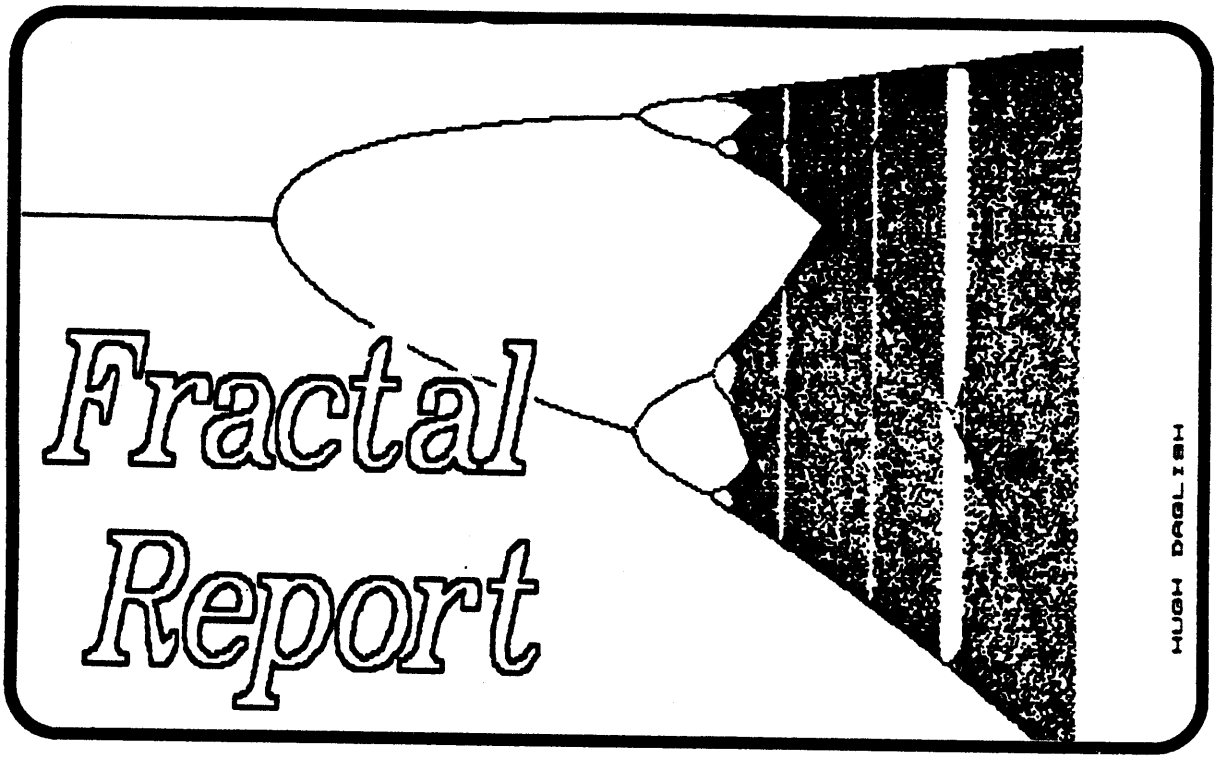


no 7

Single copy rate £2. Subscription rates six issues: – £10 (UK only) £12 Europe £13 elsewhere.

Cheques in British Pounds should be drawn on a UK bank and should be made payable to "Reeves Telecommunications Laboratories Ltd." Alternatively, dollar checks for $23 can be accepted if drawn on a U.S. bank and made payable to "J. de Rivaz".

Subscribers who are successful in getting one or more articles or letters published in a given series of six issues get the next volume of six issues free of subscription.

All new subscriptions are backdated to the start of the current volume.

Editorial

Welcome to the second volume, and many thanks to all those who renewed their subscriptions. Below I mention some American contemporaries. This will serve two purposes. It will provide readers who wish to expand their horizons with sources for further information, and there is the hope that the editors of these journals will publish similar reviews giving their readers the same opportunity. The subscription arrangements for *Fractal Report* in America (only) require a check for $23 made payable to "J. de Rivaz". This brings by airmail six issues backdated to the start of the current volume. $46 brings volume 1 as well, together with issues 0 & −1.

Some people said they like reviews of conferences etc., although they are few and far between. I would direct these people to an excellent and polished review newsletter called *The Dynamics Newsletter* published by Ralph Abraham at the Aerial Press, PO Box 1360, Santa Cruz, California 95061−1360, U.S.A. I know it's American, but the coverage is worldwide and if you get out and about to go to conferences, then you'll not mind raising the few dollars that a subscription requires. It an excellent but struggling publication, so if you like this sort of thing then it needs your support. Write to Mr Abraham for a sample issue and subscription details. He would also be interested in conference announcements or reviews of past conferences, for publication in *The Dynamics Newsletter.*

At this juncture I can't fail to mention the other American newsletter that is aimed at more experienced fractal enthusiasts, *Amygdala.* It spreads its net wide as to article type, but it does come up with some real gems at times. Many of you have bought copies of the QL machine code Mandelbrot and Julia programs. The core of these originated from an article in *Amygdala* describing an Amiga program. Mr Richard V. Robinson's fractal music cassette came from another *Amygdala* article. (See issue 6 editorial for information.) Although these gems will appear in *Fractal Report*, probably in more accessible form eventually, there will be a long time delay. Therefore if you want to be in the forefront a subscription to *Amygdala* would not come amiss. Address: Box 219, San Cristobal, New Mexico 87564, USA.

Nearest to *Fractal Report* in content is Dr Clifford Pickover's bi−annual newsletter *The Journal of Chaos and Graphics.* Dr Pickover isn't as strict as I am in aiming for practical information, but many of the contributions are similar to *Fractal Report* articles. It appears that some early issues were sent out free. Dr Pickover can be contacted at the IBM Watson Laboratory, Yorktown Heights, NY10598, U. S. A. He is also anxious for written contributions to this and other journals.

Although some of you wanted an arrangement to pay for American subscriptions in sterling, we haven't come up with a formula that is economic of money and time to make this possible. For those readers who might want to buy things by mail from America, it is not that hard to get a $US dollar account. Ask your UK bank for an introduction to an American bank where you can open a dollar account by mail. If you keep more than two or three hundred dollars in the account, then there are no banking charges on some accounts. It has not been illegal for British people to have overseas accounts for some time. Mrs Thatcher returned this freedom soon after she came to power. It was denied citizens during the war in most countries, but the UK with its history of non−Libertarian governments was one of the last to restore it.

Pictures from Complex Polyinomials using Newton's Algorithm

John C. Topham

Refering to the article W.E. Thomson wrote in Issue 1 of 'Fractal Report', I would like to extend his idea for drawing pictures from solutions of z^n = 1 to encompass fifth degree polynomials of the form,

$$c_5z^5 + c_4z^4 + c_3z^3 + c_2z^2 + c_1z + c_0 = 0 \qquad (1)$$

where $c_0,.....,c_5$ are real or complex constants and z = x + yi (x,y are real variables).

There are two methods you can use to draw such pictures.

With the first method you choose the coefficients, c_0 to c_5, and set up the two relevant equations for x_{n+1} and y_{n+1} by working through Newton's formula. An example is shown below. Then you use the final equations to find the roots of the polynomial.

In the second method you first set up your polynomial in the form that already shows the roots as follows:

$$(z + r_1)(z + r_2)(z + r_3)(z + r_4)(z + r_5) = 0 \qquad (2)$$

where $r_1....r_5$ can be real or complex numbers. Multiplying this out you eventually get the equation in the form of (1) but with coefficients in term of the roots in (2). With this method you have the opportunity to experiment to see if new phenomena occur if the roots are widely or closely separated. If some of the roots you choose are complex then you will more than likely get a polynomial with complex coefficient, whereas in the case of the first method you can choose only to have real coefficients if you wish.

In each case I used a program that plots out pixels in much the same way Julia Sets are drawn, i.e. each pixel's coordinates are used as the intial values for the system. When the system is iterated a string of numbers are generated. However, instead of checking for any thresholds, the program checks to see which of the polynomial's roots this string of numbers is settling down to. When this root is found the program selects the colour chosen for that root and allocates it to that pixel.

To find the two equations to iterate using the first method you proceed as follows taking the same equation Mr Thomson used as an example,

$$z^3 = 1 \quad \text{or} \quad z^3 - 1 = 0$$

where z = x + yi (x,y are real numbers, here, of course, c_5, c_4, c_2 and c_1 are zero, and c_3 and c_0 are 1), you use Newton's Method of root finding in much the same way Mr Thomson described i.e using the function $f(x_n) = (z_n)^3 - 1$ and its derivative $f'(z_n) = 3(z_n)^2$ and accordingly you get

$$z_{n+1} = (2(z_n)^3 + 1) \; / \; 3(z_n)^2 \qquad (3)$$

Substituting $(x_n + y_n i)$ for z_n and $(x_{n+1} + y_{n+1} i)$ for z_{n+1} in (3) we get

$$x_{n+1} + y_{n+1} i = (2(x_n + y_n i)^3 + 1) \; / \; 3(x_n + y_n i)^2$$

Multiplying these terms out

$$x_{n+1} + y_{n+1} i = 2((x_n)^3 + 3(x_n)^2 y_n i - 3x_n (y_n)^2 - (y_n)^3 i) + 1)$$

$$/ \; 3((x_n)^2 + 2x_n y_n i - (y_n)^2)$$

Using the quotient rule for complex numbers you then get

$$x_{n+1} + y_{n+1} i = (ac + bd) + (bc - ad) i \; / \; (c^2 + d^2)$$

where $a = 2(x_n)^3 - 6x_n (y_n)^2 + 1$ $\quad$ $c = 3(x_n)^2 - 3(y_n)^2$

$b = 6(x_n)^2 y_n - 2(y_n)^3$ $\quad$ $d = 6x_n y_n$

To obtain the necessary equation you decompose the above equation as follows

$$x_{n+1} = (ac + bd) \; / \; (c^2 + d^2)$$

$$y_{n+1} = (bc - ad) \; / \; (c^2 + d^2)$$

Using this system to generate the roots i.e. writing an iterating program to home in on the roots after choosing intial values at random, , I found in this case they were $z_1 = 1$, $z_2 = -0.5 + 0.866i$, $z_3 = -0.5 - 0.866i$.

These are used in the program shown in Listing 1. Results of this program are shown in figure 1. They are much the same as Mr Thomson's.

If you wanted to draw pictures of higher degree polynomials with lower degree terms in them, using the coefficient choosing method, you simply use the same algorithm to work out the equations and apply them in the program.

To find the roots with this method you proceed as follows. The system is iterated and a root is arrived at. This root is stored and allocated a colour. The system is iterated again. If the root is arrived at again it carries on as normal allocating the first colour as before, however, if a new root is arrived at then this is stored in a second location and a second colour is allocated. Having confidence in the method we assume that the final number of roots found will be the same as the order of the polynomial, thus a degree five polynomial will have five regions on the picture with different colours.

Some of the results are shown in figures 2 and 3. Listings of the programs for both methods for drawing these are available on request if anyone is interested.

The programs used behave quite well. You can see that well ordered pictures result.

With the large size of the equations the iterations tend to be rather slow on the QL. As you see by the results I don't find it too inconvenient. However, if you have the use of a faster computer you could carry out a lot more experiments in a more reasonable span of time.

In conclusion, consider the equation:

where 'e' is the expontial function, z is a complex number. Here you have a type of polynomial, which should have roots, i.e. when $z = -e^{z}$ and $z = e^{(-z)}$.

If you proceed as before with the Newton method and using Euler's equation:

$$e^{z} = e^{(x+yi)} = e^{x}(\cos(y) + \sin(y)i)$$

you can also derive two equations that you can use to iterate. Figure 4 shows results of this system. It shows that there are many if not an infinite number of roots. This does not seem so surprising since the nature of sine and cosine functions is cyclic. I am currently working on the general equation up to fifth order:

$$(Z + C_1e^{r1z})(Z + C_2e^{r2z})(Z + C_3e^{r3z})(Z + C_4e^{r4z})(Z + C_5e^{r5z}) = 0$$

where $C_1 \ldots C_5$ and $r_1 \ldots r_5$ are complex numbers. I have yet to work out the function's derivative! We will see if anything surprising will come of the pictures it will provide, if any!

Announcements

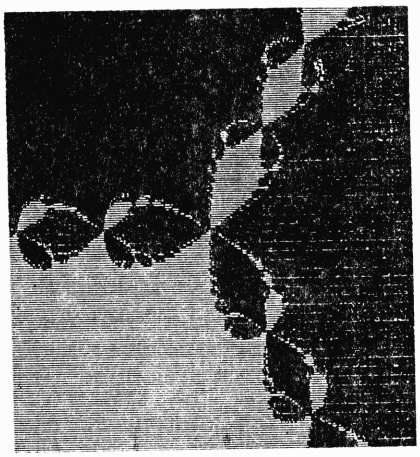
We have had a number of announcements from readers. Mr Bob Harris, of Holly Oakes, Mill Lane, Lymington, Hants, SO41 8LN has some Commodore 64 Mandelbrot and associated programs. If you send him a blank 5.25" disk and return postage and packaging, then he will fill it for you. If you send him two disks, then he will send some 3D wire frame programs and files as well. His offer is in the interests of encouraging more Public Domain C64 programs in this area.

Mr P.J. Tewkesbury, of 26, Gathorp Road, Northern Moor, Manchester M23 0AS has an Amstrad CPC6128 and offers a Hisoft Pascal Mandelbrot generator. A blank 3" disk and £1 gets you this program. A full machine code version is in preparation.

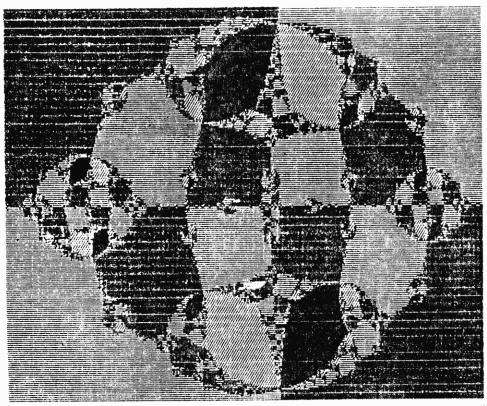
Mr N.W. Fenwick of 164a, High Street, Swanage, Dorset BH19 2PE will send a collection of Amiga A500 Public Domain software for £2.50 including disk, postage and packing. He has also experimented with single lens reflex VDU photography, and recommends using a tripod with cable or timer release in pitch darkness. A shutter speed less than 1/50 sec, say 1/15 sec on 100 ASA film produces a good print. He sent us a print that shows off the Amiga's excellent capabilities on the Mandelbrot Set.

Ms Heather Whitby, of The Association of Maths Teachers, 7, Shaftesbury Street, Derby DE3 8YB, has mentioned her organisation's sets of "Logo Trees" (£3) and "Mandelbrot" postcards (£1.50) delivered. The former includes a program. Although only in monochrome, you get good value for each set, and they can make interesting and attention getting means of communication. "Logo Trees" also has a frieze of printouts and some Logo listings. The Association has an open challenge to produce realistic willows, cypresses, poplars, and oaks.

FIGURE 1



A.



B.

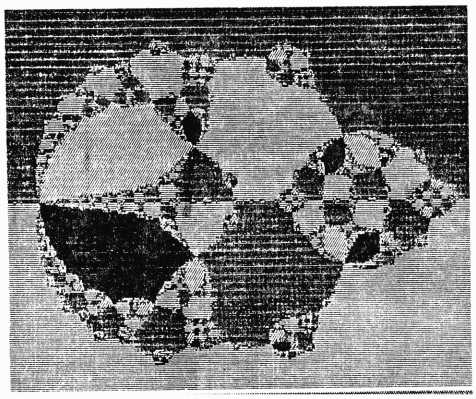
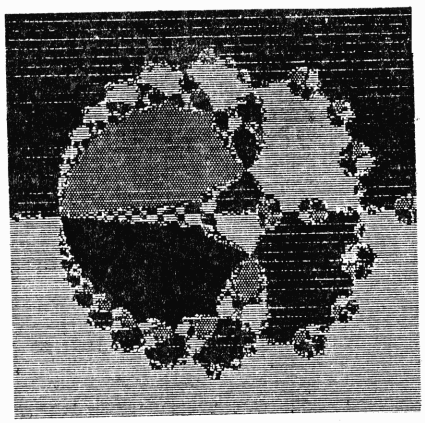
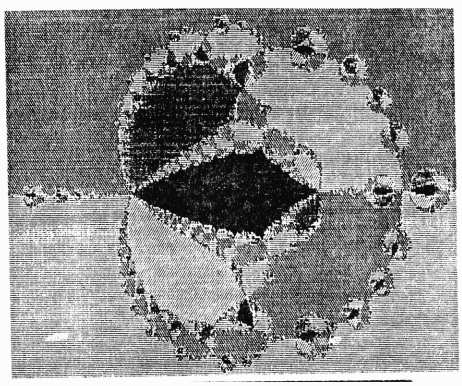


FIGURE 2

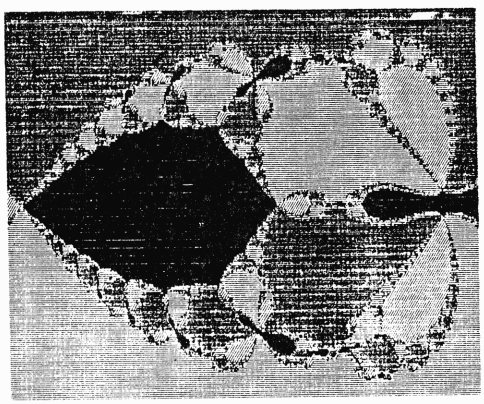
C.



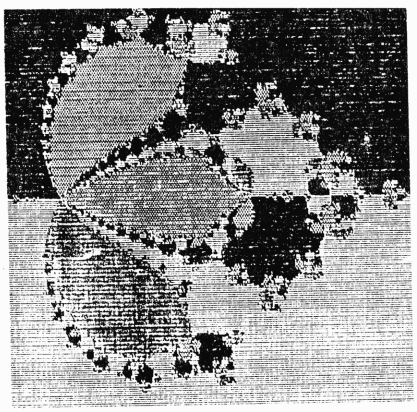
D.



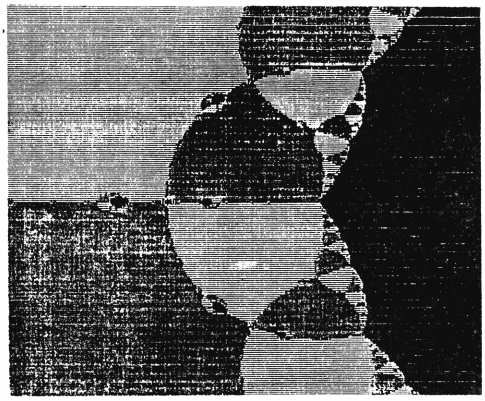
E



F



A.



B.

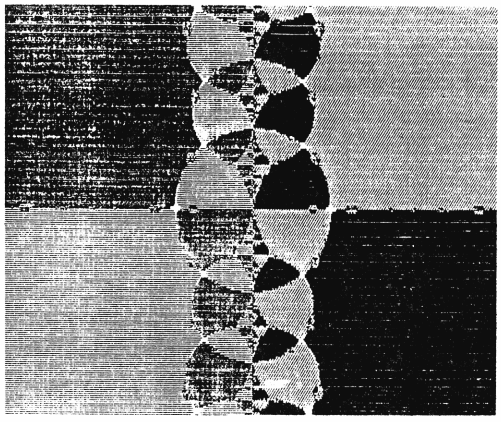


FIGURE 3

C.

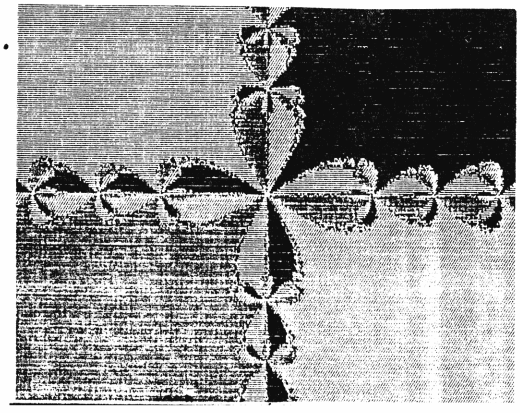
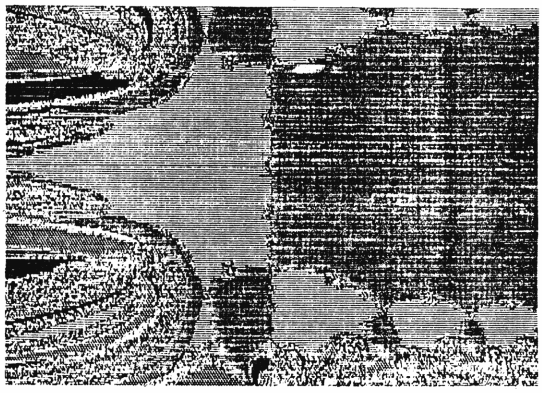
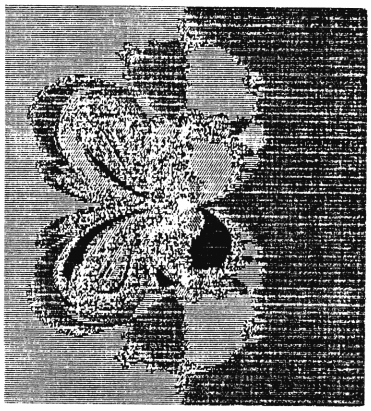


FIGURE 4

A.



B



Picture Dimensions:

Figure 1. Polynomial: $z^3 - 1 = 0$. xmin = -1.5, xmax = 1.5
ymin = -1.5, ymax = 1.5

Figure 2. A & B. Polynomial: $-z^4 - 2z^3 - z^2 - 2 = 0$.
A. xmin = -0.554, xmax = 0.064
ymin = -0.225, ymax = 0.225
B. xmin = 0.378, xmax = 0.666
ymin = -0.123, ymax = 0.123

C. Polynomial: $-z^4 - 2z^3 - z^2 + z - 2 = 0$
xmin = -0.083, xmax = 0.059
ymin = -0.07, ymax = 0.07

D. Polynomial: $-z^5 - z^4 - 2z^3 + z - 2 = 0$
xmin = 0.222, xmax = 0.7
ymin = -0.185, ymax = 0.185

E. Polynomial: $-z^4 - 2z^3 + 2z^2 + 2z + 2 = 0$
xmin = -0.365, xmax = -0.074
ymin = -0.107, ymax = 0.107

F. Polynomial: $-z^5 - z^4 - 2z^3 - z^2 + z - 2 = 0$
xmin = 0.2, xmax = 0.635
ymin = -0.225, ymax = 0.225

Figure 3. A. 3rd Order polynomial from roots:
(-2 + 0.5i, -2 - 0.5i, 1)
xmin = -1.5, xmax = 1.5
ymin = -1.5, ymax = 1.5

B. 4th Order polynomial from roots:
(-1 - 0.2i, -1 + 0.2i,
1 - 0.2i, 1 + 0.2i)
xmin = -1.5, xmax = 1.5
ymin = -1.5, ymax = 1.5

C. 4th Order polynomial from roots:
(-0.7071 - 0.7071i, -0.7071 + 0.7071i,
0.7071 - 0.7071i, 0.7071 + 0.7071i)
or from polynomial: $z^4 + 1 = 0$
xmin = -1.5, xmax = 1.5
ymin = -1.5, ymax = 1.5

Figure 4. A. & B. Polynomial: $(z + \exp(z))(z + \exp(-z)) = 0$
A. xmin = -3, xmax = 3
ymin = -3, ymax = 3
B. xmin = -0.105, xmax = 0.105
ymin = -0.138, ymax = 0.138

```
100  REMark    *****                LISTING 1                          *****
110  REMark    *****               ===========                         *****
120 :
130  REMark    *****      Iteration of Newton's Method                 *****
140  REMark    *****      For z^3 = 1  or  z^3 - 1 = 0                 *****
150  REMark    *****   Written by John C. Topham for the QL            *****
160 :
170 PIXEL_DIMENSIONS
180 PICTURE_DIMENSIONS
190 ITERATION_LOOP
200 STOP
210 :
220 DEFine PROCedure PIXEL_DIMENSIONS
230 across=300:down=200
240 END DEFine
250 :
260 DEFine PROCedure PICTURE_DIMENSIONS
270 xmin=-1.5  :xmax=1.5
280 ymin=-1.5  :ymax=1.5
290 END DEFine
300 :
310 DEFine PROCedure ITERATION_LOOP
320 :
330 CLS
340 Dx=(xmax-xmin)/(across-1):Dy=(ymax-ymin)/(down-1)
350 xdis=INT((444-(across))/2):ydis=200-INT((200-down)/2)
360 :
370 FOR yp=1 TO down
380   AT£0,0,0:PRINT£0,'Line=';yp;"   "
390   FOR xp=1 TO across
400     xn=xmin+(xp*Dx):yn=ymin+(yp*Dy)
410     flag=0
420     :
430     REPeat loopI
440      a=2*xn*xn*xn-6*xn*yn*yn+1
450      b=6*xn*xn*yn-2*yn*yn*yn
460      c=3*xn*xn-3*yn*yn
470      d=6*xn*yn
480      xm=(a*c+b*d)/(c*c+d*d)
490      ym=(b*c-a*d)/(c*c+d*d)
500      FIND_ROOT
510      IF flag=1:flag=0:EXIT loopI:END IF
520      xn=xm:yn=ym
530     END REPeat loopI
540   END FOR xp
550 END FOR yp
560 END DEFine
570 :
580 DEFine PROCedure PLOT_POINT
590 BLOCK 1,1,xdis+xp,ydis-yp,ink_p
600 END DEFine
610 :
620 DEFine PROCedure FIND_ROOT
630 IF xn+yn==(SQRT(3)-1)/2:  ink_p=2:PLOT_POINT:flag=1:END IF
640 IF xn+yn==(-SQRT(3)-1)/2: ink_p=4:PLOT_POINT:flag=1:END IF
650 IF xn+yn==1:              ink_p=7:PLOT_POINT:flag=1:END IF
660 END DEFine
```


2023 note:UK printers were often set to print £ instead of # , only a problem when printing program listings...

A Three Dimensional Julia Plot with Z – Trajectories.

by John de Rivaz

First may I apologise to any author who feels that his article has been displaced by this one. However I feel that this illustrates how a few simple lines of BASIC (in this case GW BASIC – I have now got a PC, and my Spectrum has given up the ghost in disgust!) can produce an interesting display, and what's more it's fun to watch it build up.

The 3D Julia pattern follows ideas given by Lewis Siegel in *Fractal Report 5* and the Z – Trajectories follow ideas given by Ed Hersom in *Fractal Report 1,* and others in unpublished works. The Julia plot is built from the program by Liam Proven in *Fractal Report 4.*

The program produced below has the 3D Julia centred (when fully displayed) at the lower portion of the screen. On the top right had side there is a "sun" consisting of all the Z – trajectories superimposed. On the top left hand side all the z – trajectories for a single line are superimposed, and erased at the end of the line. In addition, pressing the "B" key at any time causes the z trajectory to be blanked at the end of each point's iteration, and pressing "N" returns it to the next line condition.

```
100 SCREEN 9:A%=350:B%=600:COLOURS=16:KEY OFF
110 XMIN=-1.75:XMAX=2:YMIN=-1:YMAX=2.25
120 P=-.39054:Q=.58769:CMAX%=100
130 DX=(XMAX-XMIN)/(A%-1):DY=(YMAX-YMIN)/(B%-1):C1%=1:M%=200:CLS
140 FOR NY%= 0    TO A%-1
150 FOR NX%= B%-1 TO 0    STEP -1
160   K%=0:C1%=C1%+1:IF C1%=COLOURS THEN C1%=1
170   X=XMIN+NX%*DX:Y=YMIN+NY%*DY
180   K%=K%+1:XN=X*X-Y*Y+P:Y=2*X*Y+Q:X=XN:'label dloop
190   IF (X*X+Y*Y)>M% THEN C%=K% MOD COLOURS :GOTO 230 :'dplot
200   IF K%=CMAX% THEN C%=15:GOTO 230:'dplot
210   PSET (550+X*5,70+Y*5),C1% :PSET(100+X*5,70+Y*5),C1%
220   GOTO 180 :'dloop
230   XX%=NX%+.9659*NY%:YY%=450-100*K%^.2-.2588*NY%: 'dplot
235   IF POINT(XX%,YY%)=0 THEN PSET(XX%,YY%),C%
240   IF INKEY$="b" THEN BLANKING =1:BEEP
250   IF INKEY$="n" THEN BLANKING =0:BEEP
260   IF BLANKING AND C%<>1 OR NX%=B%-1 THEN LOCATE 1,1:FOR J=1 TO 11:PRINT "
                ":NEXT J
270 NEXT NX%
280 NEXT NY%
```

This program is given not just as something to type in and run, but something to play with. You can change the Julia coordinates in line 120, for example. Can you find a quicker way to blank out the left hand Z – Trajectories than printing strings of blanks (line 260)?

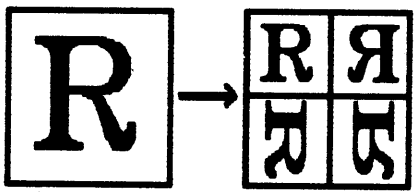
SPACE FILLING ITERATIVE PATTERNS

Hugh Daglish

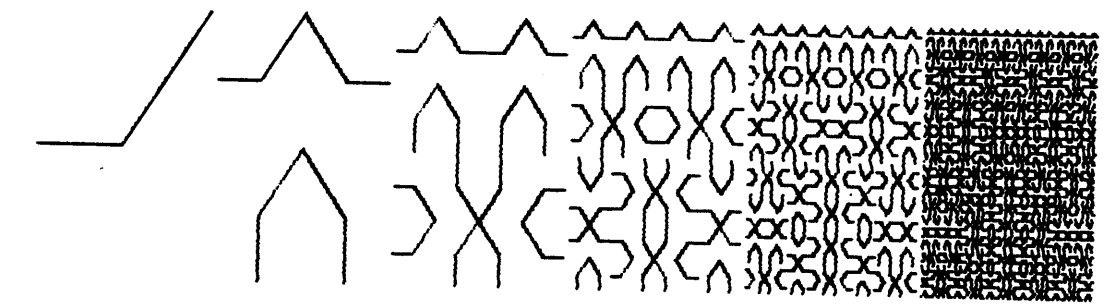
A characteristic of virtually all the programs discussed in "Fractal Report" is the generation of increasingly complex images by the progressive manipulation of comparatively simple data. Apart from their mathematical interest, some of the patterns produced are aesthetically pleasing and can be useful in desk-top publishing. For example, the Deterministic Algorithm described by Michael Barnsley* can be used as the basis for creating space-filling patterns, of any desired density and complexity. Such patterns can be useful, for example, as backgrounds to titles, such as that used for "Fractal Report".

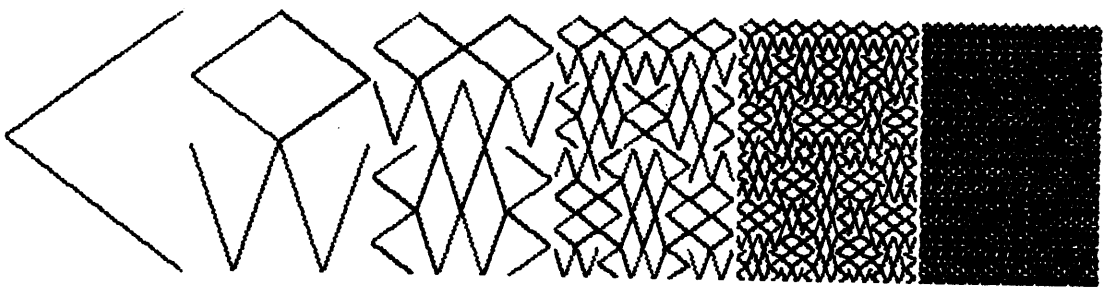
Acceptable results can be obtained using a PC with a graphics adapter, and a dot-matrix printer. The basic procedure is as follows. Two square arrays are defined (say 120 x 120 elements each), where each element represents a pixel on the screen. A simple pattern (such as a straight line) is loaded into the first array, to provide the starting data. The second array is divided into quadrants and a suitable algorithm is then used to transfer the original pattern into each quadrant, scaling the pattern down, and inverting it, rotating it or otherwise transforming it. Although an identical transformation could be used for each quadrant, the result would be somewhat trivial, and it is obviously better to choose four similar but different transformations.

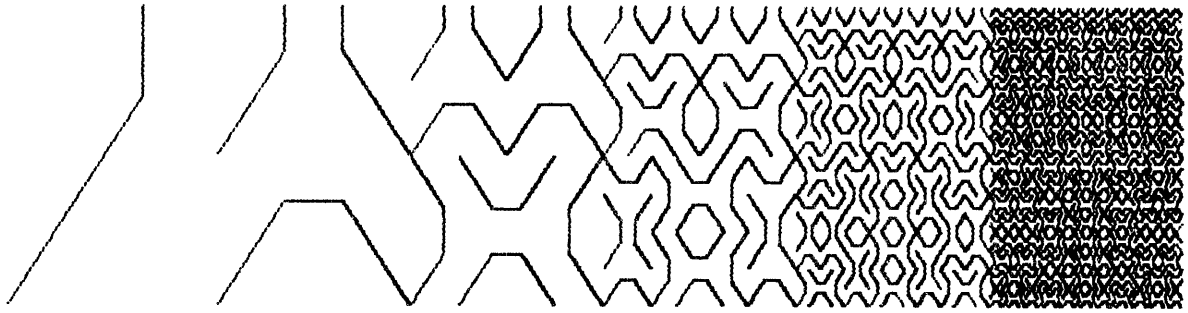
To clarify this, the diagram shows how the letter "R" might be transformed by rotation and reflection.

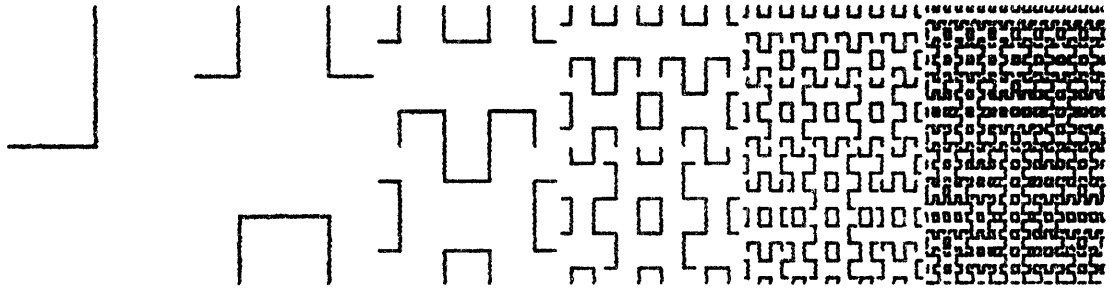


Using this principle, the next four illustrations show how a single initial line can be transformed into a useful complex pattern, using this particular transformation.









These patterns were generated by a Basic program: because graphic procedures are often machine dependent, I have only shown the key elements of the program here. A subroutine for stopping the program is useful as it enables patterns to be aborted if they look too uninteresting!

```
CLS                                     "Clear Screen"
DIM S$(100, 100): DIM T$(100, 100) "Define arrays on the screen"
LET offset% = 10, drop% = 70            "Position output"
LET lim% = 100                          "Size of pattern"
FOR i% = 1 TO lim%                      "Create starting pattern"
LET T$(i%,i%) = "1"                     "Plot the starting pattern"
PLOT (i% + offset%, i% + drop%)
NEXT i%
IF INKEY$ <> "" THEN GOSUB 8000     "Subroutine to stop program"

REM DATA FOR THE FOUR TRANSFORMS
LET a1 = 0.0, b1 = 0.5, c1 = 0.5, d1 = 0.0, e1 = 0.0, f1 = 0.0
LET a2 = 0.5, b2 = 0.0, c2 = 0.0, d2 = 0.5, e2 = 0.0, f2 = 0.5
LET a3 = -0.5, b3 = 0.0, c3 = 0.0, d3 = 0.5, e3 = 1.0, f3 = 0.5
LET a4 = 0.0, b4 = -0.5, c4 = 0.5, d4 = 0.0, e4 = 1.0, f4 = 0.0

REM MAIN PROGRAM
LET offset% = 40
FOR iter% = 1 TO 5
FOR i% = 1 TO lim%
FOR j% = 1 TO lim%
IF T$(i%, j%) = "1" THEN
S$(a1 * i% + b1 * j% + e1 * lim%, c1 * i% + d1 * j% + f1 * lim%) = "1"
S$(a2 * i% + b2 * j% + e2 * lim%, c2 * i% + d2 * j% + f2 * lim%) = "1"
S$(a3 * i% + b3 * j% + e3 * lim%, c3 * i% + d3 * j% + f3 * lim%) = "1"
S$(a4 * i% + b4 * j% + e4 * lim%, c4 * i% + d4 * j% + f4 * lim%) = "1"
END IF
IF INKEY$ <> "" THEN GOSUB 8000
NEXT j%
NEXT i%

REM PLOT SEQUENCE
FOR i% = 1 TO lim%
FOR j% = 1 TO lim%
```

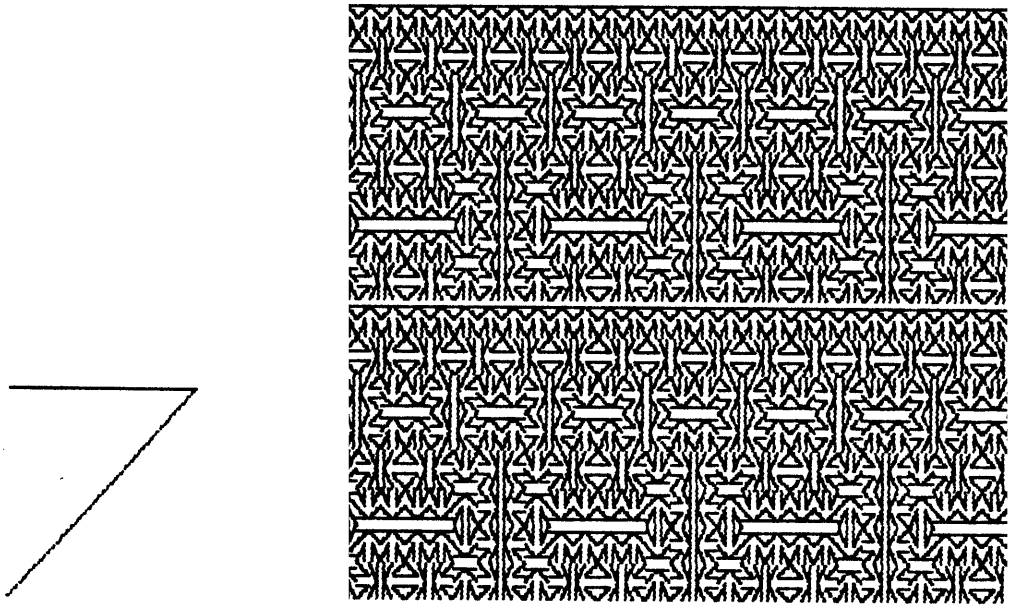
```
LET T$(i%, j%) = S$(i%, j%)      "Copy new array into the old array"
LET S$(i%, j%) = ""              "Clear the array for next loop"
IF T$(i%, j%) = "1" THEN
PLOT (i% + offset% + iter% * lim%, j% + drop%)
END IF
IF INKEY$ <> "" THEN GOSUB 8000
NEXT j%
NEXT i%
NEXT iter%
STOP

8000 REM  "Subroutine to stop program safely: not shown here"

END
```

Changing the starting data or the transformation coefficients enables this program to be used to generate a wide variety of patterns. The range can be further extended by changing one of the linear transformations to a non-linear form, and so on. (It might then be wise to add extra "IF" statements, to prevent accidental assignments beyond the range of the two arrays).

Larger arrays than about 110 x 110 are not practicable with my system, so to produce larger patterns, I use a version of the program which plots only the fourth iteration, but plots it up to twelve times, in two or three rows. The resulting pattern can then be extracted for use in a commercial "Paintbrush" program.



---

* "Fractals Everywhere", M Barnsley, Academic Press 1988.

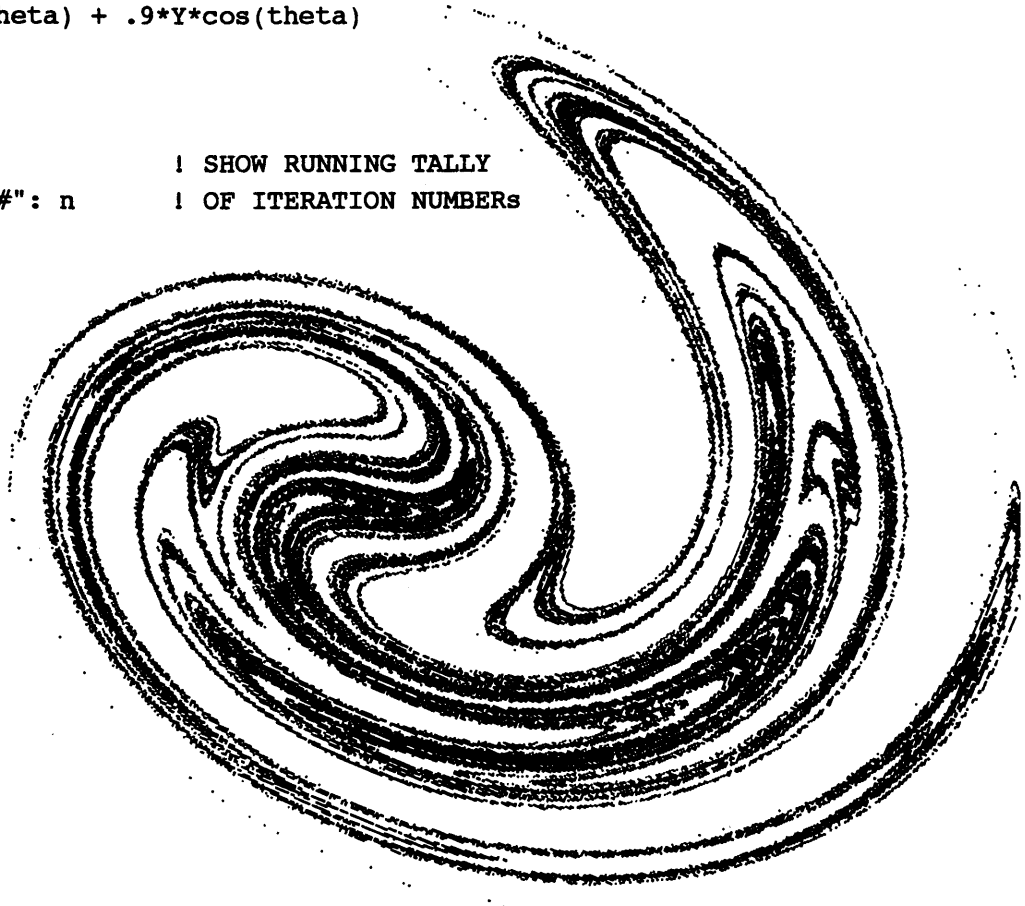
```
      REM ************************************************************************
      REM * True BASIC PROGRAM NAME:  Ikeda Map                                  *
      REM * WRITTEN BY:  John Corbit                DATE: August 16, 1989       *
      REM *                                                                      *
      REM * The Ikeda map is the chaotic attractor generated by iterating       *
      REM * the expression:                                                      *
      REM * Z(n+1) = A + B*Z(n) exp [i*k - ((i*p)/(1 + (ABS(Z(n)))^2))]         *
      REM * where Z = X + iY is a complex variable and orbit points for the     *
      REM * system are plotted in the complex plane. Solving for X and Y,       *
      REM * X(n+1) = A + B*X(n)*cos(theta) - B*Y(n)*sin(theta)                  *
      REM * Y(n+1) = B*X(n)*sin(theta) + B*Y(n)*cos(theta)                      *
      REM * where theta = k - (p/(1 + (X(n)^2 + Y(n)^2)))                       *
      REM * and where A = 0.85, B = 0.9, k = 0.4, and p = 7.7                   *
      REM * Note that there is a critical value for p. When p < p(c) = 7.2688 *
      REM * the attractor is restricted to a smaller central region. For       *
      REM * more information see C. Grebogi, E. Ott, F. Varosi, & J.A. Yorke, *
      REM * Laboratory of Plasma Research, University of Maryland, College     *
      REM * Park, MD 20742, U.S.A.                                             *
      REM ************************************************************************


OPEN #1: SCREEN .125,.875,0,1      ! OPENS "SQUARED UP" WINDOW
SET WINDOW -1.2,-1.7,1.3           ! SCALE THE WINDOW
           X  X  Y  Y

LET X = 0                          ! INITIAL VALUES
LET Y = 0
LET p = 7.7

FOR n = 1 TO 100000                ! MAIN PROGRAM
    LET theta = .4 - (p/(1+(X^2 + Y^2)))
    LET X1 = .85 + .9*X*cos(theta) - .9*Y*sin(theta)
    LET Y1 = .9*X*sin(theta) + .9*Y*cos(theta)
    PLOT X1,Y1
    LET X = X1
    LET Y = Y1
    SET CURSOR 2,60                ! SHOW RUNNING TALLY
    PRINT USING "###,###": n       ! OF ITERATION NUMBERs
NEXT n

END
```



COLOURED IFS TILINGS

By Uwe Quasthoff

1. Introduction

Assume we have an affine self-similar fractal of fractal dimension 2 (like Mandelbrot's fudge flake [2], for instance). Then the whole object can be partitioned into (non-overlapping) smaller copies of this object. Colouring these smaller copies gives very attractive pictures.
Our projects divides into two problems. First we have to assign a colour to each point of the fractal and second there is a general problem creating "massive" fractals (i.e. of dimension near 2) without missing pixels. Using an iterated function system (IFS) and the random algorithm takes a very long time to fill a massive fractal up to the last pixel. In section 3 we introduce a tree parsing algorithm which fills any IFS fractal in a non-random way.

2. Addresses and colours

As in [1, chapter 4] we can introduce the address of a point. Assume we have three transformations w_0, w_1 and w_2. Now take a point z and apply some of these transformations. For instance, let

$$y = w_2(w_1(w_0(w_0(w_2(z))))).$$

Then y has the address 21002 with repect to z. In an IFS, the transformations are contractions. Hence, if we consider only points with addresses of a certain minimum length, then the pixel which corresponds to this address will no more depend on the point z.
To assign a colour to a point we use the last part of its address. If we have three transformations it turns out to be good to take the last three digits of the address and read this as a number in base 3. Hence, we get a number c satisfying $0 \le c \le 26$. If we have a micro displaying 16 colours with numbers 0 and 1 for white and black, respectively, we calculate a colour by 2+(c MOD 14). In general, the modulus should be relatively prime to the number of different addresses, so in some cases it might be better to take 13 instead of 14.
Next we describe how we can actually calculate the colour using the address.

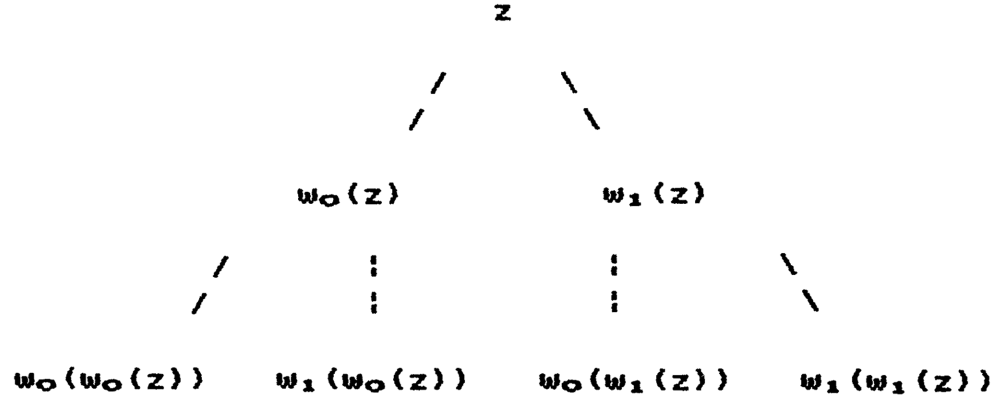
Algorithm 1. Random IFS algorithm with colours. Assume we have N transformations W(0), W(1),..., W(N-1) and use the last K digits of the address (The size of small parts depends on K, a good relation is $N^K<40$). z denotes a point in the plane to be transformed by one of the transformations W(i). The algorithm never terminates.

STEP 1. [Initialize]. Choose a suitable starting point z. (In many cases, z=0 will do.) Let c=1.

STEP 2. [Choose transformation]. Choose a random integer i satisfying $0 \le i \le N-1$. Apply transformation W(i) to z, i.e. z=W(i)z.
STEP 3. [Colouring the point]. Let c=(N*c+i) MOD N^K. Draw the point z with colour c MOD 13. Goto Step 2.

3. Tree parsing for IFS's

Assume the transformations w_i differ only by some rotation and translation. In this case they have the same contraction properties and we can start with one point z and calculate all points y with addresses of length smaller than a certain number to fill the whole fractal. If we have transformations with different contraction properties we can use the probabilities assigned to the tranformations to bound the length of the address. In general, if we repeatedly use strongly contracting transformations we will need a shorter address.
Let us consider the following example: We have two transformations w_0 and w_1 and use addresses up to length 2. Let z be the starting point. Then we consider the following points which can be arranged in a tree:



To calculate and draw all these points we can use an recursive algorithm for parsing binary trees. In each vertex we first have to calculate the corresponding point, second we have to process the left and third the right subtree starting in this vertex.
Using this algorithm we would get the following sequence of points:

$$z,\ w_0(z),\ w_0(w_0(z)),\ w_1(w_0(z)),\ w_1(z),\ w_0(w_1(z)),\ w_1(w_1(z)).$$

Algorithm 2. Tree parsing for IFS's. Let N, K and W(i) be as in algorithm 1. Further let P(i) be the probability of applying W(i). We use the following stacks: SC() for colours, SZ() for points SP() for probabilities and ST() for transformations. Q is the stack pointer. PP is a product of probabilities and EPS a small number bounding the depth of the tree.

STEP 1. [Initialize]. Let Q=0, SC(0)=1, and SP(0)=1
STEP 2. [Begin at some vertex]. Let I=ST(Q), z=SZ(Q), PP=SP(Q), C=SC(Q).

STEP 3. [Apply W(I), save, and draw]. Let z=W(I)z, PP=PP*P(I), C=(N*C+I) MOD N^K and Q=Q+1. Let SZ(Q)=z, SP(Q)=PP, SC(Q)=C. Draw the point z with color 2+(C MOD 13).
STEP 4. [Move left]. If PP>EPS then let I=0, ST(Q)=0 and goto step 2 else let Q=Q-1.
STEP 5. [Move right]. Let I=I+1, ST(Q)=I. If I<N then goto step 2.
STEP 6. [End]. If Q>0 then let Q=Q-1, I=ST(Q), goto step 5.

Last we give a BASIC program displaying a plane filling dragon curve in colour. The body of the program is a direct translation of the above algorithm. The point z of the plane is given by its co-ordinates x and y.
The first DATA statement contains the number n of transformations, the number k of digits of the address used for colouring and the values of xscale, yscale, xoffset and yoffset for plotting the points. In the next n DATA statements the parameters for the affine transformations are given.

```
10    DATA 3,3,180,150,180,100
20    DATA .5,-.28867,.28867,.5,-.5,.25,.3333
30    DATA .5,-.28867,.28867,.5,.5,-.25,.3333
40    DATA 0,-.57735,.57735,0,0,0,.3334
100   DIM sc(20),sx(20),sy(20),sp(20),st(20)
110   READ n,k,xscale,yscale,xoffset,yoffset
120   DIM a(n),b(n),c(n),d(n),e(n),f(n),p(n)
130   FOR i=0 TO n-1
140   READ a(i),b(i),c(i),d(i),e(i),f(i),p(i)
150   NEXT i
160   REM STEP 1: Initialize
170   q=0:sc(0)=1:sp(0)=1:eps=0.00001
180   REM STEP 2: Begin at some vertex
190   i=st(q):x=sx(q):y=sy(q):pp=sp(q):c=sc(q)
200   REM STEP 3: Apply transformation, save and draw
210   xnew=a(i)*x+b(i)*y +e(i)
220   ynew=c(i)*x+d(i)*y+f(i)
230   x=xnew: y=ynew:pp=pp*p(i)
240   c=(N*c+i) mod N^K
250   q=q+1:sx(q)=x:sy(q)=y:sp(q)=pp:sc(q)=c
250   xb=x*xscale+xoffset
270   yb=y*yscale+yoffset
280   COLOR 2+(c mod 13)
290   DRAW xb,yb
300   REM STEP 4: Move left
310   IF pp>eps THEN i=0:st(q)=0: GOTO 180
320   q=q-1
330   REM STEP 5: Move right
340   i=i+1: st(q)=i
350   IF i<n THEN GOTO 180
360   REM STEP 6: End
370   IF q>0 THEN q=q-1:i=st(q): GOTO 330
```

References.

[1] Barnsley, M.: Fractals everywhere, Academic Press 1988
[2] Mandelbrot, B.: The fractal geometry of nature, Freeman 1983

Intersecting universes

Michael Kirsch
58 Milton Park,
London N.6 5QA

It is well-known to fractalists that all over the complexities surrounding the Mandlebrot set there are little Mandlebrot figures, each subtly different from all the others.

Now, if you go to coordinates

Minimum fractal X = - 1.7665074446

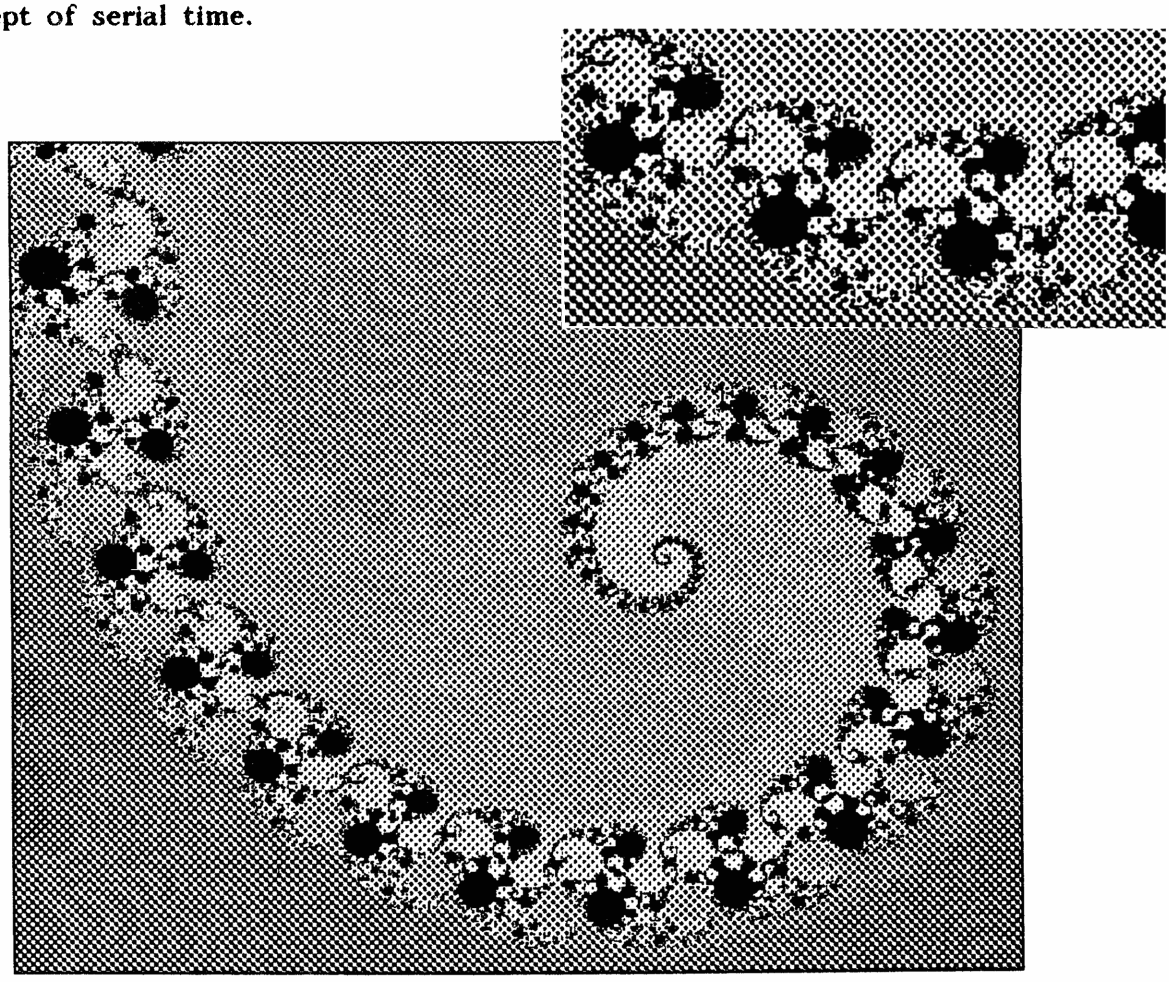
Minimum fractal Y = 0.041724440016

size: $2.3648553613 \times 10^{-8}$

max iterations: 600

you will find yourself looking at the edge details of the tail one of these.

This is where the little Mandlebrot people live, or rather, they have a kind of being. To actually make them live it needs the introduction of the concept of serial time.



You will notice on close inspection that what appears to be a set of humanoid figures arranged in a clockwise spiral is really ONE figure at different stages of Mandlebrot life. That is, you will notice that each figure is slightly different from those on either side of it, as though they were sequential frames of a cine film.

Now, to make them live it requires a program which can pick each figure out as a block, rotate it to the vertical, size it, and store it as a cine frame rather in the way the FR ZOOM program[1] allows.

If you then projected them on the screen in clockwise order, or anti-ckolockwise for that matter, you would see a Mandlebrot creature flash through a short and completely predestined " life. "

The edges of the Mandlebrot set contain all sorts of little cinematic creatures, usually only enough to make a dozen movie frames.

In the environs of the coordinates given above, there is a large number of similar spirals, **all inhabited with some kind of cinematic creature.**

It might be interesting to study these shy little creatures.

The Mandlebrot set and its infinitely detailed edges are a kind of **diagramatic universe,** having some of the basic essentials of any **working universe.**

1. There is a set of self-consistent laws in operation in the Mandlebrot universe, governing for instance, the style of detail inside each of the miniature Mandlebrot figures, and governing the angle at which the little Mandlebrot figures are set in the matrix of the plane of complex numbers.

2. The Mandlebrot universe has the necessary game aspect. That is, it has **a matrix** : the complex plane, **a set of counters:** the number of iterations at each point in the plane, and **a set of rules** which, when operated on the counters, in the matrix, makes the game take place.

3. It is all spoken for by ONE equation. The Mandlebrot creatures face the challenge of elucidating that equation from the rules which they may notice in operation in their universe. If their lives were not so short, and if only time were operating consistently in the Mandlebrot universe, they might be able to infer the details of Mandlebrot equation from the laws of the universe it generates and rules. But I think this knowledge might be wasted on them.

If you were to find or formulate the right equation, you could generate this shin-barking universe with it.

The realm of fractals is inhabited by a number of intersecting universes. There is one equation which, when iterated, creates very credible landscapes and relief maps of hitherto unknown countries.

I have this equation mounted for me in the FRACSURF program.[2]

You have to give this program the following:

A fractal number. The start-up fractal number on my copy of this program is 0.75

A seed. It has to be an integer, or zero. Try zero; it is the program's start up seed.

The fractal dimension of this works out at 2.334. One does not have to enter this.

Choose any viewing position. The start-ups are:

viewing position: - 200, - 900, 400
looking at: 0, 0, 0
light position 900, -100, 800

on my disc.

Given a few minutes, on this basis, the FRACSURF program will create and display a very plausible relief map of what could be a region of the Orkneys.

Now, if you alter the seed value and the fractal number only very slightly a completely different landscape will be generated.

Presumably, amongst all the possible fractal relief maps, there is one which closely mimics say, the contours of the British Isles.

To find it, you only need to know the seed value and fractal number.

If this program could be made to work in reverse, and allow one to input a particular contour map, say, of Australia, to be given the fractal coordinates, **we would have an immensely powerful way of storing any number of such maps.** All you would need is the FRACSURF program, and the coordinates which make it draw a particular contour map. You could store a million detailed maps in this form on the disk I am using for this letter, if you had a program which would spit out the fractal cordinates of any relief map presented to it.

You would need only one copy of the program, and have room for millions of sets of coordinates on one little disk. In a powerful processor, the program would be able to generate a detailed picture on the screen from a set of coordinates in a microsecond or so, and so, in theory, and with a lot of software development, the fractal equations could be used to store immense amounts of pictorial information and be able to display it cinematically.

I found your journal - Fractal Report, issue O -, very interesting, but although I know something about mathematics, I was puzzled by some of the algebraic notation I came across.

I am a complete beginner at programming, but I do know how to program my TI 59 Programmable Calculator and PC-100C printer.

As a matter of fact, I have written a few score programs for it exercising **a working universe model.** I would like to install them on disk for an ST. Perhaps you could put me in touch with someone who might be interested in helping me.

---

1. Both these programs are in the public domain, else I shouldn't be able to afford them.

2. The FRACSURF program is also in the public domain.