

Supplement to Chapter 2 of *The Science of Digital Media* – Digital Image Representation

Programming Exercise – Digital Imaging > Algorithmic Art > Mandelbrot and Julia Fractals¹

Introduction:

A fractal is a geometric structure or image that is characterized by patterns that repeat at descending levels of detail. Fractals exist in natural phenomena – in the shapes of ferns leaves and cauliflowers; the undulations of a coastline whether seen up close or at a great distance; or the repetitive structures of mountains from the grand view to the scale of pebbles.

Fractals can also be generated graphically on a computer, created as visual reflections of computations with interesting mathematical properties. The Mandelbrot fractal, named for the mathematician who discovered it, is one of the most intriguing and beautiful computer-generated fractals, illustrating in infinite detail the property of "self-similarity." You can see this in the two pictures below. The picture on the left is the classic Mandelbrot fractal at full view. It is characterized by two main black "balls," trimmed around their circumferences with the same type of black ball. The picture on the right is the same fractal computed at a finer level of detail – as if you've zoomed in on the original fractal, but without losing any of the resolution. You can see the ball-like structures reappearing, no matter how much you zoom into the fractal. In fact, they reappear in the colored areas as well, like black pits breaking open as you get closer.

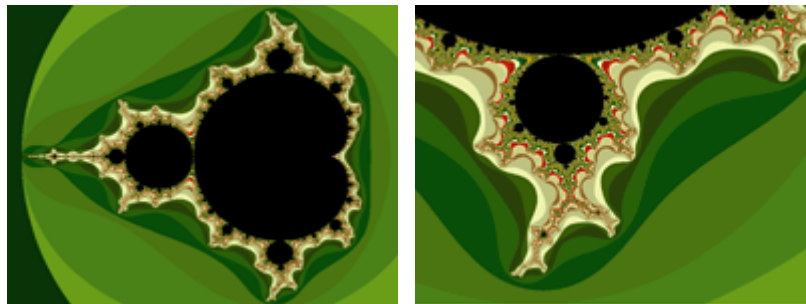


Figure 1 Two views of the classic Mandelbrot fractal

Instructions:

How a Mandelbrot Fractal Is Computed

A Mandelbrot fractal is created as a bitmap. Each pixel in the bitmap is related to a complex number by a method we'll explain below. Computation is done for each pixel to determine if the complex number to which the pixel

¹This material is based on work supported by the National Science Foundation under Grant No. DUE-0340969. This worksheet was written by Dr. Jennifer Burg, burg@wfu.edu.

is related belongs in a set called *the Mandelbrot set*. If so, the pixel is painted black. If not, the pixel is painted a color that depends on how many computations were required to discover that the point does not belong in the Mandelbrot set.

To determine the color for each pixel in the bitmap, you apply the iterative equation

$$f(z) = z^2 + c$$

where z and c are complex numbers. This equation is iterative in that you begin with initial values for z and c , and then you use the results of the i^{th} computation as the value of z for the $i+1^{\text{st}}$ computation. c remains the same as you apply the iterative equation to compute the color of a given pixel. After you compute the color for a pixel, you reset z to 0, change the value of c to relate to the next pixel, and begin the iterative equation again. The algorithm is given below.

```
algorithm mandelbrot_fractal
/*Input: Horizontal and vertical ranges of the complex number plane.
Resolution of the bitmap for the fractal.
Color map.
Output: A bitmap for a fractal.*/
{
/*constant MAX is the maximum number of iterations*/
  for each pixel in the bitmap {
    map pixel coordinates (x,y) to complex number plane coordinates (cr, ci)
    num_iterations = 0
    z = 0
    while num_iterations <= MAX and not_unbounded* {
      z = z2 + c
      num_iterations = num_iterations + 1
    }
/*map_color(x,y) uses the programmer's chosen color map to determine the color of
each pixel based on how many iterations are done before the computation is found
to be unbounded*/
    if num_iterations = MAX then color(x,y) = BLACK
    else color(x,y) = map_color(num_iterations)
  }
}
/*We have not explained not_unbounded here. For an explanation of the
termination condition and computations using complex numbers, see the
programming assignment related to this section.*/
```

So what are the initial values for z and c , and how do they relate to the pixels? Say that you're ready to compute the color for the pixel at position (x,y) . The initial value of z is 0 for the computation of every pixel. The initial value of c is gained by relating the pixel's (x,y) position to some number on a complex number plane. Complex numbers have real and imaginary components. Assume that the real and imaginary components of z are called z_r and z_i , respectively. Similarly, the real and imaginary components of c are called c_r and c_i . That is,

$$z = z_r + z_i * i \text{ and } c = c_r + c_i * i, \text{ where } i = \sqrt{-1}$$

Since c has two components, you can relate c to a two-dimensional plane where the horizontal direction corresponds to c_r and the vertical direction corresponds to c_i . This is what we call the *complex number plane*. Let's say that we're going to look at the portion of the complex number plane that goes from -2.0 to 2.0 in the horizontal direction and -1.5 to 1.5 in the vertical. (Others have figured out for us experimentally that these ranges work well for generating a Mandelbrot fractal.) Think about how you would map the pixel at position (x,y) to a number on the complex number plane, (c_r, c_i) . Assume these definitions:

- The bitmap has m rows and n columns. (For our example, $m = 300$ and $n = 400$.)
- The complex number plane goes from h_b to h_e in the horizontal direction, and $h = h_e - h_b$. (For our example, $h_b = -2.0$, $h_e = 2.0$, and $h = 4.0$.)
- The complex number plane goes from v_b to v_e in the vertical direction, and $v = v_e - v_b$. (For our example, $v_b = -1.5$, $v_e = 1.5$, and $v = 3.0$.)

Now you can use these formulas to map from a pixel position to an initial value for c :

$$c_r = h_b + \left(\frac{x}{n} * h \right)$$

$$c_i = v_b + \left(\frac{y}{m} * v \right)$$

If you try this on pixel position is (258, 210), you get

$$c_r = -2.0 + \left(\frac{258}{400} * 4.0 \right) = 0.58$$

$$c_i = -1.5 + \left(\frac{210}{300} * 3.0 \right) = 0.6$$

Thus, pixel (x,y) 's color is computed with initial values of with $c_r = 0.58$, $c_i = 0.6$, and $z = 0$.

The next thing you need to know is the termination condition for the computation. We'll give you the termination condition, since others have already worked it out for you experimentally. The computation ends when either

(a) $z_r * z_r + z_i * z_i \geq 4.0$

or

(b) 1000 iterations have been done.

Termination condition (a) is based on a property of the iterative equation. Specifically, if at any point in the computation $z_r * z_r + z_i * z_i \geq 4.0$, then z will just continue to grow after that point – i.e., the computation is unbounded. In that case, starting value c is not in the Mandelbrot set.

Doing Computations With Complex Numbers

You might be puzzled about how to do calculations with complex numbers, since they have a real and imaginary component. It's really not hard. Recall that

$$z = z_r + z_i * i \text{ and } c = c_r + c_i * i, \text{ where } i = \sqrt{-1}$$

as described above. Let z_new be the value of z that will become the input to the subsequent computation, with components z_new_r and z_new_i .

Consider first what $z^2 + c$ yields.

$$\begin{aligned} z_new &= z^2 + c = \\ &= (z_r + z_i * i)^2 + (c_r + c_i * i) = \\ &= (z_r^2 + (2 * z_r * z_i * i) + (z_i^2 * i^2)) + (c_r + c_i * i) = \\ &= z_r^2 + (2 * z_r * z_i * i) - z_i^2 + c_r + (c_i * i) = \\ &= z_r^2 - z_i^2 + c_r + [(2 * z_r * z_i * i) + (c_i * i)] \end{aligned}$$

To do this computation using only real number values (with $\sqrt{-1}$ implicit in the imaginary component of each complex number), you need to separate out the real from the imaginary components (the ones with a factor of i), as was done in the last line above. This gives

$$\begin{aligned} z_new_r &= z_r^2 - z_i^2 + c_r = \\ &= z_r * z_r - (z_i * z_i) + c_r \\ &\text{and} \\ z_new_i &= 2 * z_r * z_i + c_i \end{aligned}$$

When you finish with the iterative computation, you determine the color of the pixel based on how many iterations were done. If you do the maximum number of iterations for a pixel and never discover that the computation is unbounded, then c is in the Mandelbrot set and the pixel is painted black. Otherwise, you use the number of iterations done as an index into an array of colors in a color map that you have chosen. (Although your number of iterations will be from 1 to 1000, you don't have to have 1000 colors. You can group a range of iterations as one color. Experiment with different color maps.)

The Assignment

Write a program to create a Mandelbrot fractal using the programming language of your choice or the language required by your instructor.

You can write your program in one of two ways:

Display the fractal on the computer screen as you compute it.

OR

Write the fractal bitmap to an external file and look at it later with Photoshop, MATLAB, or some image viewing program.

Design your own color map.

Allow the user to specify the dimensions of the fractal at run-time.

Tips for Programmers



You could write this program in any number of computer languages, including Java, Macromedia Director, or C/C++.

Adobe Director is a friendly language (though not the fastest) for displaying the fractal directly at execution time because it allows you to do operations on bitmaps. You can also easily create color palettes.

Java is also a good language for directly displaying creating graphical images.

If you want to work in C/C++, you need a graphics library such as those written for XWindows.

If you write your file out to a bitmap, you can save it as a raw file with R, G, and B color channels, one byte for each. You can open such a file and look at it in Photoshop. You'll just have to specify whether you interleaved the RGB channels or wrote all the R data, then all the G, and then all the B.

More Challenges

(1) Julia fractals are computed in the same way as Mandelbrot fractals, except that z is related to the pixel position, and c is a constant that is the same for all pixels. Different constants result in differently-shaped Julia fractals.

Try to implement the Julia fractal computation. Here are some constants that produce interesting-looking fractals:

$$c_r = -1.67, c_i = 0$$

$$c_r = -1.5, c_i = 0$$

$$c_r = -1.0, c_i = 0.2$$

$$c_r = 0.3, c_i = 0.54$$

$$c_r = 0.3, c_i = 0$$

$$c_r = -0.9, c_i = 0.255$$

$$c_r = 0.255, c_i = 0$$

$$c_r = 0.26, c_i = 0.5$$

$$c_r = 0.255, c_i = 0.5667$$

(2) Mandelbrot fractal computations require a lot of calculations and can take a few seconds or even minutes, depending on your computer and the view of the fractal you're drawing. It's a program that lends itself well to parallelization. If you have access to a parallel computer, try a distributed parallel implementation, for example using a cluster of computers and MPI as the programming language.

Creative Thinking

See what creative things you can do with color maps and other pixel processing of the fractals, like rotating images or trying to create 3-D effects.