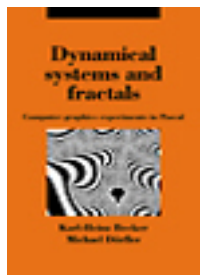


Dynamical Systems and Fractals

Computer Graphics Experiments with Pascal

Karl-Heinz Becker, Michael Dörfler, Translated by I. Stewart

Book DOI: http://dx.doi.org/10.1017/CBO9780511663031

Online ISBN: 9780511663031

Hardback ISBN: 9780521360258

Paperback ISBN: 9780521369107

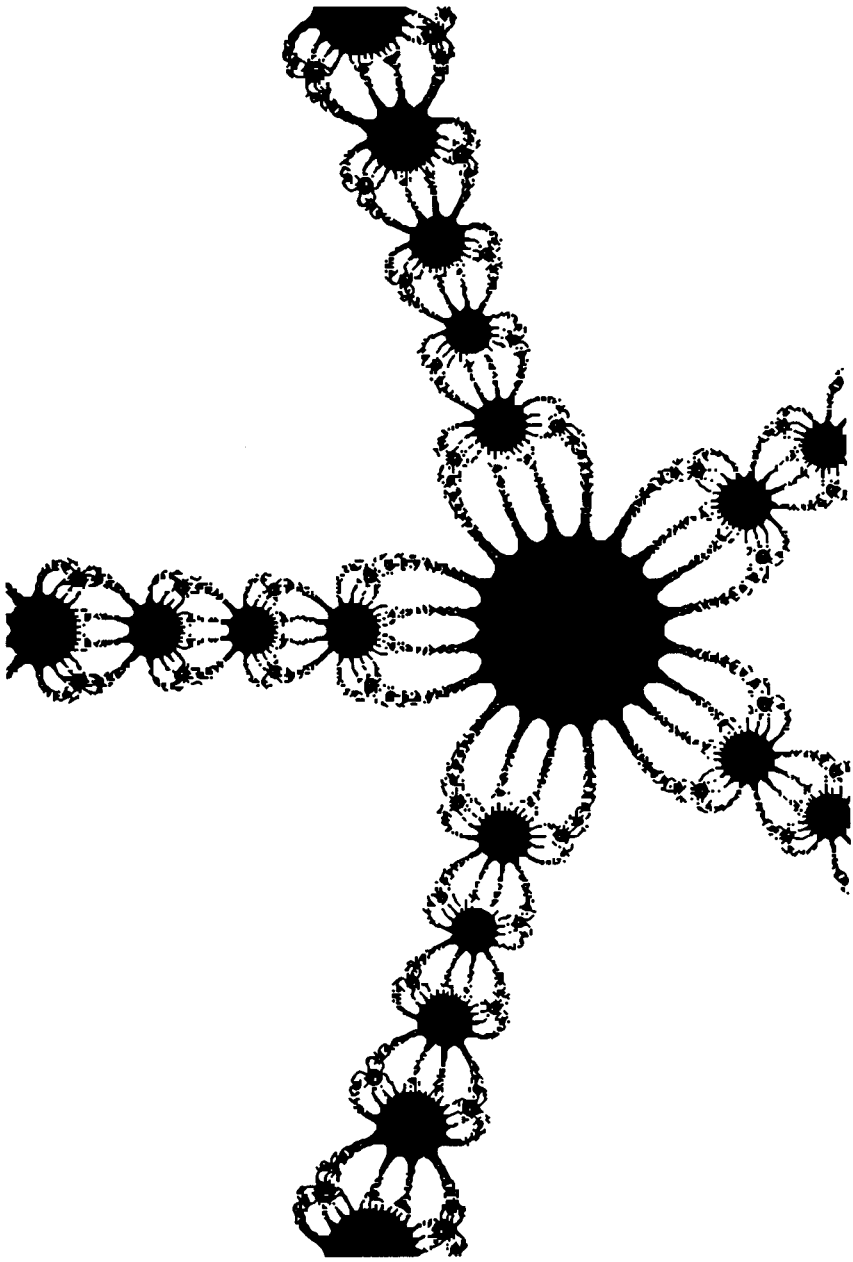
Chapter

5 - Complex Frontiers pp. 91-126

Chapter DOI: http://dx.doi.org/10.1017/CBO9780511663031.007

Cambridge University Press

5 Complex Frontiers



5.1 Julia and His Boundaries

We again state the question on the domains of influence of attractors. Where must we start the iteration, to be certain of reaching a given attractor? The precise boundaries between the initial zones should not be investigated. We are not exaggerating when we say that they are invisible. In order to get at least the attractors in the simplest possible fashion, we will use an arrangement as in Figure 5.1-1.

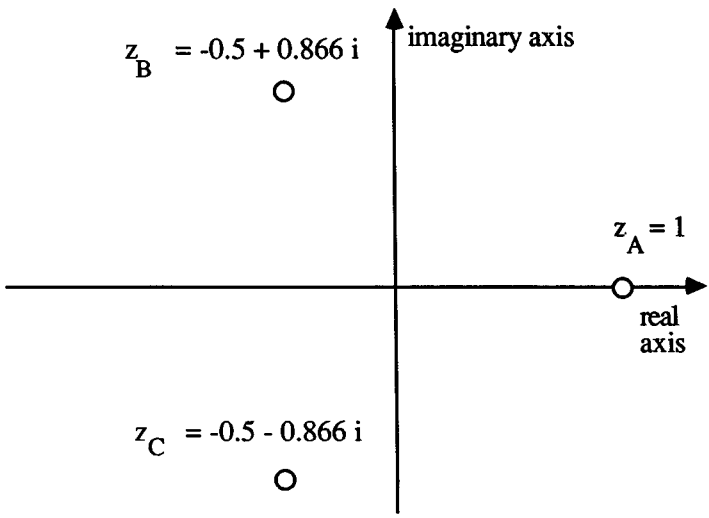


Figure 5.1-1 Position of three point attractors in the Gaussian plane.

The imaginary parts of z_B and z_C are irrational numbers. For instance

$$z_B = -\frac{1}{2} + \sqrt{-\frac{3}{4}}\ .$$

We will finish up at one of these three points from any initial value (except $z_0 = 0$) using the following iteration equation:

$$z_{n+1} = \frac{2}{3} z_n + \frac{1}{3z_n^2}\ .$$

Such points and such equations naturally don't fall from out of the blue. This one arises, for example, if you try to use Newton's method to find the complex zeros of the function[1]

$$f(z) = z^3 - 1.$$

It is easy to prove that each of the points z_A, z_B, z_C defined in Figure 5.1-1 satisfies

[1] If you do not see the connection here, take another look at §4.1.

$z^3 = 1$. We show this here for z_B:

$$z_B^3 = \left(-\frac{1}{2}+\sqrt{-\frac{3}{4}}\right)^3$$

$$= \left(-\frac{1}{2}+\sqrt{-\frac{3}{4}}\right) * \left(-\frac{1}{2}+\sqrt{-\frac{3}{4}}\right) * \left(-\frac{1}{2}+\sqrt{-\frac{3}{4}}\right)$$

$$= \left(-\frac{1}{2}+\sqrt{-\frac{3}{4}}\right) * \left(\frac{1}{4}-\frac{3}{4}+\sqrt{-\frac{3}{4}}\right) = \frac{1}{4}+\frac{3}{4} = 1 .$$

Thus z_A, z_B, z_C are the three *complex cube roots of unity.*

We have already dealt with calculation rules for complex numbers in §4.2. If we apply Newton's method, we get

$$z_{n+1} = \frac{2}{3}(x_n+i*y_n) + \frac{x_n^2-y_n^2-i*(2*x_n y_n)}{3*(x_n^2+y_n^2)^2} .$$

Thus for the complex number z_{n+1} we have

$$z_{n+1} = x_{n+1} + i*y_{n+1}$$

so that we can obtain equations for the real and imaginary parts x and y:

$$x_{n+1} = \frac{2}{3}x_n + \frac{x_n^2-y_n^2}{3*(x_n^2+y_n^2)^2} ,$$

$$y_{n+1} = \frac{2}{3}\left(y_n - \frac{x_n*y_n}{(x_n^2+y_n^2)^2}\right) .$$

In Program Fragment 5.1-1, we denote the values x_n and y_n by `xN` and `yN`, etc. The instructions for the two iteration equations can in principle be found as follows:

Program Fragment 5.1-1 (See also Program Fragments 5.1-2 and 5.1-3)

```
...
xN     := xNplus1;
yN     := yNplus1;
xNplus1  := 2*xN/3+(sqr(xN)-sqr(yN))
                 /(3*sqr(sqr(xN)+sqr(yN)));
yNplus1  := 2*yN/3-(2*xN*yN)/(3*sqr(sqr(xN)+sqr(yN)));
...
```

Using these,whichever initial value $z = x+i*y$ we start with, after suitably many iterations we end up near one of the three attractors. We can recognise which by looking at the distance from the known attractors.[2] If this distance is less than some

[2]If we do not know the attractors, we must compare the current value z_{n+1} with the previous value z_n. If this is less than epsilon, we are finished.

preassigned bound epsilon, we say 'we have arrived'.[3]

To formulate this test in Pascal we require a boolean function. It has the value `true` if we have already reached the relevant attractor, and otherwise the value `false`. For example, for the point z_C the test becomes the following:

Program Fragment 5.1-2

```
FUNCTION belongsToZc (x, y : real) : boolean;
   CONST
      epsqu = 0.0025;
   (* coordinates of the attractor zc *)
   xc =  -0.5;
   yc =  -0.8660254;
BEGIN
   IF (sqr(x-xc)+sqr(y-yc) < = epsqu)
      THEN belongsToZc := true
      ELSE belongsToZc := false;
END;  (* belongsToZc *)
```

The variable `epsqu` is the square of the small number 0.05, `xc` and `yc` are the coordinates of the attractor z_C, and `x` and `y` are the working coordinates which will be modified and tested during the investigation. The calculation for the other attractors requires similar programs.

In order to obtain an overview of the basins of the attractors and the boundaries between them, we explore point by point a section of the complex plane, which contains the attractors. We colour the initial point for the iteration series according to which basin it belongs to.

We give the method for drawing the boundaries in the following Program Fragment. In a few places we have slightly modified Example 5.1-1, to make the algorithm quicker and more elegant. In particular we do not need to distinguish between x_n and x_{n+1} in the contruction of the mathematical formula. Computers work with assignments, not with formulas.

Program Fragment 5.1-3

```
PROCEDURE Mapping;
VAR
   xRange, yRange : integer;
   x, y, deltaxPerPixel, deltayPerPixel : real;
BEGIN
   deltaxPerPixel := (Right - Left) / Xscreen;
   deltayPerPixel := (Top - Bottom) / Yscreen;
```

[3]In fact in Program Fragment 5.1-2 we compare the square of the distance with the square of epsilon, eliminating the need to extract a square root.

```
    y := Bottom;
    FOR yRange := 0 TO Yscreen DO
    BEGIN
       x := left;
       FOR xRange := 0 to Xscreen DO
       BEGIN
          IF JuliaNewtonComputeAndTest (x, y)
             THEN SetPoint (xRange, yRange);
             x := x + deltaxPerPixel;
       END;
       y := y + deltayPerPixel;
    END;
END; (* Mapping *)
```

In contrast to the more linear structures of the previous chapter, we no longer compute a hundred points in each series. The 120 000 points of the chosen section[4] obviously need more computing time. A complete calculation requires, in some circumstances, more than an hour. The procedure `Mapping` searches through the screen area one pixel at a time. For each screen point it computes the universal coordinates `x` and `y`. It passes these variables to a functional procedure, which in this case is called `JuliaNewtonComputeAndTest`. We use such an unequivocal name to distinguish this procedure from others which play similar roles in later programs. The corresponding screen point is coloured, or not, according to the result of this function. The procedure `Mapping` uses 7 global variables, which we already know from other problems:

`Left, Right, Bottom, Top,`
`MaximalIteration, Xscreen, Yscreen.`

For a computer with 400 × 300 pixels on the graphics screen, we might for example set up the computation as follows:

`Xscreen := 400; Yscreen := 300;`
`Left := -2.0; Right := 2.0; Bottom := -1.5; Top := 1.5;`

Program Fragment 5.1-4

```
FUNCTION JuliaNewtonComputeAndTest (x, y : real) : boolean;
   VAR
      IterationNo : integer;
      finished : boolean;
      xSq, ySq, xTimesy, denominator : real;
      distanceSq, distanceFourth : real;
BEGIN
   StartVariableInitialisation;
```

[4]We here refer to a section of 400 × 300 pixels.

```
REPEAT
   compute;
   test;
UNTIL (IterationNo = MaximalIteration) OR finished;
   distinguish;
END; (* JuliaNewtonComputeAndTest *)
```

Now the procedure `JuliaNewtonComputeAndTest` is formulated in reasonable generality. It makes use of four local procedures. The first sets up the values for the local variables:

Program Fragment 5.1-5

```
PROCEDURE startVariableInitialisation;
BEGIN
   finished := false;
   iterationNo := 0;
   xSq := sqr(x);
   ySq := sqr(y);
   distanceSq := xSq + ySq;
END (* startVariableInitialisation *)
```

The next procedure does the actual computation.

Program Fragment 5.1-6

```
PROCEDURE Compute;
BEGIN
   IterationNo := IterationNo + 1;
   xTimesy := x*y;
   distanceFourth   := sqr(distanceSq);
   denominator   :=
           distanceFourth+distanceFourth+distanceFourth;
   x          := 0.666666666*x + (xSq-ySq)/denominator;
   y          := 0.666666666*y -
                    (xTimesy+xTimesy)/denominator;
   xSq        := sqr(x);
   ySq        := sqr(y);
   distanceSq := xSq + ySq;
END;
```

A few tricks have been used so that the most time-consuming calculation steps, especially multiplication and division, are not carried out twice. For example the expression $\frac{2}{3}x$ is not coded as

```
2*x/3.
```

In this form, the integer numbers 2 and 3 must first be converted to real numbers every time the procedure is used. Further, a division takes more time than a multiplication. So a more efficient expression is

```
0.666666666*x.
```

After each iterative step we must test whether we have 'arrived' near enough to one of the attractors. Moreover, we must be careful that the numbers we are calculating with do not go outside the range within which the computer can operate. If that happens, we stop the calculation.

Program Fragment 5.1-7

```
PROCEDURE test;
BEGIN
   finished := (distanceSq < 1.0E-18)
      OR (distanceSq > 1.0E18)
         OR belongsToZa (x,y)
            OR belongsToZb (x,y)
               OR belongsToZc(x,y);
END;
```

Finally we must distinguish what should be drawn[5]. The points which belong to the boundary are those which, after that maximal number of iterations, have not converged to any of the three attractors.

Program Fragment 5.1-8

```
PROCEDURE distinguish;
BEGIN
   (* does the point belong to the boundary? *)
   JuliaNewtonComputeAndTest :=
      IterationNo = MaximalIteration;
END;
```

We interpret Program Fragment 5.1-8 as follows. We include all points in the boundary for which the computation, after a given number of iterations, has not reached an attractor. In Figure 5.1-2 this `maximalIteration = 15`. In all other cases the

[5]We can carry out the investigation most easily when the computer has a colour graphics screen. Then each basin of attraction is given a colour, and the boundaries are easily visible.

equality condition is not fulfilled, so the iteration is stopped before the variable `iterationNo` gets that high.

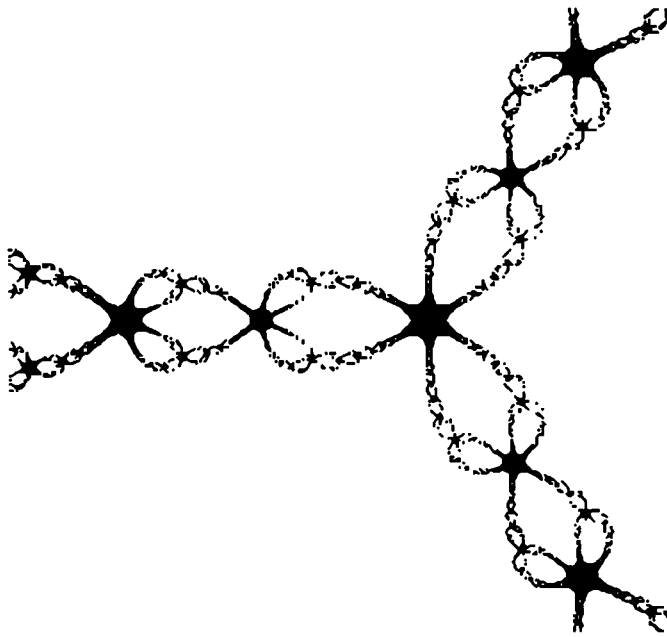


Figure 5.1-2 Boundary between the three basins after 15 iterations.

If instead of the boundary we draw the basin of attraction of one of the three attractors, as in Figure 5.1-3, we can use the functional procedure `belongsToZc` defined in Program Fragment 5.1-2.

Program Fragment 5.1-9

```
PROCEDURE distinguish
BEGIN
   (* does the point belong to the basin of zC? *)
   JuliaNewtonTestAndCompute := belongsToZc (x,y)
END;
```

It is of course entirely arbitrary, that we have chosen to draw the basin of the attractor z_C. In Exercise 5.1-2 we give hints for computing the other two basins. Perhaps you can already guess what form they will take?

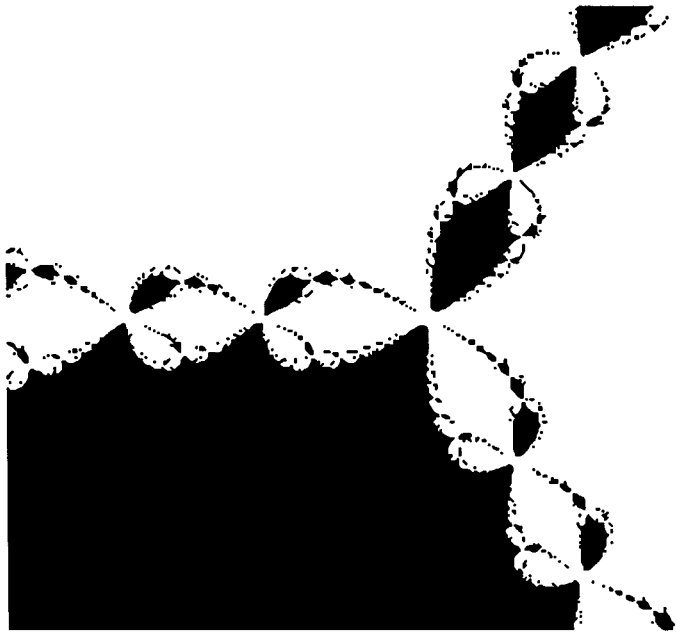


Figure 5.1-3 Basin of the attractor z_C.

In a different style of drawing, as in Figure 5.1-4, we take into account the number of iterations required to reach any particular one of the three atractors. A point is then coloured if this requires an odd number of iterations. In Figure 5.1-4 you should look for the three point attractors (see Figure 5.1-1). They are surrounded by three roughly circular areas. All points in these regions of the complex plane have already reached the attractor, to the relevant degree of accuracy, after one iteration.

From there we see in turn alternating regions of black and white, from which we reach the attractor in 2, 3, 4, ... steps. In other words, the black regions correspond to initial values which take and odd number of steps to reach an attractor. In this way we obtain pictures reminiscent of *contour lines* on a map. If you think of the attractors as flat valleys and the boundaries between them as mountain ranges, this interpretation becomes even better. The peaks become ever higher, the longer the computation, deciding to which attractor the point belongs, lasts.

If the contour lines get too close together, there is always the possibility of drawing only every third or every fourth one of them.[6]

[6] In the Program Fragment we use the MOD function of Pascal. It gives the remainder upon division, e.g. `7 MOD 3 = 1`.

Program Fragment 5.1.10

```
PROCEDURE distinguish;
BEGIN
   (* does the point reach one of the three attractors *)
   (* after an odd number of steps? *)
      JuliaNewtonComputeAndTest : =
         (iterationNo < maximalIteration)
           AND odd(iterationNo);
END;

PROCEDURE distinguish;
BEGIN
   (* does the point reach one of the three attractors *)
   (* after a number of steps divisble by 3? *)
   JuliaNewtonComputeAndTest : =
      (iterationNo < maximalIteration)
        AND (IterationNo MOD 3 = 0);
END;
```

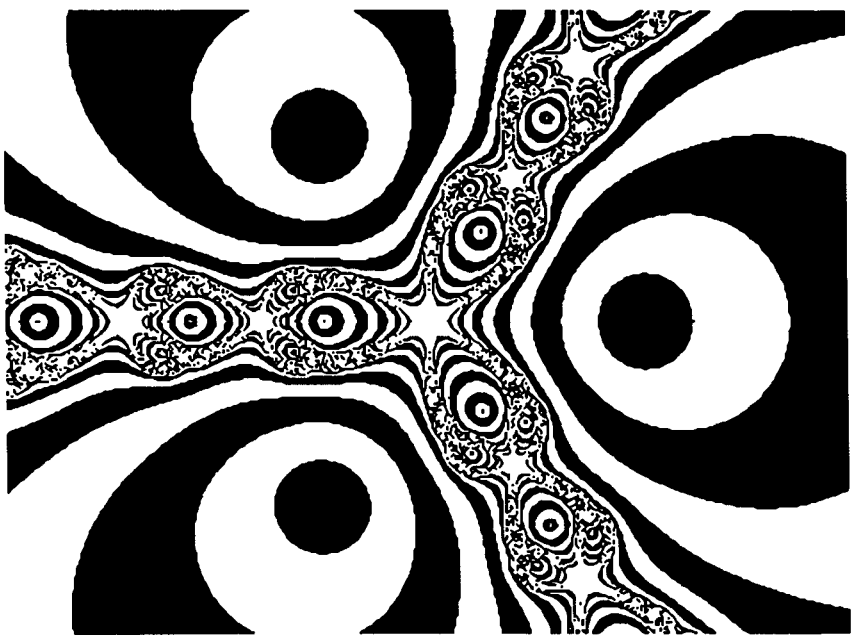


Figure 5.1-4 Contour lines.

```
PROCEDURE distinguish;
BEGIN
   JuliaNewtonComputeAndTest : =
      (IterationNo = MaximalIteration) OR
         ((IterationNo < Bound)
            AND (IterationNo MOD 3 = 0));
END;
```

The three variations show how you can combine these methods of graphical representation to give new designs (Figure 5.1-5). The global variable `Bound` should be about half as big as `MaximalIteration`.

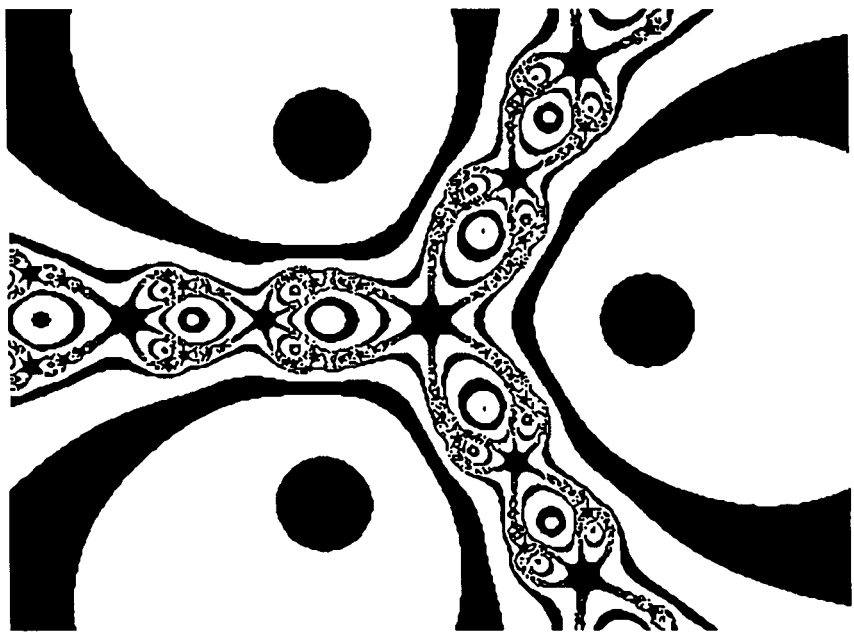


Figure 5.1-5 Every third contour line, and the boundaries, in a single picture.

There is a further possibility for drawing the pictures which we will only sketch here (see Execise 5.1-8). It is possible to show all three basins in the same picture. Different grey tones represent the different basins (Figure 5.1-6).

If you look at the basin of the attractor z_C (Figure 5.1-3) it may not be at all clear how the regions within it are divided up. Near the attractor itself the region is a connected piece. But at the boundaries they seem to get increasingly confused with each other. The divison nevertheless exists: it only *appears* confused. We already know from the example in Chapter 4 that the basins of attraction can never overlap.

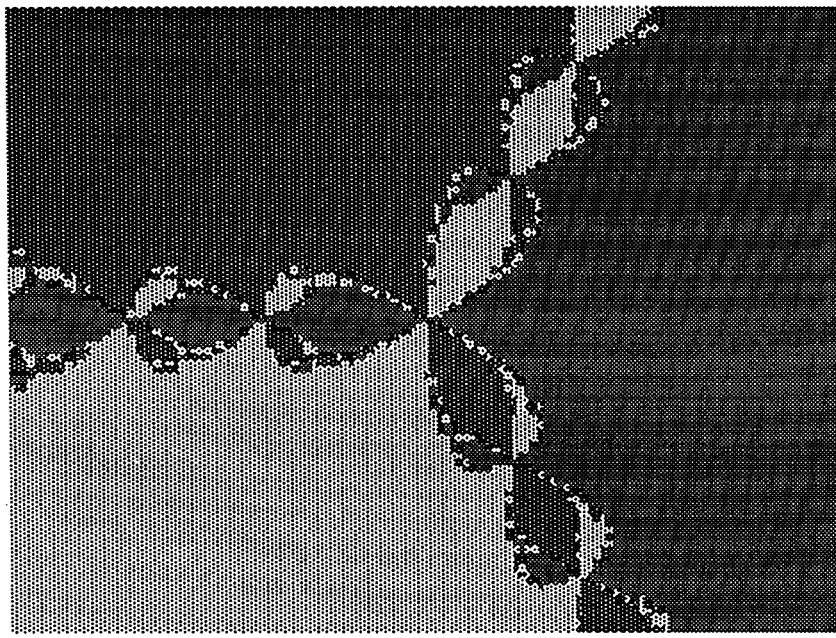


Figure 5.1-6 The basins of the three attractors z_A, z_B, z_C.

Peitgen and Richter, in Research Group in Complex Dynamics, Bremen (1984a), pp. 19, 31, describe this phenomenon by equating the three basins of attraction with the territories of three superpowers on a fantasy planet:

> 'Three power centres have divided up their sphere of influence and have agreed to avoid simple boundaries with only two adjoining regions: each boundary point must be adjacent to all three countries. If the computer graphic did not show a solution, one would scarcely believe that it existed. The key to the trick is that everywhere two countries come next to each other, the third establishes an outpost. This is then in turn surrounded by tiny enclaves of the other powers - a principle which leads to ever smaller similar structures and avoids a flat line.'

These 'similar structures' are what we have already encountered as self-similarity: here we meet them again. Meanwhile this behaviour has become so natural to us that it is no longer a surprise!

When the basins of attraction seem so ragged, there is one thing in the picture that still hangs together: the boundary. Here we observe a member of an entirely new class of geometrical figures, a *fractal*. This concept has been developed over the past twenty years by the Franco-Polish mathematician Benoît B. Mandelbrot. By it he refers to structures which cannot be described by the usual forms such as lines, surfaces, or solids.

On the one hand, the boundary in Figure 5.1-2 is certainly not a surface. For each

point that we have investigated, after sufficiently many iterations, it can be assigned to one of the three attractors. Only the point (0,0), the origin, obviously belongs to the boundary. Therefore the boundary always lies between the screen pixels, and has no 'thickness'.

On the other hand, the boundary is certainly not a line. Try to work out its length! In any particular picture it looks as though this can be done. But if we magnify small pieces, we find a remarkable phenomenon: the more we magnify - that is, the more closely we look - the longer the boundary becomes. In other words, the boundary is infinitely long and has zero width. To such structures, 'between' lines and surfaces, mathematicians attribute a *fractal dimension* which is not a whole number, but lies between 1 and 2.

Two properties, closely related, are characteristic of fractals:

- *Self-similarity*, that is, in each tiny piece we observe the form of the entire shape.
- *Irregularity*, that is, there are no smooth boundaries. Lengths or areas cannot be determined.

Once this concept had been drawn to the attention of researchers, they soon found many examples of it in Nature. The coastline of an island is a fractal from the geometric point of view.

It is possible to read off the length of a coastline from a map with a given scale. But if we use a map with a different scale, that can change the result. And if we actually go to the beach, and measure round every rock, every grain of sand, every atom... we encounter the same phenomenon. The more closely we look, that is, the larger the scale of the map, the longer the coastline seems to be.

Thus many natural boundaries, by the same principle, are fractal.

Fractals with dimension between 2 and 3 are the main surfaces that will concern us. Every morning when we look in the mirror we notice that:

- Skin is fractal - especially when one is 'getting on in years'. It has the basic purpose of covering the body with the smallest possible surface. But for several other reasons a crumpled structure is preferable.

It is even more obvious, when we leave the house:

- Clouds, trees, landscapes, and many other objects, when viewed at a different level, appear fractal. To investigate these is the subject of rapidly growing research activity in all areas of natural science.

The consequences of this discovery for biology are explained in Walgate (1985) p.76. For small creatures, for example insects on a plant, the living space grows in an unsuspected fashion. Consider as a model case two species, differing in size by a factor of 10. They must expand their population on one of the available surfaces. There should therefore be 100 times as many small creatures as large. But this argument takes on a different appearance if we think of the surface of a leaf from the

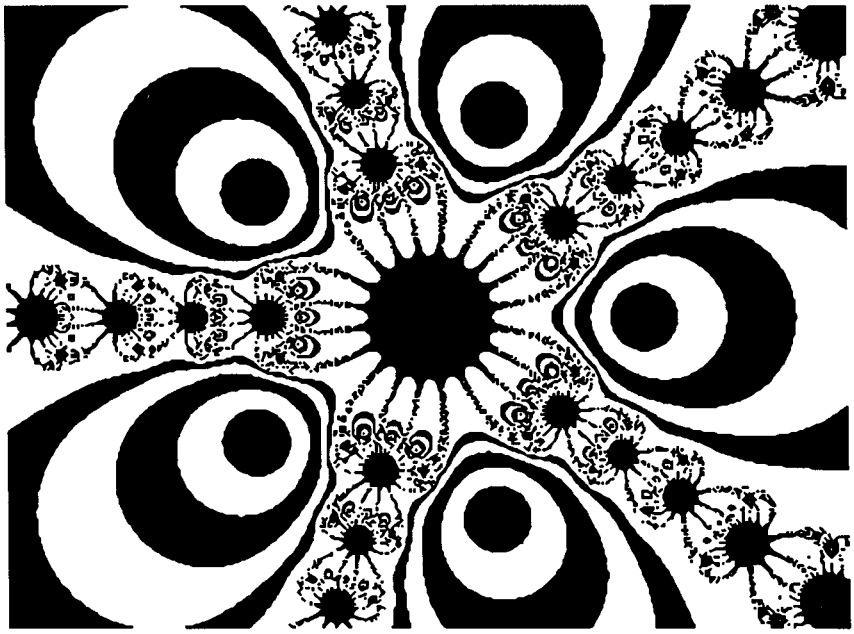


Figure 5.1-7 A Julia set with fivefold symmetry.

fractal point of view. Then we find about 300 to 1000 times as many small insects in the same space that a single creature, 10 times as big, would occupy. This has actually been verified experimentally: see Morse *et al.* (1985). Thus there are many more tiny insects on a plant than had hitherto been thought. 'The smaller the organisms, the larger the world in which they live.'

An entire series of physical processes, whose detailed description raises many other problems, are of a fractal nature. Examples include Brownian motion and the study of turbulence in the flow of gases and fluids. This is true even though natural fractals can obviously display self-similarity and piecewise crumpling only up to some limit, whereas mathematical fractals have these properties completely and at every level.

As often happens in mathematics, parts of the scientific preparatory work had already been carried out some time ago. But the discoveries of the French mathematicans Pierre Fatou and Gaston Julia were already becoming somewhat forgotten. In honour of the French researcher who studied the iteration of complex functions prior to 1920, fractal boundaries (Figure 5.1-2) are known as *Julia sets*.

The possibilities raised by the computer now make it possible, for the first time, to investigate this fundamental area. It is indisputably to the credit of the Bremen Research Group of H. O. Peitgen and P. Richter to have excited attention - not only in the technical world - with the results of the 'computer cookery' of their work in graphics.

Their pioneering work is well known internationally.

The investigation of Newton's method carried out above, for the equation

$$z^3-1 = 0,$$

can be applied equally well to other functions. To show you one result, Figure 5.1-7 indicates what happens for

$$z^5-1 = 0.$$

You will find some hints about it in Exercises 5.1-4 and 5.1-5.

In other exercises for this chapter we suggest experiments, leading to innumerable new forms and figures, which we have scarcely been able to explore ourselves. So varied are the possibilities opened up, that you are guaranteed to produce pictures that nobody has ever seen before!

Computer Graphics Experiments and Exercises for §5.1

Exercise 5.1-1

Apply Newton's method to the function

$$f(z) = z^3-1.$$

In the iteration equation

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)}$$

insert the expressions for the functions $f(z)$ and $f'(z)$. Show that this leads to the equation

$$z_{n+1} = \frac{2}{3} z_n + \frac{1}{3z_n^2}.$$

For the complex numbers z_A to z_C in Figure 5.1-1 compute the value of z^3. What happens?

Exercise 5.1-2

On the basis of Program Fragments 5.1-1 to 5.1-3, write a program that allows you to compute the basins and boundaries of the three attractors in Figure 5.1-3.

Next investigate and draw the basin of the attractor

$$z_A = 1$$

with the iteration formula

$$z_{n+1} = \frac{2}{3} z_n + \frac{1}{3z_n^2}.$$

Compare the resulting picture with those that you get for

$z_B = -0.5 + 0.8660254 * i$

and

$z_C = -0.5 - 0.8660254 * i,$

see Figure 5.1-3.

It is not a coincidence that the pictures resemble each other in various ways. The reason involves the equivalence of the three complex cube roots of unity.

Exercise 5.1-3

Using the method of magnifying detail, investigate sections of Figures 5.1-1 to 5.1-6. Choose regions that lie near boundaries. In all these cases we find self-similarity!

Check that if we increase the number of iterations (and also increase the admittedly not short computation time) self-similar structures continue to appear.

Exercise 5.1-4

If you want to apply Newton's method with powers higher than 3, for example to equations such as $z^4-1 = 0$ or $z^5-1 = 0$, you must know the appropriate complex roots of 1. These give the positions of the attractors.

In general the nth roots of 1 are given by

$$z_k = \cos\left(\frac{k*360^\circ}{n}\right) + i*\sin\left(\frac{k*360^\circ}{n}\right)$$

where k runs from 0 to $n-1$. Produce (with the aid of a computer program) a table of roots of unity z_n up to $n = 8$.

Exercise 5.1-5

Investigate and draw Julia sets resulting from Newton's method for

$z^4-1 = 0$ and $z^5-1 = 0$.

The mathematical difficulties are not insuperable. In Program Fragment 5.1-3 you must construct a modified function `JuliaNewtonComputeAndTest`. If you cannot solve this and the following exercises straight away, you can take a look at §6.4. There we summarise the important rules for calculating with complex numbers.

Exercise 5.1-6

We apply Newton's method and thereby get beautiful, symmetric computer graphics. Although the above examples seem to be well founded, they are certainly not to be thought of as 'sacred'. As with our excellent experience in changing formulas in the previous chapter, so too we can change Newton's method somewhat.

Starting from z^p-1, we insert in the equation

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)}$$

the factor v (which is of course complex), obtaining:

$$z_{n+1} = z_n - v*\frac{f(z_n)}{f'(z_n)} .$$

First decide for yourself, without drawing, what the attractors will be. How far do they correspond to the complex roots of unity, that is, with the solutions to $z^p - 1 = 0$?

Start with values v that lie near 1. Investigate the influence of v on the form of the Julia sets.

Exercise 5.1-7

Investigate the changes that occur if we modify the formula by putting an imaginary summand $i*w$ in the denominator:

$$z_{n+1} = z_n - \frac{f(z_n)}{i*w+f'(z_n)} .$$

Again work out, without drawing, what the attractors are. Draw the way the basins of attraction fit together.

How far does the modification of the equation influence the symmetry of the corresponding pictures?

Exercise 5.1-8

If you wish to work with grey tones, as in Figure 5.1-6, but these are not directly available on your computer, Figure 5.1-8 shows how to proceed.

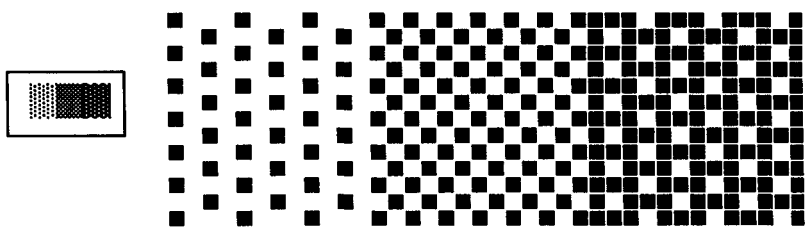


Figure 5.1-8 Grey shades, normal size (left) and magnified, to show individual pixels.

To achieve this, not every point should be coloured, even when after calculation it is known in which basin it belongs. The drawing depends in which basin and also in which row or column of the screen a point occurs.

In particular the conditions for drawing points should be:

```
IF (odd(row) AND (column MOD 4 = 0))      (* light grey *)
   OR (NOT odd (row) AND (column MOD 4 = 2)) THEN ...

IF (odd(row) AND odd(column))             (* medium grey *)
   OR (NOT odd (row) AND NOT odd(column)) THEN ...

IF (odd(row) AND (column MOD 4 <> 0))     (* dark grey *)
   OR (NOT odd (row) AND (column MOD 4 <> 2)) THEN ...  .
```

5.2 Simple Formulas give Interesting Boundaries

The various pictures of previous pages, and the innumerable sections and variations that you have derived from them, owe their existence to the calculating performance of the computer. Without this, no one would have made the effort to perform hundreds of thousands of completely different calculations - especially since it appears that the pictures become ever more ragged and complicated, the more complicated (and ragged?) the underlying formulas become.

Again, this conjecture was first considered by B. B. Mandelbrot,[7] who already knew in 1979-80 that the production of 'richly structured' pictures does not necessarily need complicated mathematical formulas. The important thing is for the iteration formula to be nonlinear. It can thus contain polynomials of the second or higher degree, or transcendental functions, or other such things.

The simplest nonlinear iteration equation that leads to nontrivial results was suggested by Mandelbrot:

$$z_{n+1} = z_n^2 - c.$$

That is, we obtain a new member of the iteration series, by taking the previous one, squaring it, and subtracting from the square a number c.

Until now, in previous chapters, we have investigated relatively complicated formulas, having one simple parameter, which can be changed. In contrast, in this Mandelbrot formula we have a very simple equation, containing not one but two parameters. These are the real and imaginary parts of the complex[8] number c. The equation

$$c = c_{\text{real}} + \mathrm{i} * c_{\text{imaginary}}$$

holds for c. For the complex number z we have as before

$$z = x + \mathrm{i} * y.$$

In other words, we can write the formula as

$$\begin{aligned} z_{n+1} &= x_{n+1} + \mathrm{i} * y_{n+1} \\ &= f(z_n) \\ &= (x_n^2 - y_n^2 - c_{\text{real}}) + \mathrm{i} * (2 * x_n * y_n - c_{\text{imaginary}}). \end{aligned}$$

[7] See the very readable article *Fractals and the Rebirth of Iteration Theory* in Peitgen and Richter (1986) p. 151.

[8] If the mathematical concepts of real, imaginary, and complex numbers still cause difficulty, we again advise you to reread §4.2.

These formulas can be set up within a Pascal program:

Program Fragment 5.2-1

```
...
xSq   := sqr(x)
ySq   := sqr(y)
y     := 2*x*y - cImaginary
x     := xSq - ySq - cReal

...
```

It is important to keep these statements in the given order, or else information will get lost.[9] In particular, contrary to alphabetical order and other custom, we must first compute the value of y and then that for x (because x appears in the equation for y).

Just like those for Newton's method, these iteration formulas seem to be fairly harmless and not very complicated. But it is not so easy to grasp the possibilities for investigation that arise when a single parameter is changed. Not only the components of c play a role, but also the initial value of z, the complex number

$$z_0 = x_0 + i * y_0.$$

Thus we have four quantities to change and/or draw, namely

$$x_0,\ y_0,\ c_{real},\ c_{imaginary}.$$

On a sheet of paper we can show only two dimensions, so we must choose two out of the four as the basis of our drawing. As in the previous chapter these are the components x_0 and y_0 of the complex initial value z_0. The computation of these pictures takes the following form. The position of a point on the screen (in screen coordinates) represents the components x_0 and y_0 (in universal coordinates). For a given value

$$c = c_{real} + i * c_{imaginary}$$

the iteration is carried out. As a result, x and y change, and with them z. After a given number of iterations we colour the point corresponding to z according to the result. If it is not possible to use different colours, we can still use black-and-white or grey tones. The method is then repeated for the next value of z_0.

The Mandelbrot formula is so constituted that it has only two attractors. One of them is 'infinite'. By the attractor '∞', we mean that the series of numbers $f(z)$ exceeds any chosen value. Since for complex numbers there is no meaningful concept of larger/smaller, we understand by this that the square of the modulus, $|f(z)|^2$, exceeds any given value after sufficiently many steps. It does not matter much which value, and we take for this bound the number 100.0 (defined as a real number so that the comparison does not take too long).

The variable `MaximalIteration` can be seen as a measure of how much patience we have. The bigger this number, the longer the calculation takes, and the longer we must wait for the picture. But if `MaximalIteration` is too small, the drawing is very

[9] You will still get interesting pictures - but not the ones intended.

poor. If $|f(z)|^2$ stays below the value 100.0 after this number of steps, we declare the attractor to be 'finite' or effectively 'zero'.

In fact the situation is somewhat more complicated. Only in some cases is $f(z) = 0$ a limiting value. In other cases we encounter a different number $z \neq 0$, a so-called *fixed point*. It lies near the origin, and satisfies $f(z) = z$. Further, there are also attractors that are not single points, but which consist of 2, 3, 4,... or more points. For an attractor with the period 3 we have

$$f(f(f(z))) = z.$$

Despite these complications, we concentrate on only one important thing: that the attractor be finite, that is, that the series should not exceed the value 100.0.

Each picture shows a section of the complex z-plane. We make the initial values z_0 the basis of our drawing. At each iteration z changes, and we give it the value computed for $f(z)$ at the previous stage.

The complex parameter c must be kept constant throughout a given picture.

To repeat: there exist two attractors, zero and infinity, whose basins of attraction border each other. As has already been explained in §5.1, we can call these complex boundaries Julia sets. For the rest of this chapter we will be concerned with their graphical representation.

Two different complex numbers c_1 and c_2 generate two different Julia sets, and thus two different graphics! The variety of possible complex numbers produces an uncountable number of distinct pictures.

The graphical appearance of these sets can be very varied too, because the form of the Julia set depends strongly on the value of the parameter c.

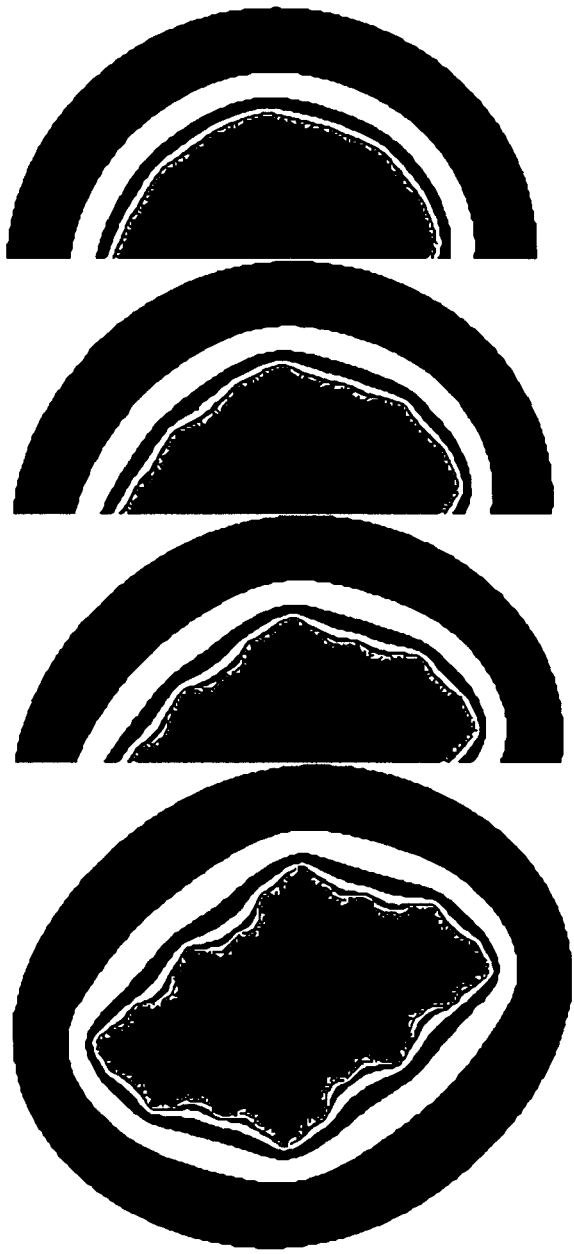
For some of the pictures on the following two pages only the upper halves are drawn. They are symmetric about the origin, as can be seen in Figure 5.2-4 (lower left) and Figure 5.2-5 (upper right). In the series Figures 5.2-1 to 5.2-8, the complex parameter c takes the values

$$c_1 = 0.1 + 0.1*\mathrm{i},\ c_2 = 0.2 + 0.2*\mathrm{i},\ \ldots,\ c_8 = 0.8 + 0.8*\mathrm{i}.$$

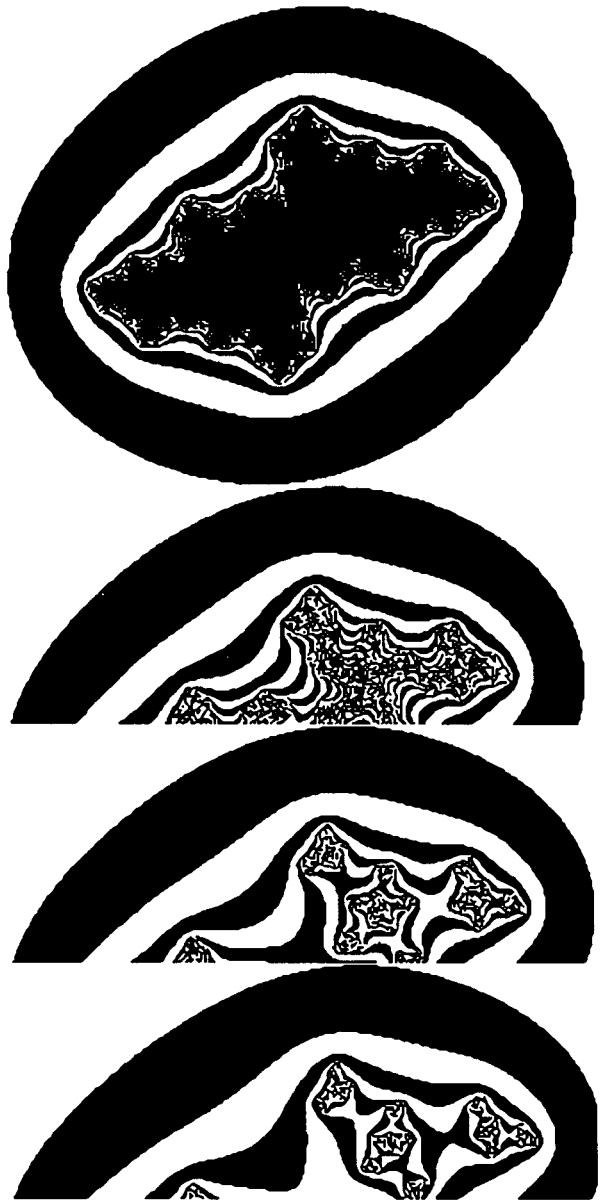
We start with $c = 0$, surely the simplest case. Without a computer it is easy to see that the boundary is the unit circle. Each z-value, whose modulus is greater than 1, has ∞ as an attractor. Each value $|z| < 1$ has 0 as attractor and should be drawn. We also colour the points with $|z| = 1$ since these too lead to a finite attractor. In terms of contours nothing much can be distinguished in this case, and we do not give a picture. But in the majority of cases, when we change c, the resulting picture yields contours. If c is increased in steps of $0.1 + 0.1 * \mathrm{i}$, we get in turn Figures 5.2-1 to 5.2-8.

A program to generate these figures obviously has certain parts in common with the graphics programs that we have already encountered (see Program Fragment 5.2-2). The remainder can be found in Program Fragment 5.1-3. Make the procedure now called `JuliaComputeAndTest` call the procedure `Mapping`. The surrounding program must supply the global variables `cReal` and `cImaginary` with suitable values.

In comparison with Program Fragment 5.2-1 a small improvement has been made.



Figures 5.2–1 to 5.2–4 Julia sets.



Figures 5.2–5 to 5.2–8 Julia sets.

As a result we save one multiplication per iteration and replace it by an addition, which is computed significantly faster.

Below we describe a complete functional procedure `JuliaComputeAndTest`, containing all relevant local functions and procedures.

Program Fragment 5.2-2

```
FUNCTION JuliaComputeAndTest (x, y : real) : boolean;
   VAR
      iterationNo   : integer;
      xSq, ySq, distanceSq   : real;
      finished : boolean;
   PROCEDURE startVariableInitialisation;
   BEGIN
      finished := false;
      iterationNo := 0;
      xSq := sqr(x); ySq := sqr(y);
      distanceSq := xSq + ySq;
   END;  (* startVariableInitialisation *)

   PROCEDURE compute;
   BEGIN
      iterationNo := iterationNo  + 1;
      y := x * y;
      y : = y + y - cImaginary;
      x := xSq - ySq - cReal;
      xSq := sqr(x); ysQ := sqr(y);
      distanceSq := xSq + ySq;
   END;  (* compute *)

   PROCEDURE test;
   BEGIN
      finished := (distanceSq > bound);
   END;  (* test *)

   PROCEDURE distinguish;
   BEGIN  (* does the point belong to the Julia set? *)
      JuliaComputeAndTest :=
         iterationNo = maximalIteration;
   END;  (* distinguish *)

BEGIN (* JuliaComputeAndTest *)
   startVariableInitialisation;
   REPEAT
```

```
        compute;
        test;
      UNTIL (iterationNo = maximalIteration) OR finished;
      distinguish;
  END; (* JuliaComputeAndTest *)
```

We recognise familiar structures in the Program Fragment. In the main part the screen section is scanned point by point and the values of x and y passed to the functional procedure `JuliaComputeAndTest`. These numbers are the starting values x_0 and y_0 for the iteration series. The global constants `cReal` and `cImaginary` control the form of the set that is drawn.

Each new pair of numbers generates a new picture!

For mathematicians, it is an interesting question, whether the Julia set is connected. Can we reach every point in the basin of the finite attractor, without crossing the basin of the attractor ∞? The question of connectedness has been answered in a difficult mathematical proof, but it can be studied rather more easily with the aid of computer graphics.

In Figures 5.2-7 and 5.2-8 we certainly do not have connected Julia sets. The basin of the attractor ∞ can be seen from the contour lines. It cuts the Julia set into many pieces. In Figures 5.2-1 to 5.2-5 the Julia set is connected. Is this also the case in Figure 5.2-6? We ask you to consider this question in Exercise 5.2-1.

As another example we will demonstrate what effect an extremely small change of c can have on the picture of the Julia set. We choose for the two parameters c_1 and c_2 the following values, which differ only by a very tiny amount:

$c_1 = 0.745\,405\,4 + i*0.113\,006\,3$

$c_2 = 0.745\,428\,0 + i*0.113\,009\,0$

Figures 5.2-9 and **5.2-10** Julia sets for c_1 and c_2.

In both cases the Julia sets appear the same (Figures 5.2-9, 5.2-10). The pictures that follow are drawn with contour lines, so that the basin of the attractor ∞ can be

deduced from the stripes. The object of investigation is the indicated spiral below and to the right of the middle of Figures 5.2-9 and 5.2-10. Figures 5.2-11 (the same as Figure 5.2-9 magnified 14 times) and 5.2-12 (magnified about 135 times) are here drawn only for c_1. These magnified pictures are also barely distinguishable from those for c_2.

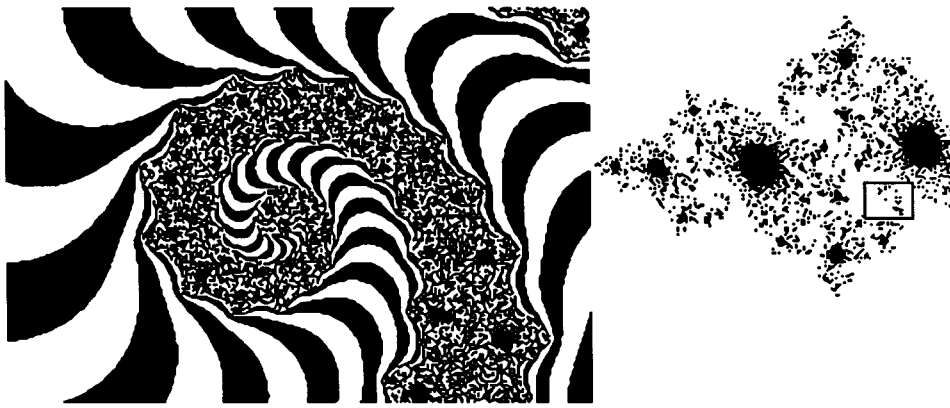


Figure 5.2-11 Julia set for c_1. Section from Figure 5.2-9.

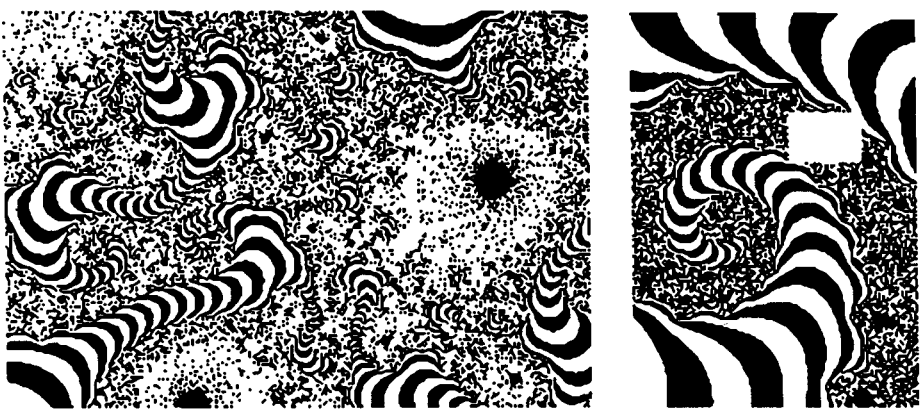


Figure 5.2-12 Julia set for c_1. Section from Figure 5.2-11.

In Figure 5.2-13 (for c_1) and 5.2-14 (for c_2) the pictures of the two Julia sets first begin to differ in detail in the middle. After roughly 1200-fold magnification there is no visible difference at the edge of the picture, at least up to small displacements.

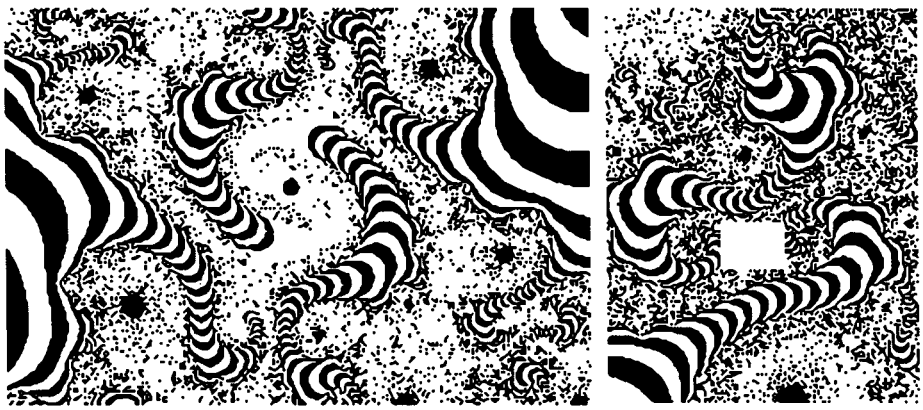


Figure 5.2-13 Julia set for c_1. Section from Figure 5.2-12.

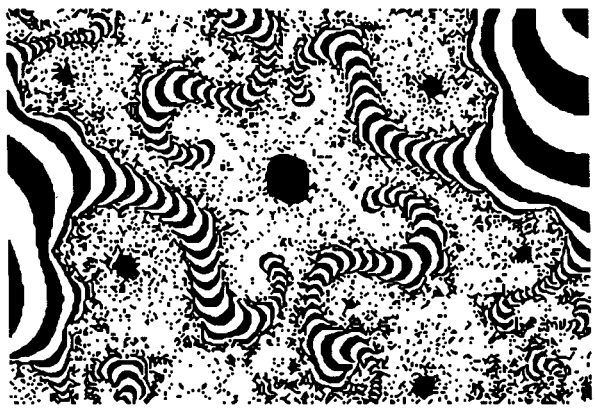


Figure 5.2-14 Julia set for c_2. Section corresponding to Figure 5.2-13.

The extreme 6000-fold magnification in Figure 5.2-15 (for c_1) and Figure 5.2-16 (for c_2) confirms the distinction.

The stripes indicating the basin of attraction of ∞ are connected together in Figure 5.2-15 from top to bottom. At this point the figure is divided into a left and a right half. As a result the Julia set is no longer connected. It is different for c_2 in Figure 5.2-16. The basin of attraction of ∞ 'silts up' in ever more tiny branches, which do not touch each other. Between them the other attractor holds its ground.

We must magnify the original Figures 5.2-9 and 5.2-10, with an area of about 60 cm^2, so much that the entire figure would cover a medium-sized farm (21 hectares). Only then can we notice the difference.

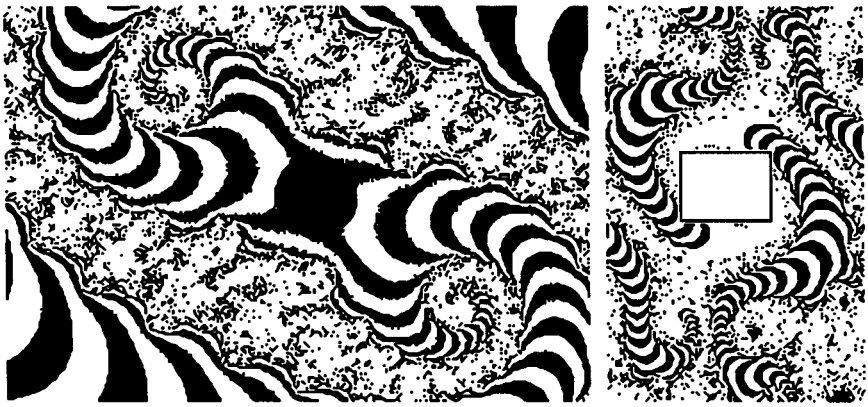


Figure 5.2-15 Julia set for c_1. Section from Fig 5.2-13.

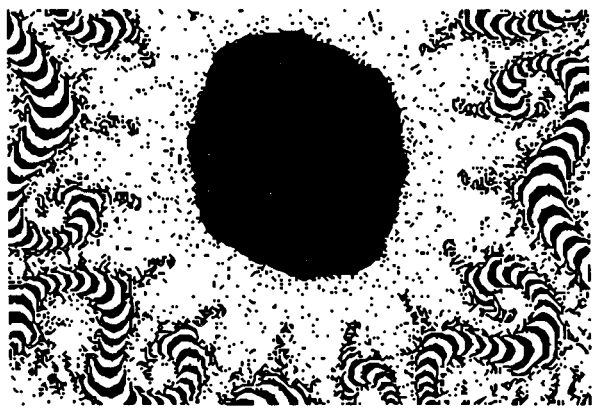


Figure 5.2-16 Julia set for c_2. Section from Fig 5.2-14.

Who would have realised, just a few years ago, that mathematicians might use computer graphics to investigate and test basic mathematical concepts? This trend has led to a new research area for mathematicians - *experimental mathematics*. Basically they now work with similar methods to those used long ago by physicists. But the typical measuring instrument is not a voltmeter, but a computer.

We have no wish to give the impression that the only interest in Julia sets is for measurement or research. Many pictures are also interesting for their aesthetic appeal, because of the unusual shapes that occur in them.

Throughout this book you will find pictures of Julia sets, and we recommend all readers to work on them with their own computers. Each new number pair

c_{real}, $c_{imaginary}$ produces new pictures. To be able to carry out the investigation in a fairly systematic way, we will first look at the sets that can be generated using c_1 and c_2.

The following pictures are generated using the same data as in Figure 5.2-10. They differ from it mostly in their size, and in the way that the ideas under investigation are represented.

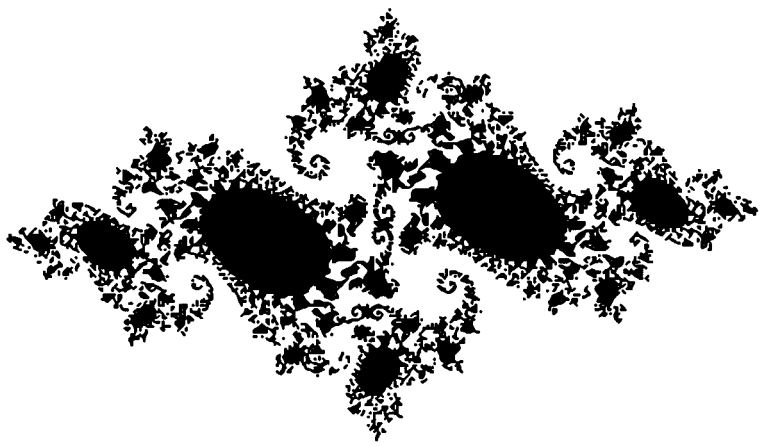


Figure 5.2-17 Julia sets with small iteration number.

As we have already shown earlier in this chapter, the form of this figure is completely connected. But that does not mean that every pixel, for which we compute the iteration series, really belongs to the figure. Because of the filigreed fractal structure of many Julia sets it is possible that in a given region all points which we investigate just lie near the figure. This is how the apparent gaps in Figure 5.2-17 arise. This holds even more when we raise the iteration number. Then there is often just 'dust' left. In Figure 5.2-18 we have therefore illustrated another limitation. It shows the points for which it is already clear after 12 iterations that they do not belong to the Julia set.

Even though these stripes, which we have referred to as 'contour lines', do not count towards the Julia set, they represent an optical aid without which the fine fractal structure could not often be detected.

If these stripes do not seem as thick and dominant as in the previous pictures, this is because not every second, but every third of them has been drawn. We interpret these regions as follows. The outermost black stripe contains all points, for which it is already apparent after 3 iterations steps that the iteration sequence converges to the attractor ∞. The next stripe inwards represents the same for 6 iterations, and so on.

It is also apparent that we cannot show all details of a fractal figure in a single picture. With a relatively coarse solution Figure 5.2-17 shows many different spiral shapes. On looking closer they disappear, to be replaced by finer structures inside the

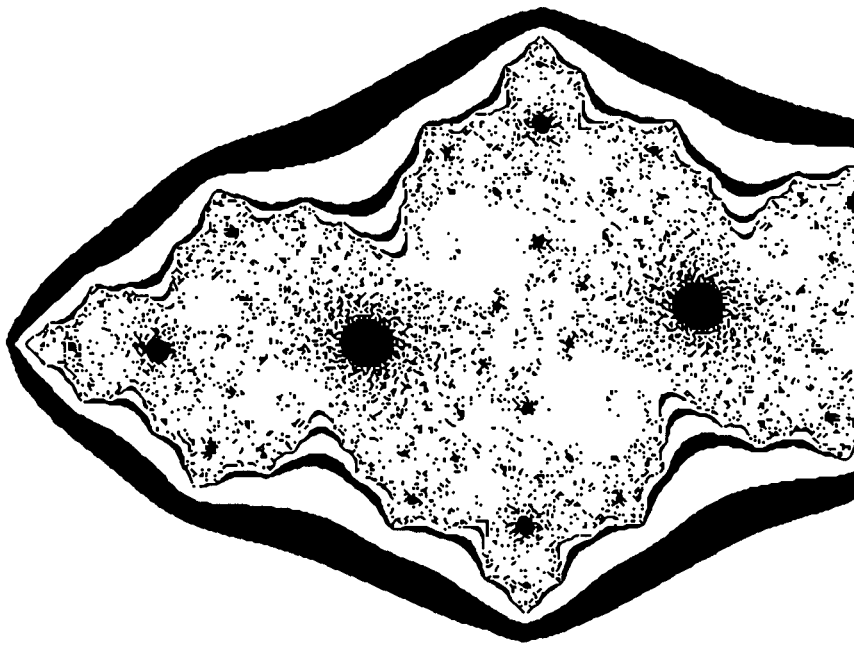


Figure 5.2-18 Julia set with higher iteration number also acts as a boundary.

large black regions.

If we investigate further details of this figure, you will perhaps notice the contiguous spirals in the middle. On the next two pages we show successive magnifications of this motif. Pay special attention to the central point. That is the origin of the coordinate system.

There we see an already known phenomenon (§2.2) in a new guise: a period-doubling scenario! The two contiguous spirals generate several smaller spirals, which you can see at the centre of Figure 5.2-19. After even greater magnification this can be seen in Figure 5.2-20, when it is possible to detect sixteen diminutive mini-spirals.

What else do you think is concealed within the black central dot?

Near the filigreed shape of the Julia set you can see still more 'contour lines' which, together with the white areas, belong to the basin of the attractor ∞.

Note also the numerous details of the Julia set in these pictures. In particular, as will become clearer, it looks as though all structures eventually belong to contiguous spirals.

At the beginning of this chapter, we stated that each c-value produces a different picture. Now we have seen that it can even be more than one picture.

Not all c-values lead to figures as rich in detail as the value $c \sim 0.745+0.113*i$,

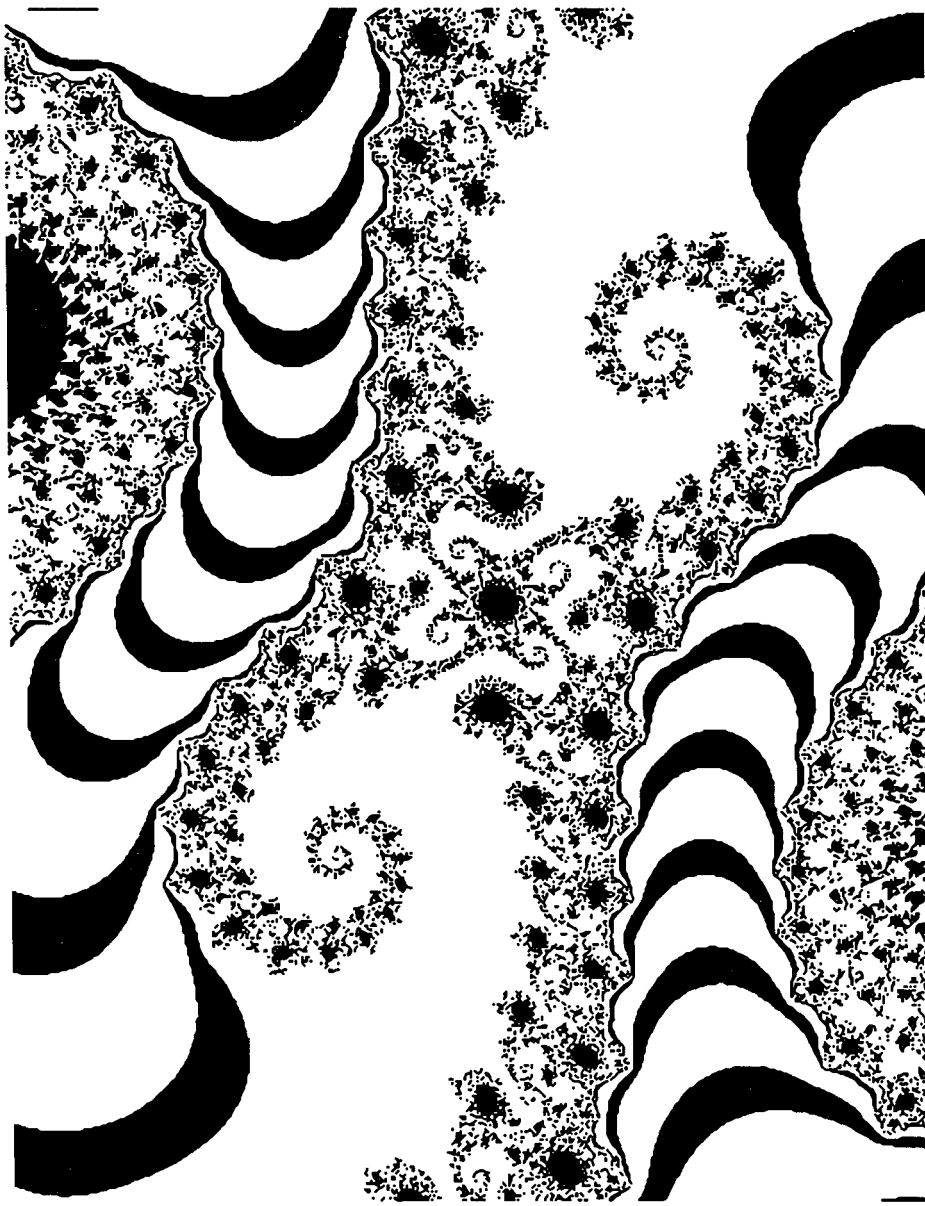


Figure 5.2-19 Section from the centre of Figure 5.2-17.

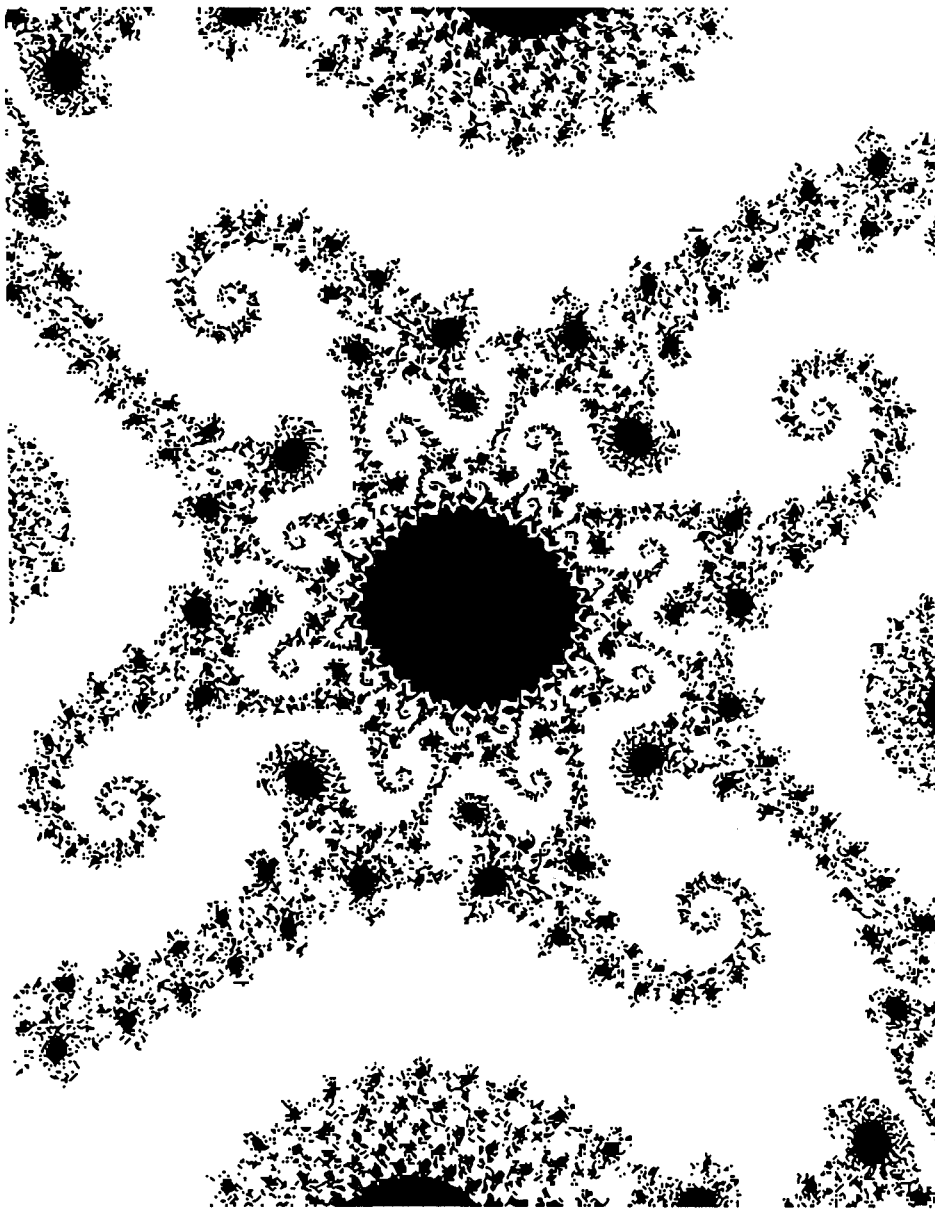


Figure 5.2–20 Section from the centre of Figure 5.2–19.

which underlies Figures 5.2-9 to 5.2-20. If you try out other values yourself, it can in some circumstances be rather boring, sitting in front of the computer, watching a picture form. For this reason we will suggest another way to draw the pictures, which is suitable for a general overview. To avoid confusion, we refer to the previous method as `Mapping`, while the new one is called `backwardsIteration`.

We begin with the following observation.

Each point that lies outside the Julia set follows, under the iteration

$$z_{n+1} = z_n^2 - c,$$

an ever larger path. It wanders 'towards infinity'. But we can reverse the direction. Through the backwards iteration

$$z_n = \sqrt{(z_{n+1}+c)}$$

the opposite happens: we get closer and closer to the boundary. Starting from a very large value such as $z = 10^6+10^6\mathrm{i}$ we reach the boundary after about 20 to 30 steps. Because each root has two values in the field of complex numbers, we get about 2^{20} to 2^{30} points ($\sim 10^6$ to $\sim 10^9$), which we can draw. This high number of possible points also makes it possible to stop the calculation at will.

In a Pascal program we implement this idea with a procedure `backwards`, which calls itself twice. This style of recursive programming leads to rather elegant programs.

The procedure `backwards` requires three parameters: the real and imaginary components x and y of a point, as well as a limit on the number of recursive steps. Inside the procedure the roots of the complex number are taken and the result is stored in two local variables `xLocal` and `yLocal`. Once the limit on the depth of recursion is reached, the two points corresponding to the roots are drawn. Otherwise the calculation continues. The constant c is added to the roots, and a new incarnation of the procedure `backwards` is called.

Extracting roots is very easy for complex numbers. To do so we go over to the polar coordinate representation of a number. As you know from §4.2, r and φ can be computed from x and y. Recalling the rule for multiplication, we see that the square root of a complex number is obtained by halving the polar angle φ and taking the (usual) square root of the radius r.

Program Fragment 5.2-3

```
PROCEDURE backwards (x, y : real; depth : integer);
   VAR
      xLocal, yLocal : real;
BEGIN
   compRoot(x, y, xLocal, yLocal);
   IF depth = maximalIteration THEN
   BEGIN
      SetUniversalPoint (xLocal, yLocal);
      SetUniversalPoint (-xLocal, -yLocal);
   END
```

```
    ELSE IF NOT button THEN (*button: break calculation *)
    BEGIN
       backwards (xLocal+cReal, yLocal+cImaginary, depth+1);
       backwards (-xLocal+cReal, -yLocal+cImaginary,
                  depth+1);
    END;
 END  (* backwards *)
```

Program Fragment 5.2-4

```
 PROCEDURE compRoot (x, y: real; VAR a, b : real);
    CONST
       halfPi = 1.570796327;
    VAR
       phi, r : real;
 BEGIN
    r := sqrt(sqrt(x*x+y*y));
    IF ABS(x) < 1.0E-9 THEN
       BEGIN
          IF y > 0.0 THEN phi := halfPi
             ELSE phi := halfPi + pi;
       END
    ELSE
       BEGIN
          IF x > 0.0 THEN phi := arctan (y/x)
             ELSE phi := arctan (y/x) + pi;
       END;
    IF phi < 0.0 THEN phi := phi + 2.0*pi;
    phi := phi*0.5;
    a := r*cos(phi); b := r*sin(phi);
 END;  (* compRoot *)
```

If you want to experiment with this version of the program, you must set the values of the real part `cReal` and the imaginary part `cImaginary` of the complex parameter *c*. Take a maximal iteration depth of, e.g.,

```
maximalIteration := 30;
```

If you make this number too large you will get a *stack overflow error*. For each new recursion step the computer sets aside yet another storage location.

In preparation, call the procedure with

```
backwards (1000,1000,1);
```

You can quit the procedure by pressing a key or clicking the mouse, whichever is set up on your computer (see chapter 11). You must hold the key down long enough so that

Figure 5.2-21 Backwards iteration, 20 seconds' computing time.

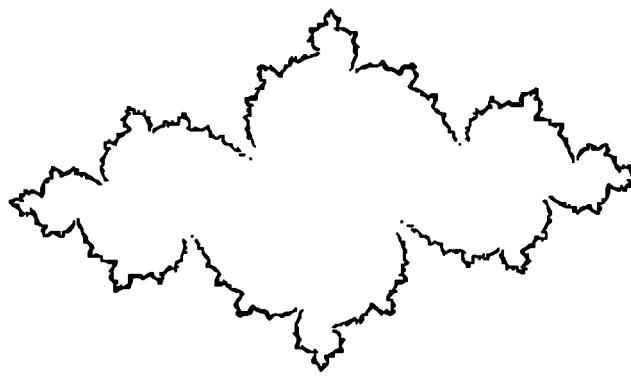


Figure 5.2-22 Backwards iteration, 4 hours' computing time.

you exit from the entire set of incarnations of the recursive procedure.

The next two pictures show examples of this method. In Figure 5.2-21 you see that already after a few seconds the general from of the Julia set is visible. The applicability of this fast method is limited by its duration. The points that lie far in from the edge are scarcely reached. Even after a few hours of computing time (Figure 5.2-22) the interesting spiral structures, which we really should expect, cannot be seen.

Computer Graphics Experiments and Exercises for §5.2

Exercise 5.2-1

Write a program to compute and draw Julia sets. It should have the facility for cutting out sections and magnifying them.

Preferably the computation should follow the 'contour line' method.

Do not forget to document your programs, so that you can pursue other interesting questions later.

Use the program to investigate the first magnification of Figure 5.2-6, to decide the question: is the Julia set connected or does it fall to pieces?

Exercise 5.2-2

Investigate similar series as in Figures 5.2-1 to 5.2-8.

The c-values can be changed along the real or imaginary axis. Can you detect - despite the complexity - some system in these series?

To save time you can exploit the symmetry of these Julia sets. You need carry out the computation only for half the points. The other half of the picture can be drawn at the same time or with the aid of some other graphical program.

Exercise 5.2-3

Find particularly interesting (which can mean particularly wild) regions, which you can investigate by further magnification.

Note the corresponding c-values, and try out similar values.

How do the pictures corresponding to two conjugate c-values differ from each other?

(If $c = a+i*b$ then its complex conjugate is $a-i*b$.)

Exercise 5.2-4

Build the procedure for backwards iteration into your program.

Use it to investigate a larger number of parameters c.

- Is the boundary of the set smooth or ragged?
- Does the set appear to be connected or does it split into several pieces?

Exercise 5.2-5

What pictures arise if, instead of large values, you start with small ones? For example,

```
backwards (0.01, 0.01, 1).
```

Compare the pictures with those in Exercise 5.2-1.

Can you give an explanation?

Exercise 5.2-6

Write a program which produces a series of Julia sets, so that the c-values are either

- random

or

- change step by step in a predetermined fashion.

Exercise 5.2-7

Attach a super-8 camera or a video camera which can take single frames to your computer. Make a film in which you compute several hundred members of a sequence such as Figures 5.2-1 to 5.2-8, to show how the shape of the Julia set changes as the value of c is altered. The results make the world of Julia sets seem rather more orderly that one might expect from the individually computed pictures.

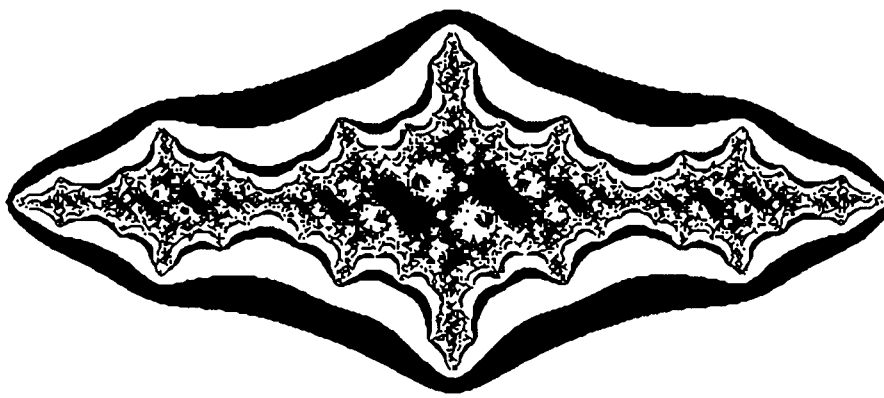


Figure 5.2-23 Yet another Julia set (just to whet your appetite).