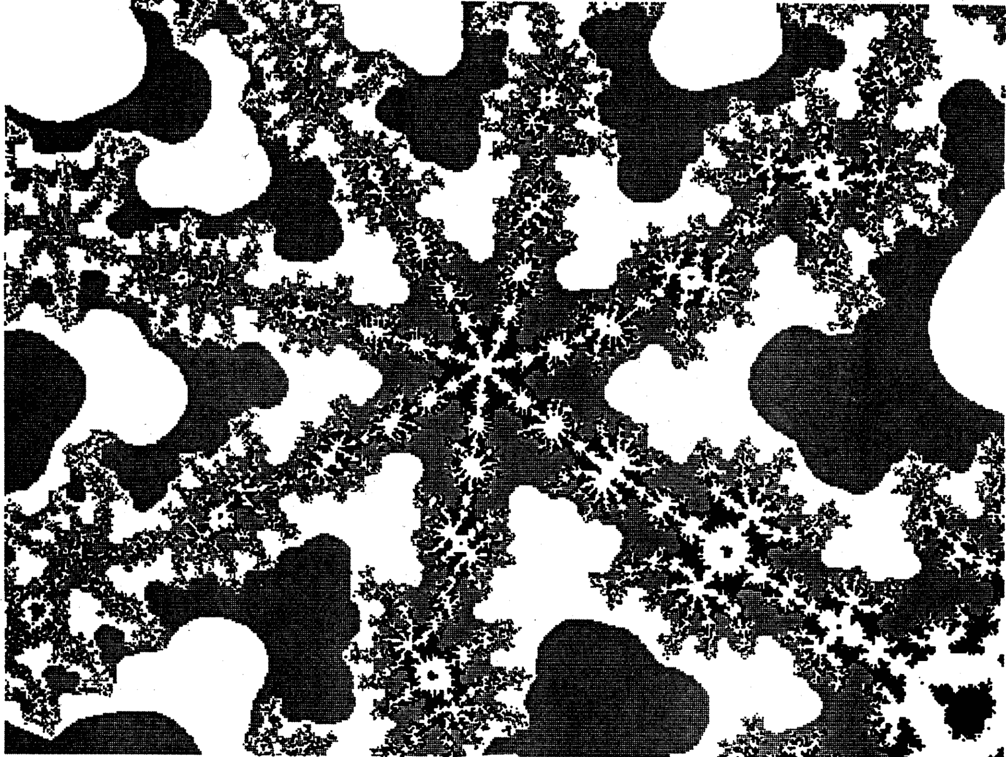
1989

Fractal Report

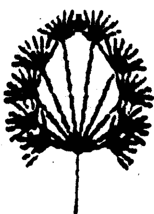
Issue 5

mandel21.vga © LTC Associates, 1989 all rights reserved



Larry Cobb Winner (mono version) M. A. Kirk

Two Views of 3D	Lewis Siegel	2
Fractal Landscapes	Kate Crennell	10
Larry Cobb Prize	Larry Cobb	16
Mandelbrot 8087	Ed Hersom	18



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

Two Views of 3-D:

Displaying Two Dimensional Fractals in Three Dimensions*

by

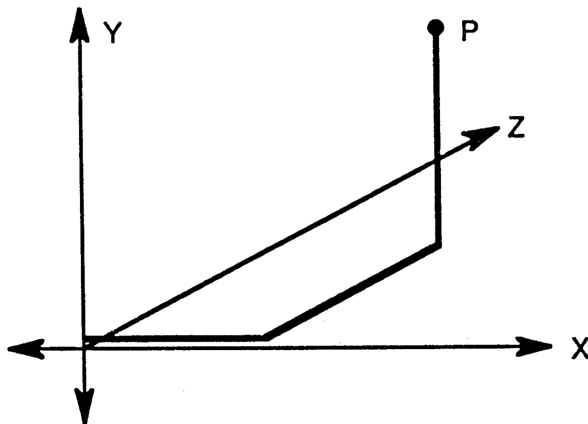
Lewis Siegel

Introduction

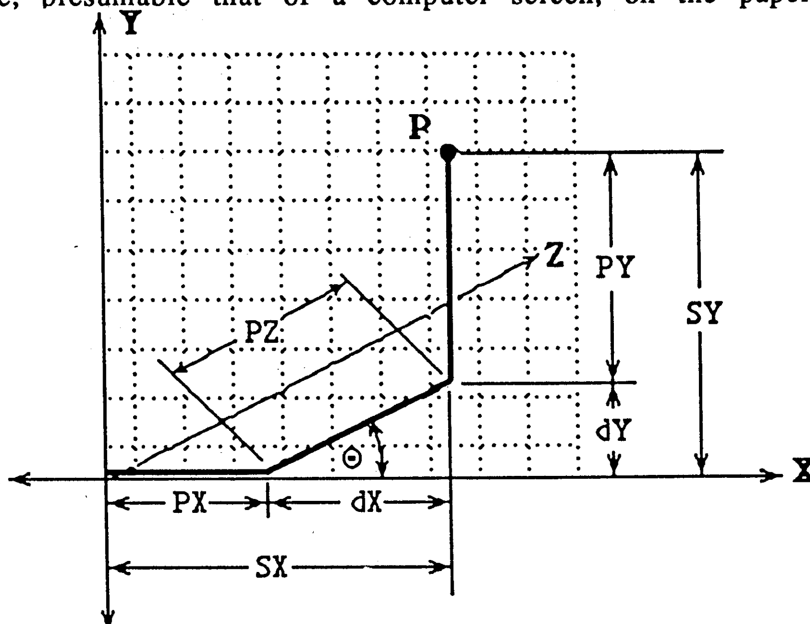
This is a quick overview of two basic 3-D transformations: the oblique transformation and the perspective transformation. It offers techniques which should enable a graphics programmer to achieve some basic 3-D effects. In this paper the x and y axes will be the horizontal and vertical axes. This makes the x-y plane parallel to the screen, and the z-axis will be the axis of depth with positive values "behind" the screen. The screen coordinates will be SX and SY; the three space coordinates will be the name of the point followed by the coordinate, i.e. PX, PY and PZ.

Oblique 3-D

This kind of transformation is analogous to the diagrams used in most math classes to describe a three dimensional space. The diagrams usually look something like this:



As you can see, this representation of a three dimensional space is completely embedded in the two dimensional space of the paper. The effect of three dimensions is achieved by placing the z axis at an angle between the x and y axes. The transformation follows directly if we imagine placing a two dimensional coordinate space, presumable that of a computer screen, on the paper:



$$dX = PZ*COS(\Theta)$$
$$dY = PY*SIN(\Theta)$$

$$SX = PX + dX = PX + PZ*COS(\Theta)$$

$$SY = PY + dY = PY + PZ*SIN(\Theta)$$

One can see that the transformation is achieved by a displacement of the x and y coordinates. If Θ is the angle that the z axis makes with x axis (in two space,) and dx and dy are the x and y displacements, then it is clear that dx =PZ*cos(Θ), and dy = PZ*sin(Θ). Thus, the transformation of a point P from three space into the two space of the screen is simply:

$$SX = PX + dx = PX + PZ * \cos(\Theta)$$
$$SY = PY + dy = PY + PZ * \sin(\Theta).$$

When using the transformation in a program, it is fastest to calculate the sine and cosine once and use those values rather than repeat the calculation for each transformation, or to simply enter the values by hand. The important values for the commonly used angles 30°, 45 °, 60 ° are sin(30°) = cos(60°) = 0.5, sin(45°) = cos(45°) ≈ 0.707107 and sin(60°) = cos(30°) ≈ 0.866025.

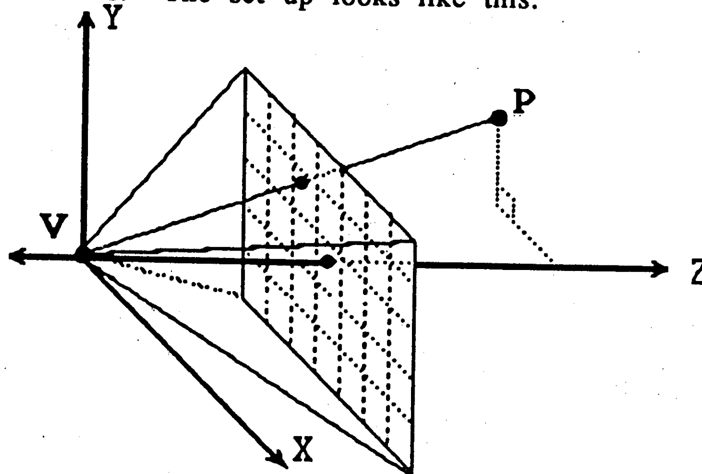
The z coordinate of the point can be scaled to produce a more realistic looking image. If the full value of the z coordinate is used, the image tends to appear elongated. By multiplying the z coordinate of the point by a constant C, the transformation is changed to:

$$SX = PX + C*PZ * \cos(\Theta)$$
$$SY = PY + C*PZ * \sin(\Theta).$$

Values of less than one for C produce a foreshortened image which appears more realistic. A commonly used value for C in drafting is 0.5. It should be noted that if C is less than one the image will be "squeezed" together, and if C is greater than one the image will be "stretched" apart. Depending on how the image is being displayed, some information may be lost in the case of $C < 1$, or gaps may appear in the case of $C > 1$.

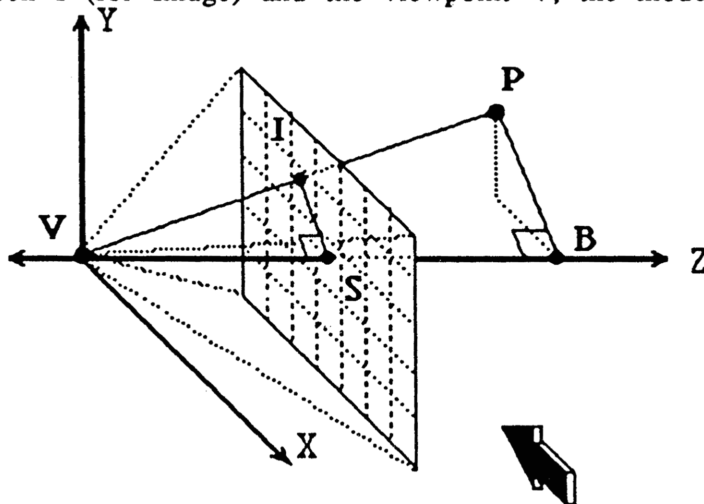
Perspective 3-D

The perspective approach to 3-D is somewhat more complicated than the previous approach. It entails the use of a "viewpoint," and involves placing the "picture plane" (the screen) in between this viewpoint and the points which are to be displayed (transformed onto the screen.) Although the viewpoint, screen, and points can be placed anywhere in three space, the mathematics is simplest when the viewpoint is at the origin and the screen is parallel to the x-y plane at an arbitrary distance along the z-axis. The set up looks like this:

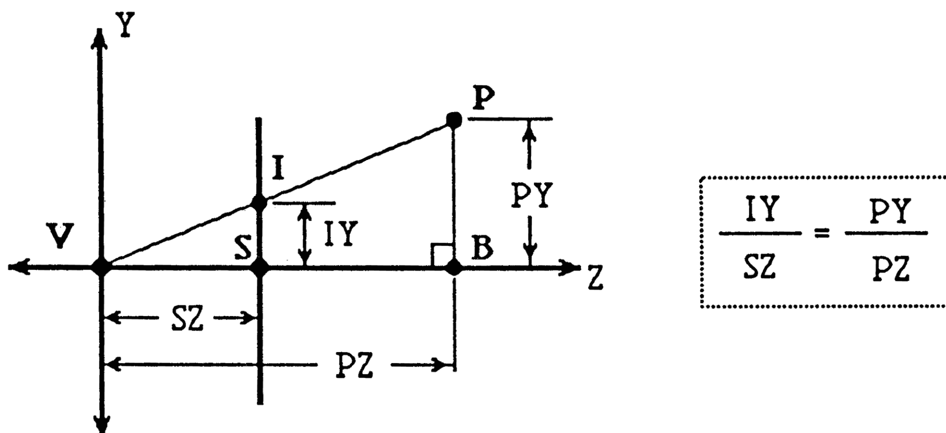


Transforming the point from three space onto the two space of the screen is really just a matter of drawing a line segment from the viewpoint to the point which is to be transformed and determining where this line segment intersects the screen. One way of doing this would be to determine parametric equations for the line segment

and then to solve for x and y by setting the z coordinate equal to the distance of the screen from the origin. However, the method of like triangles provides a simpler and quicker transformation. To do this, one must imagine a line segment from the point P to the z axis which is also perpendicular to the z axis. If the point where this line segment intersects the z axis is called B (for base,) the point where the z axis intersects the screen S (for screen), the point to be transformed P, the image of the point on the screen I (for Image) and the viewpoint V, the model will look like this:



Now there are two similar right triangles VSI and VBP. Solving for IY is easy if the model is viewed from the side, which produces a projection like this:



Using similar triangles, it is clear that IY=SZ*PY/PZ. This same method is used to solve for IX, by looking <u>down</u> on the model from the top. Then, IX=SZ*PX/PZ. Note that SZ is the distance of the screen from the x-y plane which was chosen arbitrarily. Changing the distance of the screen from the x-y plane alters the severity of the perspective. If the screen is close to the x-y plane, the perspective will be more dramatic than if it is farther away.

Now, the transformation is simply:

IX = SZ*PX/PZ,
IY = SZ*PY/PZ.

If the machine which you are working on does not have its origin in the center of the screen or you want the viewpoint in some position other than the center of the screen, simply add displacements to the transformed coordinates to place the viewpoint in the desired spot. At this point the perspective transformation has been achieved using <u>only the coordinates of the point being transformed and the distance of the screen from the origin</u> with only four multiplications.

This scheme works well if the viewpoint, plane and points being transformed are arranged in the fashion described above. If the relationships are changed -- for

example by putting the viewpoint on the same side of the screen as the points -- the transformation will hold, but the results are not predictable. Most likely the points will not be mapped onto the screen, the image will be inverted, or, in some cases, extreme points will reach the precision of the machine and be mapped to zero, in which case they will be mapped to the screen with undesirable results. Watch out for points that lie in the same x-y plane as the viewpoint. In this case, PZ = 0 and a division by zero error will occur.

These transformations are useful, but they do not provide any information about precedence, i.e., which points are in front of others. They work well, however, with wire framing in which you simply transform the points one at a time and connect them the same way you would normally. If there is some prior knowledge of the object to be displayed, precedence can be "faked" by simply plotting the furthest points first so that the closer points will be plotted on top of them.

Often in the discussion of perspective projections the question of vanishing points arises. Vanishing points are important in drafting when a perspective projection is being rendered. They are determined by the shapes of objects and their relationships to each other and the viewplane (screen), as well as by the relationship of the viewplane to the viewpoint. Because of this, they do not come into play when points and therefore objects are being transformed mathematically. Rather, they arise naturally as objects or parts of objects approach large distances from the viewpoint and, if desired, can be calculated from the transformation and the objects being displayed.

Two Dimensional Fractals in Three Dimensions

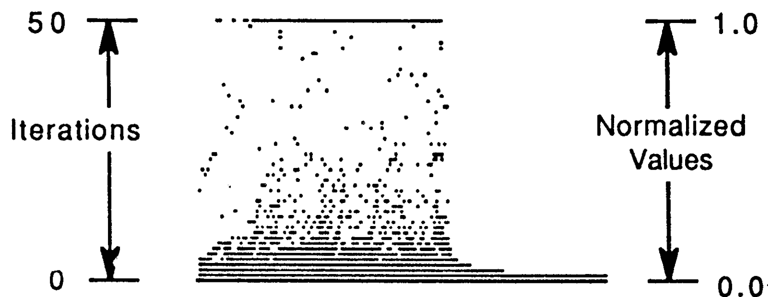
Using the transformations given above, it is an easy task to display the Mandelbrot set or Julia sets in three dimensions. Simply use the dwell value of each point in the set as the third coordinate. If the y coordinates from the originally two dimensional set are changed to the z (depth) coordinates, and the dwell values are used as the new y coordinates, images of each of the dwell sets (a set of points of equal dwell) are obtained which have been "extracted" into three dimensions. It is best to use as many different colors as possible for each dwell set, as they will overlap. †

It is important to display the image in a manner so that the furthest points are plotted first. For the Mandelbrot and Julia sets, it is best to generate them from top to bottom, rather than left to right, so that each line being displayed is at a constant depth from the screen. If the "top" of the image is placed deeper in the x-z plane than the bottom, the image will be displayed as a sequence of planes parallel to the screen, each one closer to the screen than the last, so that the final image can be viewed correctly. If the order is reversed, points which should be displayed will be hidden by points which should not and visa-versa.

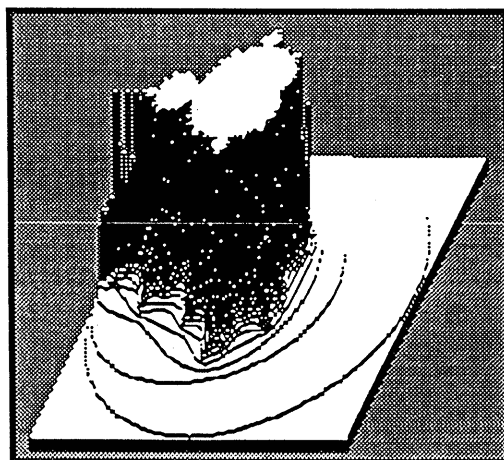
There is one remaining problem with displaying fractals in three dimensions using this method. The dwell values for the large, outermost dwell sets are all very small, and they increase rapidly as the dwell sets get closer to the Mandelbrot or Julia set being displayed. This causes the large, solid dwell sets for most images to lump together near the x-z plane and the remaining dwell sets to form a field of broken up points. The image that results usually shows the Mandelbrot or Julia set floating above a lot of confetti, and a few solid dwell sets lurking together toward the bottom of the image. Here is a side view of the Mandelbrot set generated at a fifty iteration limit, shown from -2.0 to 2.0:

----

† It is important to note that this technique simply displays an otherwise two dimensional fractal in three dimensions. A "true" three dimensional fractal is one which is generated using four dimensional complex coordinates called "quaternians."



This image shows seven or eight solid dwell sets at the bottom, a wide band of broken up dwell sets across the center, and the Mandelbrot set hovering above the whole thing at the top. Here is a picture of the resulting image:

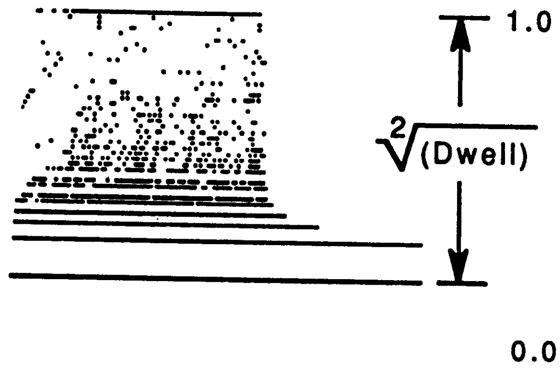


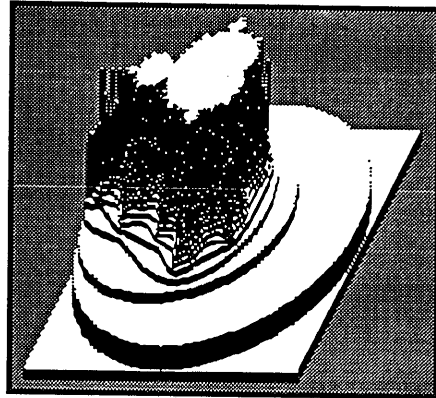
Looking at the right side of the cross section above, we can see that the dwell sets seem to be distributed in a hyperbolic manner. It is this "bowing in" which should be "straightened." The dwell sets need to be redistributed so that the larger sets are distributed more widely and the "confetti" is "squeezed" together closer to the main set. One simple way to do this is by "normalizing" the dwell values of each point, and then to replace the dwell value with a root of the normalized value, as follows:

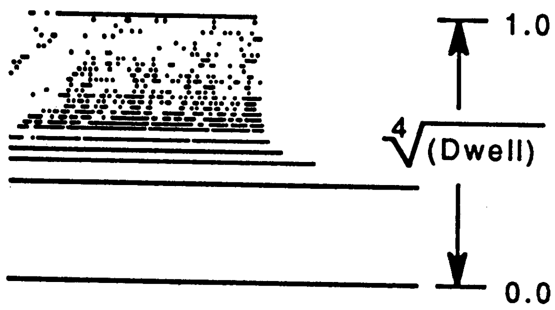
Before a point is displayed, its dwell value is divided by the maximum iteration value. This results in a new value which is between zero and one. The Mandelbrot or Julia set itself will have a value of one, while the outermost dwell set will have a value of zero.

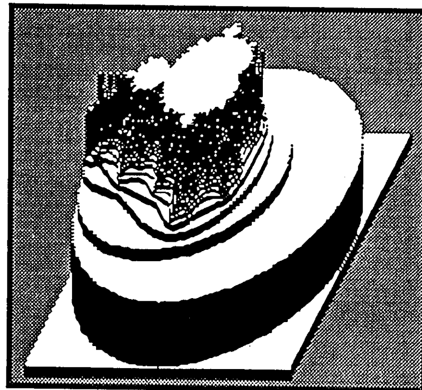
Now, since all of the dwell values are between zero and one, we can attempt to raise the dwell sets by taking the square root, or any root, of these values. Remember that a root of a number between zero and one is larger than the number itself. This causes the values closer to zero to be raised by an amount greater than the values closer to one, so that the lower dwell values are raised up and the confetti is "squeezed" together. Note that values of zero and one, which correspond to the Mandelbrot or Julia set and the outermost dwell set, are unchanged. The dwell sets are now redistributed along a parabolic curve. Admittedly, this does not precisely undo the seemingly hyperbolic distribution, but between zero and one, the curves are similar enough for the parabolic distribution to be effective.

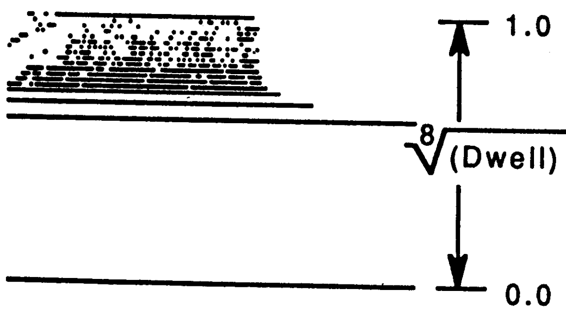
Here is the same set shown above redistributed with different root values:

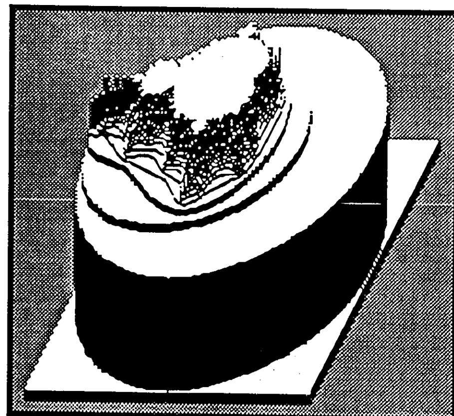












Among these graphs, the fourth root appears to be the best choice.

Now, all that remains is to obtain a new screen coordinate value by multiplying the normalized, redistributed values by the Original iteration limit. The final code in Pascal for a "square root extraction" might look something like this:

```
NewDwell := trunc(MaxIterations * sqrt(OldDwell / MaxIterations));
```

Since Pascal does not have any intrinsic root or power functions, other root values can be obtained using the Natural Logorithm function and Exponent functions. For example, a fifth root can be calculated using the following identity:

$$X^{1/5} = \exp[(1/5)\ln X].$$

Conclusion

Using these techniques, it is easy to obtain many interesting new views of familiar two dimensional fractals. In each case, the transformations can be implemented in only one or two lines of code. Because of this, it is not difficult to alter all ready existing code. Usually it can be done be simply replacing the line which plots a pixel with a call to a procedure which performs the transformation. The procedure will need only the x, y and z coordinates (where z is the dwell value) of the point which is to be transformed, and the resulting x and y screen coordinates can be passed back or plotted in the procedure itself. The resulting three dimensional representations of Mandelbrot and Julia sets can help to show the relationship of dwell sets in a way that cannot be achieved by simply cycling the color palette, and with some creative rendering, can provide very beautiful images.

Further References

Foley, J.D. and Van Dam, A., *Fundamentals of Interactive Computer Graphics*, ch. 7-8, Addison-Wesley, 1982.

Pietgen and Saupe, *The Science of Fractal Images*, Sec. 2.7, Springer–Verlag, 1988.

*This work was done in a senior seminar in Fractal Graphics at San Francisco State University taught by Lawrence S. Kroll, PhD. Pictures were generated on a Macintosh using Turbo Pascal.

Following is pseudo-code for generating a three dimensional version of the Mandelbrot set similar to those shown previously. It should be noted that in this example the Mandelbrot set is being displayed while it is being generated. It is suggested that these two procedures be done separately; if the dwell values for the Mandelbrot set are saved in a data file, a separate program which performs only the three dimensional transformation will show the set very quickly and allows for different views to be displayed without a regeneration of the set each time. This approach along with methods for data compaction is another topic and may be written up in the near future.

```
Program Mandel3d;
{This Program Displays a 3-D image of the Mandelbrot Set on a 300x300 Raster with its
origin in the lower left hand corner}

function GetDwell(cx,cy : real;MaxIter : integer):integer;
{This function returns the dwell value of the complex point (cx,cy)}
var
   zx,zy,temp : real;
   NumIter : integer;

begin
   zx:=cx;
   zy:=cy;
   NumIter:=0;

   while ((zx*zx+zy*zy < 4) and (NumIter < MaxIter)) do
      begin
         temp:=zx*zx - zy*zy + cx;
         zy:=2*zx*zy + cy;
         zx:=temp;
         NumIter:=NumIter + 1;
      end;

   GetDwell:=NumIter;
end;

Procedure Transform3d(x,y,z : integer; var xout,yout : integer);
{This procedure actually does the 3-D transformation}
var temp : integer;

begin
   xout:=trunc(x + 0.707107*y);      {It's only two lines!}
   yout:=trunc(z + 0.707107*y);
end;

Procedure MakeMandel;
{This procedure generates and displays the Mandelbrot Set}
var
   x,y,x_increment,y_increment : real;     {complex coordinates}
   sx,sy,sz,tx,tz : Integer;               {screen coordinates}

const
   left = -2; right = 2; bottom = -2; top = 2;  {complex borders}
   screenleft = 0; screenright = 300; screenbottom = 0; screentop = 300; {screen borders}

begin
   x_increment:=(right - left)/(screenright - screenleft);
   y_increment:=(bottom - top)/(screenbottom - screentop);

   y:=bottom;
   for sy:=screentop downto screenbottom do            {vertical scan}
      begin
         x:=left;
         for sx:=screenleft to screenright do          {horizonatal scan}
            begin
               sz:=GetDwell(x,y,100);
               Transform3d(sx,sy,sz,tx,tz);            {the transformation!}
               PutPixel(tx,tz);
               x:=x + x_increment;
            end;
         y:=y + y_increment;
      end;
end;

Begin
   MakeMandel;
   readln;
end.
```

FRACTAL LANDSCAPE

K.M.Crennell Aug 89

Program 'Mounts' draws a random landscape showing mountain peaks behind a lake with low foothills in the foreground. (See screen dump for an example) The mountains are made by starting with a triangle, finding the midpoints of the sides, randomly displacing them by a small amount, and making four new triangles by joining the new points to the original vertices, then operating similarly on each new triangle, and then operating similarly....... until the degree of realism wanted is reached. This can take a long time, depending on your processor and what level of recursion you want in your picture. In the diagram, the highest peak has 6 levels, the others 5 and the foothills only 2; the plotting time was about 5 minutes. The only graphics routine needed is to draw a filled triangle in the current colour.

The program was written in 'C' for an Acorn Scientific co-processor, with graphics routines written to simulate the BBC-BASIC graphics routines which use the graphics extension ROM standard on the BBC-Master.
The routines used are:

```
        colr  computes the colour and plots a triangle
        peak  controls the drawing of each peak
        pnt   returns 2 points xz,yz xp,yp randomly displaced in 2 directions
              from a given point x,y
        vertx makes 4 new triangles from the 2 sets of points
```

Graphics routines are:

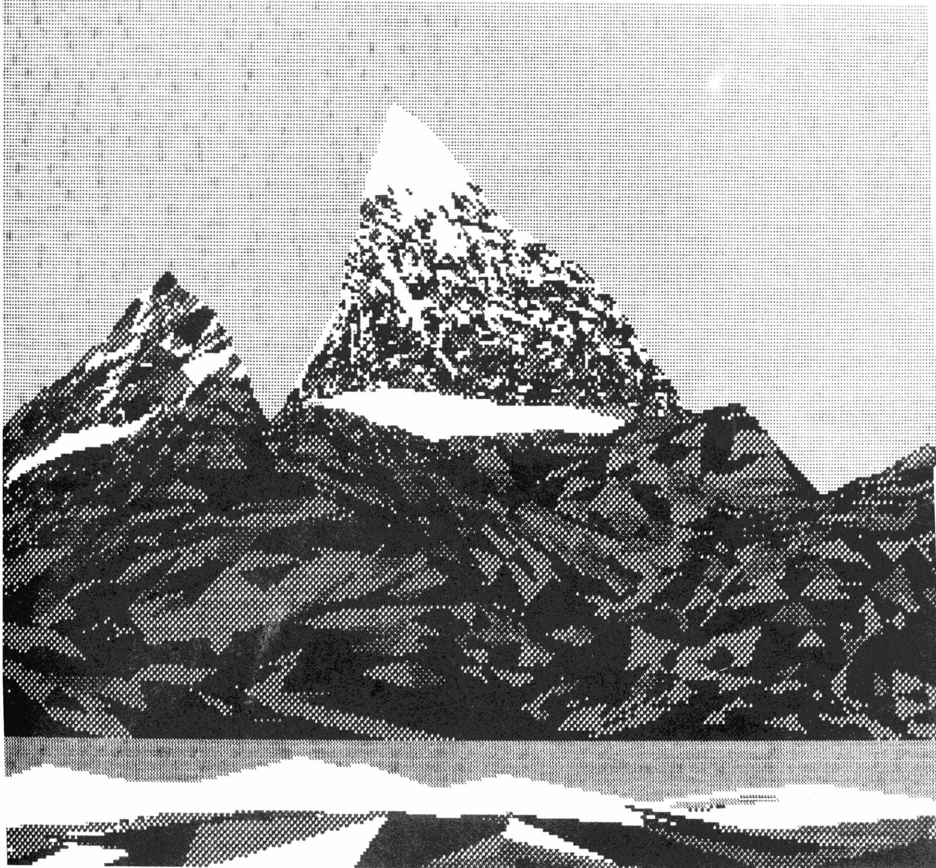
```
        mode(1) initialises graphics screen to 4 colours 1279 by 1023
        vdu23   sets up user defined shaded patterns
        vdu19   sets the colours to Black, White, Blue, Cyan
        gcol    selects the colour or pattern
        rectan  draws a filled rectangle in the current colour
        trian   draws a filled triangle in the current colour
```

This program was based on one written in BASIC for the BBC-B microcomputer by Michael Batty, printed in his book "Microcomputer Graphics" published by Chapman and Hall, ISBN 0-412-28540-1 (paperback). Chapter 5 contains a good explanation of the principles of construction of these fractal landscapes, and several other programs, including a version of "Planet rise over Labelgraph Hill". These programs use only the graphics of the BBC-B, with 4 colours in mode 1, and no graphics extension ROM to give more realistic colouring. I rewrote the program in 'C' mainly because Batty's programs run very slowly, taking several hours to make landscapes like the one shown in the screen dump. Less realistic plots can be made more quickly by reducing the levels of recursion, or the number of peaks in the picture.

Continued on next page

Amygdala Reprints *Turbo Mandelbrot Sets*

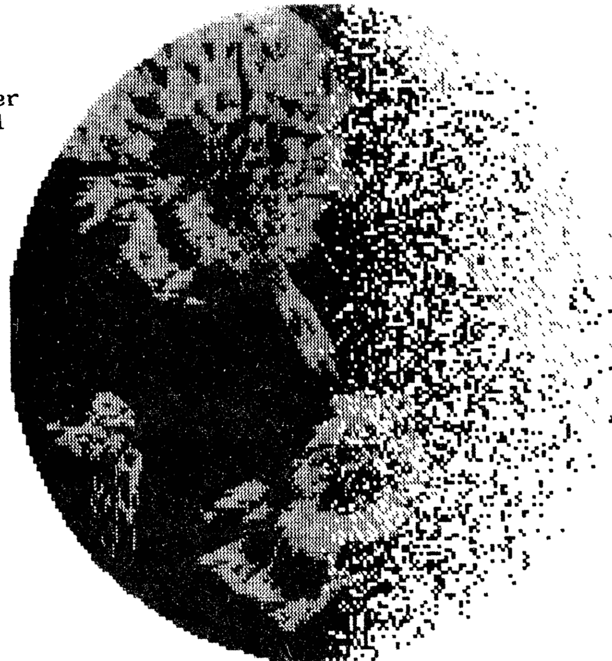
The American Publicaion *Amygdala* reprinted Dr Dietnar Saupe's article from issue 1 in its issue 17. Amygdala is published from Box 219, San Cristobal, NM 87564, USA, from where further details can be obtained. The issue also included details of a fractal music tape, and I have requested a copy of this for evaluation and possible sale to *Fractal Report* readers. I have also asked the author if he has any simple fractal music programs for computers that BEEP and which will help beginners in Fractal Music get started.



Editorialette

It was originally intended that this issue of *Fractal Report* should follow at the end of September or early in October, and in fact it was sent to our printer during the second week of September. However he was taken ill through overwork and eventually we recovered the copy to have it done elsewhere. I have taken this opportunity to alter this editorial slightly. **Issue 6 will be posted sometime in January 1990.**

Dr Keith Wood has written a PC program that compiles fractal equations in machine language. It is advertised in *Electronics World and Wireless World* October 1989 for £15 (£16 overseas) from him at 33, Glan Aber Park, Liverpool, L12 4YP. It includes an editor and syntax checker, file compression, and display editor. The colour pictures in EWWW suggest that this program is very good value to those with PCs (5.25", add £1.50 for 3.5") with EGA and/or VGA, DOS 2.0 or higher and at least 256k.



Planet Rise Over
Labelgraph Hill

Printout by
Kate Crennell

from
BBC Basic
program in
Batty:
*Microcomputer
Graphics*

```
/*         FRACTAL Mountains    K.M.Crennell     31 July 89     */
/*                                                             */
#include <stdio-h>
/*  #include "graphics-h" you may need this for your graphics  */
  int yt,kw,xz,yz,xp,yp;
  main()
  {
    FILE *stream, *fopen();
    int num,d,x1,y1,x2,y2,x3,y3,l,kl,nread,seed[2],rand();
    char *fname;
    void vdu(),mode(),gcol(),move();  /*        graphics routines   */
    void peak();
    printf(" enter filename ");scanf("%s",fname);
    stream=fopen(fname,"r");    /*    open datafile for reading        */
    printf(" enter 2 random seeds >0 and <20000 ");
    scanf("%d %d",&seed[1],&seed[2]);  /* read these from keyboard  */
/*   peaks behind lake have different random start from mounds in front */
    mode(1);  /*  initialise graphics 4 colours 1279 by 1023 pixels */
    vdu23(15,1,3,3,1,1,3,3,1);    /*  shaded patterns defined        */
    vdu23(14,1,2,2,1,1,2,2,1);vdu23(13,1,0,0,1,1,0,0,1);
    vdu19(0,4);vdu19(1,0);vdu19(2,6);vdu19(3,7);  /* set colours    */
/*        start of drawing each set of peaks                         */
    for(l=1; l <=2 ; l++)
```

```
    {
      nread=fscanf(stream,"%d %d %d ",&num,&yt,&kw);srand(seed[1]);
/*   num=how many peaks, yt=height of snow line, kw=type              */
      for(kl=1; kl <= num ; kl++)
      {                      /*    for each peak      */
       fscanf(stream," %d %d %d %d %d %d %d ",&d,&x1,&y1,&x2,&y2,&x3,&y3);
/*       read level of recursion and initial co-ordinates for peak       */
        peak(x1,y1,x2,y2,x3,y3,d); vdu(7);  /* ring bell after plotting */
      }              /*   end of for on kl    (each peak in this set)      */
      if (l==1) { gcol(0,2);rectan(0,0,1300,150); /*  draw the lake   */ }
    }              /*   end of for on l   (set of peaks to draw) */
/*              save the screen to disc              */
    kl=open("P1ALPC",1); write(kl,0XFFFF3000,0X5000);close(kl);
    fclose(stream); /* close datafile stream */
    exit(2);
  }

void  colr(wx,wy,xx,xy,zx,zy,type) int wx,wy,xx,xy,zx,zy,type;
    {
    int ytt,zt,tt,gc1,gc2; void gcol(),move(),plot(),trian();
     /*  given the 3 sets of co-ordinates and type for a triangle
         decides colour dependent on height and plots triangle    */
    if ( wy > xy )
      ytt=wy;
    else
      ytt=xy;
    if ( ytt < zy )  ytt=zy;
    zt=yt-(ytt*ytt)/yt;
    tt= ( (float)rand()/16383.)*yt ;
    gc1=0;
    if (ytt > 0.9*yt)
      gc2=3;    /* decide on shaded colour   */
    else
      {
      if (ytt < 0.2*yt )gc2=1;
      else
        {
        if (tt+tt+tt < zt+zt )gc2=1;
          else
            {
            if (zt+zt+zt < tt+tt)gc2=3;
            else
              {
               gc2=0;
               gc1=16*((tt+tt+tt)/zt);
              }
            }
        }
      }
    gcol(gc1,gc2);
    if (type == 3)trian(wx,wy,xx,xy,zx,zy);
      else plot(85,zx,zy);
    }                          /* end of function colr */

 void peak(x1,y1,x2,y2,x3,y3,d) int x1,y1,x2,y2,x3,y3,d;
     {
     int xa,ya,ax,ay,xb,yb,bx,by,xc,yc,cx,cy;void colr(),vertx(),point();
```

```
 /*  xn,yn are co-rdinates of triangle     d=depth of recursion        */
    pnt(x2-x1,y2-y1); xa=x1+xz;ya=y1+yz;ax=-xp;ay=-yp;
    pnt(x3-x1,y3-y1); xc=x1+xz;yc=y1+yz;cx=xp;cy=yp;
    pnt(x3-x2,y3-y2); xb=x2+xz;yb=y2+yz;bx=-xp;by=-yp;
 /*       randomly displace midpoints  of sides       */
    if  (d==0)  {ax=0;ay=0;bx=0;by=0;cx=0;cy=0;} /* initial displacements */
    if ( (d > 0) && (kw == 1) ) goto done;
    colr(x1,y1,xa+ax,ya+ay,xc+cx,yc+cy,3);
    colr(xa+ax,ya+ay,xc+cx,yc+cy,xb+bx,yb+by,1);
    colr(xc+cx,yc+cy,xb+bx,yb+by,x3,y3,1);  /* plot the 4 new triangles */
    colr(xa+ax,ya+ay,x2,y2,xb+bx,yb+by,3);
done:
    vertx(x1,y1,x2,y2,x3,y3,xa,ya,xb,yb,xc,yc,d);
    }        /*     end of function peak  */

void  vertx(a1,b1,a2,b2,a3,b3,aa,ba,ab,bb,ac,bc,d)
    int a1,b1,a2,b2,a3,b3,aa,ba,ab,bb,ac,bc,d;
    {   /* makes the 4 triangles from the randomly displaced points   */
    void peak();
    if  (d==0) return;
    peak(a1,b1,aa,ba,ac,bc,d-1);
    peak(ac,bc,ab,bb,a3,b3,d-1);
    peak(aa,ba,a2,b2,ab,bb,d-1);
    peak(aa,ba,ab,bb,ac,bc,d-1);
    }                  /* end of function vertx */

void pnt(x,y)  int x,y;
    {              /*   computes the random midpoint displacements   */
    float r,w,fx,fy;
    int rand();
    r = 0.5 + (((float)rand()/16383.)-0.5)/3.;
    w=(0.03+((float)rand()/16383.)/25.);
    if(rand() > 8191) w=-w;
    fx=(float)x;fy=(float)y;
    xz=(int)(r*fx-w*fy);yz=(int)(r*fy+w*fx);
    xp=y/20;yp=-x/20;
    }        /* end of function pnt  */

/*           typical data file follows
          4 more pointed peaks with greater recusion  behind lake
4 900 0
6   -50  -50 1400 -150  550 900
5 -1000 -100  750 -100  250 700
5 -1000 -100 1900 -100 1000 550
5     0 -100 2600 -100 1250 500
4 90 0
2  -500 -100 1500 -100  100  80
2  -700 -100 1850 -100  500  65
2  -500  -50 1200  -80  890  57
2     0  -30 1800 -100 1100  85
      4 lower mounds before the lake       */
```

On the Persuit of Perfection

felt that poorly written text may be difficult to understand and hence frustrate the reader. Mr felt that a good command of English suggested a good command of programming.

Einstein would have been prevented from writing as he suffered from dyslexia and could not read or write until after fourteen. I doubt that anyone would refuse an article from Einstein if he were alive today.

Logic is a product of the left hemisphere of the brain, language the right so skill at one doesn't imply skill at the other. People tend to dominate at one or the other, it is rare for someone to be good at both.

In my opinion as long as the message of the article is communicated then it has served its purpose. I would rather have an informative, original article in poor English than an unoriginal article in perfect English.

I don't claim that what I have written is perfect English, but hopefully it gets my point across. Pity is doesn't have anything to do with fractals.

The "LARRY T COBB PRIZE" Awards

Again, I can't claim to be inundated with entries for the competition. Either you don't want a competition, (which is a shame because sharing co-ordinates is great fun) or you are overwhelmed by the quality of the entries and dare not enter, (which is silly because any variation you have found could interest other readers). This is likely to be the last competition unless interest grows, or maybe we will run it less often.

If you have an entry send it direct to me at the address quoted in the advertisement for my program. The prize is a full colour print of the winning entry, and a black and white version will probably be printed in Fractal Report.

I am sure that our editor will not be upset if I quote his "unofficial" entry. "Have you ever considered doing the Mandelbrot Set itself with the hottest colour, eg a white with a dash of yellow, with a sharpish transition to the next shade of yellow and then gradually tapering off through reds and blues to blackness at the edges?" It certainly gives an interesting effect and John says that the idea came from reading a science fiction novel where the universe had a distorted geometry. Colour variations and pixel allocations of well known fractals can make interesting contributions.

Another entry can from someone who would probably wish to remain anonymous! He is one of the many, happy (I hope) users of my DRAGONS fractal investigation software. Unfortunately he did not realise that his entry was on of the default conditions built in for the Julia Set. For those of you that do not have the program, the c centre co-ordinates are 0 + 0j, z = -0.74543 + 0.11301j with a side of 3.5. This gives a very intricate, swirling pattern and requires a lot of iterations to fill in the detail.

The first international entry comes from Mr M A Kirk, living in Geneva. He accuses me of having "either lots (and lots) of spare time" or of having a Cray to heat my home! Unfortunately, I have neither but I do run my computer overnight, on many occasions, and let it do the work.

His entries are two areas of the Mandelbrot Set. The first entry has a side of 0.000166 and a c centre value of -0.664338585 + 0.453785253. He stresses the importance of colouring the image and suggests using warm colours from yellow to red. Then cycling the colour palette gives "the illusion of flames dividing and subdividing."

He describes his second entry as representing ice crystals and assigns 32 shades of blue colours to the fractal. The side dimension is 0.0232377 with the c centre at -0.669774665 + 0.45774595j with a colour offset of 10. This is the one I like best and it wins this issue's prize colour print.

Steve Wright's article in the last issue gave an interesting twist to the Mandelbrot Set. By giving a starting value to z, the Set is distorted or

"melted," as Steve calls it. Here are the co-ordinates, including the starting value for z or "melt factor": side 0.036, c centre -0.095 + 0.658j and z = 0.3 + 0.3j.

There seems to be very little work done in this area, perhaps because everyone is so overwhelmed by the infinite range of the more familiar version. Offsetting z does have an interesting consequence; the Mandelbrot Set is four dimensional but if only either the real part or the imaginary part of z is varied, then a three dimensional figure results. In other words, a 3D Mandelbrot could be contructed by incrementing the real (or the imaginary) value of z in steps. If only the real or the imaginary is non zero, the Mandelbrot Set keeps some kind of symmetry about the real axis throughout, and it does not "melt", but it does produce interesting distortions. I wonder what the complete 3D image looks like or if anyone has the time to find out?

Larry Cobb

ADVERTISEMENT

FRACTAL INVESTIGATION PROGRAM for IBM PC Compatibles — New release - version 2

DRAGONS is a very comprehensive program for exploring the infinite variety of Mandelbrot fractals. Although it has many options, it is easy to use after a little practice. No mathematical knowledge is assumed and it comes complete with examples and full instructions.

- Calculates and displays five types of Mandelbrot Fractals using the maximum number of colours and pixels available in CGA, EGA and VGA
- Menu driven, with built-in defaults, for ease of use.
- Screen cursors are used to select an interesting area and magnify it to full screen size. This process can be repeated up to a magnification of 500,000,000,000 times.
- A cursor simplifies the location of fractal dragons or Julia Sets
- Co-ordinates can be stored and transformed for later calculations
- Allows lengthy fractals to be generated in parts. The computing can be done whenever it is convenient.
- Colours can be changed for maximum effect, both before and after calculation.
- Built-in conversions for dragon "tiling" technique.
- Different pixel-grouping algorithms can be selected for the best results.
- Uses a Maths Co-processor, if fitted, to speed up calculations by 10 to 20 times.
- Includes EGA & VGA screen blanking to protect display phosphors
- Supports Amstrad 2000 series in 256 colour EVGA mode.
- A catalogue of fractals can be compiled using the INFO.EXE utility

The software costs £14 including examples and full instructions. Owners of DRAGONS 1.2 (and above) can update for £5 if they return their original disks. Please state whether 5¼ or 3½ inch disks are required and order from:

Larry Cobb, c/o Bay House, Dean Down Drove, Littleton, Winchester, Hants, SO22 6PP

Mandelbrot – 8087

Ed Hersom

* The Intel 8087 was a Numeric Data Processor chip

In Issue 2 it was pointed out that I had not given any code for my "Z-trajectories" (Issue 1). At the time I did not think this would be of general interest, but as the use of the 8087 coprocessor has been mentioned and I use this chip, I would like to describe how I write my code. There is a difficulty in that I program in FORTH, but I hope this will not make what I say completely incomprehensible! Another general point is that if the maximum benefit is to be gained from the 8087, the core of any program must be in machine code, that is machine code for 8087. To start, therefore, I must say something about FORTH and the code for the 8087.

FORTH uses the term "words" for almost everything: procedures, variables, constants, operators and so on. A word has an identifier which can be any string of ASCII characters not containing a space. The definition of a new word, for a procedure, say, commences with a ":" followed by the name of the new word, followed by a list of previously defined words which have to be executed in sequence, and ends with a ";". Such a word is called a secondary since it does not really do anything itself, it just calls other words. These words, if secondaries, do the same thing, i.e. call other words, but the tree-like structure must have ends where the words consist of machine code and do something useful. These words are called primaries. To define a primary the name is preceded by CODE and is followed by a list of words which create code. The definition is terminated with END-CODE. All the words, that is all those originally supplied with the system together with those defined for use for a current program, form the "dictionary". Groups of words, called "vocabularies", can be included or omitted at will.

The 8087 employs a stack of registers 8 deep by 80 bits wide, and has a powerful arsenal of instructions to operate on the numbers stored on this stack. There are also, of course, instructions to transfer numbers between the top of the stack and the store, but these are normally only(!) 64 bits long. Numbers in the store may be integer or floating point, but the long 80 bit numbers on the stack are always floating point. If calculations can be carried out with a minimum of transfers between the 8087 and the store, then rounding errors as well as transfer times are reduced. In my FORTH system I have a vocabulary of words which create code for the instructions. Only a small selection is required here, and to demonstrate what each word does I give the state of the stack before and after its operation. For example, if y and then x have been put on to the stack, i.e. x is on top, then

```
apdup      \ y,x  - - - y,x,x
```

means "duplicate the top item on the stack". (ap stands for "arithmetic processor", the 8087). The stack contents are shown left-to-right with the top on the right. The " - - - " separates "before" and "after". The "\" is used to indicate comment and in any FORTH code means "ignore everything to the end of the line". (A line terminator, Carriage Return or Line-feed, is ignored in FORTH, apart from marking the end of a comment.)

Other instructions are:

```
apover     \ y,x - - - y,x,y
faddp      \ y,x - - - (x+y)  Pop the 2 top items, leave sum on stack
fmulp      \ y,x - - - xy
fsubp      \ y,x - - - (x-y)  Note that register 1 is subtracted from
                               register 0, the top of the stack.
```

Hence,

```
apdup faddp \ y,x - - - y,2x
apdup fmulp \ y,x - - - y,x^2
```

If n is any integer in the range (1,7), then "n fmuli" means multiply the top of the stack with the number in the nth register and similarly for "n faddi". "n xchi" means exchange top of the stack with the nth register. So

```
1 fmuli     \ y,x  - - -  y,xy          Compare this with fmulp above.
2 faddi     \ z,y,x - - - z,y,(x+z)
2 xchi      \ z,y,x - - - x,y,z
```

Finally, fstiw[bx] means "convert the number on the top of the stack to an integer 1 word long (16 bits), store it at the address held in the BX register of the 8086 and pop the original number off the stack". All these words, except this last one, create code which includes a "WAIT" instruction. This is a necessary instruction to ensure that the two processors remain in step. fstiw[bx], however, needs a WAIT to be explicitly inserted.

Fortunately this small selection of the 8087 code is all that is required for my example, and we can now get on with Mandelbrot. The word, NEXTXY, is to code just one iteration, i.e. writing z' = z^2 + k where z' = x' + iy', z = x + iy and k = KR + iKI, we calculate x' = x^2 - y^2 + KR , y' = 2xy + KI . The code is written as a "macro", that is, when NEXTXY is called it will write the code, not execute it. The comments, however, show what happens when the code is executed and the contents of the stack are shown after the execution of the code. Further, at execution time, KI,KR,y,x must already be on the stack. x' and y' will replace x and y after execution.

```
: NEXTXY            \ Create a secondary word called NEXTXY
  ASSEMBLER         \ Include the assembler vocabulary. This is not
                    \ usual in a secondary definition.
                    \ KI,KR,y,x            Starting condition
  apover            \ KI,KR,y,x,y
  1 fmuli           \ KI,KR,y,x,xy
  apdup faddp       \ KI,KR,y,x,2xy
  4 faddi           \ KI,KR,y,x,(2xy+KI) = y'
  2 xchi            \ KI,KR,y',x,y
  apdup fmulp       \ KI,KR,y',x,y^2
  apswap            \ KI,KR,y',y^2,x
  apdup fmulp       \ KI,KR,y',y^2,x^2
  fsubp             \ KI,KR,y',(x^2-y^2)
  2 faddi           \ KI,KR,y',(x^2-y^2+KR) = x'
;                   \ end of word
```

The word to execute this code can then be simply

```
CODE NEXTZ            \ Create a primary called NEXTZ
     NEXTXY           \ Insert the code here
     NEXT,            \ Go to next word
END-CODE              \ End of code for NEXTZ
```

This is what I used for calculating my z-trajectories. A program was written which loaded the appropriate values for KR and KI and 0,0 for the initial values for x and y. NEXTZ was called repeatedly and after each call, copies of the new values of x and y were extracted from the 8087 and plotted.

A Mandelbrot calculation requires the iteration coded above. It does not require extraction of the point at each stage, but it does require a check to see that the number of iterations has not exceeded a certain limit and that |z| has not become too large. In most languages this "house-keeping" can take a significant amount of time and FORTH is no exception. For maximum speed I have therefore included some 8086 code to do a complete Mandelbrot calculation for one value of k. The word is called M-BROT and is defined below. Since FORTH makes extensive use of the Reverse-Polish notation, assembler words often appear as other assemblers' notation in reverse. For example:

DX PUSH = PUSH DX and CX, DX MOV = MOV CX, DX and so on.

M-BROT requires KI,KR,x,y on the AP-stack (8087) as before, but it also requires the iteration limit, LEVEL, on the 8086 stack (usually just referred to as the "stack"). It returns the actual number of iterations, COUNT, on this stack and leaves KI,KR and the final values of x and y on the AP-stack.

```
\ Stack: LEVEL - - - COUNT.   AP-Stack: KI,KR,y,x
CODE M-BROT              \ Create a primary word called M-BROT
   DX POP                \ Pop LEVEL into DX
   CX, DX MOV            \ Copy into CX
   BX, # WORK MOV        \ Address of WORK into BX for future use
 1$: NEXTXY             \ Insert code here and label the entry point 1$
     \ AP-stack: KI,KR,y,x    where x & y are now the new point
   apover apdup fmulp \ KI,KR,y,x,y^2
   apover apdup fmulp \ KI,KR,y,x,y^2,x^2
   faddp                 \ KI,KR,y,x,(x^2 + y^2) = |z|^2
   WAIT  fstiw[bx]       \ Store the integer value of |z|^2 into
   AX,  WORK MOV         \      WORK and bring it back into AX
   AX, # 4 SUB  2$ JA \ Jump to 2$ if |z|^2 - 4 > 0
  1$ LOOP                \ Decrement CX and repeat from 1$ if non-zero
 2$: DX, CX SUB          \ Subtract CX from DX so that DX now contains
  DX PUSH                \    the COUNT and push it to the stack
   NEXT, END-CODE        \ Go to next word.  End of code for M-BROT
```

For those interested in speed, I timed this word for 100 times, each time with the level set at 1500 at a point where COUNT = LEVEL, i.e a point of the Mandelbrot set. Total time was 32.1 secs. so dividing by 150,000 gives an iteration time of 214 microsecs. (Processor is a 80186 running at 8 Mhz)

Finally, let me add that FORTH programs do not usually have so much, or indeed any, machine code. FRACTAL calculations are exceptional, but I have shown how FORTH can readily cope.