Assignment repository: https://github.com/roelno/graphics_fractals

Yue Zhang

## Project Summary

The purpose of this assignment is to learn to build and use Image data structure and use it to create some fractal noise functions.

- Created Image structures and associated functions.
- Implemented functions to create images of a Julia set and Mandelbrot set, utilizing Image structures and functions.
- Create functions to create other fractal noise: I created a burning ship.

## Task 1: Image Structure

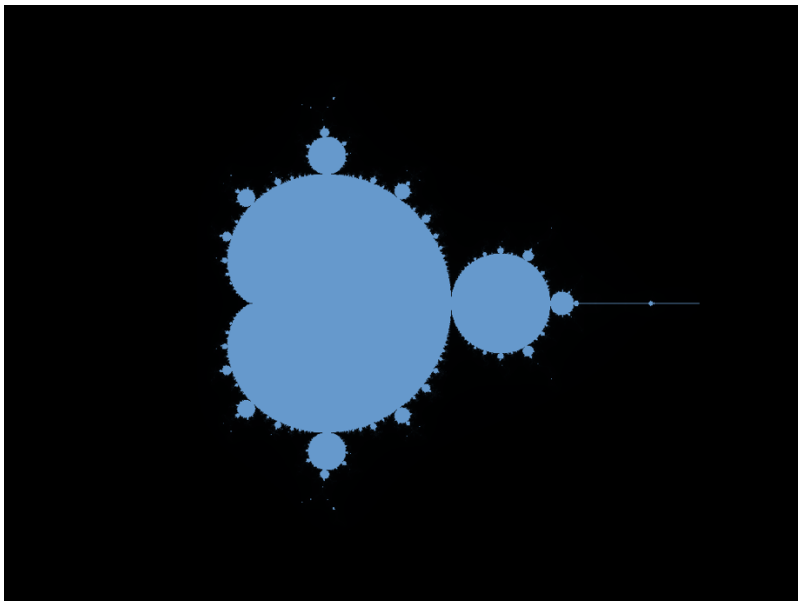
This task created Image structures and associated functions.

To create the *black.ppm* and *skyblue.ppm* images, we first initialized an image structure and allocated memory for a 400x600 image. For *black.ppm*, we used the *image_reset* function to set all pixels to black (RGB values of 0) with full alpha and depth, then saved the image using *image_write*.

Next, for *skyblue.ppm*, we set each pixel in the image to a sky blue color (RGB values of 0.4, 0.6, 0.8) using the *image_setf* function. This image was also saved using *image_write*. Finally, we deallocated the image memory with *image_dealloc* and freed the image structure using *image_free*.

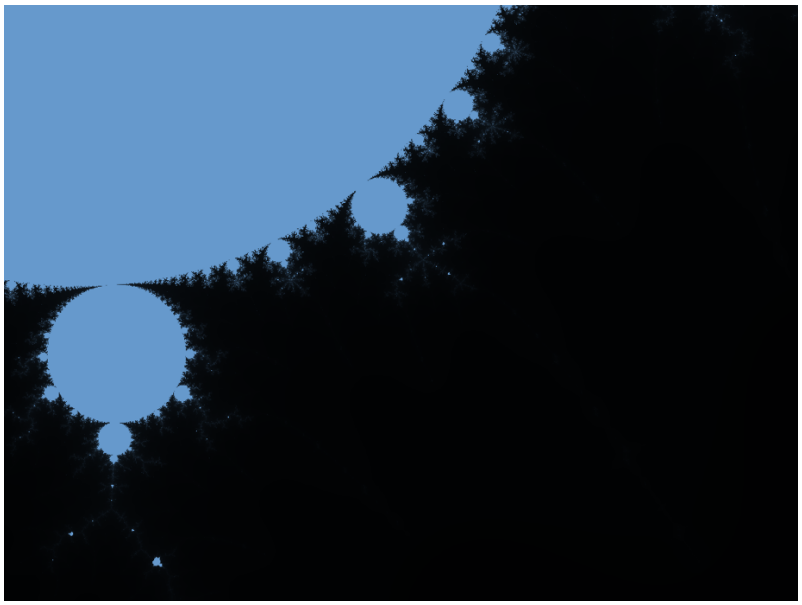
## Task 2: Mask Generating

This task implemented functions to create images of a Julia set and Mandelbrot set, utilizing Image structures and functions.

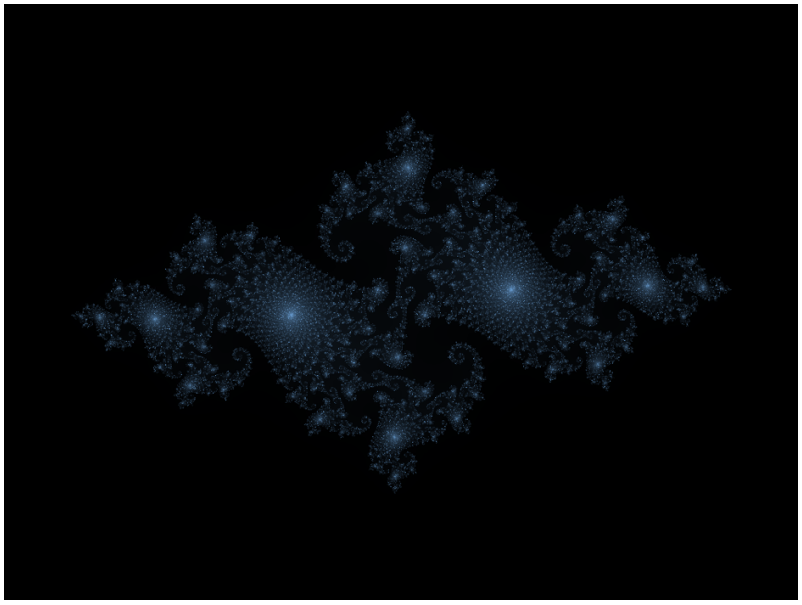


To create this image, we first implemented the *mandelbrot* function, which maps each pixel to a point in the complex plane. The function iterates up to 1000 times per pixel to determine if the point belongs to the Mandelbrot set using the escape condition `cabs(z) > 2.0`. Each pixel's color is set based on the iteration count before escape, creating a gradient effect.

Then we created this image by initializing a 750x1000 image and generating a full Mandelbrot set for the region (-1.5, -1.5, 4) using the *mandelbrot* function. This function maps each pixel to a point in the complex plane, iterates to check for escape conditions, and assigns colors based on iteration counts.



We created this *Mandelbrot.ppm* image by initializing a 750x1000 image and generating a Mandelbrot set for the region (1.755, -0.02, 0.02) using the *mandelbrot* function. This region is chosen to zoom in on a specific area of the Mandelbrot set, revealing more detailed and intricate patterns that are not visible in the wider view used in full Mandelbrot.

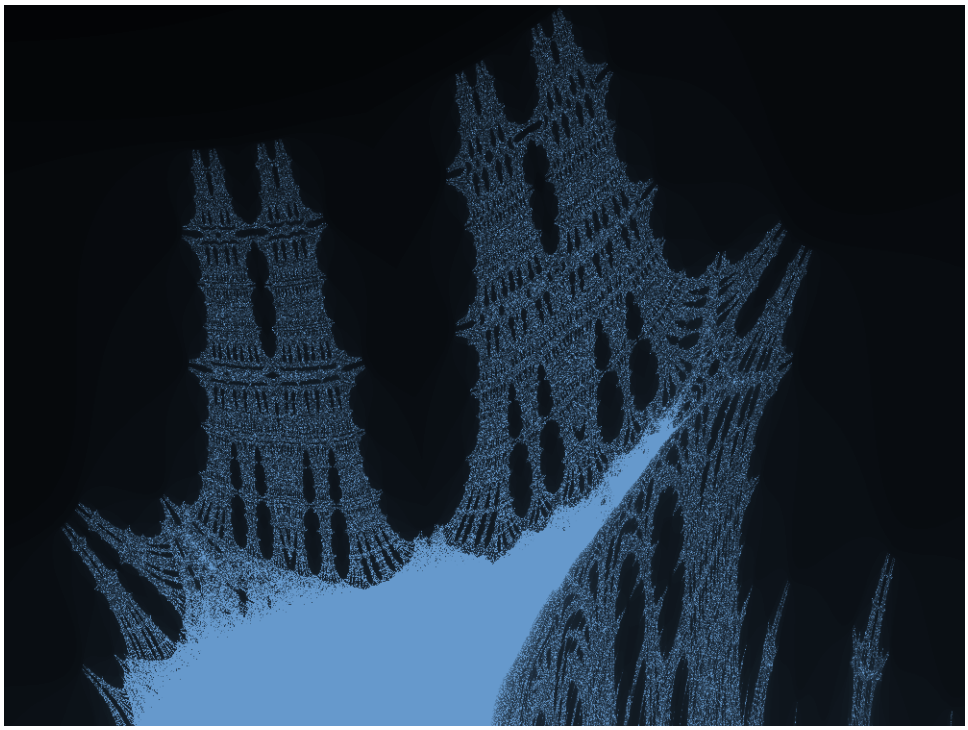


To create this image, first we implement the Julia function: It iterate over each pixel in the image, mapping it to a point (x, y) in the complex plane within the specified region. For each point, it initializes a complex number “z” and iterates up to *max_iter* times, updating “z” using the formula “z = z * z + c”, where “c” is a constant complex number. If the magnitude of “z” exceeds 2, the loop breaks, and the iteration count determines the pixel color.

Then we created this julia.ppm by initializing a 750x1000 image and generating a Julia set for the region (-1.8, -1.35, 3.6) using the *julia* function. This function maps each pixel to a point in the complex plane, iterates to check for escape conditions with a constant complex number $c$ = $0.7454054 + 0.1130063i$, and assigns colors based on iteration counts.

## Task 3: Fractal Noise – Ghost Ship

Create functions to create other fractal noise: I created a ghost ship.

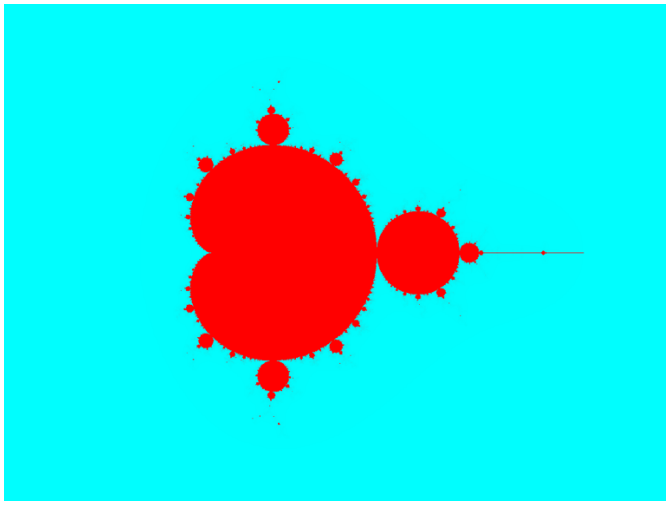


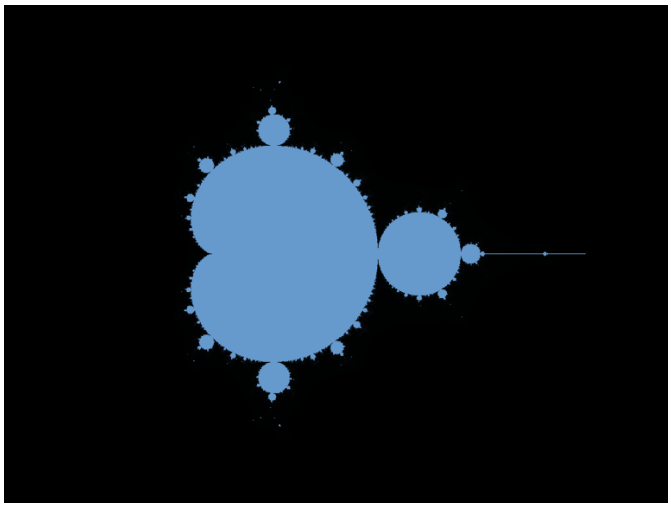
We created this image by initializing a 750x1000 image and generating a Burning Ship fractal for the region (-1.8, -0.08, 0.1) using the *burning_ship* function. This function maps each pixel to a point in the complex plane, iterates to check for escape conditions with the formula involving absolute values of the real and imaginary parts, and assigns colors based on iteration counts.

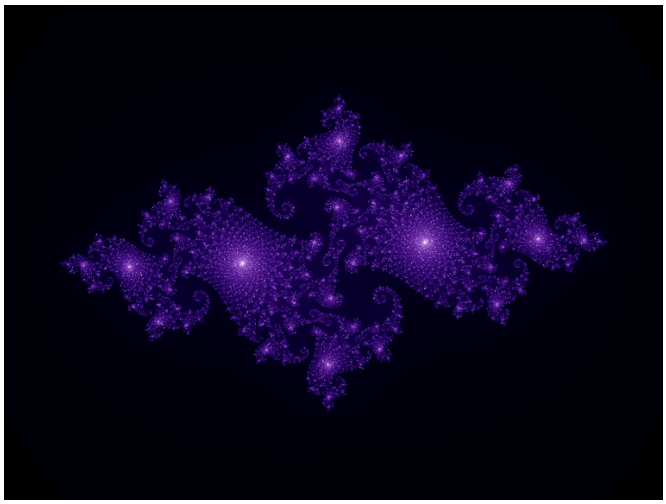
The formula of building this ship is from: https://robotmoon.com/burning-ship-fractal/

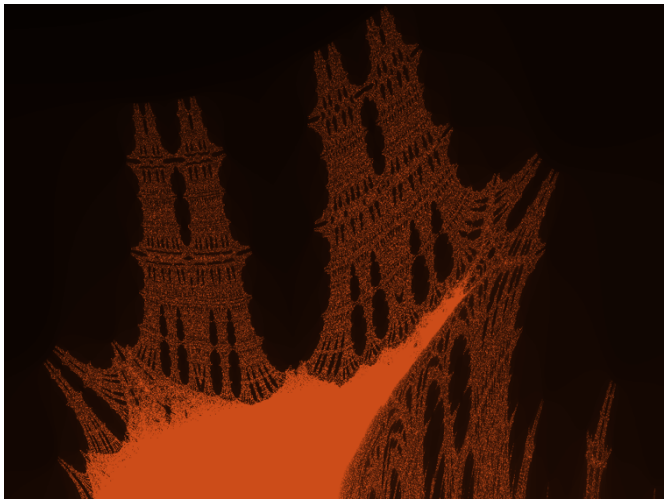
## Extension: A method of Coloring the Fractals

This task is an extension task to find a method to coloring the fractals.









I use three different coloring methods.

- For Mandelbrot, the method sets the red channel to the normalized iteration count (color), and the green and blue channels to the inverted value (1.f - color). This results in a distinct color transition where the red channel increases while the green and blue channels decrease.
- For Julia, the method also sets the red channel to the normalized iteration count (color), the green channel to the square of this value (color * color), and the blue channel to the square root of this value (sqrt(color)). This creates a non-linear color gradient, providing more color variation and detail in the fractal image.
- For Ghost ship, it sets the red channel to 0.8f * color, the green channel to 0.3f * color, and the blue channel to 0.1f * color, making the red more standing out. This result make the ship looks like burning!

## Reflection

In completing this homework, I gained a deeper understanding of how to manipulate and generate fractal images programmatically. By working with the Mandelbrot, Julia, and Burning Ship fractals, I learned how to map image pixels to points in the complex plane and iteratively apply mathematical formulas to create detailed visual patterns. This exercise reinforced my knowledge of complex numbers and escape-time algorithms, which are fundamental to fractal generation.

## Acknowledgements

Complex number: https://en.wikipedia.org/wiki/Complex_number

Burning Ship: https://robotmoon.com/burning-ship-fractal/