Thesis Paper

Mikail Gedik

June 5, 2020

Contents

1 Abstract

This paper tackles the calculation of a few selected fractals and shines light on the core aspects I have implemented and furthermore optimized to use all the resources provided by the computer. My journey begins at a single threaded Java program and ends in a multi threaded C application able to make use graphics cards. Additionally, I will make an easy-to use yet powerful UI, which will enable even tech-unfamiliar people to use my software.

1.1 Program Capabilities & Expectations

Here I will be listing all capabilities my program should have, from easiest to hardest.

1.2 General

1.2.1 Operating System Independence

As a Linux user, I am heavily outnumbered by Windows and Mac users. Because of this, my software has to run on the other platforms, or else very few people will be able to test or run my program. Due to me beginning the project in Java, this will only become an issue when switching to C++. I will be using the Vulkan API to use video cards, which is also platform independent and shouldn't cause any issues.

1.3 Backend

1.3.1 Basic Java Single Threaded Mandelbrot Set Calculation

The first goal is to make a simple Java program, which will calculate the Mandelbrot Set and output the result into a file. It is not yet intended to be structurally divided and good as shown in INSERT_IMAGE_NAME, but more closely together as there are yet not enough parts to separate the code.
Next up would be the implementation of the settings using a simple database like a list.

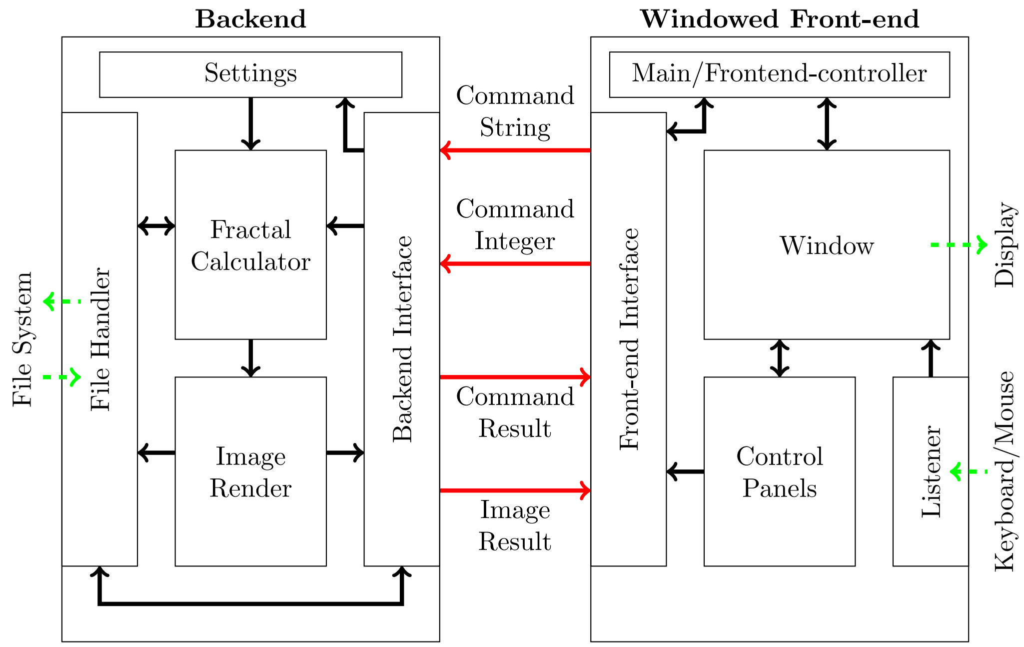
1.3.2 Basic Java Single Threaded Mandelbrot Set Calculation

1.4 Windowed Frontend

1.5 Other Known Programs

2 Program Structure

My program is separated in two halves, the front- and backend. While the latter is autonomous, the former is dependent on the backend, as it must know its interface, which is the connecting bridge between them. The front-end can request or order a command through a command string or an integer, i. e. a character sequence or a number. The backend answer these with a command result, which may also be an image.
Because the backend is detached from the front-end, I may also develop multiple front-ends for different needs. The front-end in the diagram below shows a possible version of window-oriented front-end, which is relatively modern and user-friendly, but for example useless in a command line environment. Thus I see the need to develop different front-ends.



2.1 Backend

The backend consists of multiple components or modules, which are tightly bound together. Each of them is responsible for a certain task and should not do anything else. The modules are only capable of communicating to each other, but not to the 'outside'. Only the interface and the file handler may connect with the front-end or the file system respectively.

2.1.1 Interface

The interface is responsible for a fast and reliable communication with the operating system and the front-end. In essence, all it has to do is accept orders from the outside, validate them and pass them on to the corresponding component in the backend. If the command expects a result, the interface will return it.

2.1.2 Settings

The settings store variables which are used in computation. It can be roughly divided in two parts, the mathematical and informational half. The former is concerned with parameters concerning the fractal, i. e. the expected width and height, the viewport and other parameters (for example the C in the Julia-Set), while the latter stores information which are necessary to know for optimizations, like the thread count of the processor, the available RAM and disk space and more technical details. Although the name settings suggests that these are variable, some may be constant through the program's life cycle and must not be altered in any way, like the CPU's name.

2.1.3 Fractal Calculator

Upon receiving the command from the interface, the fractal calculator starts to calculate a set of points according to the parameters fetched from the settings module. After having created all necessary information to start the rendering process, the calculation's result is given either to the image render module or to the file handler. The fractal calculator is the core part of my thesis paper and I will spend most of my time tinkering with and finding optimizations for it.

2.1.4 Image Render Module

Having calculated the appearance of a fractal is not the same as showing it on the display. The image render module is capable of turning an array of points into an image. While this is rather simple with 2D fractals, it becomes and more of challenge for 3D fractals or even multidimensional fractals. Anybody who has already had some experience in 3D rendering knows that there is more to it than just shapes. Lighting and sampling are a game changer in this regard and make an image more pleasing to look at.
After rendering the image, it is sent to the interface or to the file handler to save a copy to the local disk.

2.1.5 File Handler

This module can connect to the local file system and read or write data there. It also compresses or decompresses it to save disks space or to optimize the disk read process. The data written there are either images of fractals, videos zooming into a fractal or raw compressed information of it.

2.2 Frontend

2.3 Connection between Front- and Backend

3 Libraries

Used (or planned) libraries:
ffmpeg for video
PPM (self coded for debugging)
stb for PNG (https://github.com/nothings/stb)