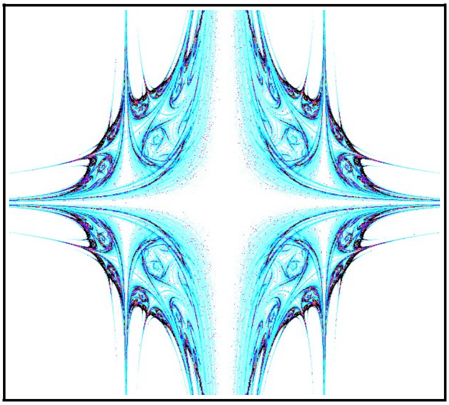


Fractals

and how to
draw them

Oliver Linton

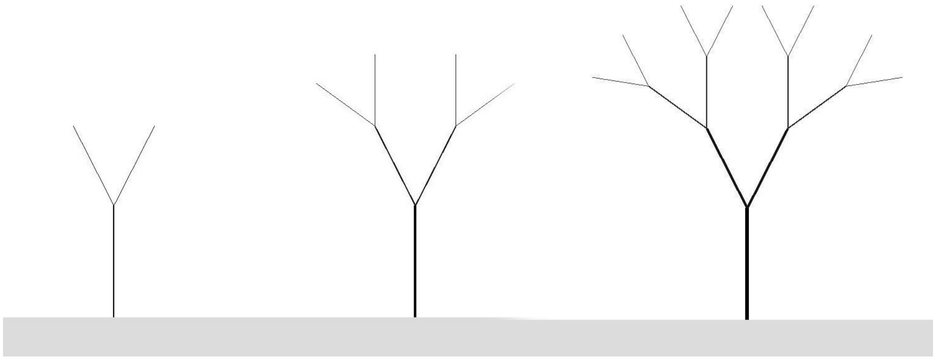
Contents

Recursion

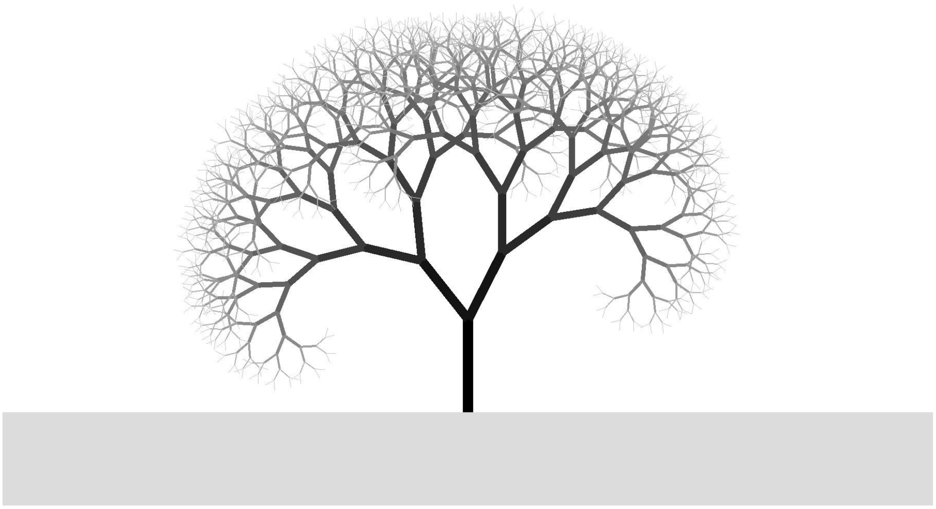
Many fractals have a recursive structure. What this means is that each part is defined in terms of the whole. This is in contrast to most structures in which the whole is defined in terms of its parts. For example, a car is made up of several distinct parts – an engine, a chassis, a transmission system etc. etc.; and each part is defined in terms of smaller and smaller unique parts and so on.

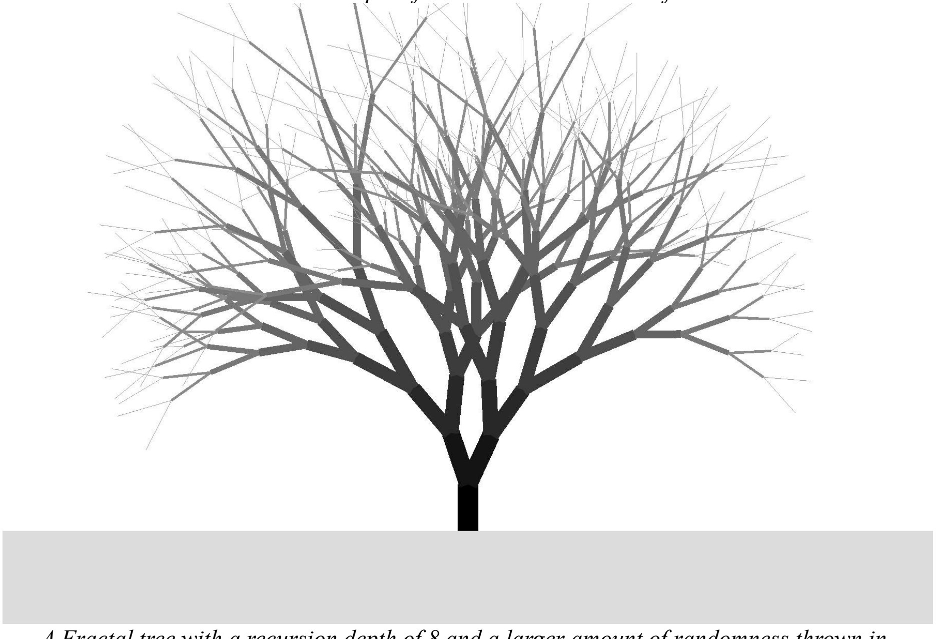
A tree does not have to be defined like this. To a large extent you can define a tree as a system of branches, each of which is a smaller tree. All you have to do is define exactly what a 'branch' is. This will do: *a 'branch' is a length of wood which starts at a 'fork' and ends at a second 'fork' from which two smaller 'branches' sprout at certain angles*. Finally we have to define the first 'branch' like this: *the first branch starts on the ground and is vertical.*

The precise shape of the resulting tree is determined by the following parameters: a) the factors R_1 and R_2 by which each branch is smaller than the branch from which it sprouts and c) the angles A_1 and A_2 between the old branch and the new.



The first three stages in the growth of a fractal tree using the same ratios and angles for the two branches

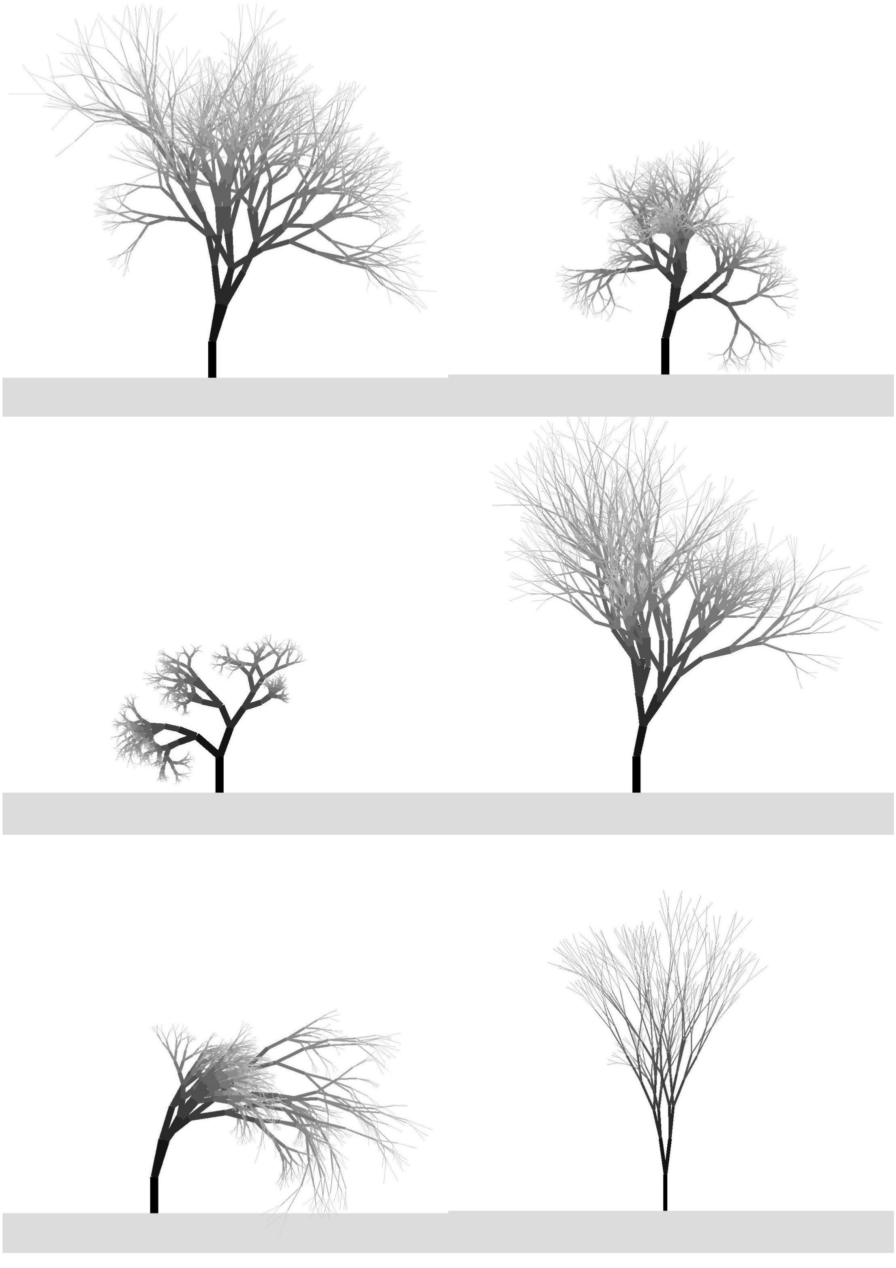

A Fractal tree with a recursion depth of 10 and a small amount of randomness thrown in


A Fractal tree with a recursion depth of 8 and a larger amount of randomness thrown in.

Fractal Tree Program

```
DIM: depth AS INTEGER = 4                                  recursion depth
DIM: $R_1$ , $R_2$ AS FLOAT = { … }                        ratios of each branch
DIM: $A_1$ , $A_2$ AS FLOAT = { … }                        angles of each branch

CALL: Branch(depth, 0, 0, 100, 90)                         call subroutine with initial stem

DEFINE: Branch(d, x, y, length, angle)
    IF d = 0 THEN EXIT                                     exit subroutine when depth = zero
    CALL: DrawBranch(x, y, length, angle)                  draw current branch
    x = x + length * COS(angle)                            update x and y coordinates
    y = y + length * SIN(angle)
                                                           call subroutine for each branch at the fork
                                                           using depth – 1, new length and angle
    CALL: Branch(d – 1, x, y, length * $R_1$, angle + $A_1$)
    CALL: Branch(d – 1, x, y, length * $R_2$, angle + $A_2$)
END_DEFINITION

DEFINE: DrawBranch(x, y, length, angle)                    Draws a suitable branch
    ...
END_DEFINITION
```

Above is a fragment of pseudocode which will generate fractal trees like those opposite. You will notice that the subroutine 'Branch' is called from within the same subroutine. This technique is known as recursion. In order for it to work you must make sure of two things. Firstly, there must be a counter which takes note of the depth which has been reached and when this counter reaches zero, the subroutine must exit without calling itself again. Second, the parameters must be called by value rather than by reference. For example the routine must pass only the values of the *x* and *y* parameters, not the parameters themselves. This is so that when the subroutine eventually returns, *x* and *y* will still have the values which they had when the subroutine was called. (Many computer languages will assume this by default but not all do.)



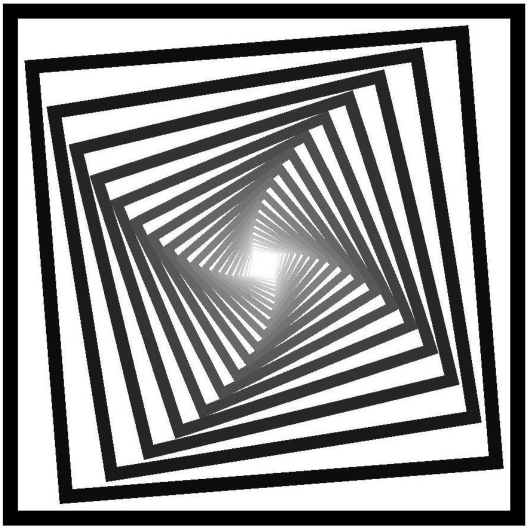
The Droste Effect

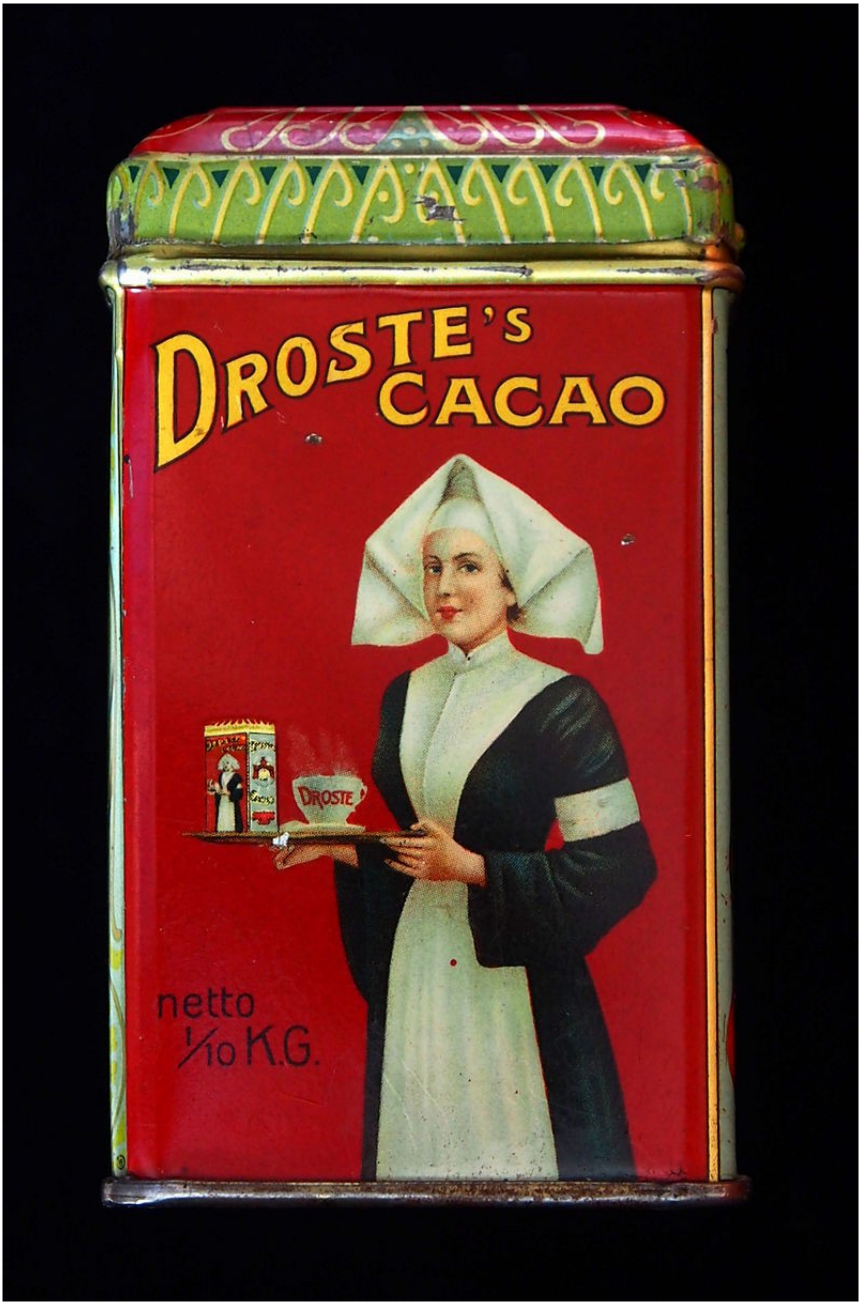
The image opposite appeared in 1904. It shows a Dutch lady holding a packet of Droste's cacao on which is printed an image of a Dutch lady holding a packet of Droste's cacao on which …

This infinite regress will also be familiar with those who have experimented with those old-fashioned ladies dressing tables with two side mirrors which could be positioned approximately parallel with each other.

Yet another method is to point a video camera at its own TV monitor. The time delay introduced by the electronic circuitry can generate fascinating swirling patterns which can persist for several seconds.

A modification of the preceding code can produce images like the one below where a routine to draw a simple picture frame includes a reference to itself, slightly smaller and slightly rotated




DROSTE'S CACAO
DROSTE
netto
1/10 K.G.

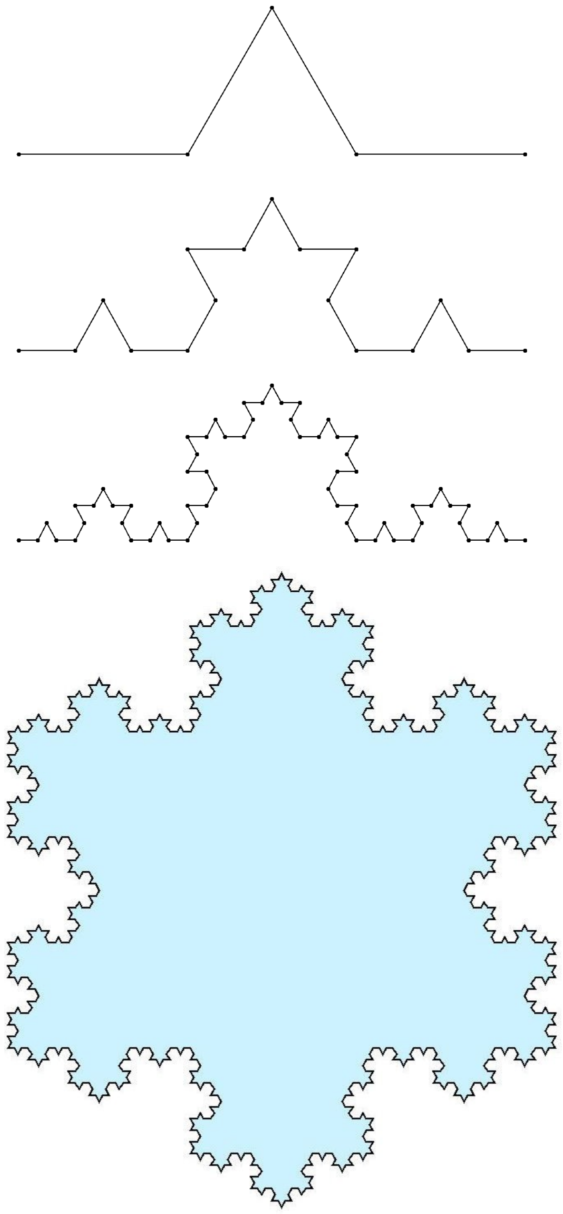
THE KOCH SNOWFLAKE

In truth, neither trees nor Droste images are true fractals because their recursion depth is necessarily limited. A true fractal has infinite detail because it has infinite depth. One of the first true fractals to be studied appeared in a paper by the Swedish mathematician Helge von Koch in 1904. It is easy to see how it is constructed: Draw a line; construct a 'tent' on the middle third; do the same with all four lines; do the same over and over again...

Every time you do this operation you increase the length of the line by one third. It follows that after an infinite number of operations the line will be infinitely long. It will, however easily fit onto the same piece of paper as the original tent. Three of these lines make a kind of 'snowflake' whose perimeter length is infinite but which encloses a finite area.

Here is the pseudo-code for the snowflake:

```
DIM: depth AS INTEGER = 4                                              recursion depth

CALL: Koch(depth,  0, 0, 100, 0)                             call subroutine with first line
CALL: Koch(depth, 100, 0, 100, -120)                        call subroutine with second line
CALL: Koch(depth,  50, 86.7, 100, 120)                       call subroutine with third line

DEFINE: Koch(d, x, y, length, angle)
    IF d = 0 THEN                                                       when depth = zero
        CALL: DrawLine(x, y, length, angle)                                  draw a line
    EXIT                                                             and exit subroutine
    length = length/3                                                    divide length by 3
    CALL: Koch(d – 1, x, y, length1, angle)                   call Koch for first segment
    x += length*COS(angle): y += length*SIN(angle)                         update coords
    CALL: Koch(d – 1, x, y, length, angle + 60)             call Koch for second segment
    x += length*COS(angle + 60): y += length*SIN(angle + 60)      update coords
    CALL: Koch(d – 1, x, y, length, angle – 60)               call Koch for third segment
    x += length*COS(angle - 60): y += length*SIN(angle – 60)      update coords
    CALL: Koch(d – 1, x, y, length, angle )                     call Koch for last segment
END_DEFINITION

DEFINE: DrawLine(x, y, length, angle)                              Draws a suitable line
    ...
END_DEFINITION
```


The Koch Snowflake

The Koch snowflake will tile the plane using successively smaller and smaller versions.

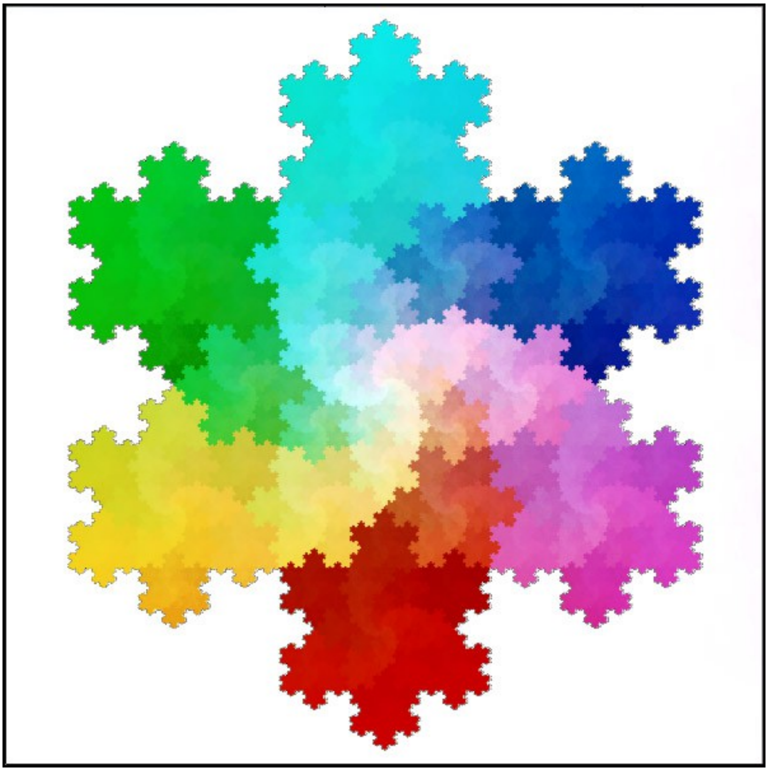
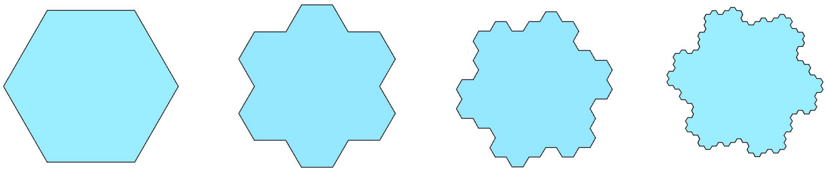


Illustration: Wikimedia commons

There are a number of variations on this theme. This one is called the 'Gosper Flowsnake' curve.



The following illustration shows how it will tile the plane. Each successive iteration is made of smaller and smaller hexagons.

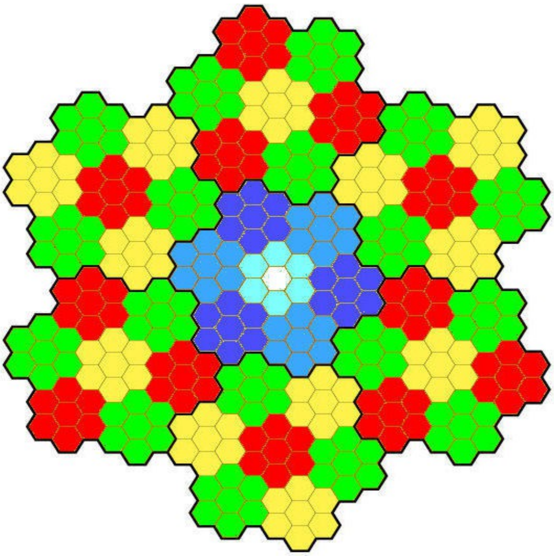
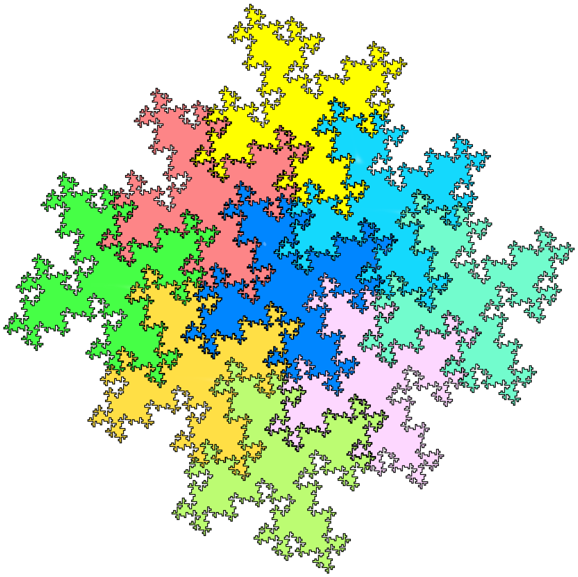


Illustration: Wikimedia commons

This one is called the quadratic Koch island curve. Remarkably, itt too tiles the plane.

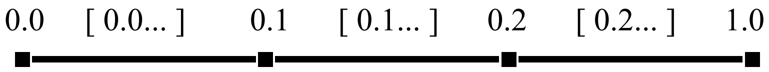


The Sierpinsky Carpet

This fractal was first described by Wacław Sierpiński in 1916. The square carpet is divided into 9 sub-squares and at each iteration, the centre square is removed.

The area which is left after each operation is easily seen to follow the series: $8/9$, $(8/9)^2$, $(8/9)^3$, …, $(8/9)^n$. Now this series tends to zero as n tends to infinity and so the area of the fractal carpet is actually zero. But this does not mean that there is nothing left! This depends on precisely how the squares are removed.

Consider a simpler procedure: Consider the line from 0 to 1. Label all the points on the line in ternary notation[1]. We can now easily divide our number line into three as follows:

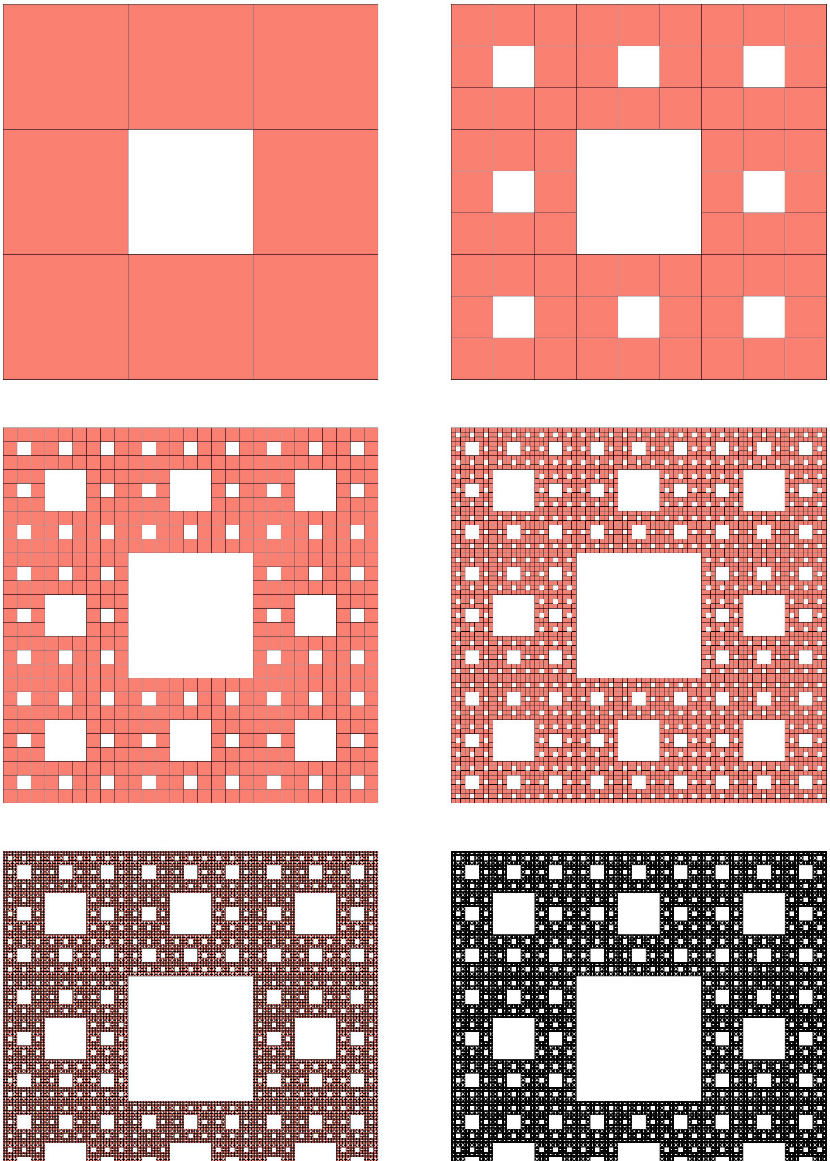


Now when we remove the middle third, it is important to say whether or not we wish to remove the end points i.e. the points labelled 0.1 and 0.2. We shall choose not to remove these. In other words we shall choose to remove all those points whose ternary label begins 0.1... with the exception of 0.1 itself.

Now we shall remove the middle third of the two remaining sections. Can you see that this will entail removing all the points beginning with 0.01... and 0.21... with the exception of 0.01 and 0.21? In the end we shall remove all points with a 1 in the ternary expansion except those that just terminate with a 1. What this means is that numbers like 0.20122 and 0.002121 are removed but $0.202\underline{02}$... and 0.00221 survive.

In the case of the Sierpinsky carpet, any point with at least one surviving coordinate survives so although the carpet is full of holes and has lost all its pile, much of the warp and weft remains intact!

1 Ternary uses the digits 0, 1 and 2. For example, the number 102 in ternary notation equals eleven because $1 \times 9 + 0 \times 3 + 2 \times 1 = 11$. Similarly the number 0.102 equals $0.407\underline{407}$... because $1 \times 1/3 + 0 \times 1/9 + 2 \times 1/27 = 11/27$



Other Sierpinsky Fractals

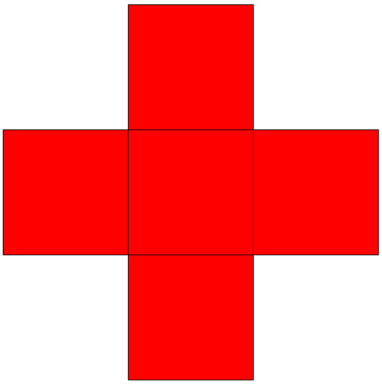
It should be fairly obvious how the fractals illustrated opposite are constructed. The upper one is called the Vicsek carpet and is generated in a similar way to the Sierpinsky carpet but this time the four corner squares are removed.

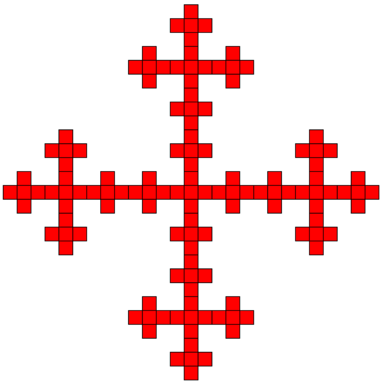
The Sierpinsky triangle replaces each triangle with a stack of three smaller triangles.

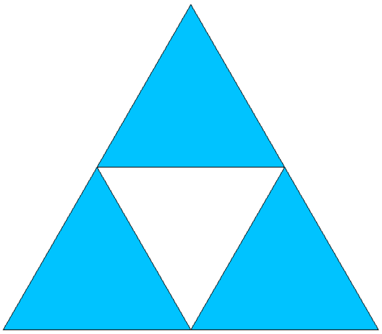
The Sierpinsky hexagon replaces each hexagon with six smaller hexagons. Notice that the holes inside the fractal are Koch snowflakes!

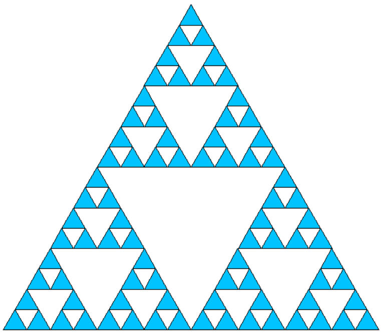
The pseudocode below can easily be adapted to draw any of these fractals with a suitable routine for drawing a shape. You might like to experiment with pentagons or even circles.

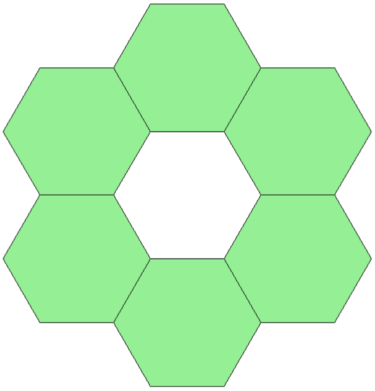
```
DIM: depth AS INTEGER = 4                              recursion depth
DIM: reductionfactor as float = ***       factor by which size is reduced

CALL: Carpet(depth,  0, 0, 100)          call subroutine with first shape

DEFINE: Carpet(d, x, y, size)
    IF d = 0 THEN                                       when depth = zero
        CALL: DrawShape(x, y,  size)                         draw a shape
    EXIT                                            and exit subroutine
    size = size/reductionfactor                            divide length
    LOOP                                        repeat the loop as required
        x  = ***: y = ***                                update x and y
        CALL: Carpet(d – 1,x, y, size)
    ENDLOOP
END_DEFINITION

DEFINE: DrawShape(x, y, size)                      Draws a suitable shape
    ...
END_DEFINITION
```

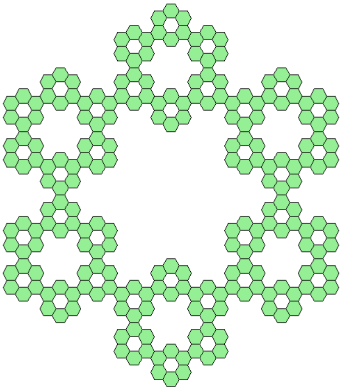












Lindenmayer Fractals

The fractals we have drawn so far have been fairly straightforward and of limited interest but in each case we have seen how the fractal is defined by a simple rule – 'draw a tent on the middle third', 'remove the corner squares' etc. etc. Lindenmayer fractals are generated by codifying these rules into a formula which defines the behaviour of a 'turtle' which draws out the fractal. To illustrate how the system works, consider how the Koch curve can be defined

Define STEP as: STEP; TURN LEFT 60°; STEP, TURN RIGHT 120°; STEP; TURN LEFT 60°; STEP

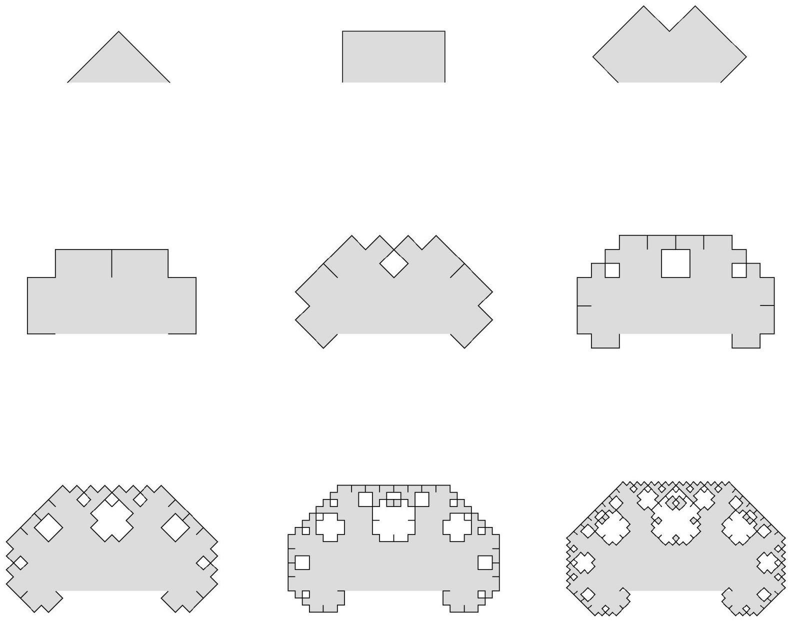
Notice how the definition is recursive (because it includes a reference to itself)). In addition it is necessary to specify how to start the process which in this case is simply to make a single STEP. In practice it is usual to specify a single angle and to give each formula an upper case letter so the above rule can be reduced to:

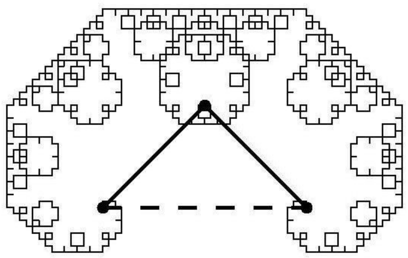
Koch curve: angle = 60°; A = A + A – – A + A; start = A
where + means turn left and – means turn right.

In the following pseudocode, whenever an “A” is encountered, the subroutine is called again with the whole formula.

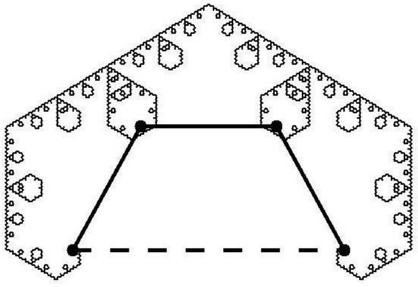
```
DIM: depth AS INTEGER = 4                                   recursion depth
DIM: formula AS STRING = ***                            Lindenmayer formula

CALL: Lindenmayer(depth,  0, 0, 100, 0, formula)            call subroutine

DEFINE: Lindenmayer(d, x, y, length, angle, formula)
    IF d = 0 THEN                                         when depth = zero
        CALL: DrawLine(x, y,  length, angle)                    draw a line
        x += length*COS(angle): y += length*SIN(angle)               update
    EXIT                                            and exit subroutine
    length = length/3                                    divide length by 3
    FOR EACH char IN formula
        IF char="A" THEN Lindenmayer(d-1, x, y, length, angle, formula)
        IF char="+" THEN angle += 60
        IF char = "-" THEN angle -= 60
    END FOR
END_DEFINITION
```

The Koch curve has a palindromic formula – i.e. it reads exactly the same backwards. Another Palindromic formula generates the Lévy C-curve shown below, as does the hexagonal C-curve.



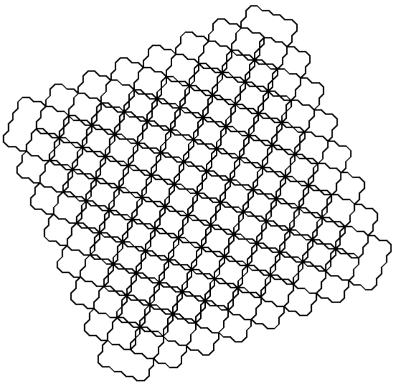


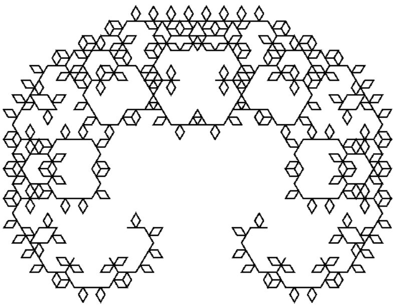
The Lévy C-curve
45°: A = + A – – A +: A

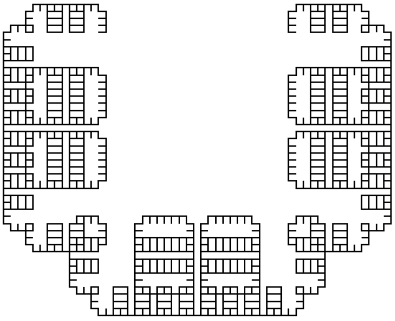


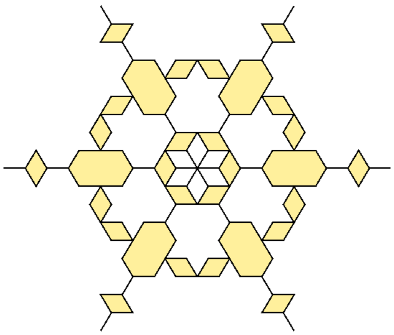
The hexagonal C-curve
60°: A = + A – A – A + : A

One problem with palindromic formulae is that they may not have the same number of +'s as -'s. One consequence of this is that the successive generations do not superimpose themselves on previous generations in the same way that the Koch curve does:


The wire mesh curve
45°: A = A – A + A – A : A + A


The leafy C-curve
60°: A = + A – A – A – – A – A – A + : A

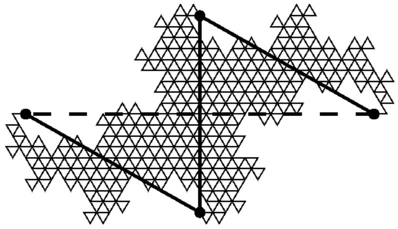

The city block curve
90°: A = + AAA + : A

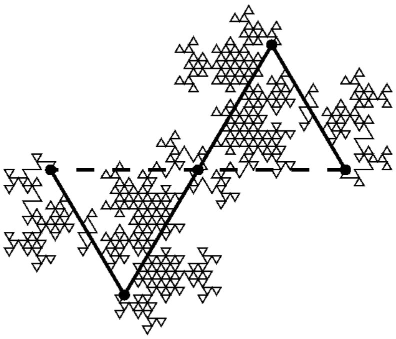

The snowflake curve
60°: A = + A – A + + A – A + : A

This type of formula does, however, generate some interesting results.

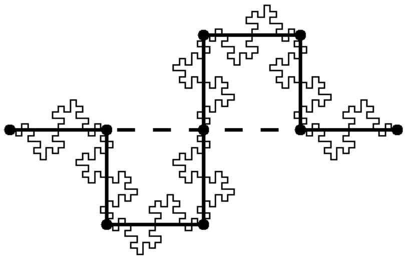
One way to make the total angle turned zero is to reverse the signs of the second half of the palindrome. One particularly interesting example of this kind of formula is the Tetra-dragon curve shown below. If you look carefully at the base lines shown in bold, you will see that while the zig-zag line has a length of 3 units, the dotted line has a length of √3 units. The Hausdorf dimension[2] of this curve is therefore $\frac{\log 3}{\log \sqrt{3}} = 2$

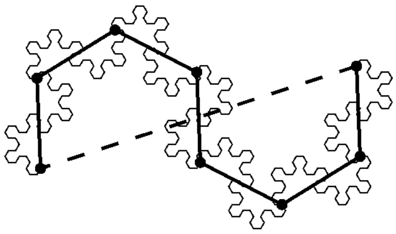
It is also the case that the curve never retraces itself and it is therefore a space-filling curve. It has the same dimensions as a complete surface. The second curve shown below also has dimension 2 and is also space-filling.


The Tetra-dragon curve
120°: A = A + A - A : A


120°: A = A + AA – A : A

Here are two more examples of palindromic formula with reversed signs.


90°: A = A – A + A + AA – A – A + A : A


60°: A = A – A – A – A + A + A + A : A

2 For a definition of Hausdorf dimensions see the Appendix.

TWIN FORMULA FRACTALS

All the fractals on the previous page employ a single formula. But many interesting fractals can be drawn using multiple formulae.

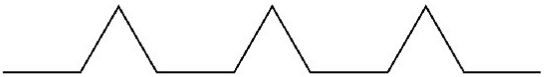
Suppose we start with the Koch curve whose formula is

60° : A = A + A – – A + A : A

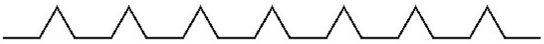
Now suppose that we want to do something different with the 'tent'. We can define a new variable B. Lets start by defining B = B. This is the complete formula:

60° : A = A + B – – B + A : B = B : A

and this is the result of the second iteration:



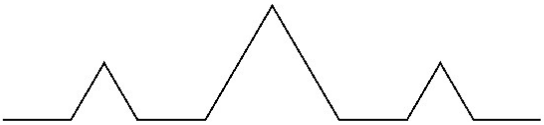
The two horizontal sections have been expanded but the middle 'tent' is unchanged because, at the end of the day, 'B' just means 'step forward one unit'. Here is the third iteration:



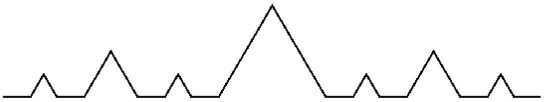
Now try to guess what the following formula does:

60° : A = A + B – – B + A : B = BB : A

i.e. replacing B = B with B = BB. If you deduced that the middle tent would be twice as big, you would have deduced correctly. Here is the second iteration:

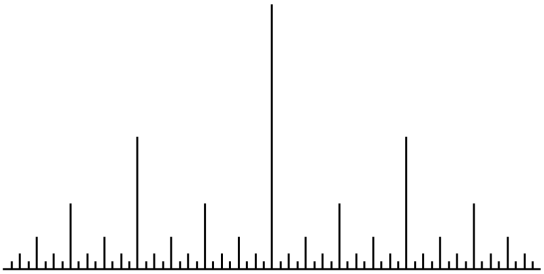


You might be a bit surprised by the third iteration, though:



On the other hand, it makes perfect sense that each new iteration would generate smaller and smaller 'tents'.

Using an angle of 90° instead of 60° generates a lovely binary ruler:



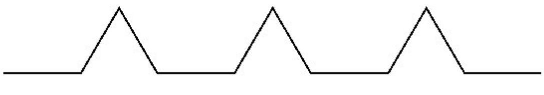
Now suppose we use the formula B = A. i.e.

60° : A = A + B – – B + A : B = A : A

You could be forgiven for thinking that this would be exactly the same as using the original Koch formula – but you would be wrong. The second iteration does what you expect (because at the end of the day (i.e. at the bottom of the recursive pile) B is the same as A. But the third iteration is different. To see why this is we have to expand the formulae as if we were doing the iterations by hand: After the first iteration A becomes (A + B – – B + A) and B becomes A so the result is

(A + B – – B + A) + A – – A + (A + B – – B + A)

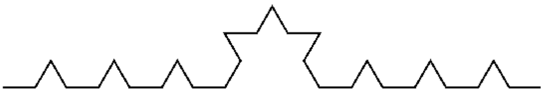
which looks like this:



The next iteration, however, becomes:

((A + B – – B + A) + B – – B + (A + B – – B + A)) + B – – B + ((A + B – – B + A) + B – – B + (A + B – – B + A))

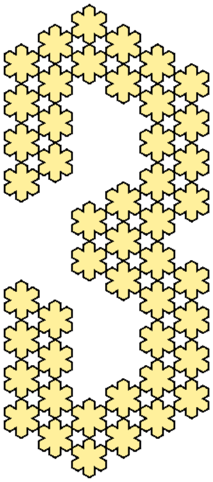
which looks like this:



Things get more interesting if we allow B to be a completely new formula but I defy any one to predict the results without trying it out on a computer. Take this formula for example:

60° : A = A + B – – B + A : B = A + A : A

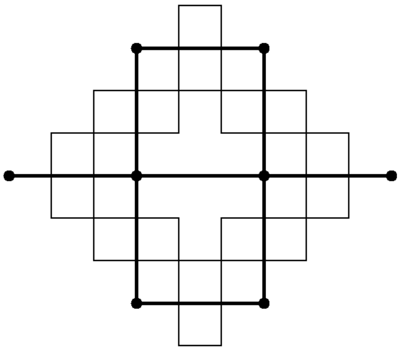
This is the astonishing result!

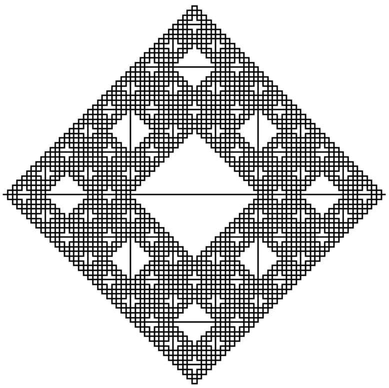


Another interesting fractal is one which generates a kind of Sierpinsky carpet. Its definition is:

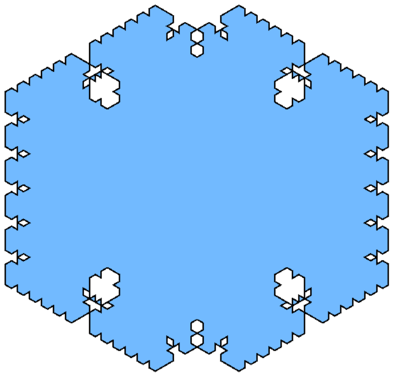
90° : A = A+A–A–A–B+A+A+A–A : B = BBB : A

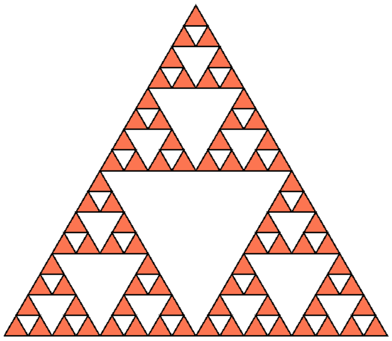
and it looks like this:

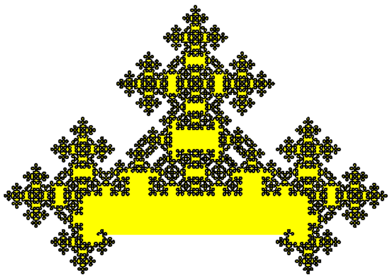

Sierpinsky carpet base lines

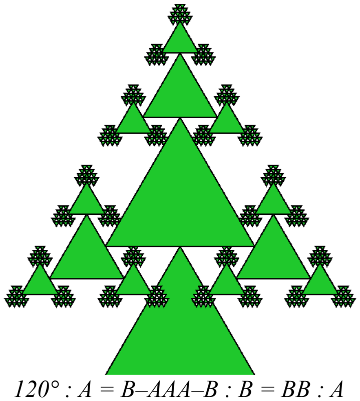

Sierpinsky carpet (depth = 5)

Here are a few more colourful examples:


60° : A = –AA+BB+AA– : B = A : A+++A

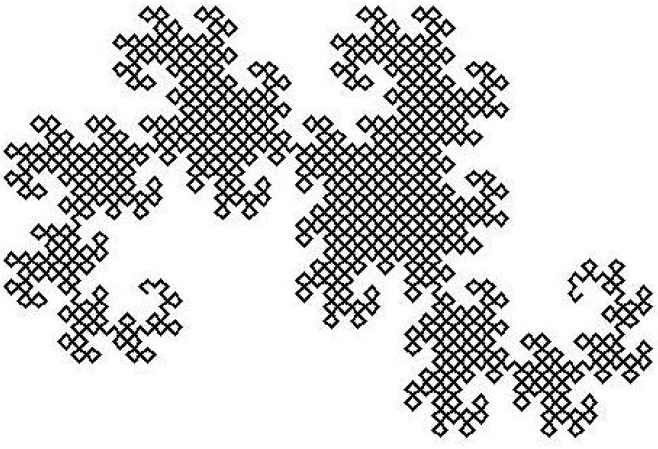

120° : A = A+B–A–B+A : B = BB : A+B+B


45° : A = +A+B+A+B+A+
B = –B–A–B–A–B– : A


120° : A = B–AAA–B : B = BB : A

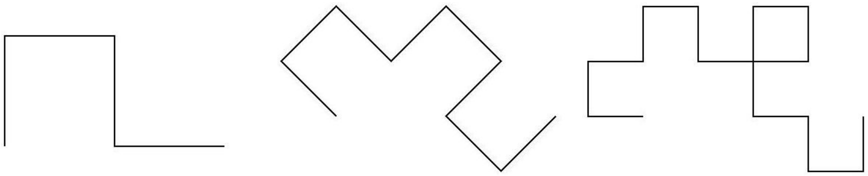
Often it is a good idea to make the definition of B a mirror image of A (i.e. the same but with all the signs reversed.) A good example is the Dragon curve whose definition is:

90° : A = A – B : B = A + B : A



The Dragon curve.
90°: A = A – B: B = A + B: A

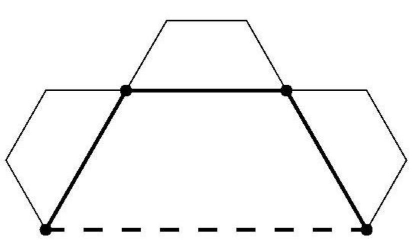
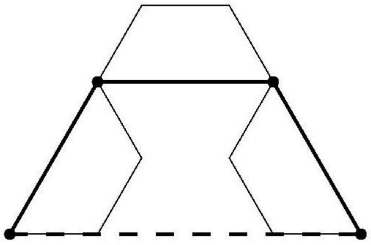
This curve has many interesting features. Like the Tetra-dragon curve on page19 it is a space-filling curve with Hausdorf dimension 2. Amazingly, you can make a Dragon curve with a strip of paper! If you fold a strip of paper repeatedly in the middle, then open it out so all the folds are at right angles, this is what you get!



If you could fold it 11 times (!) it would look like the figure above.

Another remarkable feature of the dragon curve it that, in spite of its complex shape, it will tile the plane!.

Yet more interesting fractals can be generated by using definitions of B in which the letters are reversed as well as the signs. The Arrowhead curve is an example. The basic pattern (shown on the left below) is this: + A – A – A + which generates half of a hexagon. To generate the Arrowhead curve, however, we need to modify this so as to produce the pattern on the right.

Instead of the three segments being the same, it is clear that the first and third segments are different so the definition of A must be

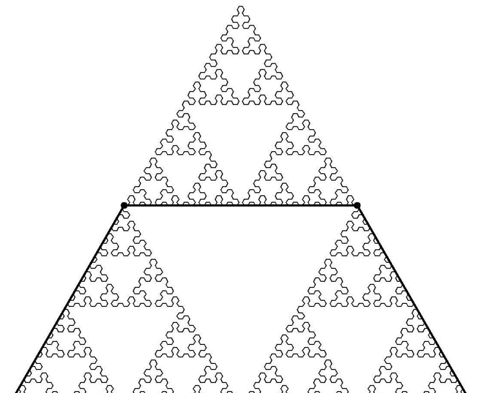
A = + B – A – B +

There are many definition of B of the form

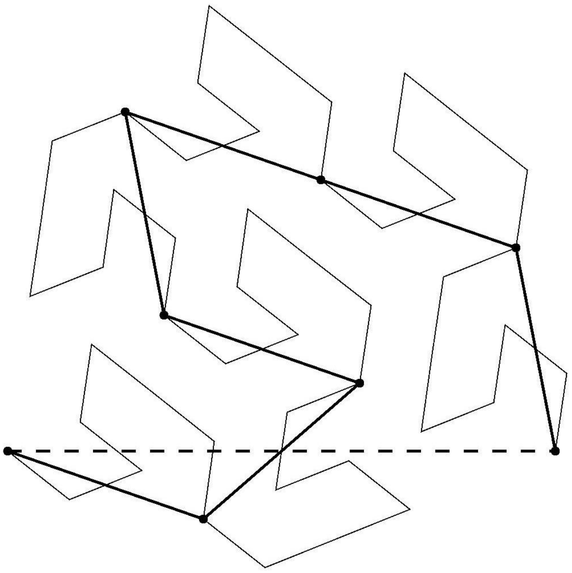
B = – X + X + X –

(where X is either A or B) will generate the next step but the one we want is:

B = – A + B + A –


The Arrowhead Curve

One of the most interesting of all the L-system fractals is the Peano-Gosper curve. It has seven segments and the characteristic angle is 60°.



The Peano-Gosper curve

Tracing through the first iteration (shown above in bold) reveals that the first iteration the formula has the form:

A = X + X + + X – X – – X X – X +

(The final + is necessary to ensure that we end up facing the same direction in which we started.)

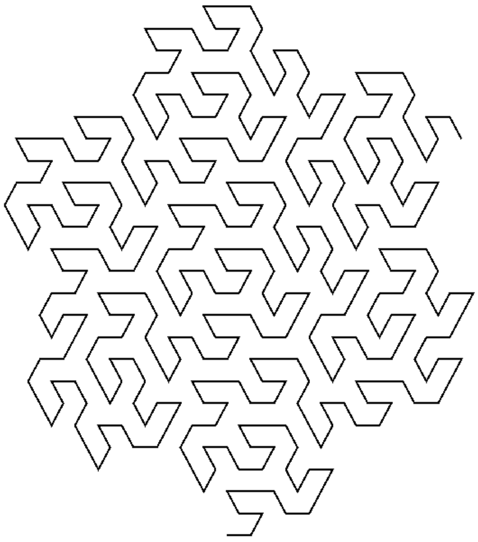
A careful examination of the second iteration shows that the second, third and last segments are reversed. The formula we need is therefore:

A: A + B + + B – A – – A A – B +

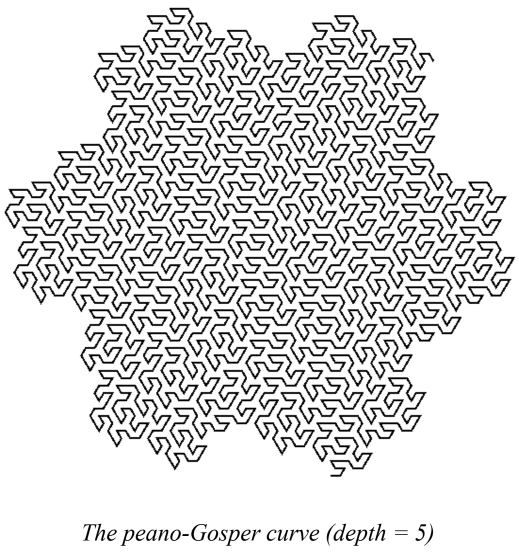
The formula for B is going to have the same basic form as that of A and to achieve the desired result we have to reverse not only the signs and the letters, but also the direction of the formula too. i.e.

B: – A + B B + + B + A – – A – B

Like the dragon curve, the Peano-Gosper curve is space-filling. This is the result:



The peano-Gosper curve (depth = 4)



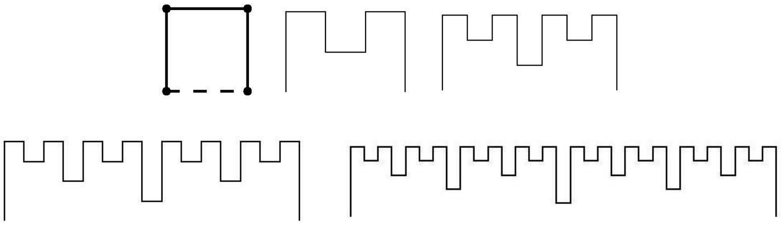
The peano-Gosper curve (depth = 5)

Interestingly, this curve fits exactly into the Gosper flowsnake curve described on page 10.

Advanced Lindenmayer Formulae

A major restriction on the formulae which we have considered so far is that, since every line is replaced by a squiggle which begins and ends at the same points, every point in one level is repeated at every deeper level. Often this is exactly what we want but if we relax this condition a whole new class of fractals become possible.

Consider, for example, a ruler which is divided into inches, half inches, quarter inches etc. etc. each division being shorter than the last. It is clear that this is a fractal structure with each division being made up of two copies of itself. What I am looking for is a formula which will produce the following sequence:



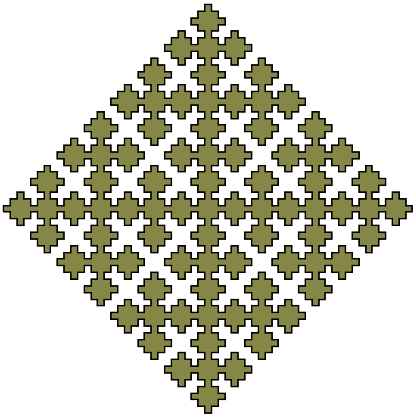
Now it is easy to see that there is no way the original four corners of the square in the first image can be placed on the eight points in the second. So how is this achieved? The answer is that we must separate the steps from the formula. Here is the solution:

angle: 90°; A = + S – a S a – S +; start: A

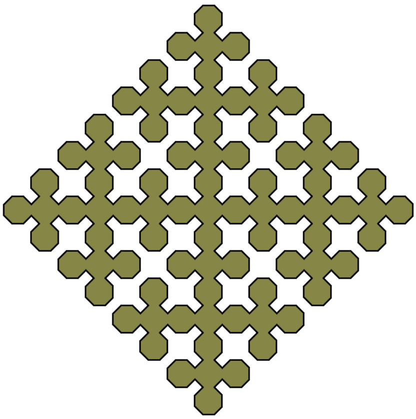
where 'S' means 'step forward' and the lowercase 'a' means 'carry out formula A but without stepping forward'.

You can see that the first iteration (which simply ignores the a's) will result in + S – S – S + which is a simple square but the second iteration will generate the following series of steps: + S – (+ S – S – S +) S (+ S – S – S +) – S + which is what we desire.

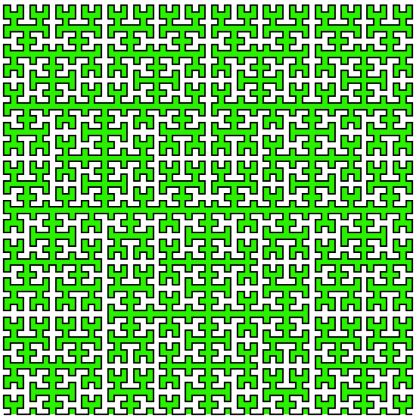
It has to be admitted that it is almost impossible to work out from the formula what the result is going to look like but the opposite page shows four more amazing fractals with their respective formulae.



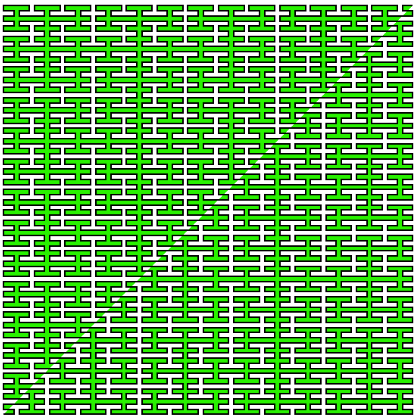
The Castle curve
A = a S – S + S – a S + S + a S – S + S – a
Angle: 90°; Start: A S + S + A S + S



The Sierpinski curve
A = + b – S – b + ; B = – a + S + a –
Angle: 45°; Start: A – – S – – A – – S



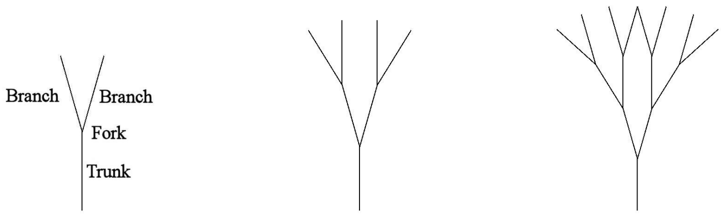
The Hilbert curve I
A = + b S – a S a – S b +;
B = - a S + b S b + S a -
Angle: 90°; Start: A



The Hilbert II curve
A = a S b S a + S + b S a S b – S – a S b S a
B = b S a S b – S – a S b S a + S + b S a S b
Angle: 90°; Start: A

LINDENMAYER WEEDS

All the Lindenmayer systems so far considered produce a single, unbroken curve. An extra refinement of the Lindenmayer system permits you to add branches. For example, suppose we wish to draw a simple fractal tree along the following lines:



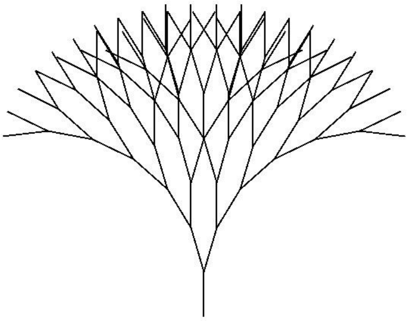
A 'tree' is basically a 'trunk' which reaches a 'fork' which divides into two 'branches' each of which itself is a 'tree'. We can translate this formally into the following sequence of instructions:

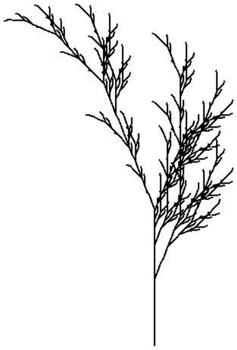
```
Draw the trunk
Turn left
Draw a tree
Return to the fork
Turn right
Draw a tree
```

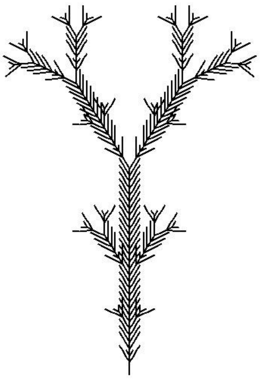
In the Lindenmayer systems we have met so far, the instruction 'Draw the trunk' is translated into 'Step forward one unit' which is coded 'S'. Turn left is '+' and turn right is '–'. 'Draw a tree' is, of course, coded as the whole sequence of instructions 'A'. But how do we 'Return to the fork'?

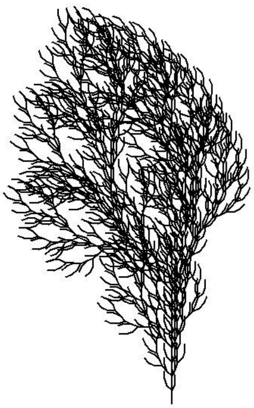
To do this we need to record exactly where we are when we reach the fork. This is done using the left square bracket '['. Then when we wish to return there we use the right square bracket ']'. The complete formula is therefore:

$$A = S\,[+A] - A; \quad Angle{:}\ 15°; \quad Start{:}\ A$$


A simple tree (depth = 7)
A = A [+A] – A
Angle: 15°; Start: A


Bourke Weed
A = B – [[a]+a]+B[+Ba] – a
B = BB
Angle: 22°; Start: A


Bourke Club Moss
A = aSb [+a] [– a]
B = b [– SSS] [+SSS] Sb
Angle: 32°; Start: A

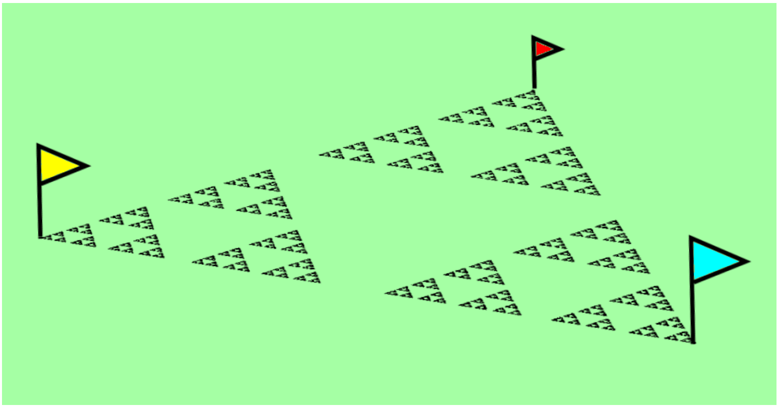

Bourke Bush
A = AA+[+A–A–A] – [–A+A+A]
Angle: 22°; Start: A

Three of the above systems are taken from Paul Bourke's excellent website:

http://paulbourke.net/fractals/lsys/

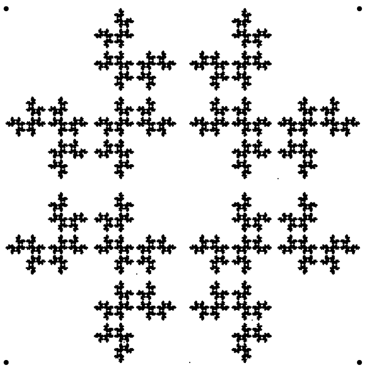
The Chaos Game

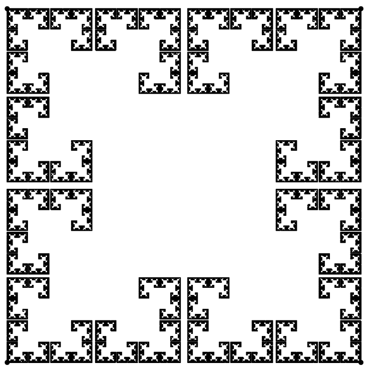
Arm yourself with thee poles and a bucket full of pebbles. Go to a nearby field and stick the three poles anywhere in the ground. Take one of the pebbles and throw it as hard as you can. Walk over to where the pebble has landed. Choose one of the flags at random and walk just over half way towards it. Place another pebble on the ground. Choose another flag and do this over and over again until you have run out of pebbles. When you have finished, look at where the pebbles have ended up. You will be astonished to see that (apart from the first few pebbles) they have all ended up in piles tracing out a Sierpinsky triangle (see page 14) as shown below.

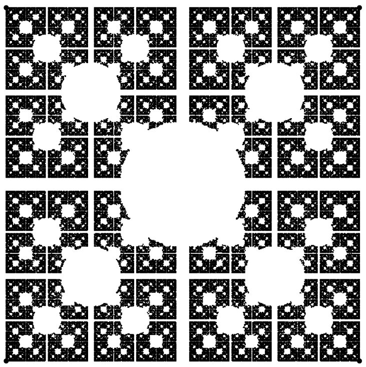


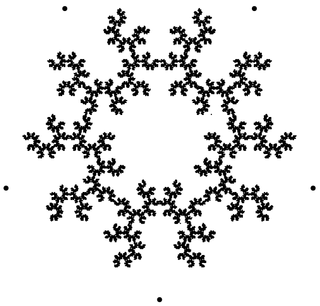
This is what is known as an attractor. If there had been just one pole, you would have placed a line of pebbles heading towards the pole; with two poles you will end up placing them in heaps along the line between the two poles; but with three or more poles, it is clear that the attractor is, in fact, a fractal.

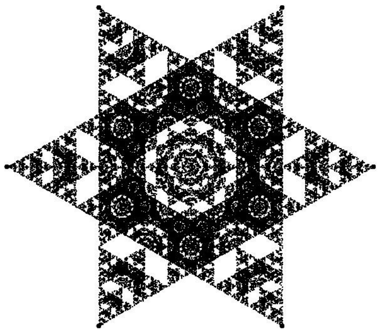
By using more poles and adding other restrictions such as 'any pole except the one you have just visited', 'any pole except the ones on each side', 'any pole except the one opposite' etc. you can generate a host of other interesting patterns. The third one opposite was generated by 'forbidding the circle in the middle'.

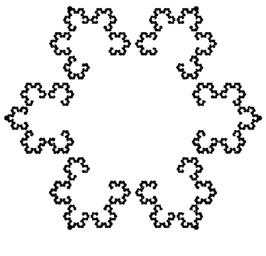

4 poles: 52%: not previous


4 poles: 51%: not opposite


4 poles: 52%: forbidden circle


5 poles: 59%: not previous


6 poles: 50%: not neighbour


6 poles: 68%: not opposite

LINEAR TRANSFORMATIONS

I don't know why the procedure described on the previous pages is called 'the Chaos Game'. It seems to me remarkable that such beautiful order can emerge from the apparently random placing of pebbles. In fact, these fractals are a special case of what is known as an Iterated Function System or IFS.

The basic principle is the same but instead of just walking a fixed fraction of the distance to a fixed pole, you calculate the position of the next pebble (x', y') using a pair of mathematical formulae:

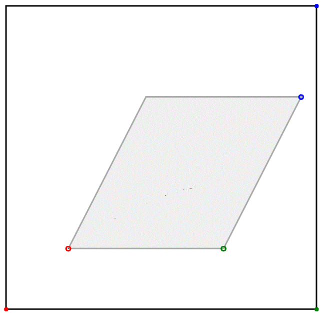
$$\begin{aligned} x' &= ax + by + e \\ y' &= cx + dy + f \end{aligned} \tag{1}$$

where (x, y) is the current location and a, b, c, d, e and f are constants. This set of equations comprises a *linear transformation*. It is linear because it only contains terms in x and y (not x^2 , xy etc.) and because of this, straight lines are always transformed into straight lines.

It is useful to see what happens to a square when it is acted upon. For example the transform

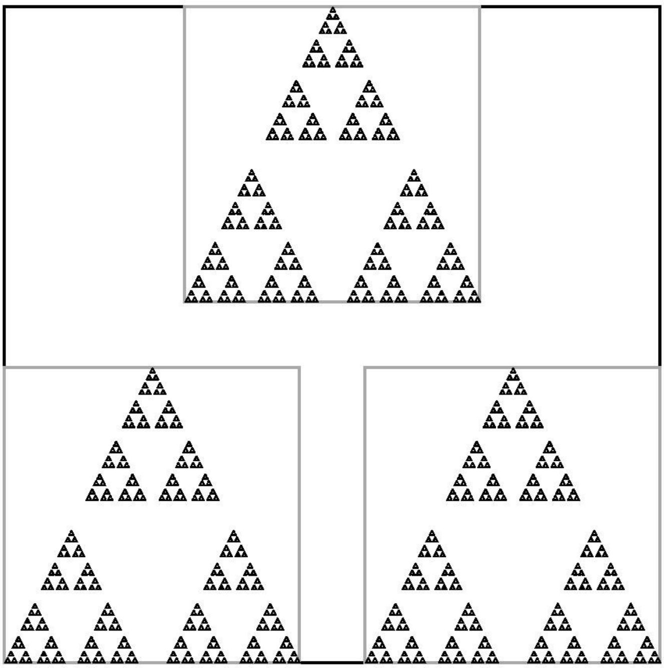
$$\begin{aligned} x' &= 0.5x + 0.25y + 0.2 \\ y' &= 0 + 0.5y + 0.2 \end{aligned} \tag{2}$$

causes a unit square to be scaled down to half its size, skewed sideways and moved to s new position like this:



The values of e and f are responsible for moving the square; it is the values of a, b, c and d which scale, skew and rotate the square.

We can now see why the Chaos Game produces the results which it does. Each time you walk towards a pole, you are reproducing the whole diagram but at a smaller scale. The pattern is therefore a result of three linear transformations as illustrated below.



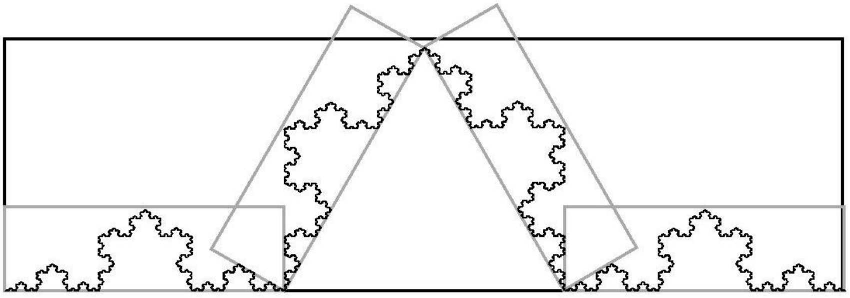
This fractal was generated using three transformations whose coefficients are:

a	*b*	*e*	*c*	*d*	*f*
0.45	*0*	*0*	*0*	*0.45*	*0*
0.45	*0*	*0.55*	*0*	*0.45*	*0*
0.45	*0*	*0.275*	*0*	*0.45*	*0.55*

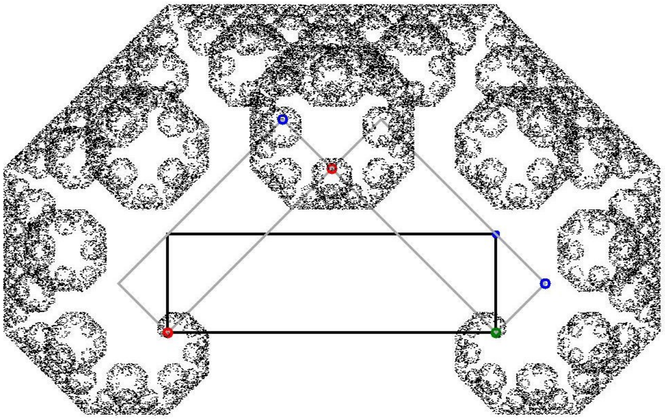
It is also easy to see why the result is a fractal. Each transformed square must contain a copy of the whole image, and, of course, each copy must contain three more copies of itself each or which must contain … etc. etc.

It is the Droste Effect all over again.

Four transforms will generate the Koch curve:

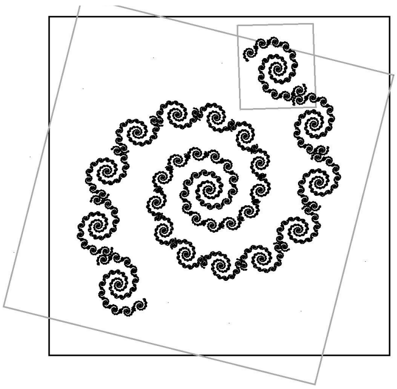


Only two are needed for the Levy C-curve:

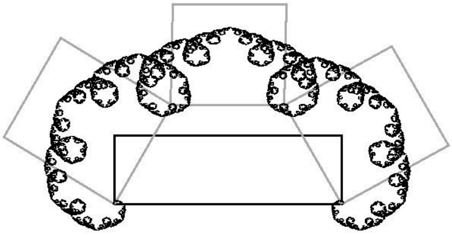


It is important to appreciate that *all the information* necessary to create the fractal resides in the transformations. The transformations define the fractal – and the fractal defines the transformations. This fact has been used to create very efficient algorithms for compressing what appears to be complex data.

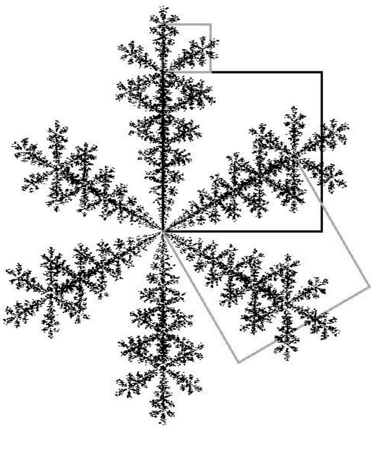
Some other IFS symmetrical systems are illustrated opposite.



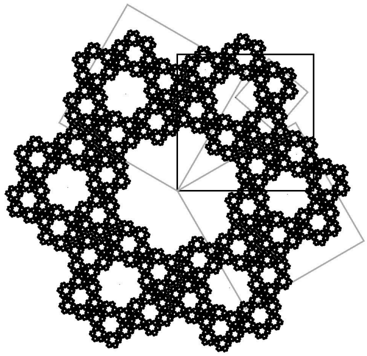
A spiral of spirals
Two transformations of very different sizes



Curlicue
Three transforms on a hexagon



Snowflake
Two transforms: one unscaled at 60°
the other reduced but not rotated



Doily
Three transforms; two at an angle
and one reduced

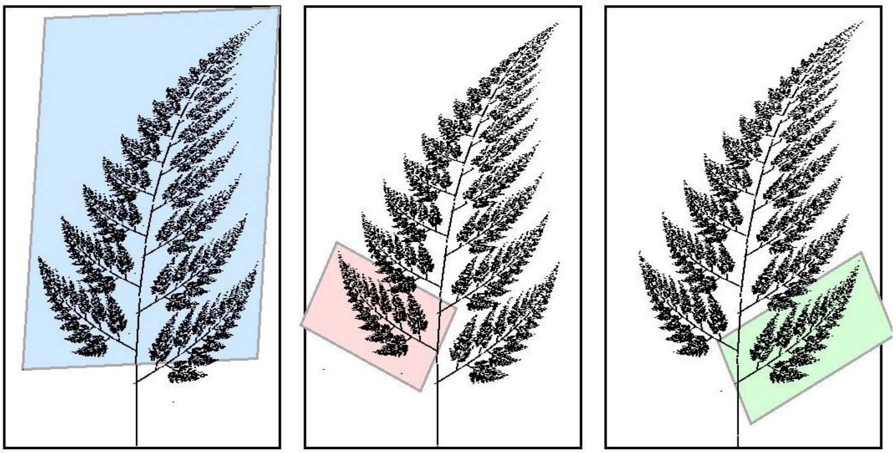
The Barnsley Fern

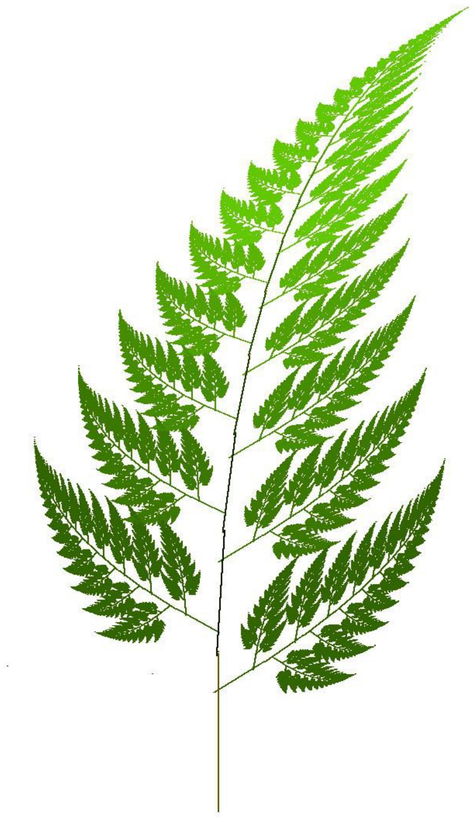
In 1988 Michael Barnsley proved an important theorem in mathematics called the collage theorem. Basically what it says is that if you can cover an arbitrary shape with smaller copies of that shape, then the transformations which define the copies will generate the original shape when used as an IFS.

It is relatively easy to see how this works in all the cases illustrated on the previous pages. This is because the rotation angles were chosen so that the resulting fractal would have a high degree of symmetry. But Barnsley's theorem applies to *any* shape. How can this be?

Well, if the shape is truly random then you are going to need a huge number of transformations to cover it completely. Only those shapes which have a degree of self similarity can be covered with a small number of transformations. As an illustration, Barnsley considered a fern.

As the illustrations below show, there are basically three ways in which a fern is self-similar, the most important of which is the fact that the whole leaf minus the lowest two leaflets is a smaller copy of the whole leaf. And of course, each of the leaflets is a much reduced copy of the whole leaf too.





This fern was created using the following transformations:

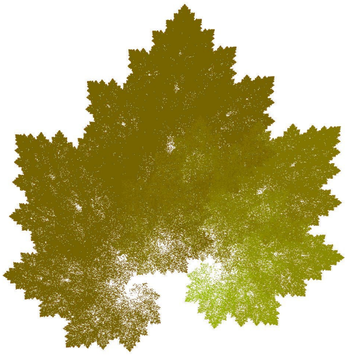
a,	*b,*	*e*	*c,*	*d,*	*f*	*p*
0.85,	*0.05,*	*0.034*	*0.04,*	*0.795,*	*0.142*	*65*
0.13,	*-0.27,*	*0.209*	*0.3,*	*0.15,*	*0.103*	*15*
-0.12,	*0.325,*	*0.264*	*0.3,*	*0.201,*	*0.044*	*15*
0,	*0,*	*0.24*	*0,*	*0.2,*	*0*	*5*

The column labelled p is a probability factor. The high value assigned to the first transform ensures that the iterated points will reach a long way up the frond.

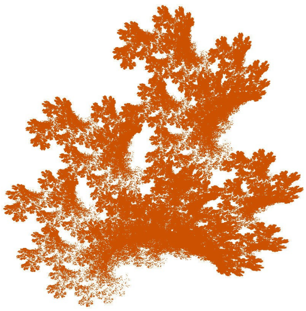
The fourth transform simply draws the stem

More

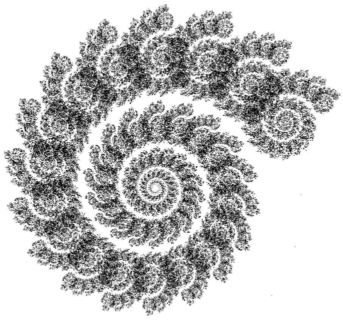
IFS Fractals in Nature



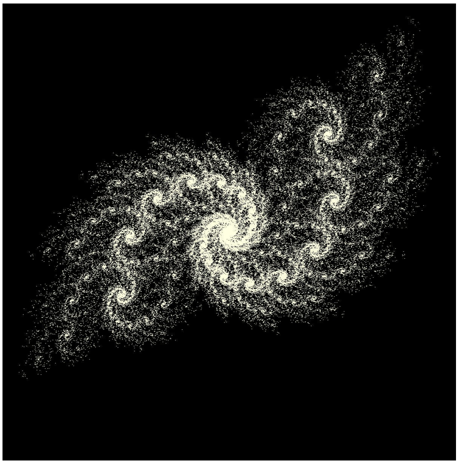
Three reduced transformations
One straight, one rotated left and the other rotated right



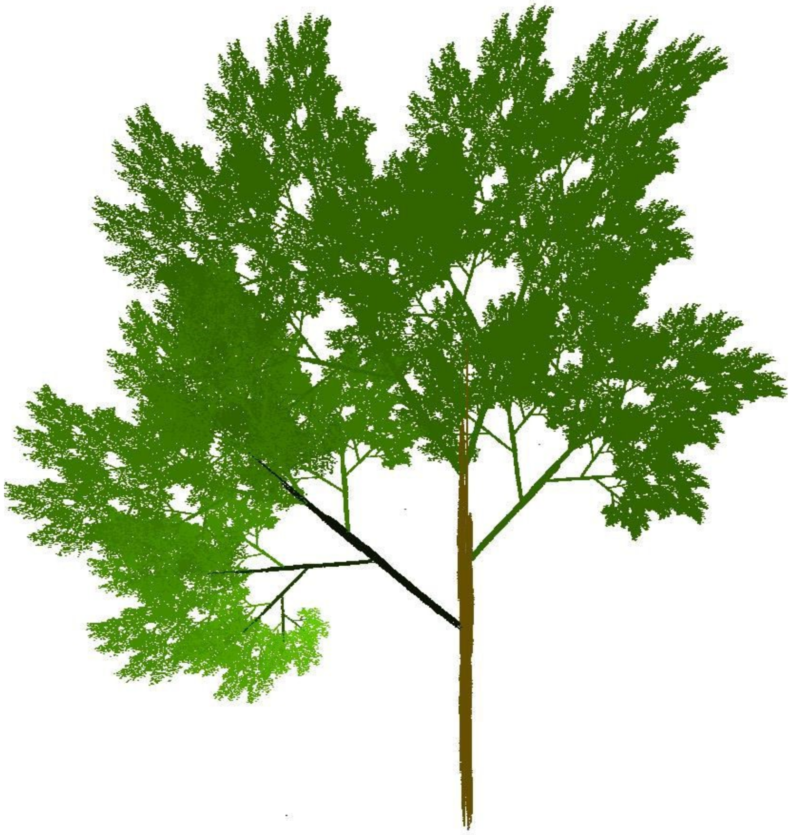
A coral
Three transformations rotated and skewed



Ammonite
Two transformations



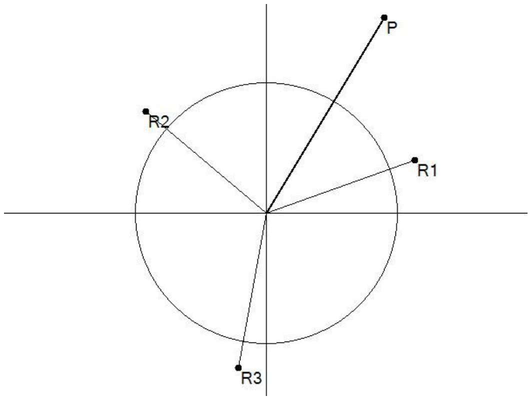
A spiral galaxy
Two transformations



A tree
Five transformations. One each for the four main branches and one for the trunk.

IFS Fractals in the Complex Plane

In the chaos game with three poles the rule was to step halfway towards a random pole. This is rather like taking the cube root of a complex number because when you take the cube root of a complex number z (at the point P) there are always three possible answers as illustrated below.



The rule is: take the cube root of the distance of the point P from the origin (this is known as the 'modulus' of z) and divide the angle it makes with the X axis (the 'argument' of z) by 3. This gives you the first root R1. The other two roots R2 and R3 are the same distance from the origin but at 120° to the first root.

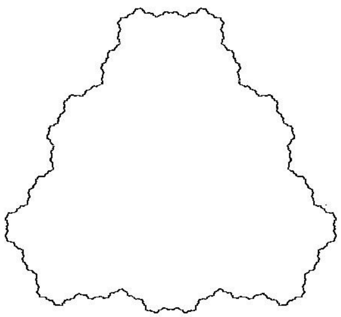
We can play the chaos game in the complex plane by simply choosing one of the three roots at random; then take the cube root of that and so on and so on.

Unfortunately the result is a bit disappointing. All we get is a simple circle. The reason is clear to see. Every time we take the cube root, the modulus gets closer and closer to 1. (Even when the modulus is less than 1, the cube root gets closer to 1.) Eventually we find ourselves hopping round a circle at random.

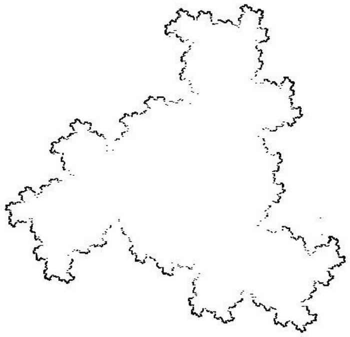
To make things a bit more interesting, what we do is to subtract a small constant (complex) number $\boldsymbol{c} = a + \mathbf{i}b$ from z before we take the cube root. To put it another way we are going to iterate the function

$$z' = \sqrt[3]{z - c} \tag{3}$$

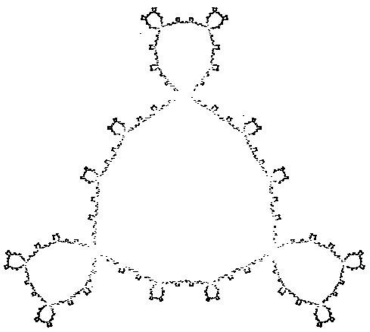
Now we are getting somewhere. This is what we get with various different values of ***c*.**



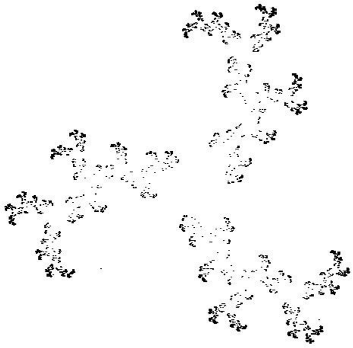
C = *0.5***i**



C = *0.5 + 0.5***i**



C = **i**

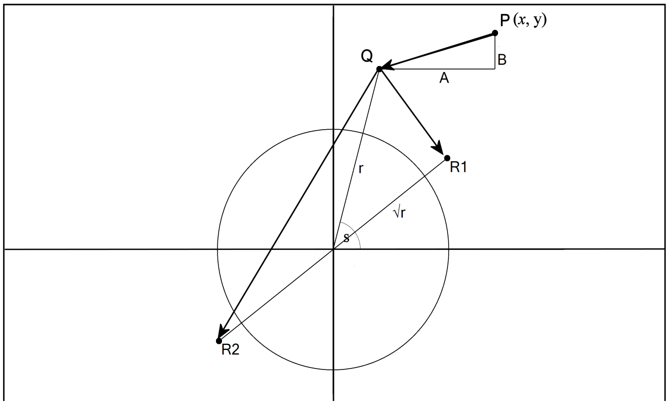


C = *0.7 + 0.5***i**

As you can see, different values of *c* generate different shapes; some of them consist of a distorted circle while others break up into fragments. Not surprisingly they all exhibit rotational symmetry of order 3. These are the Julia sets of equation (3).

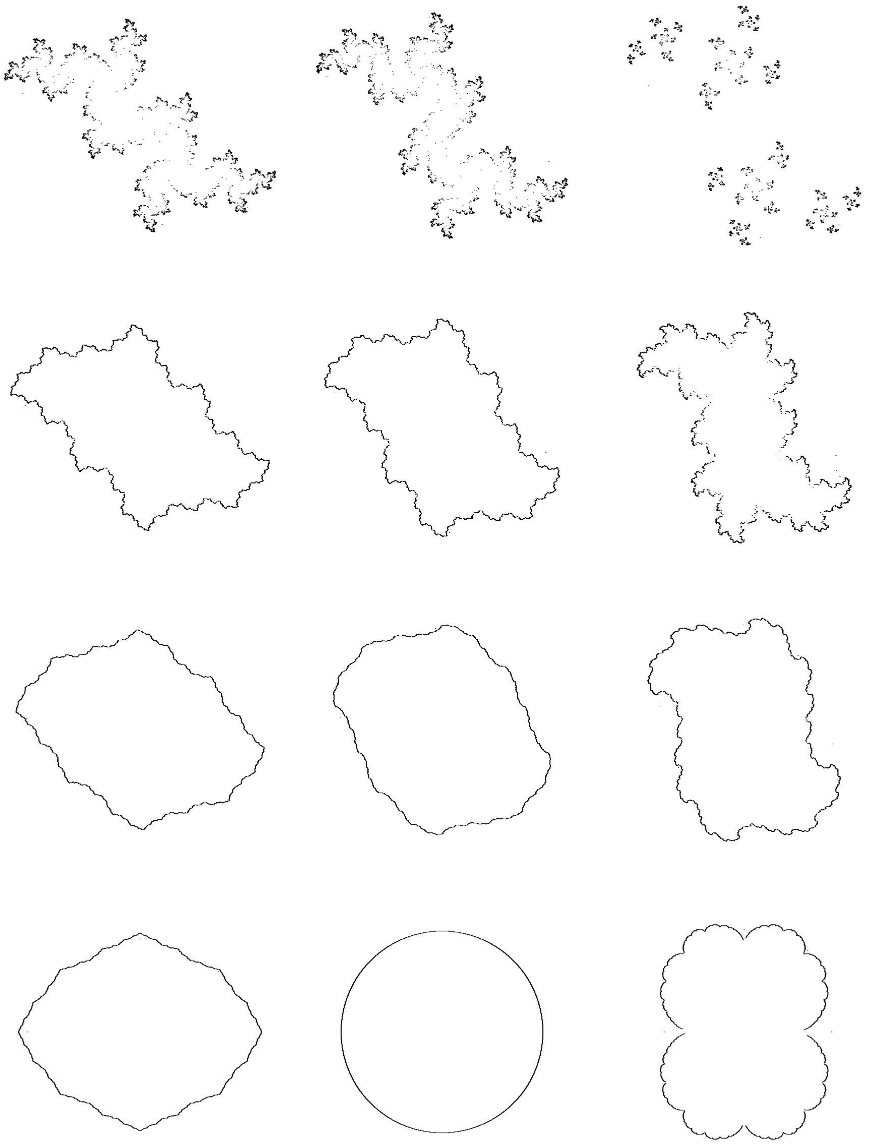
Standard Julia Sets

Although the Chaos Game only works with a minimum of three poles, the complex chaos game works just as well with square roots. This time we subtract $\boldsymbol{c}$ then takes the square root of the modulus and halve the angle. The following diagram shows the whole process and corresponds to the equation $z' = \sqrt[2]{z - c}$



The following pseudo-code shows how to do it:

```
DIM: x = 0, y = 0                                         any initial point will do

FOR I = 1 to 10000                                   plot as many points as required
   CALL: Iterate(x, y)                                              update x and y
   CALL: PlotPoint(x, y)                                plot a point at a suitable point
NEXT

DEFINE: Iterate(BYREF x, BYREF y)            x and y must be called BY REFERENCE
    x = x – a : y = y-b                       so that they are changed by the routine
    DIM: r = SQRT(x*x + y*y)
    DIM: a = ARCTAN(x, y)                                ARCTAN must deal correctly
    x =  SQRT(r) * COS(a / 2)                                 with all four quadrants
    y = SQRT(r) * SIN(a / 2)
    IF RANDOM(2) = 0 THEN x = -x: y = -y      choose one of the roots at random
END_DEFINITION

DEFINE: PlotPoint(x, y)            Plot a point on the screen corresponding to (x, y)
    …
END_DEFINITION
```



A grid of Julia sets with a = -0.25, 0 and 0.25; b = 0, 0.25, 0.5 and 0.75
Note that all the sets are rotationally symmetric. Sets with negative b are the same as those with positive b. The further you go from the origin the more fragmented the sets become.

HOPALONG FRACTALS

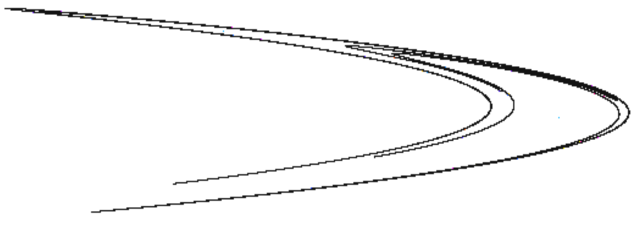
Basically, any IFS system, whether is be the Chaos Game, a set of linear transformations or a complex mapping can result in one of four possibilities. The iterated point can either vanish off to infinity, home in on a fixed point, cycle round a set of fixed points or wander through an infinite set of fixed points without ever repeating itself. Obviously we are most interested in the latter behaviour and it is this kind of behaviour which is termed a 'strange attractor'.

In all the cases we have studied so far, we have had to introduce an element of randomness into the process in deciding which transform to use or which root to take. If we don't do this, all our examples simply home in on a fixed point. For a long time it was thought that this would always be the case with a single transform but in 1976 the biologist Robert May drew attention to the curious behaviour of the simple logistic equation $x' = Ax(1 - x)$ which exhibits remarkable chaotic behaviour when $A > 3.75$. (For a lot more detail on this fascinating equation see my companion book: Chaos and the Logistic Equation.) Since the equation only has one variable, the resulting patterns are not very exciting but soon, other researchers were devising single-valued transforms in 2 or more variables which exhibited visually more interesting behaviour.

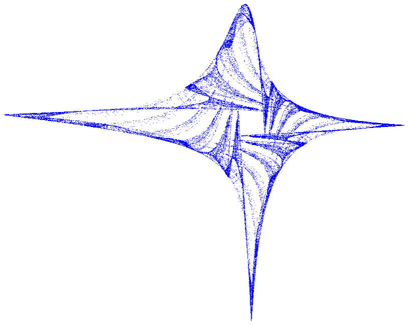
One of the first was discovered by Michel Hénon:

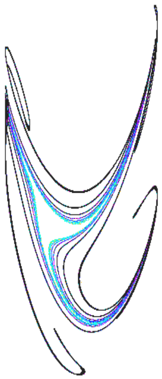
$$\begin{aligned} x' &= 1 - ax^2 + y \\ y' &= bx \end{aligned} \tag{4}$$

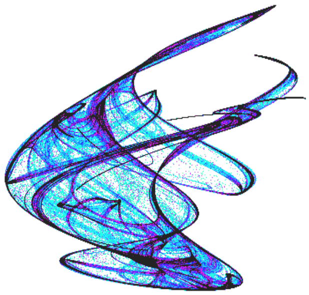
where a and b are constants. Typically $a = 1.4$ and $b = 0.3$. This is what it looks like:

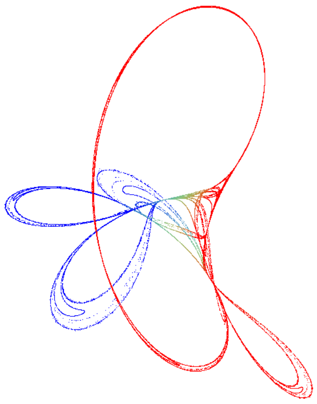


Since then thousands of other examples have been found, some of which are illustrated below.


Star of Bethlehem
$x' = 0.6 - 1.1y$
$y' = 1.2x - 0.6xy^2$


Linton's 'Ghost'
$x' = \sin(1.2y)$
$y' = -x - \cos(2y)$


De Jong
$x' = \sin(2y) - \cos(2.5x)$
$y' = \sin(x) - \cos(y)$


Tinkerbell
$x' = (x^2 - y^2) + 0.9x - 0.6y$
$y' = 2xy + 2x + 0.5y$

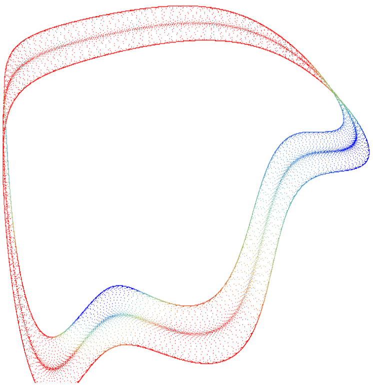
Contrary to what you might think, the point (x, y) does not move smoothly along these apparently linear structures; rather it hops seemingly at random from one point to another – hence the name 'hopalong fractals'. Detailed analysis shows that the structures have infinite detail and the way the point moves is chaotic in the sense that two points initially very close together soon diverge widely apart.

The range of different shapes is astonishing. Some like the Star of Bethlehem resemble crumpled pieces of paper; others like Linton's 'Ghost' and the Tinkerbell attractor are more like tangled balls of string; others combine both features while the majority defy description. Many have a three dimensional appearance even though they only employ two variables x and y. The reason for this is rather subtle. Take the Tinkerbell attractor for example. This appears to have a large loop which passes over a number of smaller loops. Now as I have said; this is an illusion. The attractor is just a flat map of points which are visited apparently at random. Every point is equally important. So is there anything that distinguishes the points in the large loop from the points in the smaller loops? The answer to this is yes. What distinguishes the two loops are the *previous values* of x and y.

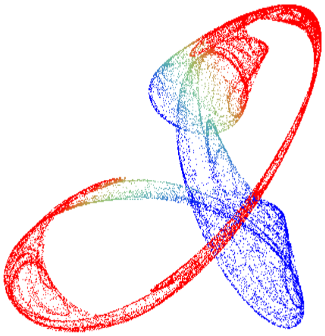
Now the equations which define the Tinkerbell attractor are quadratic – that is to say, they contain terms like x^2 and xy. Notwithstanding, given the current values of x and y the next pair of values is unique. On the other hand, if you were asked to find the *previous* values of x and y you would have to solve a quadratic equation which, in general, has two solutions. What this means is that while every point has only one offspring, it may have two parents (or more in the case of cubic and other equations). Usually, however, one of those parents will not be on the attractor so it can be discounted. Very occasionally, however, both parents are potentially on the attractor and this is the case at the points when the large loop crosses the smaller loops. All the points on the large loop arise from previous points with a positive value of y; these points have been coloured red. All the points on the smaller loops (coloured blue) arise from points with a negative value of y. The points where the loops cross are points which can be reached from either parent.

One way of making this crystal clear is to introduce a new variable z obeying the equation $z' = y$ and plot the point (x, y, z) in three dimensions. This can be done on a computer screen by rotating the object and viewing it from different angles or (as here) by colouring the points according to the z coordinate. The results can be extraordinarily beautiful.

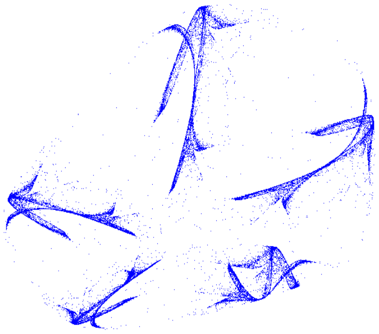
Even more variety can be achieved by making z an equal partner in all three equations. Some of the amazing results are illustrated below (though, sadly, not in 3D!)



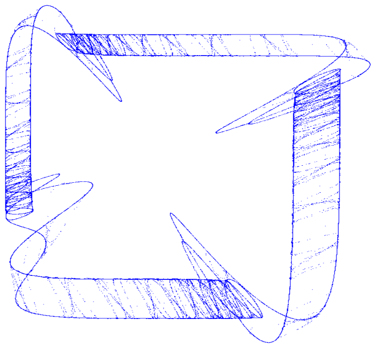
Ribbon
$x' = 0.3x - y$
$y' = x^2 - z$
$z' = x - 0.2y + 0.8y^2$



Bow
$x' = 1 + y - xz$
$y' = -0.8x^2 - 0.5y^2 + 0.1z^2$
$z' = x + y$



Jellyfish
$x' = -0.4 + 0.7y$
$y' = 1.2x - z + 0.7x^2 - 0.9xy$
$z' = -x + y - x^2 - y^2$



Square dance
$x' = -1.2z^2$
$y' = 1.2z + 0.6x^2$
$z' = -y - 1.2x^2$

Barry Martin Fractals

In September 1984 an article appeared in Scientific American which described an amazing IFS discovered by Barry Martin of Aston University, Birmingham. To understand how it works, lets start with a simpler set of functions namely:

$$\begin{aligned} x' &= x+y \\ y' &= -x \end{aligned}$$

If you follow this iteration by hand you will find that it returns to itself after exactly 6 iterations. For example the point (1,1) becomes (2, –1) then (1, –2) then (–1, –1), (–2, 1), (–1, 2) and back to (1, 1). These points trace out a slightly bowed rectangle.

If now you try the functions:

$$\begin{aligned} x' &= bx+y \\ y' &= -x \end{aligned} \tag{5}$$

where b is not equal to 1, you will find that any initial starting point will typically trace out an ellipse. (I say 'typically' because there are, no doubt, certain values of b which will render the orbit periodic.)

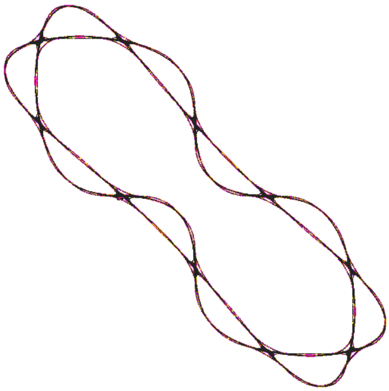
I suspect that Barry Martin wondered at this point what would happen if he tried putting $x' = bx^2 + y$ but he will have discovered that all initial points spiral off to infinity. No doubt he tried $x' = b\sqrt{x} + y$ too but that leads to an error whenever x is negative. The obvious thing to do therefore it to invent a sort of continuation of the square root function into the negative region by taking the square root of the absolute value of x and then replacing the sign. This is the sort of thing:

$$\begin{aligned} x' &= \mathrm{SGN}(x)\sqrt{|bx|}+y \\ y' &= -x \end{aligned} \tag{6}$$

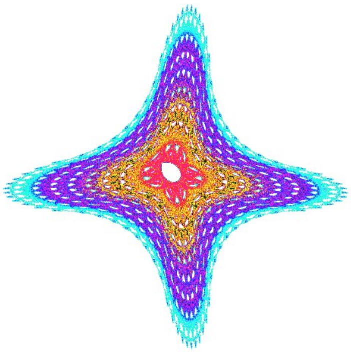
This looked promising (see the illustration opposite top left) but there weren't enough parameters to fiddle with. Eventually he came up with the following functions:

$$\begin{aligned} x' &= y \pm \mathrm{SGN}(x)\sqrt{|bx - c|} \\ y' &= a - x \end{aligned} \tag{7}$$

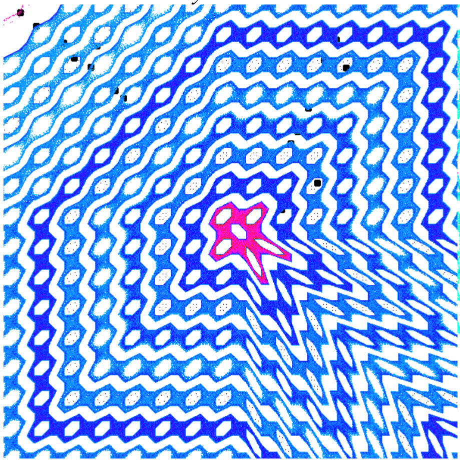
which generates the remarkable fractals opposite.



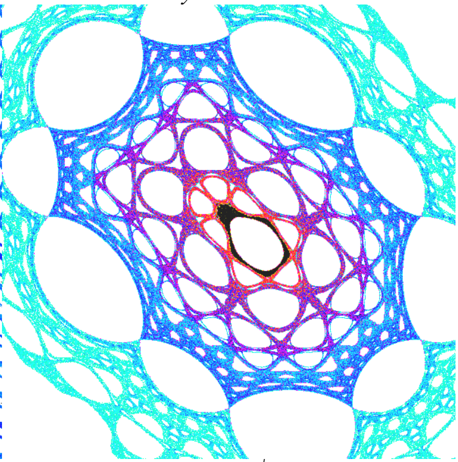
x = y + SGN(x)√|4x|
y′ = – x



x = y + √|2x|
y′ = 1 – x



x = y + ABS(x)
y′ = 1 – x

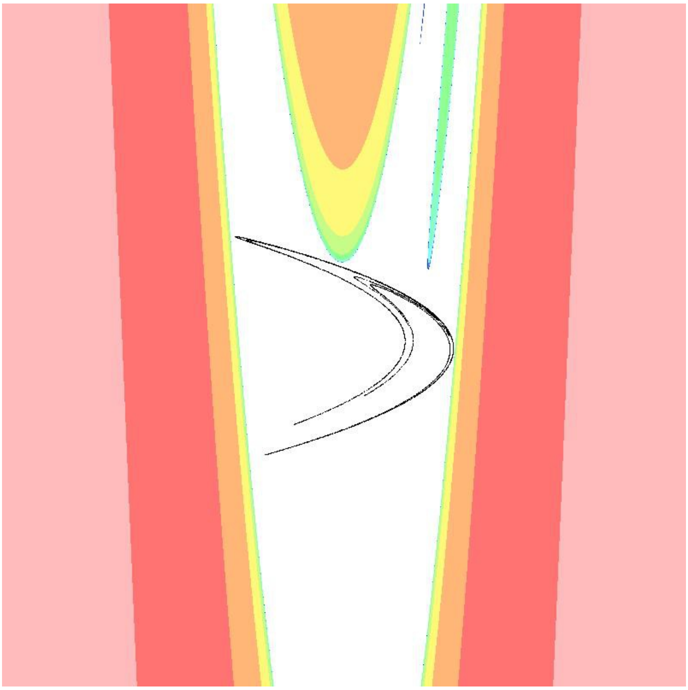


x = y + SGN(x)√|4x – 0.5|
y′ = 1 – x

It is obvious that these images are fractal in the sense that they are full of detail but they cannot be called strange *attractors*. This is because different starting points generally produce different results. In fact, most starting points appear to generate different cyclic patterns. For example, if the starting point is inside one of the empty ovals in the fourth image above, the pattern generated will be a set of 10 ovals within the empty spaces, not a fractal.

Basins of Attraction

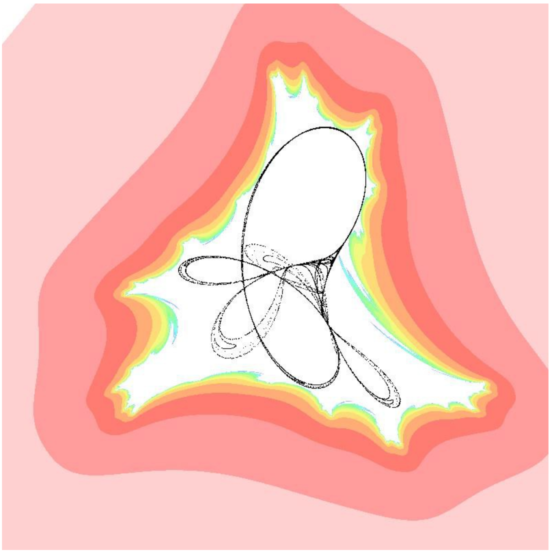
Any attractor, whether strange or periodic, has what is called a 'basin of attraction'. This is the set of all starting points which home in on the attractor. The illustration below shows the basin of attraction of the Hénon attractor.



The basin of attraction of the Hénon attractor

To create the above image, points which wandered off to infinity have been coloured according to the number of iterations needed to reach an arbitrary bailout value. Starting points inside the white region soon converge on the attractor.

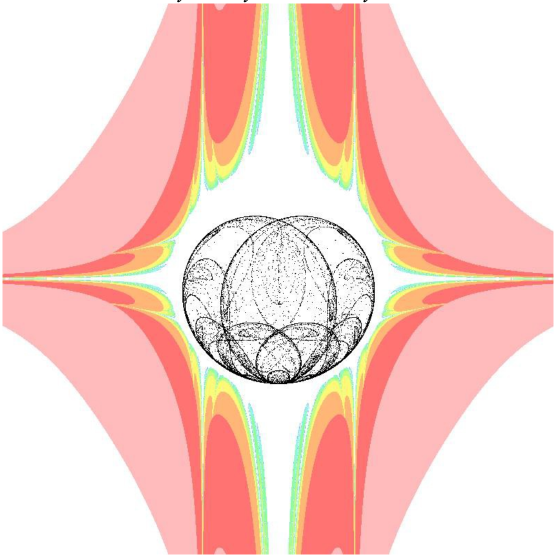
The Hénon attractor has a fairly simple basin but some attractors have fractal basins and the shape of the basin can be as interesting as the fractal itself. Two such basins are illustrated opposite.



The Tinkerbell basin of attraction

$x' = (x^2 - y^2) + 0.9x - 0.6y$

$y' = 2xy + 2x + 0.5y$



The Butterfly Wings basin of attraction

$x' = xy$

$y' = x^2 - 1.9$

Constructing Attractors

It seems impossible to predict the shape of an attractor given the set of equations which define it, particularly as sometimes, small changes in the values of the constants in the equation can make dramatic changes to the attractor. There are, however, a few general rules which we can apply.

Obviously, attractors may be moved and scaled using a simple linear transformation, for example, by replacing all instances of x with $ax + b$. Sometimes this technique can be used to simplify a set of equations but this is not always possible

Any set of equations with the following form:

$$\begin{aligned} x' &= x \times f(x^2, y) \\ y' &= g(x^2, y) \end{aligned} \tag{8}$$

will be symmetrical about the Y axis. This is because whether x is positive or negative, y' will always have the same value but the sign of x' will be determined by the sign of x. The butterfly wings attractor is a good example.

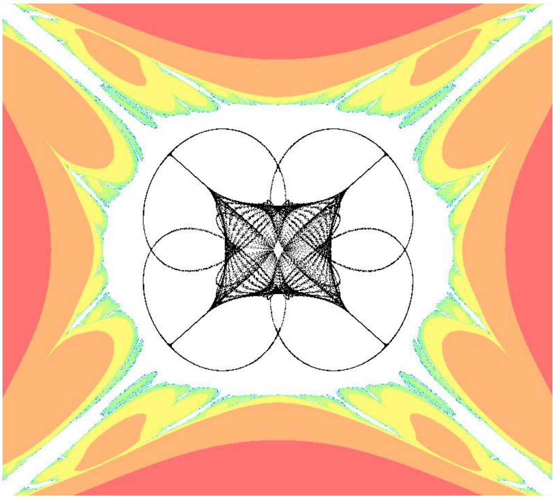
Likewise, to construct an attractor which is symmetrical about the X axis you need the following set:

$$\begin{aligned} x' &= f(x, y^2) \\ y' &= y \times g(x, y^2) \end{aligned} \tag{9}$$

To generate an attractor with both symmetries, use:

$$\begin{aligned} x' &= x \times f(x^2, y^2) \\ y' &= y \times g(x^2, y^2) \end{aligned} \tag{10}$$

The Spider's Web attractor shown opposite is a lovely example, particularly when it is rotated in 3D.

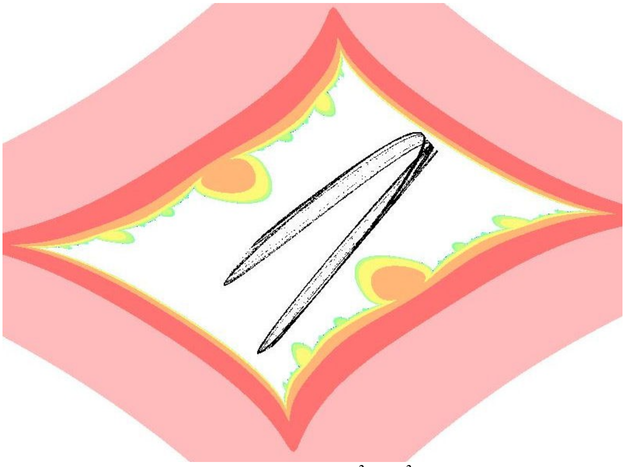


The Spider's Web basin of attraction

$x' = x\,(1.2 - x^2 + y^2)$

$y' = y\,(1.2 + x^2 - 1.5y^2)$

Is is fairly obvious that the basin of attraction of any attractor with a degree of symmetry will share that symmetry. The converse, however, is not true. In the following case, it is clear that the basin of attraction can still have rotational symmetry even when the attractor itself is asymmetrical:



$x' = 1.6 - 0.4\,x^2 - y^2$

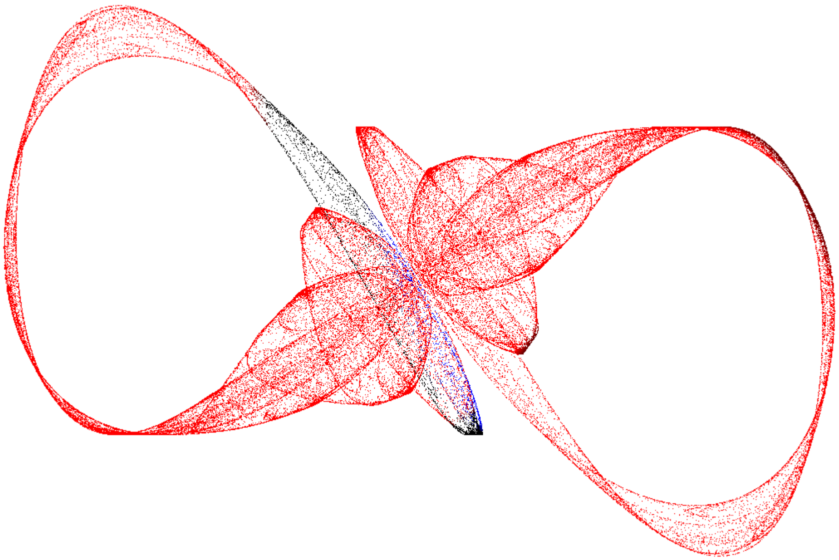
$y' = 1 - 1.5\,xy$

The reason why the basin has rotational symmetry is because after the first iteration, both the point (x, y) and the point $(-x, -y)$ will jump to the same point. This will happen whenever all the terms are even functions of x and y.

To construct an *attractor* with rotational symmetry it is necessary that the two points (x, y) and $(-x, -y)$ will follow the same orbit with reversed signs. This can be achieved by using *odd* terms only in both equations – i.e. terms like x, x^3, xy^2 etc. and no constants.

Here is a lovely example with a particularly simple set of equations called the Butterfly knot. Its equations are:

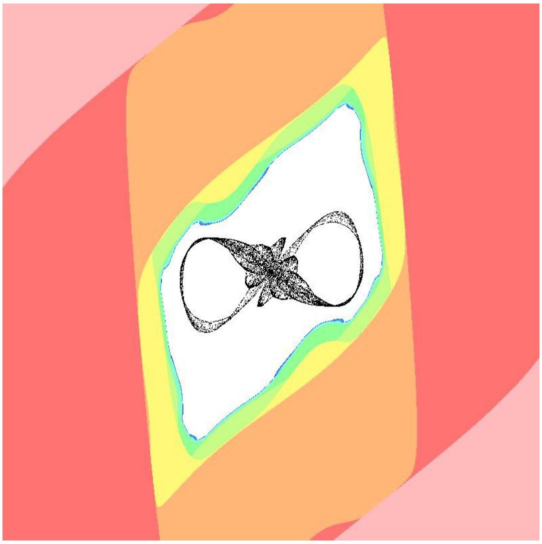
$$\begin{aligned} x' &= -0.8x + y \\ y' &= x(1.2 - x^2) \end{aligned} \tag{11}$$



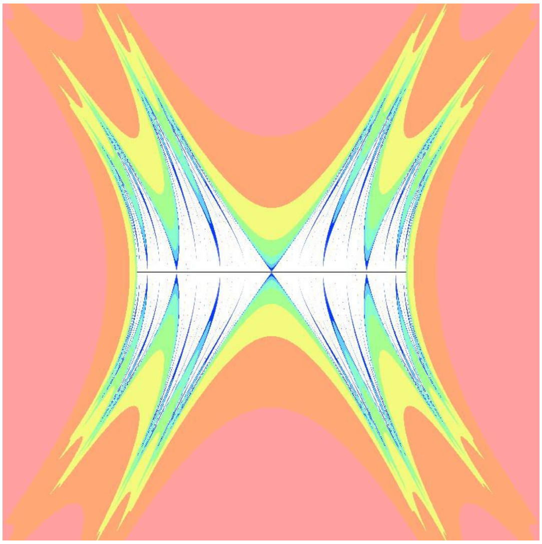
The Butterfly Knot

Who would not like to receive a birthday gift tied with this ribbon!

It cannot be said to have a very interesting basin of attraction though.


The basin of attraction of the Butterfly Knot

In the following case, the attractor is just a thin line but the basin of attraction has a fine fractal structure.

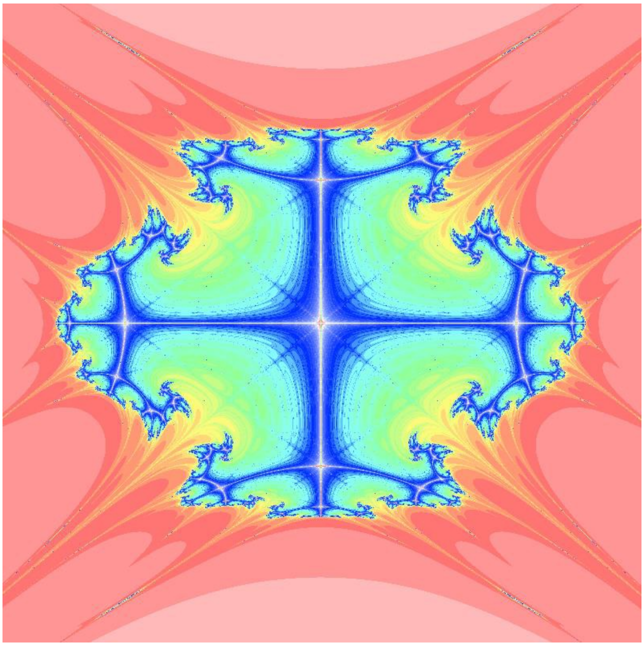

The basin of attraction of the function

$$x' = -1 + 2x^2 - y^2$$
$$y' = xy$$

Chaotic Basins of Attraction

Generally speaking basins of attraction are fairly well-behaved. They may have rather crinkly edges and curious shapes, but they do not often exhibit much fractal behaviour.

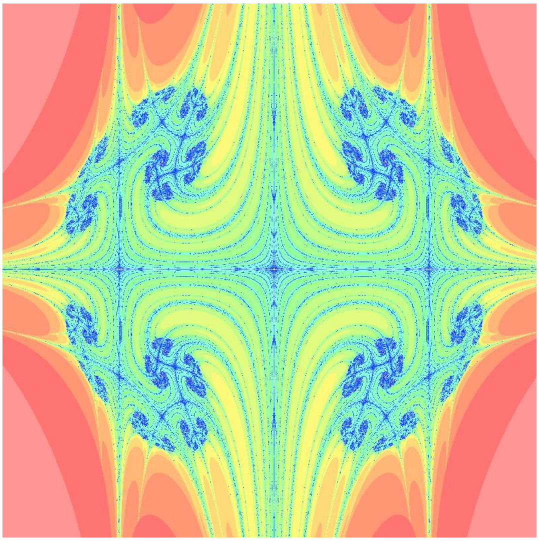
But even the attractors like the Tinkerbell and Spider's Web can show interesting basins of attraction if you alter one of the parameters a bit to push it into a chaotic mode. For example – if you change one of the parameters of the Spider's Web attractor, the attractor itself disappears and is replaced by the fractal structure pictured below. (Strictly speaking, I suppose these should be called 'regions of repulsion' not 'basins of attraction' because there is no attractor – all points rush off to infinity more or less quickly.)



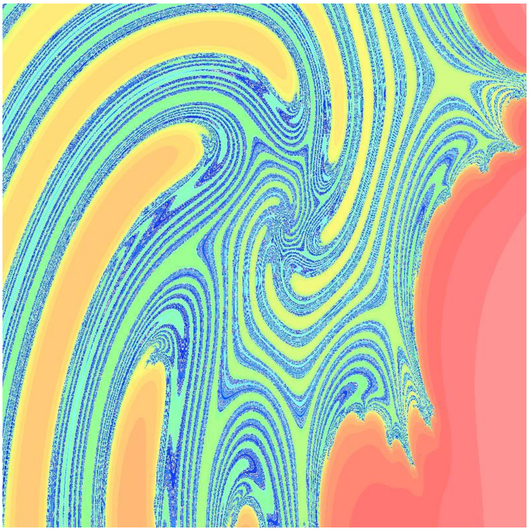
The Spider's Web Fractal basin of attraction

$x' = x\,(1.4 - x^2 + y^2)$

$y' = y\,(1.2 + x^2 - 1.5y^2)$



The Butterfly Wings fractal basin of attraction when
$x' = xy$ *and* $y' = x^2 - 2.1$



A detail of the Tinkerbell fractal basin of attraction when
$x' = (x^2 - y^2) + 1.1x - 0.6y$
$y' = 2xy + 2x + 0.5y$

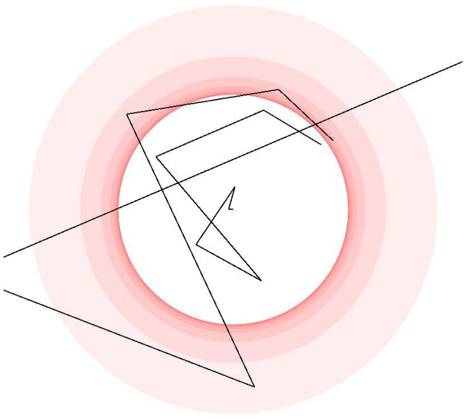
The Standard Function

Of all the hopalong functions, the simplest and most widely studied is the following:

$$\begin{aligned} x' &= a + x^2 - y^2 \\ y' &= b + 2xy \end{aligned} \tag{12}$$

where a and b can take any values.

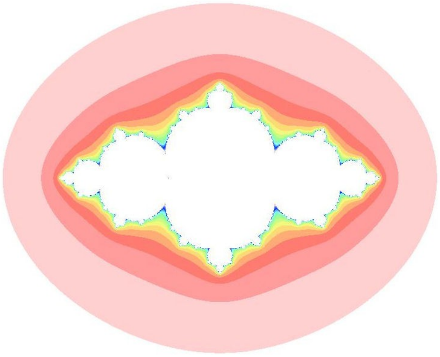
When a and b are both zero, the basin of attraction is the unit circle and the attractor is a single point at the origin.



The basin of attraction of the standard function when a = 0 and b = 0
Two typical orbits are shown, one originating inside the unit circle, the other outside.

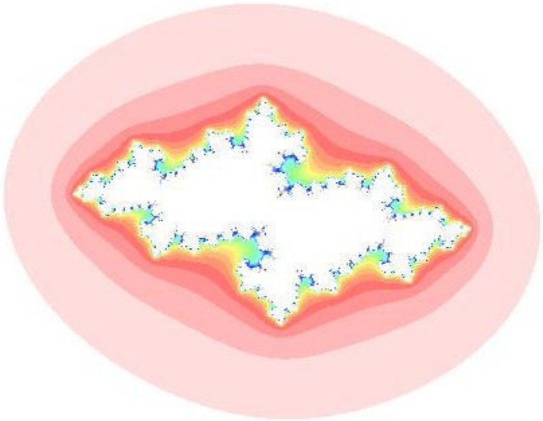
As a and b are varied, the perimeter of the basin of attraction takes on a variety of interesting fractal shapes. For example, as a is increased in the negative direction, the circle elongates and is gradually pinched into a series of approximately circular lobes.

(The reason for the bilateral symmetry of the basin is that, provided $b = 0$, changing the sign of x or y only changes the sign of y' – not its magnitude.)



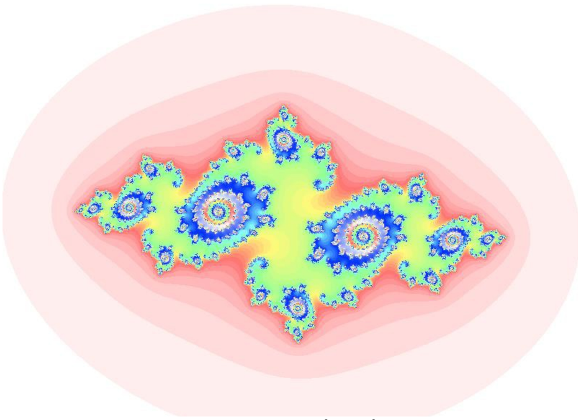
$x' = -0.7 + x^2 - y^2$
$y' = 2xy$

Setting b to something other than zero has the effect of skewing the basin so that it no longer has bilateral symmetry, only 1 rotational symmetry:



$x' = -0.7 + x^2 - y^2$
$y' = 0.25 + 2xy$

If either *a* or *b* stray too far away from the origin, the attractor disappears and the basin of attraction breaks up into fragments. There is still an infinite set of quasi-periodic points (i.e. points which do not escape to infinity) inside this image so the Julia set is still there; it is just too small to see. Its presence is, however, indicated by the lovely spirals which have infinite depth.



$x' = -0.75 + x^2 - y^2$
$y' = 0.18 + 2xy$

You will have noticed that the shape of these basins is similar to the Julia sets which we plotted on page 44. But why is this?

The Julia sets on page 44 were generated by a rather complicated algorithm which involved iterating the complex function

$$z' = \sqrt[2]{z - c} \tag{13}$$

The result of this operation is that all points in the plane home in on a strange attractor – the Julia set.

Now the reverse of this function is

$$z' = z^2 + c \tag{14}$$

and the result of this operation is that many points in the plane will diverge to infinity, but with certain small values of ***c***, some points home in on an attractor.

It is particularly easy to convert this complex equation into a hopalong function. If $z = x + \mathbf{i}y$ then $\mathbf{z}^2 = x^2 + 2\mathbf{i}xy - y^2$ and hence we get:

$$\begin{aligned} x' &= a + x^2 - y^2 \\ y' &= b + 2xy \end{aligned} \tag{15}$$

which is equation (12).

The following pseudo-code shows how easy it is to implement this algorithm on a computer.

```
DIM a = ***, b = ***                          set a and b to a point in the plane

FOR q = 0 TO screenheight
    FOR p = 0 TO screenwidth
        DIM: count=0
        (x, y) = complex(p, q)                                   set (x, y) to start
        REPEAT
            DIM tempx = x                                         keep a copy of x
            x = a + $x^2 - y^2$                                           update x
            y = b + 2 × tempx × y                                        update y
            count = count + 1
        UNTIL EITHER count = 1000 OR $x^2 + y^2$>10000
        IF count=1000 THEN
            CALL : PlotPoint(p, q, Colour.Black)
        OTHERWISE
            CALL : PlotPoint(p, q, Colour(Count))
    NEXT
NEXT

DEFINE complex(p, q)
    ***          return a complex number corresponding the the point (p, q)
END_DEFINITION

DEFINE PlotPoint(a, b, colour)
    ***           Plot a point on the screen corresponding to (a, b) in colour
END_DEFINITION

DEFINE Colour(c)
    RETURN ***                                           return a suitable colour
END_DEFINITION
```

The bailout value of 10000 is quite arbitrary, as is the maximum value of the count. It may be assumed that if the point survives for 100 iterations, it has found the attractor. Likewise, if it reaches a point which is more than 100 units from the origin, it may be assumed to be heading off to infinity.

Note the use of a temporary value for *x*. It is essential that, when calculating *y,* the original value of *x* is used, not the new one.

Reverse Hopalong Attractors

As we have seen, the boundary which separates the two basins of attraction is none other than the Julia set created by the reverse function. This suggests that we should be able to find interesting Julia sets by iterating the reverse functions of the hopalong functions which we have discovered.

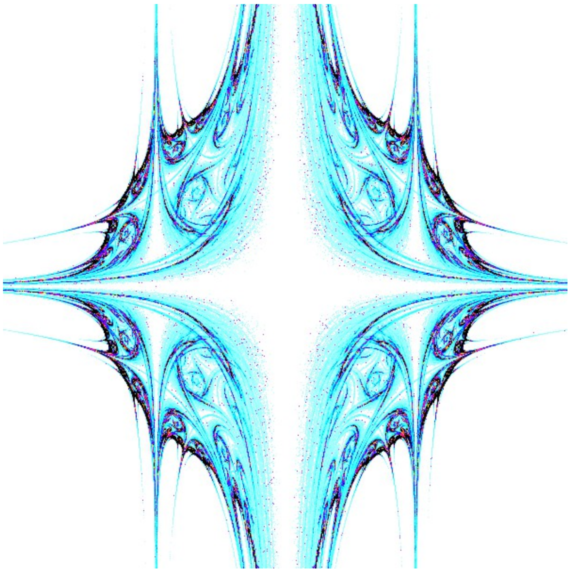
Unfortunately, this is not always easy. One function which is easy to reverse is the Butterfly Wings attractor whose equations are:

$$\begin{aligned} x' &= xy \\ y' &= x^2 - 1.9 \end{aligned} \tag{16}$$

The reverse of this is:

$$\begin{aligned} x' &= \pm\sqrt{y + 1.9} \\ y' &= \pm\frac{x}{\sqrt{y + 1.9}} \end{aligned} \tag{17}$$

As with the earlier Julia sets, we must choose either the positive of the negative square root at random. This is the astonishing result:



The reverse Butterfly Wings attractor

The outline of the basin of attraction shown on page 53 is clear – but unlike the previous case, the interior is not empty. The reason for this is that, because we are choosing a square root at random, we are exploring all possible routes to the attractor – not just the reverse of the route which would be taken by the original function. Nor is there any guarantee that a point which actually starts on the Julia set will remain there. On the other hand, we can be sure that all points reached by this reverse function will remain within the boundary defined by the Julia set.

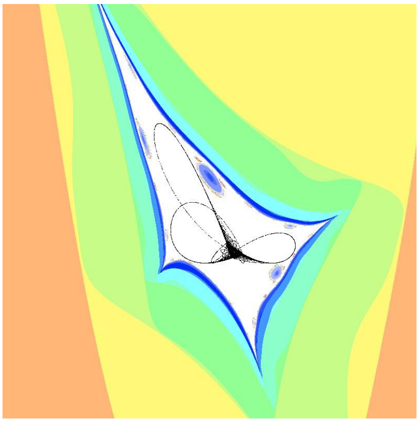
Other suitable examples are not easy to find. Some function appear to wander about within the basin without appearing to migrate to the Julia set at all. Here is one which does trace the expected outline.

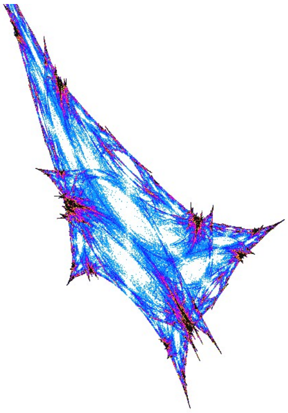
The forward functions are:

$$\begin{aligned} x' &= 0.1 - 1.1(x + y) \\ y' &= x^2 + 1 \end{aligned} \tag{18}$$

and the reverse functions are:

$$\begin{aligned} x' &= \pm\sqrt{y + 1} \\ y' &= \frac{0.1 - x}{1.1} \pm \sqrt{y + 1} \end{aligned} \tag{19}$$


Forward hopalong attractor


Reverse hopalong attractor

The Mandelbrot Map

We now come to the most famous fractal of them all – the Mandelbrot map.

We saw on page 44 how the action of repeatedly subtracting a constant and taking the square root of a complex number results in a Julia set with a familiar fractal shape. We also noted that for certain constants, the Julia set is connected. What we did not point out at the time was that if the Julia set is connected, then all points inside the boundary transform into points inside the boundary, and, of course, all points outside it transform into points outside it. The reason for this is that, as we have seen, a Julia set marks the boundary between different basins of attraction of the reverse function. It is often a good idea to emphasize the different regions by filling the interior of a connected Julia set in a different colour. This is called a 'filled Julia set'. All the points of a filled Julia set are stable under iteration of the reverse function.

What is this reverse function? It is, of course:

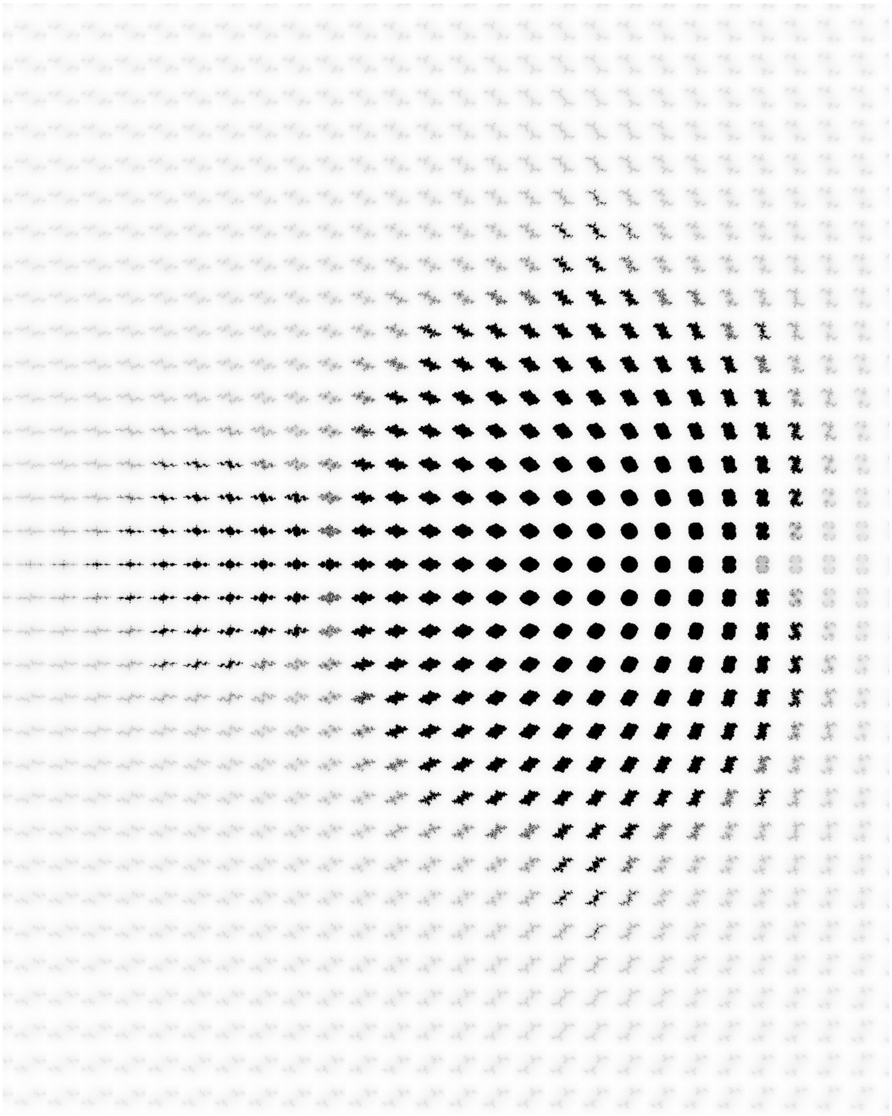
$$\boldsymbol{z}' = \boldsymbol{z}^2 + \boldsymbol{c} \tag{20}$$

which may easily be turned into a hopalong function in x and y coordinates:

$$\begin{aligned} x' &= a + x^2 - y^2 \\ y' &= b + 2xy \end{aligned} \tag{21}$$

Now lets draw a large collection of filled Julia sets for different values of $\boldsymbol{c}$ clustered round the origin. The result is shown opposite. All the connected sets lie inside a large heart-shaped blob and the faint outline of the familiar Mandelbrot Set can be seen emerging.

The question now arises – how can we easily distinguish those Julia sets which are filled from those which are not? A close examination of the filled Julia sets will reveal that, whenever the set is connected, the starting point of (0,0) (called the 'critical point') will be stable – but when the set is disconnected, this point is always unstable.

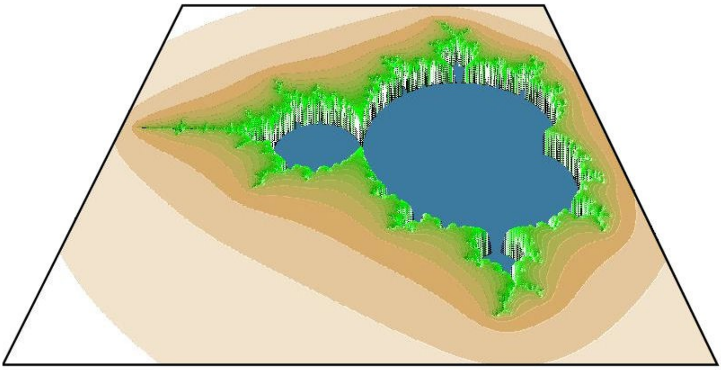


A map of all the filled Julia sets of the reverse function $\mathbf{z'} = \mathbf{z}^2 + \mathbf{c}$ *round the origin.*

This suggests a simple method of plotting the map in detail – just plot a black dot at every point ***c*** in the complex plane for which the starting point (0, 0) fails to diverge off to infinity. The result is shown opposite and it is immediately clear that we have discovered something quite special. The main body of the set is a cardioid but numerous lobes sprout off its edge and there also appear to be a few isolated blobs here and there.

In order to reveal even more detail in the region just outside the set we colour these pixels according to the length of time taken for ***z*** to escape beyond an arbitrary bailout value. The result is no less stunning for being so familiar.

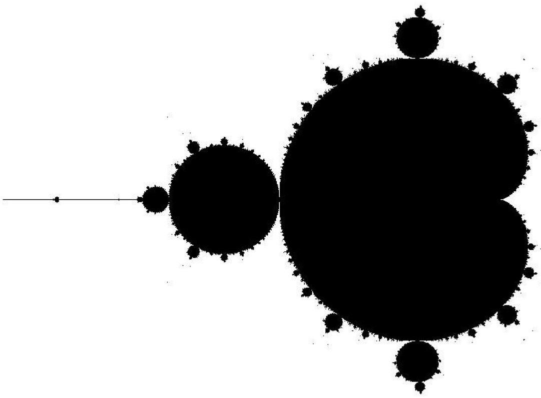
The stable region can be imagined to be at the bottom of a deep crater. Precipitous slopes tower round the shores of the lake while the whole is set in a featureless upland plateau.



The Mandelbrot Map in 3D

It is essential to realise that while there is a different Julia set for every point in the complex plane – but there is only one Mandelbrot Map.

The pseudo-code for a simple Mandelbrot program is listed opposite. It is worth comparing this code with that on page 63. Basically the only difference is that in the Julia code the constant (a, b) is fixed and (x, y) is set to the screen coordinates while in the Mandelbrot code the constant (a, b) is set to the screen coordinates and (x, y) is set to zero.



*The Mandelbrot Set – i.e. the set of all values of **c** which result in a connected Julia set*

```
FOR p = 0 TO screenheight
    FOR q = 0 TO screenwidth
        DIM: count=0
        (x, y) = (0, 0)                                     set (x, y) to (0, 0)
        (a, b) = complex(p, q)       set constant (a, b) to the screen coordinates
        REPEAT
            DIM tempx = x                                        keep a copy of x
            x = a + $x^2 - y^2$                                          update x
            y = b + 2 × tempx × y                                        update y
            count = count + 1
        UNTIL EITHER count = 1000 OR $x^2 + y^2$>10000
        IF count = 1000 THEN
            CALL : PlotPoint(p, q, Colour.Black)
        OTHERWISE
            CALL : PlotPoint(p, q, Colour(Count))
    NEXT
NEXT

DEFINE complex(p, q)
    ***         return a complex number corresponding the the point (p, q)
END_DEFINITION

DEFINE PlotPoint(a, b, colour)
    ***          Plot a point on the screen corresponding to (a, b) in colour
END_DEFINITION

DEFINE Colour(c)
    RETURN ***                                         return a suitable colour
END_DEFINITION
```

The Critical Point

An important question must now be answered. What is so special about the point (0, 0)? Why do we have to start at this point and what happens if we don't?

Let us review what actually happens when a starting point z_0 is iterated using a certain constant $\boldsymbol{c}$. The first iteration will take it to a point $z_1 = f^1(z_0)$; the second to $z_2 = f^2(z_0)$ etc. These points can be referred to as the subsequent 'images' of z_0.

Now any given starting point z_0 may be classified as follows[3] (the terminology is my own):

- Points which eventually diverge off to infinity are called 'open'
- Points which do not diverge off to infinity are called 'closed'
- If a closed point is surrounded by other closed points, then the point is said to be 'stable'; if, however, points nearby are open, then the point is said to be 'unstable'.
- If a point is part of a finite periodic cycle then it is called 'periodic'.
- If a point is both closed, stable and periodic then it is called 'superstable'.

The first thing to point out is that for *any* value of $\boldsymbol{c}$ there is always at least one closed point which instantly maps onto itself – the solution to the equation $f^1(\boldsymbol{Z}) = \boldsymbol{Z}$. There are also many periodic points. e.g. the solutions to the equation $f^3(\boldsymbol{Z}) = \boldsymbol{Z}$ will have periodicity 3. In general, however, these points will be unstable. We are only interested in the points which are not only closed but stable. If $\boldsymbol{c}$ lies outside the Mandelbrot set, the Julia set is disconnected cantor dust and there are no stable points.

3 This list is not comprehensive. There are points called Misiurewicz points which are closed and unstable but which lead to periodic cycles and there are (I believe) points which are closed and unstable but which have no periodicity at all and behave chaotically.

Now if $\boldsymbol{c}$ is inside the main cardioid of the Mandelbrot set, then there exists a unique superstable point $\boldsymbol{Z}$ where $f^1(\boldsymbol{Z})$ is equal to $\boldsymbol{Z}$. Obviously if we were to start at $z_0 = \boldsymbol{Z}$, the point would be instantly stable but we clearly cannot always do this because $\boldsymbol{Z}$ depends on $\boldsymbol{c}$. On the other hand, provided we start reasonably close nearby, z will home in on $\boldsymbol{Z}$. The set of all points which home in on $\boldsymbol{Z}$ is, of course, the filled Julia set appropriate to the value $\boldsymbol{c}$ – its 'basin of attraction'.

If $\boldsymbol{c}$ lies inside the secondary lobe 2 (the biggest secondary lobe) then there is no single point which is stable. There is, however, a pair of points $\boldsymbol{Z}_1$ and $\boldsymbol{Z}_2$ such that $f^2(\boldsymbol{Z}) = \boldsymbol{Z}$. Similarly, in lobes with periodicity n there will by a cycle of stable points such that $f^n(\boldsymbol{Z}) = \boldsymbol{Z}$.

We cannot work out in advance which points are super-stable (except by solving a potentially infinite number of high order polynomial equations) but there are plenty of other points which home in on the stable cycle. The question is – of all these stable points, is there one which is stable for all values of $\boldsymbol{c}$ – i.e. is there a point which is common to all the connected Julia sets? And if there is, why does it have this property?

In my companion volume 'Chaos and the Logistic Equation' I showed that a point is only stable if the absolute value of the gradient of the function is less than 1 at that point. I also showed that the gradient of the nth order function at any point z_0 is equal to the product of the gradients of the basic function at the first n iterates of z_0. For example: suppose a function $z_1 = f^1(z_0)$ (with a constant c) causes z_0 to iterate to z_1, z_2 etc., the gradient of the third order function $f^3(z_0)$ will be equal to the product of the gradients of the first order function at z_0, z_1 and z_2.

$$\left[\frac{d\,f^3(z)}{d\,z}\right]_{z_0} = \left[\frac{d\,f^1(z)}{dl\,z}\right]_{z_0} \times \left[\frac{d\,f^1(z)}{d\,z}\right]_{z_1} \times \left[\frac{f^1(z)}{d\,z}\right]_{z_2} \qquad (22)$$

The condition that the absolute value of this parameter must be be less than 1 can always be met by choosing z_0 to be such that $\left[\frac{d\,f^1(z)}{d\,z}\right]_{z_0} = 0$. It follows that, if we start with z_0 at the place where the gradient of the function $z_1 = f^1(z_0)$ is zero, the gradient of all the higher order functions will be zero at $z = z_0$ and that, if there exists a nearby point $\boldsymbol{w}$ where $f^n(\boldsymbol{w}) = \boldsymbol{w}$ and where $\left| \left[\frac{d f^1(z)}{d z} \right]_w \right| < 1$, z is going to home in on it. This point (i.e. the point z_0 where the gradient of the first order function is zero) is called the *critical point* of the function.

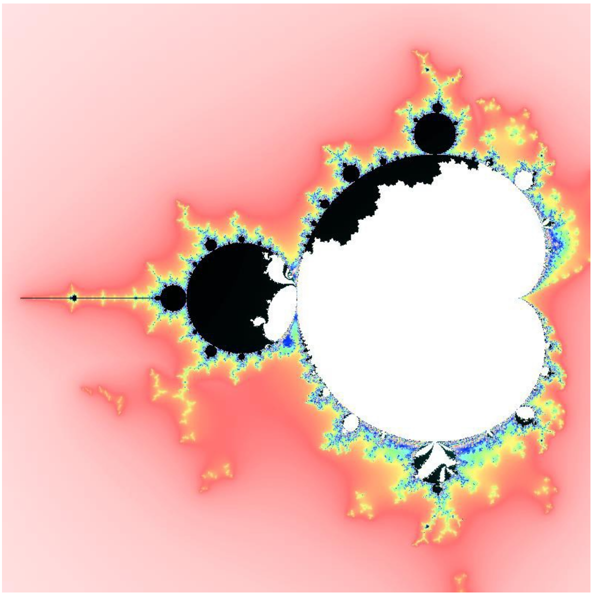
In the case of the standard Mandelbrot function $z_1 = z_0^2 + \boldsymbol{c}$ the point where the gradient is zero will be when $\frac{d z_1}{d z_0} = 2 z_0 = 0$ – i.e. $z_{\text{crit}} = (0, 0)$

For example, suppose that $\boldsymbol{c} = (-0.9, 0)$. (this point is just inside the secondary lobe 2 on the principal axis.) The following table shows what happens to z_0 (= 0) for the next few iterations. $G_1(z)$ is the gradient of $f^1(z)$ (= $2z$) at z and $G_2(z)$ is the gradient of the second order function $f^1(z)$ (= $4z(\mathbf{z}^2 - 0.9)$ (all the imaginary components are zero.)

	z	$G_1(\boldsymbol{z})$	$G_2(\boldsymbol{z})$
0	0.000	0.000	0.000
1	-0.900	-1.800	0.324
2	-0.090	-0.180	0.321
3	-0.892	-1.784	0.373
4	-0.105	-0.209	0.372
5	-0.889	-1.778	0.390
6	-0.110	-0.219	0.389
7	-0.888	-1.776	0.396
8	-0.111	-0.223	0.396
9	-0.888	-1.775	0.398
10	-0.112	-0.224	0.398
11	-0.887	-1.775	0.399
12	-0.113	-0.225	0.399
13	-0.887	-1.775	0.400

The first column shows z gradually homing in on a period 2 cycle. The gradient of the first order function oscillates up and down so the point is prevented from settling on a single value; but the gradient of the second order function is much better behaved and since it is always less than 1, z will home in on the two points where $f^2(\boldsymbol{w}) = \boldsymbol{w}$.

The following image shows the sort of things which happen if we do not start at the point (0, 0). The stable region of the map (shown in white) is distorted and certain values of ***c*** which were stable now become unstable. It is only when we start at the point (0, 0) that the stable area is maximum and there are no values of ***c*** which are unstable in the standard map which become stable when a different starting point is used.


Distorted map starting at (0.25, 0.5)

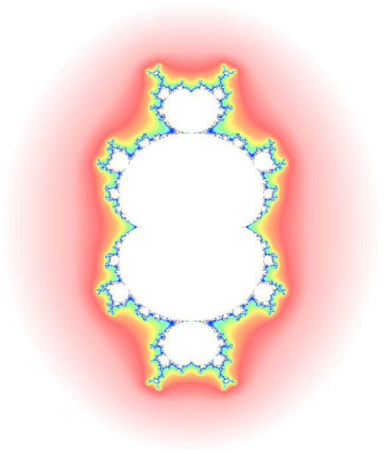
Cubic and higher order functions generally have more than one place where the gradient is zero and, in consequence, they have more than one critical point. If this is the case, the Mandelbrot set is usually taken to be the set of all the points where at least one of the critical points is stable.

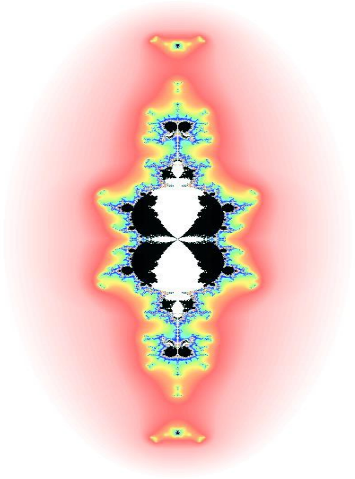
There is a lot more to be said about the Mandelbrot Map. For more detail see my companion volume 'The Mandelbrot Map – a layman's guide'.

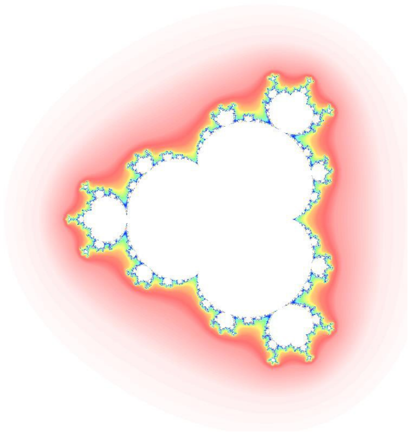
Mandelbrot Maps of Other Functions

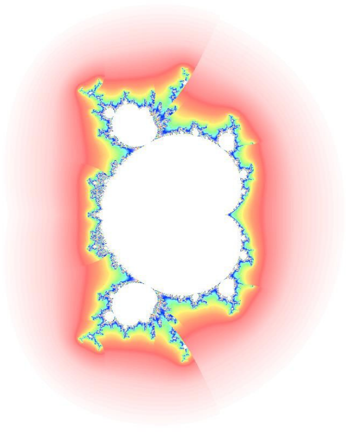
Here is a gallery of Mandelbrot Maps of some different functions.

The cubic equation has two critical points. In the second illustration, points which are stable from both are coloured white, points which have only one stable starting point are coloured black.

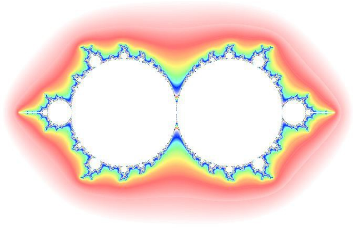

The cubic Mandelbrot map
$z' = z^3 + c$


A different cubic map
$z' = z^3 + z + c$

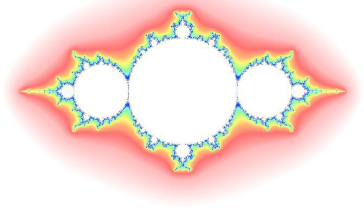

The quartic Mandelbrot map
$z' = z^4 + c$


Mandelbrot 2.5
$z' = z^{2.5} + c$

Note that in the case of the Mandelbrot 2.5 map (i.e. $\boldsymbol{z}' = \boldsymbol{z}^{2.5} + \boldsymbol{c}$) then map has flaws in it. This is because there is always ambiguity when calculating fractional powers.

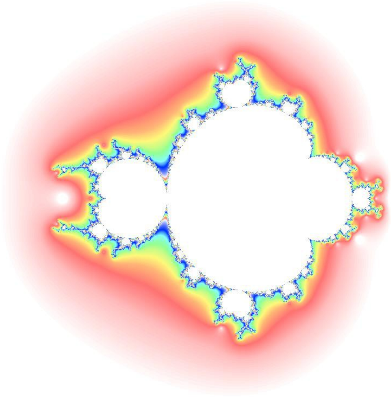


The quadratic logistic map
$z' = cz(1 - z)$

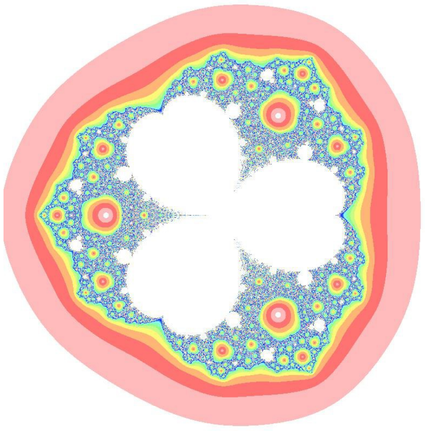


The cubic logistic map
$z' = cz(1 - z^2)$

I call these 'logistic' maps because of the analogy with the 'Logistic Equation' $x' = Ax(1 - x)$.

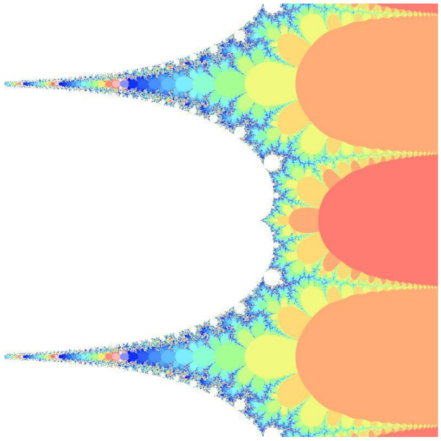


A quadratic / reciprocal map
$z' = z^3/(z - 1) + c$

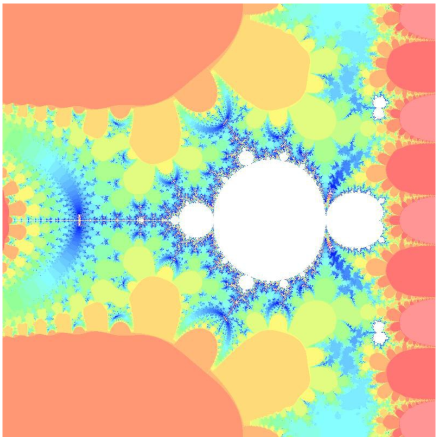


The trefoil map
$z' = c(z^2 + 1/z)$

The trefoil map appears to have regions of almost total chaos.

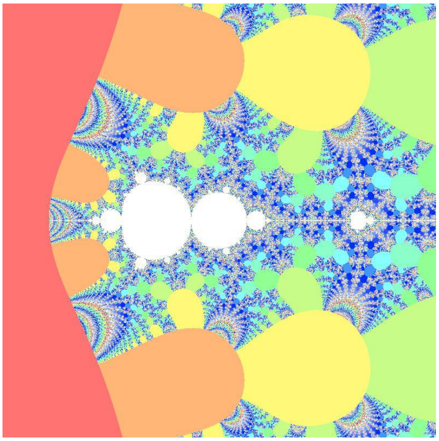


The exponential map
$\boldsymbol{z'} = e^z - Rz + \boldsymbol{c}$ (R = 1)



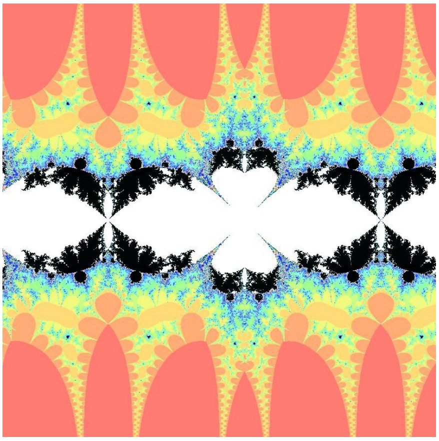
A different exponential map
$z' = e^z - Rz + \boldsymbol{c}$ (R = 1.2)

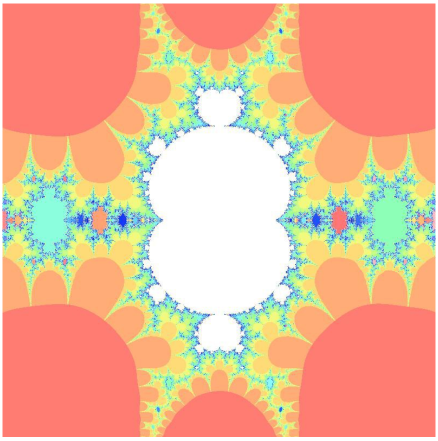
The exponential map $z' = e^z + \boldsymbol{c}$ does not have a critical point but stable regions exists if we subtract a factor Rz. When R <= 1, the stable region comprises most of the negative half of the plane but when R>1 the stable region reduces to a Mandelbrot-style blob. Amazingly, perfectly formed minibrots can be found everywhere in both images.



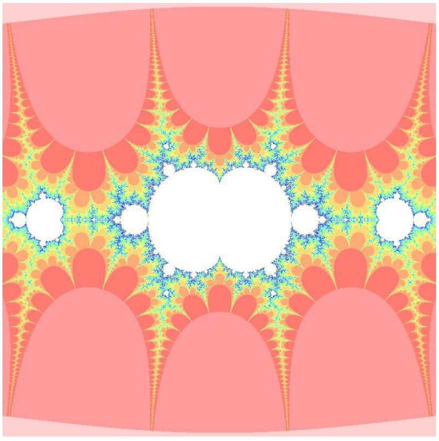
A different exponential map
$\boldsymbol{z'} = \boldsymbol{cz}e^z$

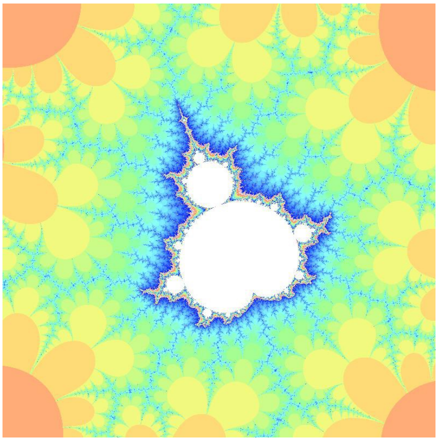
The formula $\boldsymbol{z'} = \boldsymbol{cz}e^z$ generates a slightly different map. The prominent lobes are not genuine features of the map; they are produced by the necessarily finite value of the bailout value.


A cosine map
$z' = \cos(z) + \boldsymbol{c}$


A different cosine map
$z' = z + \cos(z) + \boldsymbol{c}$

Maps based on trig functions like cosine generally repeat along the real axis. Many have multiple critical points.


A different cosine map
$z' = \boldsymbol{c} \cos(z)$


A minibrot in the same map
$z' = \boldsymbol{c} \cos(z)$

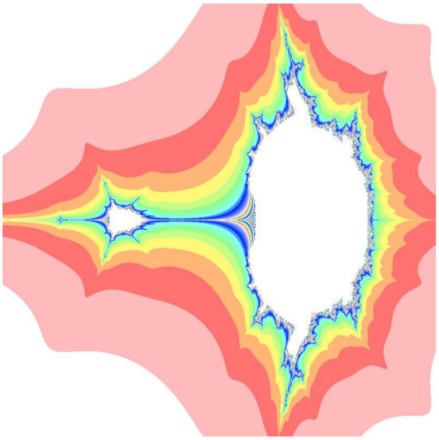
Minibrots like the one illustrated above can be found in almost all Mandelbrot maps. I suspect that this is because, on a sufficiently small scale, all curves look like parabolas so where there is a value of ***c*** where the orbit of the critical point is stable, nearby values of ***c*** will essentially behave as if they are generated by a simple quadratic function.

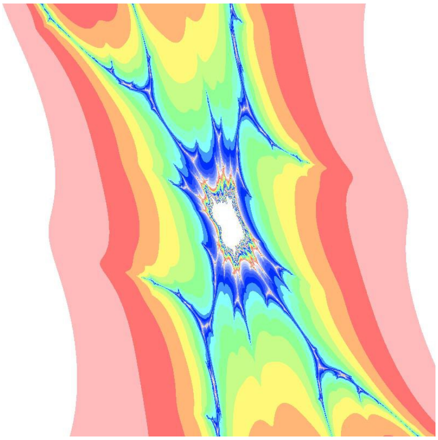
Non-standard Mandelbrot Maps

When translated into x and y coordinates, the standard Mandelbrot algorithm becomes a simple two dimensional hopalong function:

$$\begin{aligned} x' &= x^2 - y^2 + p \\ y' &= 2xy + q \end{aligned} \tag{23}$$

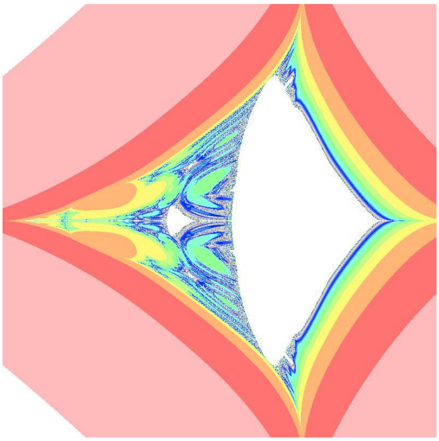
It is natural to ask, what do the Mandelbrot maps look like for functions of x and y which are not simply translations of complex polynomials. For example, what does the Mandelbrot map look like if we change that 2 into a 4? Here it is:

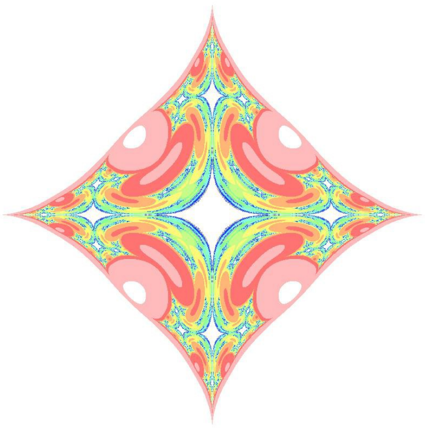

Modified Mandelbrot


Modified Mandelbrot (detail)

While it still has the basic structure of the Mandelbrot map and it appears to be differentiable (that is to say, there are no places where the depth algorithm produces a sudden large change), it does not have the integrity or beauty of the original.

Changing the $- y^2$ into $+ y^2$ distorts the map in a different way and introduces apparently chaotic regions (i.e. the map is not differentiable). The Julia set iullustrated corresponds to a point in the main 'minibrot' – i.e. inside the small white sector to the left of the main area of stability.


Modified Mandelbrot

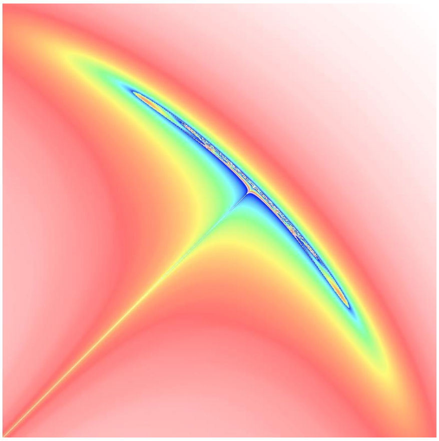

Julia Set

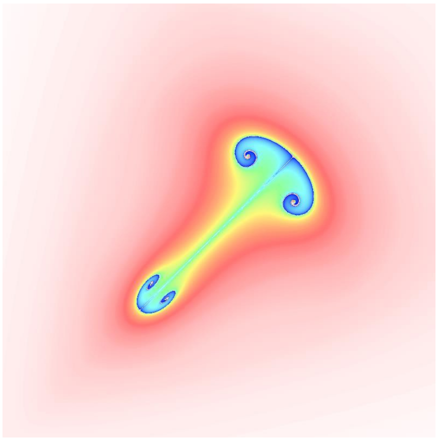
One of the problems is that for a function to produce a viable Mandelbrot map, it should, ideally, have one critical point. i.e. a place where the gradient of the function is zero. One such function is:

$$\begin{aligned} x' &= x^2 + y + p \\ y' &= x + y^2 + q \end{aligned} \tag{24}$$

and its critical point is at the origin.

It has the following map but it has no stable basins of attraction. Its Julia sets are quite attractive.


Non-standard map


Non-standard Julia Set

Newton-Raphson Julia Sets

As we have seen, Julia sets mark the boundary between different basins of attraction. Points inside a filled Julia set home in on a periodic cycle; points outside it shoot off to infinity. On the other hand, there are functions which never shoot off to infinity but which have two or more attractors. One way of creating mappings with more than one basin of attraction is to consider the Newton's method for finding the roots of an equation. This claims that if z is close to one of the roots of an equation $f(z) = 0$, then $z' = z - \dfrac{f(z)}{df/dz}$ will be even closer.

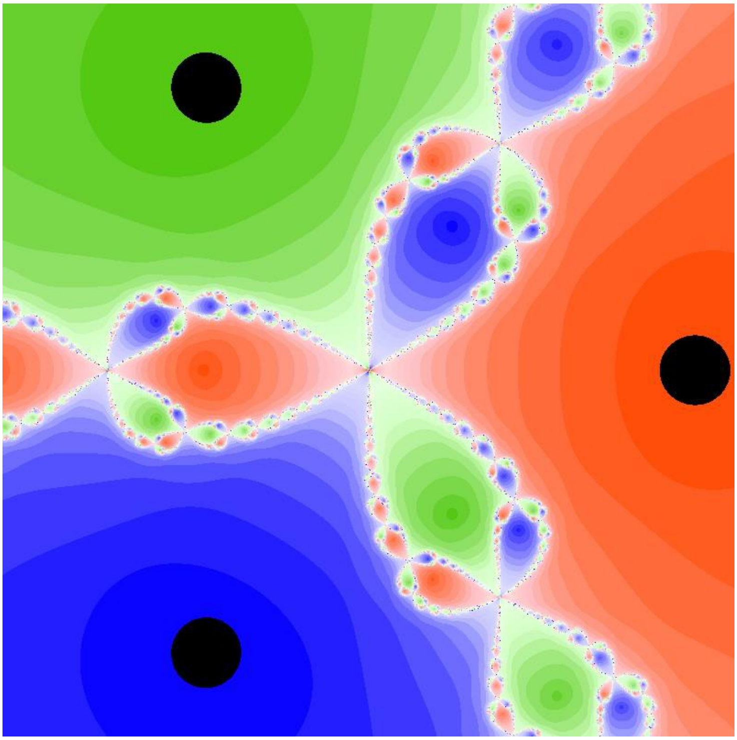
For example, if $f(z) = z^3 - 1$, then $df/dz = 3z$ and our mapping looks like this:

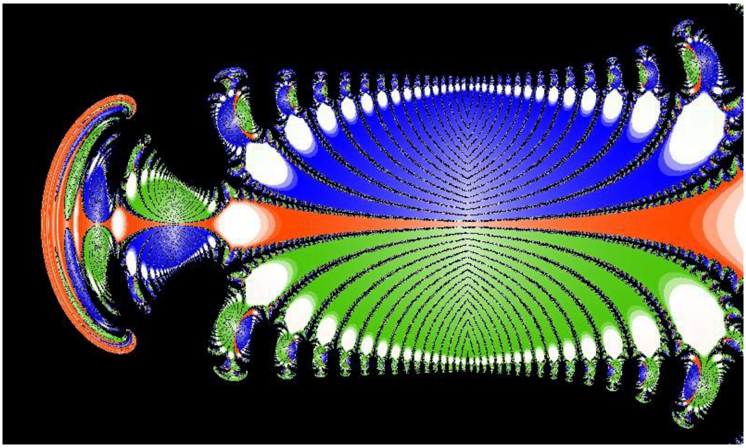
$$z' = z - \frac{z^3 - 1}{3z} = \frac{2z^3 + 1}{2z} \qquad (25)$$

Translating this into x, y coordinates is not that easy and if you wish to program this mapping then it is probably best to use dedicated routines for multiplying and dividing complex numbers. (These are not difficult but are beyond the scope of this little book.)

The result of this mapping is illustrated opposite. The three black dots are the three cube roots of 1. Each basin of attraction is colour coded in red, green and blue. The boundary between the basins of attraction is an infinitely detailed chain of pearls, each with a different colour. The white pixels in the image effectively trace out the Julia set for this mapping. Points on this boundary will take an infinite length of time before deciding which root to go for.

Opposite below is a Julia map for the roots of the equation $e^z = 1$. This equation has an infinite number of roots. The principal one (0, 0) is just off to the right of the picture and points which end there are coloured red. Points which migrate towards the positive roots are coloured blue and points which migrate towards the negative roots are coloured green. (The black regions take too long to compute.)


The Julia map for the Newton-Raphson formula for the cube roots of unity.


The Julia map for the Newton-Raphson formula for the equation $e^z = 1$

Dynamic Systems

All the examples we have looked at so far are examples of *discrete* dynamics. They are systems which jump from one point to the next at discrete intervals. Of more interest to physicists are *continuous* dynamic systems which move smoothly from one state to the next. These are usually characterized by one or more *differential equations*. A simple example is the harmonic oscillator which obeys the following equation which basically says that the acceleration of the oscillator is proportional to the distance from the origin and directed towards the origin:

$$\frac{d^2x}{dt^2} = -\omega^2 x \tag{26}$$

We can equally well write this equation in the form of two first order differential equations like this:

$$\begin{aligned} dx/dt &= y \\ dy/dt &= -\omega^2 x \end{aligned} \tag{27}$$

where y is the velocity of the oscillator.

The solution to this set of equations is well known:

$$\begin{aligned} x &= A\cos(\omega t) \\ y &= A\omega\sin(\omega t) \end{aligned} \tag{28}$$

Now suppose we wish to simulate this behaviour on a computer. To do this we have to calculate what happens to x at discrete time intervals dt. We also have to keep track of the velocity y at each point. The following pseudocode will do the trick:

```
DIM y = ***                              set the initial velocity
DIM x = ***                              set the initial position
DIM ω = ***                              set the angular velocity
DIM dt = ***                             set the time increment

LOOP
    a = -ω² x                            calculate the acceleration
    y = y + a * dt                       update the velocity
    x = x + y * dt                       update the position
ENDLOOP
```

What we have here looks suspiciously like a discrete dynamic system. In fact if we eliminate the variable *a* and simplify things a bit we can see that at heart we have a single linear transformation:

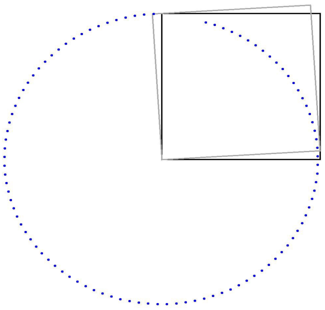
$$\begin{aligned} x' &= x + y\,dt \\ y' &= y - \omega^2 x\,dt \end{aligned} \tag{29}$$

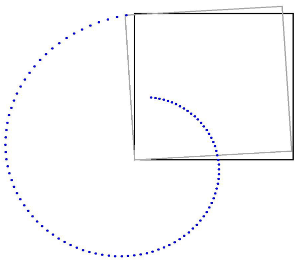
or, to put it in the discrete iterative form with which we are familiar:

$$\begin{aligned} x' &= x + by \\ y' &= y + -bx \end{aligned} \tag{30}$$

where *b* takes the role of the time increment *dt* and ω has been set to 1 for simplicity.

Provided *b* is very small, *x* and *y* will describe a circle.


A circle approximated by a single linear transformation


A single linear transformation with a point attractor

Now as we have seen, this solution does not qualify as an attractor because different starting points generate different circles in the same way that different starting points generate different fractals using the Barry Martin algorithm.

If we introduce a small reduction factor in the *x* and *y* terms, we can generate a spiral. Now all initial points will home in on the origin.

$$\begin{aligned} x' &= 0.99x + by \\ y' &= 0.99y + -bx \end{aligned} \tag{31}$$

as illustrated in the second of the above pictures.

This transform can be put back into the form of equation (12) by splitting off the small difference and incorporating it into the incremental term as follows:

$$\begin{aligned} x' &= x + b(ax + y) \\ y' &= y + b(ay - x) \end{aligned} \tag{32}$$

Now, when *a* is negative, the point will spiral into the origin and when *a* is positive, the point will spiral out to infinity.

That is all we can do with a single linear transformation. Either the point will describe a closed loop or it will spiral into a point or off to infinity. If we want to generate more interesting behaviour, we might first consider using a non-linear transformation.

Equation (14) can be generalised to the following:

$$\begin{aligned} x' &= x + \delta \times f(x,y) \\ y' &= y + \delta \times g(x,y) \end{aligned} \tag{33}$$

where $f(x,y)$ and $g(x,y)$ are other functions of *x* and *y* and δ is a very small constant.

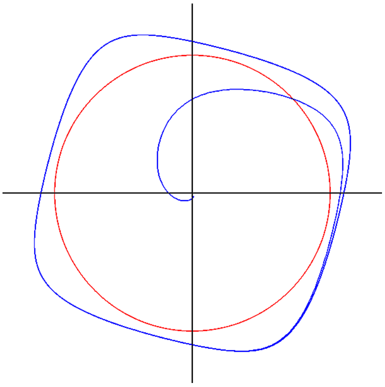
But here we come across a simple consideration. Any functions *f* and *g* which result in genuine attractors can only take the form of periodic loops for the simple reason that in a 2 dimensional system a continuous line can never cross itself. (When it reaches the crossing point, which way will it go?). We can, however, use what we have learned to construct a set of equations which has a periodic loop as an attractor. One possibility is to devise a way of making *a* negative whenever the point strays too far from the origin and positive whenever it wanders too near. A suitable function would be:

$$a = 1-(x^2 + y^2) \tag{34}$$

and the complete transformation would be

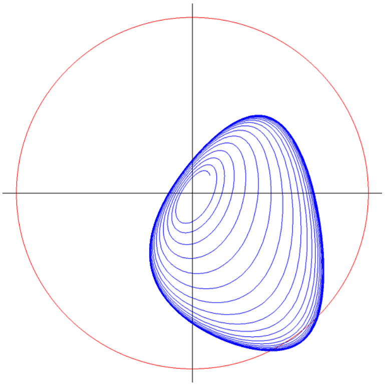
$$\begin{aligned} x' &= x + \delta \times b((1 - (x^2 + y^2))x + y) \\ y' &= y + \delta \times b((1 - (x^2 + y^2))y - x) \end{aligned} \tag{35}$$

This is, in fact, a cubic equation and it generates a circular periodic attractor whose 'basin of attraction' is the whole plane. Illustrated below is a slightly simpler cubic equation which generates a rounded square.


A cubic differential attractor in 2 dimensions

$$x' = x + \delta(x + y - x^3)$$
$$y' = y + \delta(y - x - y^3)$$

The functions do not have to be cubic. Even quadratic equations will generate periodic attractors. In the case below the basin of attraction is quite small.


A quadratic differential attractor in 2 dimensions

$$x' = x + \delta(0.9x - y - 2x^2)$$
$$y' = y + \delta(2x - 0.8y - 2y^2)$$

Differential Attractors in 3 Dimensions

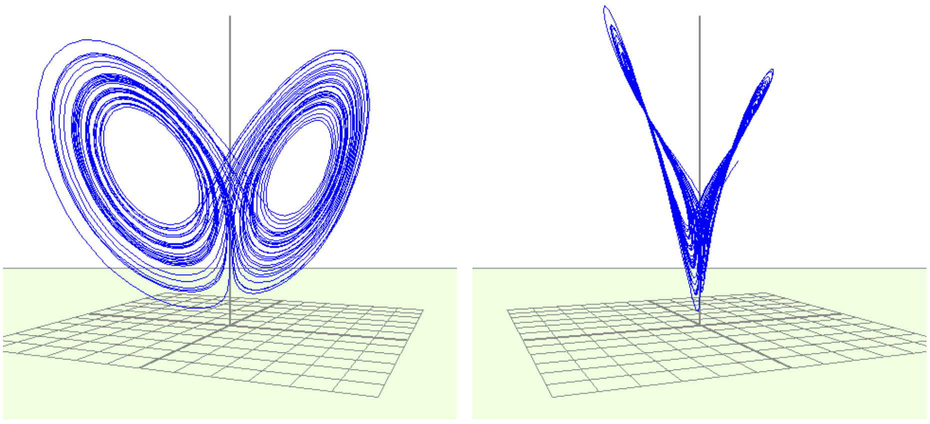
Because 2 dimensional systems are limited to periodic attractors, it follows that if we are to find any strange (i.e. fractal) attractors in a continuous dynamical system it must have 3 or more dimensions. In other words it must be of the form:

$$\begin{aligned} x' &= x + \delta \times f(x,y,z) \\ y' &= y + \delta \times g(x,y,z) \\ z' &= z + \delta \times h(x,y,z) \end{aligned} \tag{36}$$

In 1963 Edward Lorentz, a mathematician and meteorologist, discovered such a system almost by accident. Here it is.

$$\begin{aligned} f(x,y,z) &= 10y - 10x \\ g(x,y,z) &= 28x - xz - y \\ h(x,y,z) &= xy - 8/3z \end{aligned}$$

In order to visualise this system, we must plot it in 3 dimensions. First x, y and z are set to any suitable starting point; δ is set to a conveniently small value and then the ball is set rolling. Sadly the images on these pages can only hint at the development of this amazing attractor.



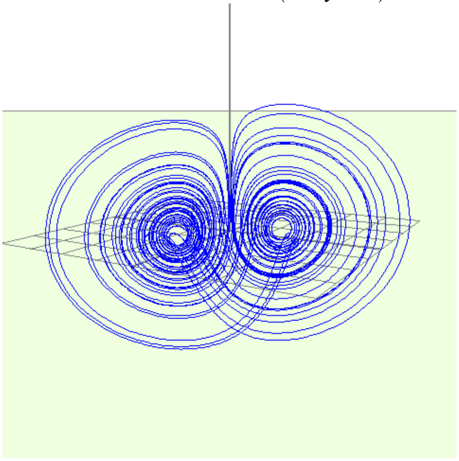
The Lorentz Attractor

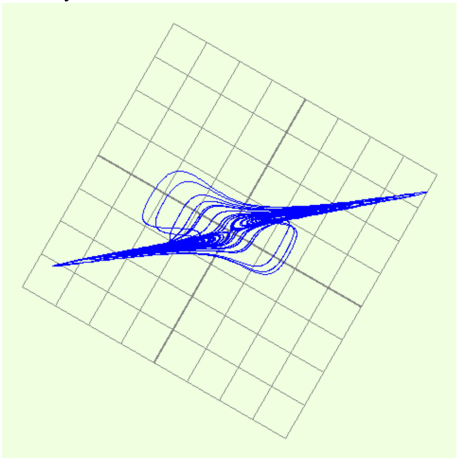
The images above show two views of the Lorentz attractor. It resembles two gramophone records which have stuck together and have been partially peeled apart. Wherever you start from the ball rapidly falls into one or other of the two whirlpools. Having rotated round and round a few times it swaps over to the other whirlpool for a while, then back again seemingly at random.

That is not all. If you throw two balls into the mincer from two very closely spaced points, for a while the two balls will stick together but sooner or later one ball will decide to continue going round one leaf while the other swaps over to the other. In other words, the system displays extreme sensitivity to initial conditions – the hallmark of chaos.

Here is another Lorentz-type attractor whose equations are:

$$\begin{aligned} f(x,y,z) &= yz \\ g(x,y,z) &= x - y + 0.7\,xz \\ h(x,y,z) &= 1 - xy \end{aligned}$$

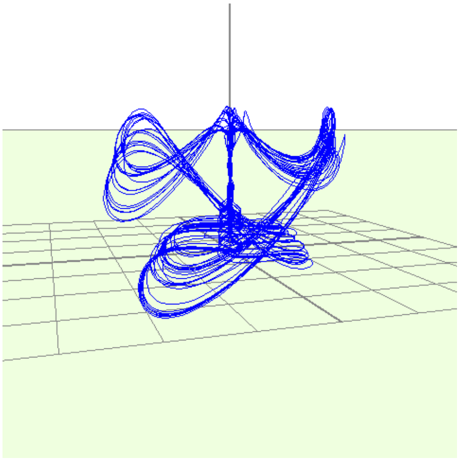
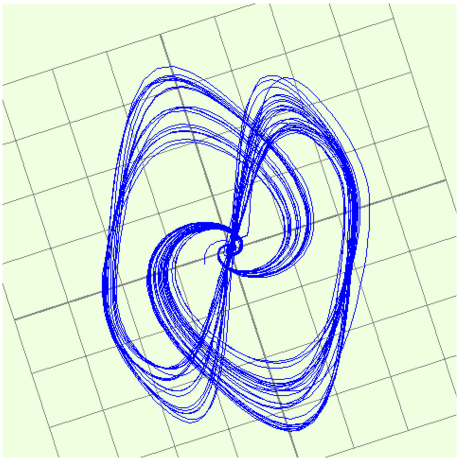




Both of these attractors are rotationally symmetric about the *z* axis. This is because the equations for *dx* and *dy* contain only *odd* powers of *x* and *y* and the equation for *dz* contains only *even* functions of *x* and *y* This ensures that, when both *x* and *y* have their signs reversed, the signs of *dx* and *dy* are also reversed but *dz* remains unaltered. Here is another remarkable rotationally symmetric attractor which I call the sprinkler:

$$f(x,y,z) = -1.5x + 1.6y + 1.1xz^2$$
$$g(x,y,z) = -2x - 0.7y$$
$$h(x,y,z) = 0.2 - x^2 + 0.4y^2$$

The first illustration below has the Z axis vertical while the second is viewed from above; the symmetry is obvious from this vantage point.

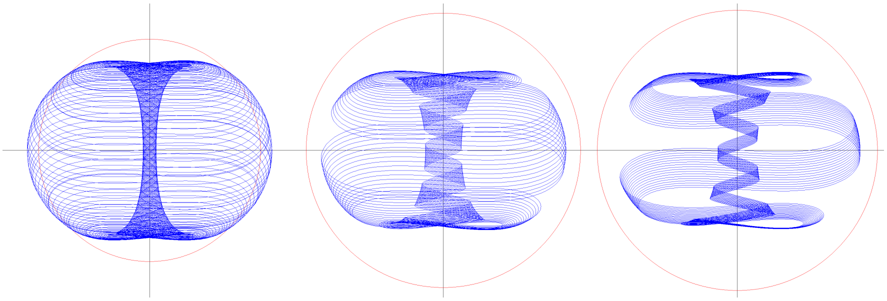



It is fascinating to watch what happens to a point as it traces its way through the attractor; first it climbs slowly up the Z axis, spiralling round as it goes; then at the top it sprays out sideways, taking one or other of the two major pathways back down towards the origin again.

(This particular attractor is not very stable. Its basin of attraction appears to be quite small and it is critically dependent on the values of the coefficients. There may well be more stable versions of the same kind of attractor.)

The following set of equations obeys the rules for rotational symmetry but does not generate a strange attractor; it generates a series of nested toruses which depend on the initial start6ing point.

$$f(x,y,z) = -2y + xz$$
$$g(x,y,z) = -2x$$
$$h(x,y,z) = 0.1 - y^2$$

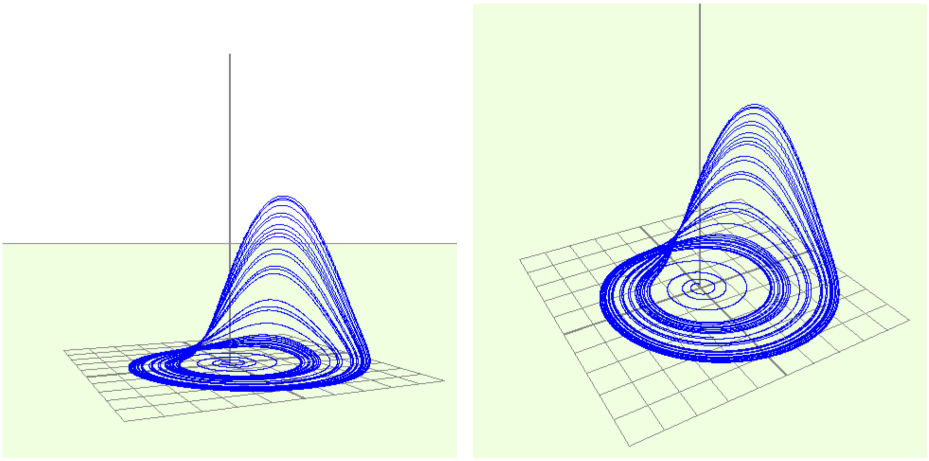


These are not fractals because each one is confined to (and in most cases completely covers) a 2 dimensional surface. Nevertheless, they are rather pretty.

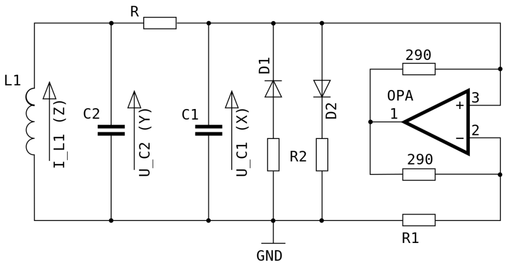
Another famous attractor called the Rössler attractor was specifically designed to be as simple as possible, having only one non-linear term in it (but the *xz* term destroys the rotational symmetry).

$$\begin{aligned} f(x,y,z) &= -y - z \\ g(x,y,z) &= x + 0.2\,y \\ h(x,y,z) &= 0.2 + xz - 5.7\,z \end{aligned}$$

Essentially the *f* and *g* functions cause the point to spiral outwards in the *xy* plane but at a certain radius the *z* coordinate begins to kick in, twisting the circular disc into a kind of Möbius band which returns the point to somewhere else in the disc.



A third attractor (known as the double scroll attractor) was discovered by Leon Chua who devised a simple electronic circuit to demonstrate how it could be implemented in a real-world situation. In the circuit below the double resistor/diode combination acts as the non-linear device which is essential in any circuit which exhibits chaotic behaviour.

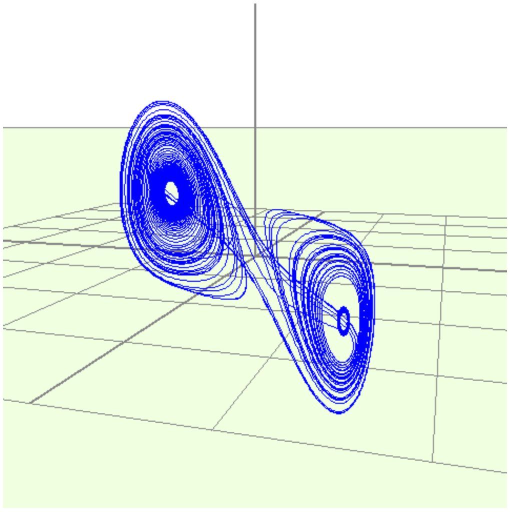


Chua's circuit

The following formula (in which the x^3 term plays the role of the non-linear resistor) shows the same behaviour:

$$
\begin{aligned}
f(x,y,z) &= x + y + x^3 \\
g(x,y,z) &= (2x - y + 3z)/6 \\
h(x,y,z) &= -y
\end{aligned}
$$

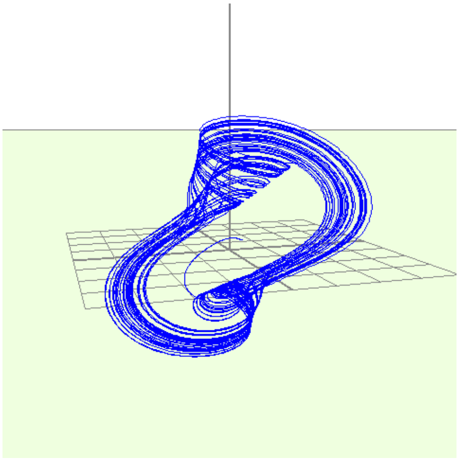
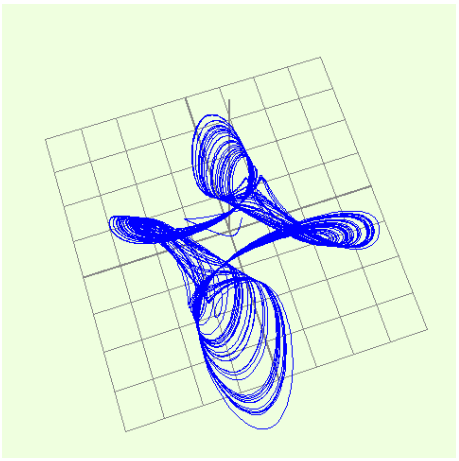
and looks like this:



The double scroll attractor has the property that all its terms are of *odd* powers of x, y and z. This means that it will be diametrically symmetric about the origin.

Here is another diametrically symmetric one which I have discovered:

$$\begin{aligned} f(x,y,z) &= -2x + 2z - 2xyz \\ g(x,y,z) &= 2x - 2z \\ h(x,y,z) &= y \end{aligned}$$

No doubt many other interesting attractors remain to be discovered.

Carr Bank; February 2019

Appendix

Hausdorf dimension

The Hausdorf dimension of a self-similar fractal is calculated as follows.

If it takes *n* copies of the fractal to make it *p* times larger, the the Hausdorf Dimension of the fractal *H* is given by the equation:

$$p^H = n$$

For example, since it takes 8 copies of a cube to make a cube twice as large, we have

$$2^H = 8$$

in which case *H* is obviously equal to 3 – the dimensions of a cube.

In the case of the Koch curve, it takes 4 copies of the curve to make one 3 times larger so

$$3^H = 4$$

To calculate this we have to take the logarithm of both sides of the equation. (It doesn't matter what the base of the logarithm is.) This gives us

$$H\log(3) = \log(4)$$

$$H = \frac{\log(4)}{\log(3)} = 1.26$$

If the fractal is not self-similar the calculation is much more difficult; indeed, the concept of a fractal dimension is not really well defined under these circumstances. A rough estimate would be to measure the length *l* of the *n*th iterate; then measure the length *l*' of the *n*+1th iterate scaled to approximately the same size. The difficulty here, of course, is defining what is meant by 'approximately the same size'.

Most of the illustrations in this book were generated using programs written by the author, many of which are available on his website: www.jolinton.co.uk.