When I watched that Veritasium video about the logistic equation, I was really intrigued by the connection between the bifurcation diagram and the Mandelbrot set. Also, in *Chaos* by James Gleick, I found Benoit Mandelbrot's story fascinating -- and the images of the Mandelbrot set included in the book were breathtaking. Wanting to recreate the beautiful complexity of the Mandelbrot set, I set out to write some code in p5.js.

Learning How To Make The Mandelbrot Set

First, I needed to figure out how the Mandelbrot set is generated so that I could do it myself. It turned out to be quite simple. In *Chaos*, Gleick provides a really helpful overview of how one can generate the Mandelbrot set:

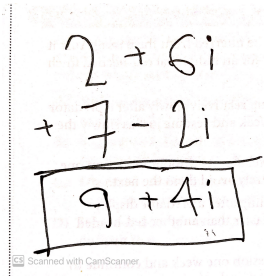
> *A Mandelbrot set program needs just a few essential pieces. The main engine is a loop of instructions that takes its starting complex number and applies the arithmetical rule to it. For the Mandelbrot set, the rule is this: $z \rightarrow z^2 + c$, where z begins at zero and c is the complex number corresponding to the point being tested. So, take 0, multiply it by itself, and add the starting number; take the result—the starting number—multiply it by itself, and add the starting number; take the new result, multiply it by itself, and add the starting number. Arithmetic [illegible] straightforward. A complex number is written with two

From my understanding, this "arithmetical rule" is the recursive sequence $z_{n+1} = (z_n)^2 + c$, where $z_0 = 0$, and we can plug in any complex number for c. To determine if c is either *in* the Mandelbrot set or *out* of the set, we need to ask our program the following question:

Does the sequence diverge to infinity?

If the answer is yes, c is *in* the Mandelbrot set. If the answer is no, c is *out* of the Mandelbrot set.

Feeling like I had a good conceptual grasp on how to draw the set, I wanted to do a quick refresher on the addition and multiplication of complex numbers before I wrote my program. I found Gleick's explanations of computations with complex numbers to be very helpful.
To add two complex numbers, we add the real part to the real part, and the imaginary part to the imaginary part. For example, we can solve (2 + 6i) + (7 - 2i) like so:

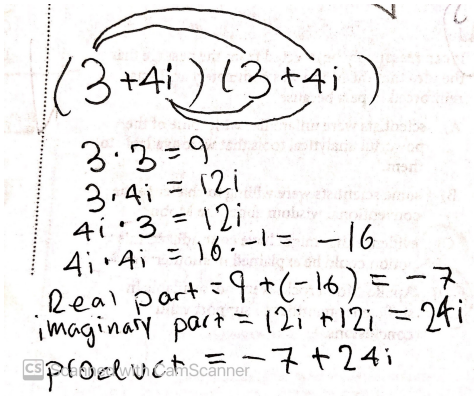


Using this knowledge, I wrote a function in my program to add 2 complex numbers, called *complexAdd()*:

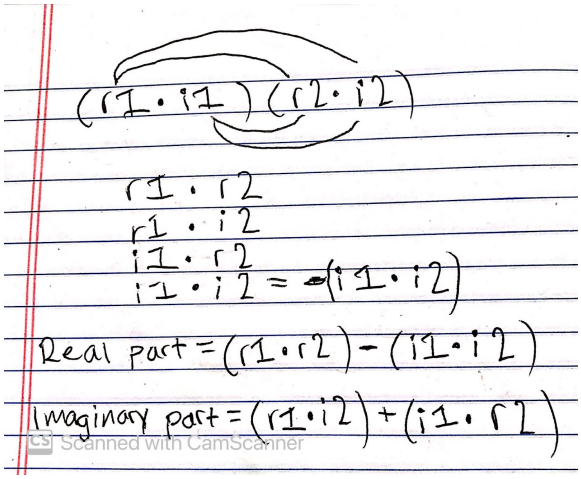
```
function complexAdd(r1, i1, r2, i2){
  return [r1+r2, i1+i2]
}
```

The function takes 4 arguments: the real and imaginary part of one complex number (*r1* and *i1*, respectively) and the real and imaginary part of another complex number (*r2* and *i2*, respectively). It outputs the complex numbers' sum as 2 values: the real part (*r1 + r2*) followed by the imaginary part (*i1 + i2)*

Multiplying two complex numbers isn't as intuitive, since the product of two imaginary numbers is a real number ($i * i = -1$). To deal with this, I prefer to treat the problem as a binomial expansion (using FOIL, better known as the "Crab Claws" method).
For example, here is how we solve (3 + 4i)(3 + 4i):



Thinking algorithmically, like a computer, we can work backwards from this process to come up with formulae for the real part and imaginary part of the product of 2 complex numbers in terms of the real parts (*r1* and *r2*) and imaginary parts (*i1* and *i2*) of the 2 complex numbers. Since a computer doesn't know how to deal with imaginary numbers, we just pretend that every number is real, and when we multiply 2 "imaginary numbers" we multiply the product by -1 (since i * i evaluates to -1):



Now that I had the real and imaginary parts of the product of 2 complex numbers in terms of the real parts (*r1* and *r2*) and imaginary parts (*i1* and *i2*) of the 2 complex numbers, I simply copied the format of the *complexAdd()* function, but inputted the appropriate expressions for the real and imaginary parts.

```
function complexMultiply(r1, i1, r2, i2){
  return [(r1*r2)-(i1*i2),(r1*i2)+(i1*r2)]
}
```

mandelbrot()

Now that I could easily perform computations with complex numbers, I was ready to generate the Mandelbrot set! I just needed a way to plug in different complex numbers, *c*, to the recursive sequence $z_{n+1} = (z_n)^2 + c$, and determine whether or not the sequence diverges to infinity. This information would tell me whether or not *c* is in the Mandelbrot set.

To accomplish this, I wrote a recursive function called *mandelbrot()* that runs the sequence $z_{n+1} = (z_n)^2 + c$ and outputs "true" if *c* is in the Mandelbrot set and "false" if it isn't. It takes 5 arguments:

- *n*, the current number of iterations into the sequence (always starts at 0)
- *Zr* and *Zi*, the real and imaginary parts of z_n (always starts at 0)
- *Cr* and *Ci*, the real and imaginary parts of *c* (the value being tested)

The function contains three parts: two base cases and one recursive step.

1. (Base Case) We check if the sequence will head off to infinity. According to James Gleick, if the real or imaginary part of z_n (*Zr* or *Zi*) is either greater than 2 or less than -2, the sequence will diverge to infinity (we can imagine this range as a 4x4 square centered at the origin of the complex number plane). If this is the case, that means that *c* is *not* in the Mandelbrot set, and we can exit the function by outputting "false":

```
if(Zr >= 2 || Zr <= -2 || Zi >= 2 || Zi <= -2){
  return false
}
```

2. (Base Case 2) We check if the sequence will *not* head off to infinity. Gleick claims that if the sequence iterates at least hundred times without z_n exiting the range [-2,2], *c* is most likely* in the Mandelbrot set. For our program, this means that if *n* -- the number of iterations -- reaches 100, then *c* is in the Mandelbrot set and we can exit the function by outputting "true":

```
if(n>100){
  return true
}
```

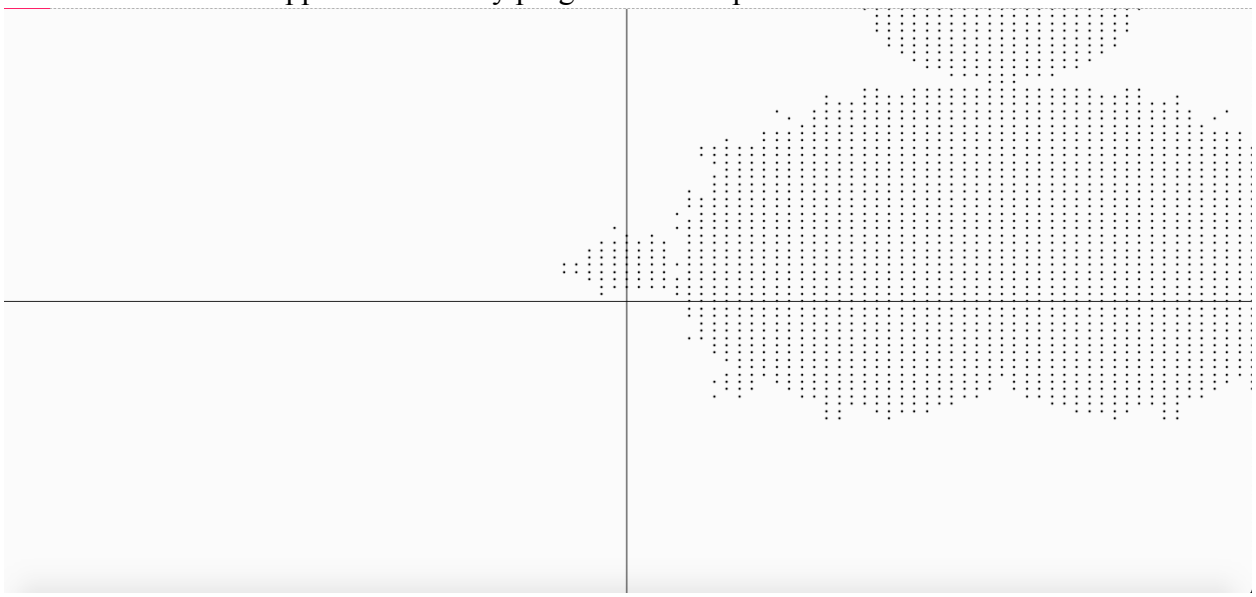
**I write "most likely" because we can actually never be certain, for some values of c, if c is in the Mandelbrot set. Who's to say that z_n won't diverge to infinity after googol iterations? 100 iterations is safe, but if we want to zoom in a lot, we should check more than that.*

3. (Recursive Step) If this last part of the function is reached, this means that the program has yet to decide whether or not z_n will diverge to infinity. So, we calculate the next term in the sequence, and feed that value right back into the *mandelbrot()* function.
    a. First, we find the $(z_n)^2$ part of the equation using the *complexMultiply()* function:
    ```
    let ZnSquared = complexMultiply(Zr,Zi,Zr,Zi)
    ```
    b. Next, we add *c* to $(z_n)^2$ using the *complexAdd()* function:
    ```
    let sum = complexAdd(ZnSquared[0],ZnSquared[1],Cr,Ci)
    ```
    c. Lastly, we input the next term of the sequence, *sum,* into *mandelbrot()*, increasing *n* by 1 (*++n*) to reflect that there has been 1 more iteration.
    ```
    return mandelbrot(++n, sum[0], sum[1], Cr, Ci)
    ```

Generating the Set

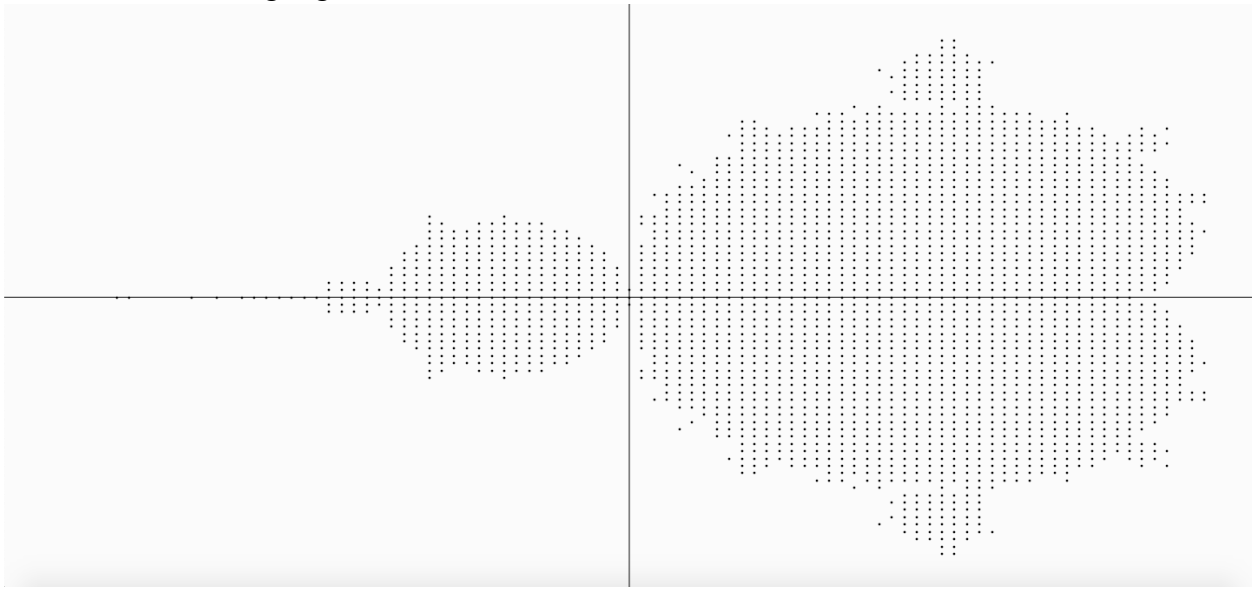
Now that I had a way to check if any complex number *c* is in the Mandelbrot Set, I could actually draw the set! I iterated through a grid of points on the complex number plane, inputting each point into *mandelbrot()* to determine if the point lies in the Mandelbrot Set. If so, I plotted a black dot at that value's location on the complex number plane.

I felt such a rush of happiness when my program first outputted this:

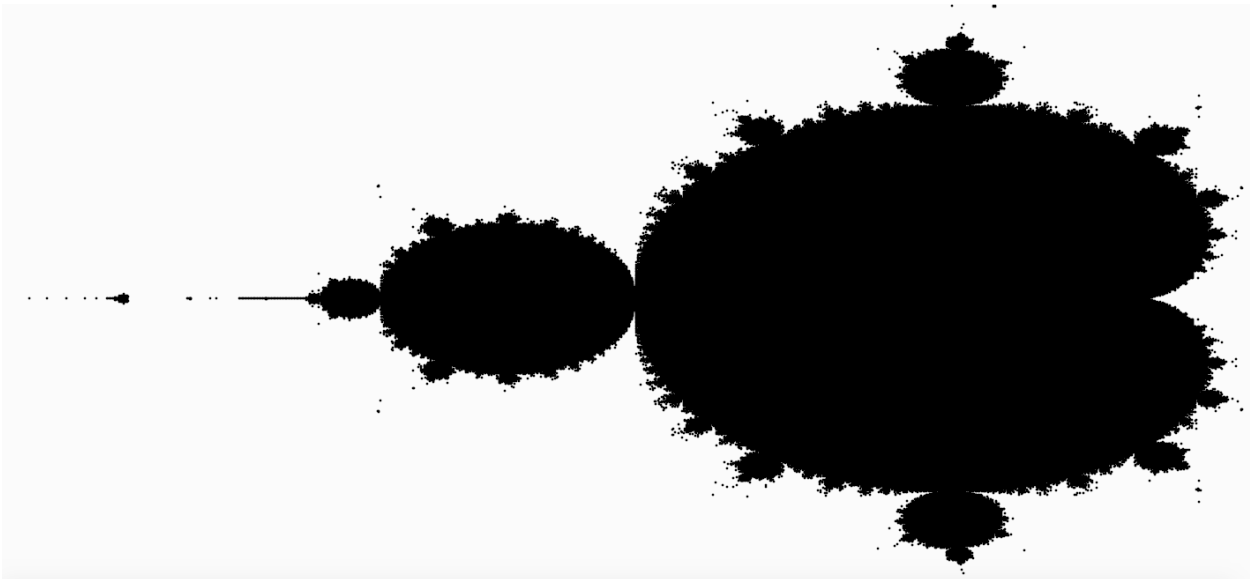


I was so excited because I immediately recognized what my program had created: the Mandelbrot set! This feeling was bittersweet, though, because for some reason it was … sideways? I soon realized that I made the silly mistake of switching the x- and y-coordinates when I plotted the points, which resulted in the rotation!

When I fixed that bug, I got this:



By checking many more complex numbers (i.e. making smaller increments between points), I created a much more refined Mandelbrot Set:



Colors

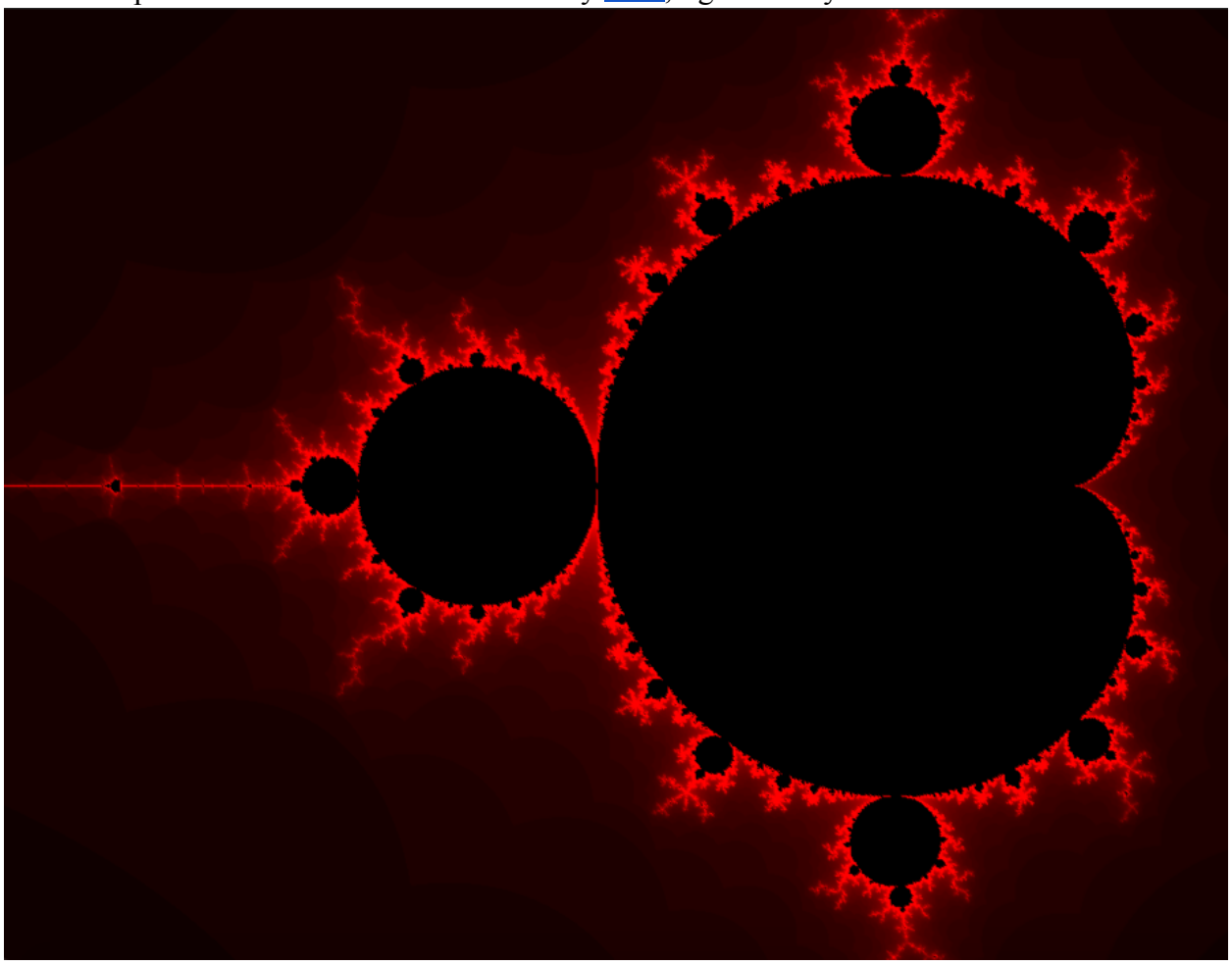
Although I was pretty happy with what I had generated, I wanted to make it more aesthetically pleasing. Having played around with Google's interactive Mandelbrot set viewer, I knew I could play around with colors a bit to make it more pretty. I did some research and learned that I could color each point *outside* of the set based on the number of iterations it takes for that *c*-value's sequence to exit the range [-2, 2] in either its real or imaginary part. That is, *c*-values with sequences that exit the range faster (it takes fewer iterations) are darker, whereas *c*-values with sequences that exit the range slower (it takes more iterations) are brighter.

For example, when we plug in -0.56 + 0.66i for *c*, the sequence takes 21 iterations to exit the range [-2,2]. If we set 40 iterations as the "maximum iterations,"* and then map (using p5's built in *map()* function) this number of iterations (10) from the range [0,40] to the range [0,255] (the "red"** component of the RGB scale -- each of the 3 colors has a range of [0,255]), we get an RGB code of (133.875, 0, 0) which looks like this: . Numbers further from the Mandelbrot Set break off after fewer iterations which results in darker shades of red. For example, the *c*-value -0.9 - 0.5i breaks off after 15 iterations, which results in an RGB code of (95.625,0,0), which is a darker shade of red:

*For higher magnification, the "maximum iterations" should be higher than 40 because points closer to the Mandelbrot Set take many more iterations than 40 to break out of the range [-2,2]

**I could've chosen red, green, or blue (or even a combo of them), but I chose red because I thought it looked cool.

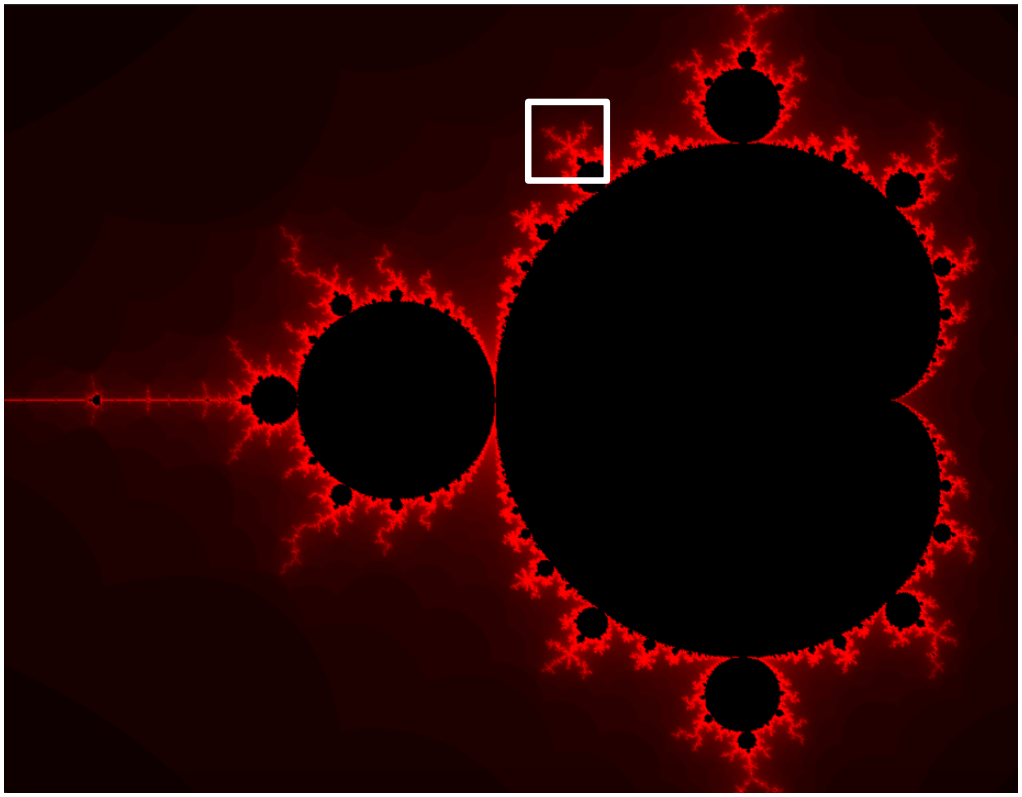
When I implemented this color scheme into my code, I got a really cool result:

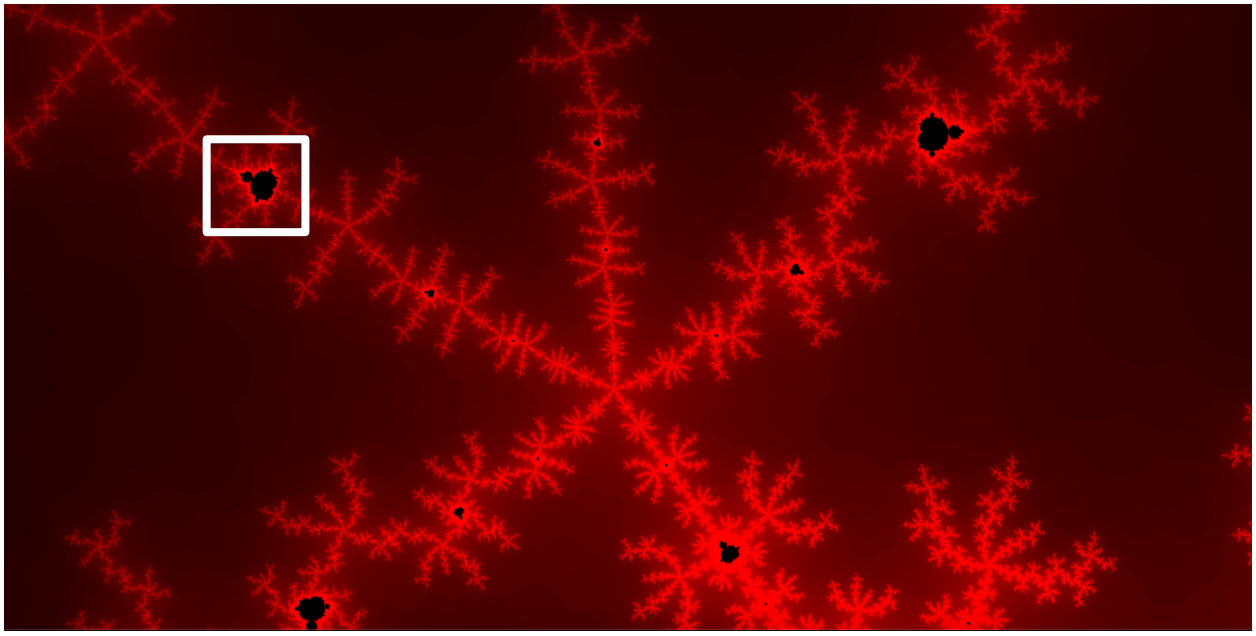


Not only is this just awesome to look at, but it's also interesting to see how the *c*-values closest to the Mandelbrot Set are colored the brightest, meaning it takes them longer to diverge to infinity. I remember in *Chaos*, Gleick had a cool analogy for this: it's almost as if the *c*-values surrounding the Mandelbrot set are being pulled into the set, like a magnet. Points closest to the set are the most attracted to the set, which is why it takes so long for them to diverge to infinity.

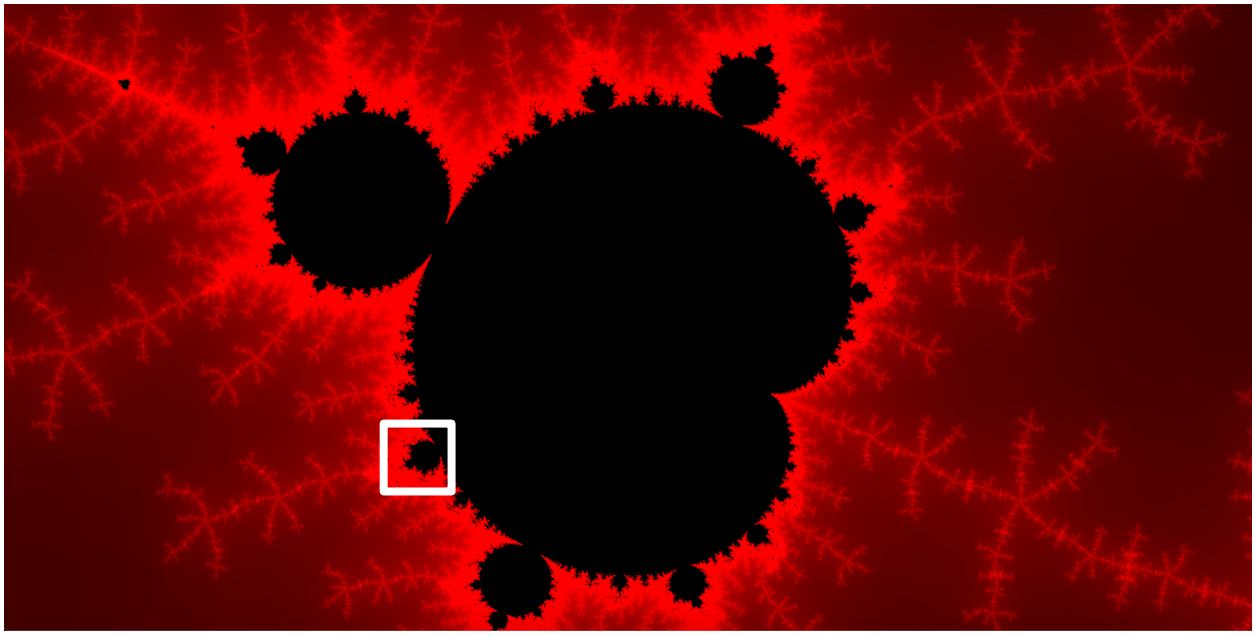
Zooming In

What I find really weird is that the designs created by this color scheme appear to have fractal attributes! We can see this fractality by zooming in more (note that as I zoom in, I also need to increase the "maximum iterations" when scaling the coloring so as not to lose any details)

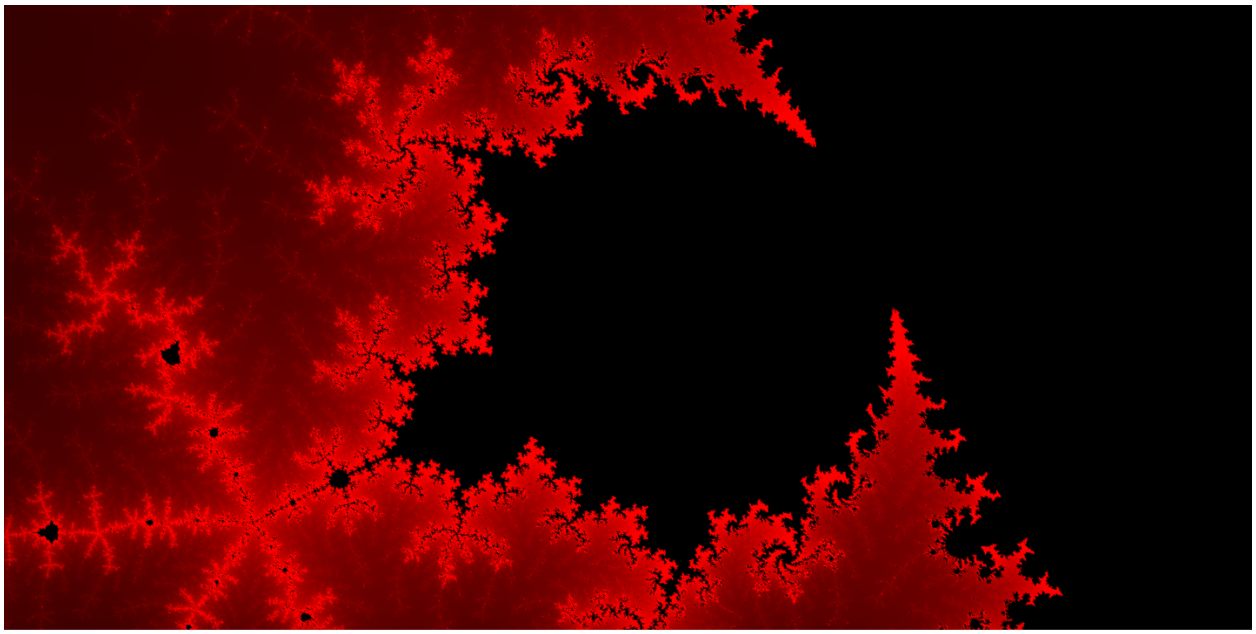




And the Mandelbrot set appears to be a fractal in addition to the color patterns outside the set because I'm finding tiny "Mandelbrot set islands" around the main Mandelbrot set.

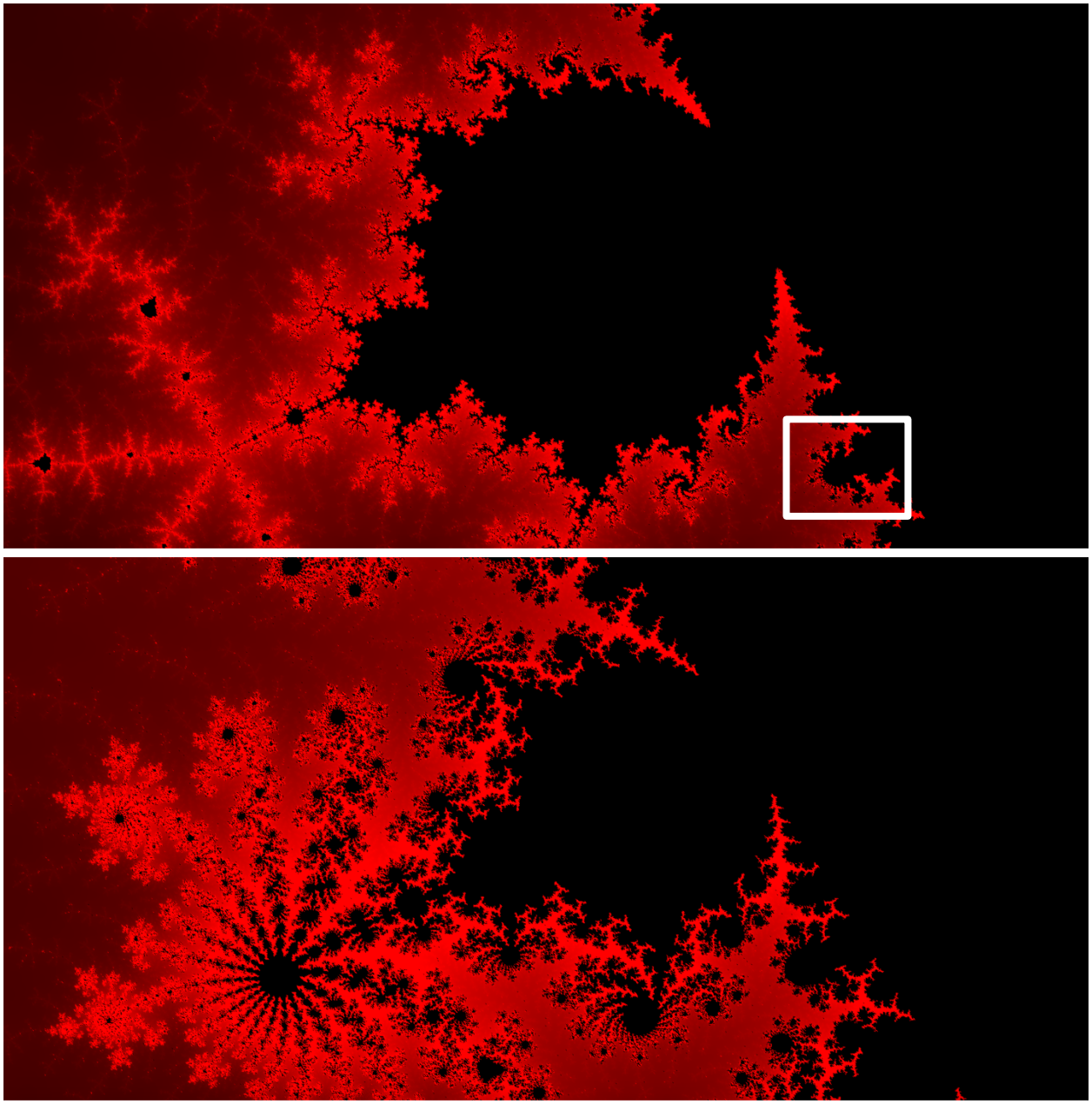


!



I was shocked when I zoomed in here. I was expecting for that little “bulb” off of the island* to look exactly like the Mandelbrot set I had been seeing on all of the previous scales I had zoomed in on, except it’s different. Although the same general structure of the shape is maintained, it’s clearly a different, and arguably more beautiful and intricate shape. My mom said it looks kind of like a coral reef. I was planning on stopping here, but I wanted to keep zooming…I was especially intrigued by this weird-looking structure in the bottom right:

**Although I am calling them "islands," I learned in Chaos that the Mandelbrot set is actually completely contiguous. I just think the parts that connect the "islands" are so tiny that my program skips over them and doesn't even check those values.*



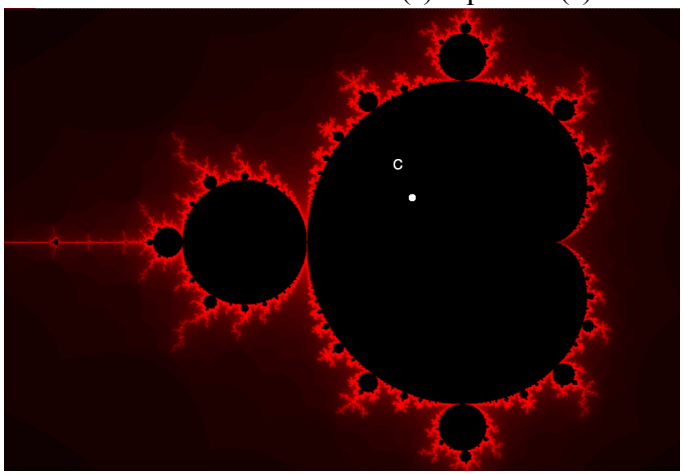
This scale is my favorite -- there's just so much going on. Considering that I found newer and newer structures on smaller and smaller scales, I wonder if there are any new structures on even smaller scales. I think people often call the Mandelbrot set a fractal, but I'm wondering if that's actually accurate? From what I've seen so far, it doesn't appear to be self-similar on very small scales… isn't that a necessity for something to be called a fractal?

Connection to Bifurcation Diagram

In addition to zooming in really far, I also wanted to explore the connection to the logistic map. In the Veritasium video, I learned that different "bulbs" correspond to different periods -- that is, a *c*-value in the "main" bulb yields a sequence with a period of 1, whereas other bulbs yield different periods.

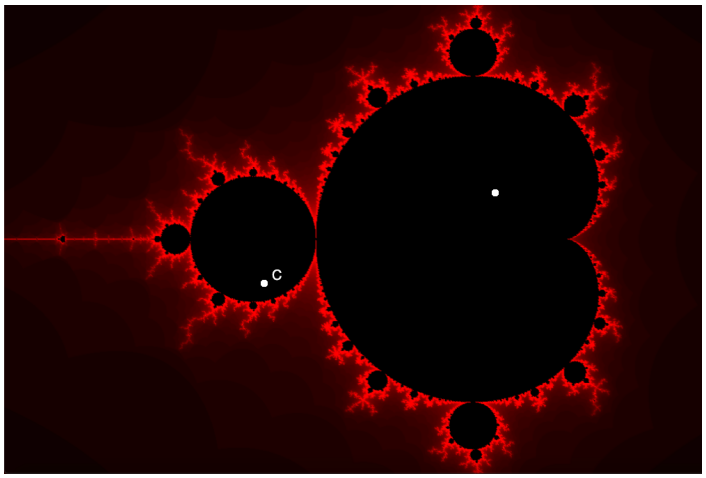
I wanted to see these different periods, so, inspired by this Numberphile video, I added in a feature to my code that allows me to see the long term behavior of each point on the Mandelbrot set. Wherever my mouse goes, my program calculates the first 500 terms of the sequence for the *c*-value corresponding to my mouse's location on the complex number plane. Then, it deletes the first 400 terms, and plots the 400-500th terms as white dots on the complex plane. This way, I can see, for any given *c*-value that we plug into the sequence, the long term behavior of that sequence.

For example, here's the long term behavior when *c* is in the "main" bulb (The location of the letter "C" represents the value for *c* and the white dot(s) represent(s) the long term behavior):



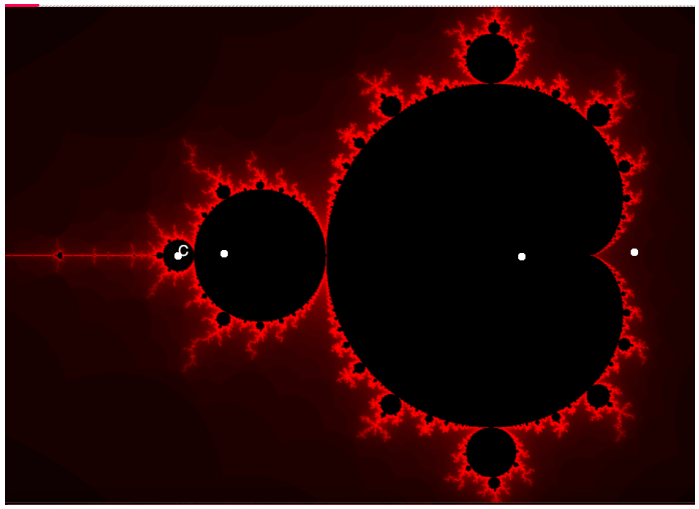
The sequence settles to a single value. Thus, it has a period of 1

Here's the behavior when *c* is in the bulb to the left:



The period *doubles* to 2

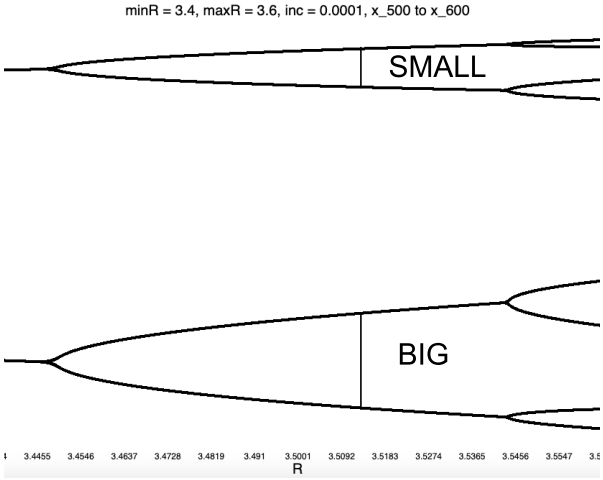
What if we move *c* to the next bulb to the left?

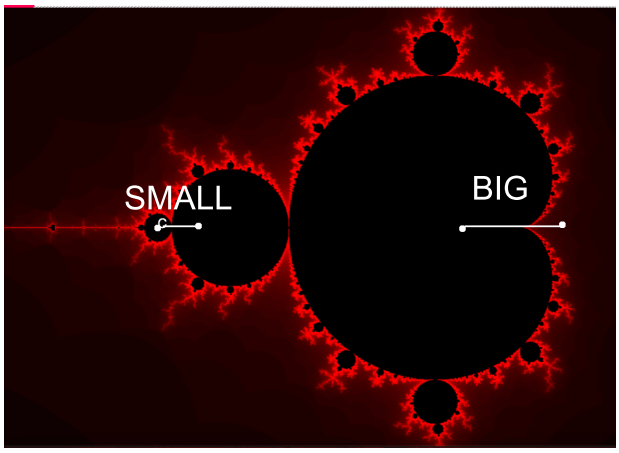


The period *doubles* again to 4

Just like the logistic equation's long term behavior's period *doubled* as we increased *r*, the period of the long term behavior of the sequence we use for the Mandelbrot set also *doubles* when we change the value of *c*!

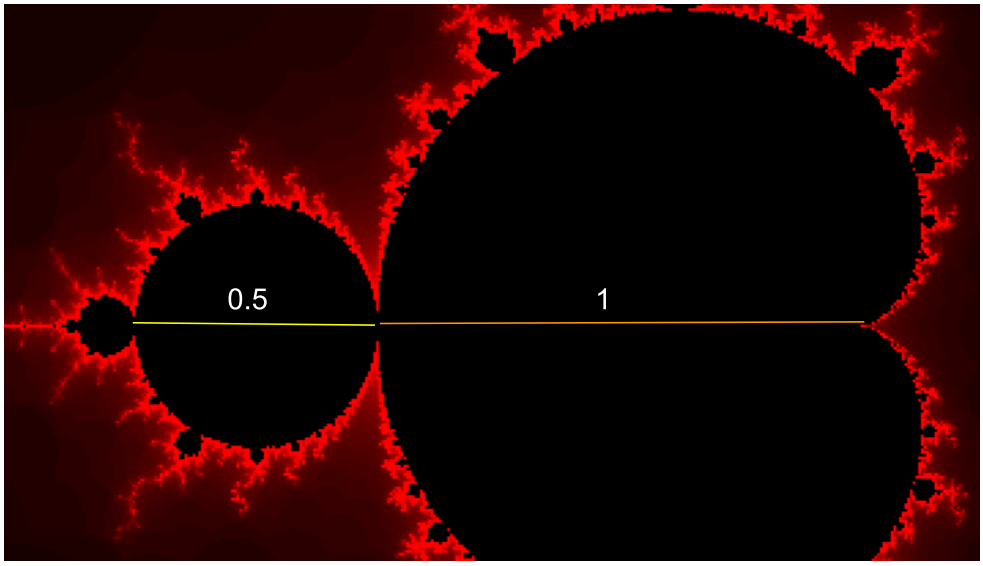
It's also interesting to see how the relationship between the distances between the points are exactly the same as the logistic map in the period 4 region:





Feigenbaum's Constant

Now that I had seen period doubling, I was wondering if I could find Feigenbaum's Constant. So, I measured the length of some of the bulbs:



1 / 0.5 = **2**

0.5 / 0.1181 = **4.234…**

.1181 / .02624 = **4.5008…**

.02624 / .005599 = 4.68655…..

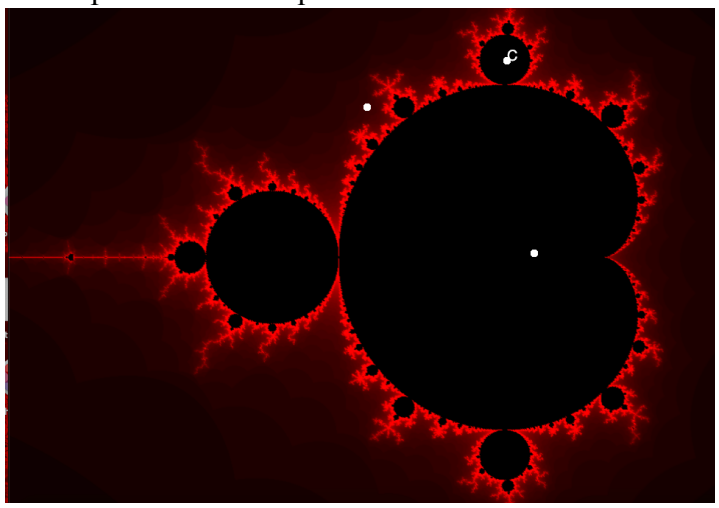
I got pretty close to 4.699. If I were to keep going I think I'd get pretty much exactly the Feigenbaum Constant (but only to a certain point because of rounding)

It's very tedious to take these measurements, so I decided not to measure another set of bulbs to get the constant elsewhere in the Mandelbrot set, however I think it should work as long as I follow one line of bulbs (meaning that when we go from one bulb to the next, the period doubles each time).

Period Lengths

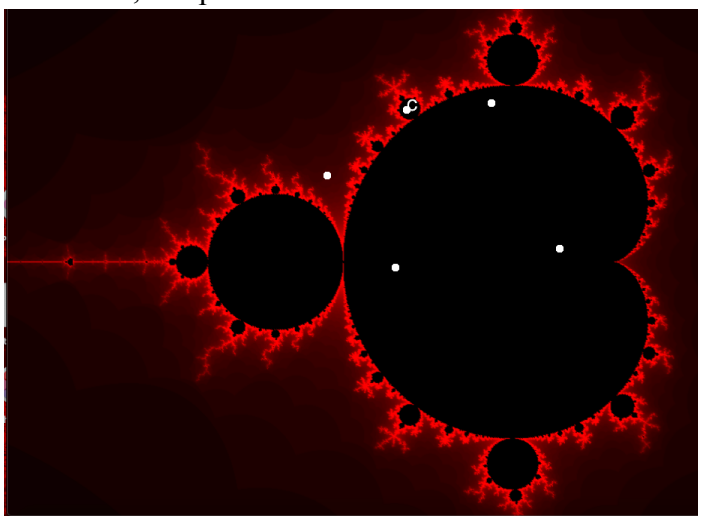
In the logistic map, we only have period *doubling*. But, in the Mandelbrot set, there's also period *tripling, quadrupling, quintupling,* and so on…

For example, check out the period of the sequence when *c* is in *this* bulb off of the main bulb:

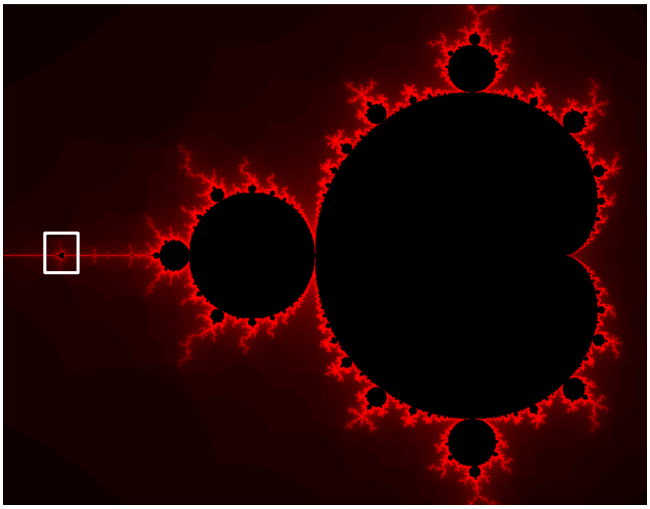


Our period is THREE!

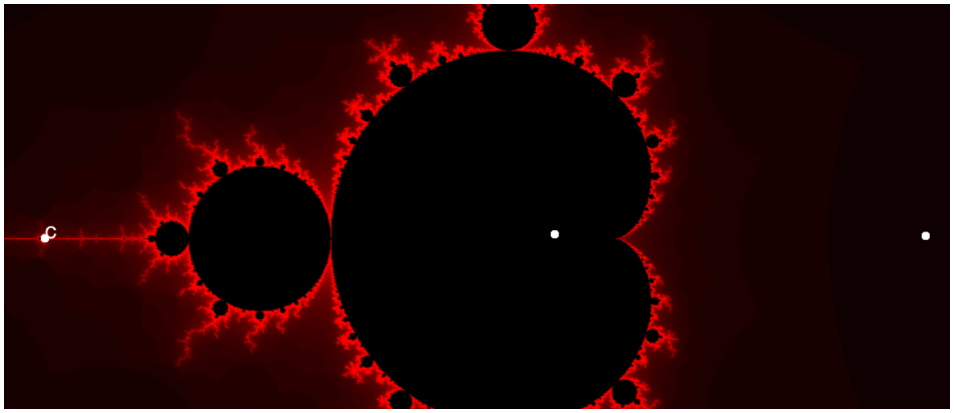
If we move *c* to a different bulb, our period is 5:



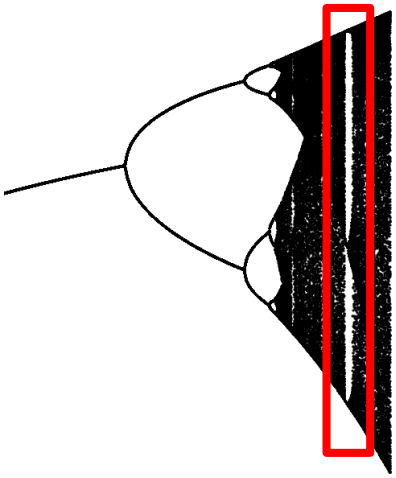
I also thought it was so cool that this little piece of the set (which is actually just a tiny Mandelbrot set!!!)



has a period of THREE:

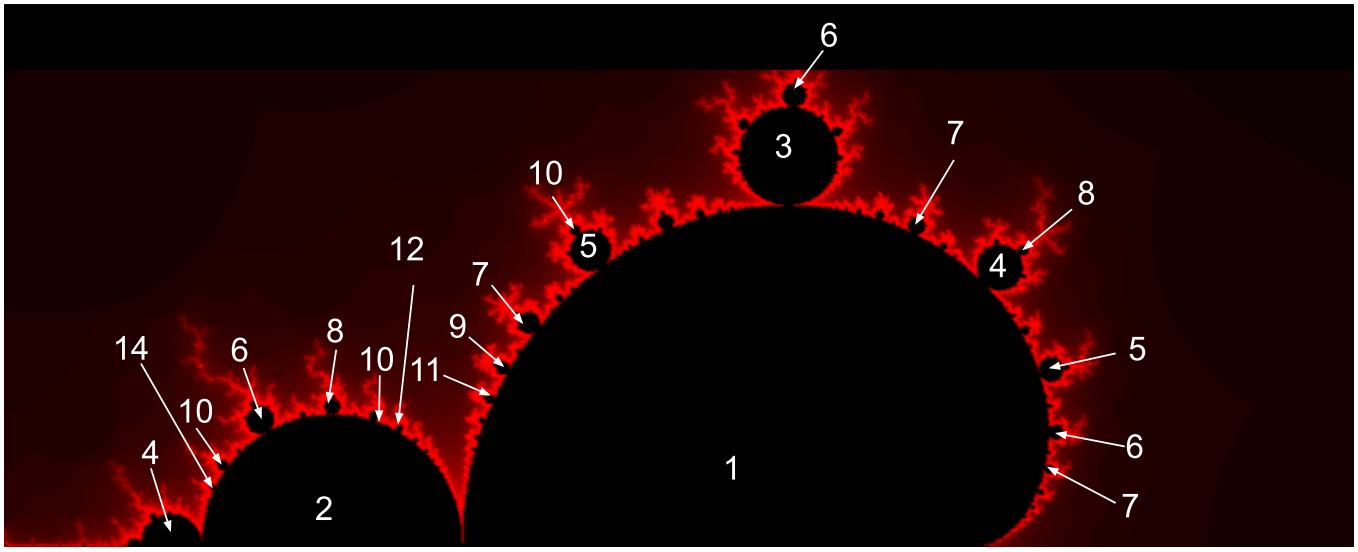


Which is actually EXACTLY what happens in the bifurcation diagram with that little area of stability:

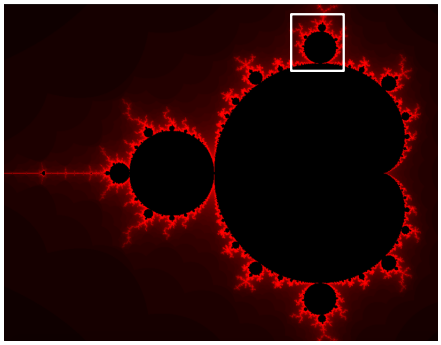


It's just so weird that this region ALSO has a period of 3.

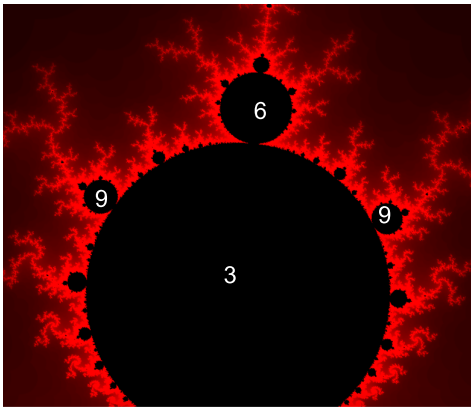
Here's a diagram of a bunch of different periods of different bulbs I was able to find:



1. In general, it seems that smaller bulbs have larger periods, and vice versa.
2. Another thing I noticed is that there are patterns between which bulbs result in period *doubling,* which bulbs result in *tripling*, and so on. Let's start by examining this bulb which has a period of 3:

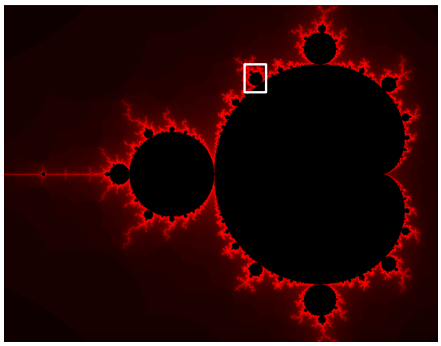


It has certain bulbs which branch off of it that have larger periods:

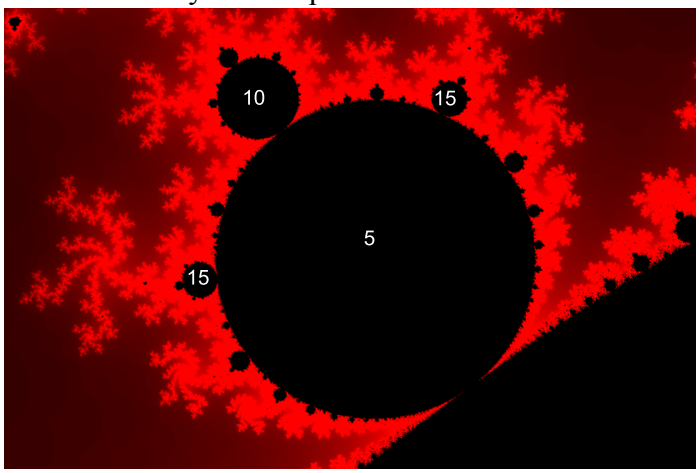


And note that when we go from the period 3 bulb to the bulb on the very end, the period *doubles* (from 3 to 6), whereas if we go to one of the side bulbs, the period *triples* (from 3 to 9).

Now, let's select a different bulb off of the main bulb with a different period:



We see very similar pattern with *its* own bulbs:



Just like the period 3 bulb, the bulb at the end of this period 5 bulb makes the period *double* (from 5 to 10), and the 2 bulbs on either side make the period *triple* (from 5 to 15)

So- not only is the Mandelbrot set self similar in terms of its shape (well, sort of) it is also seems self similar in terms of the way that the periods change from any given bulb to its "branch bulbs" (the smaller bulbs that connect to a bulb)

Julia Sets

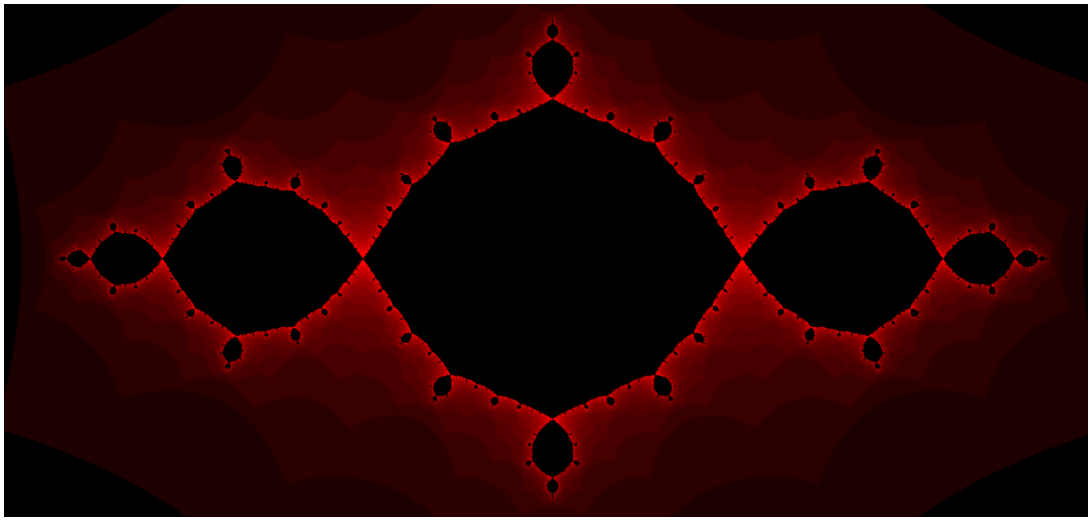
I also watched this Numberphile video about Julia Sets. They are very similar to the Mandelbrot set in that they are both based off of the sequence $z_{n+1} = z_n^2 + c$

However, there is an important distinction:

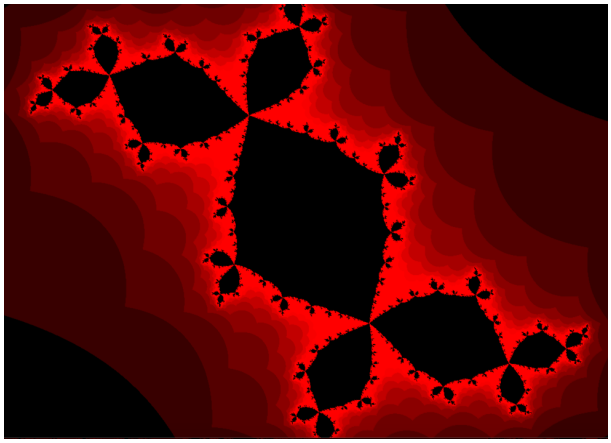
- For the Mandelbrot set, we ALWAYS keep z_0 at 0 (that is, the sequence always begins at 0) and we include any given number in the set if, and only if, when we plug that number into the sequence for ***c*****,** the sequence stays bounded.
- For any given Julia Set, we always keep c at a certain value (it can be any value-- which is why there are Julia *Sets* plural), and we include any given number in the set if, and only if, when we plug that number into the sequence for z_0, the sequence stays bounded.

Because the distinction is so simple, making my program generate Julia Sets only involved changing a couple lines of code.
Here are some cool Julia Sets I was able to generate:

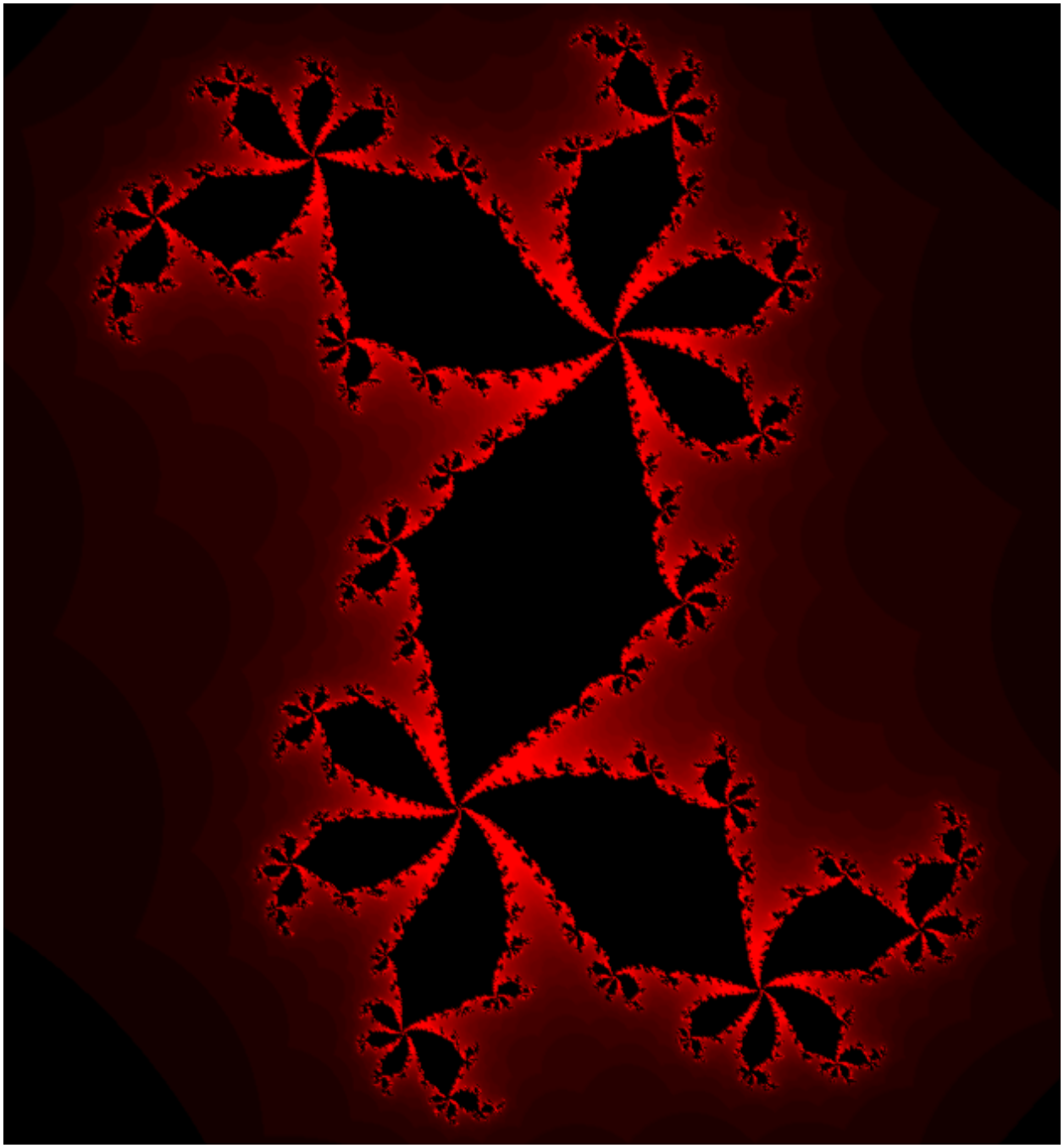
$c = -1 + 0i$



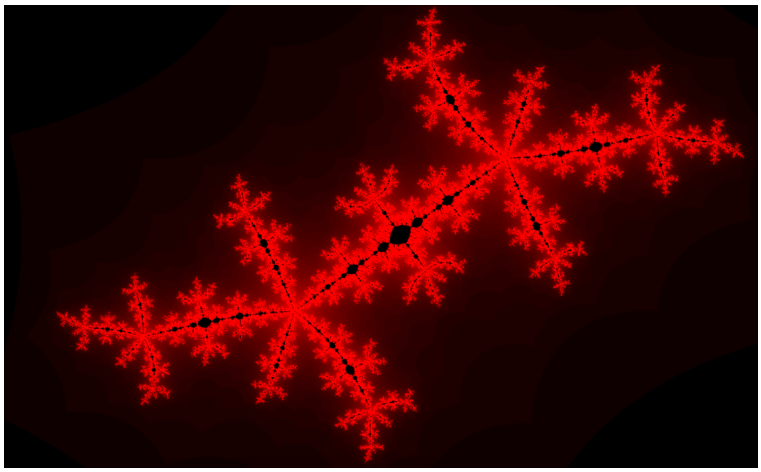
$c = -0.12 + 0.75i$



$c = 0.382 + 0.331i$



c = -0.54055 - 0.6127i



More Mandelbrot

I'm starting to wonder if there will always be new structures to discover in this shape. It's really not a conventional fractal.