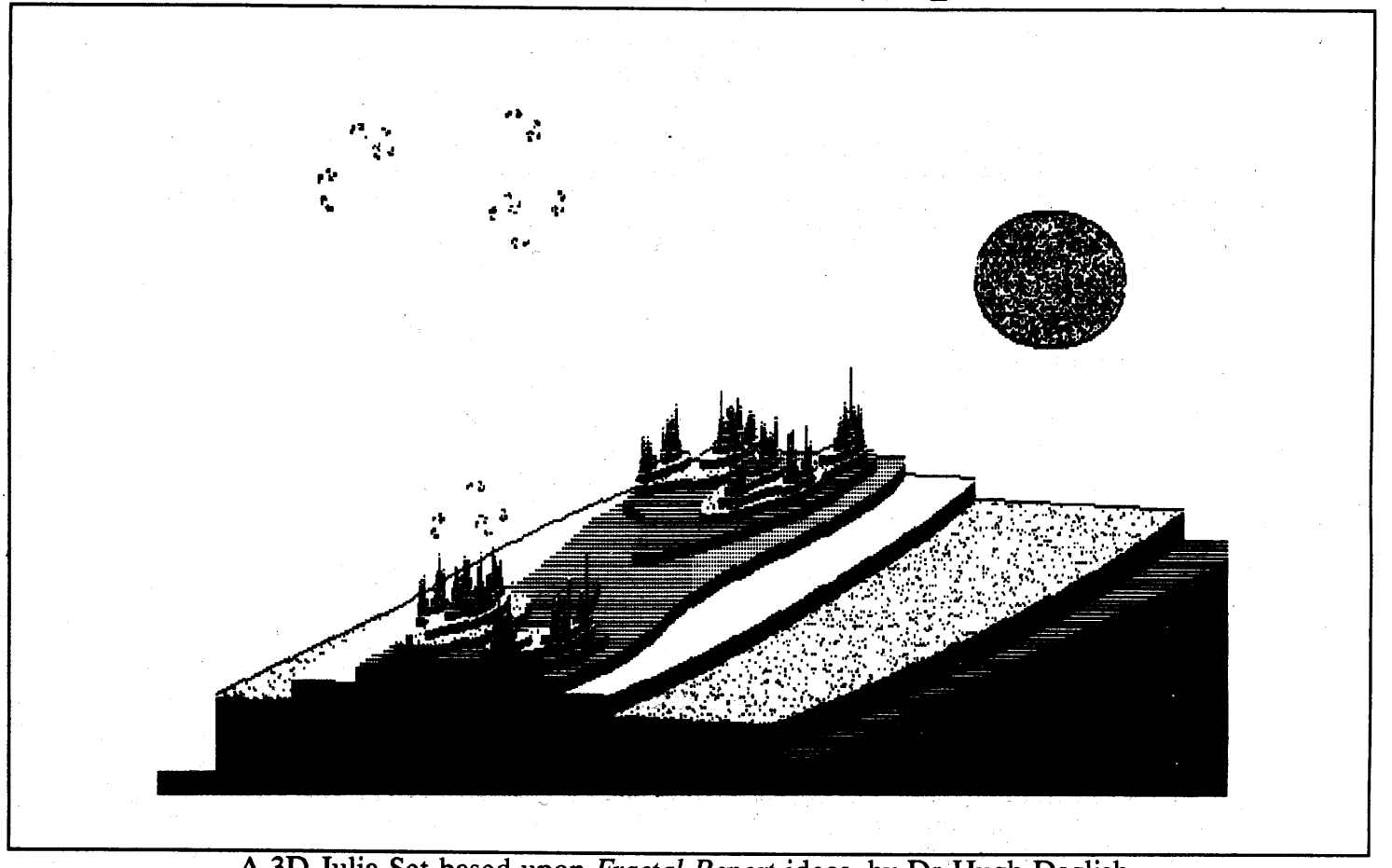
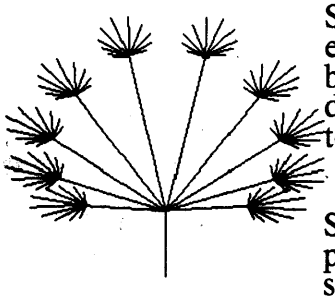
FRACTAL REPORT 12



A 3D Julia Set based upon *Fractal Report* ideas, by Dr Hugh Daglish.



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 2 no 12 First published December 1990. ISSN applied for.

SIMPLE SPEEDUPS

Paul McGilly
43 Wellington Road, Hatch End, Pinner, Middx, HA5 4NF
(PM5 @ YORK.VAXA)

On most microprocessors multiplication is an extremely time consuming operation compared with addition or memory accesses. (eg. 70 cycles vs 4 cycles on a 68000). Therefore one should minimise the number of multiplications in a program. This can be illustrated by the following pseudocode for the inner loop of the Mandelbrot algorithm :

6 multiplications

```
REPEAT
   TMP=X*X-Y*Y+XC
   Y=2*X*Y+YC
   X=TMP
   C=C+1
UNTIL (C>MAXIT OR X*X+Y*Y>4)
```

3 multiplications

```
XTMP=X*X : YTMP=Y*Y
REPEAT
   Y=(X+X)*Y+YC
   X=XTMP-YTMP+XC
   C=C+1
   XTMP=X*X : YTMP=Y*Y
UNTIL (C>MAXIT OR XTMP+YTMP>4)
```

Changes such as the above can produce a speedup of upto 40% depending upon the language and processor.

Similar speedups can be made to most of the programs that have been published in this journal. The general technique is to remove all constant calculations from the inner loop and retain only a single copy of any repeated code. The Henon map program by Andy Lunnes from Issue 4 is now used to illustrate this point :

Original Program
(Adapted for Atari ST, Fastbasic)

```
GRAFRECT 0,0,640,400
CLG 0
INPUT A
FOR XS=-0.1 TO 0.8 STEP 0.05
FOR YS=-0.1 TO 0.8 STEP 0.05
X=XS
Y=YS
I=1
REPEAT
   XX=X*COS(A)-(Y-X*X)*SIN(A)
   Y=X*SIN(A)+(Y-X*X)*COS(A)
   X=XX
   PLOT 150*X+320,150*Y+200
   I=I+1
UNTIL I>1000 OR X>1000 OR Y>1000
        OR X<-1000 OR Y<-1000
NEXT YS
NEXT XS
```

Optimised Program

```
GRAFRECT 0,0,640,400
CLG 0
INPUT A
AC=COS(A) : AS=SIN(A)
FOR XS=-0.1 TO 0.8 STEP 0.05
FOR YS=-0.1 TO 0.8 STEP 0.05
X=XS : Y=YS
I=1
REPEAT
YXX=Y-X*X
Y=X*AS+YXX*AC
X=X*AC-YXX*AS
PLOT 150*X+320,150*Y+200
I=I+1
UNTIL I>1000 OR ABS(X)>1000 OR
   ABS(Y)>1000
NEXT YS
NEXT XS
```

The speedup achieved in this case is over 50%.

Faster and Nicer Fractal Sets

Jules Verschueren
Binnenstraat 53
B - 3020 Veltem
Belgium

The Basic program corresponding to this article was developed with the idea to describe techniques -not yet published in "Fractal Report"- in order to increase both speed and beauty of fractal sets. Most of these methods are my own original work.
Although the classic Mandelbrot/Julia algorithm is very familiar to all of you, the pictures you can produce with this program are probably quite different from what you are used to (see figures).
The program contains many remarks in order to minimize the following explanation

A. A faster picture production is achieved by selection of the

1) desired screen resolution / reduction,
2) length / height aspect correction (both 1 and 2 are indicated by the picture window lines and are described in routines "graphics" and "adaptgrid"),
3) symmetry use - described in "checksymmetry": 1/2 to 3/4 of all calculations might be saved,
4) "attraction-stop" technique.

Item 4) deserves more explanation as it is probably new to most of the readers and saves most of the calculation time (at least when part of the picture belongs to the set). It also allows several ways of making the internals of the set more attractive.
I discovered this type of technique via experimentation on the iteration process itself, eg. by watching the evolution of Z^2, both during the iteration for one starting point as well as iterations of neighbouring points. This evolution can be followed by printing the consecutive values of Z^2 or by pixeling the corresponding positions (as described in Fractal Report 1, page 16: Z-Trajectories; by Ed Hersom). All points starting within the 'M' set are finally attracted to 1 point (for starting point with origin in large cardoid) or a stable cycle of 2 (origin in large circle left of cardiod and with centre (-1,0), 3 (origin in top and bottom blob on cardoid), 4 (origin in second circle left of caroid), etc... points. You can easily verify this with this program as you can choose the "Maximum values to check for a stable Z-Cycle" (ie. variable MaxCycle). Try the M-set with MaxCycle values of 1, 2, 3, 4...
There will be no further speed gain after checking for a 3-cycle in the M-set as keeping track of and checking for an additional match for all points in the set -see routine "testcycle"- will take at least as long as the early iteration stop for a few additional points. The situation is obviously totally different when you make a zoom of one of these sections, or when you produce a Julia set from one of these points (see below).
Further speed is gained by optimizing the calculations (especially taking care not to duplicate multiplications) and using one and the same iteration routine for both the Mandelbrot and Julia set. This prevents checking for the type of set during each and every iteration step.

The "testcycle" routine is only applicable for points in the set and is therefore only turned on when the previously calculated point was inside the set. The last Z^2 values are stored in the "CycleZ" array and the "CycleZ(Iter MOD MaxCycle)=Z2" statement replaces the oldest Z^2 with the new one. The "FOR" loop tests whether the new Z^2 has been encountered before, ie. a stable cycle has been found and the iteration can be stopped. This test is not necessarily an

exact match (ie. DifZ=0), but can be a very small difference - possibly adapted for grid size, number of iterations or Mandelbrot/Julia type of set. Changing DifZ will have an influence on the pattern that is produced inside a set and this influence is larger for Julia sets.
Consider eg. Fig. 4, a Julia set with C=-0.11+0.6557i: this point lies near the root of the top blop of the M-set and if iterated there, settles down to a stable 3-cycle with attraction coordinates (-.23+.174i), (-.087+.575i) and (-.433+.555i). Although it may take several hundreds of iterations before the cycle is absolutely stable -as judged from identical consecutive Z^2's, and thus dependent on calculation accuracy- a good approximation (controlled by variable "DifZ") is usually achieved much quicker. Since no or only minimal changes can occur on further iteration, further calculations become unnecessary and the iteration is stopped. One could think of the M-set as a large bassin of different attractors, while Julia sets behave quite differently. As in Fig 4 (using "Attraction value" or Z^2) each bassin is made up of one color and all these points are attracted to one and the same Z^2 or attraction point, equal to (one of) the corresponding M-set final values. Not surprising, the number of different bassins/colors/dragon arms or heads corresponds to the attraction cycle number. Therefore "MaxCycle" in Fig 1, 4 and 5 must be respectively 2, 3 and 11 to give the highest speed gain. Both a lower (no early calculation stops) or higher value would have slowing effects as unnecessary checks are performed.

B. The internal set coloring techniques

These are completely linked to the above "Attraction-Stop" method with exception of the "Minimum Z" (InType 3) technique which has been described in Algorithm 1.1, 1989, page 9: Inside the Mandelbrot Set; by CA Pickover. This method just keeps track of the smallest Z^2 produced in the iteration process and colors the pixels accordingly by multiplying with some value -"InCont". Fig 3, a Mandelbrot type set with ZRealStart=1.4 is an example of this technique (InCont=400).
The second method -"Attraction value"- colors the pixel according to the final/ stable Z^2 value. As this is the same in one particular Julia bassin, the different bassins can all be colored differently. As it takes much longer for some points in the bassin to be attracted, it is usually necessary to continue the iterations until no change in consecutive Z^2's is found (ie. DifZ=0).
The first method -"Attraction count"- (InType 1) is really original. Just use the number of iterations it takes to achieve a stable Z^2 cycle and use it to color the pixel. The results can be quite remarkable as is visible from Fig 1 and 2. Even using the plain count ("Iter") to attraction (Fig 1, InCont=999) can transform a simple Julia set into a very attractive and colorful picture. It even shows consecutive bifurcations (doubling of spikes) from the center going outwards. The speed difference for picture production with the "Attraction-Stop" as opposed to the classic iteration method is quite extreme. The variable "Contour=InCont/InMax" is introduced to give more color possibilities: values lower than InMax will produce larger bands of the same coloring as eg. shown in Fig 2, the "classic?" Mandelbrot set.
The "calccolor" routine actually calculates the color for each pixel. If the point belongs to the set, "MaxIter" is a constant number in each of the 3 methods and as changing this value slightly will usually not influence the second term of the color calculation (attraction-stop occurs after a fixed number of iterations) this can be used to adapt the color for a particular region or bassin according to your liking. Try adding 1 to MaxIter in Fig 1: blue should become green, green --> cyan --> red --> magenta --> yellow --> white --> blue.

I hope you like the figures and have lots of fun with the program.

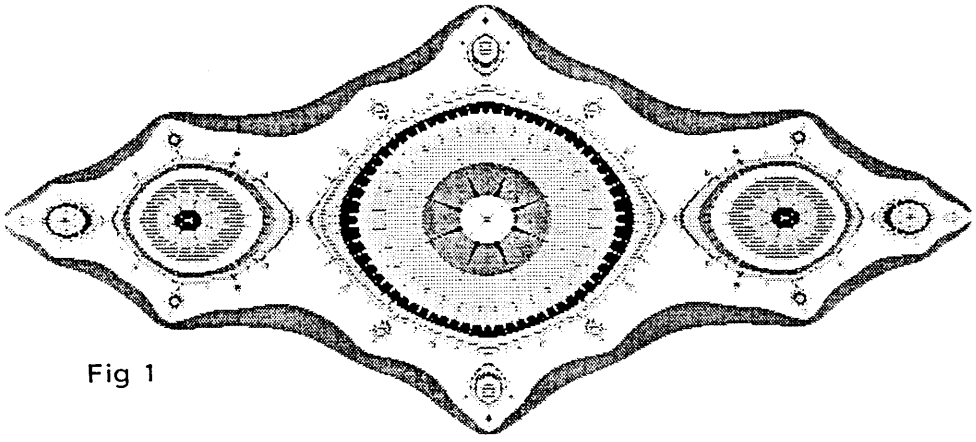


Fig 1

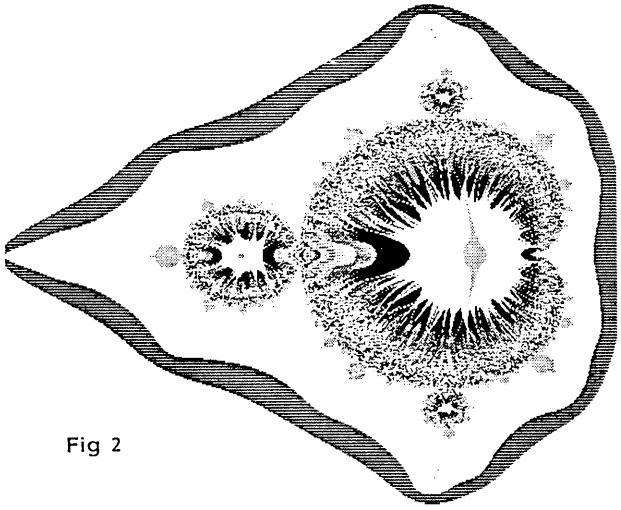


Fig 2

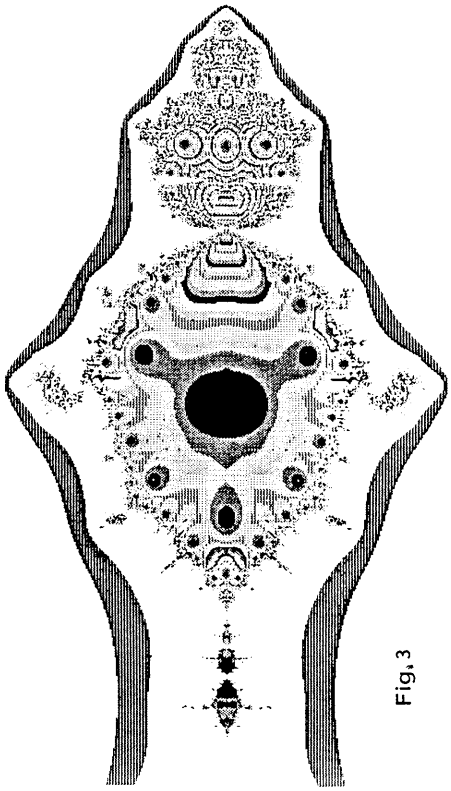


Fig. 3

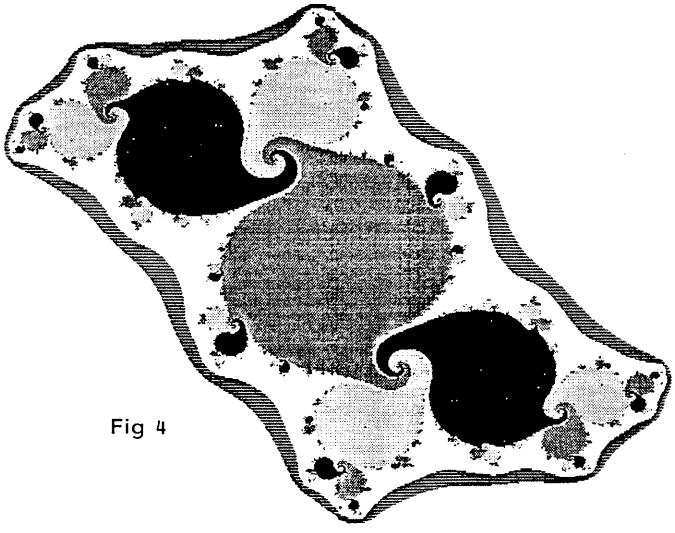


Fig 4

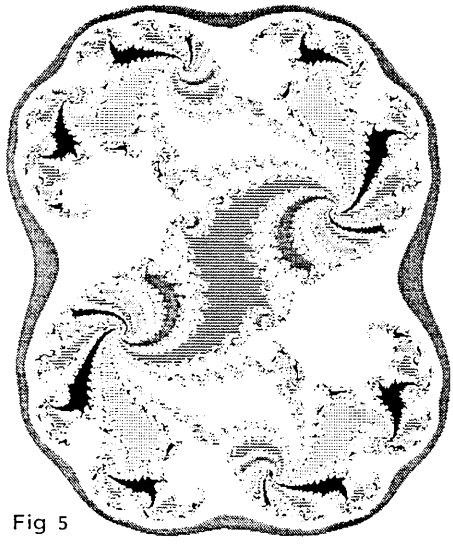


Fig 5

```
Rem  Quick basic Mandelbrot (with starting ZReal/ZImag <=> 0) / Julia program
rem  (c) L.J. Verschueren, Binnenstraat 53, B-3008 Veltem, Belgium;   30/7/90
rem  Can directly be used with CGA, EGA or VGA card
rem  With fast Set calculation and 3 different Internal Set coloring techniques

DEFINT I,K,M,O,S,X-Y                    'all variables starting with these letters are integers
'DEFDBL C-H,L,N,R,T,Z                   'double precision in the calculations - use when Length<0.001

Mode=12: MaxMode=0                                      'first try VGA, then EGA, then CGA
ON ERROR GOTO screenerror
SCREEN Mode: MaxMode=Mode                               'select highest possible resolution

dataentry:
  SCREEN 0: WIDTH 80                                    'text mode
  PRINT"Fractal Type (0=Julia, 1=Mandel): ";: Type$=INPUT$(1): PRINT Type$
   IF Type$<>"0" AND Type$<>"J" AND Type$<>"j" THEN Mandel=1 ELSE Mandel=0
  INPUT"Enter Window Length and Height (separated by a comma):", Length, Height
  INPUT"Low Window X and Y Start coordinates:", LowX, LowY
  INPUT"Escape Value for Z*Z and Maximum Iterations:", Escape2, MaxIter
  IF Mandel THEN                                        'possibility for Mandel type sets with starting Z<>0
    INPUT"Z-Real and Z-Imag Start values (0,0 = classic Mandel set):", ZRealStart, ZImagStart
   ELSE INPUT"Real and Imaginary Constant:", CReal, CImag      'Constant from Z*Z+C for Julia sets
  END IF
  PRINT"Select High or Low resolution (H/L): ";: Res$=INPUT$(1): PRINT Res$
   IF Res$="l" OR Res$="L" THEN
     IF Mode>2 THEN Mode=7 ELSE Mode=1                  'resolution 320x200
     INPUT" Additional Reduction factor (0.1-1): ", Reduction
     IF Reduction<0.1 THEN Reduction=0.1 ELSE IF Reduction>1 THEN Reduction=1
    ELSE Mode=MaxMode: Reduction=1                      'highest resolution
   END IF
  PRINT"Internal set Contours (Y/N)? ";: InCont$=INPUT$(1): PRINT InCont$
   IF InCont$<>"N" AND InCont$<>"n" THEN InSet=1 ELSE InSet=0
  IF InSet THEN
    InMax=999                 'changing this arbitrary value will influence the "internal" pattern from InType 1
    INPUT"Maximum values to check stable Z-Cycle (2-20):", MaxCycle      'stable attraction points to be checked
     REDIM CycleZ(MaxCycle)                             'array to store Z*Z
    PRINT"Choose Internal contours Type (0-3):"         'coloring within the set
    PRINT"  0=None, 1=Attraction Count, 2=Attraction Value, 3=Minimum Z    : ";: InType$=INPUT$(1): PRINT InType$
     IF InType$<"1" AND InType$>"3" THEN InType=0 ELSE InType=VAL(InType$)
    IF InType THEN PRINT"Enter Internal contours (1-";InMax;: INPUT")"; InCont
    IF InType=1 THEN Contour=InCont/InMax
  END IF
  IF Mandel OR InType<>2 THEN DifZ=.0001*Length ELSE DifZ=0   'testing of stable cycle in set; adapted to zoom window
                                                              'color technique 2 in Julia set uses exact match
graphics:
  SCREEN Mode                                           'graphics screen according to hardware and selected resolution
  IF Mode=1 OR Mode=7 THEN ScreenX=319 ELSE ScreenX=639
  IF Mode<9 THEN ScreenY=199 ELSE IF Mode=9 THEN ScreenY=349 ELSE ScreenY=479
  IF Mode=1 THEN OUT &H3D9,16                           'CGA color palet 2: green, red, yellow
  WINDOW(0,0)-(ScreenX,ScreenY): CLS
  IF Mode>2 THEN                                        'EGA/VGA colors
    SetCol=14: MaxCol=7: StartCol=9: Red=4              'only high intesity colors
   ELSE
    IF Mode=2 THEN                                      'CGA black/white 640x200
      SetCol=1: StartCol=0: MaxCol=2: Red=1             'black also counted as a color
     ELSE SetCol=3: StartCol=1: MaxCol=3: Red=2         'CGA 320x200
    END IF
  END IF
  Iter1=6: Iter2=5                                      'for contour color outside the set

 checksymmetry:                                         'with regard to axes and origin (0,0)
  IF Height=-2*LowY THEN MirrorTB=1 ELSE MirrorTB=0     'symmetry Top/Bottom
  IF Mandel THEN IF ZRealStart<>0 AND ZImagStart<>0 THEN MirrorTB=0  'no symmetry if start not on an axis
  Sym=0: MirrorLR=0                                     'used for Julia sets only
  IF Mandel=0 THEN                                      'Julia set
    IF Length=-2*LowX THEN MirrorLR=1                   'symmetry Left/Right
    IF MirrorTB AND MirrorLR THEN Sym=1                 '(0,0) point symmetry
    IF CImag<>0 THEN MirrorTB=0: MirrorLR=0             'only point symmetry if CImag<>0
  END IF

 adaptgrid:                                             'every reduction in grid size means less calculations
  XStart=0: YStart=0: Aspect=Length*.75/Height          'to allow images <> screen Y/X ratio (=3/4)
  IF Aspect<1 THEN XStart=CINT((ScreenX-ScreenX*Aspect)/2)
  IF Aspect>1 THEN YStart=CINT((ScreenY-ScreenY/Aspect)/2)
  XStart=XStart+CINT((ScreenX\2-XStart)*(1-Reduction)): YStart=YStart+CINT((ScreenY\2-YStart)*(1-Reduction))
  DX=Length/(ScreenX+1-2*XStart): DY=Height/(ScreenY+1-2*YStart)        'grid step sizes
  LINE(XStart-2,YStart-2)-(ScreenX-XStart+2,YStart-2): LINE-(ScreenX-XStart+2,ScreenY-YStart+2)
  LINE-(XStart-2, ScreenY-YStart+2): LINE-(XStart-2, YStart-2)          'lines around picture
  IF Mandel THEN CReal=LowX-DX ELSE ZRealStart=LowX-DX                  'DX again added in "FOR X"-loop
  IF MirrorTB THEN YEnd=ScreenY\2: ScreenY=ScreenY-1: ELSE YEnd=ScreenY-YStart
  IF Sym OR MirrorLR THEN XEnd=ScreenX\2: ScreenX=ScreenX-1: ELSE XEnd=ScreenX-XStart
```

```
  startcalc:
   Time0=TIMER                              'start timing
   FOR X=XStart TO XEnd
     Iter=0                                 'ensure that CheckDifZ will be 0 at start of new column (Y)
     IF Mandel THEN CReal=CReal+DX: CImag=LowY-DY: ELSE ZRealStart=ZRealStart+DX: ZImagStart=LowY-DY
     Z2RealStart=ZRealStart*ZRealStart
     FOR Y=YStart TO YEnd
       IF InSet AND Iter>=MaxIter THEN CheckDifZ=1 ELSE CheckDifZ=0
         'testing for stable (Z*Z)-cycle only starts when requested and when previous point belonged to the set
       Iter=0: Z2Small=Escape2: ZReal=ZRealStart
       IF Mandel THEN CImag=CImag+DY ELSE ZImagStart=ZImagStart+DY
       ZImag=ZImagStart: Z2Real=Z2RealStart: Z2Imag=ZImag*ZImag: Z2=Z2Real+Z2Imag
       FOR I=0 TO MaxCycle-1: CycleZ(I)=Z2: NEXT I      'initiate array with initial Z*Z value
       GOSUB iterate: GOSUB calccolor                   'iterate, calculate color and put pixel(s)
     NEXT Y
   NEXT X
   LOCATE ,1: PRINT CINT(TIMER-Time0);"sec"     'show calculation time
   Wait$=INPUT$(1)                              'wait until key pressed - possibility for PrtScreen/Restart/Exit
   IF Wait$<>Chr$(27) THEN GOTO dataentry       'restart
END                                             'exit program when "Esc" key is pressed


iterate:                                        'same routine used for Mandelbrot and Julia sets
  WHILE Iter<MaxIter AND Z2<Escape2
    NewZReal=Z2Real-Z2Imag + CReal: ZImag=(ZReal+ZReal)*ZImag + CImag: ZReal=NewZReal
    Z2Real=ZReal*ZReal: Z2Imag=ZImag*ZImag: Z2=Z2Real+Z2Imag
    Iter=Iter+1: IF CheckDifZ THEN GOSUB testcycle
  WEND
 RETURN

calccolor:                                      'determine color of pixel
  IF Iter>=MaxIter THEN                         'point belongs to the set
    IF InType=1 THEN Iter=MaxIter+CINT(Iter*Contour)
    IF InType=2 THEN Iter=MaxIter+CINT(Z2*InCont)
    IF InType=3 THEN Iter=MaxIter+CINT(Z2Small*InCont)
    IF InType=0 OR InSet=0 THEN Kolor=SetCol ELSE Kolor=Iter MOD MaxCol + StartCol
   ELSE IF Iter<=Iter1 AND Iter>Iter2 THEN Kolor=Red ELSE Kolor=0       'point outside set, method 1
'  ELSE IF Iter<=Iter1 AND Iter>Iter2 THEN Kolor=Iter MOD MaxCol +StartCol ELSE Kolor=0 'outside set, method 2
  END IF
 putpixel:
  IF Kolor THEN                                 'put pixel only when Kolor<>background
    PSET(X, Y),Kolor: IF MirrorTB THEN PSET(X, ScreenY-Y),Kolor
    IF Sym THEN PSET(ScreenX-X,ScreenY-Y),Kolor: IF MirrorLR THEN PSET(ScreenX-X,Y),Kolor   'only for Julia set
  END IF
 RETURN

testcycle:
  IF InType=3 THEN IF Z2Small>Z2 THEN Z2Small=Z2      'keep smallest Z*Z
  FOR I=0 TO MaxCycle-1                               'check for stable attraction cycle in the set
    IF ABS(Z2-CycleZ(I))<=DifZ THEN I=MaxCycle: IF InType=1 THEN Iter=Iter+MaxIter ELSE Iter=MaxIter
  NEXT I
  CycleZ(Iter MOD MaxCycle)=Z2                        'keep last Z*Z value(s)
 RETURN                                               'if stable cycle found then stop iteration via Iter

screenerror:                                          'Err=5 : illegal function call
  IF MaxMode=0 THEN
    IF Mode=12 THEN Mode=9 ELSE IF Mode=9 THEN Mode=2 ELSE PRINT:PRINT"SORRY, NO GRAPHICS":STOP
    RESUME                                            'try lower screen Mode
   ELSE RESUME NEXT                                   'continue if other error detected
  END IF


'suggestions:
'Julia, Length=3.22: Height=1.6: LowX=-1.61: LowY=-.8: Escape2=100: MaxIter=200: CReal=-1: CImag=0
      'Resolution EGA: Z-cycle=2: Attraction count: Contours=999; Fig 1
'Mandelbrot, Length=2.6: Height=2.5: LowX=-2: LowY=-1.25: Escape=100: MaxIter=500: ZReal=0: ZImag=0
      'Resolution EGA: Z-Cycle=3 : Attraction count: Contours=170; Fig 2
'Mandelbrot, Length=.88: Height=.56: LowX=-1.43: LowY=-.28: Escape=100: MaxIter=202: ZReal=1.4: ZImag=0
      'Resolution EGA: Z-Cycle=2 : Minimum Z: Contours=400 (Iter1=8, Iter2=7); Fig 3
'Julia, Length=2.6: Height=2.26: LowX=-1.3: LowY=-1.13: Escape=100: MaxIter=802: CReal=-.11: CImag=.6557
      'Resolution EGA: Z-Cycle=3: Attraction Value: Contours=29; Fig 4
'Julia, Length=1.8: Height=2.3: LowX=-0.9: LowY=-1.15: Escape=100: MaxIter=999: CReal=.32: CImag=.043
      'Resolution EGA: Z-Cycle=11: Attraction Value: Contours=30; Fig 5
```

AREA PRESERVING MAPPINGS

Paul Gailiunas.

In 2-D any point can be specified by two coordinates (x,y), and it can be mapped to any other point by a rule which in general looks like:

$$x = f(x,y)$$
$$y = g(x,y).$$

Interesting graphics can be produced by plotting points as this mapping is applied repeatedly. Many kinds of behaviour are possible, but if mappings are restricted to those which preserve areas then cases where parts of the plane either shrink to nothing or expand to infinity are excluded. In mathematical terms this requires that

$$\left| \frac{\partial f}{\partial x} \cdot \frac{\partial g}{\partial y} - \frac{\partial g}{\partial x} \cdot \frac{\partial f}{\partial y} \right| = 1$$

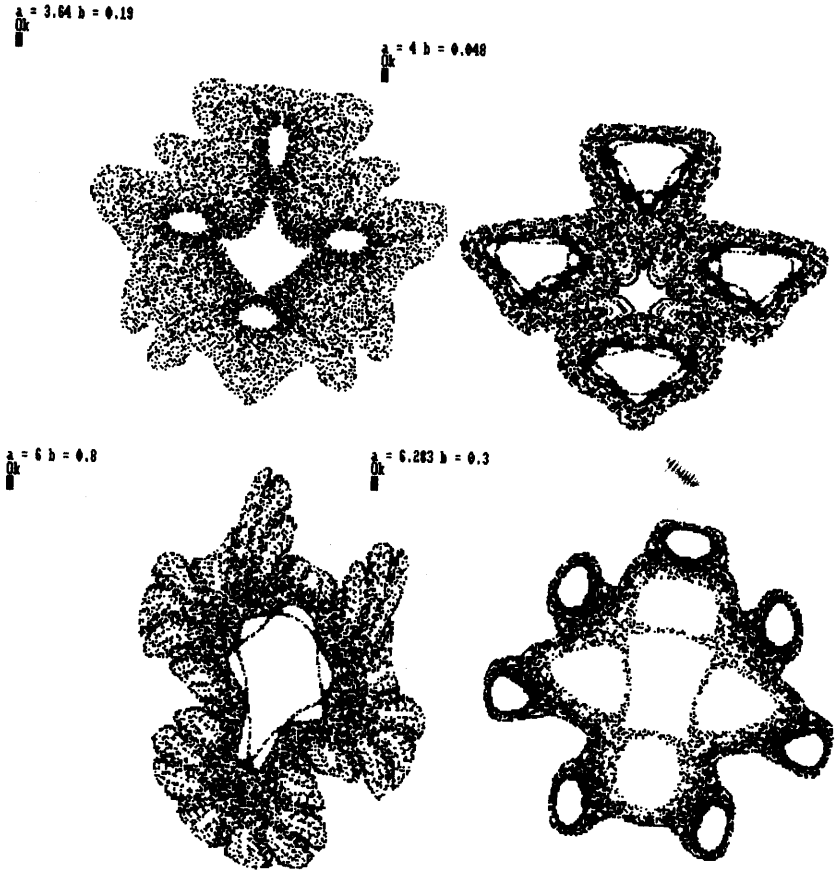
Some of the best known area preserving mappings are Martin's Mappings:

$$x = y + f(x)$$
$$y = a - x,$$

and one of the simplest has f(x)=sin(x), but despite its simplicity it still shows much of the behaviour which is typical of area preserving mappings. One of the most obvious things you will notice if you investigate this mapping is that, depending on the value of a and the starting position, (x0,y0), sometimes a shape is repeated four times. If this happens then there will be values for (x0,y0) which produce four dots. These are known as *periodic points* (of period four). Sometimes there will be only one shape, with a single *fixed point* associated with it.

For any value of a it is possible to get a good idea of what is going on by plotting say 200 points starting from various positions in the plane. In fact, because of the periodicity of the sine function, any pattern is repeated at intervals of 2π in either the x or y direction so that only a square of side 2π need be considered. I have used the following BASIC program written for Amstrad PCW using the DWBAS extension (from PCW-World, Cotswold House, Cradley Heath, Warley, West Midlands. B64 7NF), using a screen-dump for varying values of a. With a machine such as an Archimedes I would expect it to be quite feasible to produce an animation. Notice that the sequence repeats as a varies, with periodicity 2π.

```
30 LDW d
75 pi=3.1415926535#: pi2=pi*2
80 k=2*TAN(24*pi/180)
90 magy=35:mag=magy/k
100 offset=40: yoffset=10
120 INPUT "a";a
130 LDW c: LDW h
150 PRINT "a =" a
155 FOR k%=0 TO 5
160 FOR j%=0 TO 5
170 x=j%*pi/3
175 y=k%*pi/3
180 FOR i%=1 TO 200
190 xx%=INT(x*mag + offset)
200 yy%=255-INT(y*magy+yoffset)
210 LDW g,xx%,yy%
220 z=y-SIN(x)
230 y=a-x
240 x=z
241 IF x<0 THEN x=x+pi2:GOTO 241
242 IF y<0 THEN y=y+pi2:GOTO 242
243 IF x>pi2 THEN x=x-pi2:GOTO 243
244 IF y>pi2 THEN y=y-pi2:GOTO 244
250 NEXT i%
260 NEXT j%
270 NEXT k%
```



(If you find it difficult to imagine what happens as the square repeats over the plane, then leave out lines 243 and 244 and allow j% and k% to go to about 17, taking care to adjust the magnification so that it all fits on the screen).

When a=0 there are fixed points at (0,0) and (π,π), and a pair at (0,π) and (π,0) which alternate. There are concentric loops about the fixed points, which are known as *elliptic fixed points*. The other two points lie in between the loops and are called *hyperbolic*. As a increases the loops begin to develop more structure, and there are five elliptic and five hyperbolic points which produce a system of five small loops. As a continues to increase more changes occur until eventually the sequence repeats when a=2π. In general there are three types of behaviour: individual points, which are either fixed or periodic of small period; *invariant curves*, like the concentric loops, which occur near elliptic points; chaotic regions which always occur near hyperbolic points. Chaotic regions may run together, but they can never cross an invariant curve.

Some of these shapes are very like those produced by the Henon map (which is also area preserving), as described by Andy Lunness in *Fractal Report* Issue 4, although in that case the behaviour over the plane is not periodic, and starting points outside a fairly limited area seem to produce divergent behaviour. This suggests the possibility of using an escape-time algorithm to produce a picture of the Julia set for this mapping, which could also be used for an animation.

The patterns produced by some other Martin's mappings (e.g. f(x)=sgn(x)*(b*x+c)) can be seen to consist of elliptic points, which could have high periodicity, with chaotic regions in between. If the starting point lies within the chaotic region then gradually the plane will be filled with dots as the mapping is repeatedly iterated, but the regions with invariant curves will never be included. This explains the characteristic appearance of these pictures.

Another kind of area preserving mapping has the general form:

```
x  =  y - f(x + b*y)
y  =  a - (x + b*y).
```

This is closely related to Martin's mappings, and simply involves replacing x by (x + b*y) in the formula. In the above example for example simply replace lines 220-240 by:

```
220 z=x+b*y
230 x=y-sin(z)
240 y=a-z.
```

This produces much larger regions of chaos for values of b not nearly equal to 0, and it is usually more interesting just to start at (0,0) and let things develop, rather than restricting everything to a 2π square and trying different starting points. In other words just use the normal Martin algorithm with the extra line included. I have found that values around a=4, b=0.2 and a=6, b=0.8 are quite interesting.

The change, x = x + b*y, is a particular example of the general affine transformation:

```
x  =  a*x + b*y + e
y  =  c*x + d*y + f
```

which will preserve areas provided that abs(a*d-b*c)=1. This suggests combining area preserving maps to produce new, more complicated, ones - an idea I have not even begun to explore. For example, what about:

```
x1=y-sgn(x)*(b*x+c)
y1=a-x
x=y1-sin(x1)
y=d-x1
```

just for starters.

Spirals: animal, vegetable, mineral or fractal?

By Nigel Woodhead

The Road to Oz

Our word spiral comes to us from the Latin spiralis - 'continually winding'. The Chambers 20th Century Dictionary gives the following definition:
"a curve (usu. plane), the locus of a point whose distance from a fixed point varies according to some rule as the radius vector revolves."
In this short piece I shall step beyond this strict mathematical definition - using the term spiral as a flag of convenience to float conceptual links between a great variety of phenomena.

Spirals are very pervasive. They occur in weather systems as tornadoes and hurricanes, and in water systems as whirlpools. From the macroscopic scale of galaxies, down to the scale of the subtle fields of force in play between the smallest particles of matter, we have evidence of strikingly similar morphology.

The spiral is not just a familiar talisman of the physical scientist however - biologists also know them well: from bacillus to brachiopod, from protein structure to the skeletons of snails and shellfish. They are also universal in the art of human cultures - as mystical or abstract symbols, pictographic characters, children's doodles, or the merely esthetic (1). A fine example of how man has abstracted spirals from Nature comes from Catalonia - in the megalithic walls of Tarragona are carved spiral characters from the ancient Iberian alphabet. This symbol is still reflected in the crest of modern Catalonia - the snail, of which there is a prolific (and delicious) local species.

But what we are really interested in here is the root of all of these patterns - not where the spiral so much as how the spiral. Answering 'how' questions involves a theory, and preferably a demonstrable and replicable formula. And in looking for these formulae, the spiral hunter soon enters the great sea of fractalia. Because spirals, wheels, chaotic curls and related phenomena can all be produced by sequential operations on equations, e.g. simple affine transformations on sets of constants, and logarithmic sequences involving complex numbers.

The 15 roots of Unity

It has long been known that if you take a complex number raised to its successive powers from 1 to 15 and map those points on the complex plane, the corresponding escape vectors cut up the circle into 15 equal segments of 24 degrees. This rotation is repeated for the next 15 powers, and so on. Similarly, powers 2,4,6,8, also rotate in a repeating double circle, and powers 3,6,9 in a triple cirle, etc.

For some numbers the vector radius increases as the powers increase, and the spiral unwinds outwards and anti-clockwise (see Figure 1). For other numbers the spiral orbit decays into the origin. This effect, when the complex number X is of an order that X**n-C = 0, i.e. the orbit neither escapes or decays, is known as the unit circle, and the power vectors are known as the roots of unity.

The spirals of shells such as those of ammonites and snails can be mimicked using variations on logarithmic sequences (2). The present day Nautilus is a particularly interesting case in that it is mapped by the equation of the unit circle - growing outwards in a rotations of 15 segments. In other words, we have a perfect example of an organism whose morphology mirrors not only its own ontogeny, but also a fundamental rule of mathematical regularity.

Mandelbrot Spirals

A number of similar 'escaping phenomena' thrive in the shallow currents around the deep lagoon of the Mandelbrot Set itself. Figure 2 shows a set of 7 spiral curls located off the major 'northern island' of the M Set (x = 0.120762 to 0.162787, y = 0.635716 to 0.667330).

Figure 3 (x = 0.128775 to 0.135012, y = 0.635716 to 0.667330) shoows an area where a number of attractors with different periodicity can be seen. In fact, each region of the complex planes is characterised by a different periodicity (3).

Figure 4 (x = -0.252662 to -0.252629, y = 0.650062 to 0.650088) shows a similar but smaller set of spirals - this time there are fourteen. Note that they are not equally distributed around the circle - populating the 'southern' part of the map, i.e lower on the y scale and closer to the main M Set more densely. Despite this, they do exhibit scaleable, rotational symmetry.

However, the most remarkable fact to my mind is that in the Figure at this scale, each curl exhibits a recursive banding - there are 14 repeating bands before each of the repeating 'branches'. This is shown more clearly in Figure 5 (coordinate centre as for figure 4). The overall effect is of 14 brachiopods or tendrils spiralling into or out from a common (starlike) origin.

More generally, the numbers 7 and 14 appear to be particularly significant in the sets of repeating phenomena around the M Set. So do the numbers 8 and 16. I would be interested to hear of a good explanation for this.

Returning to our dictionary definition of spirals as 'continually winding' we can see from zooming in and out around Figure 3 that this is not quite true for phenomenological spiral, which are finite. Like other chaotic curls, self-similarity is stable only within a relatively bounded region - a few 'generations'. At the outer limits the spirals 'explode' into other phenomena. Iteration and scale affect the image's 'ramifications', and the amount of 'noise' around the repeating areas.

I hinted earlier that we could explain spirals mathematically. Well, no, not really. We need to know why they occur as well as how to model them. Pickover (4) speculates that spirals may be biologically useful as they keep long spaces compact, as in the digestive tract. But at the more microscopic and macroscopic levels we can still only observe the spiral; we do not know how or where it is coded into an organism's DNA; we do not know what force takes over from gravity to form the double spiral arms on the massive scale of galaxies. The origins of the great family of spirals are arguably the commonest and most enigmatic of all the strange attractors.

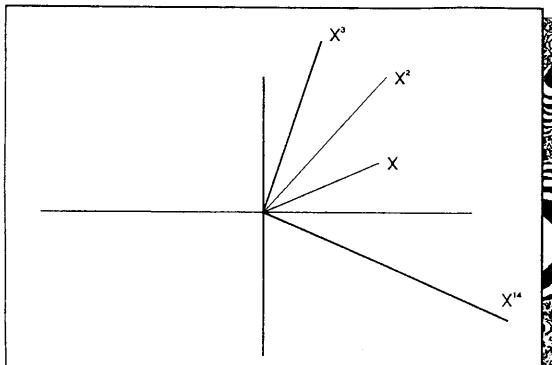


Fig. 1

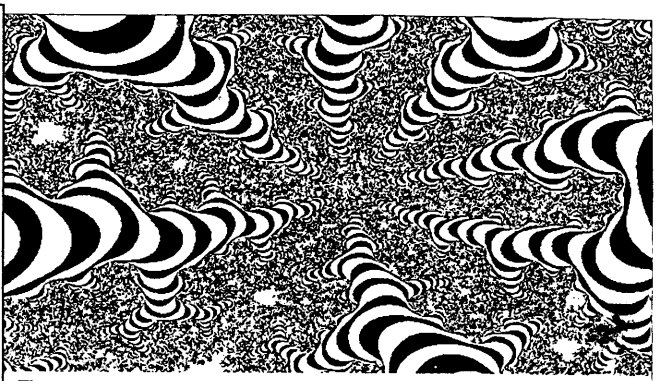


Fig. 2

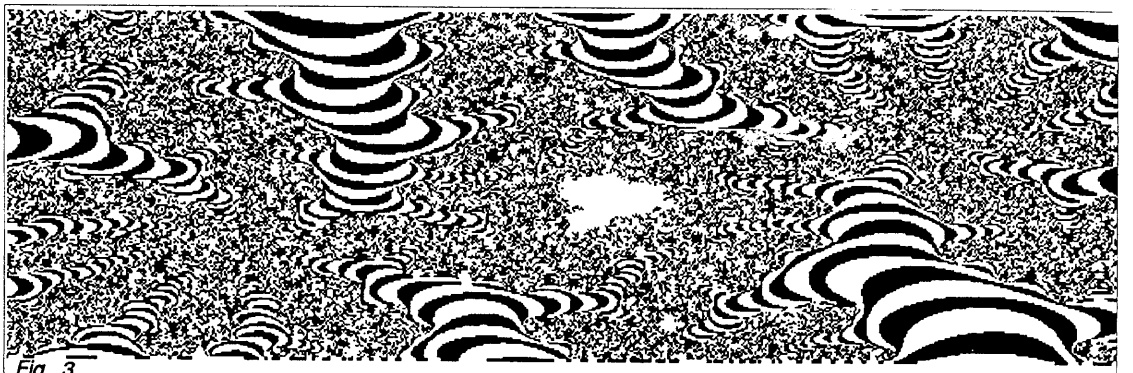


Fig. 3

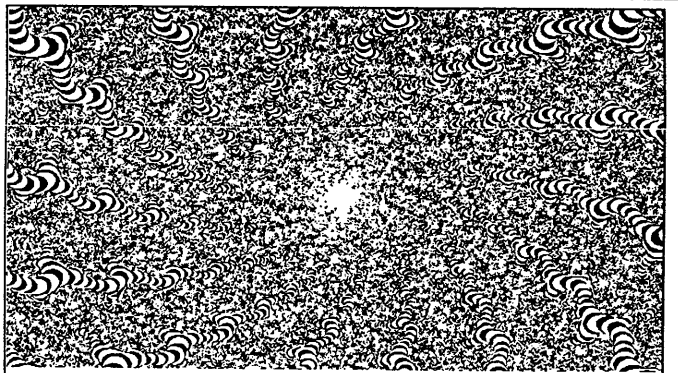


Fig. 4

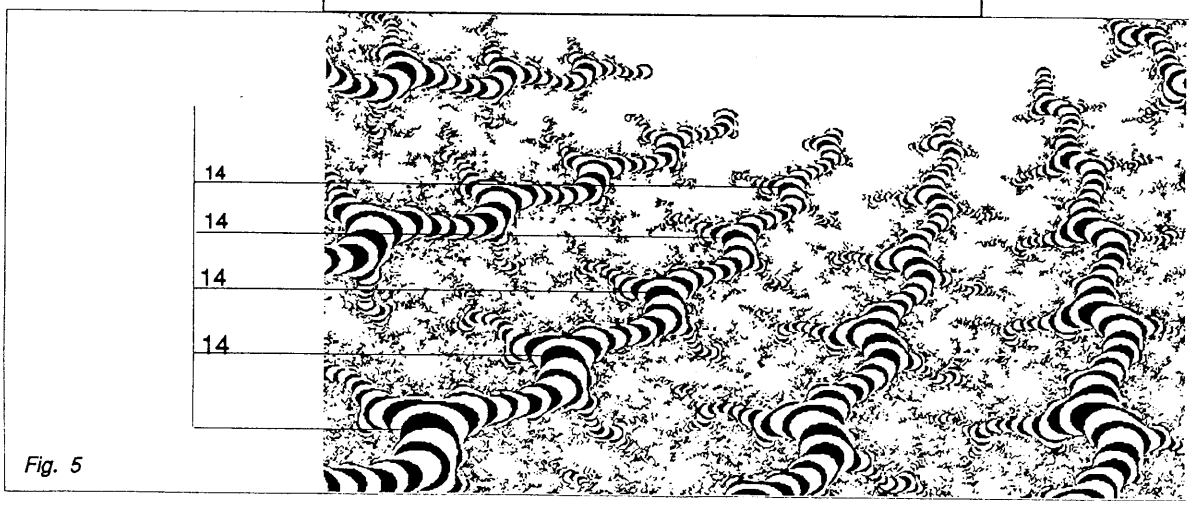


Fig. 5

get the cross product to give Normal N:

```
nx=py*qz - pz*qy
ny=px*qz - pz*qx
nz=px*qy - py*qx
```

Now, given a light source with coordinates (lx,ly,lz) we find the vector L, of the light, which is simply the coordinates of the light. We find the normal N as before and then change both to unit vectors. This is done by dividing each the component of the vector by the pythagorean length of the
vector.

i.e. sqrt(x*x + y*y + z*z)

The amount of light reaching a surface is proportional to the Cosine of the angle between the surface and the light. By a strange coincidence, there is another vector operation, the dot product which is equal to the cosine of the angle between two vectors, for unit vectors. We work this out ...

```
dotprod = lx*nx + ly*ny + lz*nz
```

This gives a value between 0 and 1 for visible surfaces. If you just want a hidden surface routine, this is far as you need to go. Just draw the triangle if the value of dotprod is positive.

To add shading, you need to be able to draw filled triangles, independent of the background. BBC Basic will do this, so does GEM on the Atari ST, but a lot of Basics and C's don't. In GWBasic you should be able to use the plain fill and some colour fiddling. Fill your palette with a smooth scale of as many greys as you can get. To find the shade, simply multiply by a suitable scale (64 for VGA) for your palette, and draw a filled triangle of that shade. If the value is negative, the triangle is hidden, and as before, don't draw it.
Tip: you can sometimes pad out a grey scale with blues of similar tone. Or, if you are a glutton for punishment, work which colours work with the Colour on the Monitor turned down.

So now you can produce shaded images of your favourite fractals. Great snapshots of your holiday on Stiperstones Ridge or Mount Mandelbrot.
On that point. has anyone managed to make the 'Brownian Surfaces' described in Mandelbrot's Fractals:Form, Chance and Dimension? These seem an ideal candidate for shading. I have only had a brief look at the book, and I couldn't see how to make them.

The shading model used in this routine is taken from a very readable book
'Computer Graphics: A Programing Approach' By Steven Harrington, published
by McGraw-Hill, ISBN 0-07-100472-6.
Other references that may be of interest are:

'Illumination for Computer Generated Pictures'
Bui Tuong Phong, Communications of the ACM, vol. 18 no. 6 pp. 311-317
This is a (considerably) enhanced version of the system described. Although I found it heavy going, and very comprehensive.

'Ray Tracing Procedurally Defined Objects'
James T.Kajiya, ACM Transactions of Graphics, Vol. 2 no. 3 pp. 161-181
This gives a method for producing ray traced pictures of fractal surfaces. Very Tricky Stuff.

'An Improved Model for Shaded Display'
Turner Whitted, Communications of the ACM, vol. 23 no. 6 pp. 343-349
Shows how optical laws can be simulated to produce photo-realistic computer graphics.

'Ray Casting for Modelling Solids'
Scott D.Roth, Computer Graphics and Image Processing vol. 18, no. 2
pp 109-144
Practical description of a ray tracing system. Quite long and detailed, but quite easy reading. My favourite, I still intend to write a Ray Tracer based on this system.

Bibliography:

1. Arnheim, R. Art and Visual Perception. Faber 1956

2. Pickover, C.A. A short recipe for seashell synthesis. IEEE Computer Graphics and Applications, November 1989, pp. 8-11

3. Becker, K-H. and Dorfler, M. Dynamical Systems and Fractals. C.U.P. 1989

4. Pickover, C.A. Computers, Pattern, Chaos and Beauty. Alan Sutton Publishing, 1990

Shaded Three Dimensional Models

By Howard Jones, 12 Fountains Garth, Bracknell, Berkshire RG12 4RH

I read with interest Jon McLaren's comments in Fractal Report issue 9 about producing shaded 3d pictures of fractals.

The following routines will produce fully shaded pictures of any object that can be represented as a series of planes. It takes as input the x,y & z coordinates of the three corners of a triangle, and the coordinates of the light source in the scene.

The program can be built up in stages, so that it is easy to see what is going on. The first stage simply changes the three dimensional object coordinates into two dimensional screen coordinates. This routine uses the parallel projection. This uses a simple addition to the x and y coordinates to create a skewed picture which is often used for drawing maps and graph surfaces.

```
:
screenx = x*10 + y*5
screeny = y*5 + z*5
:
```

The two multipliers scale the image for the screen, with values of 10 and 5, a 20*20 grid fills a 320*200 pixel screen. Change the values to fit your screen, but they should be roughly in the ratio 2:1.

For each of the points in the triangle, the transformation is performed, and the resulting three points plotted as a new triangle. If this repeated for all the planes in the scene, you will have a 'wireframe' picture of the scene.

Next we can add the shading part. Even if you don't have many colours, you can still use this routine to give hidden plane pictures.

For this, you must ensure that when the coordinates are passed to the routine, they are clockwise from the direction they are to be viewed. This is usually not a problem, as the points will be fed to the routine by some sort of loop, so that once you have figured out which way it should be, it will work for all points. You must also draw the most distant points first (sorry). Again, if you are drawing from an array/grid this won't be hard. Also note that x and y are the plane of the screen, and that z is positive going into the screen, with z=0 at the surface of the screen.

For this, it helps if you know a little about vector maths. A vector is a direction, with an x, y and z component. The triangles can be described as two vectors, both from one corner, towards one of the other corners. The vectors are found by getting the difference between the x, y and z coordinates of the points at each end of the vector. The two vectors define the plane of the triangle to be drawn. Now, by using the 'cross product', we can get a Normal to the plane of the triangle. This normal points out of the visible side of the triangle.

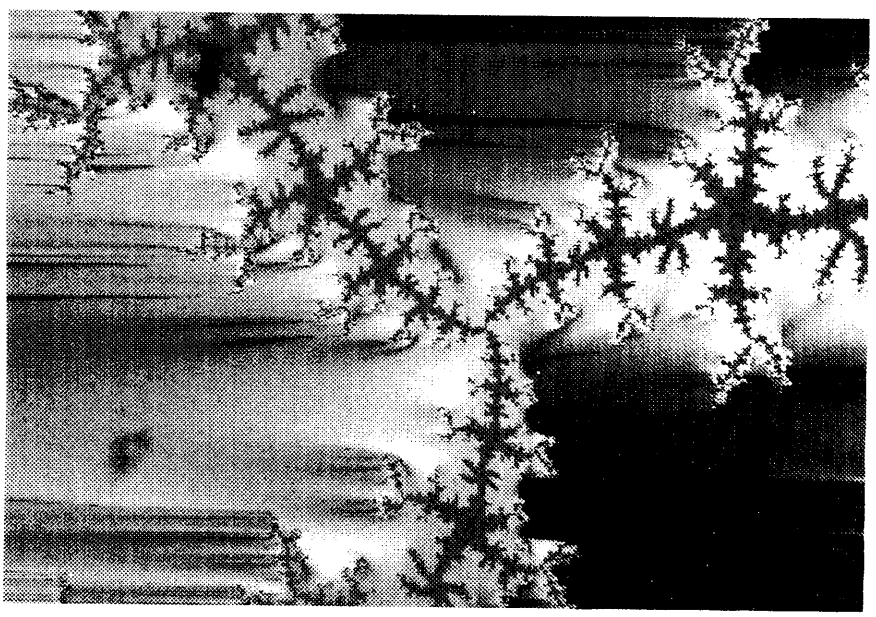
... with triangle points ax,ay,az, bx,by,bz, and cx,cy,cz :

```
find vector P:    and vector Q:
 px=cx-ax          qx= bx-ax
 py= cy-ay          qy= by-ay
  pz= cz-az         qz= bz-az
```

Full Algorithm:

```
InitGraphics
  :
  Main Loop calling Triangle(x1,x2,x3,y1,y2,y3,z1,z2,z3):
  :
End

defproc
Triangle(x1,x2,x3,y1,y2,y3,z1,z2,z3)

  px=x3-x1; py=y3-y1; pz=z3-z1;          Calculate two vectors...
  qx=x2-x1; qy=y2-y1; qz=z2-z1;

  nx=py*qz-pz*qy;                        Get Cross Product...
  ny=px*qz-pz*qy;
  nz=px*qy-qy*px;

  a=sqrt(nx*nx+ny*ny+nz*nz);             Normalize Vectors...
  nx=nx/a; ny=ny/a; nz=nz/a;

  a=sqrt(lx*lx+ly*ly+lz*lz);
  lx=lx/a; ly=ly/a; lz=lz/a;

  dotprod = lx*nx + ly*ny + lz*nz        Get Dot Product...

  if(dotprod>0)
    sx1= x1*10 + y1*5;
    sy1= y1*5 + z1*5;
      :
      Repeat for sx2/sy2 and sx3/sy3
      :
    drawtriangle(sx1,sx2,sx3,sy1,sy2,sy3,dotprod*64)   Draw plane!

endproc
```

Graphic by Dr Ian Entwistle



HYPERBOLIC PATTERNS WITH RECURSION

Ettrick Thomson

The fascinating Hyperbolic Patterns of Uwe Quasthoff, *Fractal Report*, Issue 11, pp 2-4, cry out for recursive programming. This SAM Basic program produces the same pattern as Fig.1 of that article: PROC reflect (Line 300) does the recursion. It is entered, in the first place, at L150, where x,y is a point in the fundamental region (the central black curvilinear triangle), 1,2,3 specify the 3 reflecting circles, and n (defined at L10) specifies the depth of recursion.

PROC reflect first finds u,v, the image of x,y with respect to the specified circle, using PROC image (L200); this will be a '1st-order image' in one of the 3 triangles surrounding the central triangle. The 2 images of this 1st-order image with respect to the 2 other circles will be 2nd-order images, lying in one of the 6 triangles surrounding those with the 1st-order images: these two 2nd-order images are found by the recursive calls of PROC reflect at L360; further recursive calls produce 3rd, 4th,...,nth order images.

With n=10, each point in the fundamental region produces 2046 images of various orders; with infinitely precise arithmetic these would be distinct, and different from the images of other points in the fundamental region; but with the discrete pixels of a computer display, there will be much confounding of points, which is taken advantage of to reduce running time. When each image has been found, the coordinates su,sv of the pixel to which it would be allotted are found (L330); the function POINT (L340) is non-zero if that pixel is already set; only if it is not set do we set it (if in a 'black' area) and carry on with further recursive calls.

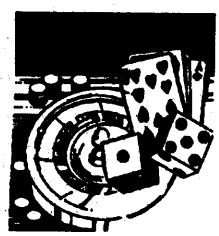
Line 10 sets various constants that depend on the computer display: xpix, ypix are the numbers of pixels in the x,y directions. The circle within which the pattern lies has unit diameter, and D is the number of pixels representing unit length. The depth of recursion, n, will have a maximum depending on the resolution of the display. The arrays xc(),yc() give the centres of the 3 reflecting circles. The x and y loops (LL60 - 170) cover the central curvilinear triangle by first covering the linear triangle with the same vertices, and then (LL80-120) testing so that LL140-150 are obeyed only for points of the curvilinear triangle. The STEP inc=1/D used for both x and y is a 1-pixel step, so that all pixels of the central triangle are set, but there are a few holes in some of the higher-order triangles. The left-right symmetry of this particular pattern is taken advantage of by: (1) the x,y loops cover only the right half of the central triangle; (2) 2 symmetrical points are plotted in the central triangle (L140); (3) 2 symmmetrical image points are plotted (L350). The parameter n of PROC reflect not only controls the depth of recursion, but its parity decides whether images are plotted, giving the alternate black and white areas.

For other patterns with 3 reflecting curves PROC reflect would

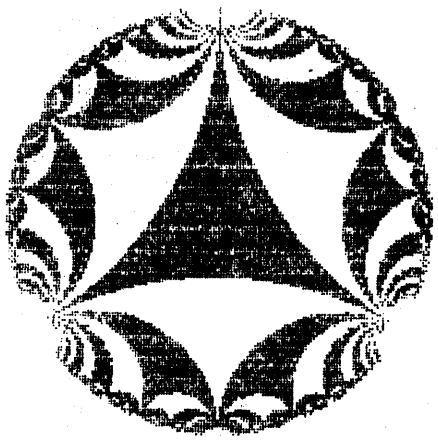
be unchanged (except that, in general, only one point would be plotted at L350); PROC image would, in general, have to select one of 3 routines, one for each curve. The main program would have to specify the reflecting curves and arrange to cover the pixels of a suitable fundamental region.

```
 5 REM hyperbolic pattern

10 LET xpix=256,ypix=176,D=160,n=10
20 LET xo=xpix DIV 2+0.5,
       yo=ypix DIV 2+0.5
30 LET inc=1/D,k=SQR (0.75)
40 DIM xc(3),yc(3)
50 LET xc(1)=0,yc(1)=-1,
       xc(2)=k,yc(2)=0.5,
       xc(3)=-k,yc(3)=0.5
60 FOR x=0 TO k/2 STEP inc
70  FOR y=-0.25 TO 0.5-2*k*x STEP inc
80   LET i=1
     DO
90    LET u=x-xc(i),v=y-yc(i)
100  EXIT IF u*u+v*v<0.75
110   LET i=i+1
120  LOOP UNTIL i=4
130  IF i=4
140   IF (n+1) MOD 2=1 THEN
       LET sx=INT (D*x+xo),
           sy=INT (D*y+yo)
       PLOT sx,sy
       PLOT xpix-sx,sy
150    reflect x,y,1,n
       reflect x,y,2,n
       reflect x,y,3,n
160  END IF
170 NEXT y
   NEXT x
   STOP

200 DEF PROC image a,b,q, REF c, REF d
210  LET a=a-xc(q),b=b-yc(q)
220  LET s=0.75/(a*a+b*b)
230  LET c=s*a+xc(q),d=s*b+yc(q)
240 END PROC

300 DEF PROC reflect x,y,p,n
310  LOCAL u,v
320  image x,y,p,u,v
330  LET su=INT (D*u+xo),
          sv=INT (D*v+yo)
340  IF POINT(su,sv)=0
350   IF n MOD 2=1 THEN
       PLOT su,sv
       PLOT xpix-su,sv
360   IF n>1 THEN
       reflect u,v,1+p MOD 3,n-1
       reflect u,v,1+(1+p) MOD 3,n-1
370  END IF
380 END PROC
```

RECREATIONAL & EDUCATIONAL COMPUTING



https://dr-michael-ecker.weebly.com/



Wolf's Dust Dr Daniel Wolf, Ph.D.

A novel method of calculating images of Z^2+Z (the Mandelbrot Set), Z^2+C (Julia Sets), Z^3+Z ('cube' Mandelbrot Set), and Z^3+C ('cube' Julia Sets) reveals heretofore hidden aspects of the fractal behavior of these functions under iteration for the region of the complex plane outside a radius of 1.46 from the point (-0.5,0).

I developed this method for my program, FractalPro. The original intent of this development was to improve the speed of calculation for points inside the above stated radius, since the Mandelbrot Set and Julia Sets exist within a circle of that radius. I have made some study of the consequences of my method of calculation and enclose some images which may be of interest to your Scientific American readers. I call my technique 'method of truncated integers'. I call the resulting fractals Wolf's Dust(M(i)) and Wolf's Dust(J(i)). When the usual Z^2+Z formula for M (the Mandelbrot Set) is applied, this would be WD(M(2)). Similarly there are WD(J(2)) for Z^2+C and WD(M(3)) and WD(J(3)) for Z^3+Z and Z^3+C, respectively.

The appearance of Wolf's Dust is (superficially) a series of concentric rings surrounding the Mandelbrot Set (or other set, depending on the function iterated). Upon closer examination, the series of concentric rings is found to be fractal, that is, it is self-similar, containing rings within the rings and quite interesting smaller scale structure which reveals whorls, swirls, and objects which resemble optical diffraction patterns and the interference patterns generated by laser light passing through irregularly shaped refractive materials (blown glass objects, etc., I've forgotten what these patterns are called).

Upon even closer examination (WD's can be enlarged indefinitely, like M) WD(M(2)) is found to contain tiny copies of M. Little baby Mandelbrot Set figures are in this dust (hence the name, Wolf's Dust of M or J or whatever) and can be magnified and resemble the ordinary images we've all seen of Mandelbrot Sets. Elsewhere the dust continues to reveal structure resembling optical interference and diffraction patterns. The regions surrounding the 'baby' M figures in the dust are also unusual in appearance, with some aspects of the 'interference' pattern showing up.

I have found points in WD(M(2)) of interest and then calculated ordinary Julia Sets (i.e. within a radius of 2 from the origin) for those points. These Julia Sets from WD(M(2)) don't behave like Julia Sets based on points of M within radius 2.

I've also taken ordinary Julia Sets of ordinary points in M and explored WD(J(2)). Again I've found copies of the Julia Set located in the WD(J(2)).

I am intrigued especially by the 'interference pattern' appearance of the various WD's. I have confirmed the existence of the WD(M(3)) and WD(J(3)), that is, for the 'cube' formulae. It would be most interesting to learn if these 'interference or diffraction patterns' have an optical counterpart. Of special interest to me is whether these patterns can be imaged or Fourier transformed to yield an interesting result. I have a suspicion that the pattern is a hologram of something else, perhaps M itself.

I believe FractalPro is the only commercial program which permits exploration of Wolf Dusts. Since most computer programs for exploring Mandelbrot and related Sets are done with high-level languages most programmers won't encounter the 'truncated integer' method. Most such programs use ordinary or double-precision floating point representations of Z and C. FractalPro doesn't. FractalPro uses specifically designed (what I believe are) unique assembly language routines with fixed point integer representations of Z and C. Instead of leaving a large 'headroom' for the integer parts of the numbers, these algorithms truncate the integer part (which improves the speed of multiplication of the fixed point numbers) so that numbers which exceed a certain size (3.9999...) aren't represented. That was a design decision for the integer algorithms since calculating M only requires comparisons of a maximum norm (size) of Z^2+Z of 2. Additional bits of precision devoted to the integer part of Z seemed wasted (and added extra right-shifts to the multiplication routines). The result is a program which is extremely fast and has more bits of fractional precision than IEEE double-precision floating point. The additional fractional precision permits magnifications over 100 trillion times, greater than most floating point-based methods. Other programmers may be able to explore WD(M(2)), but they'll have to find some method of truncation not usually found in commercial Mandelbrot Set exploration programs.

The result of the truncation is an apparent 'artifact' (Artifractals?), the rings of dust surrounding the sets. The 'artifact' is no ordinary kind of artifact, though. It's not just some bothersome blob. Because of the truncation's effect on the iterated function at these radii, the consequence is as if the function were iterated modulo 4 (with fractional remainders allowed). Thus arises Wolf's Dust. It may contain even more variety and infinitude than M itself (since it contains plenty of M's in addtion to the interference patterns). I do not

yet know if all the little M's inside WD(M(2)) are connected. I doubt it. WD(M(2)) seems to be like M turned inside out. The interference patterns look like they might be the result of a circular wave converging on M from the outside and then reflecting and interfering with itself.

I've also been viewing a new video from Art Matrix. The Mandelbrot Set zoom sequences in the video (made on a Cray? at Cornell) can be easily generated with FractalPro on an Amiga (with quality remarkably close to the stuff done with the Cray), and I've done panning, Julia Set zooms, and 'cube' formula zooms with FractalPro, too

Ref. Mr. Tom Marlow's notes on page 14:

Mr. Cade Roux contacted me about finding the Wolf Dust using my commercially available FractalPro programs for Amiga (information enclosed) a few weeks ago. I have known of its existence for about 3 years and never publicised it much. There are hundreds of FractalPro users in 10 different countries, but no one commented on the presence of WD until Mr. Roux (which has surprised me, since its presence is sort of obvious). I mostly regarded it as a 'artifact' (artifractal) and knew it was itself fractal way back when I first wrote the earliest version of FractalPro in 1988. The reason for its existence is related to truncating the integer part of the numbers used in the calculation. A small part of it can be found inside the range I mentioned in my letter to Dewdney (near the main spike). I can force its appearance even closer to the 'origin' by reducing the precision of the integer part of the complex numbers. It can effectively be forced to 'disappear' by increasing the precision available for the integer. I don't believe it can be seen at all unless integer techniques are used instead of the usual floating point. Integer methods are (I believe) only practical in assembly language (the language in which I wrote all of FractalPro). Its particular appearance in FractalPro is due to a happy medium I found experimentally which provides a certain integer precision resulting in very nice Mandelbrot (and related) images yet which (nearly) minimizes the computation times (they are substantially faster with greater fractional precision than floating point algorithms on the Amiga and most other machines). The integer methods I use to represent the complex numbers provide for some variation which allowed my early experimentation and my decision on the 'happy medium' I mentioned above. I did some exploration in the WD a long time ago but Mr. Roux's letter in August made me realize it had a separate potential for very beautiful images and I decided to 'let the cat out of the bag' regarding its origin. That's why I wrote Dewdney (cc to Roux). Mr. Marlow's letter doesn't mention which algorithm or machine he uses, but is the first time I've heard of (possibly) another program which permits exploration of WD. I hope he's using FractalPro.

Announcements

TI99/4A Program Exchange Offer

Mr Stephen Shaw wrote enclosing prints of the work he had done with his Texas TI99/4A. He is happy to exchange disks or BASIC listings with other TI99/4A owners. His address is 10, Alstone Road, Stockport, Cheshire, SK4 5AH. His programs include gingerbread men, Connett Circles, Feigenbaum Plots, and some graphics from Dr Pickover's book.

[2023: You can see these graphics at:
http://shawweb.myzen.co.uk/stephen/sdlbasic.htm]
(Archived: https://tinyurl.com/sdlbasic)

He also notes that if you look at some of the Connett Circles, you can see squares around the circles. I think that this may well be due to computer rounding errors, but maybe readers have other ideas.

Plea for help with Amstrad CPC664

Mr Bev Mason, of Highlands, Bromsash, Ross – on – Wye, Herefordshire HR9 7PR asks for some help with the best language to use for fractals with the 64k Amstrad CPC664. He finds assembler too slow to write and his BASIC compiler can't handle hyperbolics etc. He also asks for some standard to be used for defining Mandelbrot pictures, and seeks an explanation of quantities used in articles such as offset, scale etc which can often mean different things in different articles on the same subject.

Creating a standard would undoubtedly be well nigh impossible, but certainly we could ask authors to spell out the meaning of terms used. Although they may be obvious to some, the range of readers of *Fractal Report* is very wide.

Art Matrix's New Video

Mr Cade Roux writes to say that the new video from Art Matrix is "stunning", although there is no documentation to go with it. He also mentions that he has two of their T – Shirts. He says that they have trouble with advertising costs, and asks any reader with retail contacts who may be interested to contact Art Matrix direct. Of course the videos will sell far beyond the computer market itself. The address of Art Matrix is PO Box 880, Ithaca, NY14851, USA. Anyone interested in buying their products direct is also advised that they accept credit cards.

He also says that he has had some slides of his own made, and is investigating a 24 – bit frame buffer for his Amiga. This will enable it to offer high resolution and more colours.

Mr Lewis Siegel

has moved without leaving a forwarding address. Anyone who knows of his whereabouts is asked to ask him to register his new address if he wishes to continue with *Fractal Report*.

Editorial

Once again we come to renewal time. This year and in the future the renewal notices will go out at the end of November for the following volume. Cheques received from readers will not be cashed until 1 February, the start of RTL's tax year. This is so that the funds received can be directed towards printing the following volume without suffering any tax penalties. The number of subscribers for volume 2 (350) has been less than those for volume 1 at the end of it's year (400). Naturally this is a disappointment, but we will continue in publication whilst the numbers are above 200, and even allowing for a similar reduction we should comfortably exceed this for the next volume. As further copies of volume 1 were sold during the year, the gap is in fact wider than the above figures suggest. However volume 2 will remain on sale in 1991 as back numbers, and it is anticipated that new readers will still want them.

I would urge readers intending to renew to do so promptly before they forget. Date your cheque 1 February 1991 if you like – it makes no difference – that is when it will be paid in or soon thereafter.) We do not normally chase those who don't renew, so it is up to you to use the enclosed form. If you have lost it, the address is on the front cover.

Subscriptions won't be acknowledged, but if you haven't received the next issue by early February, then please let me know promptly, – don't leave it several months. We'll have the back numbers if you do, but obviously we don't want to keep you waiting.

Actually the main threat to *Fractal Report* is not so much the lack of subscribers or the number of articles, but the violently accelerating speed of time! A number of contributors are helping with alleviating this by providing material on disk or in a camera ready form that can be used without any cutting and pasting, which apart from being a great help also adds to improving the appearance of the newsletter. Ideally articles should fill a whole number of pages, not waste space with lots of blank lines etc, and include illustrations fitted in spaces in the text, eg around program listings. I still seek new articles, although we have held a number over this month. The more articles I have to choose from the better the quality of each issue.

The success of *Fractal Report* has to a large amount been due to the generosity of *Computer Shopper,* (14, Rathbone Place, London, W1P 1DE) who run free spots for clubs and specialist publications. A number of other magazines gave us free publicity, but only *Computer Shopper* repeated it relentlessly month after month (with only a few misses.) *Micro Computer Mart,* for example, have a club spot, but charge clubs large sums of money for insertion. Indeed, the VAT on the advertising bills from such magazines, on its own, would be enough to close a publication like *Fractal Report.*

I would unreservedly recommend *Computer Shopper* to all readers who don't know it as having an excellent mix of advertisement and editorial material for all main makes of computer. It is also half the price of any comparable competition.