

Chapter 7 - The Fractal Link

When chaotic dynamic systems were first being explored, **Fractal Geometry** wasn't though to have much in common with the world of bifurcation diagrams or attractors. However, as we saw in the last chapter, there *was* a connection between the two, deep inside the structure of strange attractors. In this Chapter, we'll take a brief detour to look at Fractal Geometry in a little more detail and explore some simple fractal shapes.

If you look around at natural and man-made objects, you will soon notice a few things. The first is that man-made objects are designed using essentially regular shapes, whereas natural objects are very irregular. As Benoit Mandelbrot pointed out in *The Fractal Geometry of Nature*, mountains are not shaped like cones, and clouds are not spheres. Cones, spheres, cubes and the like are all traditional, Euclidean geometry shapes, created by man for use in our own structures; there's nothing special about them. As well as the irregularity, there is what's best called a 'roughness' about natural objects. And, if you do look at natural objects from different distances, you'll see that they also exhibit self similarity. That is, a mountain observed from a mile away will exhibit a similar jaggedness to part of the mountain peak observed at close quarters. This isn't exhibited, for example, by a sphere; if I look at a sphere from a distance, then look at it close up, there is a difference.

This concept of self similarity is often described in terms of what's been called the '*coastline of Britain*' problem. If you start off with a typical atlas map of the world, and use various means to measure the perimeter of the coast of mainland Britain, then you'll get a particular value. Now, turn to the pages showing a larger scale map, and repeat the exercise. You'll find that the larger scale map shows more of what Slartibartfast, in the '*Hitch-hiker's Guide to the Galaxy*', by Douglas Adams, called the 'crinkly bits'. If you measure the new perimeter by including all the crinkly bits, the bays and coves and inlets, then the distance obtained will be larger. Going to a set of Ordnance Survey maps for the coast will give an even longer distance, because the detail shown on the map will be greater. Now, if we forsake our maps and start walking around the coast, we'd have an even longer distance to cover, as we could follow the absolute line of the coast to a couple of feet. And, if we had the services of an ant, the distance covered would be even greater, because the ant could explore the gaps between pebbles! Taking this to an extreme situation, it's possible to argue that the coastline of a country, if viewed in extreme close up, would constitute an infinitely long line enclosing a finite area. In addition, a highly magnified view of a coast line would show similar crinkliness to the coast line as a whole. This self similarity is a common feature of many natural shapes. This type of self-similarity has been described as *statistical self similarity*, as the similarity exhibited is not precise and the shape does **not** repeat itself exactly at different scales. However, it's close enough to be recognisably the same.

For example, clouds exhibit a statistical self similarity, as do trees and plants such as ferns. If you don't believe me, go out and look for yourself. I have to admit, that since becoming a bit of a fractophile I also spend a lot of time looking at my rather moth-eaten potted fern! Indeed, it seems that the natural world is built out of shapes exhibiting these properties. These shapes are all **fractal**, as defined in the last Chapter, having an HB dimension that's different to their Euclidean dimension. To start examining the weird and wonderful world of fractals, it's a good idea to 'build' one and look at it in greater detail. The simplest shape to start with is called the **Koch Snowflake**. This shape hasn't got anything to do with chaotic systems as explored in previous parts of the book, but will prepare us to handle the fractal nature of some other iterative systems that we will explore later in the book. Mathematically generated fractal shapes are exactly self similar; that is, they are made up of the same shapes at different scales.

It's interesting to note that these shapes were discovered long before computers and their graphics abilities, and that all the ground work was done before anyone had even dreamt of chaotic systems. In addition, fractal shapes were deemed to be so far out of the normal mathematical 'establishment' that they were called 'pathological' curves, and it's even reported that mathematicians cast doubt on the mental stability of some of the original workers in the field!

The Koch Curve

This curve was described by Helge von Koch in the early 1900s. It is created by the following sequence of steps:

1. Take an equilateral triangle, and build another equilateral triangle in the middle of each side of the shape, the new triangle having a base length of $1/3$ of the length of the side.
2. Repeat *ad infinitum*.

This simple recipe has in it two steps that are at the heart of fractal geometry; the self similarity at all scales (if we repeated the operation an infinite number of times, a magnified view of any section of the Koch curve would look the same as any other part at any magnification), and the same step is repeated an infinite number of times to generate the fractal. The process that draws the curve actually calls itself. This is called **recursion**. It should be noted that these mathematically constructed shapes don't actually become fractal until the number of iterations involved is infinite. Obviously, we're not going to be able to wait that long, so we effectively get an approximation to a fractal shape. Anyway, before going into detail on the theory side, let's build a Von Koch Snowflake. This is simply three Von Koch Curves joined together. Listing 7.1 shows how this is done, and you will see that there are two new programming techniques used in this listing.

Recursion

Recursion is the name given to the process in which a procedure or function in a computer program calls itself (direct recursion) or calls another routine which then calls itself (indirect recursion). I remember some years ago coming across a very facetious definition of the word 'recursion'; Recursion - see recursion! In the Von Koch curve procedure, the routine works by decrementing 'depth' and only draws a line when 'depth' reaches 0. If you work through the procedure, you'll see how the routine works; it's not transparently obvious at first, so be patient! A call to the curve generator requires 2 parameters; the depth, which gives you a measure of the 'crinkliness' of the resulting curve, and the length, which is effectively the length of the starting line. You will find that, with any recursive routine used to draw fractal curves, the following is true:

1. The greater the depth parameter is, the longer the routine will take to run.
2. The greater the depth parameter is, the more 'crinkly' the curve will be.
3. For a given depth, some values of length do **not** appear to work. In general terms, the higher the value of depth, the higher the value of length will need to be.
4. In this particular routine, a large 'length' parameter will give a large snowflake on the screen, unless the drawing routines are modified to scale the X and Y distances drawn.

The process of recursion will clearly give rise to self-similarity; after all, if we repeat a task using parameters that have been modified by that task, we're bound to get some self similarity along the way. It's a little like the repeated iterations we used to generate the various mappings from the Logistic Equation, but there the equation didn't call itself; it was used repeatedly to process data that it had already processed.

Recursive Programming

I can't resist the pun; 'Here there be dragons!' (see the dragon curves, later in this Chapter). Recursive programming can be a bit of a minefield in some languages due to the construction of the language interpreter or compiler. The routines listed here that use recursion have been tested on PCs, using Turbo Pascal and Visual BASIC, but tread warily on other dialects and languages. The reason for this is that many languages are designed and implemented with recursion very far down the list of desirable qualities. When a recursive routine is used, a set of calls to that routine are stored in the computer's 'stack' an area of memory designed to keep track of where the computer is in your BASIC or Pascal program. Some dialects don't like this at all, and either generate an error message or, worse still, crash the computer. So, here are a few pointers for pain free recursion.

1. - ALWAYS save your work before test running a program. Murphy's Law will always make sure that if you test a recursive program before saving it you will crash the computer.
2. - Consult your language compiler / interpreter manual. Some languages allow you to have some control over stack size, etc. and so if the manual offers guidance, follow it.

3. - PC languages frequently have a stack size limited to 64k, irrespective of the available RAM on the computer. This offers a finite limit on the depth of recursion you can go to before the stack fills up. Stack overflows are frequently 'fatal' error; you will be unable to finish the program run, although the language will frequently return you to a safe place. Some compilers offer the chance of turning off stack error checks; don't, until you've got some experience.
4. - When testing a program, start with a small depth of recursion just to see what happens.
5. - Within procedures that are to be called recursively, define no more local variables than are actually needed for the correct execution of the program. Don't forget that each time the procedure is called, recursively, a new set of local variables are defined which could easily eat up memory.
6. - Many languages have 'dirty tricks' associated with them for increased speed. I try to avoid them all the time when programming; I **always** avoid any 'dirty' techniques when writing recursive code.

Turtle Graphics

The process of drawing Fractal curves and other fractal shapes is not easily accomplished using what might be termed 'conventional' drawing methods. Usually, for drawing a curve, we use some formula or other, call it a few hundred times to get a series of plottable points, and then plot the points. Despite the apparent simplicity of the Koch Curve shown in Figure 7.1, there is no algebraic formula that will define the position of points on the curve! This is another common feature about fractal curves; they may look simple, but to draw them requires a new way of looking at geometry and computer graphics. A useful technique known as '**Turtle Graphics**' has been shown to be a simple and effective way of drawing these shapes. The original 'turtle' was actually a computer controlled 'buggy' with a pen, driven by a computer running a language called 'Logo'. This was designed as an educational tool, allowing children to explore mathematics in concrete terms of shape drawing, whilst also giving an insight into programming and the mental processes involved in that field. In Turtle Graphics, there are a collection of commands which tell the Turtle, or a screen representation of it, what to do. So, 'Forward 100' will send the turtle moving forwards 100 turtle 'units', which might be inches on the floor, or pixels on the screen. A command such as Turn 90 would turn the turtle 90' clockwise, and so on. By combining these commands together into procedures, shapes of great complexity could be drawn.

Well, here I've defined a couple of Turtle Graphics commands in the Pascal toolkit, and use them to draw curves. Front is the command for 'Forward'. (The reason for this is simple; there are

some languages that seem to object to the use of the name 'forward' as a procedure or function name, and I got rather fed up of the errors! The 'Turn' command takes an argument in degrees and turns the 'turtle' through that number of degrees clockwise (or, if the argument is negative, anticlockwise). The 0' point is at 3 on a clock face, +90 at 6, 180' at 9, 270' at 12 and 360' back at 3.

The Curve Itself

If you run the program listing given, the snowflake will be drawn on the screen. It consists of three Koch Curves drawn one after another, after turning the turtle. The Koch Snowflake that we've drawn has clearly got a finite length; I could measure each side, and add the lengths together, and that would give me a total length for the perimeter of the snowflake. This is because I've only developed the snowflake to a specified depth of recursion. If I took the recursion depth to infinity, then the shape resulting would be truly fractal. In fact, the curve that would result from this would be a little odd, because the direction of the curve would change at each point in the curve! This isn't too easy to visualise, but you can see how we might end up there if you modify the depth variable in Listing 7.1. The Koch curve, which would be drawn by just calling the Koch routine once with a suitable depth and length parameter, is a useful experimental beast for us, because it is so simple but it reflects the behaviour of other fractal curves. The first thing we can do is to examine a method of quantifying the fractal dimension of

these shapes. So far, we've said that something like the Koch curve would have a fractal dimension of between 1 and 2; but what would it be?

We'll start off with looking at two important ideas about fractal shapes. They all have what is called an **initiator** and a **generator**. The initiator for a Koch curve is shown in Figure 7.1a - it's simply a straight line. The generator for a Koch Snowflake would be an equilateral triangle. The initiator of a fractal shape is the euclidean shape that gives rise to the fractal curve. The generator, on the other hand, is what the initiator is deformed into as the fractal curve is developed. The generator consists of a series of line segments. Here, for example, the generator for both Curve and Snowflake is shown in Figure 7.1b. It consists of 4 line segments, and in the fractal generation process each straight line segment of the curve would be replaced by a generator like this. Then, if the process were allowed to continue, the line segments of the generator would in turn be replaced by new generators! An examination of the generator for the Koch Curve will show that it has 4 segments, call this number N , and that each segment is $1/3$ the length of the initiator - call this number l , where, for this case, $l = \text{initiator length} / 3$. The fractal dimension of a curve can be given by:

$$D = \text{LOG}(N) / \text{LOG}(1/l)$$

This is a general relationship for many simple fractal curves. For the Koch Curve, we can stick some figures in and arrive at:

$$D = \text{LOG}(4) / \text{LOG}(1/(1/3))$$

$$D = \text{LOG}(4) / \text{LOG}(3)$$

$$D = 1.26218$$

This calculation of the fractal (more formally, Hausdorff-Bessicovitch) dimension won't work for all types of fractal, but will for the sorts of fractal curves that we'll be discussing in this Chapter. The greater than value of D, the more irregular is the fractal being described. For example, a fractal curve with a D value of 1.5 would be more irregular than the Koch Curve. It's interesting to note that estimates of the D value for a typical coastline is about the same as that for a Koch curve.

As already mentioned, the Koch curve, when the recursion is taken to infinity, would have a length of infinity. If we think about this a little more, then you can see how, as the HB dimension increases from 1 to 2 then any line drawn will come to look more and more like a surface; i.e. it will start filling up more and more of the 2 dimensional plane. Such curves, with relatively high HB dimensions, are called **Space Filling Curves**. However, these curves still

remain lines rather than surfaces; breaking the curve in one place will give two separate curves; with a true surface this wouldn't happen. There are many variants of the Koch Curve that have been published in books and magazines; the simplest is to replace the equilateral triangle generator of the Koch Curve with a square, thus giving a generator with 5 segments rather than the normal 4. This will give a slightly different curve with a higher HB Dimension.

The path that the turtle takes when creating the generator for this curve is as follows:

```
Front(1/3)
Turn(-90)
Front(1/3)
Turn(90)
Front(1/3)
Turn(90)
Front(1/3)
Turn(-90)
Front(1/3)
```

Where l is the length of the initiator line. You might like to modify the program given to generate this curve, as shown in Listing 7.2. When experimenting with this curve, the same rules apply as did with the triangular version. You might also like to try generating a square version of the Snowflake, or altering the distance in one of the Front commands to a different value to $1/3$.

Fractal Brownian Motion

The Koch Curve, whilst providing a useful model for understanding fractals, isn't used to model anything in the real world. Let's now look at a fractal curve generating method that can model some natural phenomena. A thing that can be modelled by a fractal curve is that of Brownian Motion. This is the random motion taken by a very small particle in a fluid of some sort as it is bombarded by the motion of the molecules of the fluid itself. The traditional demonstration was to take something like indian ink and make a solution in water. Then, under a microscope, the motions of the ink particles could be followed as they were knocked around by the invisible water molecules. Brownian motion is interesting as a fractal system for the following reasons:

1. - It's real; you can see it under a microscope, and the model created as a fractal curve can accurately reflect what is seen in real life.
2. - It's a fractal generated by **random** means; in the case of the Koch curve we saw a specific course of action laid down to generate the curve. Here, we can use random numbers to help generate the fractal curve. Such curves are called **Random Fractal Curves**.
3. - It's an example of a statistically self similar system.

An in depth analysis of this subject can be found in '*The Science of Fractal Images*' edited by Peitgen and Saupe. In that work, you'll find many algorithms to generate this curve, and

Listing 7.3 is based on the *Random Cuts BM* algorithm they describe. On running the program with the default values you should see a jagged line generated. Lower values of Wiggle give a less bumpy ride. For higher values you may need to alter the value of the variable 'scale'. Note that Gaussian random numbers are used. All this means is that the random numbers returned by the random number function are modified slightly to give a range of numbers that follows a Gaussian distribution. Other methods of generating this type of curve include what is called **mid point displacement**, where a line is drawn and the mid-point of the line is pushed up or down (selected randomly) by a randomly selected amount. Each of the two halves is then subjected to the same treatment, and so on *ad infinitum*. This recursive process again gives rise to a fractal representation of Brownian Motion.

One thing to note about the generation of random fractals is that the calculation of the HB dimension is clearly not as straight forward as it was for the Koch Curve. However, a value can be calculated mathematically, but that's beyond the scope of this book.

Cantor Set

So far, the fractal curves that we've looked at have HB dimensions between 1 and 2, and so they lie somewhere between being a line and a surface. We can go in the other direction, however, and generate patterns that have HB dimensions between 0 - representing a collection of

points - and 1. Such 'shapes' are often called **dusts**, because they have the appearance of a sprinkling of dust particles on a surface. The simplest example of this is called the **Cantor Set**, which was discovered by Georg Cantor in the nineteenth century. The recipe for producing a Cantor Dust, or Cantor Discontinuum, as it is occasionally called, is quite simple.

1. - Take the numbers between 0 and 1, and represent them as a line.
2. - Remove the central third, leaving, if you like, 0 to 0.333... and 0.6666... to 1.

Repeat this process, *ad infinitum*

Again, taking the process to infinity is needed to generate the true Cantor Dust; doing the operation a finite number of times will generate approximates to the dust. The implications of this method of generation are that you will end up with an infinite of points, arranged in clusters, that have a total length of 0! Now, what is the HB Dimension of this set? Well, we can calculate it;

$$D = \frac{\log(N)}{\log(1/l)}$$

where N is the number of segments in the generator of the fractal curve and l is the length of each of the segments as a fraction of the full length of the initiator. N is 2 - there are 2

segments generated, and $l=1/3$ - each segment is $1/3$ of the full length of the initiator, which is a straight line. Thus, the HB dimension is:

$$\begin{aligned} D &= \text{LOG}(2) / \text{LOG}(1/1/3) \\ &= 0.301 / 0.477 \\ &= 0.631 \end{aligned}$$

This makes the Cantor Set a fractal dust with properties intermediate between those of scattered points and a full line. It is *not* just a regular scattering of points! Not surprisingly, changing the size of the fraction removed will generate dusts with different fractal dimensions. In addition, it's possible to have a random Cantor Dust, where the fraction removed each time is randomly generated, sometimes a third, sometimes a quarter, and so on. You will often read of the Cantor Set being 'disconnected'; this is a mathematical term referring to the fact that it is a dust. Within the clusters, there is self-similarity - each cluster is, after all, arrived at by taking out the middle third in the case of the 'standard' Cantor Set. In the case of the random Cantor Set, there would be statistical self similarity rather than exact self similarity. Listing 7.4 shows how a simple dust can be generated. However, we're limited by the screen so you'll soon see that the pattern no longer changes. Each line of dots is a step of removal of thirds.

As far as we're concerned, the Cantor Set has another interesting property; it provides us with a link to chaotic systems. To see how this is, we need to consider the work of Benoit Mandelbrot as he dealt with the problem of telephone line noise for IBM. When computers communicate over telephone lines, there is always going to be some noise. Whenever a particularly 'loud' burst of electronic noise occurs, data might be lost and the transmitting end of the link will have to send that chunk of data again. The more errors, the slower the link will be at transmitting a given amount of data. The particular problem that Mandelbrot was asked to look at was that of intermittent noise bursts which defied the efforts of the engineers to cure them. To start with, the problems came in bursts - you might have the odd hour with no problems, then a burst of noise, then no problems again. Funnily enough, on examining the period of time with noise problems, you'd find periods of time with no noise, and others with noise. Going down further into the noisy parts of the hour, you find periods of time with no noise and periods with noise.

What Mandelbrot did was to treat the intermittent problem of noise in the system as a Cantor Set, with the noise instead of the 'line' that we drew. Noise free periods are represented by the segments taken out of the line. Ultimately, we get a 'dust' of noise clusters. The result in practical terms was that the engineers were going to have to learn to live with these noise clusters and develop means of catching the errors and correcting them quickly. From our point of view, though, we have what was on the surface an essentially chaotic event, the random noise, being

modelled mathematically by simply treating the events as a Cantor Dust in *time* instead of space.

The Cantor set provides us with a method of looking at **Intermittency** - that is, events happening on an occasional basis. Many chaotic systems exhibit intermittency; remember the Logistic Equation, and how periodic areas popped up from chaotic regions of the bifurcation diagrams? Well, that's an example of intermittency. Here, we see that we can model intermittency in some systems using a fractal object. The nature of noise on a phone line is also an example of a **discontinuous** phenomena, where a smooth transition from one state to another does not exist; you've either got noise there or you haven't. In some ways, this is analogous to the discontinuity observed as the Logistic equation slips into chaotic behaviour after a sequence of period doubling.

More importantly, though, the Cantor Set allows us to tie up an annoying feature about Strange Attractors; remember that in a Strange Attractor, the trajectory does not intersect itself, as that would constitute a closed loop and indicate a periodic system. However, we know that these strange attractor trajectories contain points that are very close together without touching each other, as shown in the Henon Attractor. This can all be explained if we treat the cross section of a strange attractor as a Cantor Set. After all, the Cantor Dust is as close to nothing as you can get whilst still being a single mathematical object, so that would allow trajectories of a Strange Attractor to apparently merge together, as in the Lorenz Attractor, without actually intersecting. Indeed, one of the definitions of a strange attractor is that it is "a non-intersecting

line of infinite length with an infinite number of loops having a Cantor Set as its cross section."

A further link between chaotic systems and fractals is found in the phrase 'infinite length' - this line is, after all, enclosing a finite volume or area of phase space, and so will, like the Koch Curve, have a fractal dimension that is different to its Euclidean dimension.

Other Fractal Curves

There are lots of other fractal curves that we can write programs to generate, and there's no way that I can cover all of them in the space available. Instead, I'll look at a few samples of the many types of curve available to us, and provide pointers for generating other curves.

Peano

The Peano Curve is a rather different type of fractal curve to the Koch Curve, in that it has an HB dimension of 2. For this reason, it is often called a **plane filling** curve - after all, that's exactly what it does. It will ultimately fill the plane in which it is drawn. One implication of this is that the traditional Peano Curve will intersect itself, unlike the Koch curve. There are variations of the Peano Curve that do not intersect in this way; these, however, will have a slightly lower HB dimension. Listing 7.5 shows a routine for generating a Peano Curve of the traditional sort, and Figure 7.2 shows the initiator and generator. Again, I've used Turtle

Graphics routines to draw the shape. You should be able to see how the generator is traversed by looking at the 'Peano' procedure in the program. You will see that the development for the Peano Curve is not easy to follow; this was one reason why the non-intersecting curve (Figure 7.2b) have been studied, as they allow the way in which the curve develops to be easily seen.

One interesting experiment that you might care to try is to use the Peano Curve to save the contents of a graphics display screen. The curve fills a plane, and if the plane is a computer display then the curve would eventually address each location on the display. Now, if we look at each pixel visited by the curve and note its colour in an array of integers, we will have a list of numbers that defines, the colour of each screen point visited by the curve. Thus, to re-create the screen, all we need to do is to provide the same start point as was used when the array was generated, the same length and recursion depth, and start the program running so that at each point visited the program sets the visited pixel on the screen to the colour held in the corresponding position in the array of numbers. Of course, this is a rather inefficient way of storing screens, but it offers some interesting features:

1. - The array of numbers is meaningless without knowledge of the fractal curve parameters used to generate the numbers from the screen image. This is a simple form of encryption.

2. - The 'resolution' of the saved image is varied by altering the number of pixels visited on the original screen. This can be done by changing the recursion and length parameters.

Listing 7.6 shows a practical demonstration of this. As you can see, I've used a bold, simple, image; these work best with this simple approach. This simple system here models a more complex approach taken by a group of British Government scientists who have explored the use of Peano Curves for the storage of data in a similar way to this, but they also include run length coding algorithms to allow compression of stored data.

Dragon and C Curves

Some fractal curves generated in this way create very convoluted shapes that are called 'dragons'. This is a little confusing, as there are a set of fractal patterns created by iteration (see Chapters 7 and 8) that are also called dragons, despite the different way of generating them. In this section I'll examine a couple of dragon curves that are generated recursively by applying Turtle Graphics techniques to an initiator and a generator. Why are they called dragons? Well, some of the curves produced do have a passing resemblance to heraldic dragons. C Curves are so called because they look like highly elaborate letter Cs.

Starting off with the C Curves, the 'recipe' is quite simple:

Front(1)

Turn(-90)

Front(1)

Turn(90)

Listing 7.7 draw a C curve. You may like to try varying the angles turned through, as well as the depth of recursion and the length of the sides drawn. If you want a 'spiky' C curve, simply make both turn commands turn through either -90' or 90'.

As to the dragon curves, these are slightly more complex, in that to draw them in the same way as we've drawn the previous curves we'd need to alternate between two separate generators as the curve is drawn. Listing 7.8 draws a Dragon Curve based upon an algorithm published in *'Dynamical Systems and Fractals'*, by *Becker and Dorfler*. The curve itself can be modified by varying the angle through which the turtle turns; certain angles give dragons that look surprisingly like rivers seen from the air.

Sierpinski Carpet

The Sierpinski Carpet is quite interesting. Rather than draw a Sierpinski curve, which consists of a line, I chose to include a Sierpinski Carpet. Although I've drawn the carpet using recursive procedural techniques, you can also generate a Sierpinski Carpet by a technique using what are called **Iterated Function Systems** (IFS). This technique is stunningly simple but quite spectacular, and I'll be considering it in greater detail in Chapter 12. However, for now I'll simply consider the generation of a carpet using recursive techniques.

There are two types of Sierpinski Curve that you'll come across in text books; these are the Triadic Curve, which ultimately gives triangular shapes, and the Quadric Curve, which ultimately gives rise to square shapes. Listing 7.9 shows a program to generate a Quadric Sierpinski Carpet. The triadic carpet is the one that usually gets the coverage, so I hope that this redresses the balance somewhat!

Fractals in the Real World

To complete this Chapter, I thought that I would briefly examine some of the places that the fractal curves explored in this chapter turn up in the real world, as well as some applications of fractals like those we've seen. In a later Chapter, we'll explore other, more complex, fractal systems and examine their practical uses. It is often a surprise for programmers to learn that their interesting shapes also have some practical significance and do occasionally show up in the real world! The self-similarity exhibited by the real world fractals is statistical; there are no naturally occurring exactly self similar fractals.

Rain and Snow

One feature of clouds has already been discussed in this Chapter; they are, like many natural objects, fractal. Indeed, film-makers have used fractal routines to produce graphical images of clouds. Shaun Lovejoy discovered by studying both satellite and radar images of clouds that the fractal dimension of clouds is the same (about 1.33) at a variety of scales. The practical upshot of this is that it's hard to get any idea of the size of a cloud if you're shown a photograph of one with nothing to act as a scale. It could be a small cloud seen close up, or a large one seen from a distance. In addition, he found that the fall of rain tends to occur in bursts of rain, at irregular intervals. This is a similar mechanism to the Cantor Set model of

transmission line noise put forward by Mandelbrot. He also determined that the perimeters of areas on to which rain is falling are also fractal, rather than smooth.

A microscopic examination of snowflakes also exhibits fractal properties, and we can model snowflakes very crudely with a fractal curve, as we saw in Listing 7.1. The reason why our Koch Snowflake doesn't look much like a normal snowflake is that a snowflake is statistically self similar, rather than exactly self similar. Also, the method by which a snowflake is built is an example of crystal growth, rather than a deterministic process. An examination of a frost covered window in cold weather will show another form of fractal object, this one grown by a DLA process as outlined below.

A final example of a fractal object in meteorology is the structure of fork lightning. Close examination of photographs of this type of lightning indicate the existence of many fingers of lightning off of the main branches.

Geography

We've already seen how a coastline might be viewed as a fractal shape. The most obvious example is probably the fjord coast of Norway, but even apparently smooth coasts consisting of sandy beaches will have a fractal dimension. It shouldn't surprise us, therefore, to find that other geographical objects have a fractal dimension as well. The coastlines of lakes will clearly have this feature, as do mountain ranges, cliff faces and other natural boundary points. Mandelbrot observed that the path of rivers can be modelled by Fractal Curves, with a HB dimension of between 1.2 and 1.3. The windier the rivers are, then the higher their Fractal Dimension. A useful rule of thumb when looking out for geographical or geological fractals is to look at pictures that do not include any means of scaling. A shape that can be modelled by a fractal curve often has the quality that by looking at it without any other information you cannot tell its size in the real world; an isolated picture of a rock might be in someones garden, or at the top of an escarpment in the alps!

If you want to experiment with generating models of coastlines using fractal curves, then you could start with the fractal Brownian Motion routine listed above. Alternatively, try Listing 7.10; this is a modified Von Koch curve generator which uses random numbers to give the turn angle and lengths traversed by the Turtle. Plots from this program are prone to suffer from

doubling back on themselves, but you can get some quite realistic looking coasts with some perseverance.

Biology

Plants are very fractally; in Chapter 10 we'll examine how models of trees and ferns can be synthesised using fractal techniques, and so it's not surprising that plants can be said to have a fractal dimension to at least parts of their structure. The easiest example is in the leaves; a fern leaf is often used as an example of a perfect fractal object, but leaves of other plants, such as oak trees or holly bushes, also have a fractal dimension. The root and branch systems of trees and plants also exhibit fractal dimensions. A further interesting point is in the fractal nature of leaf surfaces when considered as an environment in which creatures can live; the pitted surface of leaves gives a larger than expected surface area that can be exploited by creatures small enough to fit.

In the animal kingdom, we can think of things like the bronchial structure in the lungs and the capillary system to be fractal. A similar system is the spiracle system in insects, responsible for getting oxygen into the insect body. There is a very good biological requirement for these structures to exhibit a fractal structure, as they all need to create a large surface area (for gas and chemical exchanges) within a frequently small volume. For example, it's been estimated that the

bronchial system of a human lung would, if opened out, have an area equivalent to that of a soccer pitch. The exchange of materials is aided at the cellular level by cells frequently having 'rough' walls to maximise their surface area. A similar argument applies to the digestive tract of mammals, where the cells of the gut are covered in villi, finger like structures that offer a larger area for absorption than the cells themselves would otherwise.

The cell walls of living things are also fractal, as are the protein molecules essential to life. For example, molecules that are designed for carrying materials around the body have a fairly high fractal dimension as it needs to adsorb molecules onto its surface and hold them. Protein molecules, like cell walls, have HB dimensions between 2 and 3 (that is, exhibiting properties between those of a surface and a three dimensional object).

Fluid Flow

Fluid travelling through other fluids, or the same fluid at a different speed, also exhibits a fractal structure. You can see this by simply looking at the turbulence in rivers, or even the smoke from a chimney; the boundary between the smoke and the air is a fractal boundary. A 'classic' example of the flow of fluid within fluid is demonstrated in a piece of equipment called a Hele-Shaw cell. This consists, in essence, of two plates of glass separated by a thin gap into which a fluid can be put, and was developed around the turn of the century to allow the

exploration of fluid-fluid boundaries, and in it a fluid is forced into the one in the cell under pressure. The resultant patterns consist of 'fingers' of the forced fluid penetrating the other fluid, have a fractal dimension. There might be economic implications in this, as oil is often extracted from oil bearing deposits by forcing water in to the oil bearing rocks - a giant Hele-Shaw cell.

Catalysts and Enzymes

Catalysts and enzymes are both chemical 'middle men'; they facilitate a chemical reaction to take place without changing their own chemical structure in the process. The general differences between the two are that enzymes, that mediate the chemical processes in living things, are made of protein and are deactivated or destroyed by temperatures outside a very precise range. The function of both is the same, however; to provide a 'jig' into which the molecules that are to react together can fit and react without extreme conditions.

A typical catalyst might be a finely divided metal powder, or a 'sponge' through which the chemical reactants can be pumped. An efficient catalyst is one that promotes the largest number of the desired chemical reactions for the smallest amount of catalyst. This usually requires a large surface area, and the best way to get that is to have a surface with a fractal dimension. In action, the catalyst adsorbs onto its surface one of the chemicals in the reaction mixture, and

effectively holds it still until a molecule of the other reactant comes along to react. Clearly, the more adsorption that can take place, the better.

With regard to enzymes, specific 'active sites' exist that hold reacting molecules in a particular orientation so that the desired reaction can take place. The enzyme molecules have a fractal surface which, incidentally, is lower than that of other protein molecules. This reflects the fact that once a reaction has taken place on an enzyme, the newly formed molecule must be able to get away fairly easily. A highly fractal surface might not allow this to happen.

Aggregation

There are certain physical processes where aggregation is an important part of the mechanism. Aggregation is the process by which a shape is built up by particles hitting an existing cluster of particles and sticking. This process is what occurs when particles build up on air filters, or other such processes where small particles can accumulate together. This process, called **Diffusion Limited Aggregation (DLA)** can be modelled with a computer program using 'particles' that move according to an algorithm like that used to simulate Brownian motion earlier in this Chapter. As soon as one of these simulated particles comes orthogonally adjacent to a seed pixel on the screen, it 'sticks' to the seed and thus starts the aggregation process, thus eventually forming an irregular, spidery aggregation. This shape is fractal; if you take two points

joined through the aggregation, the straight line distance is shorter than the path traversed by following the particles between the two particles of interest; this is analogous to the 'coastline' problems mentioned above.

DLA is a phenomenon that turns up in many physical systems; a typical example is the build up of copper on an electrode that is used in electrolysing copper sulphate solution. Copper is deposited on one of the electrodes in a way that is very much like the behaviour modelled by a DLA system. The formation of the irregular structure is a little unexpected at first; one might expect that a 'lump' would grow from the initial 'seed' with all possible attachment points to the seed being equally exploited. Well, that's how things start but after a while any protruding areas of the growth around the seed shield any areas close by from further attachment by particles; thus the protrusions are further favoured and the dendritic structure is formed. A further indication of the fractal nature of this structure is that it is self-similar. Larger runs of this sort of model with several millions of points will often lead to a structure that is more regular and is similar to a cross in structure. Listing 7.11 gives a simple DLA simulation that you might like to experiment with. One interesting experiment to try with a DLA model is to build a software snowflake. This can be done by introducing some rules into the model which influence the probability of a pixel being added to a certain point on the structure. Some guidelines for this are given in the program listing for the DLA; the results can be more snowflake like than the simple

Koch snowflake mentioned earlier in this chapter, due to the close similarity of the DLA model to the way in which real snowflakes grow in the atmosphere.

The DLA model can be used as a model for the growth of drainage basins or cities, as can a related model called the DBM model. DBM stands for Dielectric Breakdown Model and is analogous to the way in which an electrical discharge occurs through an insulator once a particular voltage - the dielectric breakdown voltage - is exceeded. The breakdown occurs from a point on the perimeter of the object carrying the electrical charge, and gives rise to a dendritic 'discharge path' similar to lightning.

In computing terms the DBM is best seen as a model which starts with a seed and then selects a place on the seed to add a pixel, starting from the seed. Future pixels are added to peripheral points on the structure, dependant upon some rules about the probability of a point being allowed to 'grow out' of a specific part of the structure. In the DLA model, growth is carried out by 'randomly walking' a pixel into the model until it hits a point on the structure to adhere to. The difference is subtle but real, and will impact on how the model evolves a structure over long periods of time.

DLA and DBM models have been used to simulate city growth, but there are clearly limitations on the use of a 'pure' fractal model like this. For example, it doesn't take into

account any restrictions of growth in the environment, such as rivers or mountains, by itself; these have to be programmed into the model as restrictions on the probability of a certain part of the structure being added to. Similarly, there's no input in simple DLA or DBM models for human intervention - the dreaded planners, etc.

A DBM model is given in listing 7.12.

Astronomy

A fine example of a system that can be modelled by a Cantor Set is to be found in the rings of the planet Saturn. Although the rings of the planet were discovered very soon after the development of the telescope, their nature was only unravelled over the years as better instruments were built and, eventually, satellites were used to take photographs of the ring structure. This eventually resolved the rings into a collection of smaller rings within rings, through which you could actually see the stars. A cross section of the rings would give us a pattern that looks very like a Cantor Dust.

A further example is to be found in the distribution of matter through space; it's not regular, but 'clumpy'. Matter in the universe tends to be gathered together in the collections of stars that we call galaxies. There is stuff in between, but not that much, and various astronomers have created models of the universe in which the galaxies follow a fractal arrangement in terms of distribution. There are problems with proving this, however, in that the dust, etc. between galaxies, as well as a variety of other problems, makes the accurate measurement of the position and speed of galaxies difficult.