Creating Fractals

Fractals are created by iterating processes.
One of the simplest fractals is known as the von Koch curve.
To create a von Koch curve, start with a line segment.

____

Divide this segment into three sections:

____ ____ ____

Now, replace the middle section with a triangle:

____/\____

Break each of these 4 new segments into three segments.
Replace the middle section of each segment with
another triangle.

Look at the first two graphics to the left to see examples of von Koch curves.
[1]
[2]

When iterated an infinite number of times,
The von Koch curve becomes both self similar and
infinite in detail, two properties of fractals.

Here is a quick review of iterating functions.

REAL FUNCTIONS

let $f(x) = x^2$

let $x_0 = 2$

$x_1 = f(x_0) = 2^2 = 4$

$x_2 = f(x_1) = f(f(x_0)) = f^2(x_0) = 4^2 = 16$

$x_3 = f(x_2) = f(f(f(x_0))) = f^3(x_0) = 16^2 = 256$

$2, 4, 16, 256, \cdots$ is the orbit for $x_0 = 2$

let $f(x) = x^2$

let $x_0 = 0.5$

$x_1 = f(x_0) = 0.5^2 = 0.25$

$x_2 = f(x_1) = f^2(x_0) = 0.25^2 = 0.125$

$x_3 = f(x_2) = f^3(x_0) = 0.125^2 = 0.0625$

$0.5, 0.25, 0.125, 0.0625, \cdots$ is the orbit for $x_0 = 0.5$

let $f(x) = x^2$

let $x_0 = 1$

$x_1 = f(x_0) = 1^2 = 1$

$x_2 = f(x_1) = f^2(x_0) = 1^2 = 1$

$x_3 = f(x_2) = f^3(x_0) = 1^2 = 1$

$1, 1, 1, 1, \cdots$ is the orbit for $x_0 = 1$

1 ~BMP: koch1.bmp
2 ~BMP: koch2.bmp

This gives us the square rules for REAL NUMBERS:
1. if $x_0 > 1$, the orbit escapes to infinity
2. if $x_0 < 1$, the orbit goes to 0
3. if $x_0 = 1$, the orbit remains at 1

Now, lets look at iterated complex functions.
Note, that z is a complex number, written
in the form $z = x + y\mathbf{i}$
Look at this example of a complex iterated square (^{2}) function:

```
let f(z) = z²
let  z₀  = 1 + 2i
     z₁ = f(z₀)  = (1 + 2i)²  = -3 + 4i
     z₂ = f(z₁)  = f²(z0)     = (-3 + 4i)² = -7 - 24i
      :
      :
     z₅ = f(z₄)  = -98248054847 - 116749235904i

     1 + 2i,  -3 + 4i, -7 - 24i, ··· is the orbit for z₀ = 1 + 2i
```

```
let f(z) = z²
let   z₀ = 0.1 - 0.5i
      z₁ = f(z₀)  = (0.1 - 0.5i)²  = -0.24 - 0.1i
      z₂ = f(z₁)  = f²(z₀)         = (0.24 - 0.1i)² = 0.476 + 0.48i
       :
       :
      z₅ = f(z₄)  = 0 + 0i

    0.1 - 0.5i, -0.24 - 0.1i, 0.476 + 0.48i, ··· is the orbit for z0 = 0.1 -
0.5i
```

```
let f(z)  = z²
let   z₀ = 0 - 1i
      z₁ = f(z₀)  =  (0 - 1i)²  = -1
      z₂ = f(z₁)  =  f²(z₀)   =  (-1)²  = 1
        :
        :
      z₅ = f(z₄)  = 1

        0 - 1i, -1, 1, 1, 1, ··· is the orbit for z₀ = 0 - 1i
```

But how can you tell if an initial seed z_0 will go to zero or infinity?

Here is how:
First, we must convert the complex number into a real number so that we can use the REAL square rules.
z_0 is the distance on the complex plane from the origin $(0,0\mathbf{i})$ to the point $(x_0,y_0\mathbf{i})$.
It is the ABSOLUTE VALUE of z_0.

It is a REAL number.

z_0 can be determined using the Pythagorean theorem:
$z_0 = (x_0^2 + y_0\mathbf{i}^2)$
where x_0 is the real part of the imaginary number, and y_0 is the complex part - ($z = a + bi$, $x_0 = a$, $y_0 = b$)

Thus, the square rules for a COMPLEX seed z_0 can be written:
1. if $z_0 > 1$ then the orbit will escape to infinity
2. if $z_0 < 1$ then the orbit will go to 0
3. if $z_0 = 1$ then the orbit will go to 1

When we were talking about the von Koch curve, we said that all you did was repetitively replace segments with
triangles. This is known as a TRANSFORMATION.
Transformations can also be done using functions.
The functional transformation $f(z) = z^2 + c$ *(where c is a complex constant, for example 1.05 + 2.05i)* is the basis of fractal geometry.

As we saw above, the function $f(z) = z^2$ followed three rules when iterated - *boring*. But the function $f(z) = z^2 + c$ *doesn't follow such rules!* In fact, it is infinitely complex which is a property of fractals.

But how do we SEE fractals?

To see the fractal described by $f(z) = z^2 + c$, we pick a value of c, then we iterate the function f(z) for different seed values of z(0) and see if f(z) converges on a number *(such as 1 or 0)* or goes to infinity.
We pick values of z(0) on the Cartesian plane *(X-Y plane)*
So if we have a 10 x 10 plane, we plug in:

z(0) = -5 + 5i, then z(0) = -5 + 4i, . . ., z(0) = 5 + -5i
or the equivalent points (-5,5) (-5,4) (-5,3) (-5,2) . . . (5,5)
In other words, we check *every point* on the plane and see what it does when iterated.
(Since there are infinitely many points, we actually pick only as many as is necessary to get an accurate image.)

If it **does not** go to infinity, we color it black,
the ones that **do** go to infinity, we color white.
Each picture is different depending on the value of *c*, that complex constant.
The bwfrac.bmp on the left was created with
c = -0.122 + 0.745i

3 ~BMP: bwfrac.bmp

All those black points were places where the iterated
function converged to a number.

If we want to get fancy, we can use color to enhance the fractal.

Instead of plotting the points that converge, we plot the points that go to infinity.
We use a different color to represent the number of iterations it took to reach infinity.
(e.g. 1 iteration is blue, 2 iterations is red, etc.)
Look at the same fractal colored in as *Julia* on the left.
[4]

This type of fractal, where we pick an initial c value
is known as a Julia fractal, named for French mathematician Gaston Julia.

The following is a simple BASIC program that draws a Julia fractal by the above method.
It may help you understand what you just read:

```
10        INPUT "Real Part: ", CX  //Input the real part
20        INPUT "Complex Part: ", CY //Input the complex part
30        SCREEN 1 //set graphics mode
40        FOR A = 0 TO 100 //we will test a 100x100 grid
50        FOR B = 0 TO 100
60                X0 = -2 + A / 25 //translate a screen pixel point
70                Y0 = 2 - B / 25 //to a Cartesian plane point
80                I = 1
90                WHILE I < 20
100                        X1 = X0 * X0 - Y0 * Y0 + CX //FOIL and multiply
110                        Y1 = 2 * X0 * Y0 + CY
120                        IF X1 * X1 + Y1 * Y1 > 4 THEN GOTO 180
130                        X0 = X1
140                        Y0 = Y1
150                        I = I + 1
160                 WEND
170                 GOTO 190
180                 PSET (A, B), 5
190       NEXT B
200       NEXT A
```

Try it out yourself! Those 20 lines of code were what started this whole thing!

Now, an IBM researcher named Benoit Mandelbrot decided it would be nice to see what the fractal would look like that if we fixed z_0 to be equal to 0 + 0**i** instead of the points on
the plane, and using *c* to represent the points on the plane.

The function is then iterated in a similar fashion.

4 ~FEE: instruc.fee Julia

The result is the famous MANDELBROT SET.
Click on the “Mandlebrot” on the left to see it.

[5]

There is only one possible true Mandelbrot set, since z_0 is fixed at 0 + 0i and c is always the points on the plane.
But there are infinitely many Julia sets, since any initial value of c can be used.
Be sure to check out the SHOW ME! example file
to see some great Mandelbrot and Julia fractals.

But what about those cool PLASMA fractals?

Plasma fractals are also created by iterating a procedure, but not a complex function as in
Mandelbrot and Julia fractals.

To create a Plasma fractal, the computer picks random colors for the corners of a box.
It then subdivides this box into four smaller boxes.
The colors of these boxes are determined by an average of the colors of their corners, plus a random amount.
So now we have colored boxes!

The program then divides each of these boxes into 4 smaller ones, and colors them by the
same method, adding a random value each time.
The amount of randomness is determined by the Roughness Constant that is input before
the plasma is drawn.

The program divides until each box is the size of one dot *(or pixel)* on the screen.
You are left with an amazing fractal cloud.

If we correspond each color to a different 3-d height, these smooth fractal clouds
are an amazingly real depiction of mountainous landscapes. The mountain on the title screen
was created with a plasma fractal! The higher the roughness constant, the more rugged the mountains are!
Click on “Plasma” on the left to see a plasma fractal.

[6]

Now, check out a 3-D landscape created with a plasma fractal!

5 ~FEE: instruc.fee Mandelbrot
6 ~FEE: instruc.fee Plasma

Click on the “fracmnt.bmp” on the left,[7]

Be sure to read some of the other sections for more information on Fractals!

7 ~BMP: fracmnt.bmp