Mandelbrot

Sam Khosravi

March 2022

1 Introduction

In this exercise we would computer the Mandelbrot set, which is computer by the recursive function f(z) = $z^2 + c$.

Here, z is a complex number, and most of the time, when squaring a number many times it will either get really big or really small. However, some complex numbers will instead converge to a certain point when squared again and again. This is what will give us the beautiful patterns and shapes we see in the Mandelbrot set.

2 Complex numbers

Here we created a module to handle the complex numbers we would use. The function new/2 would take in a natural and imaginary number and then return them as a tuple. The function add/2 takes in two tuples with two different complex numbers and then adds the natural parts together and the imaginary parts together and then returns them as a tuple. The function sqr/1 takes in a tuple of a natural and imaginary number and then returns the square of them as a tuple. The math here is simple, as two imaginary numbers multiplied will give us a natural number, but the part where we multiply the natural and imaginary number with a two will remain a imaginary number, and is returned in a tuple together with our natural result. The function abs/1 uses the elixir square root function to add take the square root of the square of our natural number added with the square of our imaginary number.

```
def new(r,i) do
  {r, i}
end

def add({r1, i1}, {r2, i2}) do
  {r1+r2, i1+i2}
end

def sqr({r,i}) do
```

```
{r*r - i*i, 2*r*i}
end

def abs({r,i}) do
:math.sqrt(r*r + i*i)
end
```

3 The Brot module

Here the mandelbrott/2 function was implemented which when give a complex number and a maximum number of iterations would return a value if the absolute value of z is larger than 2, or return 0 if it is not. The code was more or less given in the instructions, and we would create a complex number with point in origo, and send it to a test/4 function which will be explained next.

```
  def mandelbrot(c, m) do
    z0 = Cmplx.new(0,0)
    i = 0
    test(i,z0,c,m)
  end
```

The test/4 function will take in i, which is the iteration we are on, z which is the point in the complex plane, c which is a complex value and then m which is the maximum number of iterations. In the code, if our absolute value of z is larger than two, we will as the instructions say only return the i. However, if we still are in the span where mandelbrot gives us nice patterns, we will instead call on test/4. This time with an incremented i value and also add the square of z with c and send it back to test to satisfy f(z) = $z^2 + c$.

```
  def test(i,z,c,m) do
    if Cmplx.abs(z) > 2 do i
  else
    znew = Cmplx.add(Cmplx.sqr(z), c)
    test(i+1,znew, c, m)
  end
end
```

4 The printer

The PPM module would write the image to a file. All code was given in the instructions. I however at the end of the task changed the depth, width and height to get a better looking picture.

5 Colors

Here the convert/2 function will take in a depth that goes from 0 to max, which is its second parameter it takes in, and returns a color in a form of a tuple. So our depth can be no more than our max number. This will in turn be used for us to divide the range 0 to m into five different sections. To get this we divide the depth and maximum depth and set it to a fraction which is represented by the value f. This fraction is then multiplied by four and then truncated to get a value x that ranged from 0 to 4. We then create a y value that is based on our x value to use to choose colors in RGB. To get the different colors we will have a case that goes into a special color section depending on our value x. I played around here a bit until i got a satisfactory image with a few pixels. When i was satisfied with the colors i generated a better quality picture with the width as 15360 and the height as 8640 for better resolution in the PPM module.

```
def convert(depth, max) do
  f = depth/max
  a = f *4
  x = trunc(a)
  y = trunc(255*(a-x))
  case x do
    0->
      {y, 0, y}
    1->
      {255-y, 0, 0}
    2->
      {255-y,y,255}
    3->
      {0, 255-y, y}
    4->
     {y,255-y,0}
  end
end
```

6 Computing the set

Here we create the module Mandel which will compute the depth of all our points. The mandelbrot/6 function was given in the instructions, and its function is to give a complex value depending on where we are in the plane, which is based on the width and height we are given. It will call a rows/4 function which in turn returns a list of rows, each row being a list of colors. The rows/5 function is pretty simple, and is used to traverse each row. For each row we traverse, we will have to call on a function onerow/5. This is because we need to go through each position in that row and return the list of colors for each separate row. When we have the list we will append it to our rowlist and call on rows/5 again but now decrementing the height to get to the next row until we reach our base case.

```
def rows(width, 0, trans, depth, rowlist) do
  rowlist
end
def rows(width, height, trans, depth, rowlist) do
  onerow = onerow(width, height, trans, depth, [])
  rows(width, height-1,trans, depth, [onerow|rowlist])
end
```

The onerow/5 function is used to go through every row. Therefore we go decrement the width until we reach 0, which is our basecase and where we return our list of colors for that row. We want the depth of each pixel, and therefore we call on mandelbrot, and then later give this to our convert function in the Color module, which transforms it into a color.

```
def onerow(0, height, trans, depth, rowlist) do
  rowlist
end

def onerow(width, height, trans, depth, rowlist) do
   i = Brot.mandelbrot(trans.(width, height), depth)
   color = Color.convert(i, depth)
   onerow(width-1, height, trans, depth, [color|rowlist])

end
```

7 Discussion

I really enjoyed the Huffman task and it was my favorite until i did Mandelbrot. This task was not only fun but interesting. In the beginning i did not know anything about mandelbrot, and i assumed that it was a made up scenario where we had a almond that would be broken up into pieces (Mandel = Almond in swedish and Brott = Rupture) and that we would have to do something with these like the Lumberjack exercise. However, when i looked up the mandelbrot set on youtube and got it explained for me i instantly knew that this exercise would be really fun. The code we had to write was pretty easy in comparison to some other tasks. When i was done with the code i tried around a few different colors until i found something i liked and then gave the width, height and depth better precision. My computer is not the fastest and it took me approximately 6 minutes to generate a picture that looked ok. However, i 4x the resolution to width of 15360, height of 8640 and a depth of 200 to get a smoother picture and left my computer running while I did other things. All and all this is a task i think should never be removed from the course as it shows a cool connection between maths and nature, and reminded me of the golden ratio.

Disclaimer: My real Mandelbrot picture exceeded the size limitation overleaf has and therefore i instead included it as a extra PDF in the task.

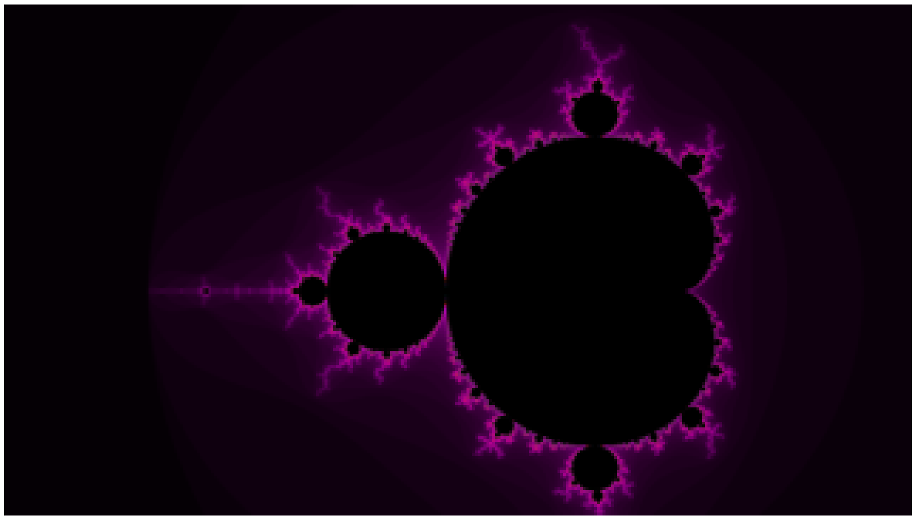

Figure 1: My Mandelbrott