Fractal Computation and Visualization*

an INFOAFP status report

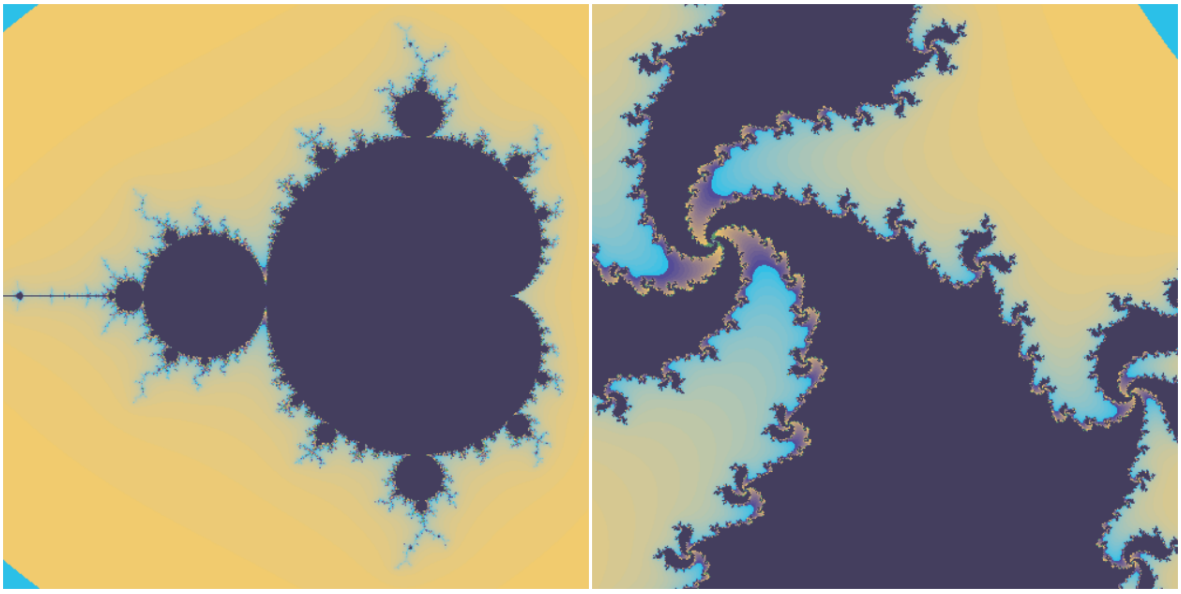
Wessel Custers 5993334
w.r.custers@students.uu.nl
Utrecht University

Xinliang Lu 0822760
x.lu3@students.uu.nl
Utrecht University

Jan Willem de Ruig 6369502
j.w.deruig@uu.nl
Utrecht University

April 19, 2023

1 Overall Progress



(a) Mandelbrot (b) Zoomed in Julia

Figure 1: Example of generated fractals.

1.1 The Fractal Model

We have managed to implement two types of fractals: the Mandelbrot and the single-Julia set. All information relevant for the generation of a fractal is stored in *GeneratorData*. Currently, only the programmer can decide on the kind of fractal generated by choosing one of two functions: one that varies the starting point $\mathbf{z}$ in the fractal polynomial $(z^2 + c)$ and one that varies the (complex) offset parameter c. It would be nice to ask the user which kind of fractal they want to generate. Our current idea is to let the user decide on what to vary and what polynomial should be used (on which the

*Please contact us for access of project repository:https://github.com/largeword/UU-INFOAFP2023-Fractals

fractal type and shape depends). Once the input is received, for example through a command-line conversation, the input is passed to a function that builds a fractal polynomial and stores it with the rest of the relevant input inside *GeneratorData*. With all the information contained within, the right kind of fractal can be computed and displayed.

1.2 Fractal Sequence Generator

In this part, we have three different main functions, namely *getIterations*, *getSequences*, and *getEscapeSteps*. *getIterations* provides the infinite iteration sequence of a point given a fractal function, and using *GeneratorData* to let the user program their fractal function. *getSequences* maps this function over the grid points. *getEscapeSteps* applies a filter to the output of *getSequences* given a maximum amount of escape steps, and returns a grid array which contains the escape step of each pixel. If it does not escape, it is assigned with the maximum escape step value.

1.3 Image Rendering

In the drawing module, we created a function which takes a fractal point's escape step as the input, then maps this to the range of a predefined color list. Choosing the closest two colors from that list, the function mixes them together with the proportion corresponding to the distance from the remapped value to two different colors. This creates a nice gradient over the pixels.

1.4 Controls

To facilitate user interaction, we have implemented a rudimentary control scheme which allows the user to zoom and pan. This is achieved by keeping track of transformation values for zoom level and x- and y-scaling respectively. At the press of a button, these values are updated respectively and a flag is raised to signal the system to re-run the generating process. There additionally exists a button to choose which fixed value of c to use for the Julia set, and a button to manually raise the re-generation flag.

2 Encountered Problems

2.1 GUI

The plan for this tool is to become a fractal exploration tool, which allows the user to define various parameters to generate a fractal around. Ideally the user would be able to tweak these parameters during runtime using some graphical user interface. However, using a simple Gloss application, this would necessitate us to manually define buttons or sliders for the user to interact with, which we reasoned to be a rather expensive time-sink. Thus we will allow the user to specify these details through the command line.

2.2 Level of Abstraction

Each parameter that we abstract over in our fractal generation process increases the complexity of the fractal creation. This has raised the question how much abstraction is realistic and necessary for our project, and how much control the user should have over this process. We are still trying to integrate our *FractalFunction* data type with the rest of our code and deciding on how to condense the required choices in this data type; choices like the degree of the polynomial, which parameter is varied (degree, starting point z or offset c) or whether we compute the absolute value of z (making it a burning ship fractal) or not. For now, these will be the choices included in our interface. If there is time left, we might look at how to implement nova and Newton fractals as the computation of these kinds follows a different approach.

2.3 Parallelism

During the trial of parallelization, our initial plan was to make our app run on the GPU because of the massive amount of pixels and steps of the fractal sequence. But we failed to do it due to unforseen

errors during the installation of Accelerate. Additionally, it does not appear to support Windows as of now. We tried to use CPU acceleration to paralellize over CPU cores, but again, due to the very deep computational mapping sequence, the overhead of parallelism, and the hardness for supporting lazy evaluation, the boost of overall running time was not obvious.

3 Future Developments

Below is our estimated project schedule for comparison as suggested in the project proposal. We are currently in week 12.

Week 9	Setting up the working environment (Github, Cabal), basic reading
Week 10	Style guide definition, First fractal implementation
Week 11	First render engine implementation, fractal rendering pipeline
Week 12	Abstract fractal definition, implement more fractals
Week 13	Implement basic movement (pan, zoom), prepare presentation
Week 14	Time for unforeseen challenges, Code cleanup, Start with final report
Week 15	Report, Code cleanup

Table 1: Estimated schedule

Considering what we have already established, and how our course changed over the past few weeks, below is our newly proposed schedule for the remaining time:

Week 13	Abstract fractal definition, user input, prepare presentation
Week 14	Parallelisation, Code cleanup, Start with final report
Week 15	Report, Code cleanup

Table 2: Updated estimated schedule

3.1 Parallelism

With the help of Ivo and David, we will try to use the latest source from GitHub to build our acceleration library dependencies, and make it run on the GPU.

3.2 Abstract fractal definiton

We will continue building an abstracted fractal generation interface and integrate it with an user input system.

3.3 User interaction

We plan on improving the interactability of the tool over the coming weeks as well, implementing a command-line set of questions through which the user can specify how to generate a fractal; either using a preset definition, or by describing their own set of parameters.