FractalStudio

Patrick Huang

May 2025

1 Introduction

In this project, we develop software to render the Mandelbrot set and related fractals. We have three goals: Creativity, performance, and user experience.

2 The fractals

2.1 Mandelbrot

The Mandelbrot set (Figure 1) is a set of points in $\mathbb{C}$ satisfying the following rule:

A point c is in the Mandelbrot set if the values $z_{n+1} = z_n^2 + c$ remains bounded, with $z_0 = 0$.

Coloring all such points white in an image, we get the recognizable fractal structure.

It can be shown that if any $|z_n| > 2$, then the z values will eventually diverge, meaning that the c value used is not part of the set. This is a useful condition in an iterative implementation.

2.2 Buddhabrot

The Buddhabrot (Figure 2) uses the same equation as the Mandelbrot, but plots the distribution of z values over all points that are *not* in the Mandelbrot set.

That is, for all values c not in the Mandelbrot set (i.e. corresponding z values eventually diverge), keep track of those z values. Then, the coordinates that get landed on more often (have more z values, across all c) have a higher value (higher brightness).

To render this, we sample random c values, check if it is in the Mandelbrot set, and accumulate z values if it is not in the set. Because z values corresponding to $|c| > 2$ diverge very quickly, it is sufficient to sample within $|c| < 2$, to capture most of the complexity in the lower magnitude values.

In practice, we don't check whether a sampled c value is *truly* in the Mandelbrot set; instead, we iterate the equation up to a predetermined number of iterations. If the z values diverge within this window, we use the c value as part of the render.

Setting a different number of iterations creates Buddhabrot renders that look different. Figure 2 was rendered at 1000 iterations.

Note that iterations and samples are distinct quantities. Iterations is the number of times the Mandelbrot equation is iterated for any one sample. There are many (e.g. millions) of samples to make the whole render.

Note that while the Mandelbrot and Buddhabrot have similar shapes, they are plots over different spaces. The Mandelbrot is a plot of whether c values cause the iterated equation to diverge. The Buddhabrot is a plot of the distribution of intermediate z values in the equation.

2.3 Nebulabrot

The Nebulabrot (Figure 3) is a 3 Buddhabrots corresponding to the Red, Green, and Blue channels, each with a different iterations parameter.

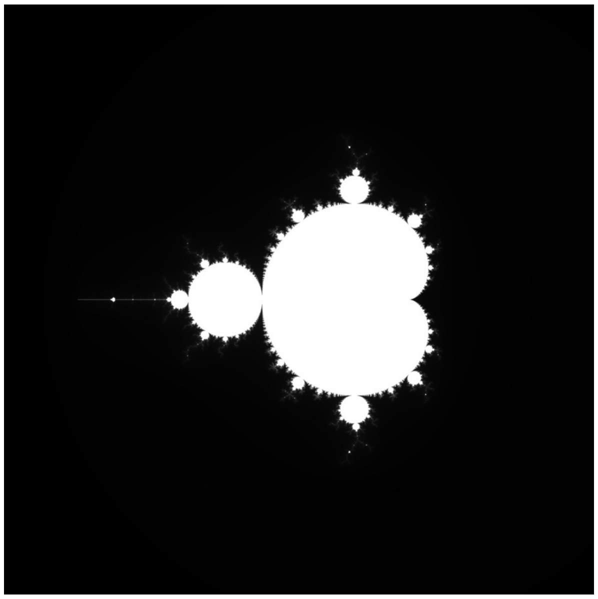


Figure 1: The Mandelbrot set.

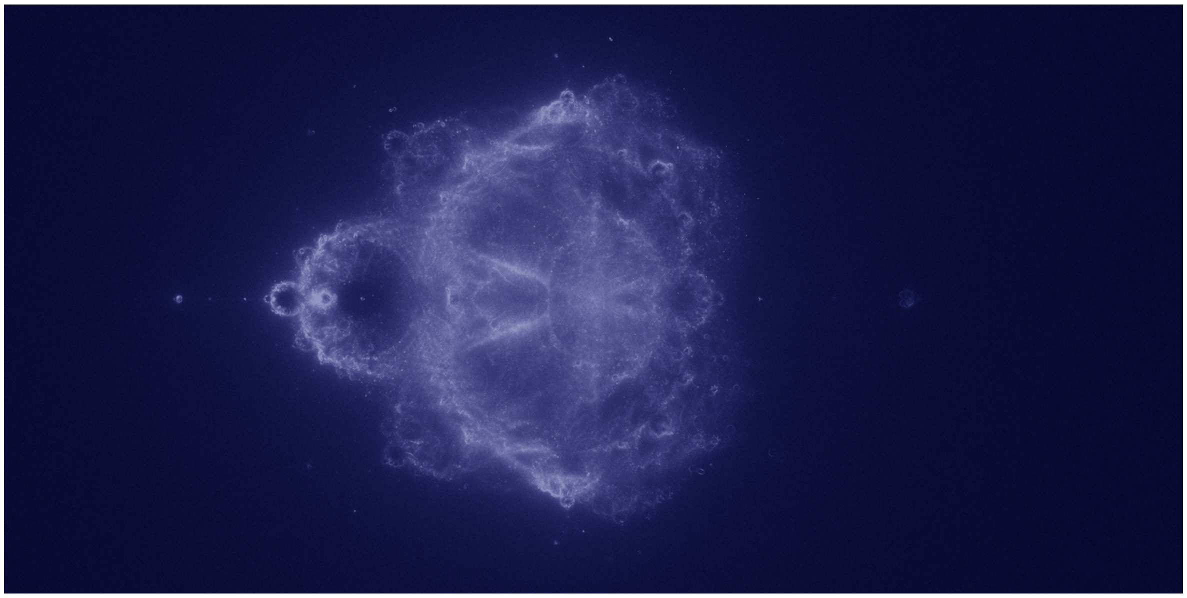


Figure 2: Buddhabrot render.

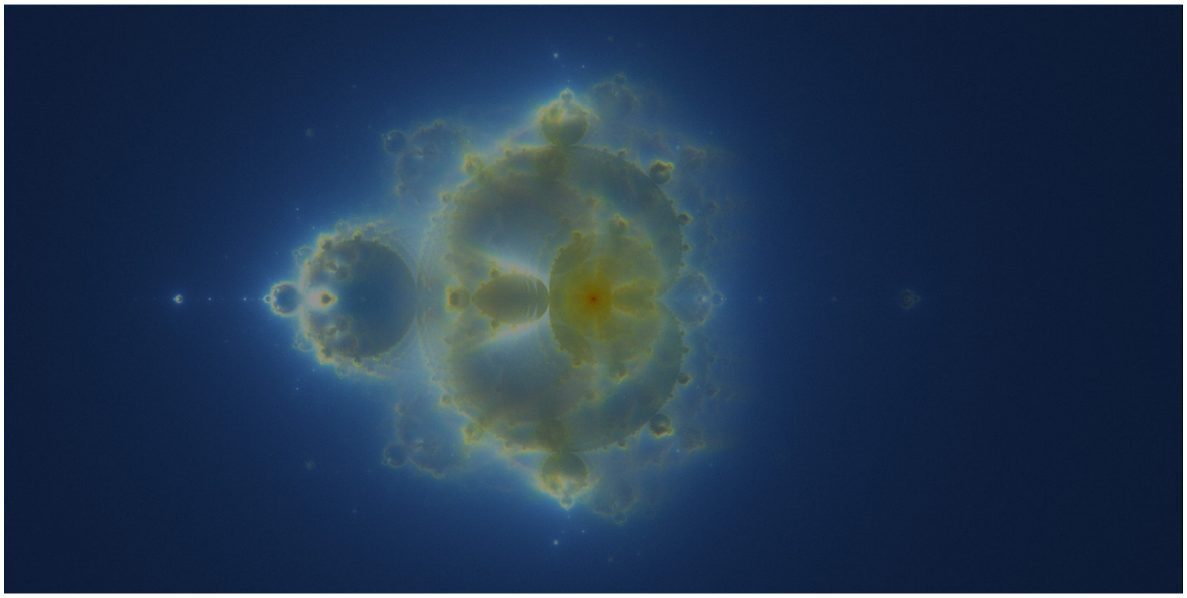


Figure 3: Nebulabrot render.

3 Rendering techniques

3.1 Iterations to escape

For the Mandelbrot set, we can color points not in the set based on the number of iterations before the z values diverge. See Figure 4.

3.2 Log

On sampled images (Buddhabrot and Nebulabrot), we apply a logarithm to the value of each pixel, to compress the orders of magnitude and output a more visually even image.

4 GUI viewer

5 Performance techniques

Coding techniques to improve performance.

Native Python code is extremely slow, so we use various methods to avoid looping in Python.

5.1 C++ and CUDA kernels

We write C++ and CUDA code to handle the rendering algorithm.

The Python program communicates by opening a process, and using stdin and stdout. Render parameters are passed via stdin, and the kernel writes the render result to stdout (usually a raw dump of image data).

5.2 PyTorch vector code

Using PyTorch is another way to run programs on CUDA, using only Python code. PyTorch tensors can be moved between the CPU and GPU very easily, and PyTorch provides an API for many common tensor operations.

To compute Mandelbrot or Buddhabrot, we create batched tensors, move them to the GPU, then apply the Mandelbrot equation. This calculates a batch of samples, with most of the heavy computation on the GPU.

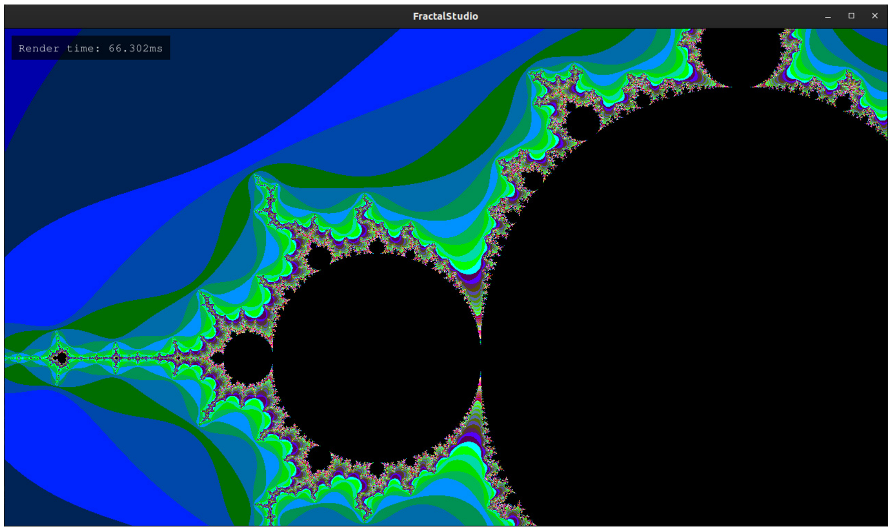


Figure 4: Mandelbrot colored based on iterations

This approach is significantly slower than native CUDA code, but much faster than native Python code. It is a few times faster than the same approach using CPU.

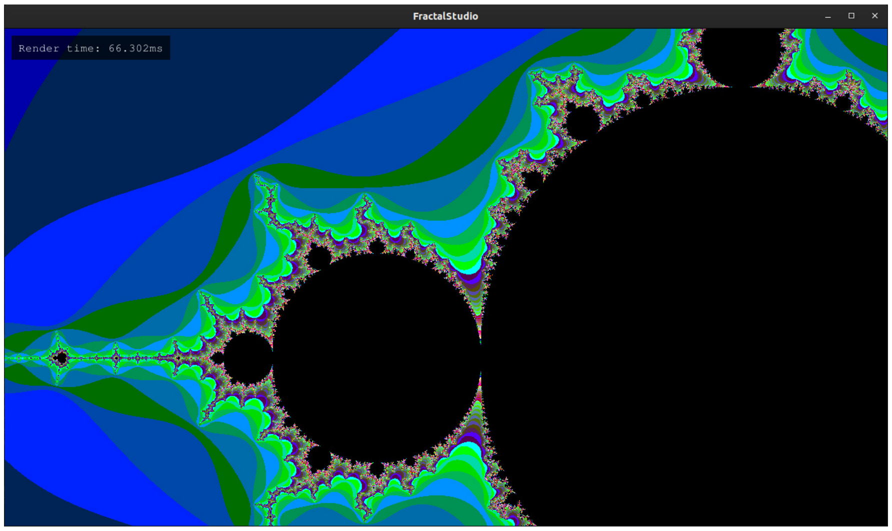


Figure 5: Mandelbrot set with color based on iterations.

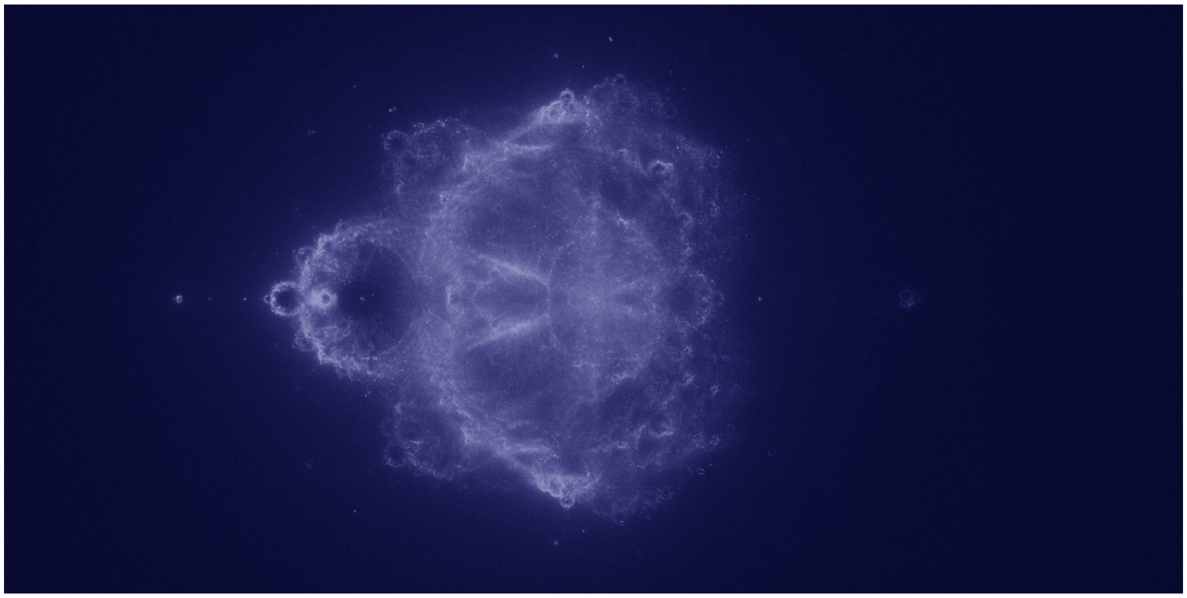


Figure 6: Buddhabrot render.

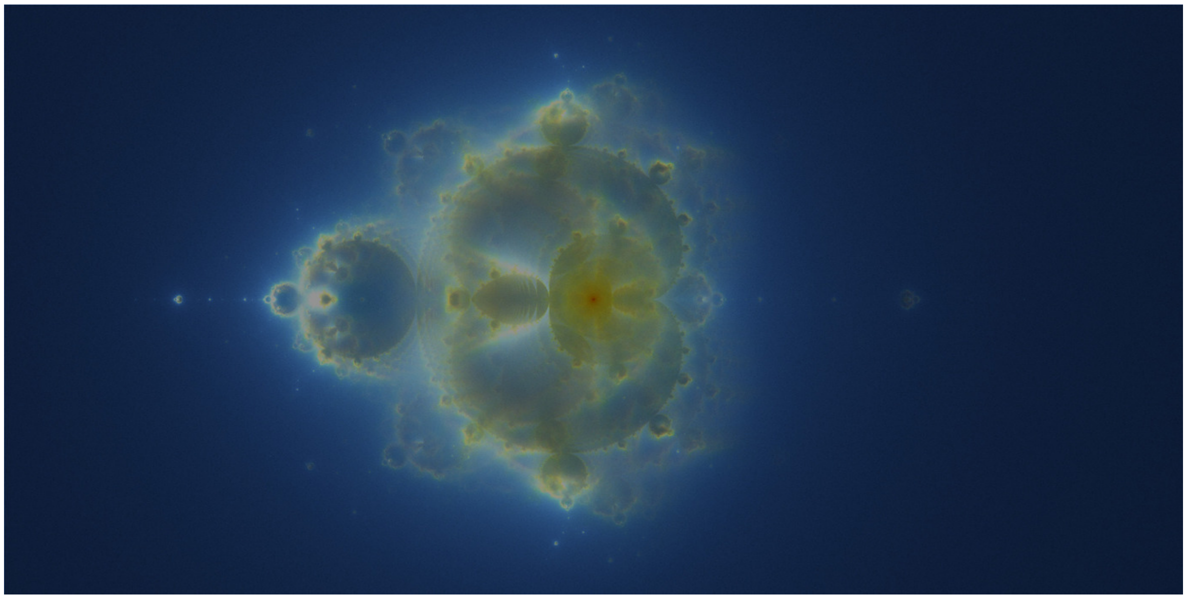
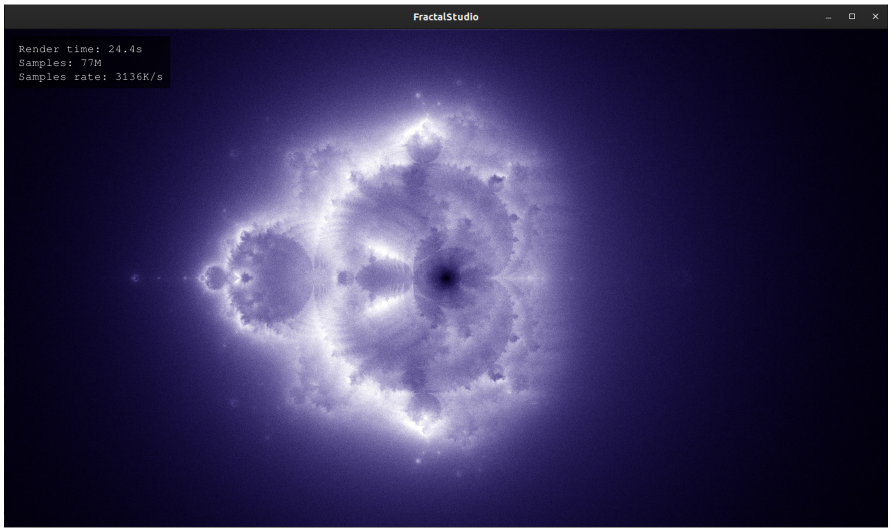


Figure 7: Nebulabrot render.



```
Render time: 24.4s
Samples: 77M
Samples rate: 3136K/s
```

Figure 8: GUI viewer rendering Buddhabrot.