Thesis Paper

Mikail Gedik

August 17, 2020

Contents

List of Figures

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 figure 1 (Page 2), 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.

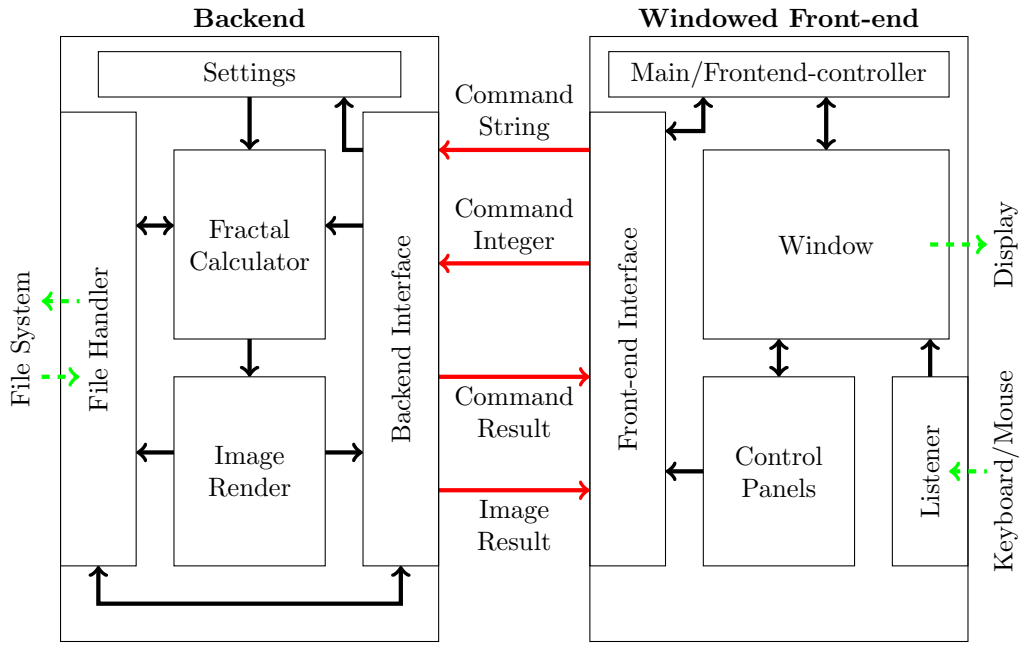
2.1 Backend

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.

Figure 1: Program structure



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 Version History

In this section we will take a look at the creation history of my program and discuss its most important aspects. Because not every version brings noteworthy changes with themselves, I have chosen split the log in two parts; All the important changes will be noted here, the lesser relevant are in the appendix A (page 5).

3.1 Java Version

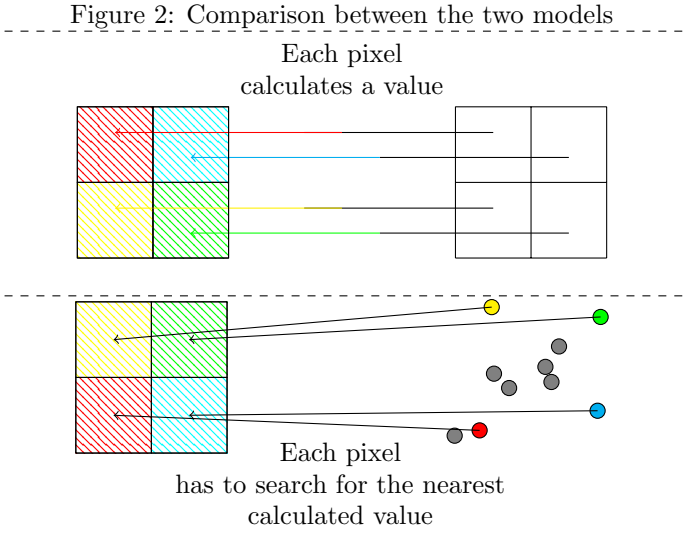
3.1.1 Java 0.0.1 (68b6e0e)

When running the program, you may notice that the timings are not as expected. Here is a sample output:

```
Version: Java 0.0.1
Time to calculate fractal: 00064 ms
Time to calculate image:   01287 ms
Saved image to: /home/mikail/Desktop/File.png
Time to save image:        00023 ms
```

As you can see, generating the actual data took way less time than transforming the data into an image. This is due to the data not being aligned with the pixels of the image. In contrast to the first version, the `MandelbrotCalculator` is not aware of the specifications of the image (i. e. location, size and resolution) and is only informed about the rough area it has to calculate. As a result, the `MandelbrotCalculator` creates a dataset that will only approximately match the pixels of the image, if at all. When the dataset is used by the `ImageCreator`, it has to search the next nearest value for each pixel, leading to tremendous render times. The figure 2 (Page 3) shows this behavior visually.
The reasoning behind this is quite simple: On one hand, the data set does not have to match the image created in the end. This makes it easy to create cropped images, images with a lower resolution (if needed) and the like. On the other hand, multidimensional fractals cannot be stored in 2D-image arrays. Although I have yet only created 2D images, I want to create a future-proof way of saving the data. Another advantage is removing the redundancy: As clearly seen in any plain example image, there are a lot of recurring patterns. In the Mandelbrot set (or any fractal for that matter), I am primarily interested in what is in and what is not in the fractal. So instead of saving each point, it would be much easier to save only the dividing line between the two areas. Thus, using an image to save the data set ceases to be an option, and I have to use my own way to save the data. This comes with the cost of of losing the ability to compress the data, because the algorithms for these run only on images, and not on custom defined data sets. Lastly, I want to address the issue with the time: I am aware of the fact that it is not acceptable that the rendering takes more time than calculating. I will try to tackle this issue in a further version.

Figure 2: Comparison between the two models



3.1.2 Java Version 0.0.3 (fcaf89f) - First UI

A simple UI has been added. The UI consists of the rendered image and log in the center, a refresh button in the south and a menu bar. Currently, only the `save`, `close` and `Open settings` menuitems are working. Saving brings up a dialog where one can choose where to export the image (currently, only `png` is supported). When opening the settings, a dialog with all the changeable options comes up. Upon changing, make sure to hit the `Save and exit` button. The program is smart enough to only update the entries that were changed.
The main aspect is the image in the center. When using the mouse wheel on it, one can zoom in and out of the set. With drag and drop, one can move around in the image. However, because on each iteration the program has to recalculate the image, it may take a long time to just move on pixel to the side.

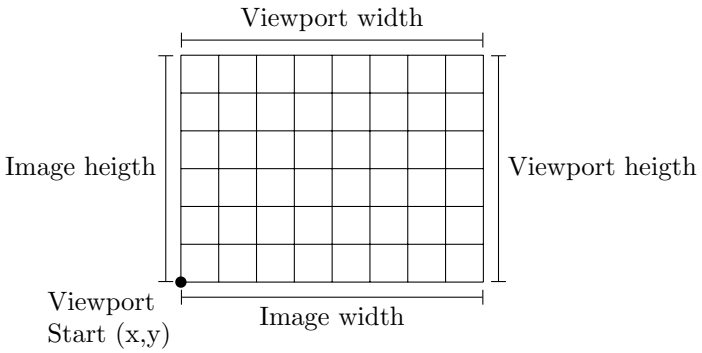
4 Key Functionalities

In this section I will explain the key functionalities of my program schematically, i. e. how the algorithms work. Naturally, there are often multiple approaches on how to solve a problem, resulting in having different solutions for the same problem. However, they are not equal in execution time and resource management, thus bringing diverse advantages (and disadvantages) in different situations.

4.1 Image Structure

After the image has been generated from the data, it is stored in a bitmap (i. e. in a raster). The underlying implementation uses a one-dimensional array. Before generation, the viewport has to be specified. It defines the area which has to be rendered.
Notice that the width and height are defined twice. Once for the image, and once for the viewport. These two values are not to be confused. The dimensions of the image describe the amount of pixels, and therefore the resolution. Changing these values will not move you around in the fractal or change your view, but you will see the image with more precision and it will appear less blurred.
The dimensions of the viewport describe the sector of the fractal which is rendered. The viewport also contains a start point, which the image dimension lacks. Moving around the point will move the area up and down or rather left and right. Changing the width and height will zoom in and out of the image and stretch it.
One big problem here is the aspect ratio. Assume that the viewport specifies a width of 10 and a height of 5, but the image is 100 pixels tall and wide. The aspect ratios are 2:1 and 1:1 respectively. The result will be a squashed/stretched image as seen in figure 4 (Page 5).

Figure 3: Image versus Viewport



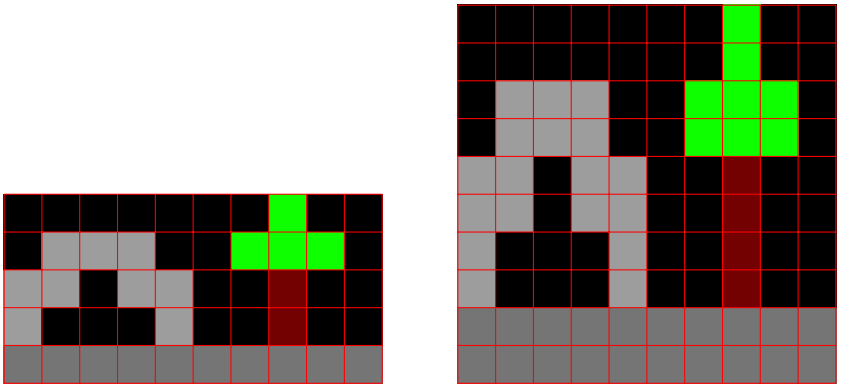
4.2 Data

Data calculated can be stored in many ways, from simple arrays to complex databanks. I have taken different approaches on how the data is stored

4.2.1 Directly in Image

The data is stored directly in the image, i. e. each pixel represents a coordinate and is given a color representing its value (currently white if it is in the set, pink otherwise). The big advantage is that the calculation can be directly shown on screen; i. e. there is no conversion between from data to the image, giving a good performance. Disadvantages are that it is not flexible. Assume that one wants to move around or zoom into the image. A new image has to be created, and if the old and new viewport overlap, some data might be recyclable. If a third

Figure 4: Aspect Ratio Problem Visualized



Viewport Image

image has to be produced, the program has to look at two different images to calculate the new image instead of one dataset.

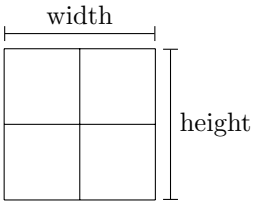
4.2.2 Simple List

The data is stored in an list, each value consist of a pair of coordinates and a value. To create an image, we assign each pixel a coordinate and iterate over the list to get the value at that location of the set. The disadvantage is the aforementioned fact: to find the value of a pixel, the whole list has to be searched. This is a very time-costly operation and leads to the absurd fact that it takes less time to create the list than the image.

4.2.3 Multiple Sorted List

Instead of shoving all the data in one big list, the data is split according to their position in smaller lists (so called clusters in my program). These clusters only save data from a specific area, and when searching for the value of a specific coordinate the program only has to search in one cluster instead of all the values.

Figure 5:



5 Libraries

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

A Complete Version History

A.1 Java Version

This is the Java version written in Java.

A.1.1 Initializer (6bdf295)

This commit existed for the sole purpose of creating the repository and verifying that it actually works.

A.1.2 First Version

Because the very first version was a simple one-file program, I decided against committing it. But for legacy purposes, I will shortly explain what the code did: It simply made an image and calculated the value of each pixel and saved the result in a file. It was all single threaded and made no use of any optimization.

A.1.3 Java 0.0.1 (68b6e0e)

According to the figure 1 (Page 2), the components are now separated in different classes and packages. The "parent" package is called `ch.mikailgedik.kzn.matur` and contains the following sub-packages, which names should be self-explanatory: `calculator`, `filemanager`, `render`, `settings`. The only outcast is the `MainClass.java` file, which lies directly in the parent package. It combines the classes and has more of a testing character than an actual use. This is due to the lack of an existing interface and frontend.

Table 1: Changelog for Java 0.0.1

Class name	Class description
`CalculationResult`	This class stores the calculated information in an `ArrayList`, which is initialized by the constructor with a given size. I have done this because each time the list has to be resized, all the elements in it have to be copied into a new array. Another feasible approach to solve the resizing problem would have been to use a `LinkedList`, as they can be expanded instantly and use only as much memory as they are big, but their performance suffers greatly when accessed randomly (as opposed to sequentially). The datatype stored in the list is defined as a generic type, and is currently the `CalculationResult.DataMandelbrot` class. Furthermore, the class implements the `Iterable` interface, so that it can be used in `foreach` loops.
`CalculationResult.` `DataMandelbrot`	This class stores one calculated point of the Mandelbrot set. The coordinates are stored as two doubles named `x` and `y`, the actual value is a boolean called `value`, which is true for all points in the set and false for all the others.
`MandelbrotCalculator`	This class does the actual mathematics regarding the Mandelbrot set. The generation is started by calling the method `calculate`, which then returns an instance of `CalculationResult` with all the calculated points. Apart from that it also holds an instance of the settings, meaning that other components must access this one to get a hold on the settings, like defining the viewport.
`FileManager`	This is the class which interacts with the filesystem. It is currently useless, because it implements a sole method, which saves an image to a file. I have only included this class, so all the code related to saving will not end up in a random class but in a specific one. This way the requirement of modularization is being fulfilled.
`ImageCreator`	The image creator converts raw data to an image. For now, it only has to create an image from the data retrieved from the `MandelbrotCalculator`. The challenge here is that the calculated points' coordinates do not match the exact coordinates of the image. Thus, each pixel has to search for the nearest point in the data set. This is a lengthy process, but I will go into more detail below.
`SettingsManager`	The `SettingsManager` contains all preset parameters and information required to calculate the fractal. The settings are stored in a `LinkedHashMap` and accessible through the methods provided. The elements of the list must implement the `Setting` interface and its inherited method `identifier()`. There are currently only two settings available, the `SettingViewport` and the `SettingNumber`, which store data for the viewport and data that can be expressed in scalars respectively.
`TestMain`	As the name suggests, this mainclass is only here to test the components. It replaces a proper frontend, which I have not yet created because of the big amount of work associated with it.

A.1.4 Java 0.0.2 (892e1e0)

This version updates the `SettingsManager` to use a `TreeMap` with strings as keys and objects as values. Previously, all the settings were sorted by their type in different classes (e.g. `SettingViewport`). As of now, the settings are stored ordered (for performance reasons) in a `TreeMap`. The main advantage is that adding a new setting is now not as cumbersome as it was before, because it suffices to add a new entry by calling the `SettingsManager.addSetting(String name, Object value)` instead of having to create a new class for it or add an additional field. The drawbacks are longer retrieving times, as the whole map has to be searched for every value rather than retrieving one class and getting the value of one of its fields. Aside from this, the default settings are not hardcoded anymore but stored in a file named `resources/settings/defaultsettings`. The `resources` directory contains all files related to resources, i. e. data required for the program (in contrast to executable code).
In order to address the values without to type out the whole key, constants exists in the `SettingsManager`. Their names should be self-explanatory. Values are retrieved by calling the `SettingsManager.get(String name)` method. However, because the datatype of the setting is often already known, I have also implemented a few methods which automatically convert the value to a given type (`getI`, `getD` and `getS` for integers, doubles and strings respectively). These methods throw an exception the setting requested does not exist.
Aside from the these changes, I made minor adjustments in the `FileManager` class. All other classes have only be changed to accustom the new `SettingsManager` class