Fractals, Chaos, and Computer Graphics

Tiziana H., Jacob P., Jaheim L.

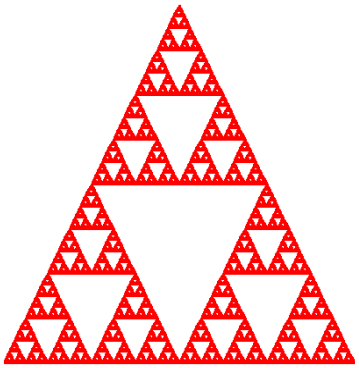
Introduction to the lab and how the math and computer science interact and assist each other:

The play “Arcadia” by Tom Stoppard has themes of showing the beauty of nature through mathematics. This is first shown when Thomasina wanted to plot things of nature because she believed nature is written in numbers. She plotted iterative equations- taking the values she calculated for y and using them as her next x. Thomasina was iterating by hand, which would take an inconceivable amount of time to depict the image she was working towards. With the technology of a modern computer, we can now plot the iterative equations Thomasina was attempting to plot by hand in moments.

Many years after Thomasina’s discovery, another mathematician named Valentine was attempting to find the population changes of the grouse from year to year. Valentine was most likely using the logistic equation to find the populations of the next generations of grouse.

In this lab, we created programs to generate the fractal images that Thomasina was trying to draw by hand. We also created experimental programs using the logistic equation that Valentine most likely dealt with.

Sierpinski Recursive:

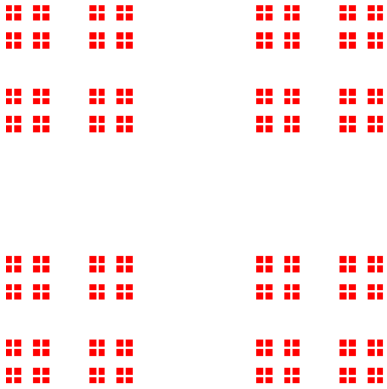


The goal of this program was to create Sierpinski's triangle with a recursive method. Sierpinski's triangle is a simple fractal that draws three triangles inside of a bigger triangle and repeats this process using recursion as shown in the picture to the left. In our program, we have a recursive method called sierpinski which handles drawing the triangles using the midpoints of the previous triangle.There are three recursive calls that each handle a different smaller triangle of the fractal. These recursive calls will cause Sierpinski to run again except with the different x and y values of the midpoints. Note that the calculations for the midpoints will be done using the points passed through Sierpinski's method parameters.

Sierpinski Iterative(Chaos):

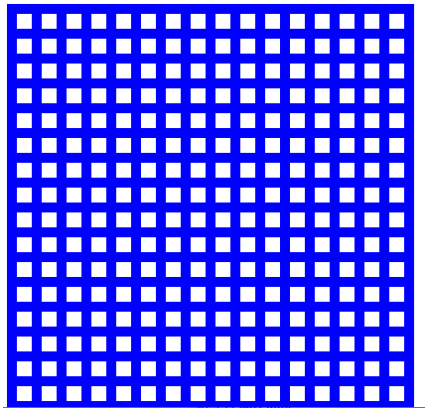
The result of this program is the same triangle as pictured in Sierpinski Recursive. Instead of recursive calls, we used a for loop that iterates a set number of times. First, we set a variable "w" equal to one of the three original vertices that were hardwired into the program. Then inside the for loop, we get another random vertex. After finding these two points, a midpoint is drawn between them. The coordinates of "w" are then set equal to the coordinates of the midpoint.

Square:



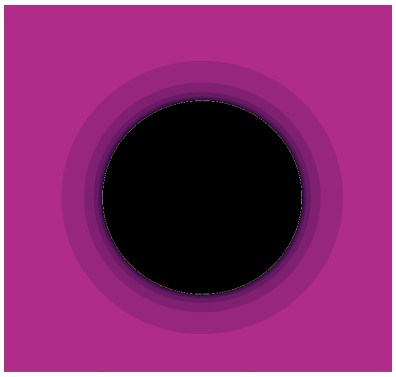
This program was more practice of how to create a fractal given a set of steps to help us create the picture to the side. For this program, we had to plot points ⅓ of the distance from the vertices of the square to another random vertex or midpoint of the square. To do this we found a distance formula that calculated ⅓ of the distance for us. We looped through this process 990,000 times to display the image to the left.

Our Own Fractal (Shape):



To further our understanding of fractals, we designed and implemented our own fractal of squares inside of squares. We used a similar setup to Sierpinski recursive. We calculated midpoints of the 4 sides of the square. Using these points, we made four recursive calls to draw squares inside of squares, which resulted in a grid-like pattern. We changed the color of the fractal to blue and changed the line stroke to a thicker setting to give the fractal a different look.

Julia Set:

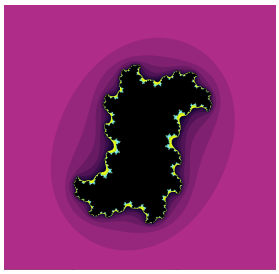
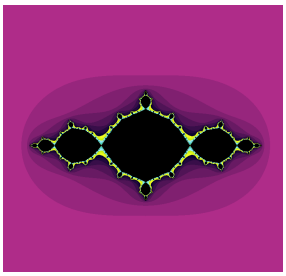
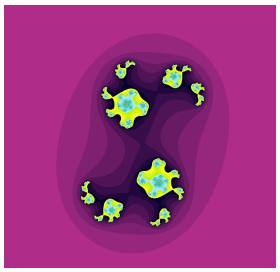


The Julia Set takes the points from a real and imaginary plane and determines whether or not that point becomes part of the “escape set” or “prisoner set”. If the point is a part of the prisoner set, it will be colored black. Otherwise, it will be assigned a color based off of its escape speed. To determine this, we were given the function $f(z) = z^2 + c$ and we passed in the complex number “z”, which had a real and imaginary part, which were obtained from the graph, and the constant value “c” which was given the set value of (0,0). The value we get from f(z) would then be used as the next "z" value. This process will continue to occur until it either exceeds a certain value, in which it would be considered an escapee, or reach a limit of 50, which means it is a prisoner. To create this fractal image, our code looped through each point on the frame and created a complex number out of them, this would be our “z” value. That “z” value and “c” were put into the function and looped through until it exceeded a certain value or failed to do so before the limit was reached. We had a variable that kept track of how long this process took and that variable would be passed into another method to determine the color of that point.

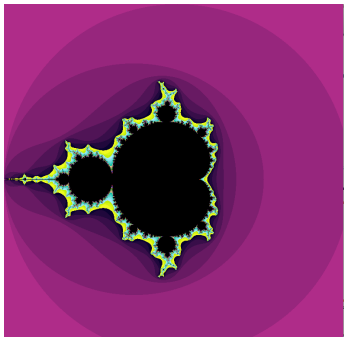
Julia Set With Varying Constants “c”:

In these programs the entirety of the process is the same as in the Julia Set except the constant value of “c” has been adjusted in each program, the changed values are shown below.

JuliaSetA: c = (.3,.4) JuliaSetB: c = (-1,0) JuliaSetC: c = (.5,.3)

Mandlebrot Set:

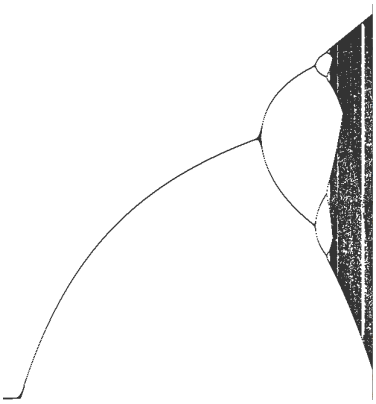


The Mandlebrot Set behaves in a very similar way to the Julia Set in that complex numbers are obtained from the graph and are put into the function $f(z) = z^2 + c$, except this time those complex number are now the “c” value and the constant value is “z” which is set to (0,0), otherwise the process is the exact same. A complex number is obtained and is made out to be “c” and is then passed into the function with “z”. The value we get from f(z) would then be used as the next "z" value. This process will continue to occur until it either exceeds a certain value, in which it would be considered an escapee, or reach a limit of 50, which means it is a prisoner. We had a variable that kept track of how long this process took and that variable would be passed into another method to determine the color of that point.

Growth Rate Chaotic Experimentation:

Using the equation **y = rx(1-x)** (where r = growth rate and x = fraction of the max population), we were asked to find the first value of “r” that would cause the population to be chaotic. To figure out this value, we used a loop that would iterate through different values of “r” starting from 3.5 with “r” increasing by 0.01 each time. In the loop, each value of “r” was used to calculate the equation 20 times with a starting “x” value of 0.5. The results of the calculations were printed out so we could locate the “r” value. With this program, we deduced that the “r” value was roughly 3.579.

Logistic equation plotting:



For this program, we needed to plot the logistic equation of a population that begins at half capacity and grows at different rates ranging from 0 to 4. The goal of this program was to analyze the graph for the point at which chaos in the population would occur. We avoided plotting the first 100 generations to allow for stabilization of the population. (The population could become extinct, stable, periodic, or chaotic in this time). For generations 101- 500, we plotted the logistic equation for the various growth rates. The graph to the left shows that as the rate increases for a population at half capacity, the population goes from stable, to periodic, to chaotic. Due to the population being a fraction, we scaled our graph in order to see the results.

Butterfly Effect (growthRate1):

For this problem, we had to demonstrate that making small changes to a population (x) we use in the equation **y = rx(1-x)** could result in very different outcomes. In order to do so, we calculated the equation with two slightly different values of “x” whilst using the same “r” value. We used two separate but identical loops that would iterate through the equation 20 times with each “x” value. Our first “x” was 0.6 and our second “x” was 0.61. Once the loops were done, we got the following results:

X = 0.6	**X = 0.61**
0: 0.96 **1: 0.15360000000000013** **2: 0.5200281600000003** **3: 0.9983954912280576** **4: 0.006407737294172653** **5: 0.02546671278776609** **6: 0.09927263731020608** **7: 0.3576703231667294** **8: 0.918969052370147** **9: 0.297859732624244** **10: 0.8365572492210314** **11: 0.5469168719870904** **12: 0.9911952284917879** **13: 0.034908990027601214** **14: 0.1347614097714162** **15: 0.46640308883134657** **16: 0.9954849902397024** **17: 0.017978497788648136** **18: 0.07062108562364684** **19: 0.2625349915559375**	**0: 0.9516** **1: 0.18422976** **2: 0.6011566221213696** **3: 0.9590693512039777** **4: 0.15702132314063577** **5: 0.5294625088791992** **6: 0.9965278422821724** **7: 0.013840407354440547** **8: 0.05459540191481478** **9: 0.20645897601829846** **10: 0.6553346689590965** **11: 0.9034845624774716** **12: 0.34880083136945345** **13: 0.9085552456217261** **14: 0.3323304450998843** **15: 0.8875476814383884** **16: 0.3992271784469178** **17: 0.9593793537449226** **18: 0.15588243741158922** **19: 0.5263324124728448**

By comparing the corresponding numbers, we saw that a .01 difference in population can lead to big changes in the population through generations.

Impressions of the lab and/or enrichment: Discuss what you liked best, what you liked least, and what you learned.

Overall, we thought this lab was a good introduction to the course. We reviewed our graphics programming knowledge along with developing an understanding of how fractals, and things in nature can be plotted.

We strongly detested the Square program for it took many moons to discover what was being asked of us and many more to figure out the proper equations and algorithms for the program. Overall 4/10 would not do it again :(here's what we struggled with ↓

“In our square program, we struggled to figure out how to plot a point ⅓ the way along a line, but found an equation which allowed us to do so. For this program, an infinite loop did not display an image because the computer would keep running and never reach setVisible. To avoid this problem, we set the loop to iterate 990,000 times rather than infinitely to display the fractal.”

We found the logistic plotting program to be quite exquisite because it gave a visual representation of chaotic populations and stuff like that