Mandelbrot set

Teodora Taleska

UP FAMNIT

E-mail: 89201041@student.upr.si

In this paper is presented a framework for rendering the Mandelbrot set. It is demonstrated how to draw the complex fractals on a canvas serially, with multiple threads and with remote method invocation (RMI) using the Escape-time algorithm.

Finally, the results of the final computations are represented in numerical form.

1. Introduction

The Mandelbrot set is the set of complex numbers ***c*** for which the function

$$f_c(z_{n+1}) = z_n^2+c$$

does not diverge to infinity when integrated from z=0, i.e., for which the sequence $f_c(0)$, $f_c(f_c(0))$, etc., remains bounded in absolute value.

Thus, a complex number ***c*** is a member of the Mandelbrot set if, when starting with $z_0=0$ and applying the iteration repeatedly, the absolute value of z_n remains bounded for all n>0.

For example, for c = 1, the sequence is 0, 1, 2, 5, 26, ..., which tends to infinity, so 1 is not an element of the Mandelbrot set. On the other hand, for c=-1, the sequence is 0, −1, 0, −1, 0, ..., which is bounded, so −1 does belong to the set.

The graphical representation of the set can be obtained by assigning a color to each point **c** in the complex plane, depending on whether (and how quickly) the sequence $f_c(z)$ diverges. More specifically, a repeating calculation is performed for each x, y point in the plot area and based on the behavior of that calculation, a color is chosen for that pixel.

Algorithm: Escape-time algorithm

```
1: function ESCAPE-TIME(ci, c, N)
2:   i ← 0
3:   z ← 0, zi ← 0, z2 ← 0, zi2 ← 0
4:  while i ≤ N and z2+zi2< 4 do
5:    zi ← 2*z*zi + ci
6:    z ← z2 – zi2 + c
7:    z2 ← z*z
8:    zi2 ← zi*zi
9:    i ← i+1
10:  end while
11:  return i/N
12: end function
```

Here, N is the maximum number of iterations, $z = Re(z^2+c) = z^2 - zi^2 + c$, $zi = Im(z^2+c) = 2*z*zi + ci$, and this algorithm assigns colors to individual pixels based on the "Escape-time" value for every c.

Since every pixel is processed independently from the others, this algorithm can easily be parallelized, and adapted to use multiple threads.

2. Problem description and design

The program was implemented in three modes: sequential (serial) mode, parallel (multi-threaded) mode and in distributed (RMI) mode.

Sequential (serial) mode: This is the most basic mode, that uses a single thread for everything.
Once the program is loaded the function for drawing the graphics on the canvas is invoked and the pixels are drawn one by one sequentially.

Parallel (multi-threaded) mode: In this mode, once the application is loaded, the work of the tasks is divided among several "worker" threads and all threads are running in parallel. After the worker threads have finished their work, they should send back their chunks to the main thread, which will reconstruct the final image.
Hence, the algorithm for painting the graphics is represented as a task, which can be divided into multiple subtasks and in this case the division is made on the x axis, where multiple threads are painting different parts of the canvas in parallel.

Distributed (RMI) mode: RMI (Remote Method Invocation) is a distributed object system. In this mode we have the illusion of calling a local method from a local class file, when in fact the arguments are shipped to the remote target and interpreted, and the results are sent back to the callers. In our program, when the methods are called from the remote interface, they are executed as in the sequential mode. Therefore, in this mode, the pixels are also painted one by one sequentially, but it differentiates in the way the methods are called.

3. Technical remarks

The program was implemented in JavaFX, which is a Java library, and and with the help of JavaFX scene builder.

A technically interesting part of the implementation was implementing the distributed part with the Remote Method Invocation distributed object system, which includes a remote interface, implementation of the remote interface, a server class, and a client class.

A point to note is that each method in the remote interface must throw a java.rmi.RemoteException.

The implementation of the remote interface is used to create a single, non-replicated, remote object that uses RMI's default TCP-based transport for communication.

In the server class, the remote object registered with the RMI registry is running on a host "`127.0.0.1`" and port 1099, which is the default port number.

And finally, the client remotely invokes any methods specified in the remote interface. To do so, the client must first obtain a reference to the remote object from the RMI registry. Once a reference is obtained, any method can be invoked.

4. Results and conclusion

In the results, we notice that as the height and width of the interface increases, more seconds are needed to render the pixels. Comparing the execution time of all the parts, the sequential part takes the least time, and the parallel part takes the most time, so it puts the distribution part in the middle.

To sum up, the presented methods are suitable for rendering the Mandelbrot set on the canvas at high resolutions.