Mandelbrot

Ruxandra-Stefania Tudose

Spring Term 2024

Introduction

This report aims to open a window onto a fascinating combination between on the one hand, mathematics and on the other hand, functional programming. By using Elixir and RGB color codes, we will visualize the points depending on them being or not in the Mandelbrot set. What is even more interesting is that despite the mathematical calculations being quite straightforward, they can be optimised in order to save up time.

The maths behind the problem

Complex numbers

Of course, in order to be able to program, so that one can generate the visualisation of the Mandelbrot set, one also has to understand the maths behind it, primarily the functions which will manipulate the complex numbers and the condition of a constant being or not in the set depending on its absolute value. Complex numbers will be represented as tuples where the first element is its real part, while the second the imaginary one. In order to decide if a number is or not in the set, we have to create, add, square and calculate the absolute value of a complex number. Therefore, functions that deal with these operations have been created in that sense.

```
def new(r,i) do {r, i} end #creating a new complex no
def add(a = {r1, i1},b = {r2, i2}) do #adding two complex no
  {r1+r2, i1+i2}
end

def sqr(a = {r, i}) do #squaring a complex no
  case i do
    0 -> {r*r, 0} #separate case for real numbers only
    _ -> {r*r - i*i, 2*r*i}
  end
end
```

```
def abs(a = {r, i}) do #computing the absolute value
    :math.sqrt(r*r + i*i)
end
```

Note: In my implementation, I did not add the atom *:cpx* that identifies that a given tuple is a complex number using pattern matching. However, after having attended the lecture, I understand that in more complex programs it is a good practice to do so since this way one can avoid possible data confusions.

Is a given point in the Mandelbrot set?

Of course, the given recursive equation can be computed many, many times, therefore the condition that saves time is that if the absolute value of a complex number is *strictly greater than two*, that number definitely is not in the Mandelbrot set. Having said that, this is the condition we have to check within the number of iterations. It should be taken into account that if the number is not in the set, we have to return the number of the iteration where the aforementioned condition was no longer respected. All these, will be done using the functions *mandelbrot/2* and *test/4* which were implemented using the previously presented methods.

```
def mandelbrot(c,m) do
  z0 = Cmplx.new(0,0);
  test(0, z0, c, m)
end
def test(i, z0, c, 1) do
  if (Cmplx.abs(z0) > 2) do #this can be improved timewise!
    i
  else
    0
  end
end
def test(i, z0, c, m) do
  if (Cmplx.abs(z0) > 2) do
    i
  else
    test(i + 1, Cmplx.add(Cmplx.sqr(z0),c), c, m-1)
  end
end
```

After having run the tests indicated in the PDF instructions, I noticed that depending on the number of iterations, a number may or may not be in the

Mandelbrot set. For example, take the number $\{0.26,\ 0\}$. In other words, for thirty iterations, this number is part of the set, while for fifty it is not since the absolute value condition is violated on the thirtieth iteration.

Generating the Mandelbrot set

In order to generate the Mandelbrot, the provided PPM module has been used, as well as the available provided RGB color transformations and conversion formulas. Since violet is my favorite color, I tried to generate it that way by changing the RGB combinations and obtained the Mandelbrot set as it can be noticed in *Figure 1*.

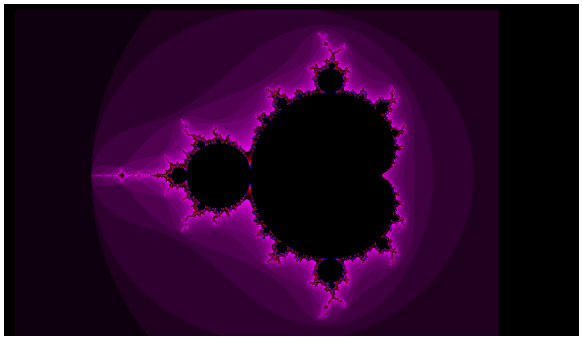

Figure 1: Illustration of the generated Mandelbrot set.

At the same time, *Figure 2* showcases a unique image which was obtained by having zoomed in where the branches of the Mandelbrot start taking spectacular shapes. This image's starting point finds itself in the complex plane at (-0.1, 0.7).

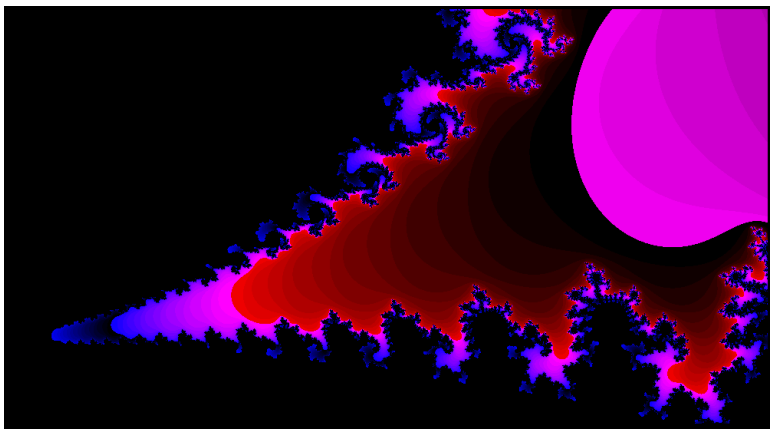

Figure 2: Zooming into the Mandelbrot illustration.

The tricky thing in my opinion was the conversion from the (x, y) plane to the complex one. As explained in the lecture, the approach consists in taking the coordinates of the image pixel, converting them to the associated complex number by using the parameter function, calculating the Mandelbrot depth depending on which the associated color is then, printed. In addition, it is also important to note that one does this by taking it row by row from bottom to top and by taking each pixel on the row until the corresponding width is reached.

Optimization

As presented in the lecture, three main optimizations can be done that seriously speed up the computations. The first one consists in saving time by not taking the square root of the complex number and instead, comparing its absolute value to four and not to two. One can also try to send as parameters in the *test* method the real and imaginary components to save time from actually identifying those before doing the calculations.

The next two updates are more advanced. Since Elixir uses references and always stores the values in memory, it is tedious to do all the complex number calculations. On the other hand, in programming languages such as C, all these are done and stored at register level. Having said that, the second update consists in calculating the Mandelbrot depth using a C file and importing the result in the Erlang virtual machine as a compiled object file and therefore, speed up the calculations considerably. I think that this technique and the idea of combining two programming languages is quite fascinating. Last but not least, since we are calculating and generating row by row, parallelism potential can be spotted here since not too many processes have to be spawned and the calculations are not too small. The result of the processes is then reported and collected.

Conclusion

All in all, this assignment mainly deals with a simple equation which leads to many complex results (the generated picture) and at the same time, the advanced optimization approaches. This task has been by far the most fun and interesting so far in this course. It's fascinating to see the visualisation of what a simple recursive equation can create.