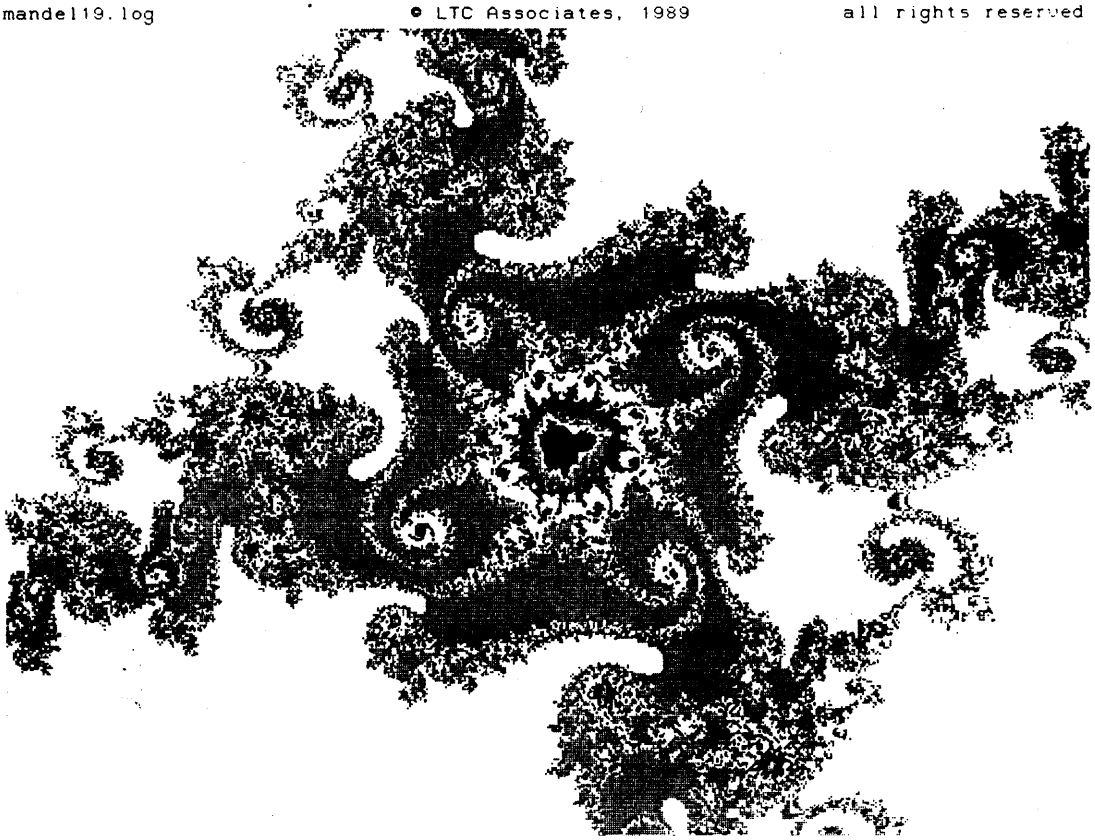
Fractal Report

Issue 2

mandel19.log



Larry Cobb winner (mono version) John Naylor

Published by Reeves Telecommunications Laboratories Ltd., West Towan House, Porthtowan, Cornwall TR4 8AX, United Kingdom. £10 UK, £12 Europe, £13 elsewhere. UK funds. (U. S. only $23 check payable to "J.de Rivaz".) Subscriptions backdated to volume start. Free subscriptions to future volumes for contributors. There are six issues per volume.

EDITORIAL *John de Rivaz*

It's official - it's MARTIN'S MAPPING. Dr Barry Martin has indicated that he feels that this would be the most appropriate name for the algorithm that is attributed to him and was used in the program in issue 0. He says that he does not feel that the name "Hopalong" given to the algorithm by Dr A.K. Dewdney in *The Armchair Universe* is suitable. Dr Martin says that he will provide us with an article some time in the future - an event I am sure that all will look forward to.

When we took in the subscriptions, we asked for readers suggestions. First may I apologise for any that I left out, especially those that sent letters and didn't write on their subscription forms! Also apologies to any Drs referred to as Mr - some of you seem coy about it - please make it clear when submitting articles.

Mr David Billington asked for plenty of practical examples, especally landscape generation. Personally I would also be interested in planet generation, especially after seeing a picture of a Jupiter model in an article on chaos in *The Planetary Report*. Three dimensional fractal generation has also been tried, such as the example pictures given in Peter Sorenson's *Fractals (Byte* Septmber 1984.) But I know of no published algorithms or programs for perspective drawings of them on small computers. (The *Byte* article did give a small listing for two dimensional slices.)

Mr O.J. Broadway asked if every illustration in *Fractal Report* could include details of how it was produced. If adhered to, this lofty (and popular) ideal may result in fewer pictures. However be sure that I will be doing my best to encourage practical articles.

Mr Owen Cunningham had the following ideas: 1. Ban the Mandelbrot Set. 2. Ban articles stating blatantly obvious facts. 3. Ban algorithm-less articles. 4. Articles should use proper English - foreign writers should hire an interpreter. 5. All algorithms to be in pseudo-code similar to PASCAL. In response I would point out that there is enormous interest in the Mandelbrot Set, and personally I have found new ideas in the articles submitted very interesting. Facts that may be blatantly obvious to some aren't so to others. Once those others have learned these facts, they may well come up with something sufficiently new to interest Mr Cunningham, so I hope that he will bear with us in the meantime. I will certainly encourage articles with algorithms, but to ban those without regardless of other considerations could be a disservice to readers. As to proper English, well as far as I am concerned what is important is subject matter - I am sure that there are many other publications that provide "deathless prose" to delight connoisseurs of the language. Further on, we will come across a gentleman who wants all programs in a standard BASIC, ie the one he uses on his machine. We can't please all of the people all of the time! Mr Cunningham should like QL SUPERBASIC, as it appears to owe quite a lot to PASCAL. Apart from all that Mr. Cunningham said he liked it!

Mr Graham W. Griffiths asks for useful applications. This would indeed be interesting if there is anything that could be explained within the space constraints of *Fractal Report*. Weather and Stock Market predictions have been vaguely hinted at in the past. Given a stock market or rainfall pattern as a series of numbers over time, could an iteration be produced to mimic these numbers? If so, it would be interesting to see what it tells us about the future.

Mr Vinay Gupta would prefer algorithms to actual programs, as he says they can more easily be converted. He would also like a reading list for beginners. Yes, a reading list may be a good idea, but I think that any readers if given the choice would prefer more articles. One possibility may be to produce a reading list as a separate publication, and add to it as time goes on. It could be free to subscribers on request. If Mr Gupta or anyone else would like to produce such a list, it would be most welcome. If more than one person produces a list, then there may be different books on each list, therefore a more comprehensive list would result. However I would like to insist that mail order sources and prices are given, otherwise people are just frustrated.

Mr G.O Lawton suggested a survey as to what equipment people are using. Many have volunteered this anyway, but we could conduct one next subscription time.

Mr Andy Lunness read issue 1 and claimed that the program for z^n=1 didn't work. If so, then how were the patterns produced? I suggest that there may be a problem in conversion. He was also worried by the fact that Dr Saupe's article contained an algorithm that called another algorithm not given in the article. Quite so - you have to get his book *The Science of Fractal Images!!!* He would also have liked a program to accompany Ed Hersom's article. But he concluded by saying that the range of articles was excellent, and apologised for nit picking, and that he would back us all the way.

Mr Paul McGilly mentioned a number of popular ideas, such as a reading list, not printing pictures without algorithm details, and requesting an article on fractal landscapes. However a particularly bright idea of his was to get volunteers with each type of machine to collect public domain fractal sofware onto one disk and offer it to readers for a modest copying fee. He will start the ball rolling with Atari ST software, which he will send to people for £3 + a disk, or £4 without. For more details, his address is in his article on page 8.

Mr Kenneth M. Murray and John B. Naylon asked for 68000 assembler language routines. As there are a fair number of QLs, Ataris and Amigas and a few Macs amongst the readership, this may not be a bad idea. However I would ask writers to make it very clear when the operating system is called for prints, beeps and plots and give enough information for conversion.

Mr Liam G.H. Proven has written about the range of programs he can offer. We haven't the space this issue to publish full details, but he has many fractal and similar programs for different machines, including IBM CGA/EGA/VGA, Macintosh, Amstrad CW, Spectrum. QL to be added soon. Prices range for £2 to £6 per disk, cartridge or cassette. SAE to Liam Proven, Heathercliffe Software, Imperial Terrace, Onchan, Isle of Man.

Mr Chris Reid asked for a letters page. Well, I do slip a few letters in odd spaces from time to time.

Mr Alan Richardson also asked for book prices and a book list. A separate publication as indicated earlier would fit the bill, although whether we would have two is doubtful. Probably the best thing would be to flag beginners' books, but what is suitable for beginners is probably a matter of personal taste.

Mr P. Sevenoaks asked for as many program listings as possible. Like others he may have difficulty in converting from algorithms, witness our continuing plea by a parent wanting to get his son off games playing for a C64 conversion of Martin's Mapping. (I don't want to have to buy a C64 to do it myself!)

Mr Cyril Thomasson prefers articles of the type written by John Sharp, with a program and properly documented references, as opposed to the Mark Datko type. Although he found it enjoyable and romantic and instructive, he found the unreferenced name dropping (Hausdorff, Richardson and Kaluza) annoying. This was partly my fault. Mr Datko did send us a separate article consisting of couple of pages of references, but many of them were similar to Mr Sharp's, and I wanted the space for more articles. As no prices or availability information was given, the references probably wouldn't have helped anyone without easy access to a good university library.

Mr David Tickle requested some good very basic background. No doubt this would not please those wanting no articles about the blatantly obvious!

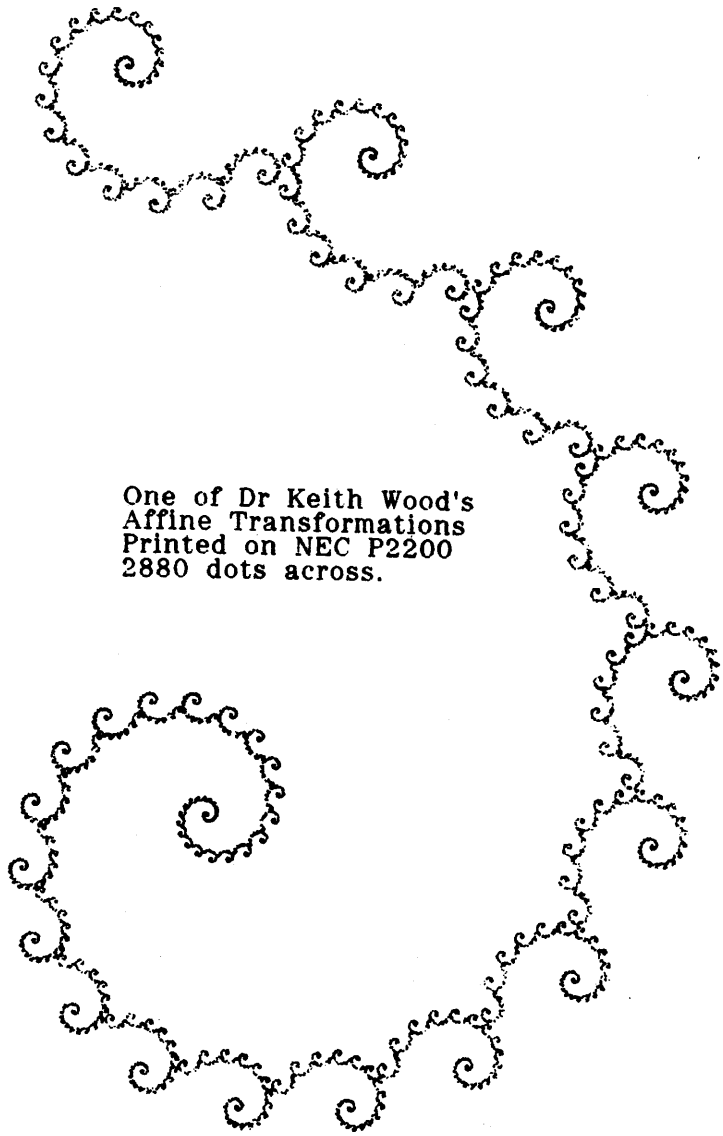
Mr Masco Verchoven was the chap who wanted everything in GW BASIC. Well, if you are going to pick a common language, that is a good one to choose, but I think it could cut out many excellent articles, and displease many readers who don't have or like PCs.

Dr Jules Verschueren wants us to concentrate on practical programming. He says that general information and reading matter is easily found in the scientific literature. Yes, I agree up to a point, but a little background does make *Fractal Report* more fun to read, and is essential to those who don't have the time or facilities to access other material.

Dr Keith Wood has a fractal suite in HD64180 assembler with 82716 graphics that runs on the SC84 computer but he thinks that anyone who has this machine probably knows already. If he is wrong, and you want one, then we'll pass on any enquiries.

Many of you expressed appreciation. However the magazine sinks or swims by the efforts of its readers. It is you the readers that should be congratulated for subscribing and submitting articles. Many similar special interest groups have extensive editorial boards and run symposia etc which only a tiny fraction of the membership attends, yet they are effectively paid for in part by all the readers of the newsletter.

Personally, as I don't like travel and only enjoy meeting other people in small numbers at a time, I hope to prove with *Fractal Report* that just as good results can be achieved by a simple straight forward honest-to-goodness publishing cottage industry that concentrates entirely on the written product. One day I hope to see an original named set or mapping or whatever first produced by a *Fractal Report* reader inspired by previously published articles in *Fractal Report* and publishing the algorithm for the first time in *Fractal Report!*



One of Dr Keith Wood's Affine Transformations Printed on NEC P2200 2880 dots across.

KAREN'S CORNER

We have added another page ie 2 sides this time to compensate for John's long editorial. As at 14th June we have 184 subscribers inc. exchange subscriptions with American groups and free subs for contributors to Issue 0. So far, we have had subs from USA, Canada, Australia, Eire, Holland, Belgium, Switzerland, France, Germany, Zimbabwe, Portugal and are hoping to get ones from Iceland, New Zealand and Czechoslavakia. We have 6 female subscribers and John is eagerly awaiting a contribution from one of them! We need 200 subs to make printing costs viable so we are seeking more publicity, currently aiming at secondary school teachers.

Apologies to Mr W.E.Thomson to whose surname we added a "p" in Issue 1. We try not to make mistakes but the harder we try the more we seem to make!

Dr Hugh Daglish recommends Martin's Mapping trials using
X1 = Y - SGN(X) * SIN(X) * SIN(X) and
X1 = Y - SGN(SIN(X)) * SIN(X) * SIN(X)
I had a go and got something that looked like a Dalinian picture frame. Dr Daglish kindly provided us with the "Fractal Report" lettering on the cover using his PaintBrush program and some more potential front cover fractals.

We have so many good articles we are not quite sure what is going in No 3 but contributions are always very welcome. Finally, we do not put a date on the cover as John says fractals are timeless!

AFFINE TRANSFORMATIONS FOR GRAPHICS.

by KEITH WOOD

There was a request for detailed explanations in issue 0, and here is an account of one type of fractal. The concept is very simple, and it is quite logical so that one can, in principle, think up a shape and write a set of transformations for it. Shapes requiring a large number of transformations will either require a lot of luck, or patience, or access to software to do it for you; but simple shapes can be done by inspection.

Imagine a circle with a radius of one, with its centre on the origin. Perhaps we would like it to be centred on the point (3,2). To do this we take a point on the circle, and add 3 to its x coordinate and 2 to its y coordinate. Of course, for a circle we can write the new equation directly, but for any shape whose equation may not be known, the procedure will work. We simply repeat the process a large number of times for different points until we have enough. This process can be represented by the equations:

$$X = x + 3$$
$$Y = y + 2$$

Where (X,Y) is the new or transformed point, and (x,y) is the initial point.

Perhaps we would like the circle to be smaller, having a radius of 0.6, say. This is just as easy. We multiply by 0.6 before adding the constant.

$$X = 0.6x + 3$$
$$Y = 0.6y + 2$$

Perhaps our circle is already centred at (3,2) and we want it moved to (1,5) and reduced to a radius of 0.3. Simply take away the values of the starting centre to move it to the origin, apply the factor, and move it to the new position.

$$X = 0.3(x-3) + 1 = 0.3x + 0.1$$
$$Y = 0.3(y-2) + 5 = 0.3y + 4.4$$

An ellipse results when the factor applied to the x coordinate is different from that applied to the y, that is the proportions of the image are changed. The factors expand or contract the image in directions parallel to the axes, but the result can be rotated.

A side problem is making sure the point being transformed does form part of the figure being transformed. In the general case we don't know, so by taking all points in turn, or by selecting at random, a point would be chosen and its value (say black or white) written to the transformed image. After a large number of repeats, the new image would be more or less complete. Later we will consider points that we know will lie on the image.

Besides translation, a rotation of the image may also be required. There isn't much future in rotating a circle, but the foregoing applies to any figure. I used a circle because it has a centre. Any figure can have a centre, we simply define it. The transformed image will relate to the transformed centre as the original image related to the original centre. Let us rotate a square, with its lower left corner at the origin and defined as its centre, making the axes two of its sides. Using a side of length one, we will rotate it by 30 degrees clockwise, so that the side which originally was the y-axis will now slope up to the right with a slope of root(3) = 1.732.

Its length is still 1 as there is no change in size, so what was the top left corner is now at the point (1/2, root(3)/2) or (0.5,0.866). Put another way, the top left corner is at (1sin30,1cos30). Points along the side in question will be shifted in proportion. A transformed point from this side of the square, for which x = 0, will be given by:

$$X = y\sin 30 \qquad Y = y\cos 30$$

Similarly, the points along the x-axis, forming the bottom of the original square, are also rotated by 30 degrees, and a point from this side for which y = 0, will be transformed by:

$$X = x\cos 30 \qquad Y = -x\sin 30$$

For any other point having non-zero coordinates, simply add the equations together:

$$X = x\cos 30 + y\sin 30$$
$$Y = y\cos 30 - x\sin 30$$

Check this for the top right corner (1,1) which becomes (1.366,0.366). The transformed coordinates give a diagonal of 1.414, as the original, and an angle with the x-axis of 15 degrees, previously 45. The equations can be extended exactly as before for changes of size and centres away from the origin. To move a square having a centre at 3,2 to a square rotated 70 degrees clockwise and reduced to 0.4 times the size of the original centred at 1,5 the equations are:

$$X = 0.4(x-3)\cos 70 + 0.4(y-2)\sin 70 + 1 = 0.137x + 0.376y - 0.162$$
$$Y = 0.4(y-2)\cos 70 - 0.4(x-3)\sin 70 + 5 = -0.376x + 0.137y + 5.854$$

This is all the maths there is in the whole process. Note that the constants are fully evaluated in the right hand version of the equation. This saves much computation time and allows a general form for the programme dealing with the equations.

To use the equations in a fractal process to generate graphics there must be more than one transformation, and which one to use is chosen at random. It needn't be chosen at random, but to cover the whole plot a system of choosing must be worked out for each plot, and while it is being thought out the random process will have it done. Being random it applies to any figure, once programmed it can be forgotten. It is achieved by generating a 'random' number and matching it to a scale of probabilities attaching to the transformations. (0=<p<.34 chooses set 1, .34=<p<0.5 chooses set 2 and so on up to 1.000, where the probability assigned to set 1 is 0.34, to set 2 is 0.16 and so-on to a total of exactly 1)

The second thing is that instead of taking a shape and moving it somewhere, there is no initial shape and the form of the resulting plot is derived from the equations. Every point calculated is plotted (after the first few), and the starting value for the calculation is the previous point. We see that the transformation is of itself, once it has discovered what that is. In other words, we plot a point, then transform that point, then transform the resulting point, and so-on. Thus every point plotted lies on a transformed image of some other part of the plot. The first few points are omitted while it settles into a routine. Until something has been plotted there is no basis to start. Ten points are enough to ensure compliance. Starting from a point known to form part of the desired plot, no points need to be omitted.

A fractal is defined as a self similar object. Part of a fractal image resembles another part and the whole without necessarily being exactly the same. The above procedure makes this happen.

A final requirement is that the transformation equations must always reduce the scale of the image, otherwise the plot might shoot off to infinity or at least be unstable.

Consider the Sierpinski triangle for which details appeared in issue 0. This is a triangle made from three similar triangles half the scale of the main triangle, each of which is made from three similar triangles half the scale, each of which . . . you've got it! See the drawing which gives the general idea. The origin is considered to be at the lower left corner.

If a point is plotted in the lower left subsidiary triangle, there will naturally be a similar point in the lower right one, separated by a distance of exactly half the length of the base of the whole triangle, say one. So an equation might be:

$$X = x + 0.5$$
$$Y = y$$

If the height of the triangle is also one, there is a point on the upper subsidiary triangle corresponding to the lower left one, displaced up by 0.5 and to the right by 0.25. Another equation might then be:

$$X = x + 0.25$$
$$Y = y + 0.5$$

The lower left subsidiary triangle is an exact half scale replica of the whole triangle, so a further equation is:

```
X = 0.5x
Y = 0.5y
```

One can think of others, which would be combinations of these. Before using them there is a further restriction, that the transformation must be reducing. the coefficients of x and y must be less than one. If they are not, control is lost. Y = y + 0.5 used repeatedly will increase without limit. Combining the above does the trick. One transformation reduces the previous coordinate by half. We can use this at the same time as adding the increment. If X = 0.5x and X = x + 0.5 both apply, then X = 0.5x + 0.5 also applies. The transformed value of x with the latter equation never exceeds 1 which is the base length of the triangle. Applying the change to the whole set of equations:

```
X = 0.5x + 0.5
Y = 0.5y              Plots a point at the lower right

X = 0.5x + 0.25
Y = 0.5y + 0.5        Plots a point at the top

X = 0.5x
Y = 0.5y              Plots a point at the lower left
```

Y = y is obviously stable, but the equations exist in pairs since they apply to coordinates, so both equations of a pair have to be adjusted.

The third pair takes any point in the whole triangle, and transforms it to the lower left subsidiary triangle. In effect the other two pairs do the same, adding constants to shift the point into one of the other two subsidiary triangles as indicated. To complete the whole plot we give each of the three pairs of equations equal probability to fill out each third of any part of the whole at a similar rate. Other probabilities simply waste time by needlessly plotting points over and over while some badly served corner of the plot gradually fills up. As it is there is still a good deal of wasted effort, but a fast programme can deal with this comfortably.

The probabilities are adjusted so that the total adds up to 1. Even if they added up to .9999, the time would come that the programme would hang when it got a random input corresponding to the .0001 not accounted for. To save a few micro seconds, the data are entered in descending order of probability, so that the number of if . . then situations computed is kept to a minimum.

Finally, sets of transformations typically give results around the origin and in the range 0 to +/- 2. To plot them the points need scaling to the dimensions of the screen, and an offset may need to be added to bring all points into the first quadrant (x and y both positive). The Basic listing uses the above data. Other sets of data (taken from Byte, January 1988) are:

```
X = ax + by + e
Y = cx + dy + f
```

FERN

m	a	b	c	d	e	f	p
1	.85	.04	-.04	.85	0	1.6	.85
2	-.15	.28	.26	.24	0	.44	.07
3	.2	-.26	.23	.22	0	1.6	.07
4	0	0	0	.16	0	0	.01

TREE

m	a	b	c	d	e	f	p
1	.42	.42	-.42	.42	0	.2	.4
2	.42	-.42	.42	.42	0	.2	.4
3	.1	0	0	.1	0	.2	.15
4	0	0	0	.5	0	0	.05

SQUARE

m	a	b	c	d	e	f	p
1	.5	0	0	.5	.5	.5	.25
2	.5	0	0	.5	0	.5	.25
3	.5	0	0	.5	.5	0	.25
4	.5	0	0	.5	0	0	.25

SPIRAL

m	a	b	c	d	e	f	p
1	.846	-.308	.308	.846	0	0	.9
2	-.163	-.163	.163	-.163	1	-1	.1

The spiral is the illustration in the article in Byte, for which the authors leave readers to find the equations. Without wishing to boast, but to illustrate the correctness of the above reasoning, I got the equations right first time.

The BASIC listing:

```
10 REM This simple routine allows a maximum of four transformations in the IFS
20 DIM A(4),B(4),C(4),D(4),E(4),F(4),P(4)
30 REM Transformation data for the Sierpinski Triangle. Line 60 no. of transformations,
40 REM lines 70,80,90 are the data for the transformations in descending order of p.
50 REM m is the first data item variable name.
60 DATA 3
70 DATA .5,0,0,.5,0,0,.34
80 DATA .5,0,0,0.5,0.5,0,.33
90 DATA .5,0,0,.5,.25,.5,.33
100 READ M
110 REM cumulative probability
120 PT = 0
130 FOR J = 1 TO M
140 READ A(J),B(J),C(J),D(J),E(J),F(J),PK
150 PT = PT + PK
160 P(J) = PT
170 NEXT J
180 REM set up for plotting mode, here > 128x128 pixels
190 GRAPH 1
200 XSCALE = 128
210 YSCALE = 128
220 XOFFSET = 0
230 YOFFSET = 0
240 X = 0
250 Y = 0
260 REM do 10000 iterations
270 FOR N = 1 TO 10000
280 PK = RND(1)
290 REM The next line is for m<=4
300 IF PK<=P(1) THEN K=1 ELSE IF PK<=P(2) THEN K=2 ELSE IF PK<=P(3) THEN K=3 ELSE K=4
310 XNXT = A(K)*X + B(K)*Y + E(K)
330 YNXT = C(K)*X + D(K)*Y + F(K)
340 X = XNXT
350 Y = YNXT
360 IF N > 10 THEN PLOT X*XSCALE+XOFFSET,Y*YSCALE+YOFFSET,1
370 NEXT N
380 ? "Type CONT to continue"
390 STOP
400 TEXT
410 END
```

Line 360 should be adjusted to your basic version. In this listing the final '1' plots white on black. Line 400 reverts to text mode when you are tired of it.

Some Basics truncate the expressions in PLOT (line 360) and if a plot is being transformed about an axis an unsymmetrical one pixel offset can result. This is corrected by adding 0.5 to the offset value which rounds the plotted value. All the calculated values are in the first quadrant for the Sierpinski triangle, there are no offset values and no need for the 0.5 addition.

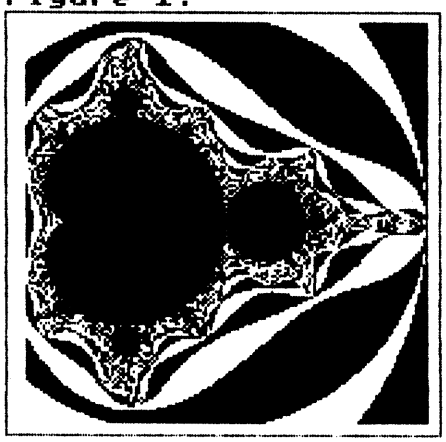
A Variation on the Mandelbrot set.

By Paul McGilly.

It is generally known that the Mandelbrot set has reflectional symmetry in the real axis (figure 1) and that Julia sets can have as many as two reflectional symmetries and one rotational symmetry (figure 2). [In general Julia sets which are not based on one of the axes only have rotational symmetry (figure 3).]

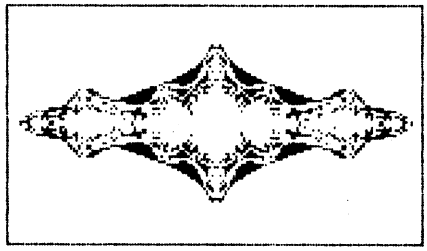
Whilst I was investigating the effect of slight changes to the Mandelbrot algorithm I happened upon the highly symmetrical set shown in figure 4. This set clearly has 3-fold rotational symmetry and 3-fold reflectional symmetry. When zooms are carried out on this set it can look remarkedly like the Mandelbrot set (figure 5).

Figure 1.



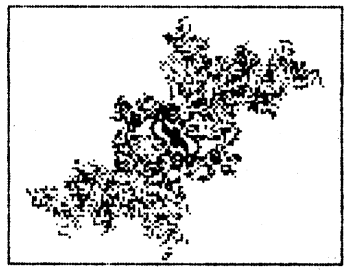
Normal Mandelbrot, showing reflectional symmetry.
XCORNER=-2 , YCORNER=-1.25
SIZE=2.5 , Colour=no. of iter.

Figure 2.



Julia , showing reflectional and rotational symmetry.
XCORNER=YCORNER=-2 , SIZE=4
JXCONST=-1.25 , JYCONST=0
Colour = No. of iterations

Figure 3.



Julia, with only rotational symmetry.
XCORNER=YCORNER=-2
SIZE=4 , JXCONST=-0.194
JYCONST=0.6557
Colour = No. of iter.
(Iter. < 15 NOT drawn.)

The normal iteration formula is :-

$$x+iy:=(x^2-y^2+a) + i(2*x*y+b)$$ where x,y,a,b are real.
(more normally $z=x+iy, c=a+ib, z:=z^2+c$)

The set shown in figures 4 and 5 is obtained by replacing the x^2-y^2 by y^2-x^2 so that the iteration formula becomes :-

$$x+iy:=(y^2-x^2+a) + i(2*x*y+b)$$

A simple program to draw pictures of the symmetrical set on the Atari ST in Fastbasic is given on the next page. It can be easy modified for other machines.

Suggestions for future work.

1) Is there anything special about the images of the Julia equivalent of this set? (eg. extra symmetries?)

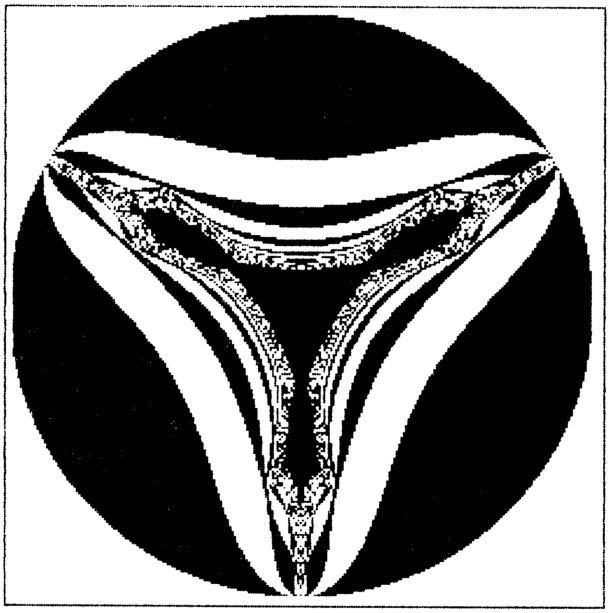
2) Why is there the high degree of symmetry?

The Author. Paul McGilly is a first year computer science/maths undergraduate at York University. Please address any correspondence to 43 Wellington Road, Hatch End, Pinner, Middx, HA5 4NF.

References. For an explanation of what the Mandelbrot set is and how to draw it see Scientific American, August 1985, Computer Recreations, pages 8-14. For similar details of the Julia sets see the November 1987 issue of Scientific American, pages 118-122.

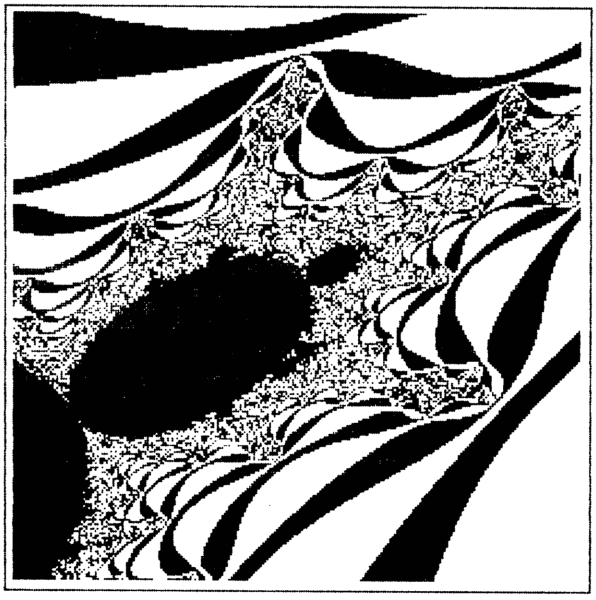
```
\ Program to draw the highly symmetrical set.
\ By Paul McGilly. (April 1989).

HIDEMOUSE                : \ Sets up a 200*200
GRAFRECT 0,0,200,200     : \ drawing window, and
CLG 0                    : \ clears the screen.

\               This section is to be changed by the user for
\               different coordinates and iteration values.
XCORNER=-2:YCORNER=-2    : \ Defines the minimum values of
                         : \ area to be drawn.
SIDE=4                   : \ Length of the sides of area drawn.
MAXIT=100                : \ No. of iterations before aborting.
SIZE=200                 : \ No. of pixels in sides of the
                         : \ square drawn.
GAP=SIDE/SIZE

XCONST=XCORNER           : \ This is a standard Mandelbrot drawing
FOR N=1 TO SIZE          : \ program. See the references for details.
  XCONST=XCONST+GAP

  YCONST=YCORNER
  FOR M=1 TO SIZE
    YCONST=YCONST+GAP

    X=0:Y=0
    C=0

    REPEAT
      XTMP=X*X:YTMP=Y*Y

      TMP=YTMP-XTMP+XCONST      : \ This is the special line.
                                : \ The normal algorithm uses :

                                : \ TMP=XTMP-YTMP+XCONST.
      Y=2*X*Y+YCONST
      X=TMP

      C=C+1

    UNTIL (C>MAXITT OR XTMP+YTMP>4)

  IF C>MAXITT THEN MARKCOL C MOD 2: PLOT M,N    : \ Plot the point
              : \ in a colour according to the no. of iterations.
NEXT M,N
```

Figure 4.



Modifed Mandelbrot. XCORNER=YCORNER=-2 , SIZE=4
Colour = no. of iterations. (MAXIT=250)

Figure 5.



Modified Mandelbrot. XCORNER=-0.8025 , YCORNER=1.055
SIZE=0.25 , Colour = No. of iterations. (MAXIT=250)

The first LARRY T COBB PRIZE Awards

My disappointment comes from the low number of entries. I hope it's because you are all feverishly searching for good entries. The main point of the competition is to exchange ideas about interesting areas of the Mandelbrot and Julia Sets, although the prize, of a high resolution colour print of the best entry, is well worth having. So don't be mean with your results - with our joint efforts, we can equal the processing power of a CRAY. Also, it's worth considering that, with so few entries, you are very likely to win the next round!

My consolation, however, comes from the very high quality of the entries. This has made it very difficult to choose a winner. Some readers sent me prints of their entries and, whilst this was useful, it is not essential - the co-ordinates, number of iterations and a short description will suffice.

The first entry, from Roderick Stewart, is from the area around the "neck" spur in the Mandelbrot Set. These patterns on the right hand side produce the trunk-like shapes, usually known as the Elephant Heads. I was so impressed with the sample output that Mr Stewart sent that I awarded him a special prize for a remarkable result from a low resolution printer.

Mr Stewart's entry was:

```
Mandelbrot routine using Log colour mapping *
offset = 0, start = 40, stop = 200 , C value = ( 0 , 0 )
side = 0.00025 , centre = ( -0.7381939 , 0.1938802 )
```

John Naylor suggested two areas on the other side of the "neck". The images are quite different here with beautiful double spirals and complex Jelly Fish shapes. I am sure that many of you will know what a extraordinary area this is for investigation, and it was for this reason that I gave him the prize. Congratulations John.

Mr Naylor's entry was:

```
Mandelbrot routine using Log colour mapping *
offset = 0, start = 150, stop = 500, C value = ( 0 , 0 )
side = 0.00002 , centre = ( -0.768358960101 , 0.108218708873 )
```

Steve Wright (probably by accident), suggested a Julia Set from a region described by John Hubbard, which was itself based on Mandelbrot's own San Marco dragon. So I can't give Steve many points for originality but I am grateful to him for reminding me how attractive these dragons are.

Mr Wright's entry was:

```
Julia routine using Log colour mapping *
offset = 0, start = 2, stop = 500, C value = ( -1.25 , 0.01 )
side = 1.5 , centre = ( 0 , 0 )
```

Michael Collins sent me an impressive range of "zooms" generated by the Psychobrot program. It's a very interesting area in the head of the Mandelbrot Set but you will need to use lots of iterations to fill in the swirling detail of these images.

* The details are suggestions only and are optimized for use with my Fractal Investigation Routine, DRAGONS

Mr Collins' entry was:

Mandelbrot routine using Log colour mapping *
offset = 0, start = 40, stop = 500, C value = (0 , 0)
side = 0.03 , centre = (-1.11 , 0.23)

With Ian Entwistle's entry, it was me that won a prize! Ian is able to produce his own high resolution colour prints and he sent me a beautiful plot. I would like to thank him for entering into the spirit of the competition although he didn't want to win the prize. He tells me that he is holding an exhibition so, if you see it advertised, I would think it well worth a visit. His prints use a novel colour decomposition technique - perhaps he can be persuaded to share his methods with the readers of Fractal Report?

Dr Entwistle's entry was:

Julia routine using Log colour mapping *
offset = 0, start = 0, stop = 300, C value = (0.32 , 0.043)
side = 3.2 , centre = (0 , 0)

Several people have asked me if the improved speed of calculation really justifies the high cost of a maths co-processor chip for the IBM PC, so I did some tests. My own Fractal Investigation program, DRAGONS, runs 16.5 times faster on the full Set! However, I have always thought that the cost of an 8087-2 was excessive at around £100 so I have made some inquiries. One company, called Netland, offers the Intel chips at the much reduced price of £50 + VAT. Their 8087 is rated at 5MHz but the Amstrad PC range use an 8MHz clock (assuming you have an Amstrad or an XT clone). This would seem to make it useless but my engineering friends assure me that chips often run much faster than specified, and Netland offer a money back guarantee if it does not work!

A cheaper improvement would come from replacing the main 8086 processor with an NEC V30 chip. Most operations would only run 20% faster but, for a special price of £12 from James Lucy, it represents a useful improvement.

If the idea of opening up your PC does not fill you with horror, both are fairly simple to install. If you are interested please direct all your inquiries and orders to the companies below, quoting Fractal Report to get the special prices. Please note that I have no connection (financial or otherwise) with either company.

Intel 8087 chips (£50 + VAT) from: Netland, Magnolia House, Stetchworth Road, Woodditton, Newmarket, CB8 9SP Tel 0638 731044

NEC V30 chips (£12, no VAT) from: Mr James Lucy, Dove Communications, 17 Taylors Road, Rowhedge, Colchester, CO5 7EG Tel 0206 868159

DRAGONS UPDATE

Version 2 of my Fractal Investigation program is now available. There are many detailed improvements but the major feature is the support for the Amstrad 2000 series 256 colour mode. Send £14 for this most comprehensive and easy-to-use program, (for IBM PC compatibles with a colour display), or SAE for more details.

Competition entries and program orders to: Larry Cobb, c/o Bay House, Dean Down Drove, Littleton, Winchester, Hants, SO22 6PP

MELTING MANDLEBROTS

by Steve Wright

By now there can't be many people who haven't heard about the Mandlebrot set, the giant fractal with hidden depths. The Mandlebrot set is a map of the chaos in the equation z=z*z+c plotted in the complex plane, that is z=x+iy and c=p+iq where i is the complex part being the square root of minus one. With initial values of zero for x and y, iterating the equation while varying p from -2.25 to 0.75 and q from -1.5 to 1.5 generates the familiar Mandlebrot set image. To calculate the Mandlebrot function the equation is simplified so that the real part x^2-y^2+p and the imaginary part 2xy+q ,are calculated separately. These parts are then substituted back into the equation as the new x and y values. It is necessary to do these calculations for each pixel of the image, until either n ,the number of times that the substitution is done, is greater than a maximum number of iterations, say 255 : or x^2+y^2 is greater than some limit. This limit is often set to 4, but better images result if this limit is increased to 100 or greater; this doesn't usually increase the computation time of the function very much, as once the limit of 4 has been exceeded, the equation becomes chaotic and usually quickly exceeds any reasonably larger limit.

The Mandlebrot function for each point (pixel) p,q is :-

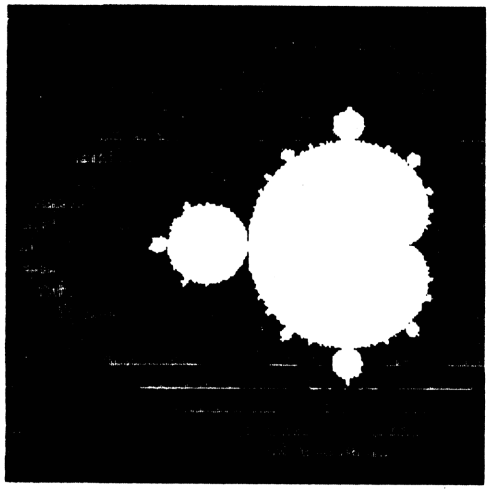
```
Mandlebrot function(p,q,)
 newx=0 : newy=0  ;initial loop values for x and y
 n=0              ;iteration count
 limit=100        ;max value for equation
  Repeat
   n=n+1
   x=newx :y=newy
   newx=x*x-y*y+p
   newy=2*x*y+q
 Until n=255 or x*x+y*y)limit
  function=n
```

The resulting number n is the colour for that pixel, unless n=255 when the colour is black. The colour is a measure of the stability of that point, black being completely stable. Near the boundary of the set (the black bit in the middle) the points are more stable and therefore to get more detail, especially at high magnification, it is better if the maximum n is increased.

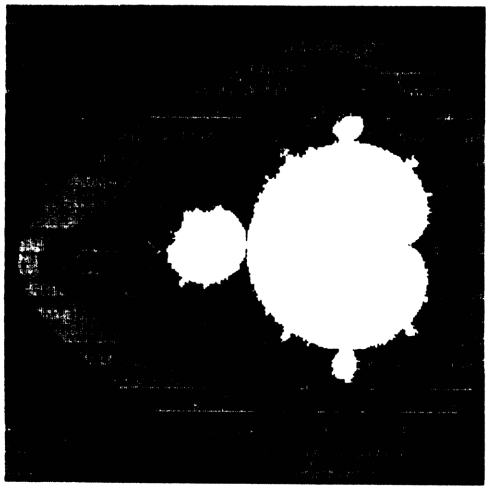
All the numbers x,y,p and q are floating point reals so with the comparison equation x*x+y*y) limit this comes to about ten floating point operations per pixel per iteration. On average this is for a screen size of 256*256 pixels something like 256*256*10*128 which is greater than 83 MFLOPS. So it can be seen that a great deal of computing power is required to generate these images in a reasonable time. This has been shown by the fact that manufactures of high power processors are begining to use the Mandlebrot set image as a yardstick to demonstrate the power of their products. It can also be demonstrated with some examples on a BBC B in basic it takes over 13 hours to generate the Mandlebrot set. Using the Archemedies in basic it takes about 4 hours, in compiled basic only 40 minutes but driving the RISC processor directly with machine code routines for the main function, this is reduced to only 2 minutes.

Parts of the Mandlebrot set can also be examined in more detail by creating a window and then magnifying that part by changing the range over which p and q are varied. This zooming in is only limited by the resolution of the maths, and shows the self similar

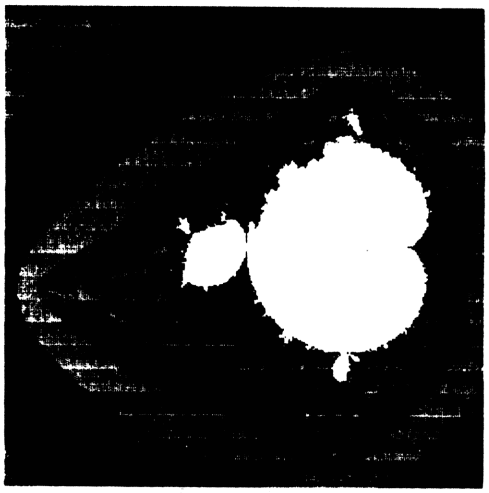
Initial values for x & y



0.1



0.2



0.3

properties of a fractal image. There are similar shapes and patterns almost everywhere you look and you can zoom in on little black spots only to find that they are small Mandlebrots which can also be zoomed in on ad infinitum.

Another interesting aspect of the Mandlebrot set is that it can also be viewed as a set of Julia sets. A Julia set is generated by using the same equation as for the Mandlebrot set, but by varying x and y (for a particular value of p and q) over the range -1.5 to 1.5 for the complete set. Details can be zoomed in on in the same way as the Mandlebrot set by using smaller ranges for x and y. However if you draw an imaginary line through the Mandlebrot set and use the values of p and q at points along this line a sequence of Julia sets can be generated. When these sequences are put together in the form of a 'movie' the Julia set image metamorphazizes; growing and changing, twisting, dancing, disintegrating, collapsing into a myriad of beautiful and chaotic yet structured patterns.

There are also other equations that can be treated in the same way to generate Mandlebot and Julia sets, for example (see PCW June '88) e^z, sinh(z) and cosh(z). However these other equations are generally more complex and therefore require even more computing power or longer times. An equation that is only slightly more complex therefore seemed the most likely candidate for investigation. This equation cz-cz^2 , is similar to the original and has similar chaotic charateristics ,but there is something odd about it. To generate a Mandlebrot image you would normally set z to zero, but with z=0 in this equation no amount of iteration will generate anything because the equation is equal to zero when z=0 and dosn't change (with z=1 the equation is also static). There seemed no other reasonable way to continue, other than to try various values for x and y. I started with 0.1 steps, and the results produced a sequence similar to the Julia set sequences.

I was intrigued to see what would happen if the initial values for x and y in the Mandlebrot function were made non zero, and to see if different values, would generate a sequence. I modified the Mandlebrot function and tried values of x and y of 0.1 to 1.0 in 0.1 steps.

I was amazed to see the Mandlebrot begin to deform, chaos within chaos, what was happening. I continued with the sequence the Mandlebrot was melting and dripping off the screen; little islands formed and drifted off to form new worlds. Mountain ranges and valleys appeared and disappeared new features were formed, 'deformed and reformed. It has to be seen to believed that a simple equation could behave in such strange and beautiful ways.

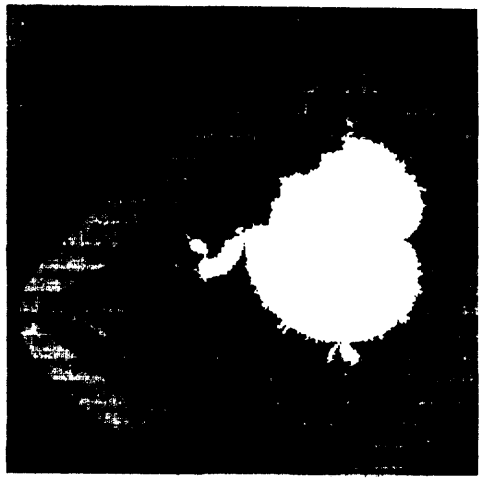
Any Mandlebrot program can be modified to produce these Melting Mandlebrots. First add an input statement to get the melt factor

```
Input "Melt factor" ,melt
```

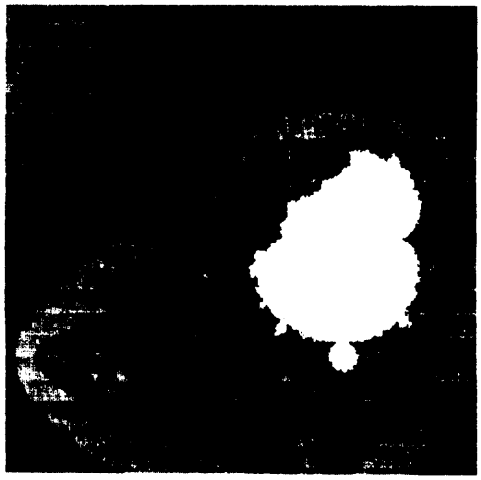
Then change the funtion so that the initial loop values for x and y are redefined

```
Mandlebrot function (p,q,melt)

  newx = newy = meltf
```

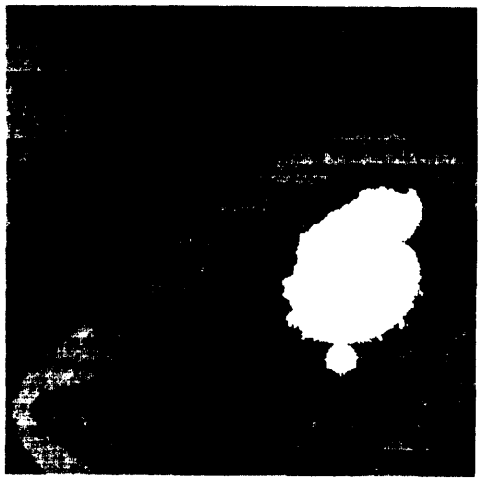
The rest of function remains the same.



0.4



0.5



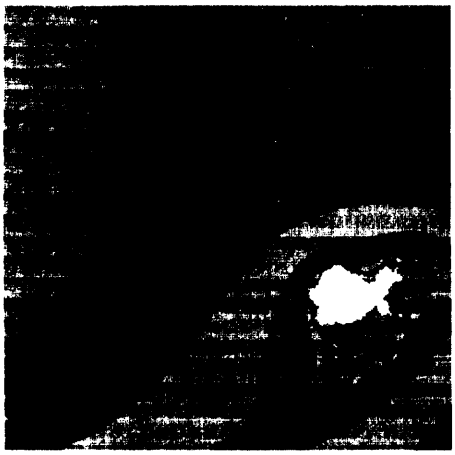
0.6

Now you can watch the Mandlebrot melt, follow the drips as they drift off the screen, fly over the newly formed hills and valleys. Zoom in on and watch these new and, 'unexplored ?!' panoramas change. I believe this is an unexplored property of the ' Mandlebrot equation ' and have been unable to unearth any explanation, or information about it.

I have looked back over the previous articles I've read in PCW ,Scientific American ,Byte ,Acorn User and followed references to various books 'The Fractal Geometry of Nature ' by Benoit Mandlebrot , 'The Beauty of Fractals ' by H.-O.Peitgen and P.H.Richter and the latest book I've seen ' The Science of Fractal Images ' by H.-O.Peitgen and Deitmar Saupe but can find nothing. It seemed possible I had made an error somewhere, so I visited a friend and we modified a demo program for the Archemedies, it gave the same results.

0.7

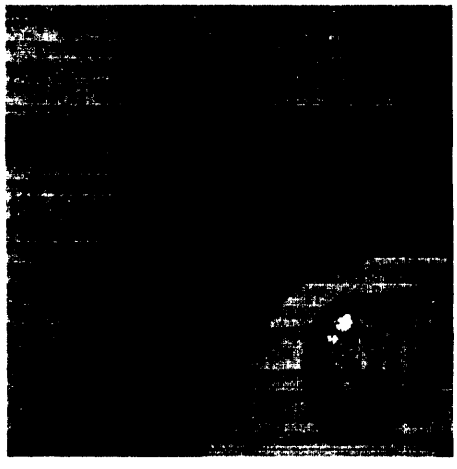
N.B. add Arch changes here if relevant
.... lines changed in the disk program are
900 added ... OR key$="I"
added lines
920 IF key$="I" THEN
930 INPUT "Melt factor :" MF
changed lines
1350 LDR u,MFVAL
1360 LDR v,MFVAL
added line
1840 .MFVAL EQUD MF *(1<<27)
the above line converts the input value to an integer as the RISC processor only deals with integers.



0.8

These new "Melting Mandlebrots" can now be examined in the same way as the Mandlebrot set, but don't forget the melt factor for x and y. Why not transform an old favourite, watch it bend, distort and melt away. But be careful always start off with a small melt factor some of the features melt away very quickly. You might need a melt factor as low as 0.01 or less especially if you choose something in the top half of the set(i.e. with q positive) where things change very rapidly. They also move around so be prepared to do a little hunting. You can also trace interesting patterns backwards to see how they were formed, try x=y=0.9 :p=-0.1016 :q=-0.71614 with a frame size 0.03. This shows a fascinating triple point where the fingers of the pointers have little triplets copies all down the side instead of Mandlebrot copies; where did it come from? A lot of the little bits that break off and drift around are very similar to Julia sets, is there any relationship linking them in some way?

For other effects the melt factor can be added for x and y separately, i.e. newx=meltx : newy=melty
This also gives strange deformations and melting sequences that can be explored.



0.9

FRACTALS ON THE BBC MICRO - AN INTRODUCTORY PROGRAM By Phil Edmonds

The following program is suitable as an introduction to fractals on the BBC micro. It uses the BBC MODE1, a four colour medium resolution screen mode. It works by iterating the function f(x)=x^2+c where x and c are both complex numbers with real and imaginary parts, with x being set to Ø on the first iteration. The program requests a range for complex and real parts for the points to be iterated and automatically scales itself so that the interval between successive points to be iterated matches the screen resolution available (320x256 pixels). It then interates the function for each point; if the iteration runs off towards infinity, the point is plotted in a colour dependant on the number of iterations it has undergone. If, after 100 iterations, the point has not run off to infinity (i.e. it is a member of the Mandelbrot set) it is not plotted but rather left blank. The resulting plot occupies a 1000x1000 grid (100<x<1100, 23<y<1023). As usual, the program's main failing is speed, so line 10 automatically saves the screen when the <ESCAPE> key is pressed, and the final line saves the screen if the program is completed. The program may thus be left to run for a long time and the resulting images stored for later viewing.

Lines 130-150 select the screen colours to be used, and clear the initial screen. These lines must be repeated in any program to later display stored images (using *LOAD PIC) to give the same colours as the original program run.

```
 10 ON ERROR *SAVE PIC 3000 7B00
 20 REM Mandelbrot set
 30 REM By Phil Edmonds
 40MODE7:PRINT:PRINT
 50 PRINT"Complex number range to be iterated:":PRINT
 60 INPUT"real lower limit ";CL:PRINT
 70 INPUT"real upper limit ";CU:PRINT
 80 INPUT"imaginary lower limit ";DL:PRINT
 90INPUT"imaginary upper limit ";DU:PRINT
100 CRANGE=(CU-CL)
110 DRANGE=(DU-DL)
120 MODE1
130 VDU19,0,6,0,0,0:VDU19,1,5,0,0,0
140 VDU19,2,4,0,0,0:VDU19,3,0,0,0,0
150 GCOL0,131:CLG
160 stepC=CRANGE/320
170 stepD=DRANGE/256
180 FOR D=DU TO DL STEP-stepD
190 FOR C=CL TO CU STEPstepC
200 A=0:B=0
210 J%=J%+1
220R=A^2-B^2+C
230I=(2*A*B)+D
240 IF R<-2 OR R>2 OR I<-2 OR I>2 THEN 280
250 A=R:B=I
260 IFJ%>100THEN290
270 GOTO210
280 GCOL0,(INT(J%/33)):PLOT69,100+(((C-CL)/stepC)*3.125),23+(((D-DL)/stepD)*3.
906)
290 J%=0
300 NEXTC
310 NEXTD
330 *SAVE PIC 3000 7B00
```

Computing the Julia Set in C language (by Cliff Wootton)

On receipt of a letter from the editor of the Fractal Report, I was inspired to sit down and see if I could write a small example program on the micro at home.

I wrote this program in C language and purposely separated any machine specific parts out so that users of machines other than the Apple Macintosh could recreate the pictures. The specific compiler I used was LightSpeed C which proved well up to the task.

I make no apologies for programming style or lack of it. This is in fact only the first program I have completed in C language since starting to learn it.

The program was designed from the notes at the back of the book by Heinz-Otto Peitgen called 'The Beauty Of Fractals' (pub Springer Verlag). Although not as algorithmic in content as the later work, 'The Science of Fractal Images', the notes are not hard to follow.

If you are implementing this program on another machine, then you may wish to change the contents of the following procedures:-

• init_window()	Initialise display.
• pause()	Wait for operator.
• draw_spot(x,y,c)	Draw a pixel.

The init_window() procedure simply clears the screen and on other machines would perform any necessary mode changes and colour table setups necessary. On the Macintosh, it simply initialises the various toolbox managers and clears the screen to solid while.

The pause() procedure should cause the screen to remain intact and wait for a manual intervention before returning control to the operating system at the end of the program. If you wish to save the screen before it is cleared, you may wish to add some code to this procedure. In the Macintosh, the pause facility instructs the dialog manager to draw a non existent dialog box and then wait for the [RETURN] key to be pressed before returning to the caller. To store the screen in the Macintosh, you can press [SHIFT]+[CONTROL]+[3] to save a snapshot of the screen into a MacPaint compatible file.

The draw_spot(x,y,c) procedure is where you set the colour of a single pixel at x,y to the colour c. In a Macintosh Plus, there is only a monochrome screen. In this procedure, I simply test to see whether the value is odd or even and then set the pixel or not depending on the result. If you have a colour display then you can generate more interesting results.

Now to the main program. First of all, we must declare the name and type of any local variables. Since these are within the braces of the main() procedure, they are only available within it and are not global. The following are declared:-

• scr_width	Width of screen display (int).
• scr_height	Height of screen display (int).
• h_pos	Current horizontal position being computed (int).
• v_pos	Current vertical position being computed (int).
• pix_col	Iteration counter used as colour value (int).
• xmin	Left edge of area of interest within fractal domain (real).
• ymin	Bottom edge of a.o.i. within fractal domain (real).
• xdelta	Horizontal extent of a.o.i. within fractal domain (real).
• ydelta	Vertical extent of a.o.i. within fractal domain (real).
• xold	Input value to iterative expression (real).
• yold	Input value to iterative expression (real).
• xnew	Output value to iterative expression (real).
• ynew	Output value to iterative expression (real).
• threshold	Value to test against to cease iteration (real).
• const1	Arbitrary constant (real).
• const2	Arbitrary constant (real).

The descriptions above should be sufficient without going into further detail. Now let us look at the rest of the program structure.

The screen width and height are initialised to the area of the display that you wish to cover. This indicates a rectangle which is mapped corner by corner to a rectangle measured within the fractal domain. It is in effect a window into the fractal domain being processed. The xmin/ymin are the bottom left corner (or upper left depending on where the origin is on your display) and the xdelta/ydelta indicates the area of the fractal domain being viewed. To pan around the fractal domain, simply modify the values in xmin/min. To zoom in or out, adjust the values in xdelta/ydelta accordingly.

The threshold value is chosen fairly arbitrarily at first but will affect the number of iterations and hence colour levels that are present in the final image. Increasing the threshold value increases the processing time somewhat. Threshold values that are too high pay no dividends as the extra detail that is computed is lost in the quantisation noise of the pixel display. The arbitrary constants also affect the number of iterations as they are added into the total each time round the loop.

After initialising the display, the program structure is essentially a series of nested loops. The first loop steps through the display one line at a time. Starting at one, the v_pos variable is incremented by one each time round the loop. Within this loop, a similar loop is executed, in this case indexing the h_pos variable one pixel along the line each time. With this structure we are addressing all pixels within the display one at a time.

At this point it is worth mentioning the difference between parallel and serial architechtures and how they will affect the performance of the program. A serial architecture will perform the same process on each pixel, one after the other. A parallel architecture on the other hand will operate on many (possibly all) pixels at once. Since the process to be performed is identical for each pixel, the computation of fractal images lends itself very well to implementation on vector (parallal) processors rather than scalar (serial) processors.

To compute the Julia set, the operations to be performed per pixel are as follows:-

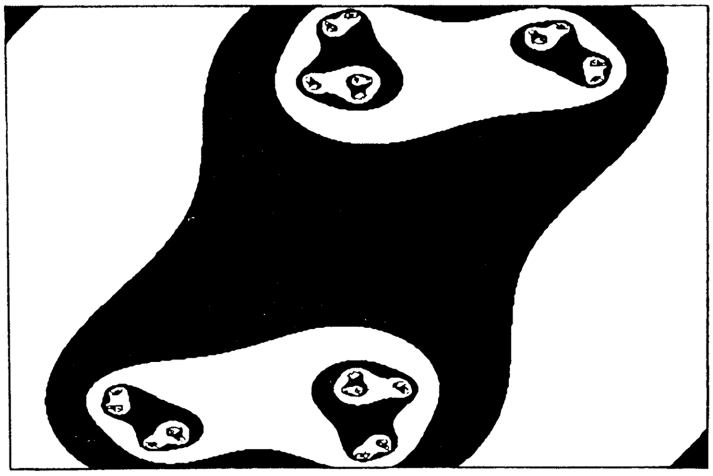
- Seed the equation with starting values.
- Start a loop structure for the iterative equation.
- Compute new values for X and Y.
- Increment the pixel colour.
- Test the threshold condition.
- If the threshold condition has not been met then repeat the iteration.
- Otherwise plot the pixel.

The seed values for the equation are coordinates within the fractal domain indexed by the same number of steps as there are pixels on the screen. That is, if we are halfway across the screen, the x input value will be equal to xmin plus half of xdelta and likewise for the y axis. This ensures that for each pixel, we will have different starting values although at this point they are linear in both axes (that is bilinear).

The loop counter is in fact the pixel colour which we star at 0 and increment by 1 each time round the loop. The iterative equation evaluates {$xold^2$ minus $yold^2$ plus one of the arbitrary constants} and places the result in xnew. The ynew value is computed from {2 times xold times yold plus the other arbitrary constant}. Once the values have been substituted back into the xold/yold variables, the threshold can be tested.

The threshold condition is met when the formula {$xold^2$ plus $yold^2$} is equal to or greater than the threshold value. It is possible to input values that when computed iteratively, will never reach the threshold and the program will become locked into a closed loop. You can add extra tests for these conditions during the iteration but performance degrades very rapidly when more conditions are being tested.

None of my images, although interesting, bore any great resemblance to any of those in Heinz-Otto Peitgen's book. Perhaps I have by mistake stumbled across a hitherto unknown fractal domain although I doubt that somewhat. The first one took some 35 minutes to compute on a Macintosh Plus. Zooming in so that small features filled the screen, (2:1 zoom) increased the computation time to about an hour. A closer zoom still (6:1) increase the rendering time to close on 2 hours. As you zoom in closer to more interesting features, the iteration count increases and hence the images take longer to compute. For this fractal, zooming in reveals the self similarity which is the hallmark of any fractal set and in the case of the equation described here the level of complexity appears to remain the same howver far you zoom in or out. Some example input values are given below in Table 1. Zooming out to an area of -10.00 to 10.00 in x and y reveals that the fractal set being computed is a single island contained wholly within the region -1.5 to 1.5 in x and y. Some computation time might be saved by evaluating only one half of the view as the other half is a reflection. I would guess that neither of these observations necessarily apply to all fractals as a general rule.



Example One

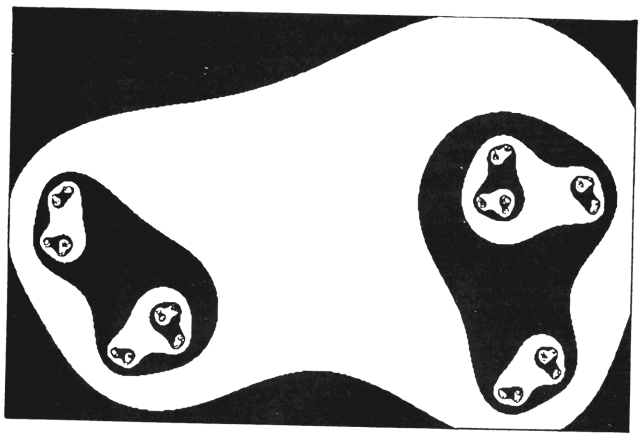
```
/************************************/
/* --- Calculate Julia sub-sets --- */
/*                                  */
/*      Written by Cliff Wootton    */
/*                                  */
/*      LightSpeed C compatible     */
/*                                  */
/************************************/

/**********************************/
/* --- Start of main program --- */
/**********************************/
       main ()
            {

/* --- Declare local variables --- */
       int   scr_width;
       int   scr_height;
       int   h_pos;
       int   v_pos;
       int   pix_col;

       float xmin;
       float ymin;
       float xdelta;
       float ydelta;
       float xold;
       float yold;
       float xnew;
       float ynew;
       float threshold;
       float const1;
       float const2;

/* --- Initialise machine specific variables --- */
       scr_width  = 511;
       scr_height = 341;

/* --- Initialise fractal domain variables --- */
       xmin       =  -1.15;
       ymin       =   1.00;
       xdelta     =   0.50;
       ydelta     =   0.45;
       threshold  =  25.00;
       const1     =   1.00;
       const2     =   1.00;

/* --- Initialise the working screen --- */
       init_window();

/* --- Start: Loop through each vertical line of the display --- */
       for (v_pos=1;
            v_pos<scr_height;
            v_pos++)
           {

/* ------ Start: Loop through each horizontal column of the current line --- */
           for (h_pos=1;
                h_pos<scr_width;
                h_pos++)
               {
```

Variable	Example 1	Example 2	Example 3
xmin	-1.50	-1.15	-1.15
ymin	-1.50	0.60	1.00
xdelta	3.00	1.45	0.50
ydelta	3.00	0.90	0.45
threshold	25.00	25.00	25.00
const1	1.00	1.00	1.00
const2	1.00	1.00	1.00

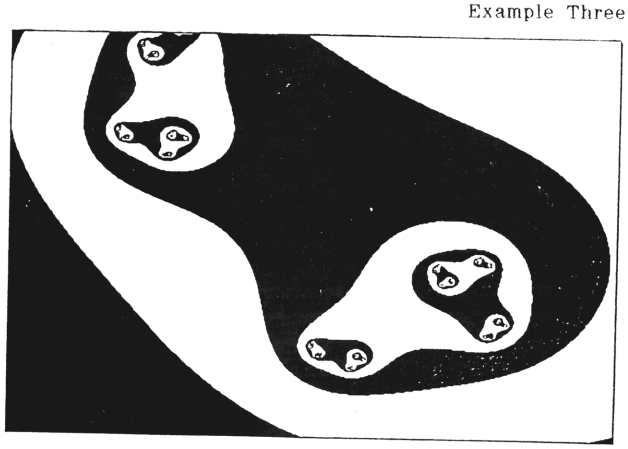
Table 1: Example input values

```
/* --------- Seed the iterative equation with initial values --- */
             xold = xmin+(h_pos*xdelta/(scr_width-1));
             yold = ymin+(v_pos*ydelta/(scr_height-1));

/* --------- Start: Loop for iterative equation until it decays --- */
             for (pix_col=0;
                  (xold*xold+yold*yold)<threshold;
                  pix_col++)
                 {

/* ------------ Iterative section --- */
                xnew = xold*xold-yold*yold+const1;
                ynew = 2*xold*yold+const2;
                xold = xnew;
                yold = ynew;

/* --------- End: Loop for iterative equation until it decays --- */
             }

/* --------- Draw point if necessary --- */
             draw_spot(h_pos,v_pos,pix_col);

/* ------ End: Loop through each horizontal column of the current line --- */
          }

/* --- End: Loop through each vertical line of the display --- */
       }

/* --- Pause at end of picture generation --- */
       pause();

/* --- End of main program --- */
       }

/*******************************************/
/* --- Initialise working display area --- */
/*******************************************/

init_window()
  {
   long port;
   Rect rect;

   InitGraf(&thePort);
   InitWindows();
   GetWMgrPort(&port);
   SetPort(port);
   SetRect(&rect, 0, 0, 512, 342);
   ClipRect(&rect);
   FillRect(&rect, white);
   HideCursor();
   PenSize(1,1);
   }


/************************************************/
/* --- Handle operator pause at end of run --- */
/************************************************/

pause()
  {
   DialogPtr dp;
   int dummy;
   dp = GetNewDialog(1, 0, -1);
   ModalDialog(0, &dummy);
   DisposDialog(dp);
   }


/*********************************************/
/* --- Draw a dot at the correct position --- */
/*********************************************/

draw_spot(x,y,c)
int x;
int y;
int c;
  {
   if (c/2 == (c-1)/2)
     {
      MoveTo(x,y);
      Line(0,0);
      }
   }

/*********************************/
/* --- End of program listing --- */
/*********************************/
```

Example Two



Example Three



The Consequences of Finite Arithmetic

W.E. Thomson.

Simon Goodwin ('Fractals, Maths & Graphics'– Issue 0) produced in parallel two versions of a Barry Martin Drawing, one using a square root and the other raising to the power one half, and found that they differed.

I suggest that this is one of the consequences of the finite arithmetic that is inevitable in any digital computing. With infinitely precise arithmetic, the two methods are bound to be the same, but in practice, slight differences in the algorithms mean slight differences in the results.

A similar result occurred with me, using the same program to produce a drawing, with a square root function, realised in two ways: the first, in Spectrum Basic, and the second in Hi-Soft Spectrum Pascal. For the same parameters (the a,b,c of Goodwin's paper) the results were different: there was a general resemblance but many differences in points of detail. This might be due to different square-root algorithms, but I put it down to a difference in arithmetic precision: the Basic's floating-point representation has a 32-bit mantissa but the Pascal's representation has only 23 bits.

The effect of finite arithmetic is more readily seen with a simpler system, the well known single-variable system defined by the recurrence relation

$$x_{n+1} = Ax_n(1 - x_n)$$

x_{n+1} will equal x_n if $x_n = (A-1)/A$, so in this program, one would expect the x of Line 50 to remain constant with the same value as at Line 20.

```
10 INPUT A: PRINT A        3.44
20 LET x=(A-1)/A
30 PLOT 0,176*x
40 FOR n=1 TO 255
50 LET x=A*x*(1-x)
60 PLOT n,176*x
70 NEXT n
```

But, for A=3.44, it does not remain constant, as the print-out shows.

A full discussion of this would take too long, so I summarize by saying that theory shows that there are many stable states, each valid for a certain range of A; the state where $x_{n+1} = x_n = (A-1)/A$ is stable only for $1 < A < 3$. Stable or not, infinitely precise arithmetic in Line 50 would maintain x at the value of Line 20; it is the rounding-off errors in finite arithmetic lead to small then ever-increasing departures from the value of Line 20, until the stable state appropriate to the value of A is reached. It is instructive to alter Line 50 so as to

vary these errors, either by rounding off to, say, 2 or 3 decimal places, or by adding a small (pseudo)random term.

A Barry Martin Drawing is produced in several phases, often with nothing apparently happening, until suddenly there is an eruption into a new phase. I suggest that these phases are stable, or quasi-stable states, and that the 'random' effects of rounding off in the arithmetic (which may differ in different computer languages) trigger the changes from one phase to the next.

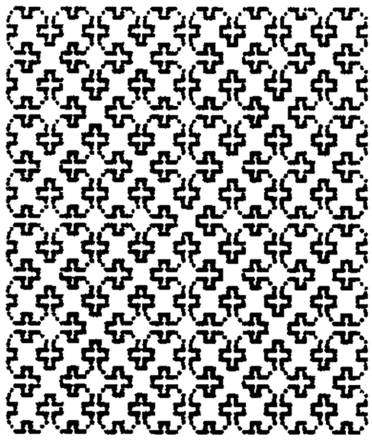
A SPACE-FILLING CURVE

by W E Thomson

The program below draws an approximation to a monster space-filling curve, as illustrated for n=4; it can be regarded as an elaboration of the Sierspinski Curve.

The language is Spectrum/BetaBasic which means that 'DRAW x,y' draws a straight line from the current point (cx,cy) to the point (cx+x,cy+y) and sets cx=cx+x, cy=cy+y. The variables p,q,r are chosen to fit the coordinates available; the curve lies within the square bounded by the lines x=q,x=q+3p,y=r, y=r+3p, where p=2↑m, m integral; n should not exceed m. Line 10 of the program specifies p,q,r suitable for a screen with maximum x-coordinate 255 and maximum y-coordinate 175.

```
 10 LET p=32,q=80,r=40
 20 INPUT n: PRINT n
 30 LET d=p/2↑n
 40 PLOT q+d/2,r+3*d/2
 50 tom n,0,d,d,0
 60 DRAW d,d
 70 tom n,d,0,0,-d
 80 DRAW d,-d
 90 tom n,0,-d,-d,0
100 DRAW -d,-d
110 tom n,-d,0,0,d
120 DRAW -d,d
130 STOP

200 DEF PROC tom n,s,t,u,v
210   IF n=1 THEN
        DRAW s,t:
        DRAW u,v:
        DRAW s,t:
        DRAW -u,-v:
        DRAW s,t:
        END PROC
220   tom n-1,s,t,u,v
230   DRAW s+t,u+v
240   tom n-1,t,-s,v,-u
250   DRAW s,t
260   tom n-1,-t,s,-v,u
270   DRAW -(u+v),s+t
280   tom n-1,s,t,u,v
290 END PROC
```



Transputer Fractal Computation

The Inmos transputer is ideal for fractal computation, and the generation and display of the Mandelbrot set has become a standard way to demonstrate the benefits of parallel processing using transputers. This is probably because it is very easy, once you have a Mandelbrot graphics program running on one transputer, to "parallelise" the program, and run it on any number of transputers. The performance increase is virtually linear, as more transputers are added, which is another reason why this demonstration is so popular.

Because the transputer has 2K (in the case of the earlier T414) or 4K (with the T800 floating point transputer) of very fast, on-chip RAM, you don't even need any external RAM on your transputers, if the program is coded in occam (or assembly language), and Inmos have built a "Mandelbrot engine" out of reject devices (the external RAM interface was faulty), with about 400 transputers, that generates the images faster than they could be taken from a hard disk.

Although I run my own business, developing and selling transputer-based systems, I can't afford to have too much expensive hardware lying around, so for my own use I have a system consisting of a single 1 Mbyte transputer module on a PC interface card for development, with four 1 Mbyte transputer modules on a second PC card, with the EGA graphics on my PC for display. I am working on a transputer-based graphics system, which will have a T800 and one of the new Inmos G300 graphics controllers, with 1 Mbyte of ordinary DRAM, and 1 Mbyte of VRAM. Display will be via a Multi-Synch monitor.

A few members of QUANTA, the QL users group, have my transputer modules interfaced to their QLs, and one of them, Jim Gilmour, has got a Mandelbrot program running on the transputer, using the QL for display. He sent me a copy of the program, which is written in C, and it took 1 min. 47 secs. to display the complete Mandelbrot set image, running on a 20 MHz T414 with 1 Mbyte of RAM.

I know someone who has built a parallel processing system using eight Z80s, just to perform Mandelbrot set calculations, with a homebrew TRS-80 - compatible system with a colour display for I/O. I've given him some 68008s to play with, so he can speed it up. A single transputer will run rings round this system of course, but it cost him next to nothing to build.

Leon Heller

PC Mandelbrot Public Domain

Mr John Marriage has advised us that a public domain disk XEM014, which produces the Mandelbrot Set, is available for the PC from the PC Independent User Group, PO Box 55, Sevenoaks, Kent, TN13 1AQ. He says that the group offers a huge catalogue of software including much which is of very high standard, a substantial magazine, and a useful helpline. They charge £3 per 5.25" disk, and £4 per 3.5" disk, plus £2 per order postage, plus £1 per disk for non-members.

Day's Squares on Tape

Mr Nick Day has made his program available on tape for any reader of issue 1 who would prefer it this way. He will send you a BBC "B" tape for £4. Mr Nick Day, 10, Nourse Close, Cheltenham, Glos GL3 0NQ.

Martin's Mapping on Macintosh

Mr Adam McLean is offering a Macintosh Application that runs quickly producing Martin's Mappings for many different functions. It is free to anyone who sends a blank disk and return postage. Mr Adam McLean, *The Hermetic Journal* PO Box 375, Headington, Oxford, OX3 8PW.