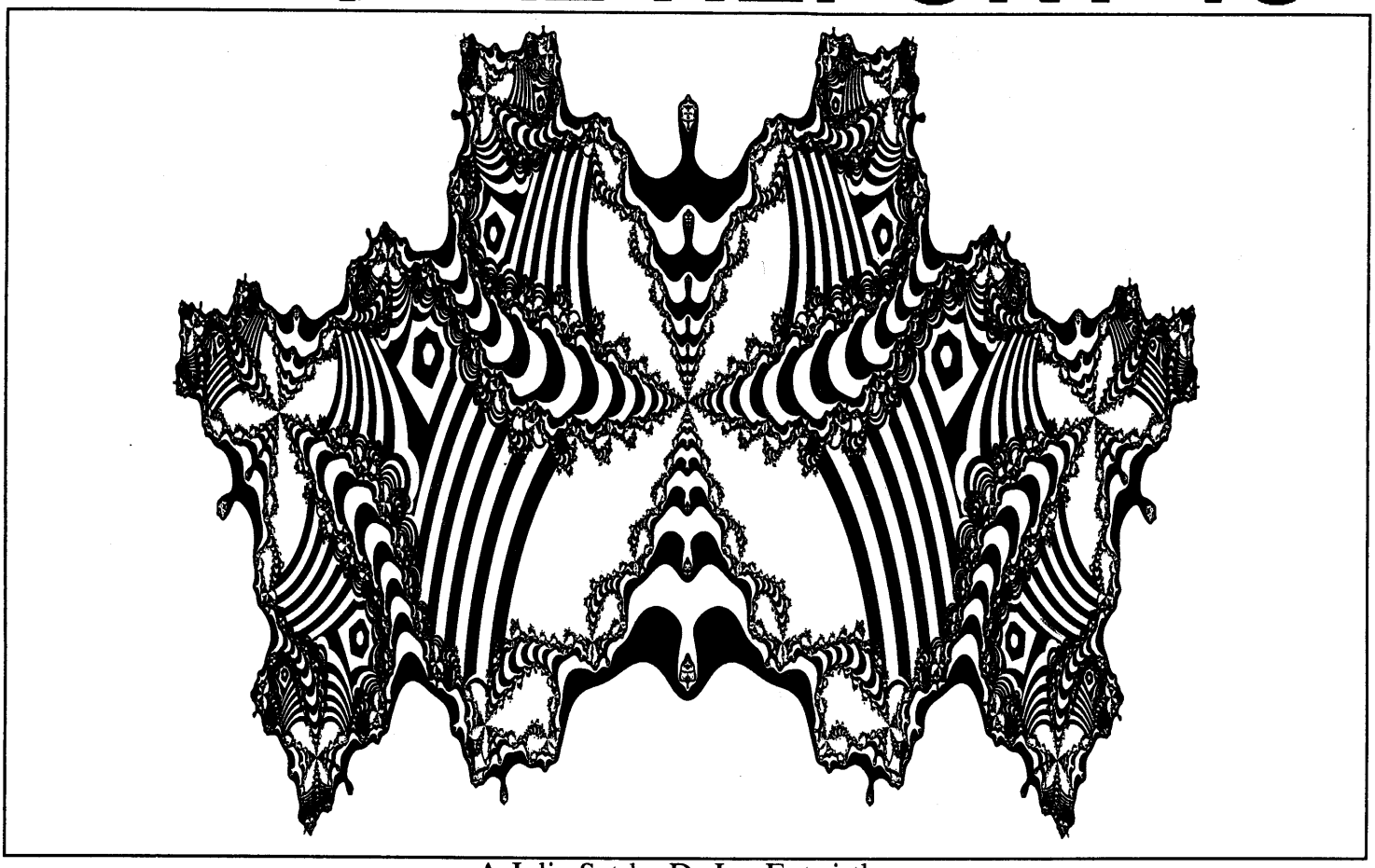
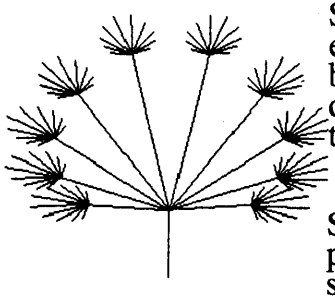
FRACTAL REPORT 13



A Julia Set by Dr Ian Entwistle.



Single copy rate £2. Subscription rates six issues: – £10 (UK only) £12 Europe £13 elsewhere. Cheques in British Pounds should be drawn on a UK bank and should be made payable to "Reeves Telecommunications Laboratories Ltd." Alternatively, dollar checks for $23 can be accepted if drawn on a U.S. bank and made payable to "J. de Rivaz".

Subscribers who are successful in getting one or more articles with programs published in a given series of six issues get the next volume of six issues free of subscription. All new subscriptions are backdated to the start of the current volume.

Fractal Report is published by Reeves Telecommunications Laboratories Ltd.,
West Towan House, Porthtowan, Truro, Cornwall TR4 8AX, United Kingdom.
Volume 3 no 13 First published February 1991. ISSN applied for.

FURTHER FRACTALS FROM NEWTON'S FORMULA

J. C. Topham

Various books and some articles in 'Fractal Report' have featured the production of fractal images from the use of Newton's method of finding the roots of equations with complex variables.

Examples such as equations $z^n - 1 = 0$, $n = 1,2,3,4,...$ (1) have been dealt with before (see Figure 1).

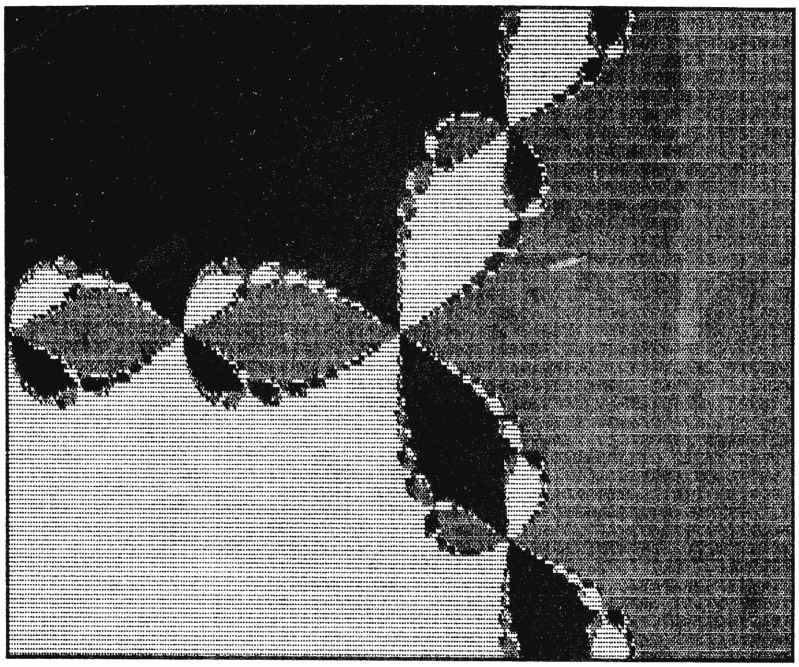


Figure 1: $F(z) = z^3 - 1$
Roots: $z_1 = 1$, $z_2 = -.5 + .866i$, $z_3 = -.5 - .866i$
Picture dimensions: X-axis -1.5,1.5 Y-axis -1.5,1.5

What would be the result if 'the power' is negative, i.e. if the equation becomes $z^{-n} - 1 = 0$ or $(1/z^n) - 1 = 0$, $n=1,2,3,...$(2). Any trained mathematician would say that this would only be a trivial rearrangement of equation (1) (multiply both sides of $z^{-n} - 1 = 0$ by $-z^n$ and you get $-1 + z^n = 0$ or $z^n - 1 = 0$). However, if equation (2) is plugged into Newton's formula as it is surprising images appear. Figure 2 shows the attraction basins of the equation: $z^{-3} - 1 = 0$, the roots of which are: $z_1 = 1$, $z_2 = -.5 + .866i$, $z_3 = -.5 - .866i$

The white background is the area where the iterations, starting from these coordinates, diverge to infinity. The areas that are shaded show that these initial coordinates converge to a root. This contrasts greatly from the previous $z^n - 1 = 0$ examples where virtually any initial coordinate homes in onto a root.

Unlike the equation $z^2 - 1 = 0$ that seems to produce no fractal behaviour, the equation: $z^{-2} - 1 = 0$ offers some images as in figures 3 and 4.

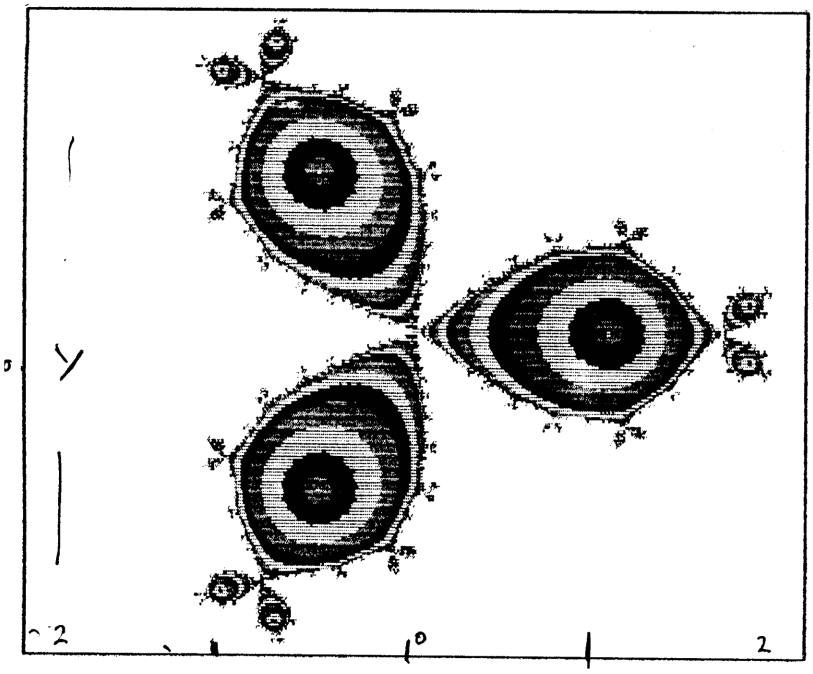


Figure 2: $F(z) = z^{-3} - 1$
Roots: $z_1 = 1$, $z_2 = -.5 + .866i$, $z_3 = -.5 - .866i$
Picture dimensions: X-axis: -2,2 Y-axis: -1.75,1.75

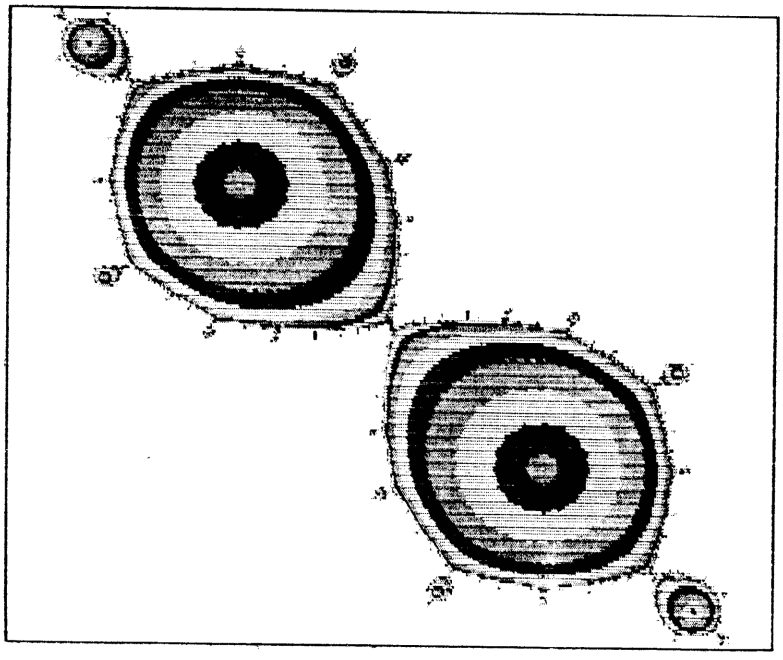


Figure 3: $F(z) = -.5 + z^{-2}$
Roots: $z_1 = -1 + i$, $z_2 = 1 - i$
Picture dimensions: X-axis: -2.5, 2.5 Y-axis: -2.25, 2.25

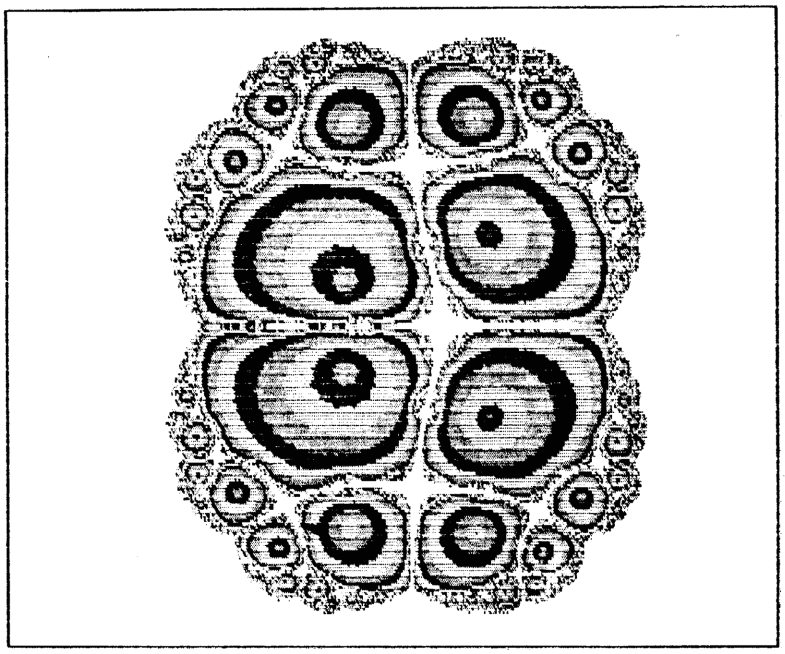


Figure 4: $F(z) = 1.6 - 2.4z^{-1} + z^{-2}$
Roots: $z_1 = .75 + .25i$, $z_2 = .75 - .25i$
Picture dimensions: X-axis -1,3 Y-axis -1.75,1.75

If the program is set up to handle any third order polynomial i.e. it calculates the coefficients when only the roots are given, interesting pictures can be produced when the roots are put close together (see figure 5).
It seems to shows a sort of interference occuring.
Other examples are shown in figures 6 to 10.

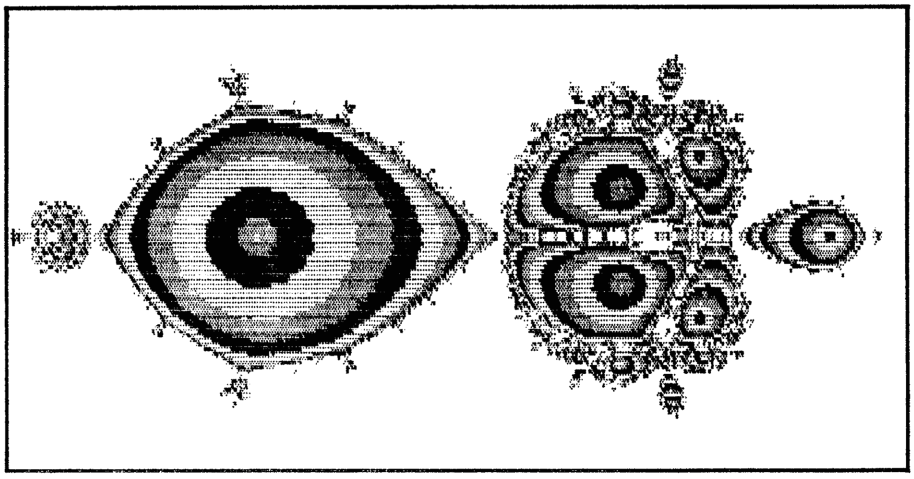


Figure 5: $F(z) = 1 - 2.45z^{-1} + 3.45z^{-3}$
Roots: $z_1 = -1$, $z_2 = .5 + .2i$, $z_3 = .5 - .2i$
Picture dimensions: X-axis: -2,1.75 Y-axis: -1,1

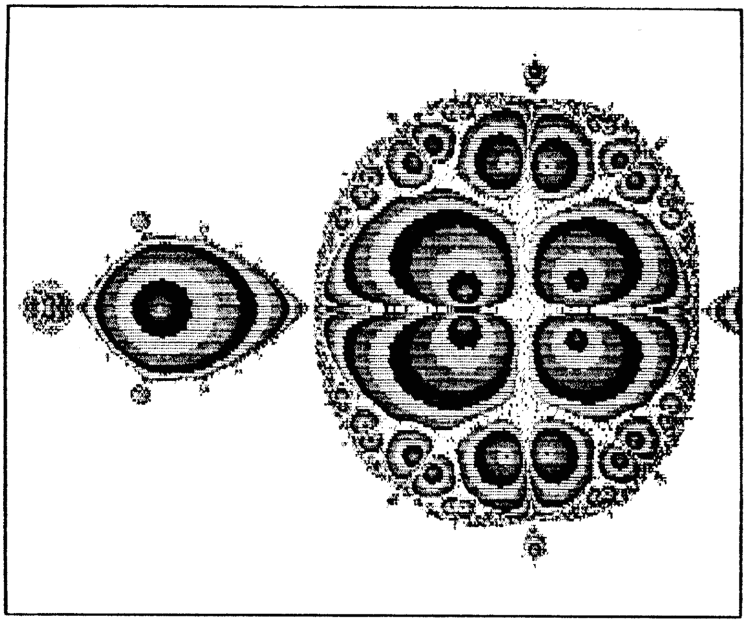


Figure 6: $F(z) = 1 - .98z^{-1} - .991z^{-2} + .991z^{-3}$
Roots: $z_1 = -1$, $z_2 = 1 + .1i$, $z_3 = 1 - .1i$
Picture dimensions: X-axis: -2,2.8 Y-axis: -2,2

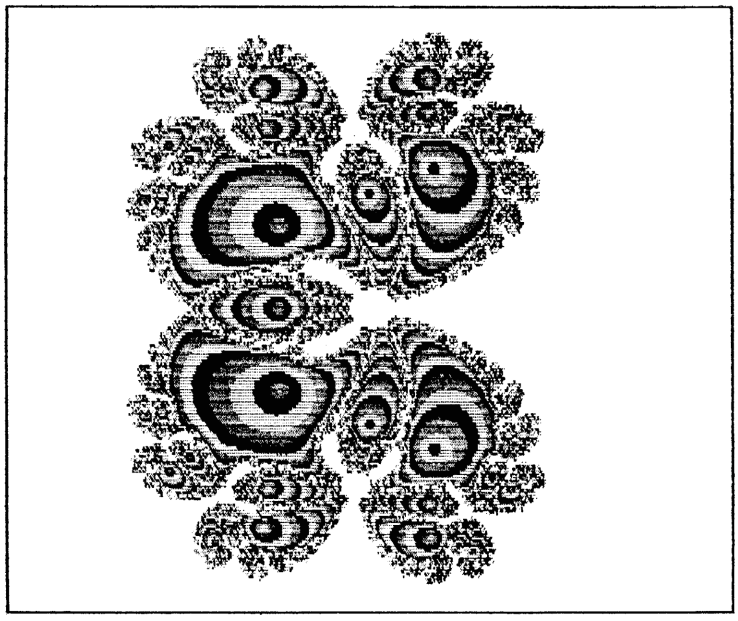


Figure 7: $F(z) = -1.185 + 2.667z^{-1} - 2.667z^{-2} + z^{-3}$
Roots: $z_1 = .75 + .75i$, $z_2 = .75$, $z_3 = .75 - .75i$
Picture dimensions: X-axis: -1.5,4.5 Y-axis: -2.75,2.75

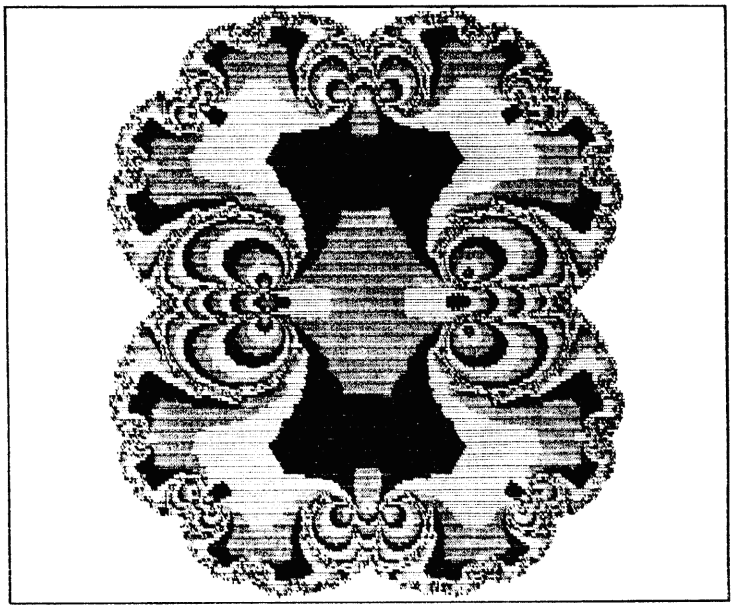


Figure 8: $F(z) = -.96 + 2.885z^{-1} - 2.923z^{-2} + z^{-3}$
Roots: $z_1 = 1 + .2i$, $z_2 = 1$, $z_3 = 1 - .2i$
Picture dimensions: X-axis: -1.5,5.5 Y-axis: -3.2,3.2

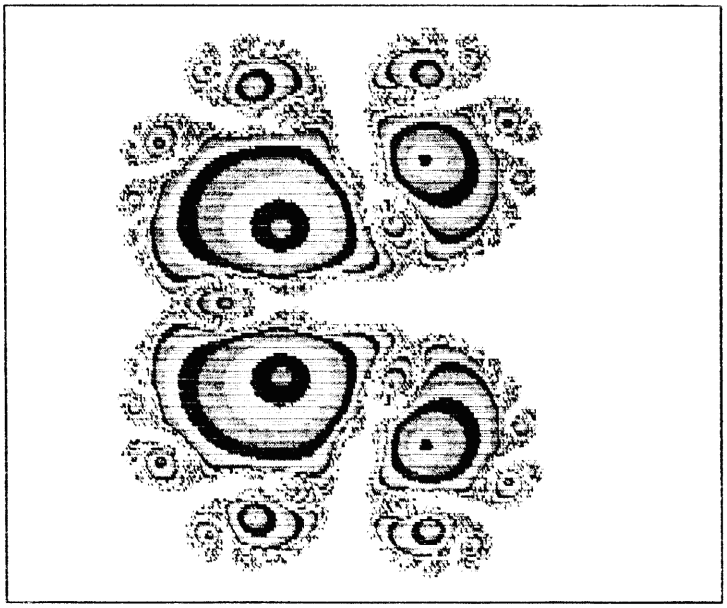


Figure 9: $F(z) = -1.28 + 3.2z^{-1} - 3.28z^{-2} + z^{-3}$
Roots: $z_1 = 1 - .75i$, $z_2 = .5$, $z_3 = 1 + .75i$
Picture dimensions: X-axis: -1.5,5 Y-axis: -3,3

Figures 10 to 12 are fourth order polynomial images.

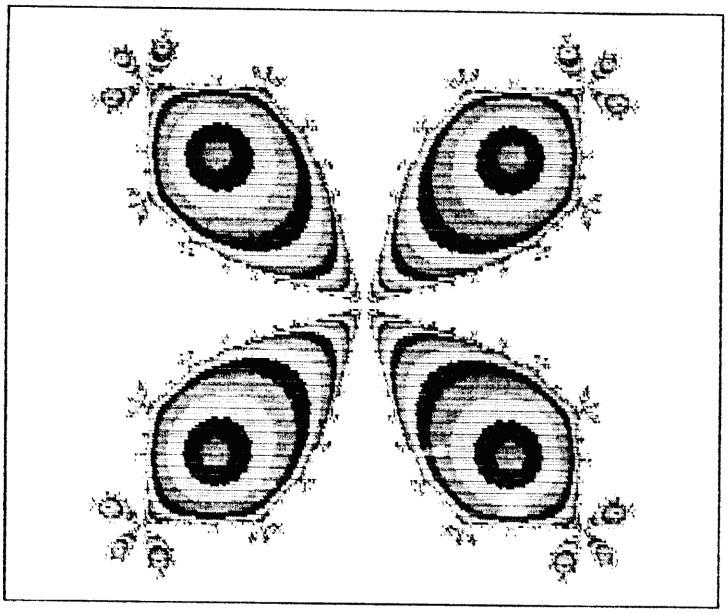


Figure 10: $F(z) = 1 + z^{-4}$
Roots: $z_1 = -.7071 - .7071i$, $z_2 = -.7071 + .7071i$
$z_3 = .7071 + .7071i$, $z_4 = .7071 - .7071i$
Picture dimensions: X-axis: -1.7,1.7 Y-axis: -1.4,1.4

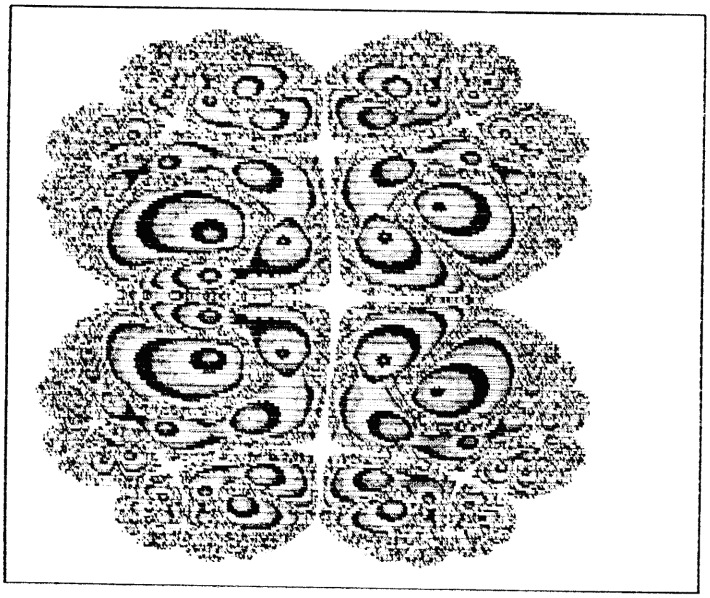


Figure 11: $F(z) = 1.42 - 4.27z^{-1} + 5.67z^{-2} - 3.73z^{-3} + z^{-4}$
Roots: $z_1 = .75 + .75i$, $z_2 = .75 + .25i$
$z_3 = .75 - .25i$, $z_4 = .75 - .75i$
Picture dimensions: X-axis: -1,5 Y-axis: -3.5,3.5

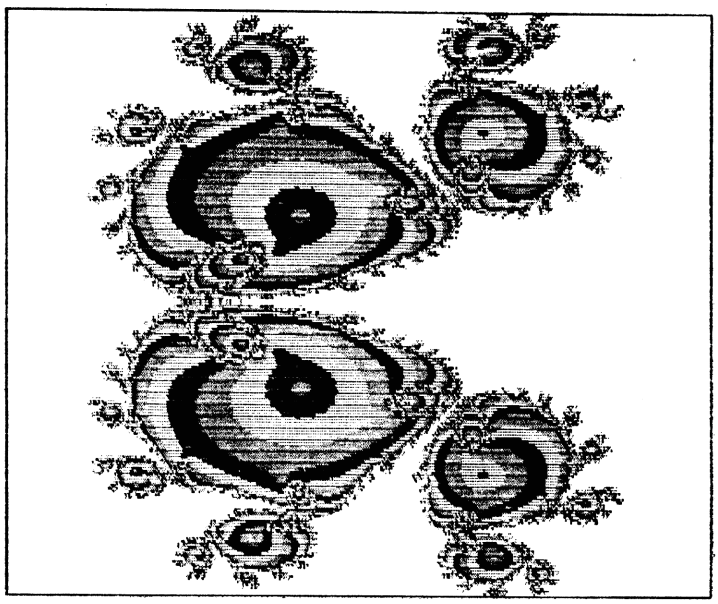


Figure 12: $F(z) = .0625 - .375z^{-1} + 1.125z^{-2} - 1.5z^{-3} + z^{-4}$
Roots: $z_1 = 2 + 2i$, $z_2 = 1 + i$
$z_3 = 1 - i$, $z_4 = 2 - 2i$
Picture dimensions: X-axis: -3,9 Y-axis: -6.75,6.75

No doubt higher degree polynomials will also produce interesting pictures.
I include a program written in QL 'Superbasic' to draw the $z^{-3} - 1 = 0$ attraction basins for anyone to experiment with.

```
100  REMark The Newton Formula used
110  REMark   on F(z)= z^(-3) - 1
120 :
130 wide=512:height=256
140 WINDOW wide,height,0,0
150 PAPER 0
160 CLS
170 across=200 :down=100
180 xmin=-1:xmax=1
190 ymin=-1:ymax=1
200 :
210 DIM ry(5)
220 ry(1)=-SQRT(3)/2
230 ry(2)=SQRT(3)/2
240 ry(3)=0
250 :
260 Dx=(xmax-xmin)/across
270 Dy=(ymax-ymin)/down
280 xpos=INT((wide-across)/2)
290 ypos=INT((height-down)/2)
300 :
310 FOR yp=0 TO down
320  FOR xp=0 TO across
330  AT 10,10:PRINT yp,xp;'
340   yn=ymin+(yp*Dy)
350   xn=xmin+(xp*Dx)
360   FOR iter=1 TO 30
370    INVERSE_Z3
380    xm=(a*c+b*d)/(c^2+d^2)
390    ym=(b*c-a*d)/(c^2+d^2)
400    IF xm^2+ym^2>1000
410     EXIT iter
420    END IF
430    IF ABS(yn-ym)<1E-2
440    IF ABS(xn-xm)<1E-2
450     FOR i=1 TO 3
460      IF ABS(ry(i)-ym)<1E-2
470       col=((iter MOD 3)+1)*2
480       PLOT_POINT:EXIT iter
490      END IF
500     END FOR i
510    END IF
520    END IF
530    xn=xm:yn=ym
540   END FOR iter
550  END FOR xp
560 END FOR yp
570 STOP
580 :
590 DEFine PROCedure PLOT_POINT
600 p=xpos+xp
610 q=ypos+down-yp
620 BLOCK 1,1,p,q,col
630 END DEFine
640 :
650 DEFine PROCedure INVERSE_Z3
660 a=4*xn-xn^4+6*xn^2*yn^2-yn^4
670 b=4*yn-4*xn^3*yn+4*xn*yn^3
680 c=3
690 d=0
700 END DEFine
```

Editorial

It is probably seasonal considerations that makes the number of short articles with programs that readers seem to like somewhat scarce this issue. However this has given me the opportunity to publish Mr Parker's article in full, which I hope will benefit some readers and indeed may prove to be helpful to contributors interested in producing new fractal programs. I have been wanting to publish this article for some while, but was reluctant because for many readers, particularly beginners, it could prove hard going. We do have a few *Fractal Report* type articles still on file, held over to make space for the long articles this time, and anticipate that by the time the next issue appears we can revert to the usual format. There are not enough to make a full issue, and I hope that some more will be forthcoming by around 1 March when issue 14 goes to the printers.

Karen Griffin, my companion for some years, will be leaving sometime later this year. Her help with *Fractal Report* has been much appreciated, even if sometimes it was a bit adamant. We intend to remain friends, and she hopes to continue to offer help and advice with my publications whenever possible.

Announcements

Archimedes Correspondents wanted

Mr J. Mourik, of 3rd Millennium, Box 11, Ammanford, Dyfed, SA18 3WB wants Archimedes correspondents. He has 3.5" disks and is interested in cellular automata, fractal dust and sparklies, and wants help in getting hard copy from images saved to disk, and someone to swap programs with.

Reader's Hall of Fame

Mr Darryl Catchpole, of Scunthorpe, got a Mandelbrot Set program published in *Amstrad Action*'s *Type – Ins* section, issue 63 pages 76/7. Unfortunately they edited out his bit about *Fractal Report*. My advice is that if you want to mention *Fractal Report* then include it in a REM statement in the middle of your program – the magazine is reluctant to edit it out then!

However if anyone writes to Mr Catchpole as a result of the article he will mention *Fractal Report* to them. He also asks me to mention to readers that if anyone wants a trainee programmer, then he makes good coffee!

Striations

Mr L.G.L. Unstead – Joss refers to Dr Wolf's article in issue 12, and mentions that variations in refractive index, e.g. in heated glass, are called "striations". He recalls learning this 50 years ago.

A Crack in Mandelbrot Space?

Mr D.I. Brett sent in some interesting prints from his Archimedes that appeared to show magnifications of "dull" areas way outside the main set, which when magnified enough, showed dots which expanded into new weird images. True to form, these images included miniature Mandelbrots. I sent them to Dr Ian Entwistle, who also has an Archimedes, but his initial reaction was that it was probably due to a precision error in the program.

Fractals: An Animated Discussion

This video has been mentioned before as being available from Germany for about DM100, but it is now available in the UK from W.H. Freeman and Co, 20, Beaumont Street, Oxford OX1 2NQ, for £40.20 including tax and post. It runs for 63 minutes and includes interviews with Edward Lorenz and Dr Benoit Mandelbrot together with animated sequences and fractal music. Thanks are again due to Dr Entwistle for this item.

Sam Coupé Subgroup?

Dr Derek Burn eulogised over Ettrick Thomson's article including a Sam Coupé program in issue 12. Dr Burn is a relative newcomer to programming, and finds it difficult to convert other BASICs to Sam BASIC.

He asks whether any *Fractal Report* reader has converted some of the programs given into Sam BASIC, and whether anyone has some hints on translating to Sam BASIC.

Obviously this would be of interest to Sam Coupé users only, so is there anyone out there interested in forming a sub group, with distribution of Sam programs or listings just to Sam Coupé owners?

Dr Burn also mentions that the Sam Coupé is flourishing, and the new company Sam Computers Ltd has also introduced a 1 megabyte memory expansion.

Amygdala Issue 21

The main article of this issue is *Polar Coordinates and the Cardioid Body of M.* It provides mathematical insight into the circular shape of the "head" of the Mandelbrot Set and the cardioid shape of the body. Other topics covered are Lissajous figures and trigonometric identities. All four of the slides were produced by *Fractint 13.0.* An advertisement flyer contains $30 rectangles many of which were filled with interesting offers, although some were somewhat expensive. One program originator wanted a whopping $20 to post the $79 disk and manual outside America. (Must be some manual!) [Address: *Amygdala,* Box 219, San Christobal, NM87564, USA.]

Chaotic Shops

Mr P.J. Mortimore kindly sent in a leaflet produced by *Strange Attractions* of 204, Kensington Park Road, London W11, whilst renewing his subscription. They provide a range of fractal artifacts from 35p to £120. There was an article about Mr Gregory Sams' store in *The Evening Standard,* undated. It concluded: *It's weird, but its going to be big, very big. After all, Mr Sams has been responsible in the past for such word shaking concepts as the first macrobiotic restaurant, whole food shops, and the VegeBurger.*

He also included an article from *Microcomputer Mart* 130 about Mr Jake Davis' operation *Frachaos*, which covers similar material. (*Frachaos*, Higher Trengrove, Constantine, Falmouth, Cornwall TR11 5QR now have a 27 page illustrated catalogue of programs, books, videos, tee shirts, music, prints and slides.)

More Mandelbrot Sets

By Chris Sangwin.
(Square by A. Byde)

There have been many variations on the old favourite of the Mandelbrot set. This is my variation and it involves, not the set itself, but the way the computer decides if the points are outside the bounds. Normally the bound if a circle produces by the pseudo code;

```
Repeat
X(n+1) = X²n - Y²n + R
Y(n+1) = 2 * Xn * Yn + I
INC Itt
Until X² + Y² >4 OR Itt=255
```

The variable "itt" decides weather the system has stabilized (reached a root) and the "$X^2 + Y^2 > 4$" decides weather the system will tend to infinity. Mandelbrot identifies the set on page 188 of his book The Fractal Geometry of Nature as " iterates of $Z_0=0$ under $Z \rightarrow Z^2 - u$ (which) fail to converge to infinity."

By changing the method of deciding weather a point will converge to infinity you can get startling new results. Of course the basic shape of the set at the centre remains unchanged.

The outer borders and the colour contours are changed drastically. I an sure that zooms into these sets will produce as many fantastic shapes as zooms into the normal version do.

The first different border tried was a square derived from the simple inequalities;

X<2 AND Y<2 AND X>-2 AND Y>-2
(X^2 <4 AND Y^2 <4)

This stops the process when X or Y are outside the square within by X= +2 or -2 and Y= +2 or -2. Another effective inequality is the inverse of a circle caused by;

X*Y<4

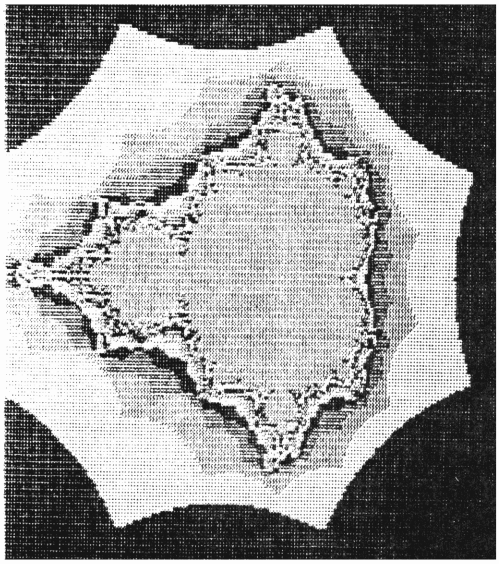
This only Works when X<>0 and Y<>0 because if either is zero then the point will never satisfy the inequality and an overflow error will result. This happened to me many times to begin with. The programmer has to be careful to make sure the loops do not pass through zero but very close either side of it.

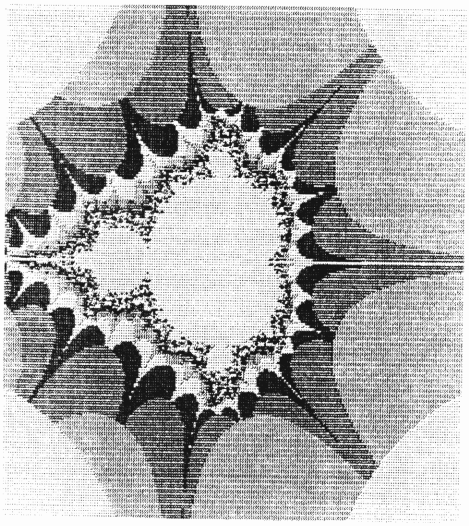
Both these new sets can be melted in the same way suggested by Steve Wright in issue 2 of Fractal Report or plotted in a 3D way.

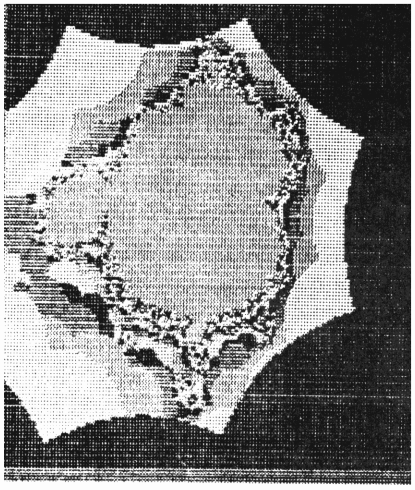
As these new variations are still technically Mandelbrots sets I am sure they would still apply for the "Larry T Cobb Prize" Awards that appear periodically.

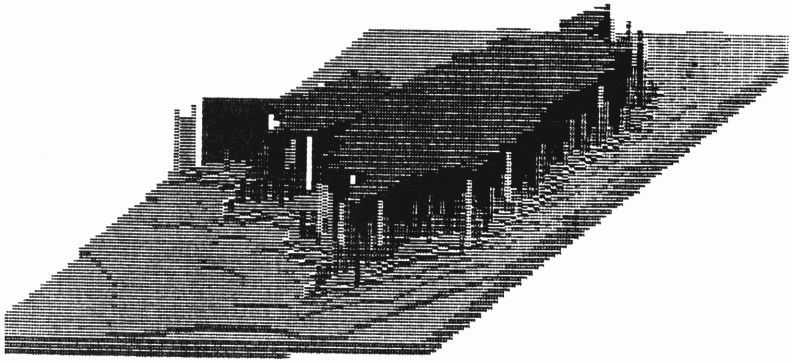
The frames that appear opposite are (Clockwise from top left);

All Frames : M-set , I=-2 to +2 , R=-2 to +2 (ie all of it)
Colouring on a 1 iteration to one colour scale
All Produced on Amstrad CPC 664 (Photo reduced)
Iterations: 100









1 : The Square variation $X^2 < 4$ AND $Y^2 < 4$
2 : Inverse version $X * Y < 4$
3 : Melted Square (no 1) melt factor = Ø.3
4 : 3d Mandelbro set. Square.
Height as in my programme in Amstrad Action no 6Ø pg 73.

COMPRESSING FRACTAL IMAGES

by Mike Parker, 22 Hutchcomb Road,
Botley, Oxford OX2 9HL.
Tel (0865) 725495 (evenings)

Once you have produced your image, what do you do next? You could either switch off your computer thus losing your image, or leave it running indefinitely thus rendering it unavailable for further work. Neither of these options are at all satisfactory. You could output the image to a colour printer or plotter if you are lucky enough to have access to such equipment. You could take a photograph of the screen (hopefully the subject of a future article). Or finally you could dump the image to disk so that it can be restored later for display or to allow incomplete images to be finished. This is obviously the best choice.

Unfortunately there is a problem with this. With low resolution images using few colours little disk space is required, but with high resolution images using many colours a large amount of disk space is required. For example, my program uses an image size of 648 by 567 pixels in 256 colours = 648 * 567 * 8 bits = approximately 360k which is one full single sided quad density disk. Some method must be found to reduce this amount of data.

Before discussing various compression methods a brief description of the method I use for plotting images is given as it has a bearing upon one of them. First choose a plotting area which is a multiple of a power of 3. In my case this $8 * 3^4$ by $7 * 3^4 = 8 * 81$ by $7 * 81 = 648$ by 567 pixels. Tile this area using squares of side 81 using the centre of each square to calculate the required colour. Then for each of these squares, split it into 9 equal squares and for each of the 8 edge squares plot the square using its centre point. Repeat until squares of side 1 have been plotted. See listing 1.

The advantages of this method are:

i) Because a power of three is used, no points need to be recalculated and replotted. The centre point of each square remaining static.

ii) No distortion is introduced as happens with power of two tiling.

iii) Structure of plot is quickly shown in low resolution with the resolution being increased by nine times on each pass.

It does have a few disadvantages as well:

i) Increased complexity of plotting algorithm.

ii) Limits choice of screen size, although this can be partially overcome by using a smaller power of three.

See the article by Jack Weber in PCW December 1986 for further details of this method.

Pleasing fractal images usually consist of largish areas of single colours with smaller areas of fine detail. If some way can be found to represent these large areas a significant saving can be made. See table 1 for a comparison of the various compression methods described below.

The commonest and easiest method to implement is that of run length coding (method 1a). The image is scanned from top left to bottom right, outputting the number of pixels of the current colour (8 bits) and current colour (8 bits). This is fine for large single colour areas but not where adjacent pixels are of different colours when two bytes are required to represent each pixel.

Variations of this method (methods 1b, 1c, 1d, 1e) can use a different number of bits for the pixel count (for example four, reducing the overhead for single pixels but reducing the compression obtained on large areas) and current colour if fewer than 256 colours are available. The optimum number of bits for the pixel count varies depending on the complexity of the image, although four bits seems to give acceptable results for plots of medium complexity.

A further improvement (methods 1f, 1g) is obtained by noting that for areas of fine detail there is a large overhead caused by having a count of one (n bits) for every pixel. By using a count of one followed by a subcount of the number of single pixels followed by those pixel colours this overhead is substantially reduced; counts greater than one are handled as before. The optimum number of bits for the count and subcount again varies depending on the complexity of the image. For simple images, a large count and small subcount are required, whereas for complex images, a small count and large subcount are required. Method 1f uses a count of four bits and a subcount of four bits; method 1g uses a count of five bits and a subcount of four bits.

A variation of the run length coding method (method 2) is to work on each colour plane independently, outputting the start colour (as 1 bit) followed by a count of the number of pixels of the same colour (n bits), the start colour being flipped after each count. A count of zero is used to indicate a maximum count and that the start colour is not to be flipped. The number of bits used to hold the count changes for each colour plane, varying from a small count for the least significant colour plane to a large count for the most significant colour plane. Optimum values again depending on the complexity of the image. Values of three, three, three, four, four, four, four and four have been used (optimised for map 45) for the examples in table 1.

Another method (method 3a) is to use a plotting area of a multiple of a power of three. In my case this is 648 by 567 pixels. For each square of side 81, check if all pixels are of the same colour. If they are then output

the current square size (81) (as 8 bits) and colour (8 bits). Otherwise split the square into nine equal squares and for each of them repeat the procedure. If a square of three by three pixels is reached which is not of one colour then output a square size of one (8 bits) and the nine colours in the three by three square (9 * 8 bits).

A refinement of this (method 3b) is not to hold the square size as 81, 27, nine, three or one in eight bits but to use zero, one, two, three or four (3 bits) instead. See listings 2 and 3.

A further refinement (method 3c) can be made by examining the data stored in the three by three square. It can contain either two, three, four or greater than four different colours. For the case of greater than four different colours, output a square size of four (3 bits) followed by the nine colours as in methods 3a and 3b above. For the case of three or four different colours, output a square size of six or five (3 bits) followed by the three or four different colours (in the order they are found in the three by three square) followed by a colour index for the nine pixels (15 bits) made up as follows. No bits are required for the first pixel as it must be the first colour found, one bit is required for the second pixel as it can be either the first or second colour found (0=first, 1=second), two bits are required for each of the remaining seven pixels (0=first, 1=second, 2=third, 3=fourth). For the case of two different colours, output a square size of seven (3 bits) followed by the two different colours (in the order they are found in the three by three square) followed by a colour index for the nine pixels (8 bits) made up as follows. No bits are required for the first pixel as it must be the first colour found, one bit is required for each of the remaining eight pixels as they can be either the first or second colour (0=first, 1=second).

A final refinement (methods 3d, 3e) can be made by examining the data stored in the three by three square in more detail. It can contain either two, three or more than three different colours. For the case of greater than three different colours, we find that many of the pixels have either adjacent colours or are black with only a few widely spaced colours. By choosing one of the colours as a base colour, many of the pixels can be represented as a small offset from this base colour as follows. Use each of the first eight colours in the square, excluding those with a value of zero (black) as the base colour and calculate the number of pixels that are not less than the current base colour and are not greater than the current base colour by more than 14 (6 for method 3e). Make a note of which base colour has the largest count. If this count plus the number of pixels in the square with a value of zero is not less than three then output a square size of five (3 bits), the number of the base colour (3 bits) and its actual colour (8 bits). Then for each of the remaining eight pixels output either a zero (1 bit) followed by the pixels actual colour (8 bits), or a one (1 bit) followed by an offset (4 bits for method 3d, 3 bits for method 3e). An offset between zero and 14 (or 6) represents a positive offset from the base colour. An offset of 15 (or 7) represents a pixel with a value of zero. If the count plus the number of pixels in the square with a value of zero is less than three, then output a square size of four (3 bits) followed by the nine colours (9 * 8 bits).

For the case of three different colours, we find that these colours are usually adjacent and so can be represented by a small offset from the first colour as follows. Output a square size of six (3 bits) followed by the first colour (8 bits). If the two remaining colours are both no more than two colours away from the first the output a one (1 bit) followed by an offset for each of the two colours (2 * 2 bits) followed by the colour index (15 bits) as for method 3c. The offset is as follows; zero = colour 1 + 1, one = colour 1 + 2, two = colour 1 - 2, three = colour 1 - 1. If either of the two remaining colours are greater than two colours away from the first then output a zero (1 bit) followed by the two remaining colours (2 * 8 bits) followed by the colour index (15 bits) as for method 3c.

For the case of two different colours we find that in most cases the two colours are adjacent and so the second colour can be represented by a very small offset from the first as follows. Output a square size of seven (3 bits) followed by the first colour (8 bits). If the second colour is not more than one colour away from the first then output a one (1 bit) followed by an offset for the second colour (1 bit) followed by the colour index (8 bits) as for method 3c. The offset is as follows; zero = colour 1 + 1, one = colour 1 - 1. If the second colour is more than one colour away from the first then output a zero (1 bit) followed by the second colour (8 bits) followed by the colour index (8 bits) as for method 3c.

A variation of this tiling method (method 4) is to work on each colour plane independently, outputting either zero (1 bit) followed by square size (2 bits; 0=81*81, 1=27*27, 2=9*9, 3=3*3) followed by the colour (1 bit), or one (1 bit) followed by the nine colours (9 by 1 bit).

In conclusion, a reduction to 60% of the original size is fairly easily obtained for all but the most complex of images and a reduction to one third of the original size is obtainable with the more complex compression methods. There are probably many other compression methods worthy of investigation. For instance, using Fast Fourier Transformations or using different tiling shapes such as triangles, as well as further variations on the methods described in this article. The application of commercially available archiving programs to the data produced by these compression methods might also provide a further reduction in size. I hope that this article has provoked some thought about the subject of storing fractals and I would welcome any suggestions for improvements to the above methods or any new ideas. There is also a need for standardising any claims made for compression rates as the same algorithm can produce such wide variations depending upon screen resolution, number of iterations, colour assignment methods, plot coordinates, etc. Any ideas?

LISTING 1

```
PROGRAM fractals;
{This program is written using Hisoft Pascal80 and illustrates the
 power of 3 tiling method of plotting Mandelbrot and Julia sets. Graphic
 handling routines have not been specified in detail as they are machine
 specific}
CONST sqinit = 81;  {Initial block size. Must be power of 3}
      width = 648;  {Width of plotting area. Must be multiple of sqinit}
      height = 567; {Height of plotting area. Must be multiple of sqinit}
      ncol = 256;   {Number of colours available}
      black = 0;
TYPE COMPLEX = RECORD
        re,im : REAL
     END;
     COORD = RECORD
        x,y : INTEGER
     END;
VAR corner,size,extent,square : INTEGER;
    map,minlevel,maxlevel,limit : INTEGER;
    scale,konst,min,max : COMPLEX;
    point,base : COORD;
    ch : CHAR;

PROCEDURE Rfill(x,y,w,h,col : INTEGER);
BEGIN
   {insert code to fill a rectangle of size w by h pixels in colour col
    with top left corner at x,y}
END;
PROCEDURE Plot(x,y,col : INTEGER);
BEGIN
   {insert code to plot a point in colour col at point x,y}
END;
PROCEDURE Calculate;
VAR col,i : INTEGER;
    temp,tempsq,znew,c : COMPLEX;
BEGIN
   i := 0;
   IF map = 1 THEN BEGIN              {initialise for Mandelbrot Set}
      temp.re := 0.0; temp.im := 0.0;
      c.re := min.re + (point.x * scale.re);
      c.im := max.im - (point.y * scale.im)   {for point 0,0 = top left
                                               corner of screen. If point
                                               0,0 = bottom left corner
                                               then c.im := min.im +
                                                    (point.y * scale.im)}
   END;
   IF map = 2 THEN BEGIN              {initialise for Julia Set}
      temp.re := min.re + (point.x * scale.re);
      temp.im := max.im - (point.y * scale.im);
      c := konst
   END;
   tempsq.re := SQR(temp.re); tempsq.im := SQR(temp.im);
   WHILE (i < maxlevel) AND (tempsq.re + tempsq.im < limit) DO BEGIN
      i := i + 1;
      znew.re := tempsq.re - tempsq.im + c.re;
      znew.im := (2.0 * temp.re * temp.im) + c.im;
      temp := znew;
      tempsq.re := SQR(temp.re); tempsq.im := SQR(temp.im)
   END;
   IF i = maxlevel THEN                  {calculate required colour}
      col := black
   ELSE IF i <= minlevel THEN
```

```
         col := 1
      ELSE
         col := ((i - minlevel) MOD (ncol - 1)) + 1;
   IF square = 1 THEN
      Plot(point.x , point.y , col)
   ELSE
      Rfill(point.x - extent , point.y - extent , square , square , col)
END;
BEGIN
   {insert screen initialisation code here (if required)}
   map := 1;                                        {Mandelbrot Set}
   minlevel := 1; maxlevel := 255;                  {min/max level of iterations}
   limit := 4;
   konst.re := -1.25; konst.im := 0.0;              {default Constant for Julia Set}
   min.re := -2.1; max.re := 2.1;                   {default plotting area}
   min.im := -2.1; max.im := 2.1;
   scale.re := (max.re - min.re) / (width - 1);     {find difference between}
   scale.im := (max.im - min.im) / (height - 1); {adjacent pixels}
   square := sqinit;                                {draw preliminary large blocks}
   extent := ENTIER(square / 2);
   size := square;
   corner := extent;
   point.y := corner;
   REPEAT
      point.x := corner;
      REPEAT
         Calculate;
         point.x := point.x + size
      UNTIL point.x > width;
      point.y := point.y + size
   UNTIL point.y > height;
   WHILE size > 1 DO BEGIN  {tile blocks with successively smaller blocks}
      square := square DIV 3;
      extent := ENTIER(square / 2);
      base.y := corner;
      REPEAT
         base.x := corner;
         REPEAT
            {plot the 8 outer segments of current block}
            point.y := base.y - square; point.x := base.x - square;
                       Calculate;
            point.x := base.x; Calculate;
            point.x := base.x + square; Calculate;
            point.y := base.y; Calculate;
            point.x := base.x - square; Calculate;
            point.y := base.y + square; Calculate;
            point.x := base.x; Calculate;
            point.x := base.x + square; Calculate;
            base.x := base.x + size;
            ch := INCH;
            CASE ch OF
               'S','s' : Save;
               'L','l' : Load
            END
         UNTIL base.x > width;
         base.y := base.y + size
      UNTIL base.y > height;
      size := square;
      corner := extent
   END;
   Save
END.
```

LISTING 2

```
{Save and Load routines for listing 1. They may need modifying if sqinit
 is changed}
VAR filei : FILE OF INTEGER;
    filer : FILE OF REAL;
    packsz,packsize,packcnt,packval : INTEGER;
    buffer : ARRAY[1..sqinit,1..sqinit] OF CHAR;

PROCEDURE Save;
PROCEDURE ReadImage(x,y : INTEGER);
BEGIN
{insert code to read block of data from screen into buffer, top left
 corner at x,y, width and height of sqinit. buffer[1,1] = top left;
 buffer[1,sqinit] = top right; buffer[sqinit,1] = bottom left;
 buffer[sqinit,sqinit] = bottom right}
END;
PROCEDURE PackInit;
BEGIN
   packval := 0;
   packcnt := 16   {integers are 16 bits long}
END;
PROCEDURE PackItem(byte,dummy : INTEGER);
BEGIN
   IF packcnt = 16 THEN BEGIN
      packval := byte;
      packcnt := 8
   END ELSE BEGIN
      packval := (packval * 128) + (byte MOD 128);
      IF byte > 127 THEN
         packval := -1 * packval;
      WRITE(filei,packval);
      PackInit
   END
END;
PROCEDURE PackFinal;
BEGIN
   IF packcnt = 8 THEN BEGIN
      packval := packval * 128;
      WRITE(filei,packval)
   END
END;
PROCEDURE Compact(x0,y0,s0 : INTEGER);
VAR x1,x2,y1,y2,s3,i,j : INTEGER;
    k : CHAR;
    flag : BOOLEAN;
BEGIN
   packsize := packsize + 1;
   flag := FALSE;
   k := buffer[y0,x0];
   j := y0;
   REPEAT              {check if block is of same colour}
      i := x0;
      REPEAT
         IF buffer[j,i] <> k THEN
            flag := TRUE;
         i := i + 1
      UNTIL (i >= x0 + s0) OR flag;
      j := j + 1
   UNTIL (j >= y0 + s0) OR flag;
   IF NOT flag THEN BEGIN
      PackItem(packsize, 3);                {block of one colour}
      PackItem(ORD(buffer[y0, x0]), 8)
```

```
    END ELSE
       IF s0 <> 3 THEN BEGIN
          s3 := s0 DIV 3;                {block not of one colour and}
          x1 := x0 + s3; x2 := x1 + s3; {not of smallest size so descend}
          y1 := y0 + s3; y2 := y1 + s3;
          Compact(x0, y0, s3); Compact(x1, y0, s3); Compact(x2, y0, s3);
          Compact(x0, y1, s3); Compact(x1, y1, s3); Compact(x2, y1, s3);
          Compact(x0, y2, s3); Compact(x1, y2, s3); Compact(x2, y2, s3)
       END ELSE BEGIN
          x1 := x0 + 1; x2 := x1 + 1;    {block not of one colour and}
          y1 := y0 + 1; y2 := y1 + 1;    {smallest size}
          PackItem(4, 3);
          PackItem(ORD(buffer[y0, x0]), 8);
          PackItem(ORD(buffer[y0, x1]), 8);
          PackItem(ORD(buffer[y0, x2]), 8);
          PackItem(ORD(buffer[y1, x0]), 8);
          PackItem(ORD(buffer[y1, x1]), 8);
          PackItem(ORD(buffer[y1, x2]), 8);
          PackItem(ORD(buffer[y2, x0]), 8);
          PackItem(ORD(buffer[y2, x1]), 8);
          PackItem(ORD(buffer[y2, x2]), 8)
       END;
    packsize := packsize - 1
END;
BEGIN
    REWRITE(filer,'  IMAGE    .DT2');
    WRITE(filer,scale.re,scale.im,max.re,max.im,min.re,min.im);
    IF map = 2 THEN
       WRITE(filer,konst.re,konst.im);
    RESET(filer,'  IMAGE    .DT2');
    REWRITE(filei,'  IMAGE    .DAT');
    WRITE(filei,map,minlevel,maxlevel,limit);
    WRITE(filei,corner,size,extent,square,base.x,base.y);
    PackInit;
    point.y := 0;
    REPEAT
       point.x := 0;
       REPEAT
          packsize := -1;
          ReadImage(point.x , point.y);
          Compact(1 , 1 , sqinit);
          point.x := point.x + sqinit
       UNTIL point.x >= width;
       point.y := point.y + sqinit
    UNTIL point.y >= height;
    PackFinal;
    RESET(filei,'  IMAGE    .DAT')
END;

PROCEDURE Load;
PROCEDURE UnpackInit;
BEGIN
    packcnt := 16;
    READ(filei,packval)
END;
FUNCTION UnpackItem(dummy : INTEGER) : INTEGER;
BEGIN
    IF packcnt = 16 THEN BEGIN
       UnpackItem := ABS(packval) DIV 128;
       packcnt := 8
    END ELSE BEGIN
       IF packval < 0 THEN
          UnpackItem := (packval MOD 128) + 128
```

```
      ELSE
         UnpackItem := packval MOD 128;
      UnpackInit
   END
END;
PROCEDURE Expand(x0,y0,s0 : INTEGER);
VAR x1,x2,y1,y2,s3 : INTEGER;
BEGIN
   packsize := packsize + 1;
   IF packsz = packsize THEN BEGIN
         Rfill(x0,y0,s0,s0,UnpackItem(8));
         packsz := UnpackItem(3)
   END ELSE
      IF s0 = 3 THEN BEGIN
         x1 := x0 + 1; x2 := x1 + 1;
         y1 := y0 + 1; y2 := y1 + 1;
         Plot(x0,y0,UnpackItem(8));
         Plot(x1,y0,UnpackItem(8));
         Plot(x2,y0,UnpackItem(8));
         Plot(x0,y1,UnpackItem(8));
         Plot(x1,y1,UnpackItem(8));
         Plot(x2,y1,UnpackItem(8));
         Plot(x0,y2,UnpackItem(8));
         Plot(x1,y2,UnpackItem(8));
         Plot(x2,y2,UnpackItem(8));
         packsz := UnpackItem(3)
      END ELSE BEGIN
         s3 := s0 DIV 3;
         x1 := x0 + s3; x2 := x1 + s3;
         y1 := y0 + s3; y2 := y1 + s3;
         Expand(x0,y0,s3); Expand(x1,y0,s3); Expand(x2,y0,s3);
         Expand(x0,y1,s3); Expand(x1,y1,s3); Expand(x2,y1,s3);
         Expand(x0,y2,s3); Expand(x1,y2,s3); Expand(x2,y2,s3)
      END;
   packsize := packsize - 1
END;
BEGIN
   RESET(filei,'  IMAGE   .DAT');
   READ(filei,map,minlevel,maxlevel,limit);
   READ(filei,corner,size,extent,square,base.x,base.y);
   UnpackInit;
   packsz := UnpackItem(3);
   point.y := 0;
   REPEAT
      point.x := 0;
      REPEAT
         packsize := -1;
         Expand(point.x, point.y, sqinit);
         point.x := point.x + sqinit
      UNTIL point.x >= width;
      point.y := point.y + sqinit
   UNTIL point.y >= height;
   RESET(filer,'  IMAGE   .DT2');
   READ(filer,scale.re,scale.im,max.re,max.im,min.re,min.im);
   IF map = 2 THEN
      READ(filer,konst.re,konst.im)
END;
```

TABLE 1

Comparison of the various compression methods operating on several of the images illustrated in "The Beauty of Fractals" by H.-O. Peitgen and P.H. Richter.

	standard	map 36	map 38	map 42	map 44	map 45
coordinate						
min x	-2.03508	-0.75104	-0.74758	-0.74591	-0.74554	-0.74547
max x	0.54204	-0.74080	-0.74624	-0.74448	-0.74505	-0.74538
min y	-1.20954	0.10511	0.10671	0.11196	0.11288	0.11298
max y	1.28375	0.11536	0.10779	0.11339	0.11324	0.11304
iterations						
min	1	1	1	1	1	1
max	254	254	762	254	254	254
mapping	linear	linear	banded	linear	linear	linear
method 1a						
size (k)	67.00	211.125	454.625	195.25	283.25	228.75
%	18.67	58.84	126.71	54.42	78.94	63.75
method 1b						
size (k)	66.00	177.275	369.625	164.875	235.125	189.375
%	18.39	49.44	103.02	45.95	65.53	52.78
method 1c						
size (k)	76.375	173.50	343.50	163.625	224.625	184.625
%	21.29	48.36	95.74	45.60	62.60	51.46
method 1d						
size (k)	103.50	182.50	323.75	175.00	224.50	191.375
%	28.85	50.86	90.23	48.77	62.57	53.34
method 1e						
size (k)	171.50	225.00	322.75	220.00	253.375	230.625
%	47.80	62.71	89.95	61.32	70.62	64.28
method 1f						
size (k)	70.125	149.625	275.125	138.125	187.125	158.25
%	19.54	41.70	76.68	38.50	52.15	44.11
method 1g						
size (k)	57.625	145.625	280.625	131.50	186.00	154.375
%	16.06	40.59	78.21	36.65	51.84	43.03
method 2						
size (k)	141.375	229.00	382.125	219.75	269.75	235.875
%	39.40	63.82	106.50	61.25	75.18	65.74
method 3a						
size (k)	93.00	198.875	341.25	188.875	246.75	219.00
%	25.92	55.43	95.11	52.64	68.77	61.04
method 3b						
size (k)	82.625	182.00	316.875	171.875	227.00	199.50
%	23.03	50.72	88.32	47.90	63.27	55.60
method 3c						
size (k)	54.875	141.50	275.00	133.00	179.875	152.25
%	15.29	39.44	76.64	37.07	50.13	42.43
method 3d						
size (k)	47.75	124.875	247.375	117.25	157.125	132.75
%	13.31	34.80	68.95	32.68	43.79	37.00
method 3e						
size (k)	47.125	119.875	248.875	113.25	151.375	127.75
%	13.13	33.41	69.36	31.56	42.19	35.60
method 4						
size (k)	52.125	161.75	304.125	140.375	193.25	162.375
%	14.53	45.08	84.76	39.12	53.86	45.26

mapping - linear = 1 iteration per colour [colour 255 not used, colour 0 = black]
banded = 3 iterations per colour (762/254)
size (k) - size of compressed data (in kilobytes). Uncompressed = 358.8K
% - percentage of uncompressed data size

method 1a - RLC (8 bit count)
method 1b - RLC (5 bit count)
method 1c - RLC (4 bit count)
method 1d - RLC (3 bit count)
method 1e - RLC (2 bit count)
method 1f - RLC (4 bit count, 4 bit subcount)
method 1g - RLC (5 bit count, 4 bit subcount)

method 2 - RLC (plane)
method 3a - Tiling
method 3b - Tiling with bit compression
method 3c - Extended Tiling
method 3d - Full Tiling (4 bit offset)
method 3e - Full Tiling (3 bit offset)
method 4 - Plane Tiling

LISTING 3

```
{Bit packing enhancements for listing 2. Machine code is recommended for
 this as Pascal version is SLOW}
power ARRAY[1..15] OF INTEGER;
power[1] := 1; power[2] := 2; power[3] := 4; power[4] := 8;
power[5] := 16; power[6] := 32; power[7] := 64; power[8] := 128;
power[9] := 256; power[10] := 512; power[11] := 1024; power[12] := 2048;
power[13] := 4096; power[14] := 8192; power[15] := 16384;

PROCEDURE PackItem(byte,bitcount : INTEGER);
VAR i : INTEGER;
BEGIN
   FOR i := bitcount DOWNTO 1 DO BEGIN
      packcnt := packcnt - 1;
      IF packcnt = 0 THEN BEGIN
         IF packval >= 16384 THEN BEGIN
            packval := (packval - 16384) * 2;
            IF byte >= power[i] THEN BEGIN
               packval := packval + 1; byte := byte - power[i]
            END;
            packval := -1 * packval
         END ELSE BEGIN
            packval := packval * 2;
            IF byte >= power[i] THEN BEGIN
               packval := packval + 1; byte := byte - power[i]
            END
         END;
         WRITE(filei,packval);
         PackInit
      END ELSE BEGIN
         packval := packval * 2;
         IF byte >= power[i] THEN BEGIN
            packval := packval + 1; byte := byte - power[i]
         END
      END
   END
END;
PROCEDURE PackFinal;
BEGIN
   PackItem(0,packcnt)
END;
FUNCTION UnpackItem(bitcount : INTEGER) : INTEGER;
VAR i,temp : INTEGER;
BEGIN
   temp := 0;
   FOR i := 1 TO bitcount DO BEGIN
      temp := temp * 2;
      IF packcnt = 0 THEN
         UnpackInit;
      IF packcnt = 16 THEN BEGIN
         IF packval < 0 THEN BEGIN
            packval := -1 * packval; temp := temp + 1
         END
      END ELSE BEGIN
         IF packval >= power[packcnt] THEN BEGIN
            temp := temp + 1; packval := packval - power[packcnt]
         END
      END;
      packcnt := packcnt - 1
   END;
   UnpackItem := temp
END;
```