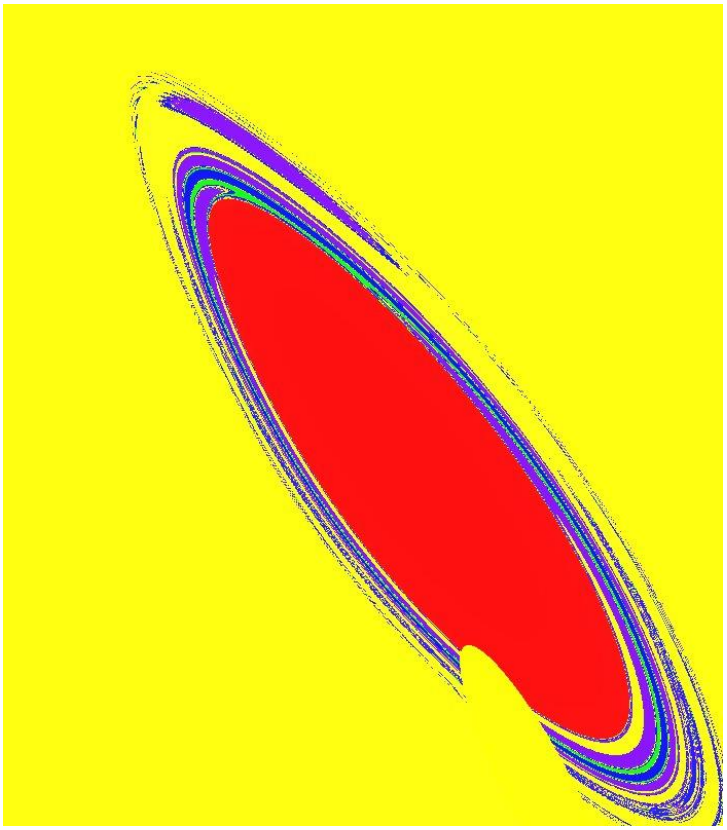
Fractals and Chaos

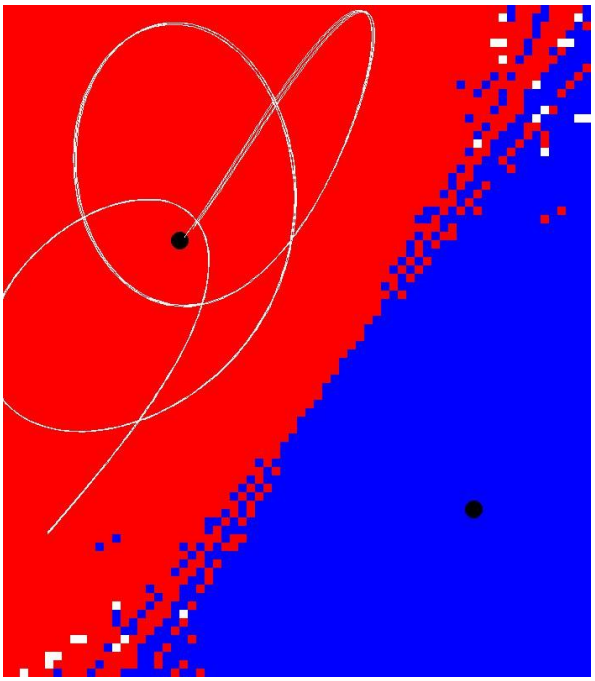
Exploring how chaos can emerge from simple rules, and how the world of Fractals is connected to chaos. A project for the Introduction to Programming course extra work.

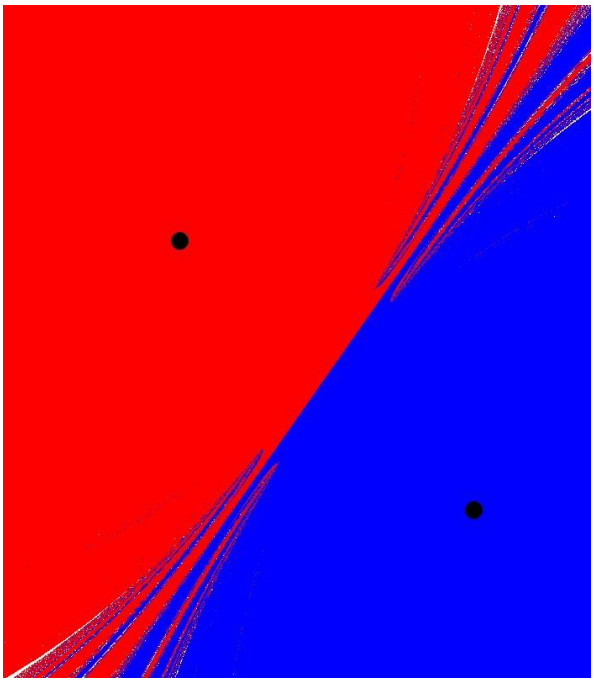


Content

Introduction

In this project the world of fractals will be explored. In particular, we will use the inherit chaos in a simple dynamical system to generate very interesting fractals. Imagine a 2D world where there are 2 planets: planet BLUE and planet RED. Now imagine we, as 3D beings, wanted to paint this 2D world. So, we choose a point in the plane, launch a spaceship from that point, and we watch what planet said point converges to. The spaceship lands on planet RED, so we paint that point RED. Then we repeat this process for every point in the plane and we get this image:





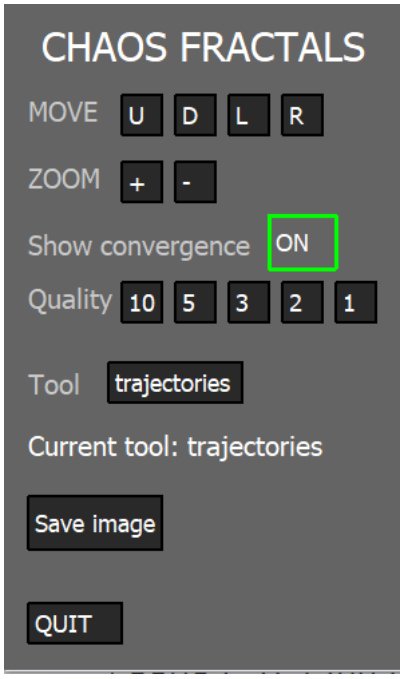
For example, in the image you can see the trajectory of one of the spaceships, which ends up converging to the red planet, hence that point is painted red. Since we are using a computer, with limited CPU and memory, we cannot run this simulation for every point in the plane. So, we divide the plane into squares, the smaller the squares the better the resolution of our image.

In this project we will first build a C++ application that allows us to explore this new world we created. Then, we will use it to observe the fractals that emerge in this world. Finally, we will draw conclusions on why chaos is connected with fractals.

UI Library

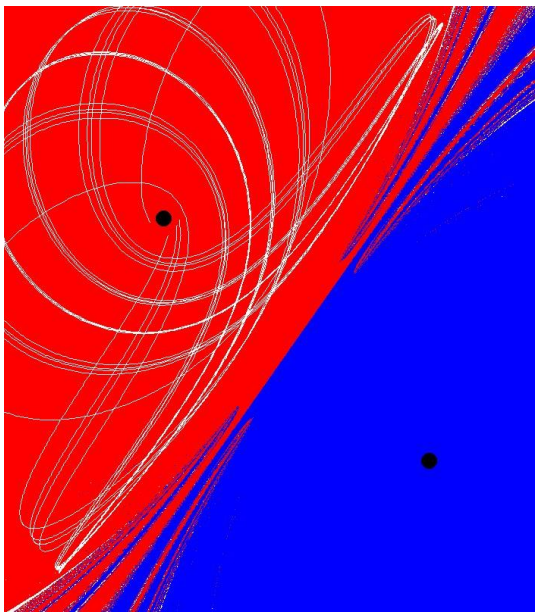
In order to build this application, a UI Library was coded as a side project. The UI Library allows to display buttons and text. The buttons have a hover animation. Everything is fully customizable in the code, though a little tedious at times. Nevertheless, it allows to detect button clicks and returns a string indicating which action should be taken. Ideas to make a Version 2 of this library would be to allow enabling/disabling of buttons, to handle button clicks by saving a callback function in code, and to come up with a easier way to allow for customization of colors and hover effect.

Tools

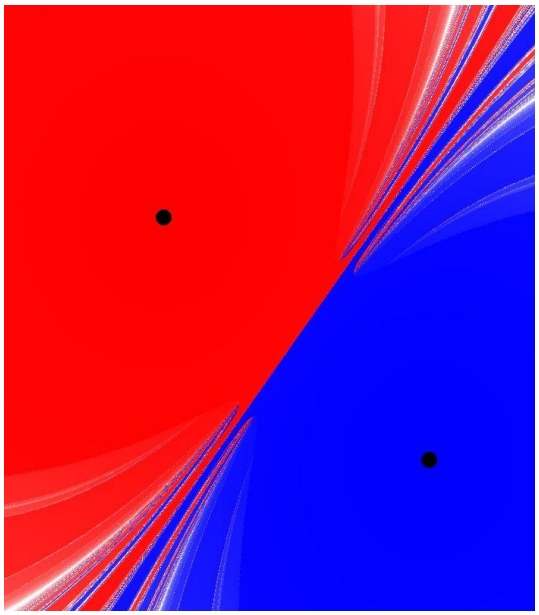


- Moving: To explore the fractals you can move using the *Up*, *Down*, *Left*, *Right* buttons.
- Zooming: To further explore the fractals you can zoom in or out with the *+*, - buttons.
- Quality: You can change the resolution of the fractal (10 being the worst, and 1 the best). This is useful because computing the fractal in high resolution takes a long time, so when exploring the fractal, it is preferable to use a lower resolution until you arrive to a point of interest.
- Trajectories: Whenever you click on the fractal, the trajectories of *spaceships* let go at that point will be shown.
- Save image: When pressed, this button saves an image of the current fractal as a jpg to the working directory.
- Zoom out film. While writing this a new button was added. After zooming in, if pressed, the program will keep zooming out and saving images. This can take hours to compute.

Chaos



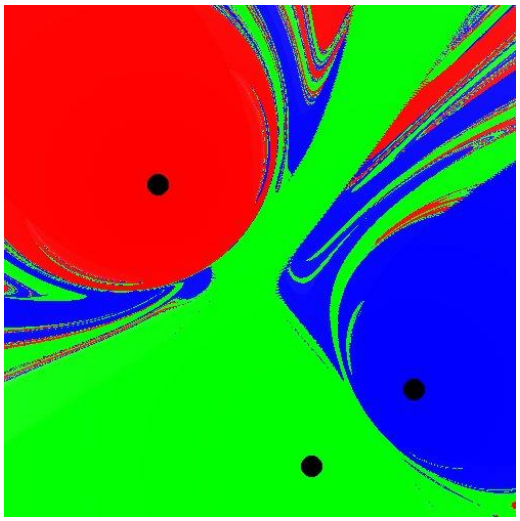
As you can see in this image, the points really close to planet BLUE will be blue, and the points really close to planet RED will be red, but in the boundaries between these two colors we get an interesting chaotic behavior. We can display the trajectories of this spaceships, to show why the system behaves so chaotically. As you can see, the trajectories can be very complex.

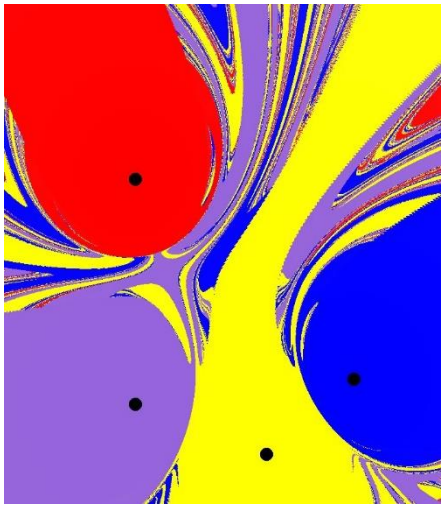


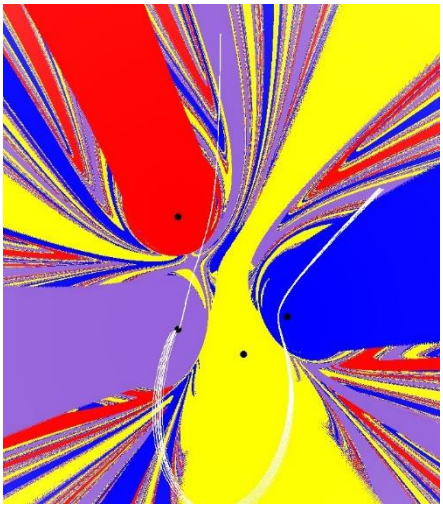
Some spaceships will even get stuck in an orbit, and never converge. Others will take a really long time but finally arrive at one planet. Since again we have limited memory and computing power, we only simulate up to 5000 steps, and we can visualize the convergence time by interpolating between the planet of convergence and the color white. That is, we will paint squares red if they converge in 0 steps, white if they never converge, and linear interpolate between those two.

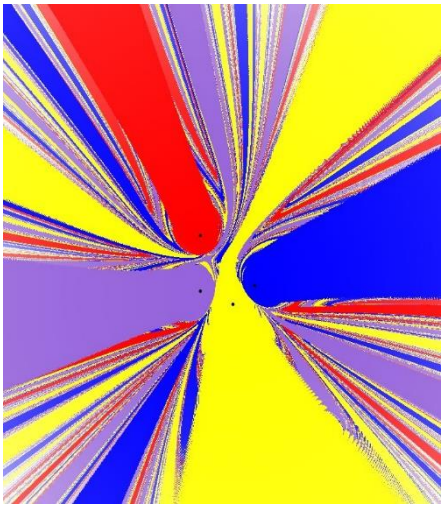
More planets

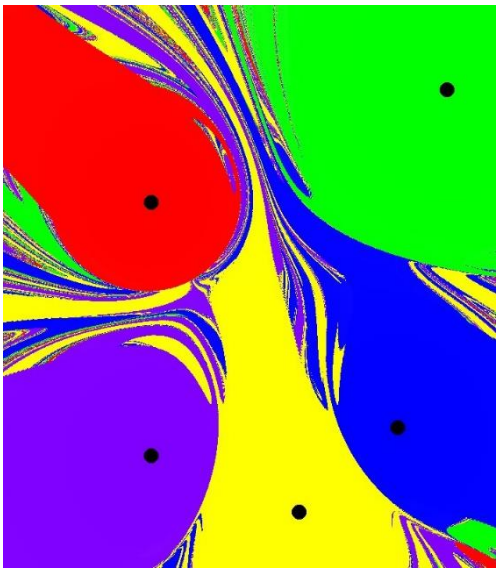
The fractal we get with 2 planets is interesting, but as we bring more complexity to the system, the images become more chaotic. Here are some images with 3, 4 and 5 planets. Consider that the complexity of the computations is of order n^2 where n is the number of planets.







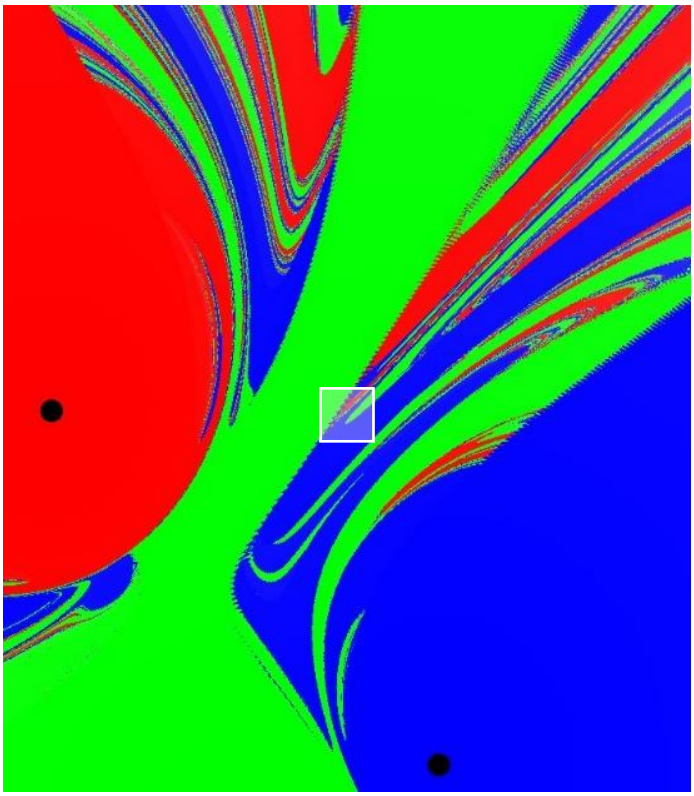


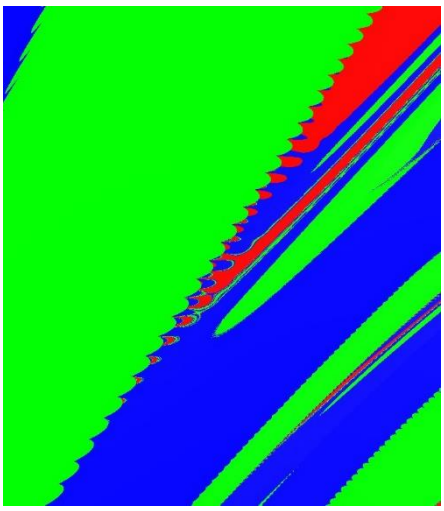


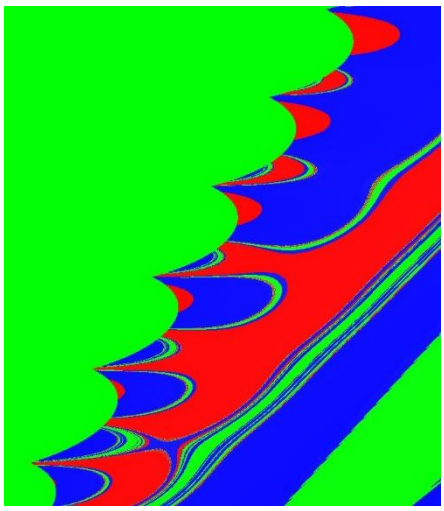
Zoom ins

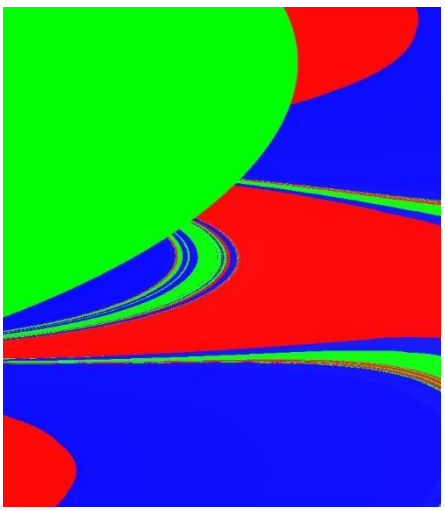
We can explore the fractals using the moving and zoom in tools. By doing this we uncover the fractal nature of the system. Meaning that, the boundaries between two colors are fractals. We can keep zooming in, but we will always find more colors between the boundaries. Up to a point where we can no longer zoom in due to computer precision.

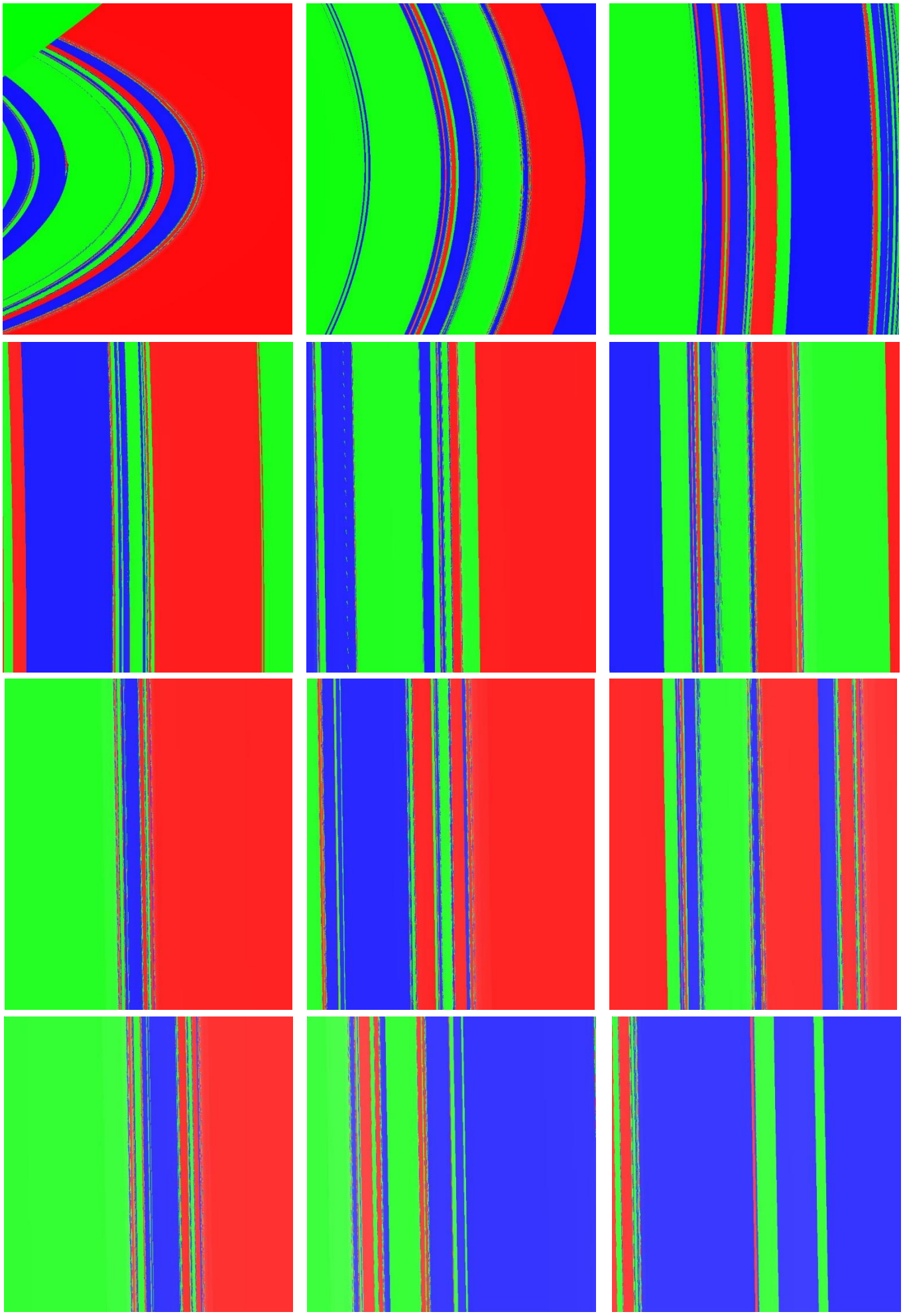
Here you can see a sequence of images zooming into one part of the fractal, indicated on this first image by the white square. An ambitious idea is to create a film generator, that creates a lot of frames by zooming into a zone of the fractal little by little.





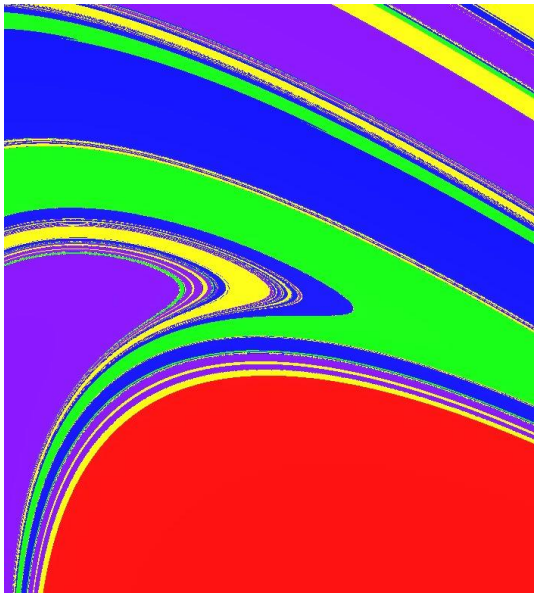


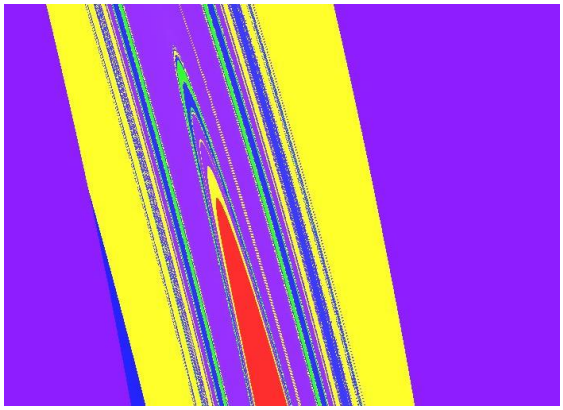


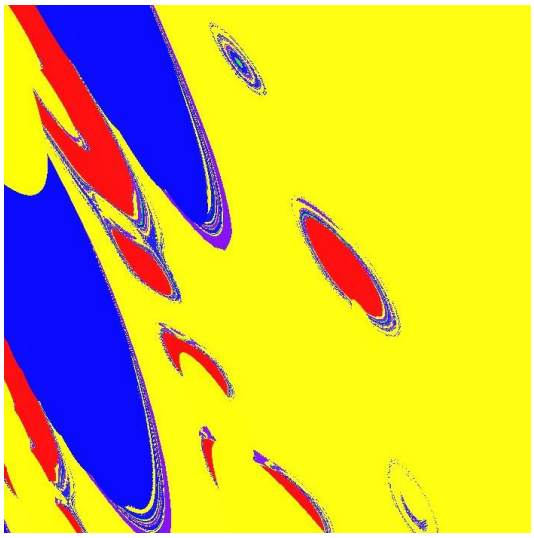


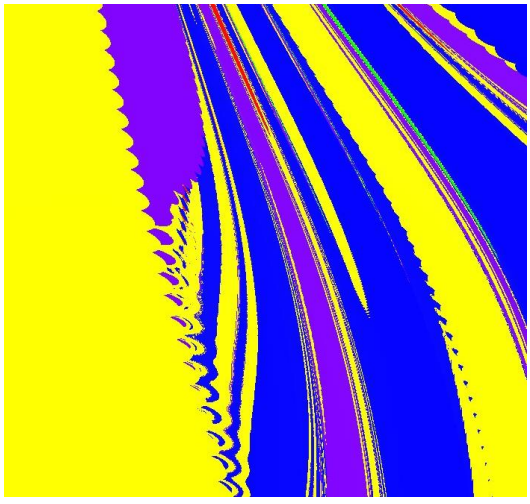
From the above sequence, note that the boundary between the colors behaves as a fractal, meaning that, even if we keep zooming in, we will not find a smooth defined border between the colors.

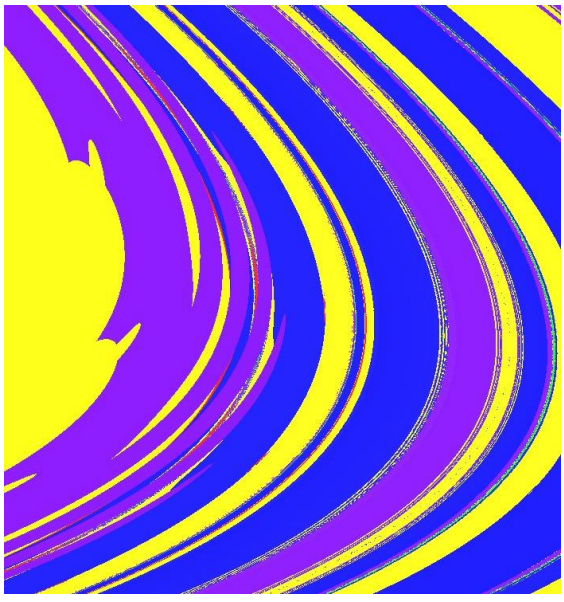
Here you can see some other images of different parts of the fractal.







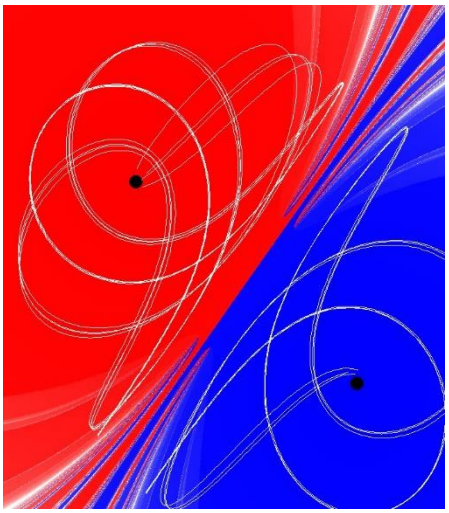




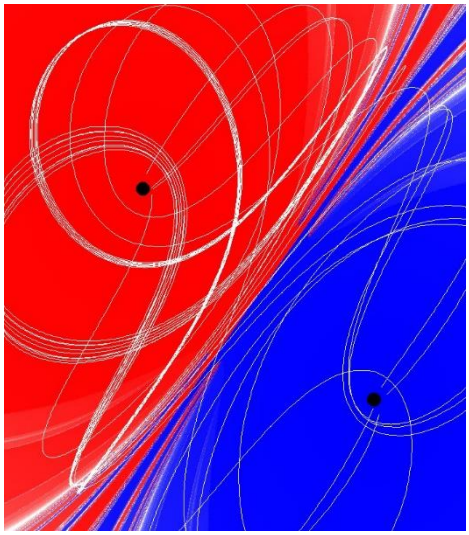
I don't want to make this section too long, so I will leave one more zoom in sequence at the **¡Error! No se encuentra el origen de la referencia.** section, at the end of the document.

When zooming in you might notice that areas only change in the boundaries, if you have a green area, it will stay green, the interesting and fractal part comes when zooming into the boundary. But why? Does it make sense that the boundaries are so chaotic? Yes, it does.

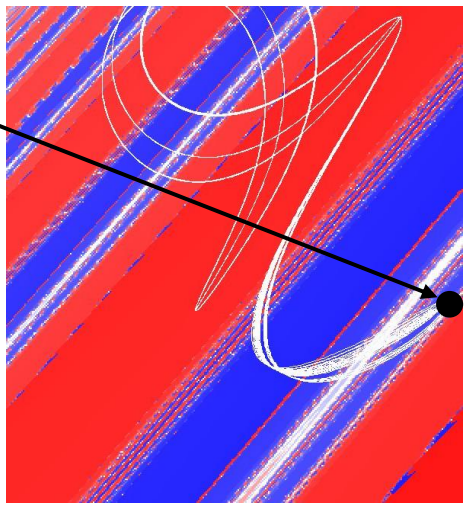
Chaos and Fractals



Here lies the connection between chaos and fractals. The fact that the boundaries are fractals is not random. Take the scenario with two planets for example. The areas close to each planet are well defined, but as you get further the trajectories can become very complex, to the point where the spaceship can converge to either planet. As a matter of fact, the partial differential equations that rule this system haven't been solved yet (perhaps they are unsolvable). These zones are so chaotic that at some points it feels as if the spaceship could converge anywhere, as you can see in the image below.



We might think that the behavior in those zones is random, so we zoom in into these seemingly chaotic zones, and as we do so, we will see that the pattern is not random. This is because, even though the system is chaotic, it is still continuous (solutions that come from infinitely close points converge to the same planet). Except in the boundaries between two colors.



As illustrated in this image, in a zone that appeared random, you can see that trajectories clearly converge to one of the planets. And when I say "clearly", I mean that the trajectory of that spaceship and the ones close to it, all converge to the same planet (continuity). You can also see that the trajectories spread the further in time they go. *Note that the trajectories are plotted without zooming in, or else we wouldn't be able to see them.*

In the boundaries, the trajectories are the most vulnerable to this chaos, and any little change can make them converge to a completely different planet. That is the reason that even if we keep zooming in, this boundaries will never be smooth, and will keep being complex. The boundaries are fractals, and that is the connection between chaos and fractals that this whole project tries to illustrate.

Conclusion

It is very interesting how from relatively simple rules (see below the code to simulate gravity force from N bodies) can emerge such complex system. This is a common thing in mathematics, and for me, it emulates how, in nature, such complex life can emerge. There are so many things we still can't understand, the human brain, black holes, subatomic particles… And all of this things emerge from some simple interaction between the particles that compose the universe.

```
// Calculates the acceleration given a position
Vector2 simulator::get_acc(Vector2 pos) {

    Vector2 acc = createVector(0., 0.); // Start with 0s

    for (int i=0; i<n_planets; i++) {
        // Calculate acc for planet
        Planet p = planets[i];
        Vector2 dir = p.pos-pos;
        double dist = norm(dir);
        dir = dir / dist;
        acc = acc + dir * (G * p.mass / (dist*dist));
    }

    return acc;
}
```

What now?

There are many ways to make this application better, that weren't explored for this project due to lack of time. This are some of the best ideas:

- Paint the fractals on a separate thread to allow the UI and rest of the application to work while the fractals are being calculated. Even, an automatic system that renders first in low resolution and then refines the image.
- A planet placing tool that allows to run other simulations without editing the code.
- Rendering full films. After exploring the fractal by zooming in, the player could choose a point, and then the application would render frame by frame a zooming in film.
- Make a better UI Library
- Another idea for optimization is utilizing the information that we know about the boundaries being chaotic and the center of the areas stable. When zooming in computations could be avoided and time could be saved by utilizing this information smartly.

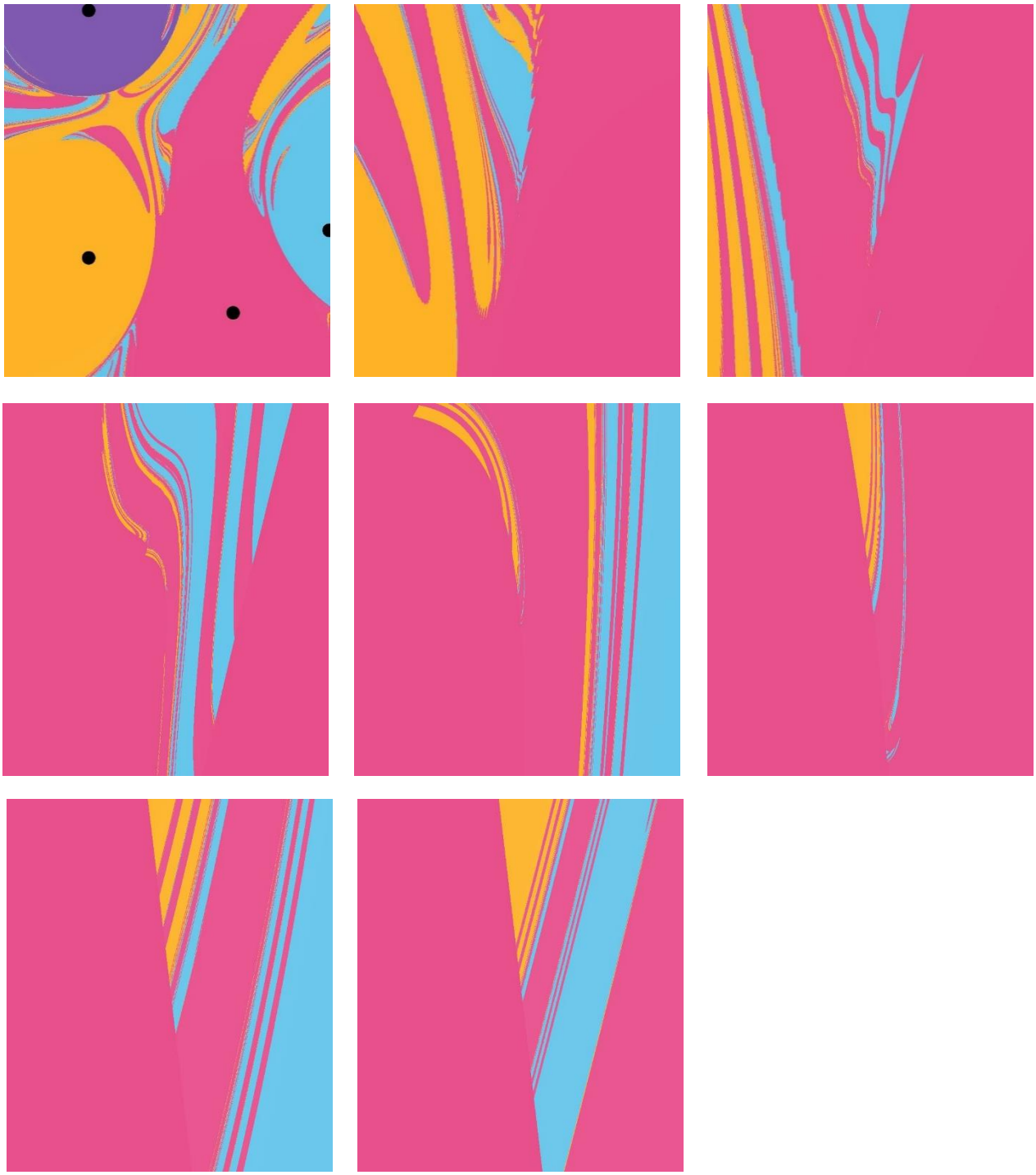
Here are other crazy ideas:

- Letting the planets move throughout the simulation, either following certain paths. The paths could be anything as simple as a line, or more complex as a wave, a circular orbit, or a bouncy planet…
- For each pixel launch 20 spaceships in slightly different position and average the color of the pixel.
- Infinite resolution. Right now, the amount of zoom in we can do is limited by the precision of our machine. In the application *double* resolution is used. There are ways to define our own numbers that can have the resolution we wish. Then we could adapt the resolution to the current zoom.

- GPU Optimization: The code run to color each pixel is independent from the rest of the pixels. Right now as the code is written, the CPU runs through every pixel in the screen and simulates the spaceship. This is highly inefficient and has a complexity of n^2. It wouldn't be much better to turn this into shader code and have the GPU run it.

Zoom in sequences

Here you have another zoom in sequence, which I thought better to put at the end, in order to not make the document too long.



And one last one with 5 planets: