Mandelbrot

Azer Hojlas

March 2022

Introduction

This assignment is based on implementing functions that collectively display a Mandelbrot set. Assuming the reader has researched mandelbrot, points, measured as complex constants c, are stable if none of the next coming values Zn approach infinity. They are unstable if any of these Zn in the set are larger than 2, as that value would quickly approach infinity given $Z_{next} = Z_n^2 + c$. If the value belongs to the mandelbrot set however, computing the series will continue in perpetuity. Thus, in order to save time, a maximum amount of iterations m is decided upon that will stop the computing once that number of iterations has been reached. This constitutes as a problem however, as this limits the amount of times the program can iterate in order to find unstable values. Thus, it is up to the user to decide an m that is a compromise between these two.

Code in depth

The ppm module will not be discussed as it was given in the assignment instructions. The same goes for Cmplx and Color. Cmplx because it is simple and doesn't require much to understand, and Color because it has to do with RGB values which the author does not fully understand. The author simply followed the assignment instructions to implement the Color module and proceeded as such. Thus that leaves Brot and Mandel as the modules that will be covered.

Brot

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

```
def test(i, z, c, m) when i < m do
    case Cmplx.abs(z) > 2 do
      false ->
        next = Cmplx.add(Cmplx.sqr(z), c)
        test(i + 1, next, c, m)
      true -> i
    end
end

 def test(_, _, _, _), do: 0
```

Brot is essential when computing whether starting points belong to the mandelbrot set or not, that is, whether they are stable or not. It starts by initiating a starting value Z_0, which for mandelbrot computations is set to initial values of zero for both the real and imaginary part. The variable i represents the iteration at which a Z becomes unstable, that is, if it is unstable. In the test/4 base-case, if the amount of iterations increments towards the earlier mentioned maximum depth, then 0 is returned. This indicates that the code assumes c to be stable given the max amount of iterations m. In the recursive clause, the boolean value $|z| > 2$ decides which course of action to take. The constant c is considered unstable if $|z|$ is larger than 2. In that case, the depth i is returned, indicating at which depth c became unstable. It can also be interpreted as the relative instability of c. The test() false condition computes the next value in the set by squaring the previous Z and adding the constant to that value. It proceeds by calling itself with the new Z and the depth counter i incremented. This continues until either z becomes larger than 2, upon which it is considered unstable and i is returned, or until the maximum amount of iterations has been reached upon which 0 is returned.

Mandel

```
def mandelbrot(width, height, x, y, k, depth) do
    trans = fn(w, h) ->
      Cmplx.new(x + k * (w - 1), y - k * (h - 1))
    end
    rows(width, height, trans, depth, [])
end

def rows(w, h, trans, depth, rows) do
    case h == 0 do
      true -> rows
      false ->
        row = row(w, h, trans, depth, [])
```

```
        rows(w, h - 1, trans, depth, [row | rows])
    end
end

def row(width, height, trans, m, row) do
    case width == 0 do
      true -> row
      false ->
        c = trans.(w, h)
        convert = Brot.mandelbrot(c, m)
        color = Color.convert(convert, depth)
        row(width - 1, height, trans, depth, [color | row])
    end
end
```

Mandel constructs an image given a set of coordinates, in the dimensions of $width * height$. The variables x and y are the coordinates (imaginary and real part) for the current constant c for which the depth is calculated. K is the distance or offset between two points, decided in the Test module. This is an interesting variable and will be mentioned in the conclusions. The initial depth is the aforementioned m in the Brot-module. In the mandelbrot/6 function, the desired dimensions as well as the starting coordinates of the upper left corner in the image are supplied as arguments. The depths of each row is then calculated by invoking rows/5. This clause will hold an accumulator, *rows*, that holds all the depths of each individual row. The base case is triggered when height is decremented to zero, that is, when all row depths have been calculated, upon which the color coded depths are returned.

In the recursive part, row/5 is called to calculate the depths of a single row. Each row() invocation is done with different height values, the width however stays constant for each rows-call. Once in row(), the depth of each coordinate is calculated by invoking mandelbrot() from the Brot module. The argument c, is calculated by invoking trans from the previous function. Each new c is calculated by decreasing the width for each iteration in $x + k * (w - 1)$. Thus, all the points for the entire row can be calculated as each new iteration brings about a new c in the direction of the row. All point depths have been calculated once width reaches zero and thus row can be returned, which contains the colors for all depths in the row. This continues along all rows until height reaches zero, upon which, as mentioned before, all the corresponding colors for each point will be returned.

Image

When the Mandel module finishes its computations, the program is left with a list of color encodings for each depth. This list of lists is forwarded to the PPM module that creates a .ppm image with a name chosen by the client in the terminal.

The following is the test-function provided by the assignment.

```
def small(x0, y0, xn) do
    width = 10000
    height = 3000
    depth = 64
    k = (xn - x0) / width
    image = Mandel.mandelbrot(width, height, x0, y0, k, depth)
    PPM.write("small2.ppm", image)
end
```

The author found out that by tinkering with the maximum depth that one could affect the contrast between various depths, which would improve the quality of the image at the cost of time complexity as a higher m requires more iterations in the Brot module. Furthermore, increasing width decreases the offset k as width acts as the denominator in the computation of k. Thus, it allows for more zooming of the pictures which allows for some exciting exploring of the picture. Ideally, the offset should be pixel-sized to allow the set to be illustrated as accurately as possible. These images however, exceed a size of 150 MB, thus, the dimensions from the original test will be used in order to have a feasibly sized image. Only the coordinates will change in the demo()-function. The author chose green as an arbitrary RGB-colouring.

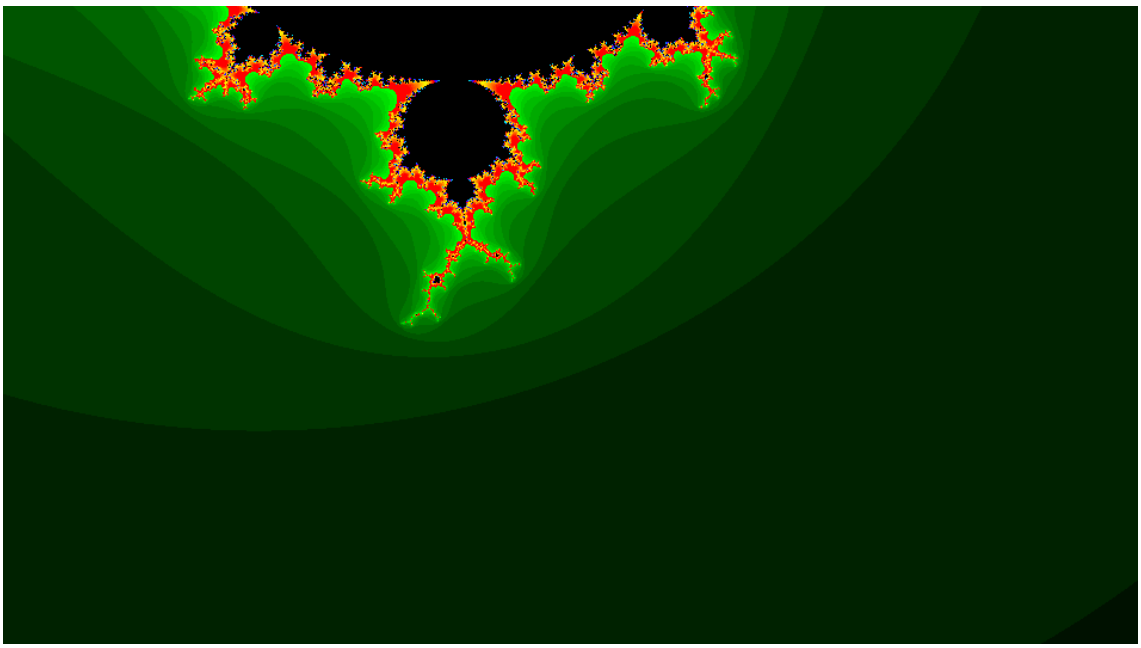

Figure 1: mandelbrot image

The choice of image was not particularly exciting as the author could not find a way of successfully converting a large-memory ppm file into a low memory png file that would show a desired specific spot. Hopefully this lower part of the mandelbrot set shows its repeating nature. That is, that the lower spherical figure or appendage to the larger one is identical in some areas, thus being self-similar.

Conclusions

Even though the submitted image in itself was not all too exciting, playing around with the demo-coordinates was very entertaining.

The coordinates of the image taken are the following:

```
x0 = -1
y0 = -0.5
xn = 1.2
```

The width and height are the same as the code from the assignment instructions. The offset k is different due to the new values of xn and x0.