Mandelbrot

Dhruba Das

Spring Term 2024

Introduction

In this assignment, we take a look at complex numbers, and the Mandelbrot set; it's defined as the set of complex numbers c for which the sequence z_n does not approach infinity; in the sequence, $z_0 = 0$ and $z_{n+1} = z_n^2 + c$.

Our main goal is to use this particular set of complex numbers to generate an image.

Cmplx module

This module can be summarised very easily: we represent a complex number with a tuple which looks like $\{: cpx, r, i\}$, where r is the real part of the complex number, and i is the imaginary part. The module contains FOUR functions: *new* (creates a new complex number given value of r and coefficient of i), *add*(which adds two complex numbers together), *abs* which gives us the absolute value of a complex number, and *sqr* which gives us the square of a complex number. An example of a function in this module is shown below:

```
def sqr(a) do
    {:cpx, r, i} = a
    {:cpx, (r*r) + (i*i * -1), (2*r*i)}
end
```

Brot module

This is the module which is responsible for checking whether a given complex number is part of the Mandelbrot set or not. It has two functions: *mandelbrot* and *test*. The *mandelbrot* function, given the complex number c and the maximum number of iterations m, returns the value i at which $|z_i| > 2$ or 0 if it doesn't for any $i < m$.

The *test* function *test* functions check if we have reached the maximum iteration, in which case it returns zero, or if the absolute value of z is greater than 2, it returns i. It also has one base case, where it returns 0 if $i = m$.

```
def test(i, z, c, m) do
        a = Cmplx.abs(z)

        if a <= 2.0 do
            z1 = Cmplx.add(Cmplx.sqr(z), c)
            test(i+1, z1, c, m)
        else
            i
        end
end
```

Colors module

This module only has one function, **convert**, which, given a depth from zero to max, gives us a color.

It does so by calculating a variable f (depth/max), which normalizes the depth to a value in the range [0, 4]; the variable a then is used to scale f to a value in the range [0, 16]; x then rounds a down to the nearest integer; y is then used to used to calculate the fractional value of a and scale it to the range [0, 255]. Finally a tuple of the following form is returned: {:rgb, 0, 0, 255 - y}. This tuple can be played around with to change the colors of the image produced.

Mandel module

This is the main module of the four mentioned so far; it's this module that is responsible for "calculating" the image. Here we have three functions: *mandelbrot*, *rows* and *row*.

The *mandelbrot* function takes parameters *width*, *height* of the image, the coordinates x and y of the center of the image, the scale k, and the *depth* parameter. It then initializes a transformation function *trans* that maps pixel coordinates to complex numbers. Then, it starts generating the Mandelbrot set image by processing rows:

```
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
```

The *rows* function generates rows of the Mandelbrot set image. It takes parameters *width*. *height* of the image, the transformation function *tr*, the *depth* parameter, and the accumulated *rows*. It recursively generates each row of the image and accumulates them. It also has e base case which just returns the rows if the height of the image is 0.

```
def rows(w, h, tr, depth, rows) do
        row = row(w, h, tr, depth, [])
        rows(w, h - 1, tr, depth, [row | rows])
end
```

The *row* function generates a single row of the Mandelbrot set image. It takes parameters current width w, height h, the transformation function tr, the *depth* parameter, and the accumulated *row*. It calculates the complex number corresponding to the current pixel using the tr function, calculates the Mandelbrot depth for that complex number, converts the depth to a color, and recursively generates the remaining pixels in the row. It also has a base case which just returns the row if the width is 0.

```
def row(w, h, tr, depth, row) do
        c = tr.(w, h)
        res = Brot.mandelbrot(c, depth)
        color = Color.convert(res, depth)
        row(w - 1, h, tr, depth, [color | row])
end
```

Final image

The final image produced, using the tuple {:rgb, 0, 0, 255 - y} in the Colors module is shown below:

NOTE: the PPM nodule will not be explained in detail as the code is given to us; it has the *demo* function which calls the *small* (method responsible for handling the dimensions and generation of the image) and stores it in a file.