Introduction

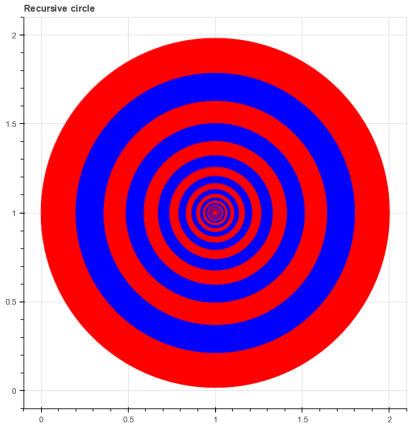
I wanted to learn to use recursion to generate fractals and self-similar shapes with interesting graphics like color gradients, starting from simple shapes like the Sierpinski triangle to generating a Mandelbrot Set and Julia sets.

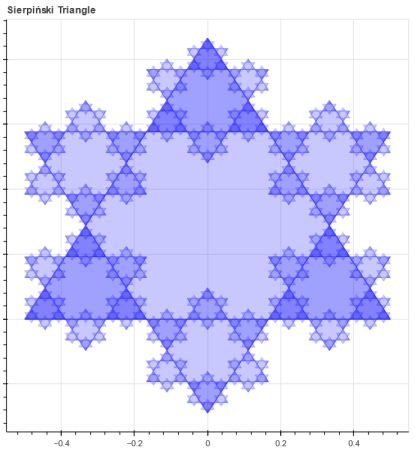
My research question is similar to the techniques learned, namely.

- How can the mathematical characteristics of recursive structures be used to create Python simulations?
- How do recursive depth and color schemes affect the appearance of a fractal?
- How can color choices lead us to draw conclusions about the behaviour of iterated functions?

Part 1: Simple Recursive Shapes

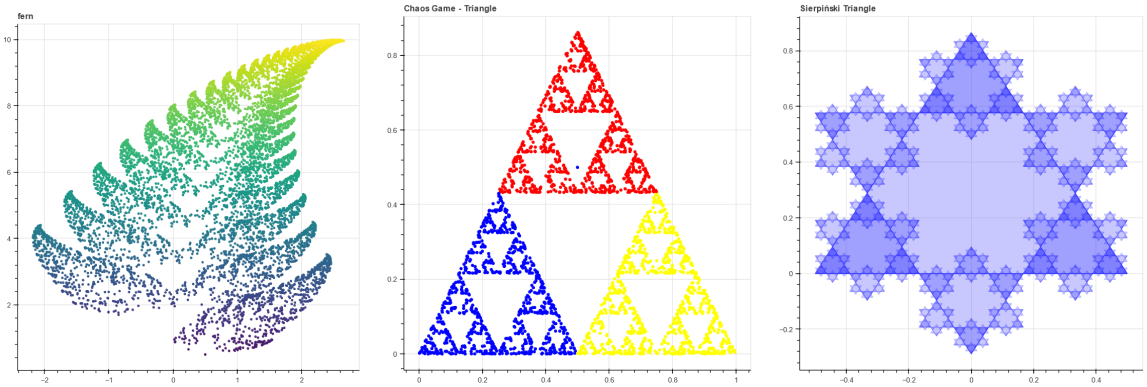
I started by learning to generate famous simple fractals. Due to the sometimes high rending times of these images, images are provided. These shapes were simulated intuitively, by looking for well-known fractals, looking at them and trying to understand how they could be created. I learned how to translate visuals into code, especially recursive code. Some of these figures were computationally expensive, especially the calculations involved to generate Koch's snowflake. I experimented with recursion depth, both for visual purposes and for practical purposes.





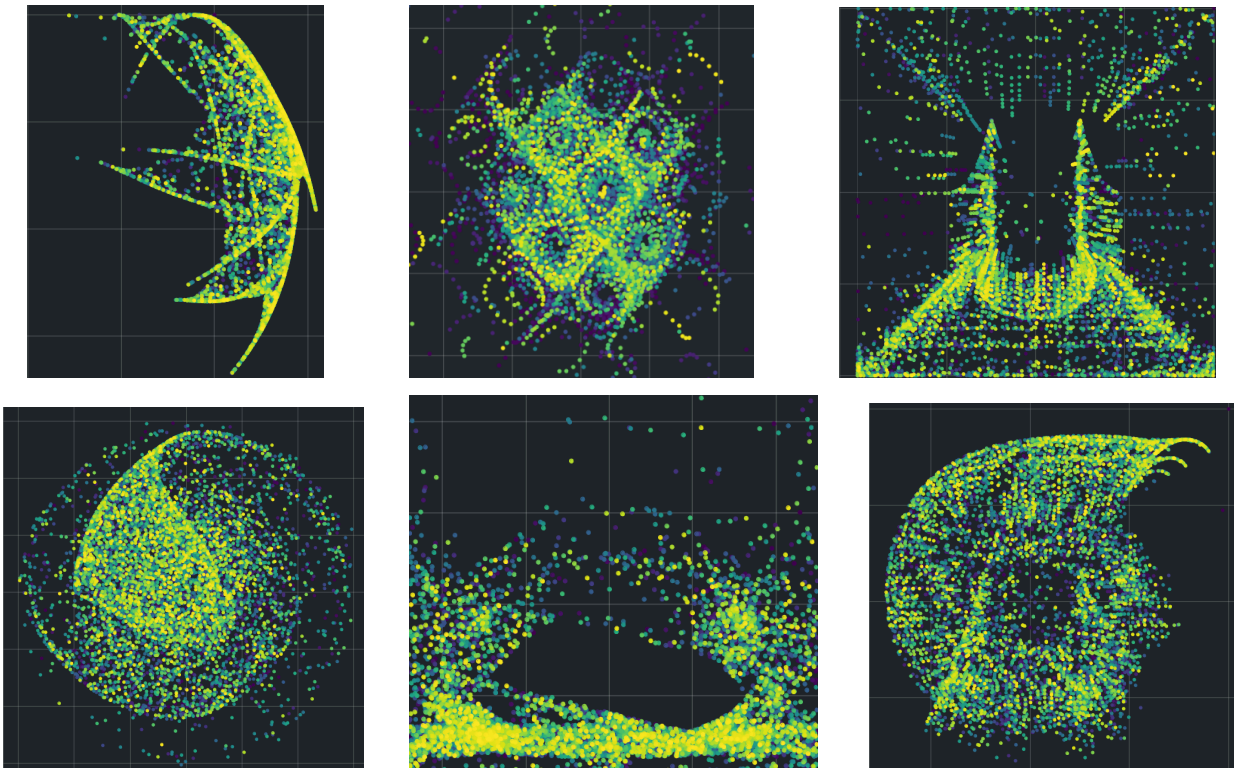
Part 2: Chaos Games

I started learning about more formal ways you could simulate recursive figures, which would also make it easier to generate figures, both computational and theoretically. The main one I did was a chaos game. I learned about these by reading online articles and watching videos on the topic (linked below). I also started creating more interesting color schemes based on xy coordinates, or what function was chosen.



Fractal flames

Chaos games, also known as iterated function sequences, can be used to create fractal flames. "The Fractal Flame Algorithm" by Scott Draves and Erik Reckase was a big source of inspiration for the The general principle is the same, however on top of the normal transforms, a post transform is applied to make things more visually interesting. Tweaking different probabilities of occurring and using different functions can generate interesting fractal flames. For example:



We can look at how the number of iterations affects the appearance of the fractal flame. This is a simplified version of the standard algorithm for fractal flames, which uses histograms and frequency to color the dots. Since dots were colored with increasingly darker colors as the iteration progressed, we can see how dots tend to get farther away the more an iteration

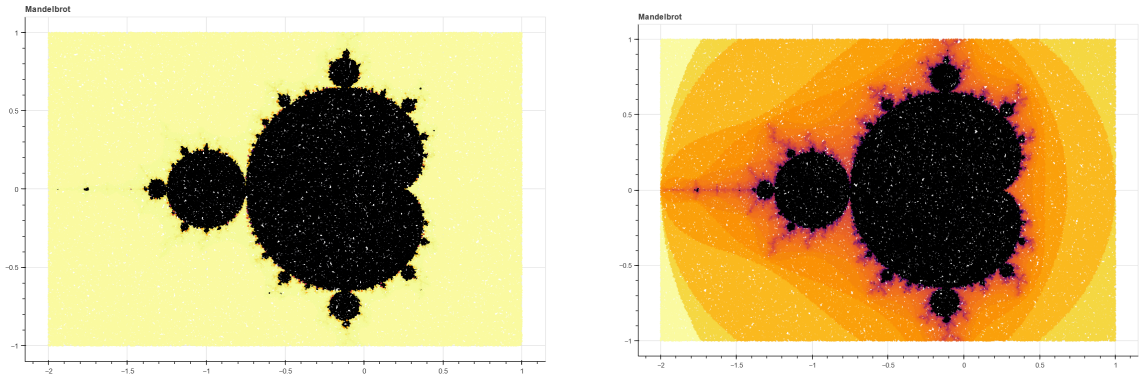
progresses. In the case of these fractal flames, mapping an excessive amount of dots could cause the final result to look too clustered and not refined enough, which explains why the original algorithm mapped dot color intensities logarithmically.

Part 3: Mandelbrot Set and Julia Sets

The main part of the project involves generating the Mandelbrot Set and Julia sets. These were both generated by running Monte Carlo simulations. We want to investigate how recursive depth affects the appearance of the Mandelbrot set.

Coloring

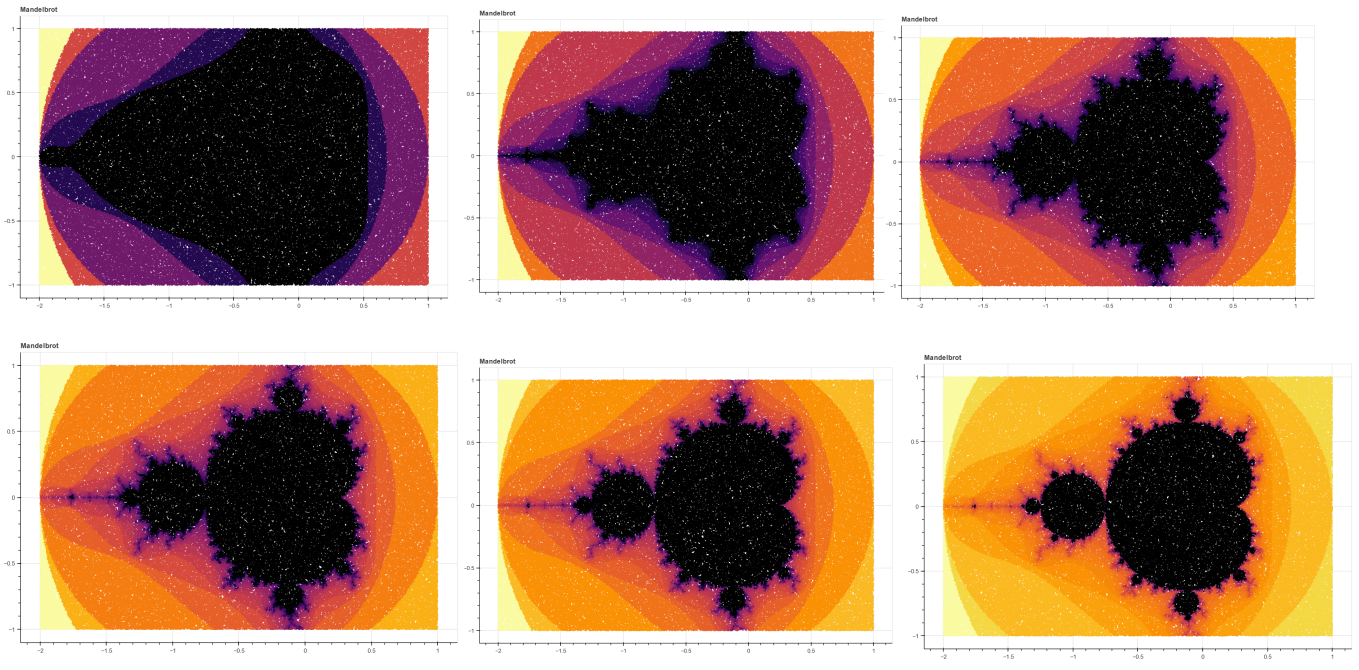
Points were colored based on their final recursion depth. This coloring had to be done logarithmically rather than linearly.



Threshold = 1000. Linear vs Logarithmic Coloring.
We can conclude that points that do not belong to the set tend to diverge fairly early.

Recursive Depth

Further evidence for the early divergence of non Mandelbrot points is seen by changing the recursive depth. You can see how the Mandelbrot's appearance changes when run with 200.00 points and the following recursive depths (threshold).

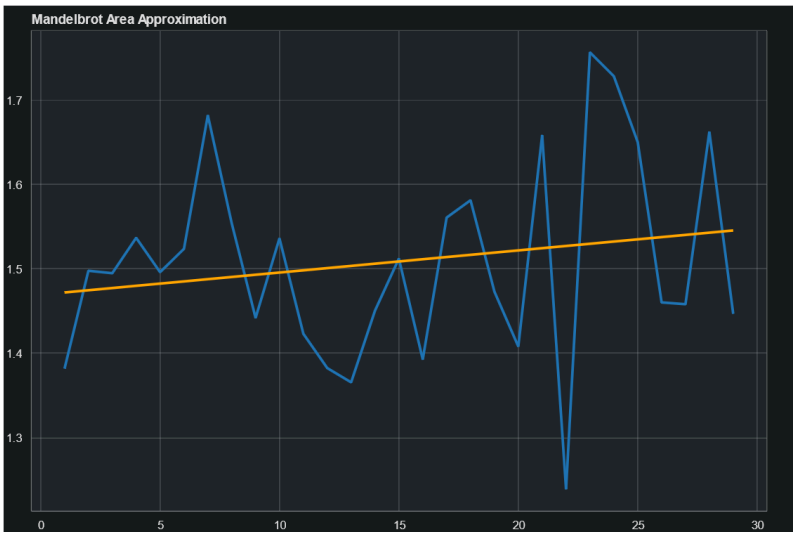


5,10, 25, 50, 100 and 1000 recursions.

The greatest changes occur between 5 and 25 recursions, with little change afterwards even after the points were increased to 5000. Note: these points were mapped with a bound of 2. However, running the simulation with a bound of 1000 and similar threshold values gave results that were visually very similar to the simulations run with a bound of 2.

Area Approximation of a Mandelbrot.

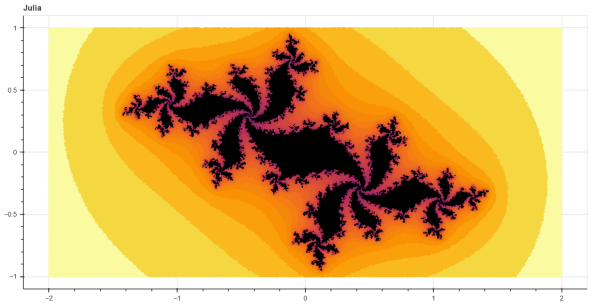
Using our Monte Carlo simulation, we can also estimate the area of a Mandelbrot.

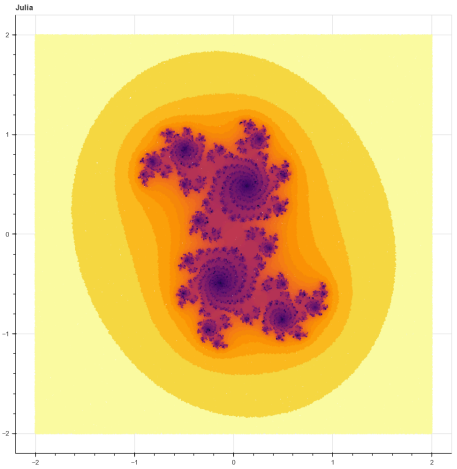


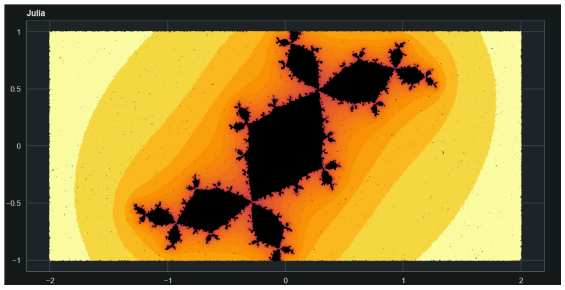
We can see that it is between 1.45 and 1.55, with a gradual increase as a larger area of the plane is considered.

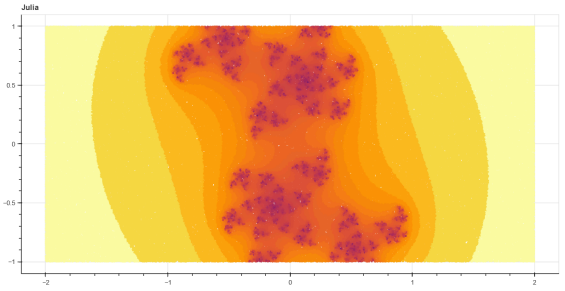
Julia Sets

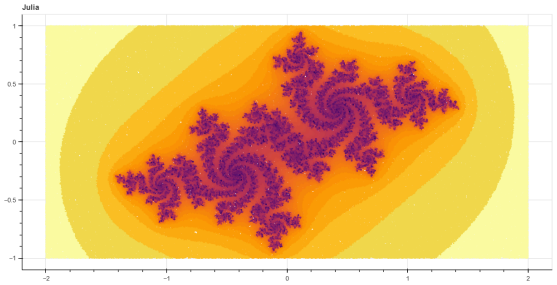
A similar algorithm to the one for the Mandelbrot set can be used to generate Julia sets, which create visually interesting renderings with different values of the constant c. Here are a few examples.

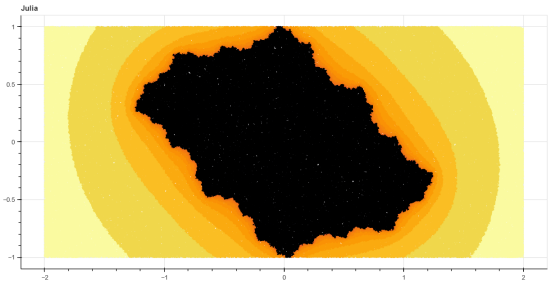












Conclusion

I learned how to use Monte Carlo algorithms and the use of randomization to generate recursive figures, sometimes using the mathematical properties of the figures themselves, and sometimes reasoning upon their characteristics(in the simpler examples). I experimented with recursive depth, bounds, and setting limits to make the processes less computationally intensive. We can see how different colors can be used to represent different aspects such as achieved recursion depth, behaviour over time, function selected, and so on. I learned to use bokeh to manipulate color

References

Videos and papers I used to understand the concepts I was applying, along with skimming wikipedia pages for the topics.

https://flam3.com/flame_draves.pdf

https://www.youtube.com/watch?v=kbKtFN71Lfs
https://www.youtube.com/watch?v=ZSS-Adr5Gcc
https://www.youtube.com/watch?v=OocR3G-UKuI
https://www.youtube.com/watch?v=BjHogfbhPoM
https://www.youtube.com/watch?v=wsIKR_okiU0

https://www.karlsims.com/julia.html