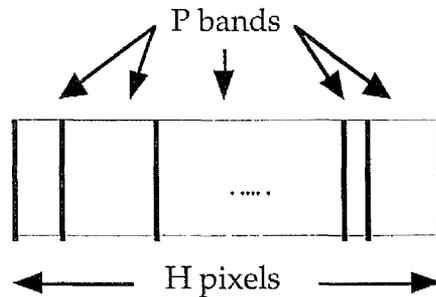


Figure 6.9 A building block divided into P vertical bands.

In such a case, (6.5) below will correctly identify the horizontal coordinates in the reduced jigsaw - although, in some cases, this can be simplified.

$$x \rightarrow P((x - M) \operatorname{div} H) + i \text{ if } x \in i^{\text{th}} \text{ band of block } (1 \leq i \leq P) \quad (6.5)$$

Similarly, if the building block consists of horizontal bands of pieces, then we can find a similar scheme to (6.5) to perform the vertical reduction. Finally, if the building block consists of both horizontal and vertical bands of pieces - which implies that all pieces in the block are whole - then we may use (6.5) and its vertical counterpart to find the overall reduction scheme - as in our first example.

Example 6.11 (Positioning the block) We noted in Example 6.7 that the *Hex* had two distinct jigsaws, one for maps 1, 2 and 3, and another for maps 4, 5 and 6 - each of which contained only rectangular pieces. The building block of Figure 6.4(a) contains partial pieces as well as whole pieces, and so the above theory cannot be applied to this block. However if, instead of taking $0 \leq x, y \leq 4$, we take $-1 \leq x, y \leq 3$, then we get the building block of Figure 6.10, which contains only whole pieces - and therefore (6.5) may be used on this block to reduce the jigsaws of maps 1, 2 and 3.

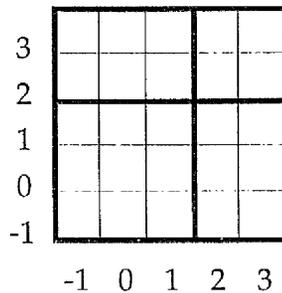


Figure 6.10 An alternative building block for maps 1, 2 and 3 of the *Hex*.

Now, the building block consists of 5 pixels both horizontally and vertically, and these pixels are divided into two pieces in each direction. Thus, in (6.5), we have $H = 5$ and $P = 2$. Further, as we see from Figure 6.10, the lower left piece of the jigsaw has minimum horizontal coordinate -1, so $M = -1$. Then the following scheme sends each piece of the building block, and consequently each piece of the jigsaw, to the correct point in the reduced jigsaw.

$$x \rightarrow \begin{cases} 2((x+1) \operatorname{div} 5) + 2 & \text{if } x \equiv 2, 3 \pmod{5} \\ 2((x+1) \operatorname{div} 5) + 1 & \text{otherwise} \end{cases}$$

$$y \rightarrow \begin{cases} 2((y+1) \operatorname{div} 5) + 2 & \text{if } y \equiv 2, 3 \pmod{5} \\ 2((y+1) \operatorname{div} 5) + 1 & \text{otherwise} \end{cases}$$

For maps 4, 5 and 6, we again use an alternative building block to the one of Figure 6.4 - one which has $-2 \leq x \leq 2$ and $-1 \leq y \leq 3$. As before, $H = 5$ and $P = 2$, but now $M = -2$ for the horizontal coordinate. Thus the reduction scheme uses the same vertical reduction as maps 1, 2 and 3, while the horizontal reduction is given by

$$x \rightarrow \begin{cases} 2((x+2) \operatorname{div} 5) + 2 & \text{if } x \equiv 1, 2 \pmod{5} \\ 2((x+2) \operatorname{div} 5) + 1 & \text{otherwise} \end{cases}$$

Figure 6.11 shows how the scheme for maps 1, 2 and 3 send the building block to part of the reduced jigsaw. This can be easily verified by simple calculation.

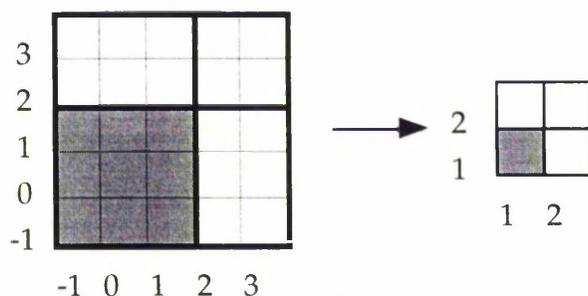


Figure 6.11 How the reduction scheme sends the building block to part of the reduced jigsaw for maps 1, 2 and 3 of the Hex.

As with the simple cases of Chapter 5, we expect that using these schemes to reduce the jigsaws, and then making use of the jigsaws within the GA, will improve its performance. Note that since the jigsaws of maps 1, 2 and 3 are identical, as are those of maps 4, 5 and 6, we use *only two reduced jigsaws to record truth values*. The results given in Table 6.1 confirm that using jigsaws does give an

improvement in the GA, with the time taken to produce the attractor being improved with ratio 1.54 : 1. Clearly, this ratio is much lower than the target ratio of 6 : 1, and indeed is lower than the ratios which we obtained in the simple cases of Chapter 5, but this was to be expected, since calculating the reduced jigsaws is more complicated when the jigsaws are only semi-regular.

	Original GA	Improved GA
Pts. Calculated	41346	7557
Unique Points	6891	6891
Time (seconds)	176.08	113.82

Table 6.1 Results of improving the GA for the Hex.

Our next example is of an IFS whose jigsaws are such that building blocks containing only whole pieces do not exist, meaning that (6.5) cannot be used for both reductions.

Example 6.12 In Example 6.8, we showed that the four maps of the *Sleepy Hollow* each had jigsaws made up from 4×2 rectangular building blocks, and further that the jigsaws for maps 1 and 2 were the same, as were those for maps 3 and 4. Thus, when we use jigsaws to improve the GA, we require only two reduced jigsaws to store truth values.

For the horizontal reduction, we note that the jigsaws divide into vertical bands of width 2, and thus the reduction involves dividing the horizontal coordinate by 2, as for the Sierpinski Gasket (Example 5.11). However, when we come to form the jigsaws, we must take two of these bands together - since the building blocks have horizontal width 4 - and thus how we reduce the vertical coordinate depends on the value of the horizontal coordinate *modulo* 4. With this in mind, the vertical reduction for maps 1 and 2 is similar to that of the Takagi Function, while the

vertical reduction for maps 3 and 4 is similar to the Sierpinski Gasket - with the reduction for the second band differing from that for the first by a shift. We see that the following scheme reduces the jigsaws correctly.

For maps 1 and 2, the vertical coordinate is found by

$$y \rightarrow \begin{cases} (y+3) \text{ div } 2 & \text{if } x \equiv 0,1 \pmod{4} \\ (y+4) \text{ div } 2 & \text{if } x \equiv 2 \pmod{4} \\ (y+2) \text{ div } 2 & \text{if } x \equiv 3 \pmod{4} \end{cases}$$

while for maps 3 and 4, the vertical coordinate is found by

$$y \rightarrow \begin{cases} (y+2) \text{ div } 2 & \text{if } x \equiv 1,2 \pmod{4} \\ (y+3) \text{ div } 2 & \text{if } x \equiv 3,0 \pmod{4} \end{cases}$$

In both cases, the horizontal coordinate is found by $x \rightarrow (x+3) \text{ div } 2$.

Using this scheme to find the reduced jigsaws, and then using the jigsaws in the GA gives the results shown in Table 6.2. For this example, the ratio is 1.47 : 1, which is clearly not as good as the target ratio of 4 : 1, but an improvement has still been made.

	Original GA	Improved GA
Pts. Calculated	17848	5048
Unique Points	4462	4462
Time (seconds)	70.95	48.23

Table 6.2 Results of improving the GA for the Sleepy Hollow.

So far in this section, our examples have been of IFS's whose jigsaws have relatively small building blocks. However, as we saw in Example 6.9, it is possible to have much larger building blocks. For such an IFS map, the reduction scheme will need many more cases - as our final example illustrates.

Example 6.13 In Example 6.9, we showed that the jigsaws for maps 1 and 2 of the *Crab* were made up from 2×10 rectangles, while those for maps 3 and 4 were made up from 10×2 rectangles, and we noted that no two jigsaws were the same, meaning that we require four reduced jigsaws to store truth values. Similar to the previous example, we see from Figure 6.7 that the jigsaw of map 1 divides into horizontal bands of width 2, and thus the vertical reduction involves dividing the vertical coordinate by 2. Then the horizontal reduction depends on the value of the vertical coordinate modulo 10, since the building block is a 2×10 rectangle. The horizontal coordinate in the reduced jigsaw for map 1 is given by

$$x \rightarrow \begin{cases} (x+3) \operatorname{div} 2 & \text{if } y \equiv 0,1,2,5,6,7 \pmod{10} \\ (x+2) \operatorname{div} 2 & \text{if } y \equiv 3,4,9 \pmod{10} \\ (x+4) \operatorname{div} 2 & \text{if } y \equiv 8 \pmod{10} \end{cases}$$

while the vertical coordinate is given by $y \rightarrow (y+3) \operatorname{div} 2$.

The schemes to reduce the jigsaws of the other three maps are similar, recalling that for maps 3 and 4 the jigsaws split into vertical bands of width 2 so that the horizontal reduction involves dividing the horizontal coordinate by 2, while the vertical reduction depends on the value of the horizontal coordinate modulo 10.

The results of reducing the jigsaws, and using the reduced jigsaws in the GA, are given in Table 6.3. Again, an improvement is made, although the ratio is only 1.39 : 1, compared with the target ratio of 4 : 1.

	Original GA	Improved GA
Pts. Calculated	19640	5634
Unique Points	4910	4910
Time (seconds)	86.35	62.12

Table 6.3 Results of improving the GA for the Crab.

Thus, once more, we see that it is possible to improve the performance of the Graphical Algorithm for an IFS whose maps have semi-regular jigsaws, although the improvement is smaller than we could achieve if the jigsaws were regular.

As we noted earlier it is important that we can find the smallest possible building blocks for a jigsaw as it is the dimensions of the building block, coupled with the nature of its pieces, which determine the complexity of the reduction scheme. In the next section, we describe how we find the smallest possible values of m and n satisfying (6.3).

6.3 Finding a Building Block of Least Dimensions

As we noted in Section 6.2, the entries of the IFS code are usually given to, at most, three decimal places, meaning that each can also be expressed as a rational number, with denominator 10^k , $0 \leq k \leq 3$. Of course, in many cases, these rationals may be simplified by cancelling common factors, until they are in lowest terms - and it is these lowest term fractions which provide us with the following method for determining the dimensions of the building blocks simply by looking at the IFS code.

Theorem 6.14 Suppose that the matrix part of the code for an IFS map may be written as

$$A = \begin{bmatrix} a_1/a_2 & b_1/b_2 \\ c_1/c_2 & d_1/d_2 \end{bmatrix},$$

where $a_1, a_2, \dots, d_1, d_2 \in \mathbf{Z}/\{0\}$, and each fraction is expressed in lowest terms. Then there is a corresponding building block with dimensions given by

$$m = \text{lcm}(a_2, c_2) \text{ and } n = \text{lcm}(b_2, d_2). \quad (6.6)$$

and this is the rectangular building block of smallest dimensions for any map with matrix A .

Proof To find the horizontal dimension of the building block, we want to find the smallest value of $m \in \mathbf{Z}$ such that

$$A \begin{bmatrix} m \\ 0 \end{bmatrix} = \begin{bmatrix} p \\ q \end{bmatrix}, \text{ for some } p, q \in \mathbf{Z}.$$

Then

$$\begin{bmatrix} p \\ q \end{bmatrix} = \begin{bmatrix} a_1/a_2 & b_1/b_2 \\ c_1/c_2 & d_1/d_2 \end{bmatrix} \begin{bmatrix} m \\ 0 \end{bmatrix} = m \begin{bmatrix} a_1/a_2 \\ c_1/c_2 \end{bmatrix}$$

and, since $p, q \in \mathbf{Z}$, we need $a_2 \mid ma_1$ and $c_2 \mid mc_1$. But a_1/a_2 is expressed in lowest terms and so a_1 and a_2 have no common factors - i.e. we must have $a_2 \mid m$, and similarly, since c_1 and c_2 have no common factors, $c_2 \mid m$. Thus m is the least integer such that $a_2 \mid m$ and $c_2 \mid m$ - namely $\text{lcm}(a_2, c_2)$. The expression for the vertical dimension is found in a similar way.

Note The statement of Theorem 6.13 requires the entries of A to be non zero - the case where one or more of the entries are equal to zero is, in fact, very simple. For example, if $a_1/a_2 = 0$ then m must be the least integer such that $c_2 \mid m$, i.e. $m = c_2$, while if $a = c = 0$ then the jigsaw pieces are bands of infinite length, and we may simply take $m = 1$. The other cases are similar.

Thus, using Theorem 6.14, we can find the smallest translation vectors simply by looking at the matrix coefficients of the IFS map, and consequently we know what the dimensions of the building block are. We will see shortly that these dimensions can tell us immediately whether or not we can improve the GA for a particular IFS by the methods described thus far, but first we give two examples of Theorem 6.14 in action.

Example 6.15 For map 1 of the *Crab*, we have $a_2 = 2$, $b_2 = 5$, and $d_2 = 2$, while $c = 0$. Then, by Theorem 6.14,

$$n = \text{lcm}(b_2, d_2) = 10$$

while $m = a_2 = 2$ since $c = 0$. Thus the building block is 2×10 , as we saw in Example 6.9.

Example 6.16 Map 2 of *Barnsley's Frizzy* has matrix A given by

$$A = \begin{bmatrix} 0.26 & -0.5 \\ 0.23 & 0.57 \end{bmatrix}$$

so that $a_2 = 50$, $b_2 = 2$, $c_2 = 100$ and $d_2 = 100$. Thus, by Theorem 6.14, $m = n = 100$ and the building block has dimensions 100×100 . The attractor is shown in Figure 6.12(a).

This last example highlights a case where the dimensions of the building block are too large to be of any practical use to us in terms of improving the Graphical Algorithm. The dimensions of the building block are 100×100 which, as we saw earlier, means that the reduction scheme will have a minimum of 200 options - and in fact, since the pieces are not all rectangular, there will be more than 200 options in the scheme - which is not feasible to program. Further, when the attractor of the Frizzy is produced by the GA at scale 70 we find that every point $(x, y) \in \mathcal{A}$ lies within the grid $\{(x, y) : 43 \leq x \leq 138, 2 \leq y \leq 75\}$, so that the dimensions of the grid are 96×74 - smaller than the dimensions of the building

block. Consequently, the building block is of no practical use, and the methods which we have described so far will not help us to improve the GA for this IFS.

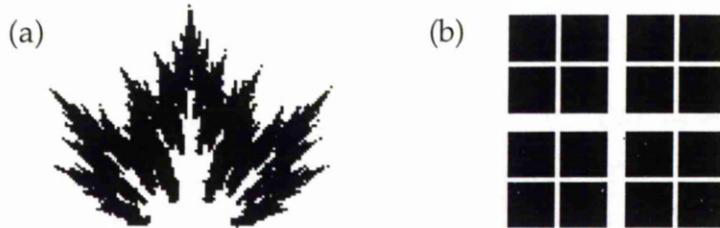


Figure 6.12 The attractors of (a) Barnsley's Frizzy, and (b) the Alt. Square.

Our final example is of an IFS whose jigsaw building blocks are much larger than any considered before, but where the general formula (6.5) is applicable, enabling us to find the reduction scheme easily, and still improve the Graphical Algorithm.

Example 6.17 The *Alt. Square* consists of four maps, each given at scale 100 by

$$w_i(x, y) = (0.45x, 0.45y) + (e_i^*, f_i^*), \quad 1 \leq i \leq 4,$$

where $(e_i^*, f_i^*) = (0, 0), (0, 50), (50, 0), (50, 50)$. The attractor is shown in Figure 6.12(b). Since each of the translations are integral, the four maps have the same jigsaw (see Notes 5.3 and 5.14), and so we may use one reduced jigsaw to record truth values. Further, since $0.45 = 9/20$, Theorem 6.14 says that $m = n = 20$, and the building blocks are of dimensions 20×20 . In fact, although the building blocks are large, the jigsaw pieces are all rectangular by Corollary 4.17. However, the pieces are not all the same size - as we see in Figure 6.13.

Notice that, since this building block is made up of both horizontal and vertical bands, we may use (6.5) and its vertical counterpart to obtain the reduction scheme.

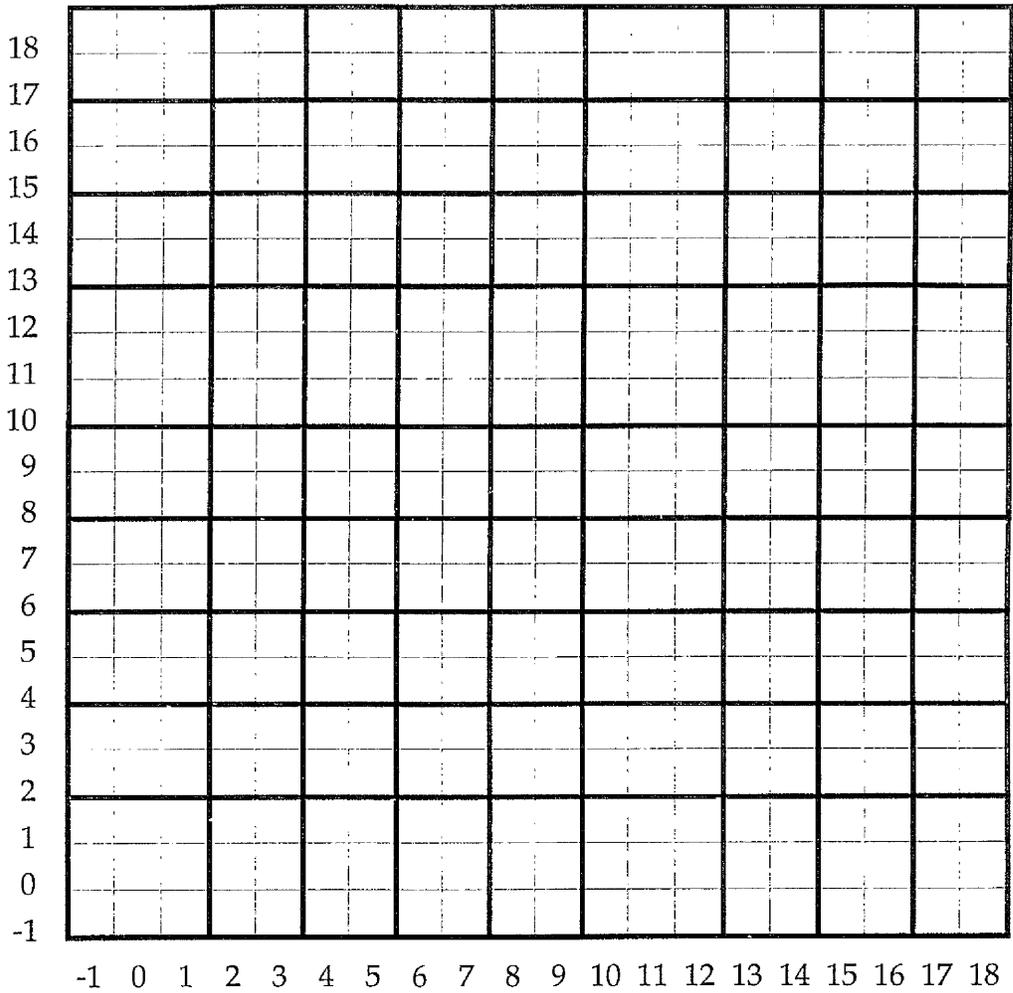


Figure 6.13 The building block for the maps of the Alt. Square.

The horizontal reduction scheme is given by

$$x \rightarrow \begin{cases} 9((x+1) \operatorname{div} 20) + 1 & \text{if } x \equiv -1, 0, 1 \pmod{20} \\ 9((x+1) \operatorname{div} 20) + 2 & \text{if } x \equiv 2, 3 \pmod{20} \\ 9((x+1) \operatorname{div} 20) + 3 & \text{if } x \equiv 4, 5 \pmod{20} \\ 9((x+1) \operatorname{div} 20) + 4 & \text{if } x \equiv 6, 7 \pmod{20} \\ 9((x+1) \operatorname{div} 20) + 5 & \text{if } x \equiv 8, 9 \pmod{20} \\ 9((x+1) \operatorname{div} 20) + 6 & \text{if } x \equiv 10, 11, 12 \pmod{20} \\ 9((x+1) \operatorname{div} 20) + 7 & \text{if } x \equiv 13, 14 \pmod{20} \\ 9((x+1) \operatorname{div} 20) + 8 & \text{if } x \equiv 15, 16 \pmod{20} \\ 9((x+1) \operatorname{div} 20) + 9 & \text{if } x \equiv 17, 18 \pmod{20} \end{cases}$$

and similarly for the vertical reduction scheme. The results of reducing the jigsaws, and then applying them to the GA are given in Table 6.4.

	Original GA	Improved GA
Pts. Calculated	20736	5184
Unique Points	5184	5184
Time (seconds)	73.7	48.1

Table 6.4 Results of improving the GA for the Alt. Square.

Thus we see that, even when the building blocks are as large as 20×20 , it is still possible to achieve a considerable reduction in the time taken to produce the attractor by the Graphical Algorithm. In this instance, of course, the reduction is aided by the fact that we require only one reduced jigsaw, and therefore, as in Section 5.4, we see a much better reduction than we would if N reduced jigsaws were required.

Note Cases (2) - (5) of the above scheme may be rewritten as

$$9((x + 1) \text{ div } 20) + ((x \text{ mod } 20) \text{ div } 2) + 1,$$

while cases (7) - (9) may be rewritten as

$$9((x + 1) \text{ div } 20) + (x \text{ mod } 20 + 1) \text{ div } 2.$$

Doing this enables us to reduce the number of cases in the reduction scheme from 9 to 4. However, altering the reduction scheme in this way within the GA program increases the time taken to produce the attractor from 48.1 seconds (as given in Table 6.4) to 52.3 seconds due to the fact that we are adding further *div* operations to the scheme.

In a case such as Example 6.17, where the building block is large, but we have $ab = cd = 0$ so that the jigsaw pieces are all rectangular, we can actually get some idea of how well the improvements will work by considering the numerators of the non-zero entries of the matrix.

Suppose that we have a map with $b = c = 0$ and $a, d \neq 0$ so that the building block has dimensions $a_2 \times d_2$. Then we can find a building block which consists only of whole pieces, and consequently we can use (6.5) and its vertical counterpart to find the reduction scheme. In this scheme, we find that the a_2 horizontal values divide into a_1 bands, since $\underline{w}_h(x + a_2, y) = \underline{w}_h(x, y) + a_1$ so that the a_2 horizontal values are sent to $\underline{w}_h(x, y) + r$ where $0 \leq r \leq a_1 - 1$. Similarly, the d_2 vertical values divide into d_1 bands, and so we can gain some insight into the sizes of the jigsaw pieces simply by knowing the values of a_1 and d_1 .

When a_1 is small compared to a_2 , dividing the a_2 horizontal values into a_1 bands will mean that the bands are all of reasonable width. However, if a_1 is close to a_2 then some of the bands will be narrow, and may even have width of just one pixel. If this is also the case with the vertical values then many of the jigsaw pieces may be single pixels, while others may consist of only two pixels - and such jigsaws will lead to only minimal improvements.

Thus when we have building blocks which consist of only whole pieces, all of which are rectangular, it is desirable to have a_1 small compared to a_2 , and d_1 small compared to d_2 , so that the pieces will be of reasonable size, and improvements will be considerable. As we have seen, we are able to improve the GA for the Alt. Square considerably - due to the fact that $a_1 = d_1 = 9$, which is small compared to $a_2 = d_2 = 20$.

Note If we have a case where $a, d \neq 0, b = c = 0$, and $a_1 = d_1 = 1$ then, by the above, the a_2 horizontal values will divide into one band, as will the d_2 vertical values. Thus the building block will consist of just *one jigsaw piece*, and consequently the jigsaw will be made up of pieces which are all the same size and shape - as we saw in the first three examples of Chapter 5.

We have seen in the course of this chapter that it is possible to make reasonable improvements to the GA for IFS's whose maps have semi-regular jigsaws, even when the building blocks for the jigsaw have dimensions 20×20 . Unfortunately, however, it is often the case that the building block is too large to be of any use in improving the GA - as we noted at the beginning of Section 6.2. In these cases, we say that the jigsaw is *irregular* - such jigsaws are the subject of the remaining chapters of this thesis.

We begin in Chapter 7 by considering two ways of finding the pieces of irregular jigsaws by scanning around a particular starting point. As we will see, only one of these methods can ever be successful and, further, improvements can only be achieved in cases where the majority of pieces contain a relatively large number of attractor points. Since this property holds for only a small number of IFS's, another method must be found - this is the topic of Chapter 8, where we will show that many maps with irregular jigsaws can be approximated so that their jigsaws become semi-regular, and argue that this is the best way by which we can improve the GA for such maps.

Chapter 7

Irregular Jigsaws

In the last two chapters, we introduced methods for improving the Graphical Algorithm for IFS's consisting of maps whose jigsaws are either *regular*, where all jigsaw pieces are congruent by a translation, or *semi-regular*, where the jigsaw is constructed from $m \times n$ building blocks which themselves are made up from a number of jigsaw pieces. These methods can be used to improve the time taken to produce the attractors of twenty-nine of our fifty IFS's, with the results obtained being shown in the Appendix. In addition to these, there are a further nine IFS's which have at least one map with either a regular or semi-regular jigsaw - applying our methods to such maps, whilst leaving the others alone, allows us to partially improve the performance of the GA for these IFS's.

However, as we noted in Chapter 6, there are many maps for which the only building block is too large to be of any use to us - often as large as 1000×1000 pixels - so we must consider different methods of improving the GA. In this chapter we will look at various possibilities and argue that, in most cases, the best hope of improvement lies in altering the maps of such IFS's - whilst not altering the appearance of the attractor beyond some acceptable level - so that we can apply the methods of previous chapters. We begin with an example which, as we will see in Chapter 8, is reasonably easy to deal with.

Example 7.1 The *4-Fold Crystal* consists of 4 maps, given at scale 100 by

$$w_1(x, y) = (0.255x + 37.3, 0.255y + 67.1)$$

$$w_2(x, y) = (0.255x + 11.5, 0.255y + 22.3)$$

$$w_3(x, y) = (0.255x + 63.1, 0.255y + 22.3)$$

$$w_4(x, y) = (0.37x - 0.642y + 63.6, 0.642x + 0.37y - 0.6)$$

For each of the first three maps we have $255/1000 = 51/200$ in lowest terms and therefore, by Theorem 6.14, the smallest building block has dimensions 200×200 - which is clearly too large for us to deal with by previous methods. Note, however, that since we have $b = c = 0$, Corollary 4.18 tells us that all jigsaw pieces are rectangular, as we see in Figure 7.1 for map 1, with maps 2 and 3 having similar jigsaws (translations of that for map 1).

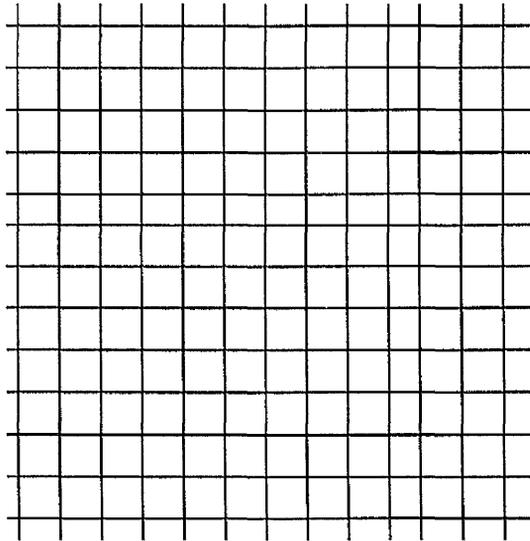


Figure 7.1 Part of the jigsaw for map 1 of the *4-Fold Crystal*.

Note that the lines in the above jigsaw are not equally spaced; the large square pieces consist of 16 pixels, while the smaller squares consist of 9 pixels (and so the rectangular pieces consist of 12 pixels, arranged in either a 3×4 or 4×3 rectangle).

For map 4, we find that the smallest building block has dimensions 500×500 - which again is too large for us to deal with by previous methods. Note that, although we do not have $ab = cd = 0$ in this case, we do have $|a - b| \geq 1$ and $|c + d| \geq 1$ so that Corollary 4.13 rules out both types of diagonal, and the jigsaw pieces are flat. Part of the jigsaw of this map is shown in Figure 7.2 - in this jigsaw the grid squares are single pixels, and so the rectangular pieces consist of just two pixels (note that this does not necessarily mean that the jigsaw is uncomplicated).

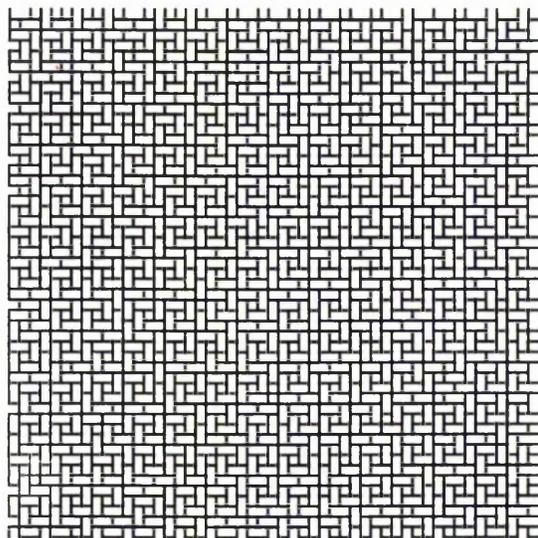


Figure 7.2 Part of the jigsaw for map 4 of the 4-Fold Crystal.

Careful consideration of the jigsaw of Figure 7.2 shows that, contrary to first appearances, there is no small block of jigsaw pieces which is repeated to form the whole jigsaw - just as we expect, since the smallest building block has dimensions 500×500 .

The maps of the 4-Fold Crystal highlight the fact that irregular jigsaws need not have complicated pieces - however, as we saw with the jigsaw of map 3 of the Von Koch Curve in Figure 5.17, many do. Figure 7.3 shows two other irregular jigsaws which have more complicated pieces.

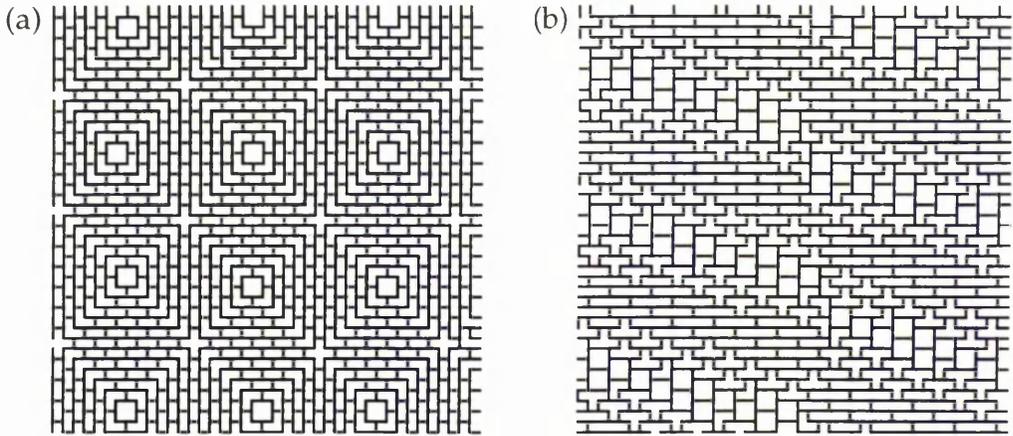


Figure 7.3 Part of the jigsaws of (a) Face map 1, and (b) Barnsley's Frizzy map 3.

Remark Jigsaw (a) of Figure 7.3 has a building block of dimensions 50×50 as it's smallest, while the smallest building block for (b) has dimensions 100×100 . Further, the smallest building block for both map 3 and map 4 of the Von Koch Curve has dimensions 1000×1000 - and so none of these maps can be dealt with by previous methods.

Since we cannot deal with maps which have irregular jigsaws by previously discussed methods, we must consider alternative ways of handling such maps. In Sections 7.1 and 7.2 we look at two different methods, both of which involve finding every pixel of the jigsaw piece from a given starting pixel.

7.1 Scanning Around the Starting Point

Recall that, before we introduced reduced jigsaws, our method for dealing with regular jigsaws was to set each pixel of the piece in turn to true - which was a simple process as we could immediately exhibit the whole jigsaw piece from any given starting position. However, in the irregular case, we cannot do this - instead, we calculate $w(x', y')$ for points (x', y') surrounding $z_j = (x, y)$, setting the corresponding jigsaw entry to true whenever (x', y') is in the same piece as z_j and

then repeating the procedure for each such (x', y') . In fact, we do this recursively, starting from (x, y) and repeating the process with (x', y') as the new starting point every time we find a point (x', y') in the same piece as (x, y) - as the following pseudo code illustrates, where $J(x, y)$ is the jigsaw position of (x, y) .

```

procedure FindAllPoints (x, y: integer)
  set J(x, y) to true
  for each pair (i, j) with  $-1 \leq i, j \leq 1$  except  $i = j = 0$  do
    if  $J(x + i, y + j)$  is false and  $w(x + i, y + j) = w(x, y)$  then
      FindAllPoints (x + i, y + j).
  
```

Notice that, because we set $J(x, y)$ to true as soon as the procedure starts, and check whether or not $J(x + i, y + j)$ is set before moving to a different layer of recursion, moving through the layers results in FindAllPoints being called at most once on any point of the jigsaw. Example 7.2 illustrates how this recursive procedure works by considering an imaginary jigsaw piece for some map w .

Note In practice we check points round the starting point in a clockwise fashion, beginning with $(x + 1, y)$.

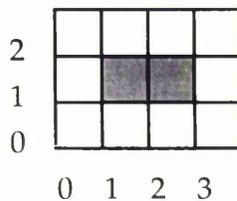


Figure 7.4 A simple jigsaw piece.

Example 7.2 Consider the simple jigsaw piece shown in Figure 7.4, and suppose that $(1, 1)$ is the first point of this piece to be chosen from the store. A call of FindAllPoints $(1, 1)$ sees us calculate $w(2, 1)$ first - and, of course, we find that

$w(2, 1) = w(1, 1)$, causing us to call `FindAllPoints (2, 1)`. This call results in calculation of $w(x', y')$ for $1 \leq x' \leq 3, 0 \leq y' \leq 2$ (but not $x' = 2, y' = 1$), with only $(1, 1)$ being in the same piece. However, we have already set $J(1, 1)$ to true and so we do not call `FindAllPoints (1, 1)` here. Once each of these points has been calculated, we move back to the top layer of the recursion and continue with `FindAllPoints (1, 1)` - resulting in calculation of $w(x', y')$ for a further 7 points. Thus we require a total of 15 calculations to find the 2 points of this jigsaw piece.

Example 7.2 highlights a possible problem with this method, as we require a large number of calculations to find a small number of points. Note, however, that this is a worst case scenario. As the Graphical Algorithm runs, more and more points are chosen from the store and more and more pieces will have their points set in the jigsaw. This means that many of the points surrounding the piece we are currently trying to find will have value true - and we do not need to calculate w for such points. In order to see whether or not this method is successful in practice, we consider the *Tree 1* IFS.

Note At any stage of the recursive process, we need only consider the eight points surrounding the starting point as we know that jigsaw pieces do not contain holes. Thus if the pixel next to the starting pixel is not in the piece then neither is any other pixel in that direction and we can move on to consider another direction. However, as we saw in Section 4.4 some maps have jigsaw pieces which are disconnected (i.e. they contain gaps). In such cases, each component of the piece will be treated as a separate jigsaw piece, meaning that the resultant point will often be duplicated - however no error occurs in the attractor.

Example 7.3 The *Tree 1* IFS consists of 5 maps, given at scale 100 by

$$w_1(x, y) = (0.195x - 0.488y + 44.3, 0.344x + 0.443y + 24.5)$$

$$w_2(x, y) = (0.462x + 0.414y + 25.1, -0.252x + 0.361y + 56.9)$$

$$w_3(x, y) = (-0.058x - 0.07y + 59.8, 0.453x - 0.111y + 9.7)$$

$$w_4(x, y) = (-0.035x + 0.07y + 48.8, -0.469x - 0.022y + 50.7)$$

$$w_5(x, y) = (-0.637x + 85.6, 0.501y + 25.1)$$



Figure 7.5 The attractor of the Tree 1 IFS.

It is reasonably easy to see that none of these maps has a semi-regular jigsaw, and it is also true that the jigsaw pieces are, in general, not simple (although Corollary 4.18 tells us that the jigsaw pieces for map 5 will all be rectangular, as $b = c = 0$). Further, no two of the jigsaws are the same and so we must use five distinct jigsaws in our method. Table 7.1 shows the results for both the original Graphical Algorithm and for our method.

	Original GA	GA using Jigsaws
Pts. Calculated	9095	2146
Unique Points	1819	1819
Time (seconds)	25.17	74.87

Table 7.1 Results for the Tree 1 IFS, with and without jigsaws.

We see from the table that, while the number of points calculated has been reduced to the point where it is almost equal to the number of unique points, the run time has increased due to the time taken to find the jigsaw pieces - and therefore this method is unsatisfactory in its present form. Fortunately, one of the properties of jigsaw pieces given in Chapter 4 allows us to improve the method for the majority of IFS maps, as we now show.

In the FindAllPoints procedure defined earlier, for a given starting point (x, y) , we examine each of the eight points surrounding (x, y) , moving a layer deeper into the recursion each time we find a point which is in the same jigsaw piece as (x, y) . However, as we showed in Section 4.3, the jigsaw pieces of the majority of IFS maps contain no bridges and therefore the pieces are edge connected (strong version of Definition 4.21(iii)) - which means that we need only consider points which are *edge* adjacent to (x, y) . Thus we have reduced the number of points to be considered from eight to four, and Table 7.2 shows that doing so improves the performance of this method by nearly 27 seconds. Once again, we reproduce previous results for comparison.

	Original GA	8 Points	4 Points
Pts. Calculated	9095	2146	2146
Unique Points	1819	1819	1819
Time (seconds)	25.17	74.87	48.18

Table 7.2 Results of examining 8 and 4 points around (x, y) within FindAllPoints.

Although reducing the number of checks involved in the FindAllPoints procedure improves its performance considerably, we see from Table 7.2 that it still takes almost twice as long to produce the attractor by this method as it did by the original GA - and therefore it would seem that this method is unsatisfactory.

Before discussing why this is the case, we consider a second example - that of the Von Koch Curve.

Example 7.4 For the *Von Koch Curve*, we use this method to find the pieces of the jigsaws of maps 3 and 4 noting that, since the two jigsaws are identical, only one array is required to store whether or not a piece has been hit. Further, maps 1 and 2 have regular jigsaws, and so we may use the methods described in Chapter 5 to deal with them. Table 7.3 gives the results of using jigsaws for all four maps, with the results for the original GA given for comparison.

	Original GA	GA using Jigsaws
Pts. Calculated	1616	406
Unique Points	404	404
Time (seconds)	4.25	3.98

Table 7.3 Results for the Von Koch Curve, with and without jigsaws.

It would appear initially that this method has enabled us to improve the performance of the GA for this IFS. However, if we use jigsaws for maps 1 and 2 of the Von Koch Curve, whilst applying maps 3 and 4 as we would in the original GA, the run time is roughly 3 seconds - and therefore using jigsaws for maps 3 and 4 actually increases the run time.

In fact, using the FindAllPoints procedure will *always* slow down the GA - to see why this is the case, we must consider how many points are examined in the course of the procedure.

Example 7.5 We consider again the jigsaw piece shown in Figure 7.4, and look at how many calculations we require in order to find the piece (recall that we required 15 calculations when we checked all eight points surrounding the

starting point). We assume that none of the pieces surrounding this piece have been hit, and therefore none of the pixels around the piece have value true in the jigsaw. Figure 7.6 shows the jigsaw piece again, with numbers showing the order in which points are checked.

2		7	4	
1	6	0	1	2
0		5	3	
	0	1	2	3

Figure 7.6 A simple jigsaw piece with numbers showing the order in which points are checked.

Thus, even using the improved method, we still require a total of 8 calculations - which includes calculation of $w(x, y)$ for the starting point (x, y) - to find the two pixels of the jigsaw piece. Of course, this is the worst case since as the GA runs many of the pixels numbered between 2 and 7 above will already have been set and will therefore not be checked when FindAllPoints is called for this piece. However, even in the best possible case, when all pixels surrounding the piece have already been set, we still need to calculate 2 points to find the piece - which is no better than simply applying w to the two pixels of the piece. In fact, calculating w for two pixels may actually be more calculation than we would do in the original GA, as only those pixels which are in the attractor will ever have w applied to them in the original GA - and there is a chance that only one of the pixels of this piece is in the attractor.

Remark Suppose that, instead of using two Boolean values, we use "three valued logic" - that is, we use integer values -1, 0 and 1, where 1 and 0 represent false and true respectively. Then whenever a point is checked, and is not in the jigsaw piece, we set it to -1. Since we check only those points whose values are 0, we can now

avoid checking points more than once within a call of FindAllPoints. However, using three valued logic causes problems with finding jigsaw pieces - as we now illustrate. During application of the GA to the Von Koch Curve, the point (34, 13) is chosen from the store, and a call to FindAllPoints leaves the jigsaw in the state shown in Figure 7.7.

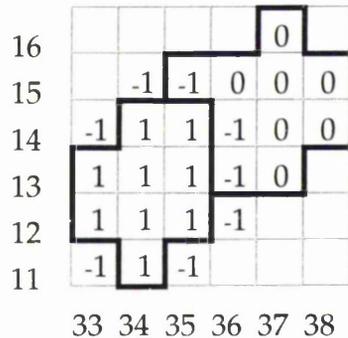


Figure 7.7 The state of the jigsaw after a call to FindAllPoints.

A few iterations later, the point (37, 13) is chosen from the store; since its value is 0, we call FindAllPoints again. Clearly all of the points marked 0 in Figure 7.7 will be found by this procedure, but those set to -1 will never be checked - despite the fact that they were checked at a previous call of FindAllPoints - and thus the piece will not be found correctly. The only way to solve this problem is to reset the -1's to 0's after each call to FindAllPoints. However, this is clearly time consuming, and therefore using three valued logic is not feasible.

Thus it is clear that the method described in this section can never be successful, irrespective of the map to which we apply it. Indeed, in cases where some of the maps of the IFS have either regular or semi-regular jigsaws, the best option so far seems to be to apply the methods of Chapters 5 and 6 to these maps, and simply use the original GA on those maps with irregular jigsaws. However, we now consider a second method which gives considerable improvement in certain cases.

7.2 Scanning Along Rows

When we consider regular jigsaws, it is a simple process to find all points $(x + i, y)$ which lie in the same jigsaw piece as (x, y) . Indeed, using (5.2), we can easily find the minimum and maximum values which i can take - we now show how we can find such values for maps with irregular jigsaws which, as we will see shortly, will enable us to describe a far better method for making use of irregular jigsaws than that of the previous section. We begin with a definition.

Definition 7.6 For a real number x , we define the *floor* of x to be the greatest integer which is less than or equal to x , and the *ceiling* of x to be the smallest integer which is greater than or equal to x . In symbols we have

$$\gamma(x) = \text{floor}(x) = \max\{i \in \mathbf{Z} : i \leq x\} \text{ and } \Gamma(x) = \text{ceiling}(x) = \min\{i \in \mathbf{Z} : i \geq x\}$$

Lemma 7.7 If $i \in \mathbf{Z}$ and $\lambda \in \mathbf{R}$ then the following hold

- (i) $\max\{i : i < \lambda\} = \min\{i : i \geq \lambda\} - 1 = \Gamma(\lambda) - 1$,
- (ii) $\min\{i : i > \lambda\} = \max\{i : i \leq \lambda\} + 1 = \gamma(\lambda) + 1$.

Theorem 7.8 Let $m, n, g, h \in \mathbf{R}$ and $i \in \mathbf{Z}$. Then $n + im \in [g, h)$ if and only if $i_{\min} \leq i \leq i_{\max}$, where i_{\min} and i_{\max} are given by

	i_{\min}	i_{\max}
(i) $m > 0$	$\Gamma\left(\frac{g-n}{m}\right)$	$\Gamma\left(\frac{h-n}{m}\right) - 1$
(ii) $m < 0$	$\gamma\left(\frac{h-n}{m}\right) + 1$	$\gamma\left(\frac{g-n}{m}\right)$.

Proof (i) If $m > 0$ then

$$n + im \geq g \Leftrightarrow i \geq (g - n)/m \Leftrightarrow i \geq \min\{i : i \geq (g - n)/m\} = \Gamma((g - n)/m)$$

Similarly,

$$n + im < h \Leftrightarrow i < (h - n)/m \Leftrightarrow i \leq \max\{i : i < (h - n)/m\} = \Gamma((h - n)/m) - 1,$$

where the last step is given by Lemma 7.7(i). Thus (i) is proved; the proof of (ii) is analogous, using Lemma 7.7(ii).

Now, we want to find all values of i such that $\underline{w}(x + i, y) = \underline{w}(x, y)$. Recalling that $w(x + i, y) = (w_h(x, y) + ia, w_v(x, y) + ic)$, this means that we want all values of i such that $w_h(x, y) + ia \in [\alpha, \alpha + 1)$ and $w_v(x, y) + ic \in [\beta, \beta + 1)$, where $\alpha = \underline{w}_h(x, y) - 1/2$ and $\beta = \underline{w}_v(x, y) - 1/2$. But $w_h(x, y) + ia \in [\alpha, \alpha + 1)$ if and only if $ih_{\min} \leq i < ih_{\max}$, and similarly $w_v(x, y) + ic \in [\beta, \beta + 1)$ if and only if $iv_{\min} \leq i < iv_{\max}$, where ih_{\min} , ih_{\max} , iv_{\min} and iv_{\max} are given by Theorem 7.8. Since both conditions must hold, we have that $\underline{w}(x + i, y) = \underline{w}(x, y)$ if and only if $i_{\min} \leq i < i_{\max}$ where $i_{\min} = \max(ih_{\min}, iv_{\min})$ and $i_{\max} = \min(ih_{\max}, iv_{\max})$. Clearly $i_{\min} > i_{\max}$ if and only if the intervals $[ih_{\min}, ih_{\max}]$ and $[iv_{\min}, iv_{\max}]$ do not intersect - which occurs if and only if the piece contains no points with vertical coordinate y and/or horizontal coordinate x .

Thus, for any point (x, y) , we can use Theorem 7.8 to find the maximum and minimum values of i such that $\underline{w}(x + i, y) = \underline{w}(x, y)$ and therefore we can find all other points of the jigsaw piece which have the same vertical coordinate. In order to find the whole jigsaw piece, we must then consider rows above and below our starting row. Initially, we have a flag with value 0 - moving up a row causes this flag to be set to 1, whilst moving down results in the flag being set to -1. This enables us to avoid rows being scanned more than once; we scan above the current row only if flag ≥ 0 and scan below only if flag ≤ 0 . The method is summarised in the following pseudo code, where $J(x, y)$ is the jigsaw position of (x, y) .

```

procedure ScanRows (x, y: integer; wh, wv: real; flag: integer)
  find  $i_{\min}$  and  $i_{\max}$ . If  $i_{\max} \geq i_{\min}$  then
    set  $J(i, y)$  to true for  $i_{\min} \leq i \leq i_{\max}$ 
    if flag  $\geq 0$  then ScanRows(x, y + 1, wh + b, wv + d, 1)
    if flag  $\leq 0$  then ScanRows(x, y - 1, wh - b, wv - d, -1)

```

Notice that when we move up or down a row we add or subtract b and d to the values of w_h and w_v respectively - this is because $w_h(x, y + 1) = w_h(x, y) + b$ and $w_v(x, y + 1) = w_v(x, y) + d$. Then a call of $\text{ScanRows}(x_0, y_0, w_h(x_0, y_0), w_v(x_0, y_0), 0)$ will result in all points of the piece being found and set to true - as the following example illustrates.

Example 7.9 Suppose we apply the above method to map 1 of the *Takagi Function*, where $a = c = d = 0.5$ and $b = 0$. Let $x_0 = y_0 = 2$ so that $w(x_0, y_0) = (1, 2)$. Then we begin by calling $\text{ScanRows}(2, 2, 1, 2, 0)$ and find that $i_{\min} = 1$ and $i_{\max} = 2$ so that we set $J(1, 2)$ and $J(2, 2)$ to true. We then call $\text{ScanRows}(2, 3, 1, 2.5, 1)$, finding that $i_{\min} = i_{\max} = 1$, and set $J(1, 3)$ to true. A call of $\text{ScanRows}(2, 4, 1, 3, 1)$ results in $i_{\min} = 5$ and $i_{\max} = 2$ - and thus there are no piece pixels with vertical coordinate 4.

We then move back to the original layer of the recursion, and move down a row, calling $\text{ScanRows}(2, 1, 1, 1.5, -1)$. This time we find that $i_{\min} = i_{\max} = 2$ so that we set $J(2, 1)$ to true. Finally, a call of $\text{ScanRows}(2, 0, 1, 1, -1)$ shows that there are no piece pixels with vertical coordinate 0 and the procedure terminates. The piece is shown in Figure 7.8 - we know from Chapter 5 that this is correct.

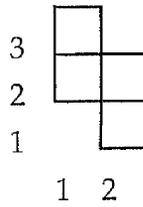


Figure 7.8 The jigsaw piece found using ScanRows.

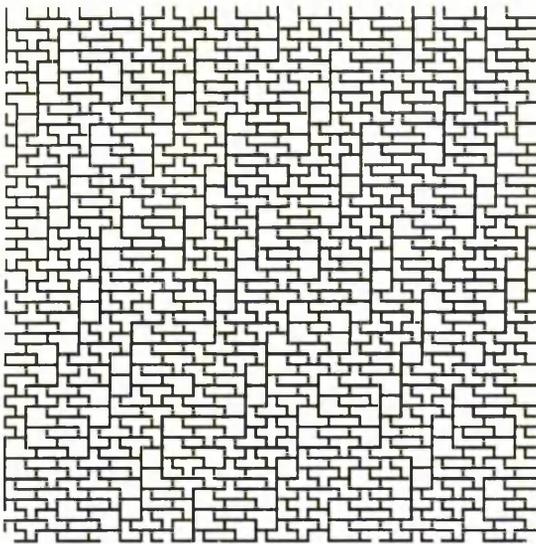
We are now in a position to consider whether or not this method will improve the performance of the Graphical Algorithm, and begin by looking at the *Tree 1* IFS once again.

Example 7.10 Recall that each of the five maps of the *Tree 1* IFS has a distinct jigsaw, and therefore we must consider each map separately. Table 7.4 shows the results obtained using the original GA, and the GA using ScanRows.

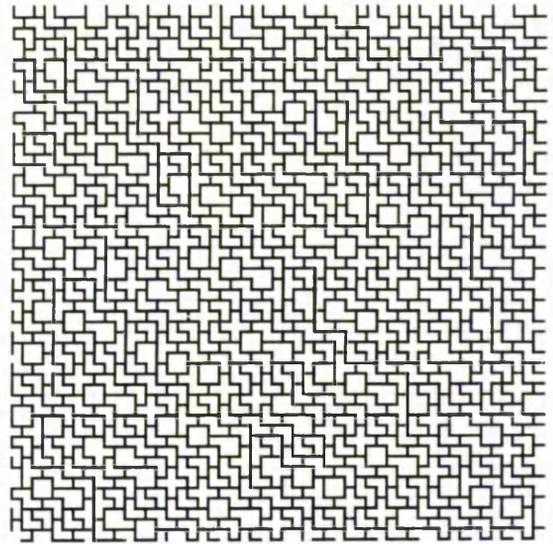
	Original GA	GA using ScanRows
Pts. Calculated	9095	2146
Unique Points	1819	1819
Time (seconds)	25.17	48.28

Table 7.4 Results for the *Tree 1* IFS, with and without jigsaws.

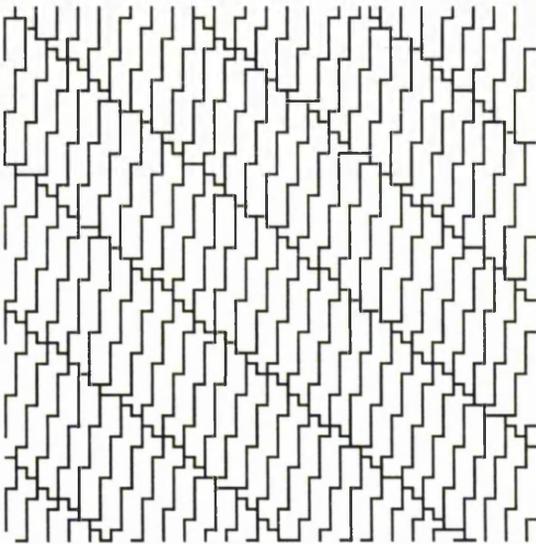
It is clear from the table that using ScanRows in this case does not improve the GA - in fact the time taken to produce the attractor is almost doubled. In order to see why this is the case, we must first consider what the jigsaws of the maps look like - parts of those of maps 1 to 4 are shown in Figure 7.9, while the jigsaw of map 5 consists of rectangular pieces which are at most 2×2 pixels in size.



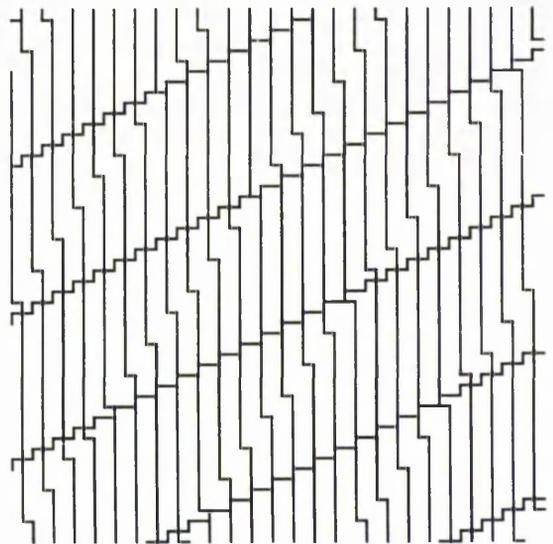
Map 1



Map 2



Map 3



Map 4

Figure 7.9 Parts of the jigsaws of maps 1 to 4 of the Tree 1 IFS.

As we see from Figure 7.9, the jigsaw pieces of maps 3 and 4 are "tall" - that is, the range of vertical coordinates of the piece pixels is large - and therefore, whatever our starting vertical value is, ScanRows will call itself recursively many times. Clearly this means that the process of finding the whole jigsaw piece will be time consuming - and consequently ScanRows is not ideal for such jigsaws. However, the range of horizontal coordinates in each piece is reasonably small. Thus if we alter the definition of ScanRows so that i_{\min} and i_{\max} are the minimum and

maximum vertical values for a given horizontal value x , and the recursive calls consider columns to the left and right of x instead of rows above and below y , we would expect the method to be more efficient. This is indeed the case, with the time taken reduced from 48.28 seconds to 39.51 seconds. Notice, however, that this is still approximately 14 seconds slower than the original GA and therefore there must be another reason for the disappointing performance of ScanRows.

Recall that, in the majority of IFS's, most of the jigsaw pieces contain non attractor points as well as attractor points, and that the maps of the IFS are only ever applied to attractor points. This essentially means that if a jigsaw piece consisting of M pixels contains only P attractor points then, to ensure that the time taken to produce the attractor is reduced, any attempt to find the jigsaw piece must take less time than applying the N maps to these P points would. Thus if we have an IFS map where most of the jigsaw pieces contain few attractor points (relative to the size of the piece) it is extremely difficult to find an efficient algorithm which finds and sets to true each of the pixels of the piece. Note, however, that while some of the examples of Chapters 5 and 6 were such that the jigsaw pieces contained relatively few attractor points, we were able to make improvements in these cases by reducing the jigsaws - something which we cannot do in irregular cases.

In fact, it is this which makes ScanRows inefficient in the case of the Tree 1 IFS. As we have already seen, the jigsaw pieces of maps 3 and 4 are reasonably large and we find that they each contain only a few attractor points. Therefore much of the calculation which results from calling ScanRows is unnecessary and the time taken to produce the attractor will be increased.

Thus in cases where the jigsaw pieces contain few attractor points relative to their size, ScanRows will fail to produce any improvement on the performance of the

Graphical Algorithm. However, as our next example shows, if all jigsaw pieces consist mainly of attractor pixels, ScanRows gives impressive results.

Example 7.11 The *Fish* IFS consists of nine maps, given at scale 100 by

$$\begin{aligned}w_1(x, y) &= (0.525x + 0.099y - 6.5, 0.734y + 12.7) \\w_2(x, y) &= (0.408x + 0.119y + 35.2, -0.144x + 0.396y + 53.2) \\w_3(x, y) &= (0.268x + 34.5, 0.379y + 12.5) \\w_4(x, y) &= (0.273x + 49.1, 0.379y + 30.7) \\w_5(x, y) &= (0.322x - 0.098y + 54.6, 0.273x + 0.216y + 0.05) \\w_6(x, y) &= (0.097x + 0.22y + 71.4, -0.107x + 0.261y + 46.9) \\w_7(x, y) &= (0.098x + 0.222y + 72.8, 0.088x - 0.247y + 44.4) \\w_8(x, y) &= (-0.357x - 0.205y + 61.8, 0.092x - 0.178y + 15) \\w_9(x, y) &= (0.017x + 0.423y + 20.3, -0.267x - 0.014y + 96.7)\end{aligned}$$

The attractor is shown in Figure 7.10.



Figure 7.10 The attractor of the Fish.

It is easy to see that the nine maps will have distinct jigsaws, and also that none of the jigsaws will be regular (although those of maps 3 and 4 consist only of rectangular pieces). Further, because the attractor is almost completely "filled in", many of the jigsaw pieces will consist solely of attractor pixels, so that we would

expect ScanRows to be successful in improving the performance of the Graphical Algorithm. This is indeed the case, as we see from the results given in Table 7.5.

	Original GA	GA using ScanRows
Pts. Calculated	61443	8426
Unique Points	6827	6827
Time (seconds)	221.73	139.58

Table 7.5 Results for the Fish IFS, with and without jigsaws.

Note Using FindAllPoints on the Fish *increases* the time taken to produce the attractor by over 70 seconds.

Thus it would appear that using ScanRows to find jigsaw pieces is feasible in cases where the majority of the pieces contain more attractor pixels than non attractor pixels. Since this property holds for a number of the IFS's which we have already seen in Chapters 5 and 6, we will now briefly consider how successful ScanRows is for IFS's with regular or semi-regular jigsaws in relation to our earlier methods. Henceforth, we will use the term *(semi)-regular* to mean either regular or semi-regular.

7.3 Scanning Rows of (Semi)-Regular Jigsaws

Recall that previously we dealt with maps with (semi)-regular jigsaws by finding a scheme for reducing the jigsaw so that each piece is represented by a single pixel. This enabled us to check whether or not applying a particular map to z_n would result in a new point simply by finding the reference point for z_n , and applying the map only if the reference point was false in the reduced jigsaw. As we saw in Chapters 5 and 6, this method gave considerable improvements in the performance of the GA, particularly in those cases where two or more of the maps

had identical jigsaws. However, we still had to calculate a reference point for every point chosen from the store (i.e. every point of the attractor) - and while the way that we did this using `div` and `mod` was quicker than finding $w(x, y)$, it seems likely that removing such calculation by using `ScanRows` would give further improvement. We begin with two examples which show that this is indeed true in those cases where most jigsaw pieces contain relatively many attractor points.

Example 7.12 The *Square* is given at scale 100 by

$$w_1(x, y) = (0.5x, 0.5y)$$

$$w_2(x, y) = (0.5x + 50, 0.5y)$$

$$w_3(x, y) = (0.5x, 0.5y + 50)$$

$$w_4(x, y) = (0.5x + 50, 0.5y + 50)$$

The attractor is shown in Figure 7.11.

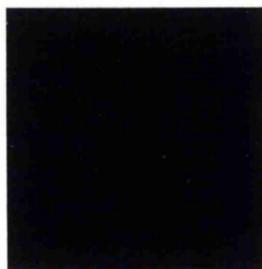


Figure 7.11 The attractor of the Square.

It is immediately obvious from Figure 7.11 that the majority of jigsaw pieces will consist solely of attractor points; the only ones which contain non attractor points being those around the edges of the attractor. Table 7.6 shows that, as expected, `ScanRows` performs extremely well for this IFS - even when compared to the previous best improvement method. Note that in both improvement methods we use one jigsaw to cover all four maps.

	Original GA	Previous Best	ScanRows
Pts. Calculated	40000	10000	10000
Unique Points	10000	10000	10000
Time (seconds)	194.38	74.7	44.65

Table 7.5 Results for the Square.

Thus it is clear that using ScanRows is far better in this case, enabling us to improve the GA by more than the target ratio of 4 : 1. Of course, this is the best possible example, since almost all of the jigsaw pieces consist solely of attractor points. However, ScanRows can also be effective when more of the pieces contain some non attractor points - as our next example illustrates.

Example 7.13 The jigsaw pieces of both the *Sierpinski Carpet* and *Peano Curve* are such that many contain solely attractor points, while the majority of the rest contain more attractor points than non attractor points. Further, recall from Chapter 5 that each IFS requires only one jigsaw. Table 7.6 gives the results of using ScanRows in each case, with the results of the previous best method reproduced from Chapter 5 for comparison.

	Sierpinski Carpet			Peano Curve		
	GA	Previous	ScanRows	GA	Previous	ScanRows
Pts. Calculated	63360	8192	8192	45000	5202	5202
Unique Points	7920	7920	7920	5000	5000	5000
Time (seconds)	239.76	62.6	30.77	155.33	37.97	19.15

Table 7.6 Results for the Sierpinski Carpet and Peano Curve.

Thus ScanRows gives far better improvements in both the Sierpinski Carpet and Peano Curve, with the ratios being only slightly less than the target ratios of 8 : 1 and 9 : 1 respectively.

However, whilst ScanRows enables us to achieve impressive improvements for IFS's such as the Square, Sierpinski Carpet and Peano Curve, there are many IFS's for which the majority of jigsaw pieces consist of mainly non attractor points - for such IFS's, ScanRows gives inferior results to previous improvement methods (where such improvements are possible). This is illustrated in our final example of this section.

Example 7.14 Many of the jigsaw pieces of both the *Sierpinski Gasket* and *Takagi Function* contain only one attractor point, while others contain only two (recall that the jigsaw pieces of both IFS's consist of four pixels). Thus neither IFS is ideal for using ScanRows - as the results in Table 7.7 show.

	Sierpinski Gasket			Takagi Function		
	GA	Previous	ScanRows	GA	Previous	ScanRows
Pts. Calculated	6237	2079	2079	574	287	287
Unique Points	2079	2079	2079	287	287	287
Time (seconds)	17.93	7.08	11.47	1.95	0.95	6.32

Table 7.7 Results for the Sierpinski Gasket and Takagi Function.

It would appear from Sections 7.1 and 7.2 that, unless we have an IFS where the majority of jigsaw pieces contain many attractor points, there is no way of using irregular jigsaws to improve the GA. However, it is often possible to find a (semi)-regular jigsaw which has few differences from a given irregular one, as Figure 7.12 shows.

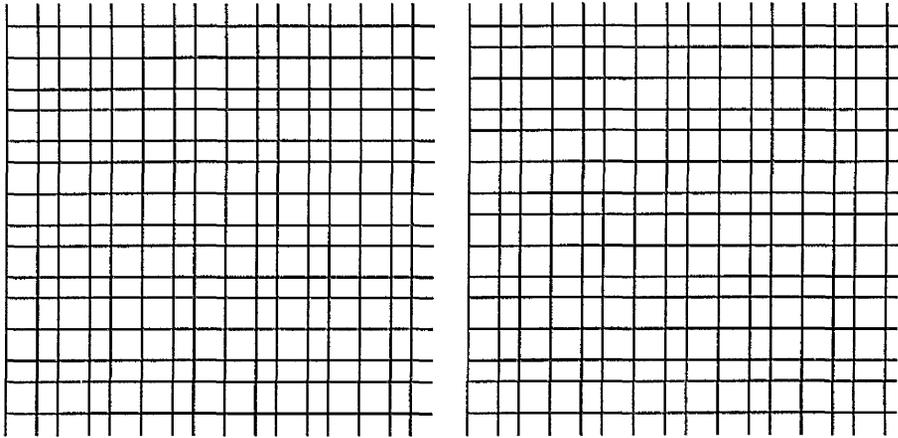


Figure 7.12 Two jigsaws arising from similar maps. The jigsaw on the left is irregular, while the one on the right is $(8, 8)$ -semi-regular.

The jigsaw on the right is $(8, 8)$ -semi-regular, and arises from the map with IFS code $(\frac{3}{8}, 0, 0, \frac{3}{8}, 0.307, 0.619)$, while the one on the left is the jigsaw of map 1 of the *5 Fold Crystal*, which has code $(0.382, 0, 0, 0.382, 0.307, 0.619)$. We see that there is very little difference between the coefficients of the two maps - but one has an irregular jigsaw (since its smallest building block has dimensions 500×500), while the other's jigsaw is semi-regular. If we could substitute the new map for the old one, and do likewise for the other four maps of the IFS which all have the same first four code coefficients, then we would clearly be able to use the methods given in earlier chapters to improve the GA. However, this is only feasible either if it does not alter the attractor at all, or if any alteration is 'acceptable', in that it does not change the appearance significantly. We will return to the question of approximating maps in order to avoid irregular jigsaws in Chapter 8.

Note When we approximate a map w in this manner, the part of the attractor produced by w may have its area altered slightly. Because of this, it may be necessary to change the translation part of the IFS code in order to ensure that the position of this part relative to the other maps is unchanged. However, this will have no effect on whether or not the jigsaw is (semi)-regular.

Before we consider such approximations, however, there is one other method which may be useful in cases where one map of the IFS may be written as a reflection of the other in either a horizontal or vertical line - as we had with maps 3 and 4 of the Von Koch Curve. We end this chapter by considering such Iterated Function Systems, and show that the performance of the Graphical Algorithm can be improved slightly in these cases.

7.3 Reflecting Maps in a Horizontal or Vertical Line

As we saw earlier, map 4 of the Von Koch Curve is simply the reflection of map 3 in the line $x = \text{scale}/2$, where for our purposes, $\text{scale} = 99$. Further, as the attractor is symmetrical about this line, we can alter map 2 slightly so that it is a reflection of map 1 without affecting which points are plotted by the algorithm, or how many points are in the attractor. The new map 2 is given by

$$w_2(x, y) = (99 - 1/3x, 1/3y).$$

Now, it would seem reasonable to assume that, since maps 2 and 4 are reflections of maps 1 and 3 respectively, the points plotted by maps 2 and 4 will simply be reflections of those plotted by maps 1 and 3. Thus we could calculate maps 1 and 3 and then plot both the points resulting from these maps, and their reflections in $x = \text{scale}/2$, noting that the reflection of a point (x, y) in this line is given by $(\text{scale} - x, y)$. As before, we can use the jigsaw for map 1 to reduce duplication.

At each iteration of the original GA, we must perform two calculations of the type $ax + by + e^*$ (where $e^* = e \times \text{scale}$) for each map, giving a total of 8. Using this method enables us to reduce the number of such calculations from 8 to 4 at each iteration, whilst adding in two calculations of the type $a - b$, which can be done much faster, suggesting that this method will save time. Combining it with the use

of the map 1 jigsaw will hopefully give us our best results yet for this IFS - Table 7.8 shows that this is indeed the case. Note that the final column contains the results of using both reflections *and* the map 1 jigsaw (the map 2 jigsaw is not required as we do not calculate w_2 directly).

	Original GA	Jigsaws for w_1, w_2	Reflections
Pts. Calculated	1616	1000	1000
Unique Points	404	404	404
Time (seconds)	4.25	2.98	1.88

Table 7.8 Results of using reflections in the Von Koch Curve.

In fact, seven of the IFS's with at least one irregular map from our list of fifty have one map which is a reflection of another in either a horizontal or vertical line - and in each case, any remaining maps have (semi)-regular jigsaws. The seven are the Face, Barnsley's Frizzy, the Grabber, the Leaf Outline, Tree 3 and the Triangle Crab. For each of these IFS's, we could apply similar methods to those described for the Von Koch Curve to obtain improvements in the time taken to produce the attractor.

Note In some references, map 4 of the Von Koch Curve is not a straightforward reflection of map 3. However, taking w_4 to be the reflection of w_3 makes no difference to the attractor and, further, it means that the two jigsaws are identical - which allows us to consider only one jigsaw. We will shortly see that there are many situations where altering maps slightly make their jigsaws easier to deal with and in a number of cases, such alterations make no difference to the attractor.

However, while making use of any maps which are reflections of others does give us a small time improvement, it does not reduce the number of points calculated as all we are doing is changing the way that some of the calculations are done. The

improvements can therefore never be as good as those that we could achieve with jigsaws, assuming we have suitable jigsaws. We have seen in this chapter that using irregular jigsaws to improve the Graphical Algorithm generally increases the run time - unless the IFS is such that the majority of jigsaw pieces contain many attractor points - and we have shown why this is the case by considering the amount of work involved in finding the jigsaw pieces. Thus our only remaining option is to approximate maps with irregular jigsaws, and hope that at least one such approximation gives an acceptable change in the attractor and a semi-regular jigsaw which can be used to improve the GA.

Chapter 8

Approximating Irregular Jigsaws

We saw in Chapter 7 that it is not feasible to use irregular jigsaws to improve the Graphical Algorithm in all but a few cases, since whatever method we use to find the jigsaw pieces causes an increase in the run time of the GA. At first it would appear that this limits the range of IFS's for which we can improve the GA to those which have at least one map having a (semi)-regular jigsaw or those for which the majority of jigsaw pieces contain a relatively large number of attractor points, but, as we suggested at the end of Section 7.2, this is not the case. If we allow small alterations in the appearance of the attractor then approximating maps may increase the number of IFS's to which we can apply our theory. In this chapter, we will consider two ways of approximating maps so that their jigsaws become semi-regular - allowing us to use the methods described in Chapters 5 and 6 to improve the Graphical Algorithm - and show that, while approximation is not appropriate for every map, it will allow us to at least partially improve the Graphical Algorithm for every one of our fifty Iterated Function Systems.

Note While approximating maps will usually cause some small alteration in the appearance of the attractor, there are cases where approximation leaves the attractor unchanged, as we will see in Example 8.7.

8.1 Approximation by Nearest "Small" Rational

When we approximate a map we want to use alternative code values which are as close as possible to the originals so that any change to the appearance of the attractor is minimal. However, in order to be able to use the methods of Chapters 5 and 6, we also want to obtain a small building block. Recall from Theorem 6.13

that if we write each of the first four entries of the IFS code as a rational number in lowest terms - a_1/a_2 etc. - then the dimensions of the smallest building block are $m \times n$, where $m = \text{lcm}(a_2, c_2)$ and $n = \text{lcm}(b_2, d_2)$. Thus we want to choose rationals P/q with q small. Although we will typically allow q to take values up to and including $q = 20$, we actually want to have $m, n \leq 20$ and so we will often require q to be much smaller than 20. Wherever possible, we will attempt to have $p = 1$. For the remainder of this chapter, we shall refer to such rationals P/q as *small*. In this section we consider choosing appropriate small rationals by inspection, making use of the fact that the denominators must not exceed 20. Example 8.1 illustrates the method for the 4 Fold Crystal.

Example 8.1 The 4 Fold Crystal consists of four maps, given at scale 100 by

$$w_i(x, y) = (0.255x + e_i^*, 0.255y + f_i^*), 1 \leq i \leq 3$$

$$w_4(x, y) = (0.37x - 0.642y + 63.6, 0.642x + 0.37y - 0.6)$$

where $(e_i^*, f_i^*) = (37.3, 67.1), (11.5, 22.3), (63.1, 22.3)$.

For each of the first three maps, the dimensions of the building block are given by the denominators of a and d (see Note following Theorem 6.14). We require the denominators of our approximations to be less than or equal to 20 - considering each possible denominator in turn, and running through all possible values for the numerator, we find that the closest options to 0.255 are as follows:

$1/2 = 0.500$	$1/3 \approx 0.333$	$1/4 = 0.250$	$1/5 = 0.200$
$2/6 \approx 0.333$	$2/7 \approx 0.286$	$2/8 = 0.250$	$2/9 \approx 0.222$
$3/10 = 0.300$	$3/11 \approx 0.273$	$3/12 = 0.250$	$3/13 \approx 0.231$
$3/14 \approx 0.214$	$4/15 = 0.267$	$4/16 = 0.250$	$4/17 \approx 0.235$
$5/18 \approx 0.278$	$5/19 \approx 0.263$	$5/20 = 0.250$	

Now, since we want the approximation to be as close as possible to the original value, we will use $1/4$ in place of 0.255 - doing so will make the jigsaws of maps 1, 2 and 3 regular, with all jigsaw pieces being 4×4 squares of pixels. Figure 8.1 shows that making this alteration has very little effect on the appearance of the attractor, while Table 8.1 gives the results obtained when we use approximations for maps 1, 2 and 3 and then make use of the resultant regular jigsaws in the GA.



Figure 8.1 The attractors produced using (a) $a, d = 0.255$ and (b) $a, d = 0.25$.

	Original GA	Improved GA
Pts. Calculated	5044	1712
Unique Points	1261	1261
Time (seconds)	13.48	7.03

Table 8.1 Results of improving the GA for the 4 Fold Crystal.

We see from the table that using approximation has enabled us to improve the performance of the GA significantly for this IFS. Note that we have not made any attempt to approximate map 4 - the reason for which will be given in Section 8.3 - explaining why there are still 451 duplicated points. In fact, this extra duplication can be almost completely removed using the ScanRows procedure described in Chapter 7 but, while the number of points calculated is reduced to 1263, the time taken to produce the attractor *increases* to 19.58 seconds.

In this example, although we calculated the nearest small rational for every possible denominator, we could easily have predicted that the best option would

be to use $1/4$. Unfortunately, however, the process is not so easy for most values which we wish to approximate. Indeed, we may occasionally need to approximate an irrational number and unless we first approximate such values by taking them to a small number of decimal places, finding the nearest small rational by inspection is not easy. Thus it is clear that in most cases we require a better method of finding the nearest small rational - the method which we shall use is *Rational Approximation*.

8.2 Rational Approximation

As we noted previously, in some cases it is easy to spot the nearest small rational to a given decimal number, as we saw in Example 8.1. However, in the majority of cases it will not be immediately obvious what the most convenient option is. In such cases, using inspection involves calculating at least 20 rationals (one for each possible denominator). Indeed, for a particular denominator it is often not easy to see what value the numerator should take, and so we may have to do considerably more than 20 calculations. However, there is a simple process for finding the best rational approximation to a given irrational number. This process - known as *continued fraction approximation* - is described in Theorem 8.2; note that it may also be used to approximate a decimal.

Theorem 8.2 [24] Let $\lambda = \lambda_0$ be an irrational number and define

$$a_i = [\lambda_i], \quad \lambda_{i+1} = \frac{1}{\lambda_i - a_i} \quad \text{for } i \geq 0. \quad (8.1)$$

Further, define the sequences $[h_i]$ and $[k_i]$ by

$$\begin{aligned} h_{-2} &= 0, \quad h_{-1} = 1, \quad h_i = a_i h_{i-1} + h_{i-2} \quad \text{for } i \geq 0 \\ k_{-2} &= 1, \quad k_{-1} = 0, \quad k_i = a_i k_{i-1} + k_{i-2} \quad \text{for } i \geq 0 \end{aligned} \quad (8.2)$$

Then the quotients h_n/k_n form a sequence of rational approximations to λ , with the corresponding error given by

$$\left| \lambda - \frac{h_i}{k_i} \right| < \frac{1}{k_i k_{i+1}}. \quad (8.3)$$

Thus, whenever we have an irrational code entry, we can find a sequence of approximations to it by Theorem 8.2, with the approximations h_i/k_i getting successively closer to the true value. However, in the majority of our IFS's, the code entries are themselves rational (e.g. finite decimals), and we require only to find suitable approximations with small denominators. Fortunately we can also apply the Theorem to $\lambda \in \mathbf{Q}$, noting that, since the approximations tend toward the true value of λ which is itself rational, we must have $h_n/k_n = \lambda$ for some finite n , with the process defined by (8.1) and (8.2) terminating at this stage since $a_n = \lambda_n$ and λ_{n+1} is undefined. We therefore have the following Corollary.

Corollary 8.3 If λ is a rational number, the sequence of approximations given by Theorem 8.2 is finite, terminating with λ .

Example 8.4 Suppose $\lambda = 0.382$. Then Table 8.2 gives the values of λ_i , a_i , h_i and k_i for $0 \leq i \leq 10$. Since $\lambda_{10} = a_{10}$ the approximation h_{10}/k_{10} gives λ as a rational in lowest possible terms.

i	λ_i	a_i	h_i	k_i
0	0.382	0	0	1
1	2.618	2	1	2
2	1.619	1	1	3
3	1.616	1	2	5
4	1.622	1	3	8
5	1.607	1	5	13
6	1.647	1	8	21
7	1.545	1	13	34
8	1.833	1	21	55
9	1.200	1	34	89
10	5.000	5	191	500

Table 8.2 Values of λ_i , a_i , h_i and k_i for $\lambda = 0.382$.

We therefore have ten non zero approximations to $\lambda = 0.382$ of which five have denominator smaller than 20, namely $1/2$, $1/3$, $2/5$, $3/8$ and $5/13$.

When we use (8.1) and (8.2) to find rational approximations to a given value, since we require the approximations to be small, we terminate the process as soon as k_i exceeds 20 and use one of the earlier approximations in place of the original value in the IFS map. Our next example shows how the approximations we found in Example 8.4 can help us to improve the performance of the GA for the 5 Fold Crystal.

Note Using this process with $\lambda = 0.255$ gives $h_2 = 1$ and $k_2 = 4$, and so the best small rational approximation to 0.255 is $1/4$ - exactly as we found by inspection in Example 8.1.

Example 8.5 The *5 Fold Crystal* consists of 5 maps, given at scale 100 by

$$w_i(x, y) = (0.382x + e_i^*, 0.382y + f_i^*), 1 \leq i \leq 5,$$

where $(e_i^*, f_i^*) = (30.7, 61.9), (60.3, 40.4), (1.4, 40.4), (12.5, 6), (49.2, 6)$.

As in Example 8.1, the dimensions of the building blocks in this case are given by the denominators of a and d , and therefore we require the denominator of the best approximation to be at most 20. As we saw in Example 8.4, there are five such options - we consider the last two, $3/8$ and $5/13$, and begin by showing in Figure 8.2 that using these values in place of 0.382 makes only a very small difference to the appearance of the attractor.

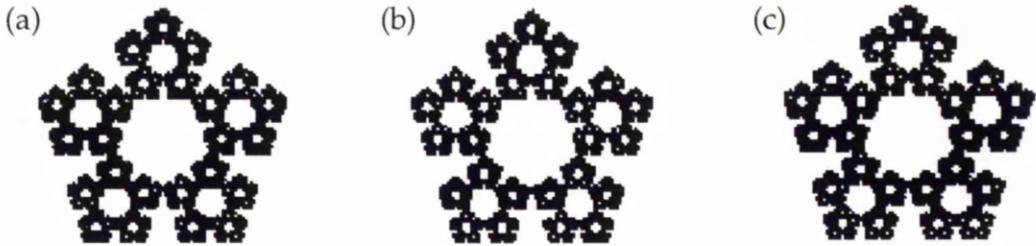


Figure 8.2 The attractors produced using (a) 0.382, (b) 0.375 and (c) $5/13$.

Clearly, if we use $3/8 = 0.375$ in place of 0.382 then the jigsaws will be $(8, 8)$ -semi-regular (as noted in Section 7.2), while if we use $5/13$ then the jigsaws will be $(13, 13)$ -semi-regular. Figure 8.3 shows the building block for map 1 using 0.375, while Figure 8.4 shows that for map 1 using $5/13$. The building blocks for the other four maps are similar.

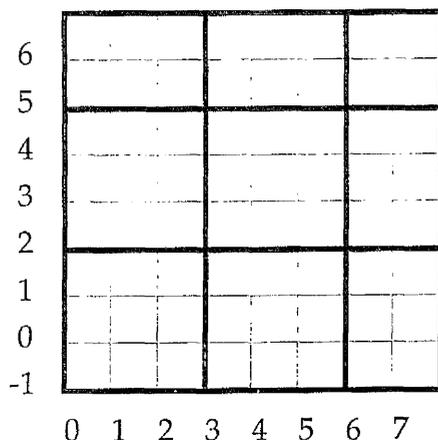


Figure 8.3 The building block for map 1 of the
5 Fold Crystal using 0.375 in place of 0.382.

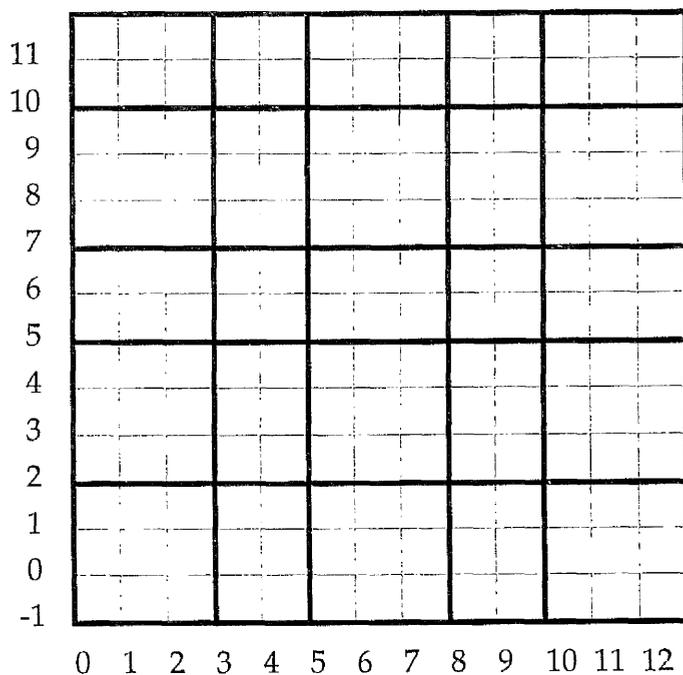


Figure 8.4 The building block for map 1 of the
5 Fold Crystal using $5/13$ in place of 0.382.

Note Both building blocks consist of horizontal and vertical bands and so we can use (6.5) and its vertical counterpart to find the reduction scheme. For example, the horizontal reduction scheme for the jigsaw formed from the building block in Figure 8.3 is

$$x \rightarrow \begin{cases} 3(x \operatorname{div} 8) + 1 & \text{if } x \equiv 0, 1, 2 \pmod{8} \\ 3(x \operatorname{div} 8) + 2 & \text{if } x \equiv 3, 4, 5 \pmod{8} \\ 3(x \operatorname{div} 8) + 3 & \text{if } x \equiv 6, 7 \pmod{8} \end{cases}$$

Thus whatever approximation we use results in building blocks which we can use to improve the GA for the 5 Fold Crystal. Table 8.3 shows the results obtained for both approximations.

	Original GA		Improved GA	
	$a, d = 3/8$	$a, d = 5/13$	$a, d = 3/8$	$a, d = 5/13$
Pts. Calculated	15075	17105	3016	3434
Unique Points	3015	3421	3015	3421
Time (seconds)	46.58	56.47	25.57	34.27

Table 8.3 Results of improving the GA for the 5 Fold Crystal for two distinct approximations.

We see from Table 8.3 that both approximations give an improvement in the performance of the Graphical Algorithm, although using $a, d = 0.375$ gives a slightly better improvement, reducing the time taken to 55% of its original value, while the figure for $a, d = 5/13$ is 61%. In fact, we could have predicted this since as the size of the denominator increases, the reduction scheme gets more complicated and consequently it takes longer to find jigsaw positions within the algorithm. Note that, as we saw in Chapter 7, this time saving would not have been possible if we had left the maps unaltered.

Thus it would appear that using some form of approximation can indeed enable us to increase the number of IFS's for which we can improve the Graphical Algorithm and, further, that rational approximation is the best available method. So far we have only considered IFS maps which have two code entries equal to zero, where the dimensions of the building blocks are given by the denominators

of the rational approximations. In these cases, we have been able to use h_i/k_i , where $k_i \leq 20$ and $k_{i+1} > 20$, as our approximation, and therefore the approximations are as good as they can be within our defined limits. However, in cases where more than two code entries are non zero, we may have to take less accurate approximations in order to ensure that the building blocks are small enough. Our next example is of an IFS which has two maps which have semi-regular jigsaws, and two maps which have irregular jigsaws - where in each case we have only one zero entry in the IFS code.

Example 8.6 Consider the *Wire* IFS, which consists of four maps, given at scale 100 by

$$w_1(x, y) = (0.5x + 0.25y, 0.5y)$$

$$w_2(x, y) = (0.5x - 0.25y + 50, 0.5y)$$

$$w_3(x, y) = (0.227x - 0.146y + 40, 0.492y + 50.4)$$

$$w_4(x, y) = (0.224x + 0.152y + 41.4, 0.5y + 50.4)$$

It is easy to see that maps 1 and 2 have (2, 4)-semi-regular jigsaws. In fact, the jigsaw of map 1 is similar to that of maps 1 and 2 of the Sleepy Hollow while the jigsaw of map 2 is similar to that of maps 3 and 4 of the Sleepy Hollow - see Example 6.8. It is also easy to see that the jigsaws of maps 3 and 4 will be irregular, and therefore they must be approximated.

For map 3, we must find small rationals which are as close as possible to each of 0.227, 0.146 and 0.492. For 0.227, rational approximation gives two possible options - $1/4 = 0.25$ and $2/9 \approx 0.222$ - and since, if possible, we wish to have $p = 1$, we will use the first option, $1/4$. Similarly, for 0.146 we use $1/7 \approx 0.143$, while for 0.492 we use $1/2$. Thus we obtain the alternative map w_3^* given below, which is (4, 14)-semi-regular. Repeating this process for map 4 gives w_4^* which is

(4, 20)-semi-regular. Note that, in this case, the best feasible approximation to 0.152 is $3/20 = 0.15$.

$$w_3^*(x, y) = (1/4x - 1/7y + 40, 1/2y + 50.4)$$

$$w_4^*(x, y) = (1/4x + 3/20y + 41.4, 1/2y + 50.4).$$

Figure 8.5 shows the attractors obtained by applying the Graphical Algorithm to the IFS's given by (a) $\{w_{1-4}\}$ and (b) $\{w_1, w_2, w_3^*, w_4^*\}$ - demonstrating that the two attractors are virtually indistinguishable.

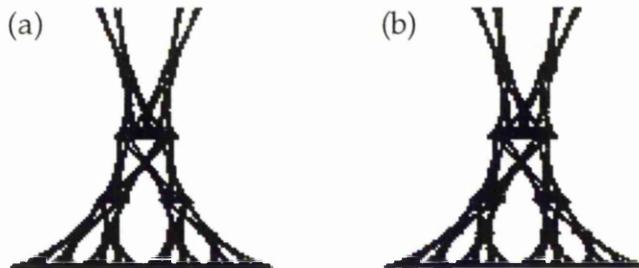


Figure 8.5 The Wire attractors obtained using (a) $\{w_{1-4}\}$ and (b) $\{w_1, w_2, w_3^*, w_4^*\}$.

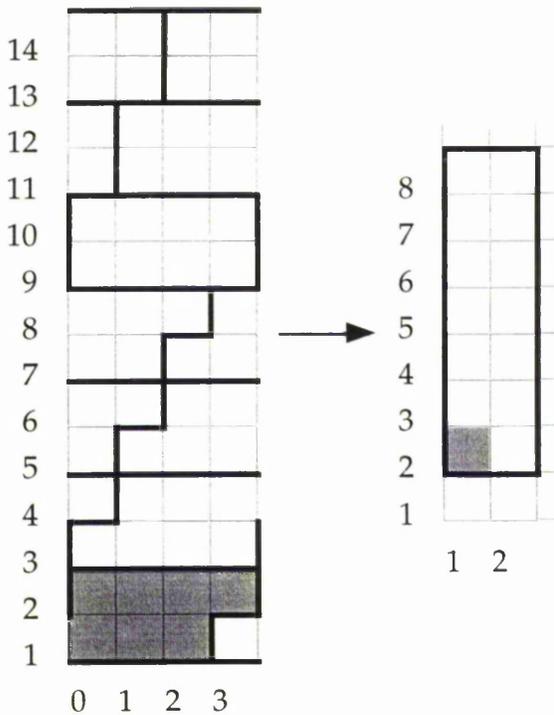
Since the two attractors are very similar, we allow the use of w_3^* and w_4^* in place of w_3 and w_4 - doing so enables us to improve the performance of the GA for this IFS using the methods described in Chapter 6. The building blocks for maps 3* and 4* are shown in Figure 8.6.

Now, since these building blocks do not consist solely of rectangular jigsaw pieces, we cannot use (6.5) to find the reduction schemes but, as with the *Crab* of Example 6.9, the schemes are still reasonably easy to find. The scheme for map 3 is as follows; that of map 4 is similar, but requires us to consider values of y modulo 20 in order to find the horizontal reduction scheme, as the building block for map 4 has dimensions 4×20 .

$$x \rightarrow \begin{cases} (x+4) \text{ div } 4 \text{ if } y \equiv 2, 9, 10 \pmod{14} \\ (x+5) \text{ div } 4 \text{ if } y \equiv 1, 8 \pmod{14} \\ (x+6) \text{ div } 4 \text{ if } y \equiv 6, 7, 13, 14 \pmod{14} \\ (x+7) \text{ div } 4 \text{ if } y \equiv 4, 5, 11, 12 \pmod{14} \\ (x+8) \text{ div } 4 \text{ if } y \equiv 3 \pmod{14} \end{cases}$$

$$y \rightarrow (y+3) \text{ div } 2$$

(a)



(b)

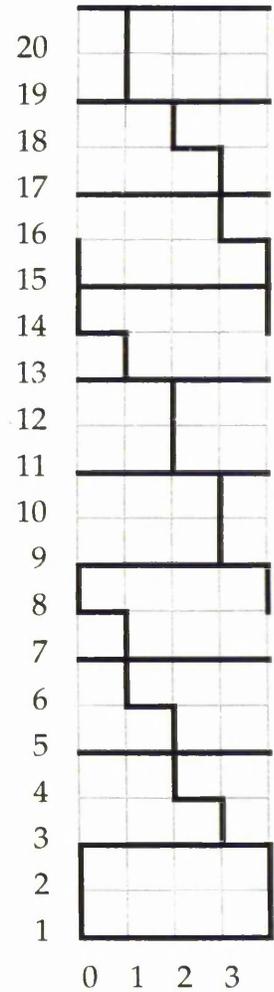


Figure 8.6 The building blocks for (a) map 3*, together with the part of the reduced jigsaw to which it is sent (shown bold), and (b) map 4* of the Wire.

Thus we are now in a position to use jigsaws for all four maps of this IFS in order to improve the performance of the Graphical Algorithm. Note that, since the

jigsaws of the four maps are all different, we must use four separate jigsaws to store all required data. The results are given in Table 8.4.

	Original GA	Improved GA
Pts. Calculated	8164	2278
Unique Points	2041	2041
Time (seconds)	21.05	10.38

Table 8.4 Results of improving the GA for the Wire.

As with Example 8.5, we see that a considerable time saving has been made by using jigsaws, better than halving the time taken to produce the attractor.

Examples 8.1, 8.5 and 8.6 have shown that using approximation is an extremely useful way of getting around the problems caused by maps with irregular jigsaws. In each case, although the attractor produced using the approximated maps was not identical to the original attractor, the two attractors were clearly very similar. In fact, as we commented earlier, there are cases where using an approximated map can lead to no alteration at all in the attractor - as our final example of this section illustrates.

Example 8.7 The *Leaf* consists of four maps, given at scale 10 by

$$\begin{aligned}
 w_1(x, y) &= (-0.47, 0.278y + 1.72) \\
 w_2(x, y) &= (0.525x + 0.020y - 0.17, 0.006x + 0.786y + 21.7) \\
 w_3(x, y) &= (0.201x - 0.241y + 0.49, 0.385x + 0.365y + 16.2) \\
 w_4(x, y) &= (-0.142x + 0.235y - 1.91, 0.377x + 0.393y + 13)
 \end{aligned}$$

Clearly each of the maps has an irregular jigsaw, but map 1 can easily be approximated to give a (semi)-regular jigsaw. Using rational approximation, we

obtain three options - $1/4$, $2/7$ and $5/18$ - and we find that replacing d in w_1 by any of these three values results in exactly the same attractor as we would get using the original w_1 , with 1093 unique points. Thus, since it has a regular jigsaw with pieces which are bands of infinite length and height 4, we use

$$w_1^*(x, y) = (-4.7, y/4 + 17.2).$$

Using this map in place of w_1 enables us to partially improve the GA for this IFS - the results are shown in Table 8.5.

	Original GA	Improved GA
Pts. Calculated	4372	3305
Unique Points	1093	1093
Time (seconds)	10.58	8.13

Table 8.5 Results of partially improving the GA for the Leaf.

Thus there are certain IFS maps which, when approximated, cause no change in the attractor. Generally we find that such maps represent the stem of a plant-like IFS (as in the above example) or something similar. In such maps, every point of the attractor is mapped onto a single line, and altering the length of this line often makes no difference to the appearance of the attractor.

We have shown that approximation can be very helpful in changing maps with irregular jigsaws into similar maps which have (semi)-regular jigsaws. However, in general, approximation can only work if the IFS code contains at least one zero entry, ruling out maps which include a rotation - although as we shall see shortly, there are exceptions to this rule. In particular, although rational approximation can be used to find the nearest small rational to a given irrational number, and can therefore help us to approximate maps with irrational code entries, any such

approximation may lead to fundamental changes in the appearance of the attractor and is therefore of no use to us. We now move on to discuss why such problems arise.

8.3 Limitations of Approximation

When we approximate an IFS map, our main concern is that the appearance of the attractor will not be altered beyond some acceptable level - which we can ensure by making only small changes to the code values [2]. However, ensuring that these changes are small enough requires us to use rational approximations with relatively large denominators (which do not exceed 20). As we noted earlier, this does not cause problems if our map is simply coordinate scaling as we can use the best approximation satisfying the limit on its denominator (as in Example 8.5). In the case of coordinate scaling followed by shear along the x -axis with parameter α , we can use the best allowable approximation horizontally, but may have to use less accurate approximations vertically in order to ensure that $\text{lcm}(b_2, d_2) < 20$ (the case of shear along the y -axis is similar). However, we should still be able to produce a good approximation to the attractor in the majority of cases since we can use the best approximation in at least one direction.

Suppose, however, that an IFS map consists of coordinate scaling, shear (which without loss of generality may be taken to be along the x -axis) and a rotation by some angle $\theta \neq k\pi/2$ ($k \in \mathbf{Z}$), so that $\cos\theta$ and $\sin\theta$ are both non-zero. Then the map has matrix

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & \alpha \\ 0 & 1 \end{bmatrix} \begin{bmatrix} r_1 & 0 \\ 0 & r_2 \end{bmatrix} = \begin{bmatrix} r_1 \cos\theta & r_2(\alpha \cos\theta - \sin\theta) \\ r_1 \sin\theta & r_2(\alpha \sin\theta + \cos\theta) \end{bmatrix} \quad (8.4)$$

where r_1, r_2 are the horizontal and vertical scale factors, and α is the shear parameter. Now, since $\cos\theta$ and $\sin\theta$ are both non-zero, each entry of (8.4) is non-zero and thus each entry must be approximated in order that the map becomes semi-regular. Note that if $\alpha = 0$ then no shear is present in the map, and the map consists only of scaling followed by rotation - we will now consider this case in greater detail, with the case where shear is present being similar. Thus we have the matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} r_1 \cos\theta & -r_2 \sin\theta \\ r_1 \sin\theta & r_2 \cos\theta \end{bmatrix}$$

Recall from (2.7) that, for such a matrix, we have

$$r_1 = \sqrt{a^2 + c^2} \quad r_2 = \sqrt{b^2 + d^2}$$

$$\theta = \arccos\left(\frac{a}{\sqrt{a^2 + c^2}}\right)$$

Clearly, approximating the matrix entries will change r_1 and r_2 accordingly but, as with the case of simple coordinate scaling, the only effect this has is to change the dimensions of the part of the attractor which the map sends points to. However, in such maps, approximating the matrix entries also alters the angle of rotation. If the changes are small enough, then this alteration will not create a problem - but if the changes are too large then the alteration to the angle of rotation will cause fundamental changes in the appearance of the attractor, and approximation will clearly not be feasible. Now, since we require the lowest common multiples of both a_2 and c_2 , and b_2 and d_2 , to be less than 20, we are usually forced to use less accurate approximations than we would like - and the angle of rotation will be changed beyond an acceptable level. As an example of this, we consider again the 4 Fold Crystal and look more closely at map 4.

Example 8.8 Recall that map 4 of the 4 Fold Crystal is given by

$$w_4(x, y) = (0.37x - 0.642y + 63.6, 0.642x + 0.37y - 0.6)$$

Then, by (2.7), we have

$$r = s = \sqrt{(0.37^2 + 0.642^2)} \approx 0.741 \text{ and}$$

$$\theta = \arccos(0.37/0.741) = \pi/3$$

so that the map consists of uniform scaling with factor 0.741, followed by rotation by $\pi/3$. Rational approximation gives four conveniently small options for each of 0.37 and 0.642, namely $1/2$, $1/3$, $3/8$ and $7/19$ for 0.37 and $1/2$, $2/3$, $7/11$ and $9/14$ for 0.642. However, since the dimensions of the building block must be small, we need the lowest common multiple of the two denominators to be less than or equal to 20. Thus we must take the second approximation for each coefficient, which in turn gives a 3×3 building block. But then,

$$r = s = \sqrt{(1/9 + 4/9)} = \sqrt{5}/3 \approx 0.745 \text{ and}$$

$$\theta = \arccos(1/\sqrt{5}) = 63.48^\circ$$

and so the angle of rotation is no longer equal to $\pi/3$. To see how this changes the attractor, consider Figure 8.7 which shows the attractor with approximated versions of maps 1, 2 and 3 (see Example 8.1) and the attractor with all four maps approximated.

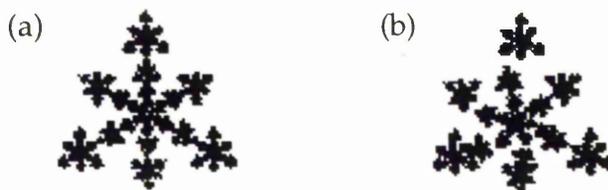


Figure 8.7 The attractor of the 4 Fold Crystal using (a) approximations to maps 1, 2 and 3 only and (b) approximations to all four maps.

It is easy to see that changing the code entries in this way has altered the whole appearance of the attractor - notice in particular that attractor (a) is connected, while (b) contains a gap - and it is clear, therefore, that using approximation is not feasible in this case.

Note A similar problem arises in maps 3 and 4 of the Von Koch Curve which both involve rotation by $\pi/3$ followed by uniform scaling by $1/3$. As in our example, approximation is not feasible for either of these maps since in order to find rationals which give little or no change in the angle of rotation, we must forfeit our wish to have small building blocks.

The main problem in Example 8.8 is that, in order to guarantee that the building block will be small, we need to take far less accurate approximations than we would like. Although this will be true for the majority of IFS maps which include rotation, there are certain cases where $\cos\theta$ is an integer multiple of $\sin\theta$ (or vice versa) - and in such cases approximation may be possible. In particular, when $\theta = \pi/4$, we have $\cos\theta = \sin\theta$ so that the four matrix entries will be equal in size, and taking any rational approximation to the original code values will affect the scaling, but not the angle of rotation - as our next example illustrates.

Example 8.9 The *Tree 3* IFS consists of four maps, given at scale 200 by

$$w_1(x, y) = (0, 0.5y)$$

$$w_2(x, y) = (0.42x - 0.42y, 0.42x + 0.42y + 40)$$

$$w_3(x, y) = (0.42x + 0.42y, -0.42x + 0.42y + 40)$$

$$w_4(x, y) = (0.1x, 0.1y + 40)$$

It is easy to see that maps 1 and 4 are both regular, with the jigsaw pieces for map 1 being infinite horizontal bands of height 2, while those for map 4 are 10×10 squares of pixels, and so we can deal with both maps using the methods of Chapter 5 - the results of doing so are shown in Table 8.6.

	Original GA	Improved GA
Pts. Calculated	5444	2819
Unique Points	1361	1361
Time (seconds)	14.28	9.98

Table 8.6 Results of using jigsaws for maps 1 and 4 of the Tree 3.

Since each of the matrix entries for maps 2 and 3 are equal in size, we may use the best rational approximation with denominator $q \leq 20$ in place of 0.42 in order to obtain a building block of dimensions $q \times q$. Using (8.1) and (8.2), we find that there are five convenient options, namely $1/2$, $2/5$, $3/7$, $5/12$ and $8/19$ - the attractors obtained with each option except the first are shown in Figure 8.8.

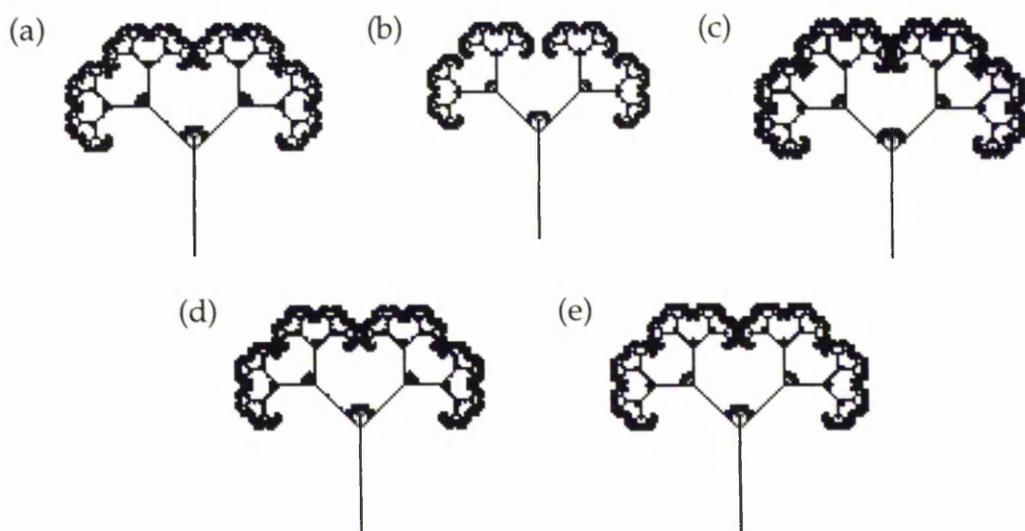


Figure 8.8 The attractors of the Tree 3 IFS using

(a) 0.42, (b) 0.4, (c) $3/7$, (d) $5/12$ and (e) $8/19$.

Thus we can achieve a reasonable approximation to the attractor using any one of the four small rationals, noting that although (b) is clearly smaller than the original attractor, this can be rectified by altering the scale slightly. In fact, in order to make the reduction scheme as simple as possible, it is preferable to use either $2/5$ or $3/7$ as our approximation. Figure 8.9 shows a small section of the jigsaws obtained for map 2 in each case, with a building block shaded. Recall that these are not the only building blocks - any pixel may be taken as the lowest left pixel of the block. Note also that the jigsaw for map 3 is identical to that of map 2, and therefore we may consider the two maps together.

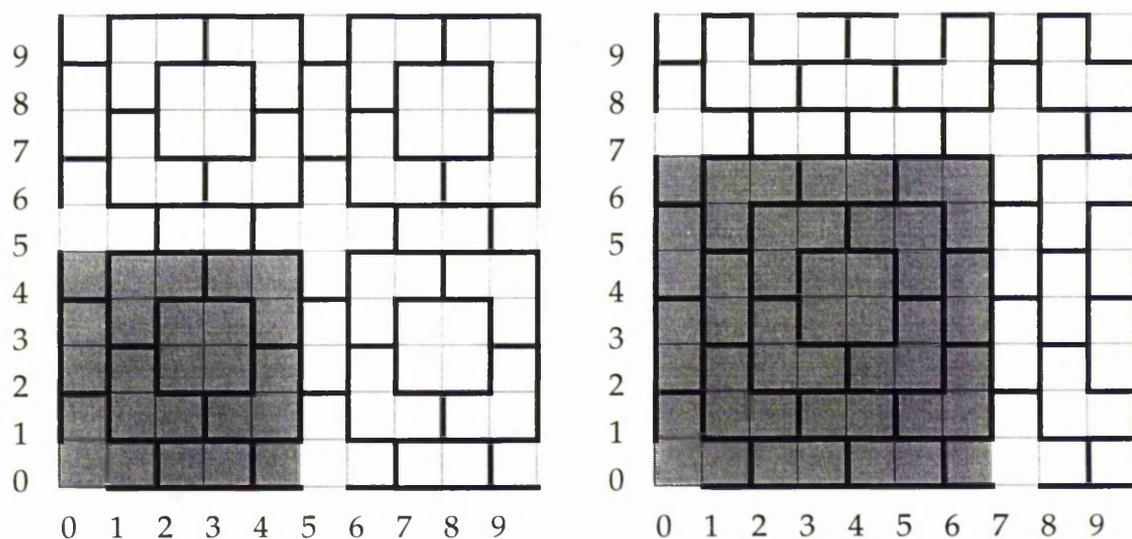


Figure 8.9 Part of the jigsaws obtained when we approximate map 2 of the Tree 3 IFS.

The jigsaw on the left shows approximation by $2/5$ while that on the right is by $3/7$.

In order to make use of building blocks in this case, we must first find a suitable reduction scheme. Suppose that the building block has dimensions $m \times n$, and that the smallest horizontal and vertical values in the attractor are given by x_{\min} and y_{\min} respectively. Then the general reduction scheme is given by

$$\begin{aligned} x &\rightarrow m((x - x_{\min}) \operatorname{div} m) + x\text{shift} \\ y &\rightarrow n((y - y_{\min}) \operatorname{div} n) + y\text{shift} \end{aligned} \quad (8.5)$$

where $xshift$ and $yshift$ are given by the following algorithm.

Calculate $\underline{w}(P)$ for each jigsaw piece P in the lowest left building block of the jigsaw.

Find the smallest value of $\underline{w}_h(P)$, and set $xshift$ to 1 for all pixels of P .

Similarly for $yshift$.

If $\underline{w}(P') = \underline{w}(P) + (i, j)$ for a jigsaw piece $P' \neq P$ in this building block, set $xshift$ and $yshift$ so that all pixels of P' will be sent to $(1 + i, 1 + j)$.

Repeat for each P' .

For an example of how we set $xshift$ and $yshift$ at step 3 of the above algorithm, see Figure 8.11.

Note It is possible that some of the jigsaw pieces will cross the boundaries of building blocks. Such pieces will be split into two separate parts by the above algorithm - this can be avoided by making small alterations to $xshift$ or $yshift$, as we will see shortly.

As we saw earlier, when we approximate 0.42 by $2/5$ in map 2 of the Tree 3 IFS, the building block has dimensions 5×5 . Further, we find that $x_{\min} = -42$ and $y_{\min} = 0$, so the reduction scheme is

$$x \rightarrow 5((x + 42) \text{ div } 5) + xshift$$

$$y \rightarrow 5(y \text{ div } 5) + yshift$$

Now, when we calculate $\underline{w}(P)$ for each jigsaw piece P in the building block with lowest left pixel $(-42, 0)$, we find that the shaded jigsaw piece of Figure 8.10 has smallest $\underline{w}_h(P)$ and therefore we want to send this piece to 1 horizontally. Further, $\underline{w}_v(P) = \min\{\underline{w}_v\} + 2$ and so we want to send this piece to 3 vertically. Thus

whenever $x \equiv -2 \pmod{5}$ and $y \equiv 4 \pmod{5}$, or $x \equiv -1 \pmod{5}$ and $y \equiv 3$ or $4 \pmod{5}$, we have

$$x \rightarrow 5((x + 42) \text{ div } 5) + 1$$

$$y \rightarrow 5(y \text{ div } 5) + 3$$

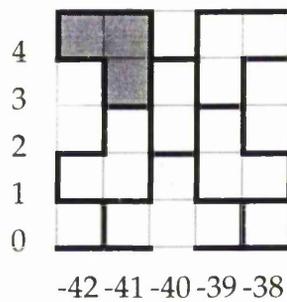


Figure 8.10 The lowest left building block for map 2 of the Tree 3. The shaded piece is sent to $(1, 3)$ in the reduced jigsaw by our general scheme.

It is now a simple process to consider each jigsaw piece $P' \neq P$, and use step 3 of the reduction algorithm to find $xshift$ and $yshift$. However, as we noted earlier, the algorithm may not give the best values for pieces which cross edges of the building block. For example, for $(x, y) = (-38, 0)$ our algorithm gives us $xshift = 4$ and $yshift = 3$ so that $(x, y) \rightarrow (4, 3)$, while for $(x, y) = (-37, 0)$, $xshift = 2$ and $yshift = 1$ so that $(x, y) \rightarrow (7, 1)$ - but we want these two pixels to be in the same jigsaw piece. To ensure this, we alter the shifts for $x \equiv 2 \pmod{5}$, $y \equiv 0 \pmod{5}$ to $xshift = 7$ and $yshift = 1$ - a similar process may be used wherever there is overlap between blocks. Figure 8.11 shows the building block and the shifts required for all values of x, y modulo 5.

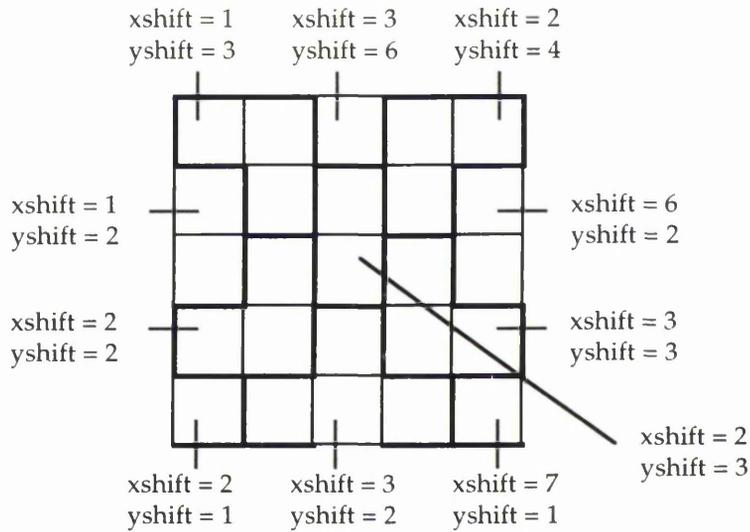


Figure 8.11 The shifts for the map 2 building block.

We are now in a position to apply this reduction scheme and consequently to use the map 2 jigsaw to improve the GA still further for this IFS. Recall that the jigsaw - and therefore the reduction scheme - for map 3 is identical to that of map 2, so that we can use one reduced jigsaw to cover both maps. The results of using jigsaws for all four maps of the Tree 3 to improve the GA are given in Table 8.7, with the results for the Original GA and Improved GA for maps 1 and 4 given for comparison. Note that the number of unique points and number of points calculated have changed from the values given in Table 8.6 - this is due to the changes made to maps 2 and 3.

	Improved GA		
	Original GA	Maps 1, 4	All 4 maps
Pts. Calculated	3964	2063	997
Unique Points	991	991	991
Time (seconds)	9.85	6.65	4.9

Table 8.7 Results of improving the GA for the Tree 3.

We can see from the table that approximating maps 2 and 3, and then making use of all four jigsaws has enabled us to half the time taken to produce the attractor of the Tree 3 IFS. Further, we have done this without altering the appearance of the attractor too much, and thus it is clearly feasible to make such alterations in this case. Note that, although using a better approximation to 0.42 (i.e. $3/7$ or $5/12$) would result in a better approximation to the attractor, the reduction scheme would clearly have many more options. As a consequence, applying the scheme within the GA would take longer and the time reduction would not be as large.

Thus, when we have a map which involves rotation by a multiple of $\pi/4$, approximation allows us to improve the performance of the Graphical Algorithm since no matter how inaccurate the approximation is, the angle of rotation will not be altered, and therefore approximating the map changes only the scale of the part of the attractor that the map sends points to. This fact enables us to add three of our fifty IFS's to the list of those which we can improve - namely the Face, the Grabber and, as we have just seen, the Tree 3.

In this chapter we have seen that when we have a map with an irregular jigsaw it is usually possible to approximate the map so that we may find a small enough building block - which in turn allows us to improve the performance of the GA for the IFS containing this map. In fact, every one of our fifty IFS's which do not have any maps with regular jigsaws has at least one map which can be approximated in this way to give a semi-regular jigsaw - and therefore the methods of Chapters 5 to 8 have enabled us to at least partially improve the performance of the GA for all fifty IFS's, with thirty-six of the fifty being completely improved by some combination of our methods. A full list of the fifty IFS's and which methods are used to enable us to use the jigsaws of their maps may be found in the Appendix. However, before moving on to summarise the results of this thesis, we return to

the first IFS which we mentioned - the Face of Example 2.19 - which highlights once more the problems which certain irregular jigsaws can cause.

Example 8.10 Recall that the Face has eleven maps - these are given in Table 2.1, with the attractor being shown in Figure 2.4(c). It is easy to see that each of the first seven maps have semi-regular jigsaws which consist of bands of infinite length. Thus, in each case, the horizontal reduction simply sends the horizontal coordinate to 1, while the vertical reduction scheme can be found using (6.5).

The remaining four maps of the IFS consist of rotation by angles which are not multiples of $\pi/4$ followed by scaling and shear, and therefore we cannot approximate the maps in order to find suitably small building blocks without altering the appearance of the attractor. Further, the majority of the jigsaw pieces are large - as Figure 8.12 shows for map 8 - and, since the attractor has only 836 unique points, each will contain only a few attractor points. Thus the ScanRows procedure defined in Chapter 7 is unsuitable in this case, and therefore we can do nothing with these four maps at present.

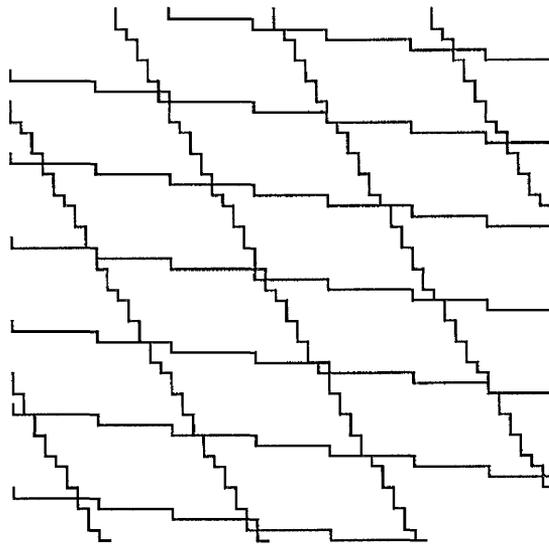


Figure 8.12 Part of the jigsaw for map 8 of the Face IFS.

Note that the pieces are similar, but *not* the same.

Notice, however, that while we cannot make use of these jigsaws at present we can use several of the Theorems of Chapter 4 to say more about the pieces. By Corollary 4.23 we know that the jigsaws of maps 8, 10 and 11 contain no rising bridges, while that of map 9 contains no descending bridges. Further, none of the jigsaws contain gaps; in each case $abcd > 0$ so Theorem 4.31 does not rule out both types of gap, but Theorem 4.36 says that if gaps exist the map will have contraction ratio greater than 0.8 - and the four maps in question have ratios 0.14, 0.13, 0.16 and 0.34 respectively.

Finally, although we cannot completely improve the performance of the GA for this IFS, we can partially improve it by using the jigsaws of the first seven maps - doing so reduces the number of points calculated from 9196 to 3826 and the time taken from 20.52 seconds to 9.28 seconds, a reduction of over 50%.

Chapter 9

Discussion

We end this thesis with a brief discussion of its results, and give three topics for future research. However, before doing so, we return to the subject of jigsaw pieces which have bridges or gaps and give some further insight into how we can deal with such pieces.

9.1 Dealing with Bridges and Gaps

As we saw, the conditions given in Corollary 4.26 rule out the possibility of jigsaw pieces containing bridges in all but seven of the maps of our fifty IFS's - these 'problem' maps were given in Table 4.1. By considering the codes we see that we must use approximation to enable us to find suitable building blocks for each of the first five maps, while the remaining two maps - maps 1 and 3 of the Triangle Crab - are semi-regular.

For map 2 of the Comet's Tail and the two maps of the Sierpinski Peak, only one of the code entries - namely ± 0.506 - must be approximated. This is done by changing ± 0.506 to ± 0.5 - using this approximation gives a new map for which both conditions of Corollary 4.26 are satisfied, and therefore we may disregard bridges in these cases. Further, the two maps of the Leaf Outline contain a rotation by an angle which is not a multiple of $\pi/4$ and so, as we saw in Chapter 8, we cannot make use of their jigsaws to improve the Graphical Algorithm.

We are therefore left with the two maps of the Triangle Crab, which have 10×10 building blocks as their smallest. The building block for map 1 is shown in Figure 9.1, where the small diagonal lines represent the bridges.

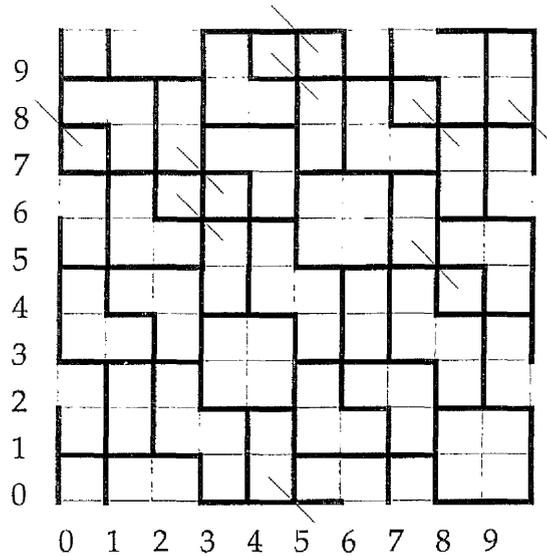


Figure 9.1 A building block for map 1 of the Triangle Crab, showing bridges.

Since the jigsaw pieces are of many different shapes, the reduction scheme for this map will clearly require consideration of values of x and y modulo 10, in much the same way as in the scheme of Example 8.9. Thus it is a simple process to ensure that the two parts of pieces containing a bridge are sent to the same reduced point. This is clearly also the case for map 3, and therefore the presence of bridges in the jigsaws of these maps make no difference to our ability to use the jigsaws.

In general, it is likely that any map whose jigsaw pieces contain bridges will be such that the jigsaw is made up of a number of different shapes of piece which do not fit together in any obvious pattern - as in the above figure - and therefore the bridges will make no difference to the way that we deal with the jigsaw. However, it is possible that the building block will exhibit a pattern such that, by ignoring the bridges (i.e. treating them as being in different pieces), we will be able to find a simpler reduction scheme. In such cases we may disregard the bridges - the only consequence of this being that we will have a greater amount of duplication of points, since each point which is the result of a piece containing a bridge will be produced twice. This is, however, justified as we require as simple a reduction

scheme as possible to allow optimal time improvement. Note that this also applies to gaps in pieces - that is, we may ignore gaps if doing so enables us to find a simpler reduction scheme and consequently achieve optimal time improvement.

We are now in a position to summarise our results.

9.2 Summary and Further Research

In this thesis we considered a particular method of screening the attractor of an Iterated Function System - the Graphical Algorithm - and showed that the number of points calculated by this algorithm is always N times the number of attractor points. We explained why this happens and showed that, for every IFS map, we can produce a jigsaw whose pieces consist of pixels which are sent to the same point under application of the map. We considered the various properties which jigsaw pieces can have, and gave conditions on the entries of the IFS code which are sufficient in order for each property to hold. We then showed that we can use jigsaws to ensure that each attractor point is hit only once by a particular map - which in turn speeds up the GA - and also showed that we often require less than N jigsaws to handle N maps, giving further improvement in the time taken to produce the attractor.

We introduced the three different types of jigsaw which can occur - regular, semi-regular and irregular - showing that, in the first two cases, we can use the jigsaws as they are, provided that, if the jigsaw is semi-regular, we can find a suitably small building block. In the irregular case, we considered two ways of finding the jigsaw pieces and showed that one such method can be used to give considerable time savings in IFS's whose jigsaw pieces contain a relatively large number of attractor points. We also saw that this method produced better results than those previously achieved for certain IFS's with (semi)-regular jigsaws.

For cases where this method was not applicable, we moved on to look at various ways of approximating irregular maps in order to find a map with a (semi)-regular jigsaw which does not alter the attractor of the IFS beyond some acceptable level. We showed that approximation is feasible when at least one entry of the IFS code is equal to zero, and also in cases where the map consists of coordinate scaling followed by rotation by a multiple of $\pi/4$, but not in cases where the rotation angle is not a multiple of $\pi/4$. Where approximation was feasible, we showed that the methods described earlier could be applied to the new map, resulting in improvement in the time taken to produce the attractor by the GA.

Finally, we noted that, by applying relevant methods, we are able to at least partially improve the performance of the Graphical Algorithm for each of our fifty IFS's - further details are given in the Appendix.

Looking to the future, it may be possible to further improve the performance of the Graphical Algorithm in cases where the jigsaws are either already semi-regular, or can be made semi-regular by approximation, by altering the way in which new points are calculated. Currently, we calculate the reference point in the reduced jigsaw using the computer operations `div` and `mod`, and then calculate the point to be plotted by applying the map. However, this essentially means that we calculate a new point twice - when in fact once may suffice. As an example, we consider map 1 of the Sierpinski Gasket, given by $w(x, y) = (0.5x, 0.5y)$. As we saw in Chapter 5, when a point (x, y) is chosen from the store, we check to see if $((x + 1) \text{ div } 2, (y + 1) \text{ div } 2)$ is set in the reduced jigsaw and apply w if and only if it is false. However, the reference point is set if and only if the point $(0.5x, 0.5y) \equiv (x \text{ div } 2, y \text{ div } 2)$ is lit in the attractor - and thus we could simply check if $(x \text{ div } 2, y \text{ div } 2)$ is lit, lighting it if it isn't and moving onto the next map if it is.

This method would clearly reduce the amount of calculation required, and also remove the need to store the jigsaws separately from the attractor - which we would expect to lead to further time savings. Note, however, that in most cases finding the equivalent map which uses only div and mod will not be such a simple process. Thus, although this method has potential, it requires more research.

It is also hoped that a method could be found which would allow us to make use of those irregular jigsaws which cannot currently be dealt with - perhaps by developing the ScanRows procedure described in Chapter 7. Such a method would enable us to use jigsaws for each of the N maps of any Iterated Function System in order to improve the performance of the Graphical Algorithm. However, as we saw in Chapters 7 and 8, this may prove to be extremely challenging.

Ultimately, we would like to develop a program which uses only the IFS code to determine how to improve the performance of the GA for a given IFS. Such a program would need to

1. Determine how many distinct jigsaws are required, grouping together those maps whose jigsaws are identical.
2. Make use of the Theorems of Chapter 4 to determine if the pieces are rectangular, or if they contain bridges and/or gaps.
3. Find the dimensions of the smallest building block and decide if the jigsaw is (semi)-regular based on these dimensions.
4. Find suitable reduction schemes for (semi)-regular jigsaws.
5. Determine whether approximation is feasible for any jigsaws of the IFS which are irregular and find suitable approximations in such cases.

Clearly the processes described would need to be performed as quickly as possible, and it seems likely that, for IFS's such as those described in this thesis whose attractors contain few points, this integrated program would be essentially slower than the original Graphical Algorithm. However, in practical applications, IFS's consist of hundreds of maps and their attractors contain many thousands of points. In such cases the original GA would appear to be impractical - but an improved version making use of all knowledge about jigsaws may be able to produce approximations to the attractor in a fraction of the original time.

Appendix

Below we give the codes of our fifty IFS's, together with which of our methods apply to which map. All values in the codes should be divided by 1000 to obtain the code entries correct to three decimal places, although wherever possible we use fractions in place of their decimal equivalents (e.g. those given by 333 and 667 represent $1/3$ and $2/3$ respectively and so on). Further, in the codes of maps 3 and 4 of the Von Koch Curve, we use an approximation to $\sqrt{3}$ to allow us to give b and c in this form - in practice we revert to using $\sqrt{3}/6$ in place of 0.289.

Following each map we give a code which states which of our methods may be used to improve the performance of the Graphical Algorithm for this map. The key is as follows:

R = regular SR = semi regular A = by approximation

while a dash means that the map cannot be reasonably improved by our methods.

Thus a map marked SRA can be made semi regular by approximation and so on.

The Codes of the Fifty Iterated Function Systems

3D Wire					
500 500 0 500 0 0	R	0 0 0 160 0 0	SRA	500 500 -500 500 0 0	SR
500 0 0 500 -500 500	R	849 37 -37 849 0 1600	-	-500 500 -500 -500 1000 0	SR
500 0 0 500 500 -500	R	197 -257 226 223 0 1600	-	Eiffel Tower	
4 Fold Crystal		-150 283 260 238 0 440	-	500 0 0 300 0 0	SR
255 0 0 255 373 671	SRA	Barnsley's Frizzy	SR	500 0 0 300 500 0	SR
255 0 0 255 115 223	SRA	500 0 0 750 640 0	SR	500 0 0 700 250 300	SR
255 0 0 255 631 223	SRA	260 -500 230 570 1630 100	-	Face	
37 -642 642 37 636 -6	-	260 500 -230 570 280 690	-	460 460 -460 460 300 300	SRA
5 Fold Crystal		Castle		460 -460 460 460 -300 300	SRA
382 0 0 382 307 619	SRA	500 0 0 500 0 0	R	500 0 0 100 0 -150	R
382 0 0 382 603 404	SRA	500 0 0 500 500 0	R	Fish	
382 0 0 382 14 404	SRA	500 0 0 500 0 500	R	525 99 0 734 -65 127	SRA
382 0 0 382 125 60	SRA	400 0 0 400 600 500	SR	408 119 -144 396 352 532	-
382 0 0 382 492 60	SRA	Comet's Tail		268 0 0 379 345 125	SRA
Alt. Square		594 0 0 594 444 -39	SRA	273 0 0 379 491 307	SRA
450 0 0 450 0 0	SR	500 -506 0 500 493 554	SRA	322 -98 273 216 546 5	-
450 0 0 450 0 500	SR	500 0 -492 500 -46 491	SRA	97 220 -107 261 714 469	-
450 0 0 450 500 0	SR	Crab		98 222 88 -247 728 444	-
450 0 0 450 500 500	SR	500 200 0 500 0 0	SR	-357 -205 92 -178 618 150	-
		-500 0 200 500 1000 0	SR	17 423 -267 -14 203 967	-
		500 200 0 -500 0 1000	SR		
		-500 0 -200 -500 1000 1000	SR		

.../The Codes of the Fifty IFS's Continued.

Flamboyant Crown		Peano Curve		Square Plant	
-250 0 0 500 0 0	R	333 0 0 333 0 0	R	667 0 0 667 0 0	SR
500 0 0 500 -250 500	R	0 -333 333 0 333 0	R	300 0 0 300 0 700	SR
-250 0 0 -250 250 1000	R	333 0 0 333 333 333	R	300 0 0 300 700 0	SR
500 0 0 500 0 750	R	0 333 -333 0 667 333	R	300 0 0 300 700 700	SR
500 0 0 -250 500 1250	R	-333 0 0 -333 667 0	R	Square Snowflake	
Girders		0 333 -333 0 333 0	R	-500 0 0 -500 1000 1000	R
333 0 0 333 0 0	R	333 0 0 333 333 -333	R	500 0 0 -500 0 1000	R
333 0 0 333 0 333	R	0 -333 333 0 667 -333	R	-500 0 0 500 1000 0	R
333 0 0 333 0 667	R	333 0 0 333 667 0	R	Takagi Function	
333 0 0 333 333 333	R	Pine Tree		500 0 500 500 0 0	R
333 0 0 333 667 0	R	798 0 0 833 98 169	SRA	500 0 -500 500 500 500	R
333 0 0 333 667 333	R	-132 -367 399 -141 500 181	-	Tree 1	
333 0 0 333 667 667	R	100 404 408 -100 365 158	-	195 -488 344 443 443 245	-
Grabber		49 0 0 333 399 10	SRA	462 414 -252 361 251 569	-
460 460 -460 460 300 300	SRA	Pseudo Plant		-58 -70 453 -111 598 97	-
460 -460 460 460 -300 300	SRA	805 40 -42 766 99 2380	-	-35 -70 -469 -22 488 507	-
Hex		226 -244 0 549 187 657	SRA	-637 0 0 501 856 251	SRA
400 0 0 400 0 300	SR	-260 281 0 447 149 793	SRA	Tree 2	
400 0 0 400 520 0	SR	182 0 0 500 71 -219	SRA	459 -226 73 602 -2 319	-
400 0 0 400 520 600	SR	Sierpinski Carpet		343 376 -203 546 -22 330	-
400 0 0 400 693 300	SR	333 0 0 333 0 0	R	136 503 -313 138 -20 217	-
400 0 0 400 173 0	SR	333 0 0 333 333 0	R	253 -490 308 350 -7 198	-
400 0 0 400 173 600	SR	333 0 0 333 667 0	R	66 0 0 479 -15 -24	SRA
Kite		333 0 0 333 0 333	R	Tree 3	
333 0 0 500 0 500	R	333 0 0 333 667 333	R	0 0 0 500 0 0	R
333 0 0 500 0 -500	R	333 0 0 333 0 667	R	420 -420 420 420 0 200	SRA
0 -500 333 0 500 0	R	333 0 0 333 333 667	R	420 420 -420 420 0 200	SRA
0 -500 333 0 -500 0	R	333 0 0 333 667 667	R	100 0 0 100 0 200	R
Koch Snowflake		Sierpinski Gasket		Tree Trunk	
333 0 0 333 0 667	R	500 0 0 500 0 0	R	500 200 0 500 0 0	SR
333 0 0 333 0 -667	R	500 0 0 500 500 0	R	500 -200 0 500 500 0	SR
333 0 0 333 577 333	R	500 0 0 500 0 500	R	500 -250 0 500 250 500	SR
333 0 0 333 -577 333	R	Sierpinski Peak		500 250 0 500 250 500	SR
333 0 0 333 577 -333	R	500 0 506 500 -48 -306	RA	Triangle Crab	
333 0 0 333 -577 -333	R	500 0 -506 500 559 200	RA	700 200 200 500 0 0	SR
Leaf		500 0 0 500 269 557	R	500 0 0 500 600 250	R
0 0 0 278 -47 172	RA	198 0 0 244 402 -75	SRA	700 200 -200 -500 0 1000	SR
525 20 6 786 -17 2170	-	Sierpinski Twist		Twin Xmas Tree	
201 -241 385 365 49 1620	-	0 500 -500 0 0 500	R	0 -500 500 0 500 0	R
-142 235 377 393 -191 1300	-	500 0 0 500 500 0	R	0 500 -500 0 500 500	R
Leaf Outline		500 0 0 500 500 500	R	500 0 0 500 250 500	R
500 0 0 500 250 500	R	Sleepy Hollow		Von Koch Curve	
176 250 -700 433 547 500	-	500 0 250 500 0 0	SR	333 0 0 333 0 0	R
176 -250 700 433 284 -187	-	500 0 250 500 500 250	SR	333 0 0 333 667 0	R
Lévy Curve		500 0 -250 500 500 250	SR	167 -289 289 167 333 0	-
500 -500 500 500 0 0	SR	500 0 -250 500 0 500	SR	-167 289 289 167 667 0	-
500 500 -500 500 500 500	SR	Spider's Web		Wheat	
Line		500 0 0 -500 250 500	R	500 0 0 500 400 400	R
500 0 0 500 5050 5050	R	500 492 0 500 250 500	SRA	500 0 500 500 100 0	R
500 0 0 500 -50 -50	R	500 -492 0 500 250 500	SRA	500 500 0 500 0 100	R
Maple Leaf		Square		625 20 624 20 1 1	-
500 0 0 759 0 542	SRA	500 0 0 500 0 0	R	Whirl	
1 0 0 272 0 0	SRA	500 0 0 500 500 0	R	0 -500 500 0 500 0	R
438 398 -87 492 -2 428	-	500 0 0 500 0 500	R	0 500 -500 0 500 500	R
-438 -398 -87 492 -4 434	-	500 0 0 500 500 500	R	500 -500 0 500 500 500	R
Mutant		Square Flake		Wire	
500 500 500 0 0 0	R	333 0 0 333 0 0	R	500 250 0 500 0 0	SR
0 500 500 500 0 0	R	333 0 0 333 667 0	R	500 -250 0 500 500 0	SR
500 -500 -500 0 500 500	R	333 0 0 333 667 667	R	227 -146 0 492 400 504	SRA
0 -500 -500 500 1000 500	R	333 0 0 333 0 667	R	224 152 0 500 414 504	SRA
Part of a Crystal		333 0 0 333 333 333	R		
-500 0 0 -500 1600 1600	R				
0 500 500 0 0 0	R				
500 0 0 500 0 800	R				

References

- [1] Barnsley M. F. and Demko, S. (1985), Iterated Function Systems and the Global Construction of Fractals, *Proc. of the Royal Soc. of London A399*, pp. 243 - 275.
- [2] Barnsley, M. F. (1993), *Fractals Everywhere (2nd Edition)*, Academic Press.
- [3] Barnsley, M. F. and Hurd, L. P. (1993), *Fractal Image Compression*, A. K. Peters Ltd.
- [4] Brammer, R. F. (1989), Unified Image Computing Based on Fractals and Chaos Model Techniques, *Optical Engineering* 28, pp. 726 - 734.
- [5] Dubuc, S. and Elqortobi, A. (1990), Approximations of Fractal Sets, *Journal of Computational and Applied Mathematics* 29, pp. 79 - 89.
- [6] Dubuc, S. and Hamzaoui, R. (1994), On the Diameter of the Attractor of an IFS, *C. R. Math. Rep. Acad. Sci. Canada* 16, pp. 85 - 90.
- [7] Edgar, G. A. (1990), *Measure, Topology and Fractal Geometry*, Springer-Verlag.
- [8] Falconer, K. (1990), *Fractal Geometry: Mathematical Foundations and Applications*, Wiley.
- [9] Fisher, Y. (1995), *Fractal Image Compression*, Springer-Verlag.

References

- [10] Goodman, G. S. (1991), A Probabilist Looks at the Chaos Game, *Fractals in the Fundamental and Applied Sciences - Proc. of the 1st IFIP Conference on Fractals, Lisbon 1990*, Elsevier, Amsterdam, pp. 159 - 168.

- [11] Hepting, D. and Prusinkiewich, P. (1991), Rendering Methods for Iterated Function Systems, *Fractals in the Fundamental and Applied Sciences - Proc. of the 1st IFIP Conference on Fractals, Lisbon 1990*, Elsevier, Amsterdam, pp. 183 - 224.

- [12] Hoggar, S. G. (1992), *Mathematics for Computer Graphics*, Cambridge University Press.

- [13] Hoggar, S. G. and McFarlane, I. (1994), Faster Fractal Pictures by Finite Fields and Far Rings, Proc. of 14th British Combinatorial Conference, Keele 1993, *Discrete Mathematics* 138, pp. 267 - 280.

- [14] Hoggar, S. G. and McFarlane, I. (1994), Optimal Sequences for Non Uniform Iterated Function Systems, *Bulletin of the Institute of Combinatorics and its Applications* 12, pp. 65 - 92.

- [15] Hoggar, S. G. and Menzies, L. (1994), Jigsaws and Faster Fractal Pictures, *Proc. of IMA Conference on Image Processing, Cranfield University 1994*, Oxford University Press.

- [16] Hoggar, S. G. and Menzies, L. (1997), Fractal Compression and the Jigsaw Property, submitted.

References

- [17] Hutchison, J. E. (1981), *Fractals and Self Similarity*, *Indiana Univ. Math. Journal* 30, pp. 713 - 747.
- [18] Knuth, D. E. (1981), *The Art of Computer Programming Volume 2 - Seminumerical Algorithms (2nd Edition)*, Addison-Wesley.
- [19] Lee, K. D. and Cohen, Y. (1990), *Fractal Attraction - An Image Producing Package for the Macintosh*, Sandpiper Software.
- [20] McFarlane, I. (1993), *Faster Fractal Pictures Using Optimal Sequences*, Glasgow University PhD Thesis.
- [21] McFarlane, I. and Hoggar, S. G. (1994), *Optimal Drivers for the Random Iteration Algorithm*, *The Computer Journal* 37, pp. 629 - 640.
- [22] McFarlane, I. and Hoggar, S. G. (1995), *Combinatorics for Faster Fractal Pictures*, *Proc. of Conference on Number Theoretic and Algebraic Methods in Computer Science, Moscow 1993*, pp. 95 - 124, World Scientific Publishing.
- [23] Monro, D. M., Dudbridge, F. and Wilson, A. (1990), *Deterministic Rendering of Self-Affine Fractals*, *IEEE Computer Graphics and Applications*, January 1995, pp. 32 - 41.
- [24] Niven, I. and Zuckerman, H. S. (1980), *An Introduction to the Theory of Numbers (4th Edition)*, Wiley.

References

- [25] Peitgen, H. O., Jurgens, H. and Saupe, D. (1988), *Fractals for the Classroom*, Springer-Verlag.

- [26] Peitgen, H. O., Jurgens, H. and Saupe, D. (1990), *Chaos and Fractals*, Springer-Verlag.

- [27] Shonkwiler, R. (1989), An Image Algorithm for Computing the Hausdorff Distance Efficiently in Linear Time, *Info. Proc. Letters* 30, pp. 87 - 89.

- [28] Wicks, K. R. (1991), *Fractals and Hyperspaces, Lecture Notes in Mathematics* 1492, Springer-Verlag.

