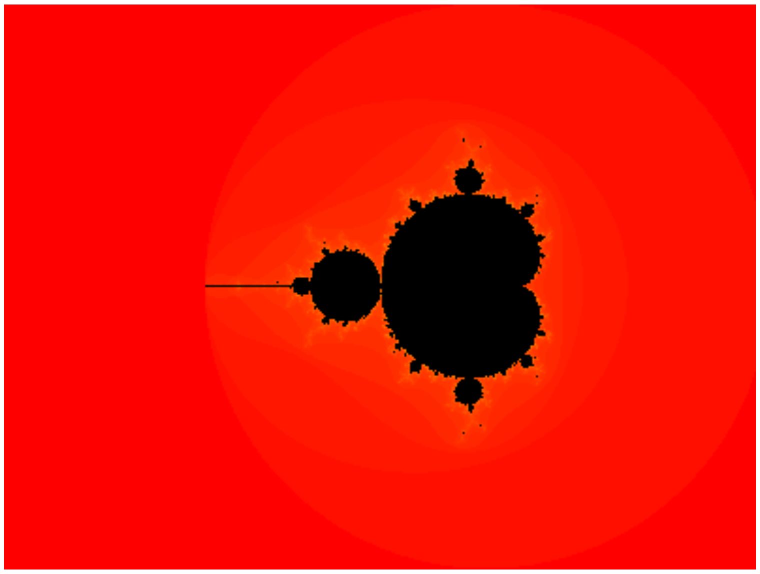
The Mandelbrot Set

A view you probably haven't seen before



The Mandelbrot Set (M) is probably the most iconic of all fractal images. The central cardioid with its multitude of disks is unmistakable. What's most remarkable about M is just how simple it is to compute. The following function is the basis of M:

$$Z_{(n+1)} = Z_{(n)}^{2} + C$$

Where Z and C are complex numbers on the Complex Plain and n is an integer representing the iteration of the formula. C is some point on the Complex plain, say `(1.0, 0.4i)`, and Z is initially `(0, 0i)`. When C is a member of M, the formula can be iterated indefinitely without the amplitude of Z ever exceeding `2.0`. Values of C outside the set will eventually escape towards ∞.

What's tricky is determining if a value C is indeed a member of M. If you iterate forever, you will never get past the first point in the set. So an arbitrary bail out value is normally set. That way, your program can finish. Here is a simple example of C Programming Language code that determines if a point is a member of the set.

```
int computePoint(double cr, double ci)
{
    int i;
    double zr, zr2, zi, zi2;
    zi = ci;
    zr = cr;
    for (i = 0; i < MAX_ITERATIONS; i++) {
        zr2 = zr * zr;
        zi2 = zi * zi;
        if (zr2 + zi2 > 4.0L) {
            break;
        }
        zi = 2 * zr * zi + ci;
        zr = zr2 - zi2 + cr;
    }
    return i;
}
```

This code performs the computation specified by the formula above for M. Instead of dealing with complex arithmetic, this example uses `IEEE-754` double precision arithmetic. The escape value of `2.0` is squared, yielding `4.0` to avoid performing an expensive square root function. The return value is the number of iterations executed by the for loop. In practice, the return value can be used to map a color for the pixel at point C so that you can make a pretty picture.

That's the traditional way to compute the points in M. Most real world code does periodicity checking or some other technique when C might be in M to reduce the total number of computations. I'm not showing any such tricks here. Instead, I wish to point out the properties of the points in M.

The orbitals image was created by tracking the cumulative effect of Z as it is iterated for different values of C in the Complex Plain. The basic idea is to start out with a black image that goes from `(-2.0, 2.0i)` to `(2.0, -2.0i)` on the Complex Plain. These intervals contain M. For each value of Z, the corresponding point in the image is located and its intensity value is incremented. Because of the nature of Z's "orbits", the log function is used and a scaler is used so that the final values range from `0` for values not in the set to `255`, the brightest intensity available for an 8 bit greyscale image. The brighter the pixel, the more times Z landed in an interval covered by that pixel.

The result is pretty interesting. So interesting in fact that to show as much detail as possible, a large image was produced. In this case, `8000x8000` yielding a 64 megapixel image. I know that computer monitors are not large enough to display the entire image without scaling it down. But I would urge you to zoom to actual size and scroll around the image. There is a surprising amount of detail to be seen.