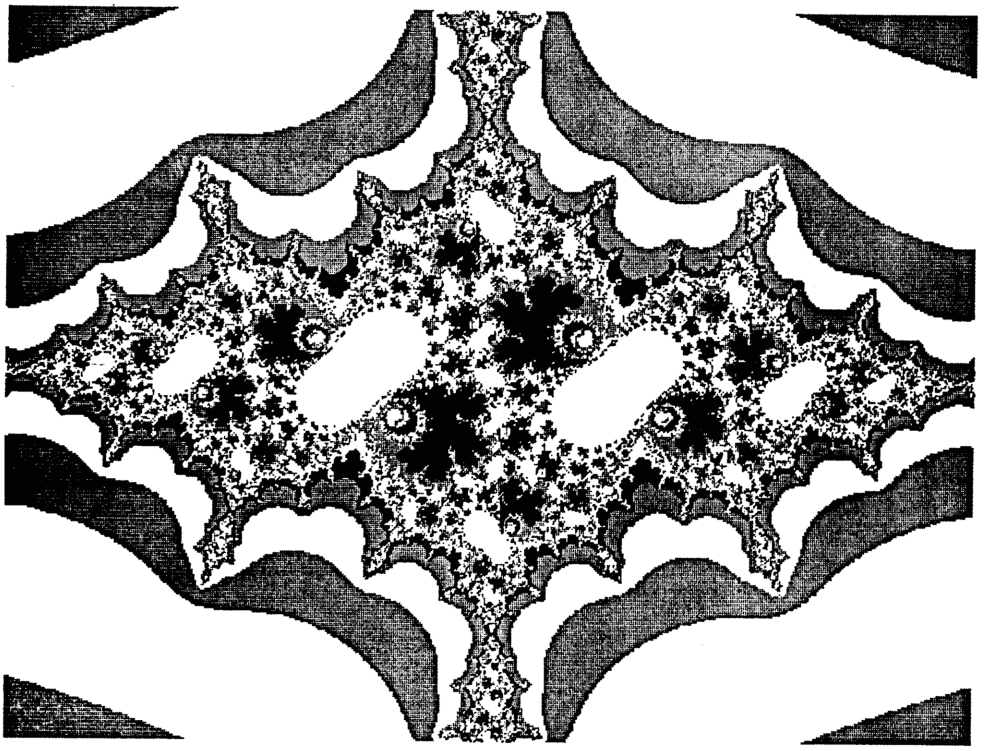
Fractal Report

Issue 3

wright.jul



Larry Cobb entry (mono version) Steve Wright

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.

JULIA SETS. by Ian Entwistle
Part 1.: Quaterions

The Brougham bridge on the Royal Canal,Dublin bears a plaque commemorating the discovery of "quaterions" in 1843 by William Rowan Hamilton (1805-65).These 4D equivalents of a complex number take the form a+bi+cj+dk where a,b,c,d are real numbers and i,j,k are imaginary numbers.Hamilton discovered that the following equations relate the "imaginary"components of the quaterions,so that ordinairy algebraic rules can be used to obtain a third quaterion from two others.ij=k,jk=i,ji=-k,kj=-i,ik=-ij.Thus the quaterion Q=a+bi+cj+dk can be squared to obtain a new function, :F(Q,q): Q→Q^2+q . Utilising Hamilton's algebra expressions for Qreal and Qimaginary can be obtained.For the sake of brevity and for those readers whose algebra is rusty the vale of Q^2 is equal to a^2-b^2-c^2-d^2+2bi+2acj+2adk.

3D fractal images from quaterion iterations have been studied(see Norton,A.Generation and display of geometric fractals in 3D,Comput.Graph.,Vol 16(1982)pp61-67) and beautiful illustrations of Q^2+q fractals have more recently been reported (Pickover,C.Visualisation of quaterion slices,Image and vision computing,Vol6(4),1988,pp235-236).the following is my own interpretation of the described method.

In the complex plane qa,qc the Mandelbrot set quaterion map is visually the same as for F(Z,c):Z→Z^2+c in the complex plane c.The Julia sets however are very different and well worth generarating even though they require more CPU time.Most Fractal Report subscribers will already have generated Julia set maps from Z^2+c. Very few alterations to the listing used is required to obtain quaterion Julia sets.Separation of Q^2+q into real and imaginary parts gives the equations (1),(2),(3),(4).

Qreal=a^2-b^2-c^2-d^2+qa (1)
Qimag.i=2ab+qb (2)
Qimag.j=2ac+qc (3)
Qimag.k=2ad+qd (4)

The iteration loop of the Julia set listing now needs a few extra variables and inputs are required for the constants qa,qb,qc,qd.Since the output will be a 2D image or a slice showing the inside of the 4D quaterion then only two of the equations,one real and one imaginary can be iterated using initial values corresponding to the pixels on the map.Thus iteration with the equations (1) and (3) using the inital values for a 2D Julia set and fixed initial values (try 0.05,0.05) for equations (2) and (4) will give a quaterion slice.The values chosen for qa and qc can be any values within the range used for Z plane Julia sets of Z^2+c.For example compare Fig.1 q(-.194,.01,.6557,.01) where the values 0.01 correspond to qb,qd for the uniterated planes.These values can be altered to look at other slices.The iteration loop of the listing will thus look something like List (1)in Basic .Other control loops are frequently used but on many computers the For-Next loop is faster.

List(1)

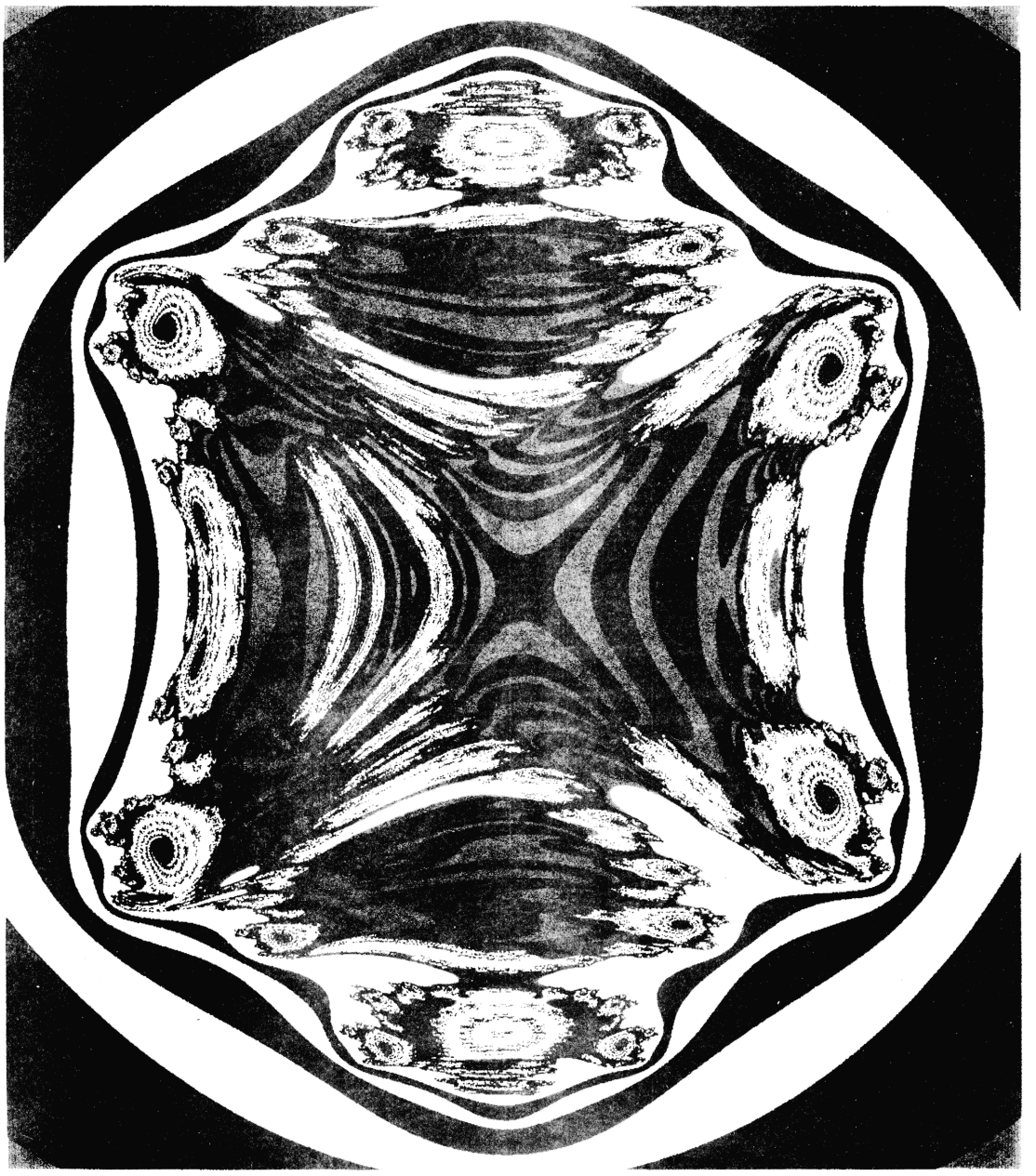
```
340  FOR TI% =1 TO IT%
350  X1X1=X1*X1:Y1Y1=Y1*Y1:Z1Z1=Z1*Z1:Z2Z2=Z2*Z2
360  X1Y1=X1*Y1:X1Z1=X1*Z1:X1Z2=X1*Z2
370  X1=X1X1-Z2Z2-Z1Z1-Y1Y1+qreal
380  Y1=(X1Y1+X1Y1)+qimagy
390  Z1=(X1Z1+X1Z1)+qimagz1
400  Z2=(X1Z2+X1Z2)+qimagz2
410  IF X1X1+Z1Z1+Y1Y1+Z2Z2 >4 THEN GOTO 450
420  NEXT
430  :REM PLOT OR PRINT PIXEL IN COLOUR CORRESPONDING TO VALUE OF TI%
```

TI%=Iteration value ,IT%=Maximum value of TI%
X1,Y1,Z1,Z2 are the variables corresponding to a,b,c,d
Note that ‖Q‖*‖Q‖=a^2+b^2+c^2+d^2

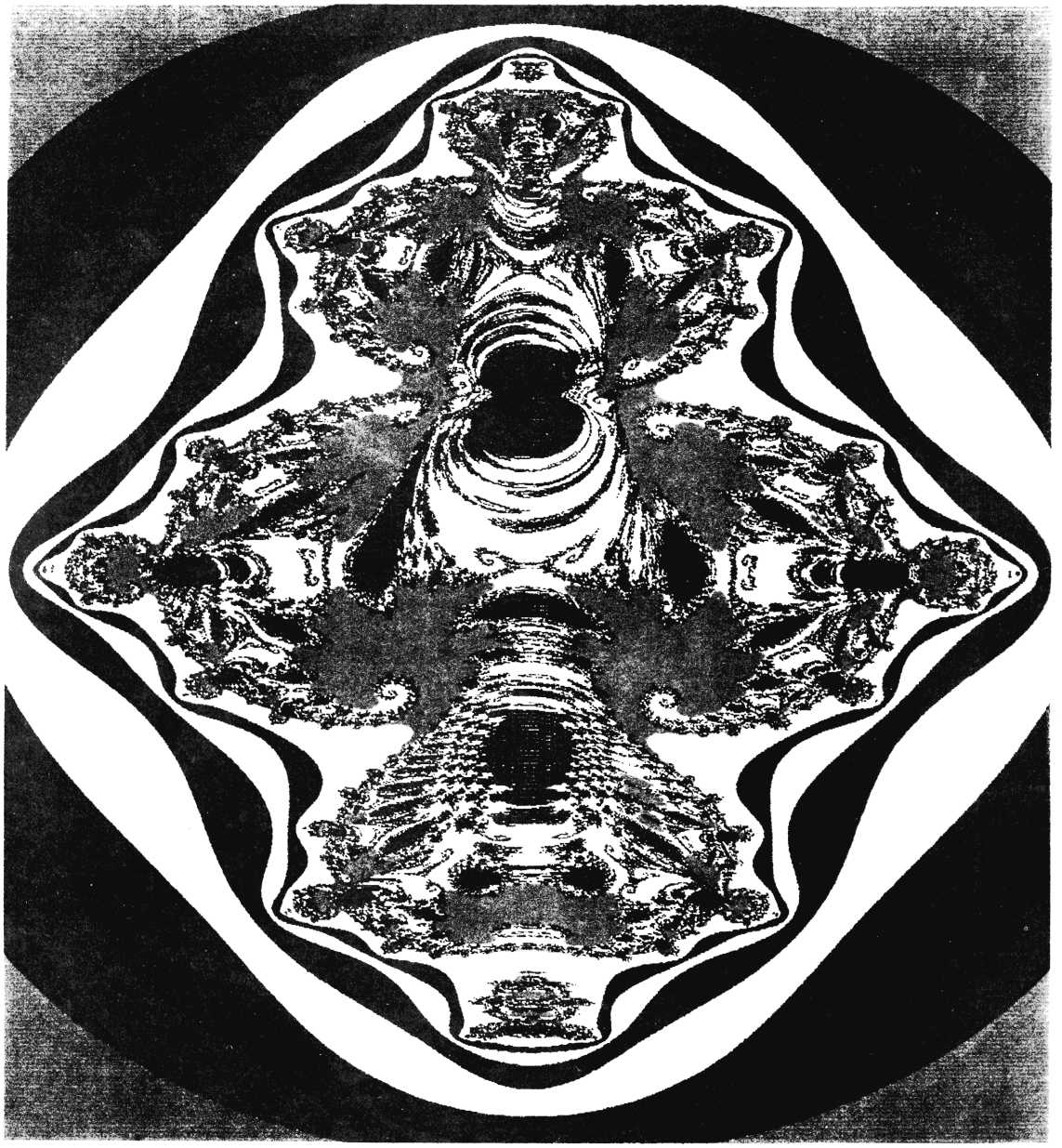
The test for "boundedness " ‖Q‖^2 >4 is sensitive to the value of IT% .The Figs. 1 to 3 were obtained using a value of 150 but quite attractive variations in the patterns are generated when IT%<150 .(Try IT%=64) In all the Figs. which differ quite markedly from the comparable Z^2+c Julia sets(see Beauty of Fractals by Peitgen and Saupe) with the the same c plane constants ,the inverse symmetry of Julia sets is absent.

The larger number of multiplications involved in quaterion iterations means that more CPU time is needed than for the Z plane Julia sets.Some interesting fractal plots can be obtained more readily by removing some of the multiplications .Put Qimag.=2a(b+c+d) and sum the imaginary values of q.For those enthusiasts with access to 32 bit machines try and generate the octonium equivalent Julia sets.Note that the relationships (mn)k=m(nk) and mn=nm are not valid for octonium multiplcation.

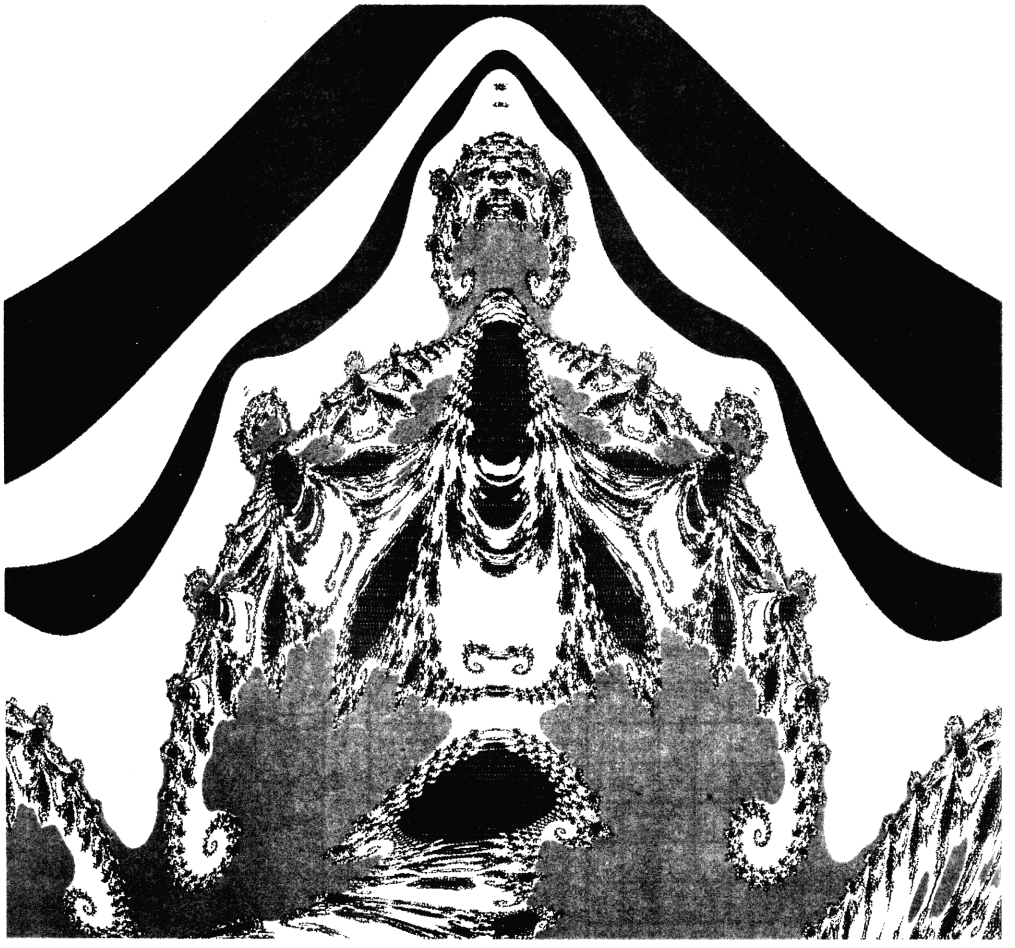
In the above article please read "quaternion" for "quaterion".



(1) 1280x2200 pixels,150 iterations,boundaries -0.9,0.9,-1.1,1.1



(2) 1080x1500 pixels,150 iterations,boundaries,-1.6,1.6,-0.9,0.9



(3) 1280x1800 pixels,150 iterations,boundaries,-0.7,0.7,-0.9,-0.15

Faking Planets
The logistic twist map on a sphere
by
Mark Datko
July 1989

Given the editorial challenge of planet generation, I would like to propose a method which on experimentation seems to give reasonable results without too much computational effort.

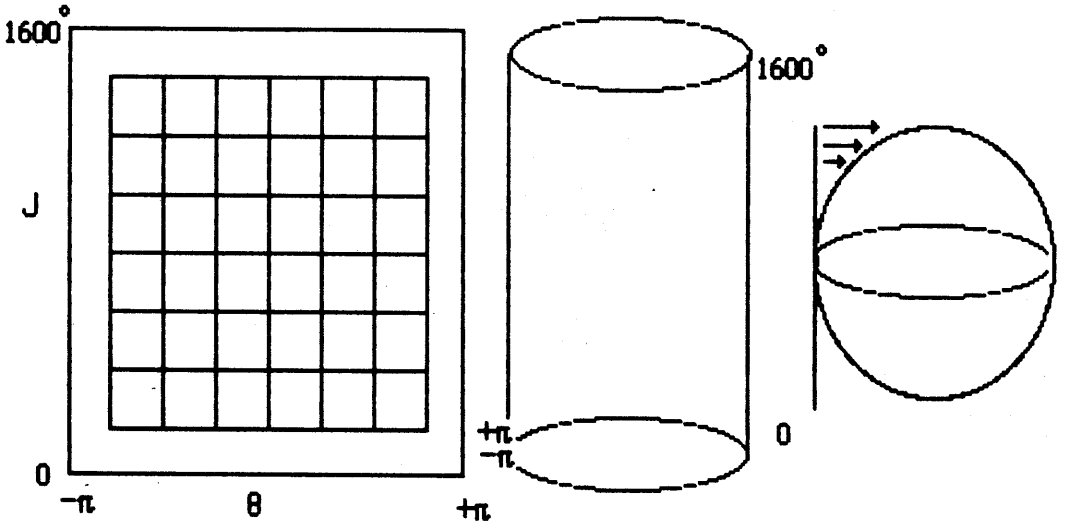
The trick is to use the logistic twist map :

$$J_{n+1} = J_n - K \sin \theta_n$$

$$\theta_{n+1} = \theta_n + J_{n+1} - \alpha J^2_{n+1}$$

which arises in the context of topological reconnection in the study of Hamiltonian systems.

The mapping can be simply be regarded as moving points on a cylinder, and if one wraps the cylinder onto a sphere one obtains a passable imitation of a Jovian style planet with "features" appearing in the swirling atmosphere and on the surface below depending on the choice of K and α.



Iterate map using grid for starting points

Identify −π and +π and "wrap" onto sphere

```
100 REMark QL specifics
110 REMark ------------
120 DEFine PROCedure sa
130 SAVE_O flp1_Jupiter                    Programs by John de Rivaz based on
140 SAVE_O flp2_Jupiter                    PASCAL original by Mark Datko.
150 END DEFine
160 DEFine PROCedure li
170 WINDOW £2, 512,200,0,0
180 END DEFine
190 MODE 4:WINDOW 512,255,0,0:PAPER 0:CLS
200 :
210 LET k=-.23:a=3.6E-2  : REMark change for different planets
220 LET numits=300       : REMark change for density of plot
230 LET mylat=30         : REMark change for viewpoint (degrees)
240 LET mylong=180
250 radius=127           : REMark adjust to fill screen
260 LET aspect_ratio=3/2 : REMark ql specific for "BLOCK"
270 LET piby28=PI/28:pipt8=PI+.8:twopi=PI*2:twopiby10=twopi/10:conrad=1.74533E-2
280 LET mylat=mylat*conrad :SINmylat=SIN(mylat):COSmylat=COS(mylat)
290 LET mylong=mylong*conrad
300 :
310 FOR x_start=3 TO 24 STEP 3
320   LET colour= 2*(1+(x_start/3) MOD 3)
330   FOR y_start=twopiby10 TO twopiby10*8 STEP twopiby10
340     LET x=x_start:y=y_start
350     FOR iteration=1 TO numits
360       LET x=x-k*SIN(y)
370       LET y=y+x*(1-a*x)
380       IF y>twopi THEN LET y=y-twopi:GO TO 380
390       IF y<0     THEN LET y=y+twopi:GO TO 390
400       LET lat=(x-14)*piby28:long=y+pipt8
410       LET COSlat=COS(lat):SINlat=SIN(lat):long=long-mylong:SINlong=SIN(long)
420       LET COSlongCOSlat=COS(long)*COSlat
430       IF COSlongCOSlat*COSmylat+SINlat*SINmylat<0
440         BLOCK 1,1,radius*(SINlong*COSlat+1)*aspect_ratio,radius*(1+COSlongCO
Slat*SINmylat-SINlat*COSmylat),colour
450       END IF
460     END FOR iteration
470   END FOR y_start
480 END FOR x_start
```

QL Program

See note below re line 480

```
100 BRIGHT 1: BORDER 0: PAPER 0 :
CLS

220 LET k=0.23: LET a=.0036
230 REM change for different
planets

240 LET numits=30
250 REM change for plot density

260 LET mylat=30
270 LET mylong=90

280 LET radius=87
290 LET radiusby2=radius/2
300 LET piby28=PI/28
310 LET pipt8=PI+.8
320 LET twopi=PI*2
330 LET twopiby10=twopi/10
340 LET conrad=.0174533
350 LET mylat=mylat*conrad
360 LET SINmylat=SIN mylat
370 LET COSmylat=COS mylat
380 LET mylong=mylong*conrad

390 REM the works!
400 FOR h=1 TO 8
410 INK 6*h/8
420 FOR v=twopiby10 TO
twopiby10*8 STEP twopiby10
430 LET x=h*3: LET y=v
440 FOR i=1 TO numits
450 LET x=x-k*SIN y
460 LET y=y+x*(1-a*x)
470 IF y>twopi THEN  LET y=y-
twopi: GO TO 470
480 IF y<0 THEN  LET y=t+twopi:
GO TO 480
490 LET lat=(x-14)*piby28
500 LET long=y+pipt8
510 LET COSlat=COS lat
520 LET SINlat=SIN lat
530 LET long=long-mylong
540 LET SINlong=SIN long
550 LET COSlongCOSlat=(COS long)
*COSlat
560 IF COSlongCOSlat*COSmylat+
SINlat*SINmylat<0 THEN  PLOT
radiusby2+radius*(SINlong*COSlat
+1),radius*(1+COSlongCOSlat*
SINmylat-SINlat*COSmylat)
570 NEXT i
580 NEXT v
590 NEXT h
600 STOP
```

Spectrum Program

Note - the Macintosh PASCAL listing runs to three pages, and as relatively few readers have the Macintosh, this is available upon receipt of an SAE.

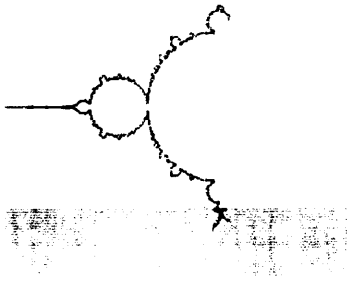
Scanning note: LINE 480 is probably wrong as variable t is not set anywhere- 480 should probably read y=y+twopi:

Project: Tightrope
Warp Speed Mandelbrot Sets

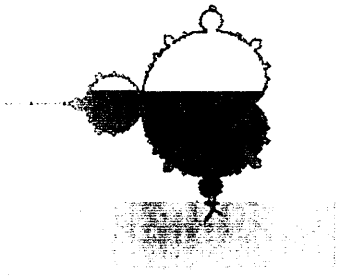
by Mark McCall

While taking a class this spring in fractal graphics at San Francisco State University, I quickly discovered how important speed is to almost any system when generating these time consuming images --- even a CRAY! Some of the largest stumbling blocks to Mandelbrot set generation are the points actually inside the set, generally condemned to lay in some sort of a time consuming calculatory limbo. If these points could somehow be predetermined, without an abundance of heavy calculations, images which include these regions could be cut down to as much or less than half their generation time. And with information which establishes the Mandelbrot set to be connected [1], this task can be accomplished.

Based on the idea that the Mandelbrot set is connected, if one knew the perimeter of the set, the interior could be quickly filled-in so that the heavy computations involved in finding points within the set could be avoided. It would be like filling an intricate glass with water. Finding the perimeter of the M-set is not as complicated as it may seem at first. In fact, it can be considered as a kind of game. The object of the game is to traverse the edge of the set, continuing all the way around the "island" it encompasses until returning back to the starting point. It is most advantageous to think of following the edge with a "traveler," a point which resembles the actions of a person hugging the wall of an enclosed region, knowing eventually the wall will lead him back to the beginning. This can be done by first determining how the traveler will traverse the boundary. In this case I will have it traverse **clockwise** about the island wall's interior. Traveling in this way, the boundary of the M-set must always be on its left. If at any time the boundary disappears from its immediate left, the traveler must re-establish contact with the wall, turning and moving to its left until the wall is again discovered, and then continuing onward about the set. If a wall is discovered in front of the traveler, while there is also a point outside the M-set on its immediate left, it must turn to its right. My traversal algorithm is as follows:



Stage I
Traversing the M-set boundary
(Note the overall complex plane traversal was west to east ⇨, south to north ⇧.)



Stage II
Filling the M-set

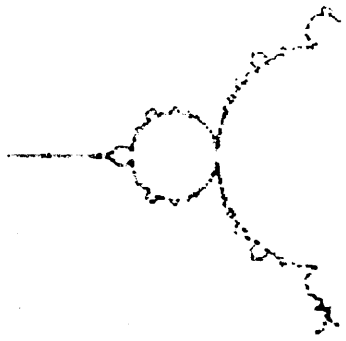
```
PROCEDURE Tightrope
{ start immediately upon entry of M-set }

orientation - face east; { IMPORTANT --- see following paragraph }
                         {                  for explanation.      }
REPEAT
  check and update dwell (iterations) of point to traveler's immed. left;
  IF (left point is NOT OUTSIDE the M-set)
    BEGIN { if }
    orientation - face to left ; { based on present orientation }
                                 { (- north, south, east, or west) }
    move forward one point;
    END { if }
  ELSE
    BEGIN { else }
    check and update dwell of point immediately forward;
    IF (forward point is INSIDE the M-set)
      move forward one point;
    ELSE
      orientation - face right; { based on present orientation }
                                { (- north, south, east, or west) }
    END { else }
UNTIL ( position - initial position );
```

Final image
Scanning note: The magazine image was black on dark grey and not very clear. The outer dark grey has been removed in this illustration.

The initial orientation of the traveler is crucial. If not blocked on all sides, the traveler must move on the very first entry of the REPEAT loop, otherwise the loop will terminate prematurely. The above orientation is based

on the assumption that the entire complex plane has been traversed line by line from **west to east ⇨, south to north ⇧**. In this way, if the algorithm is entered immediately upon discovery of a previously unvisited point within the M-set, it is known that the points immediately south and to the west of this spot are outside the M-set. Thus, if not completely blocked, an orientation of east will allow the traveler to move upon first entry into the loop, as required. **Other traversal patterns of the complex plane may require different initial orientations.**

The entire program will require a way to keep track of points being visited, such as a two dimensional array representing the portion of the complex plane being investigated. For the algorithm, itself, two conditions must be known of each point: (1) has this point been visited? and (2) is this point inside the M-set? (e.g. use a 3 value variable: unvisited - 0, inside set - 1, outside - 2) After the Tightrope algorithm has finished, and before the overall traversal is continued, this information will allow the interior of the M-set to be filled-in. Due to the nature of the algorithm, the outline will consist of a series of points within the M-set, designating its edge, surrounded by a series of points outside the M-set. Thus, unvisited points on a line between a point within the M-set and a point outside the M-set can be filled as being members of the set.



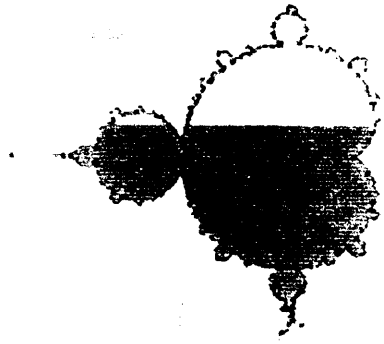
Example of Stage I incorporating a distance estimation routine with the Tightrope algorithm

NOTE:
Under certain circumstances the created outer line of points which is not in the M-set will be incomplete to the east of the point of origin, therefore DO NOT attempt to fill this row.
(IMPORTANT: this is assuming the filling is being done in the SAME fashion as the overall traversal of the complex plane was done before entering the M-set and initiating the traversal algorithm; i.e. west to east, south to north.)

All points outside the boundary of the actual complex plane region being investigated should be considered walls, points OUTSIDE the M-set. This way, if the Mandelbrot set is not entirely within the image window, the edge of the window will be treated as the boundary of the set.

Also, concerning the Tightrope procedure, please note the difference between the complex plane's global coordinates of north, south, east & west, and the traveler's own local coordinates of left, right & forward.

The final code for this algorithm is tight and relatively fast. By incorporating it with other speed algorithms such as a (exterior) distance estimation routine, the resulting Mandelbrot generation may require observers to wear seat belts!
(It should also be possible to use this method with other connected sets.)



Example of Stage II incorporating a distance estimation routine with the Tightrope algorithm

For a print-out of my code (written in PASCAL), send $4.00 to:
Mark McCall
225 Shevelin Rd.
Novato, CA 94947
U.S.A.

The address of the university follows:
San Francisco State University
(Computer Science Department)
1600 Holloway Avenue
San Francisco, CA 94132
U.S.A.

Reference:

[1] **Douady, A., Hubbard, J.H.**
Iteration des polynomes quadratiques complexes.
Referred to by Heinz-Otto Peitgen
in *The Science of Fractal Images.*
New York: Springer-Verlag, 1988.

MORE AFFINE TRANSFORMATIONS - AND SOME NOT SO AFFINE

by KEITH WOOD

The first article showed how to manipulate a transform and put it to work developing a fractal image. There are limitations. From a descriptive point of view, if a transformation reduces the image (a contractive transformation) then the repeated use will result in images smaller and smaller. To complete an image, there needs to be a way of getting back to the beginning, so that the whole image at full size gets fleshed out.

The examples quoted last month achieve this by adding a constant to an otherwise reducing image. Thus the three half size images of the Sierpinski triangle are placed on the screen in the right place to make up the whole image. It follows from this that each half size image is made up of three quarter size images, and so-on. While this is what the Sierpinski triangle is, it is not what other triangles are. The square similarly had four equations generating four half size squares so placed as to construct the square. In principle, if an image can be made by superimposing smaller versions of itself, then that gives the clue to the equations required to generate the figure. The reductions and translations are all that is supplied to the programme, and it can happen that the programme interprets its directions on the basis of a different shape than the one supposed, so one doesn't always get what one wants! The programme produces the simplest, with the fewest sides, of the possibilities.

As an example, form an arrow from three smaller arrows: the two tails to the arrow are reduced and rotated from the main point, and the main point is a reduction of itself moved up so that the points coincide. This last transformation has a high probability so that a sufficient number of successive transformations can occur to fill the point to the tip. Note particularly that rotating the main point to form one of the tails is a randomly selected transformation, and can be selected twice or more times in succession, which will produce more spikes in unwanted places. It works in this case because the reduction involved at each rotation brings these second and third etc. order repeats within the boundary of the main figure.

ARROW

m	a	b	c	d	e	f	p
1	.766	0	0	.766	.234	0	.6
2	-.5	.42	-.42	-.5	.234	-.04	.2
3	-.5	-.42	.42	-.5	.234	.04	.2

What other methods are available for "getting back to the beginning"? It will be seen that the method used above produces a solid or filled image, either totally as in the case of the square, or partially, as in the case of the Sierpinski triangle. The tree, however, consisted of a line drawing. This gives an immediate clue that something is different. The first two transformations decrease the image size to 0.6 of the original, while rotating it 45 degrees. One is to the left, the other to the right. The screen clearly demonstrates the 45 degree branching. At the same time the Y value is increased by 0.2. This is necessary to make the branches come from the main stem part way up. At the scale of the calculation the whole tree is less than 0.5 high. The third equation reduces the image to 0.1 of the original, 0.2 up. The whole tree image transformed this way makes the blob on the main stem out of which the branches grow, and it is repeated down the chain at sucessive branchings. The fourth and final equation takes the value of Y and halves it, while setting X to zero. This equation creates the main stem, and because of its low probability it is unlikely to be used many times in succession, which is why the main stem peters out into disconnected spots without actually reaching ground! It is this main stem which is transformed into the branches to create the tree. It follows that none of the branches quite connect to the stem they branch from. The blob on the stem helps to conceal this defect.

What causes the difference is that the "getting back" is not a transformation, except in a special sense. As the X coordinate is set to 0 it means that the transformed image is given zero width, so the start is not recognisable as a contraction from the whole. One can just as readily regard it as an equation generating a value using the Y as a seed. The equation makes a shape which is copied throughout the image, but which itself is not a copy of the image. There is no law which says that copies of the image should be kept in proportion, but the idea of a dissimilar start shape leads to a different class of subjects.

To demonstrate this, the following table uses the Y value as a seed to generate a line with both positive and negative X coordinates, which is then transformed successively up the tree to the top.

CHRISTMAS TREE:

m	a	b	c	d	e	f	p
1	.8	0	0	.8	0	.2	.75
2	0	.4	0	.1	0	.1	.1
3	0	-.4	0	.1	0	.1	.1
4	0	0	0	.3	0	0	.05

The first transformation replicates the branches, and has a high probability to plot points all the way to the top. Fewer points are needed at the top, but enough in the time allowed dictates the probability to use. On a higher resolution plot more iterations would be used, giving more chance, but needing more too. Depending on the plot the probability may need changing for different resolutions. The thing to do is to try it and adjust probabilities until the different areas of the figure fill out at roughly equivalent rates. The other three equations all generate straight lines from Y as a seed. Two do the left and right branches, and the third the trunk.

Some of the figures discussed (the tree, spiral, christmas tree, fern) generate a shape at the "beginning" of the subject which is replicated many times in generating the whole image. If the probability is too low the tip of the christmas tree, fern etc. will be omitted rather as though someone had torn it off. The value needs to be in the range 0.6 to 0.9. It follows that in any image of this type there can only be one replicating feature. One cannot design a replicating system to create a branch and a different replicating system to add brances to a tree. A fern works because a side lobe is the same shape as the whole leaf. Using the X or Y value as a seed in an equation gets around the problem and brings in a new class of images in which the feature replicated is independent of the shape of the whole figure. X and Y are not independent, using both produces echoes of the whole figure.

One important limiting factor is that a linear equation cannot generate a curve. Where a curve is required it is generated by successive rotations. Such a curve "uses up" most of the available probability, so that the resulting plot cannot itself be replicated other than as a by-product of the replication generating the curve (fern, spiral).

The system of using a pair of linear equations has been extensively developed with a view to computer generation of pictorial subjects. It is claimed that a data reduction of up to 10,000 to 1 can sometimes be achieved. An image is analysed and broken down into a collection of images such as the ones we have tried, and the tables of data (perhaps 100 to 300 all told) form the record of the image which can be regenerated just as the data given here provides images when processed. The original of perhaps 1024 x 1024 pixels is reduced from 1 megabyte (excluding colour and intensity information) to a kilobyte or so. Images by telephone line and from remote satellites would be sent much more rapidly. To regenerate images quickly (the ultimate aim is to do it in real time) parallel processing is involved, each processor doing one set of transformations and pooling the results. There are other ways, too.

For our purposes, we can break with such rigid disciplines and propose equations other than linear. On the basis that one can do anything with polynomials, a programme can be written for polynomials of any degree, but the object becomes self defeating in that process time is stretched out and our table of values becomes enormous and filled largely with zeros. As a compromise, for experimentation and to demonstrate what can be achieved, we will allow one polynomial equation pair, the rest being linear as before. The Basic listing which follows is such a routine, which has also been extended to allow up to 10 sets of equations.

Notice that the sets of data number one less than M; the polynomial pair of equations is written out in lines 500 and 510. There is no probability spelled out for this pair, the value implied is the difference between the sum of the linear probabilities and 1. If there are fewer than 10 pairs of equations and if the probabilities for them add up to 1 then the second order pair will be ignored. This routine will therefore replace the one given before.

LISTING WITH ROSE DATA

```
10 REM The listing for iterating a set of affine transformations which include one
20 REM polynomial pair. As the pair can have many constants
30 REM the simplest way is to write out the equations as a line of the programme.
40 REM A table of data would be time consuming to manipulate if it had provision
50 REM for many additional constants per pair of equations.
60 REM Up to 10 transformations can be handled here.
70 DIM A(10),B(10),C(10),D(10),E(10),F(10),P(10)
80 M = 6
90 DATA .5,.866,-.866,.5,0,0,.40
100 DATA .5,.289,-.289,.5,0,0,.11
110 DATA -1,0,0,1,0,0,.12
120 DATA -1,0,0,-1,0,0,.12
130 DATA 1,0,0,-1,0,0,.12
140 PT = 0
150 FOR J = 1 TO M - 1
160 READ A(J),B(J),C(J),D(J),E(J),F(J)
170 READ PK
180 PT = PT + PK
190 P(J) = PT
200 NEXT J
210 FOR J = M TO 10
220 P(J) = PT
230 NEXT J
240 REM set up for 192x180 plotting mode
250 GRAPH 1
260 XSCALE = 75
270 YSCALE = 90
280 XOFFSET = 95.5
290 YOFFSET = 89.5
300 X = 0
310 Y = 0
320 REM do 10000 iterations
330 FOR N = 1 TO 10000
340 PK = RND(1)
350 IF PK<=P(1) THEN K=1 ELSE IF PK<=P(2) THEN K=2 ELSE IF PK<=P(3) THEN K=3 ELSE 460
360 XNXT = A(K)*X + B(K)*Y + E(K)
370 YNXT = C(K)*X + D(K)*Y + F(K)
380 X = XNXT
390 Y = YNXT
400 IF N > 10 THEN PLOT X*XSCALE+XOFFSET,Y*YSCALE+YOFFSET,1
410 NEXT N
420 ? "Type CONT to continue"
430 STOP
440 TEXT
450 END
460 IF PK<=P(4) THEN K=4 ELSE IF PK<=P(5) THEN K=5 ELSE IF PK<=P(6) THEN K=6 ELSE 480
470 GOTO 360
480 IF PK<=P(7) THEN K=7 ELSE IF PK<=P(8) THEN K=8 ELSE IF PK<=P(9) THEN K=9 ELSE 500
490 GOTO 360
500 XNXT = 0.125*(7 - X*X - X - X)
510 YNXT = 0.217*X*X - 0.144*X - 0.361
520 GOTO 380
```

LARCH

m	a	b	c	d	e	f	p
1	.8	0	0	.8	0	.2	.6
2	-1	0	0	1	0	0	.27
3	0	0	0	.5	0	0	.03
4	XNXT = -0.186*Y*Y - 0.654*Y + 0.830						
	YNXT = 0.457*Y*Y - 0.251*Y - 0.006						

THE BIG BANG

m	a	b	c	d	e	f	p
1	.966	.259	-.259	.966	0	0	.25
2	.966	-.259	.259	.966	0	0	.25
3	.8	0	0	-.8	0	0	.18
4	0	1	-1	0	0	0	.08
5	0	-1	1	0	0	0	.08
6	-1	0	0	-1	0	0	.08
7	XNXT = 0.1*X + 0.9						
	YNXT = 0.1*(1 - X*X)						

The Big Bang illustrates an important facet of the process. The random number generator has to be good. All computers, being built to behave perfectly rationally, use a pseudo random sequence in place of true random numbers. There are many of these, which differ in their performance. A truly random generator will always cover all possible points in a display given long enough, and the number of unfilled points will decrease with time. Running the big bang for 20,000 iterations appears to add no more points to the display from about 12,000 on, even though there are some noticeable gaps.

Finally, another pitfall is illustrated by STAR1. Run this, and it will fill in a couple of dozen points and then appear to stop. The reason is that the mechanism for generating a point on the horizontal line which is iterated round the other four lines plots points on those other lines which generate the same points again when used as seeds for the line generator. It simply plots the same points again and again. To overcome this problem a non-linearity can be added to the line generator with the use of the polynomial, as in STAR2. Both line generators are present in STAR2 and the original is reversed in sign to make the coverage more uniform.

STAR1

m	a	b	c	d	e	f	p
1	0	1.051	0	0	-.100	.309	.34
2	.309	.951	-.951	.309	0	0	.33
3	.309	-.951	.951	.309	0	0	.33

STAR2

m	a	b	c	d	e	f	p
1	.309	.951	-.951	.309	0	0	.3
2	.309	-.951	.951	.309	0	0	.3
3	0	-1.051	0	0	.100	.309	.2
4	XNXT = 1.902*Y*Y - 0.951						
	YNXT = 0.309						

That's the lot on this topic. Please send your favourite tables of coefficients to John for publication. We'd all like to see them. To plot several shapes on the same screen iterate one, and then GOTO the next, and so-on, repeating the programme as often as there are different shapes to plot.

Unfortunately, at the moment I have one computer which prints the screen but its Basic doesn't produce graphics, and another which produces graphics but doesn't print the screen. So you'll have to do the above for yourself!

Editorialette

This issue is being published concurrently with the next, in order to work off a backlog of articles before the autumn. This issue contains a nice selection of articles likely to interest the reader who wants to move past the basic sets and mappings, whereas the other issue of the pair has more than its usual share of basic material which has been requested by many.

I have had a query as to which is the most negative mini-Mandelbrot set along the negative real spike. Any mathematically minded reader who can provide a mathematical argument as to where this is, is invited to reply. It is my hunch that the spike extends to minus infinity, although it gets smaller and smaller and the mini-Mandelbrots also get smaller and smaller.

The Immortalist Society sent me a paper *Automated Development of Tarski's Geometry* by Art Quaife, director of Trans Time Inc., the cryonic suspension company. It looks interesting but is somewhat beyond my understanding - if anyone interested in computer automated mathematics would like a copy, then please let me know. A short appreciation may interest other readers.

More about Iteration Algorithms

Jules Verschueren
Binnenstraat 53
B - 3008 Veltem
Belgium

1. Pitfalls.

To display any pixel on a computer screen both the x- and y-coordinates have to be known, either in their absolute (eg. 0-639 and 0-199 for a PC with CGA card) or relative (window) position. A clear separation between these 2 coordinates is therefore necessary during each iteration step. Let's start from a general formula (actually the affine transformations) :

$$X(n+1) = aX(n) + bY(n) + e$$

$$Y(n+1) = cX(n) + dY(n) + f$$

where X and Y also can be more complex (eg. polynomial or a function) and a, d, b, c, e and f are eg. the reduction, rotation and translation factors.
Computers do obviously not recognise the subscripts n or n+1 and the translation into any computer language can create a problem. The solution is actually simple if we interpret and use NewX and NewY for respectively X(n+1) and Y(n+1) and take care not to forget to re-assign these new values to X and Y before the pixel is displayed onto the screen. The correct programme (in a GWBasic type language) for one transformation could look as follows :

```
1    FOR n = 1 to MaxPoints
2      NewX = aX + bY + e
3      NewY = cX + dY + f
4      X = NewX
5      Y = NewY
6      PSET (X, Y)
7    NEXT n
```

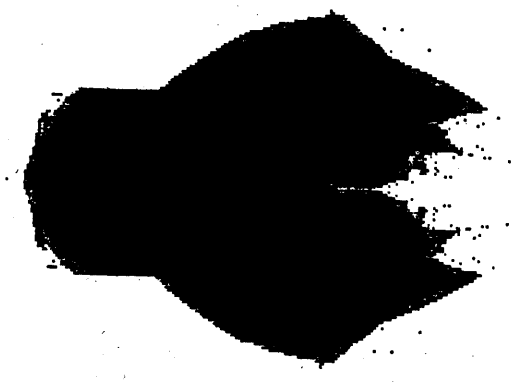
```
{ or alternative programme:
{   FOR n = 1 to MaxPoints
{     OldX = X
{     X = aX + bY + e
{     Y = cOldX + dY + f
{
{     PSET (X, Y)
{   NEXT n
```

However, we don't like too many new variables, especially when they are not strictly necessary...
So, we use Y = cX + dY + f in line 3 and delete line 5. Still correct!
The errors creep in when we also use X instead of NewX or when the new line 3 (Y = cX + dY + f) is placed before line 2 in the above program. The reason for the error is that a new value has already been assigned to X or Y when the old value still had to be used in the calculation of the remaining coordinate.

Although I'm sure that almost all of us know the above, this error actually still occurs quite often when we transform a formula or algorithm into a computer programme and this gives mostly rise to very severe headaches and frustrations since it disables you to reproduce any of the example figures.

Even the best programmers sometimes fall into this trap as shown by the resulting 'Mandelbrot set' images from 2 of the major articles in Fractal Report 1 :

Top right page 19 (J. Marriage)



Mid page 3 (Dietmar Saupe)

Although these sets have similar fractal properties, they obviously differ from the real Mandelbrot set (with apologies for the bad quality 1986 ZX-printing).

2. Colouring.

After having worked with several pure iterative (ie. iteration of a start point according to a formula like that of Martin, Devaney, Mira, Chernikov ...), mapping (eg. Henon, De Vogelaere, Helleman, Peitgen ...) and grid systems (eg. cyclic systems, but also to some degree the Mandelbrot and Julia like sets) I believe that one of the best ways to colour these fractals is to divide the maximum number of iterations by the total number of different colours that you can or would like to use. An equal number of consecutive pixels of each colour is then displayed and usually clear, separate regions (mostly in an expanding manner) develop. The total number of pixels required for a nice picture is mostly high, eg. 50,000.
In addition to the pleasing pictures that in most cases are obtained in this way, the method also reveals more about the underlying structure and evolution of the system. This information can be used by eg. physisists to find solutions to the instable orbits of elementary particles in particle accellerators or by astronomers to determine the orbits of asteroids (eg. Henon mappings: bright colours with a high no. indicate stable orbits while dark colours are mostly part of chaotic behaviour)...

Say we would like to use the 7 high intensity colours (ie. nos. 9 - 15 or blue to white) on our PC, then we proceed as follows (in a GWBasic like language):

```
ColorCount = MaxPoints \ 7 + 1          {\ is an integer division}
FOR i = 0 TO MaxPoints
  Kolor = i \ ColorCount + 9            {? in SuperBasic: INK i \ ...}
  (calculations according to formula to obtain new x,y)
  PSET (x, y), Kolor                    {? in SuperBasic: BLOCK ...
                                                    or POINT x,y }
NEXT i
```

Alternatively you can change the colour for each consecutive iteration by running through a number of colours as suggested by Simon Goodwin in Fractal Report 0. However, this probably only gives good distinct regions when the no. of stable orbits is some multiple of the no. of colours. Also, instead of using 3 statements to determine the actual colour, this can more elegantly, faster and shorter be programmed using modulo arithmetic :

```
above example :     Kolor = i MOD 7 + 9                  { nos. 9-15 }
Simon's line 155 :  colour% = colour% MOD 7 + 1          { nos. 1- 7 }
```

Pictures From Chaotic Dynamic Systems Without Attractors

John C. Topham

In the book 'The Science of Fractal Images', R.L. Devaney has written a chapter on fractal patterns arising in chaotic dynamical systems. He mentions a system discovered by M. Henon that has a 'strange attractor'. The system is shown below;

$$x_{n+1} = 1 + y_n - 1.4(x_n)^2$$

$$y_{n+1} = 0.3x_n$$

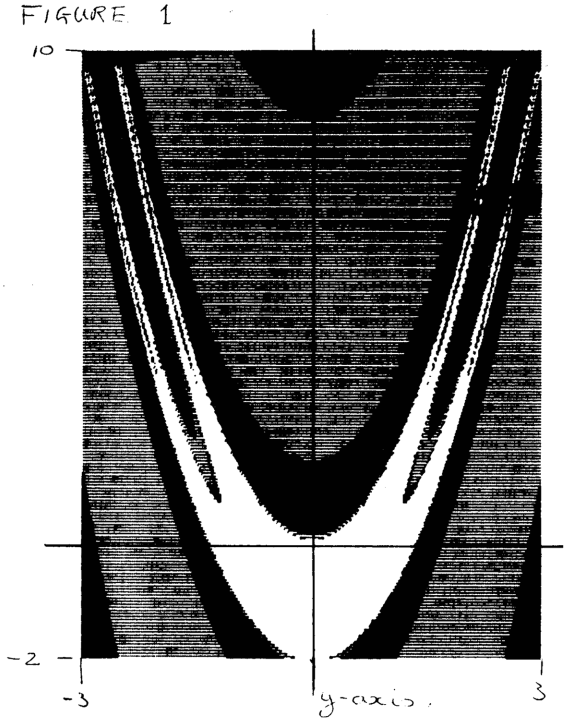
Now using the 'do-it-yourself' techinque for drawing Julia sets described at the end of the book 'The Beauty of Fractals', we substitute the operative equations in that technique with the ones above.
The resultant picture is shown in figure 1. This shows few fractal characteristics.
However, if one inserts the system shown below:

$$x_{n+1} = -1.5 - 0.5x_n + 2.4(y_n)^2$$

$$y_{n+1} = -0.45 + 0.5x_n$$

a picture shown in figure 2 and 3 emerges.

FIGURE 1



Unlike Henon's system the above equations do not converge to any attractor, strange or otherwise. All points in the plane studied rapidly zoom off to infinity. However, if one limits the number of points generated, studies the rate at which they reach a certain threshold, and assigns a colour to each pixel according to this rate the fractal pattern then results.

By altering the coefficients of the above system one can get various interesting patterns.

One interesting aspect of patterns produced by the above system is the type of optical disortion that seems to be occurring to the main fractal elements. This is something that is not often seen in other fractal patterns.

A simple basic program as used on the Sinclair QL is given for people to experiment if they wish.

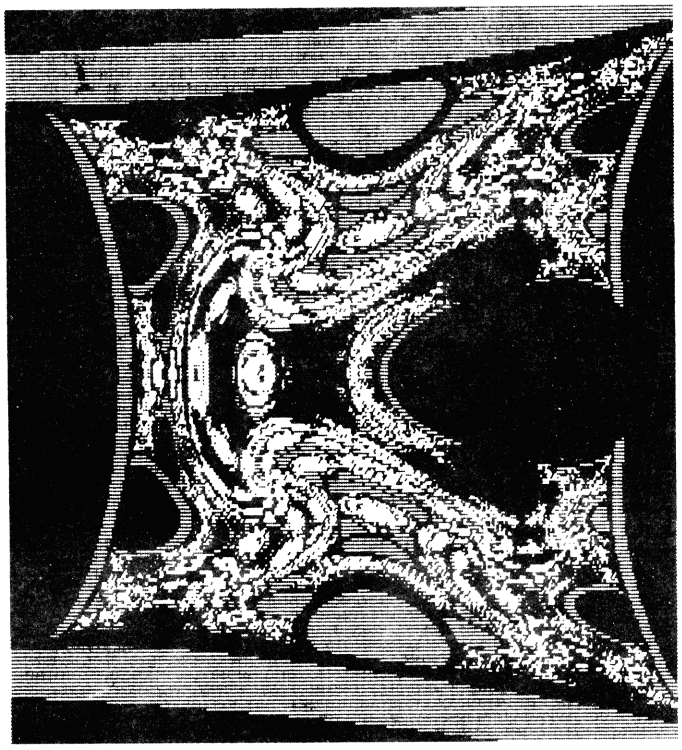
Figure 2. Overall view of fractal pattern from system below:

$$x_{n+1} = a + bx_n + c(y_n)^2$$

$$y_{n+1} = d + ex_n$$

Section 1

a = -1.5
b = -0.5
c = 2.4
d = -0.45
e = 0.5



Section 2

across = 300
down = 200

Section 3

xmin = -3.5
xmax = 4.5
ymin = -2
ymax = 2

Section 4

maxiter = 30
threshold = 500

Section 1: shows the coefficients of the system

Section 2: 'across' represents number of pixels used across picture
'down' represents number of pixels used down picture

Section 3: Area of the plane studied

Section 4: Constants of thresholds used (see basic program)

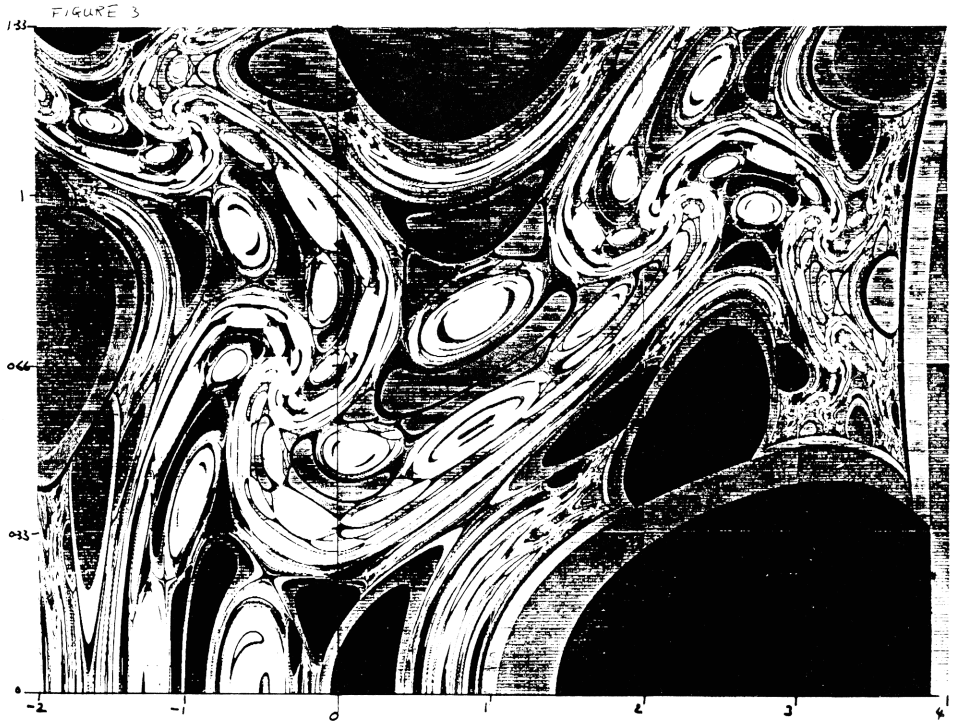


Figure 3 This is the upper portion of figure 2. The numbers shown indicate its position relative to the x and y-axes.

If the following coefficients are used another pattern arises:

a = 1; b = -2.4; c = -.98; d = 0; e = 0.71

These are shown in Figure 4. There is a resemblance between figure 4b and one of the poincare maps discovered by Henon shown on page 148 of James Gleick's book, 'Chaos: Making a New Science'. I do not know if this system is mathematically linked to his.

References:

'The Beauty of Fractals' H.O. Peitgen, P. Richter, Springer-Verlag, Heidelburg (1986).
'The Science of Fractal Images' H.O. Peitgen, D. Saupe (editors) Chapter 3,'Fractal patterns arising in chaotic dynamical systems' pp. 137-167 Springer-Verlag, New York (1988).
'Chaos: Making a new science' James Gleick, Heinemann, London (1987).

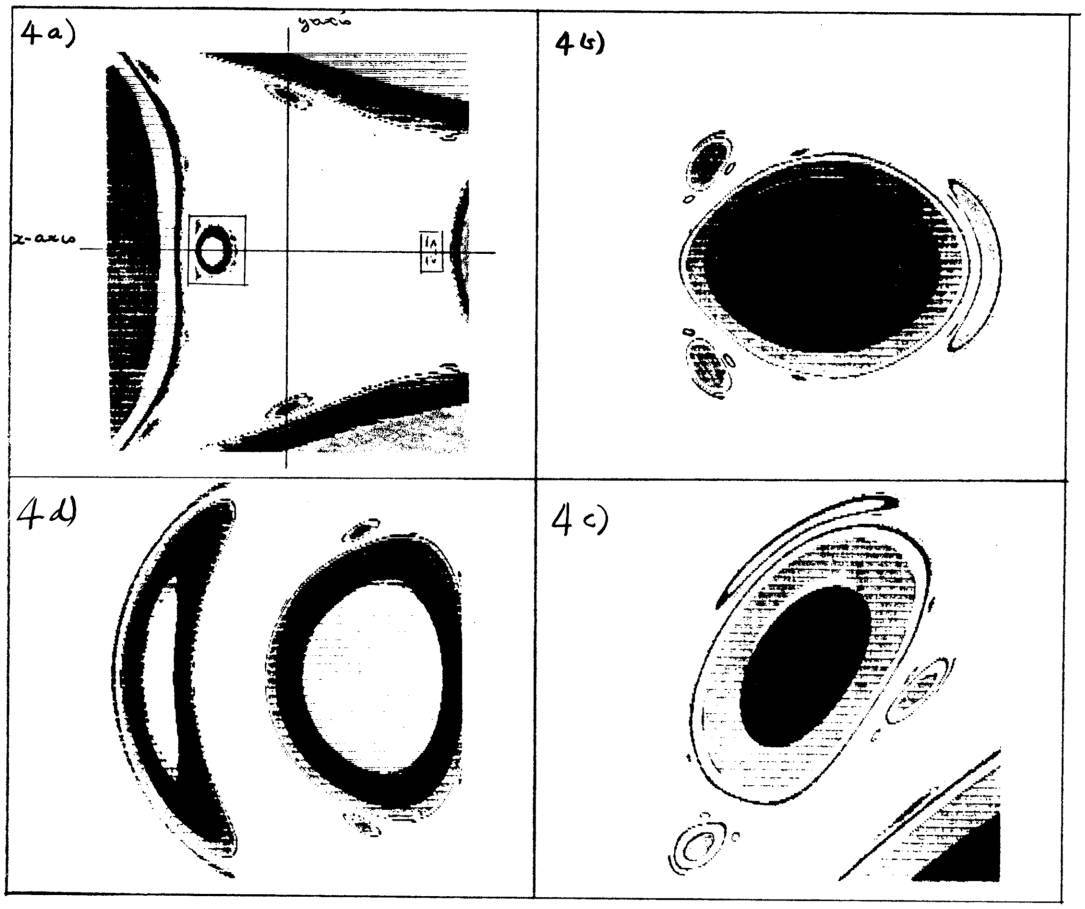


Figure 4 a) This shows the overall view of the pattern. The coordinates of this pattern are:

Xmin = -1.5; Xmax = 1.5; Ymin = -1.05; Ymax = 1.05

b) This shows detail of 'a'. The coordinates are:

Xmin = -0.8; Xmax = -0.4; Ymin = -0.25; Ymax = 0.25

c) This shows detail of 'b'. The coordinates are:

Xmin = -0.78; Xmax = -0.702; Ymin = 0.0737; Ymax = 0.1716

d) Another part of 'a'. The coordinates are:

Xmin = 1.125; Xmax = 1.22; Ymin = -0.095; Ymax = 0.095

```
100 REMark **** 2-Dimensional Chaotic Dynamical Systems ****
110 :
120 REMark    ****  Setting screen for QL format           ****
130 MODE 4:PAPER 0:PAPER#0,0:CLS:CLS#0
140 :
150 REMark    ****  Intialising screen centring constants ****
160 xpos=250:ypos=100
170 :
180 REMark    ****  Pixel dimensions                      ****
190 across=300:down=200
200 :
210 REMark    ****  Coefficients of system                ****
220 a=-1.5 :b=-.5 :c=2.4
230 d=-.45 :e=.5
240 :
250 REMark    ****  Limit coordinates in the plane        ****
260 xmin=-3.5 :xmax=4.5 :ymin=-2 :ymax=2
270 :
280 REMark    ****  Threshold constants                   ****
290 maxiter=30 :Cresh=500
300 :
310 REMark    ****  Determining coordinate of each pixel  ****
320 Dx=(xmax-xmin)/across
330 Dy=(ymax-ymin)/down
340 :
350 REMark    ****  Setting each pixel                    ****
360 FOR xnn=0 TO across-1
370 FOR ynn=0 TO down-1
380 k=0
390 xn=xmin+Dx*xnn:yn=ymin+Dy*ynn
400 :
410 REMark    ****  Main iterative loop                   ****
420 REPeat loop
430 k=k+1
440 IF xn^2+yn^2>Cresh:PLOT_POINT:EXIT loop:END IF
450 IF k>maxiter:EXIT loop:END IF
460 xm=a+b*xn+c*yn*yn
470 ym=d+e*xn
480 xn=xm:yn=ym
490 END REPeat loop
500 :
510 END FOR ynn
520 END FOR xnn
530 :
540 :
550 REMark  **** QL system of assigning colour to pixels  ****
560 REMark  **** in mode 4 and plotting them              ****
570 DEFine PROCedure PLOT_POINT
580 IF k MOD 3=0:col=2:END IF
590 IF k MOD 3=1:col=4:END IF
600 IF k MOD 3=2:col=7:END IF
610 BLOCK 1,1,xpos-.5*across+xnn,199-(ypos-.5*down+ynn),col
620 END DEFine
```

Mr Topham has a more advanced version of this program available for the Sinclair QL. He will send it upon receipt of £4 and a blank microdrive cartridge or 3.5' disk. He will send a listing and conversion notes for anyone who wishes to convert to other machines for £2. He also has other articles in preparation on a simpler z^n+1 program and images from the solution of fifth degree complex polynomials with complex coefficients, and we look forward to these in due course. He is happy to correspond with anyone having difficulties with his programs.

114, Mid Street, South Nutfield, Redhill, Surrey, RH1 4JH