Mandelbrot Set

Ashvin Oli

December 17, 2020

Contents

1 Introduction **2**

2 Implementation Details **2**

2.1 Transformation Simple 2

2.2 Linear Mapping 2

2.3 A bit of Complex algebra 2

2.4 Zooming in 2

3 Setting up SDL2 **3**

4 Building **3**

5 Code **3**

5.1 Headers and function initialization 3

5.2 Main logic 5

5.3 Map Function 6

6 Output **6**

1 Introduction

Mandelbrot set is the set of complex numbers c such that starting from $z_0 = 0$ and applying:

$$z_{k+1} = z_k^2 + c$$

repeateadly,$\forall k > 0$, $|z_k| \leq 2$

2 Implementation Details

Drawing mandelbrot set is quite easy. All we need to do is map each pixel with a complex number(c) and repeat the iteration to a "Max" number. If $|z_k| \leq 2$ upto max iteration then we color the pixel black else we color it white, or for more fun we might use the iteration count itself for coloring. Real part of complex number corresponds to x-coordinate, and imaginary part corresponds to y-coordinate. x-coodinate is columns number of screen, and y is the row number, when we try to map (x,y) of a grid to rows and columns of a matrix. The screen in which we plot is a matrix, so we draw such that the center is (0,0) and also:

$$|c| > 2 \implies |z_1| > 2$$

$$\implies |c| \leq 2$$

for any chance of convergence. So, we have to scale our screen or else only very small portion of the screen will have any drawing and that is no fun. So apply two transformation to (x,y) of each pixel. We first shift the origin to center, then we scale it such that $\forall c \in$ transformed set, $|c| \leq 2$ i.e the full screen has the width of 4 i.e radius of 2.

2.1 Transformation Simple

X $= (x - \frac{width}{2}) \times \frac{4}{width}$
Y $= (y - \frac{height}{2}) \times \frac{4}{width}$
Note: I have scalled equally in both directions to prevent distortion of image, and y axis is inverted, but this doesn't much affect our shape.

2.2 Linear Mapping

To make the function broader and allow zooming easily we may also define:
X $= output_{xmin} + \frac{output_{xmax} - output_{xmin}}{input_{xmax} - input_{xmin}} \times (x - input_{xmin})$
Y $= output_{ymin} + \frac{output_{ymax} - output_{ymin}}{input_{ymax} - input_{ymin}} \times (y - input_{ymin})$

2.3 A bit of Complex algebra

Lets say $z_k = z_{k_r} + z_{k_i} i$ and $c = c_r + c_i i$ then

$$z_{k+1} = (z_{k_r}^2 - z_{k_i}^2 + c_r) + (2 z_{k_r} z_{k_i} + c_i) i$$

where $c_r = X$ and $c_i = Y$.

2.4 Zooming in

Zooming is simply scaling the portion up or down. This can be easily achieved by chainging the ouput$_{\text{max}}$ and output$_{\text{min}}$ limits for x and y. Also maximum iteration count and precision has to be increased. In the program you may scroll up or down to zoom in but remeber that it takes time for image to render to wait for image to render before zooming continuously.

3 Setting up SDL2

To setup SDL2 in windows follow these instructions. Setting up SDL2 in linux follow this site.

4 Building

Simply:

```
make mandel.exe
```

5 Code

5.1 Headers and function initialization

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <SDL2/SDL2_gfxPrimitives.h>
#include <SDL2/SDL.h>


#define WIDTH 1000
#define HEIGHT 600
int MAX_ITER= 50;

double out_max_x= 2;
double out_min_x=-2;
double out_max_y= 2;
double out_min_y=-2;

int draw(SDL_Renderer **,int);
double map(double,double ,double, double, double);

int main(int argc, char *argv[])
{
  if (SDL_Init(SDL_INIT_VIDEO))
    {
        printf ("SDL_Init Error: %s", SDL_GetError());
        return 1;
    }
    SDL_Window *window = NULL;
    SDL_Renderer *renderer = NULL;

    window = SDL_CreateWindow("Mandelbrot Set", SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERE
    if (window == NULL)
    {
        printf ("SDL_CreateWindow Error: %s", SDL_GetError());
        SDL_Quit();
        return 2;
    }

    renderer = SDL_CreateRenderer(window, -1, SDL_RENDERER_ACCELERATED);
    if (renderer == NULL)
    {
      SDL_DestroyWindow(window);
      printf ("SDL_CreateRenderer Error: %s", SDL_GetError());
      SDL_Quit();
      return 3;
    }

    SDL_Event event;
    int quit = 0;

    //Factor is a random number that will spice things up for the image.
    int factor = 10;

    //Default value of to_render is true and is set true again when the user draws rectangle
    int to_draw = 1;

    //Clear using white color before going inside the loop
    SDL_SetRenderDrawColor(renderer, 255, 255, 255, SDL_ALPHA_OPAQUE);
    SDL_RenderClear(renderer);

    //Relative position of mouse_x and mouse_y
    int mouse_x, mouse_y;

    //Mapped mouse_x and mouse_y
    double mouse_x_mapped ,mouse_y_mapped;
    while (!quit){
     while (SDL_PollEvent(&event))
       {
       if (event.type == SDL_QUIT){
          quit = 1;

       }else if (event.type == SDL_MOUSEWHEEL) {
        float offset_x,offset_y;
        if(event.wheel.y > 0)
         // scroll up
         {
          printf("\r%-40s","Scrolled Up. Wait for image to render!");
          fflush(stdout);
          offset_x = (out_max_x - out_min_x)/4;
          offset_y = (out_max_y - out_min_y)/4;
          MAX_ITER += 20;
         }else if (event.wheel.y < 0)
         // scroll down
         {
          printf("\r%-40s","Scrolled Down. Wait for image to render!");
          fflush(stdout);
          offset_x = (out_max_x - out_min_x)*2;
          offset_y = (out_max_y - out_min_y)*2;
```

```
                MAX_ITER -= 10;
              }
            SDL_GetMouseState(&mouse_x,&mouse_y);
            double smaller = WIDTH > HEIGHT ? HEIGHT:WIDTH;
            double mouse_x_mapped = map(mouse_x,0,smaller, out_min_x,out_max_x);
            double mouse_y_mapped = map(mouse_y,0,smaller, out_min_y,out_max_y);
            out_min_x = mouse_x_mapped - offset_x;
            out_max_x = mouse_x_mapped + offset_x;
            out_min_y = mouse_y_mapped - offset_y;
            out_max_y = mouse_y_mapped + offset_y;
            SDL_SetRenderDrawColor(renderer, 255, 255, 255, SDL_ALPHA_OPAQUE);
            SDL_RenderClear(renderer);
            to_draw = 1;

          }
        }

        //Draw pixels on the renderer
        if (to_draw) {
          to_draw = draw(&renderer,factor);
          SDL_RenderPresent(renderer);
          printf("\r%-40s","Image Rendered! You may now zoom.");
          fflush(stdout);
        }


    }

    //free resources
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);
    SDL_Quit();
    return 0;
}
```

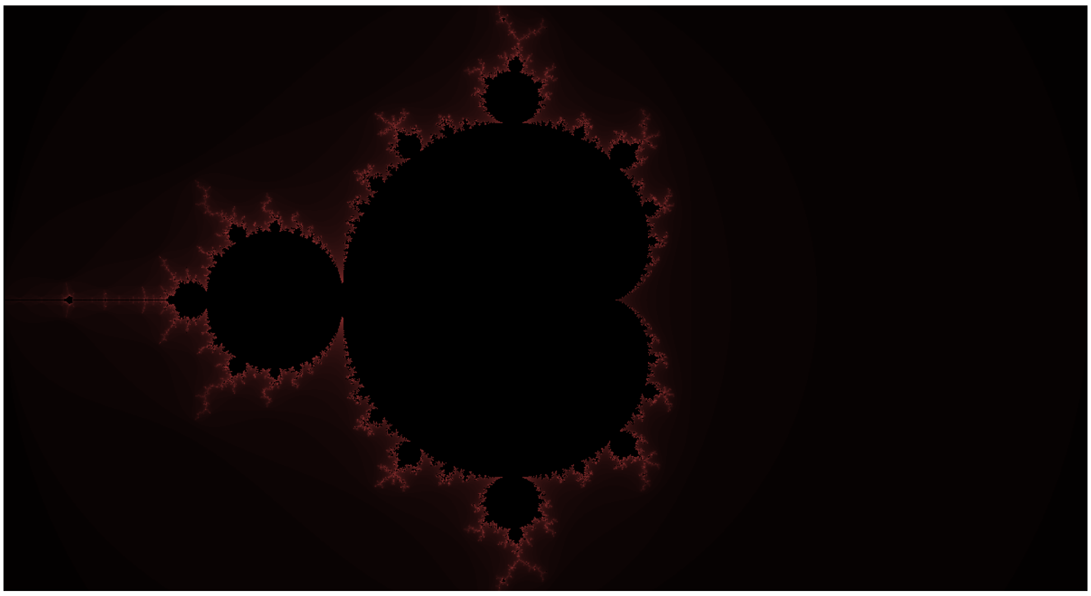
5.2 Main logic

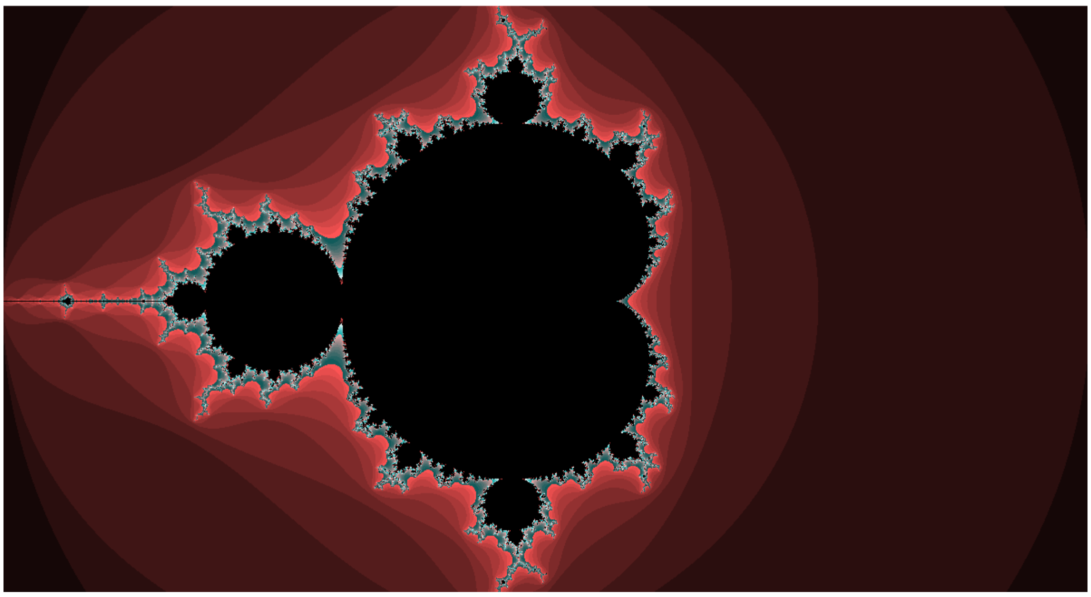
```
int draw(SDL_Renderer **renderer,int factor){
   for (int x = 0; x < WIDTH; x++) {
      for (int y =0;  y < HEIGHT; y++) {
      //Transforming and scaling such that origin is center and  radius of 2 around it. Scali
        double smaller = WIDTH > HEIGHT ? HEIGHT:WIDTH;
        double c_real = map(x,0,smaller, out_min_x,out_max_x);
        double c_img = map(y,0,smaller, out_min_y,out_max_y);

        double z_real = 0;
        double z_img = 0;
        int iter_count = 0;
        while (pow(z_real,2)+pow(z_img,2) <= 4 && iter_count < MAX_ITER) {
          double temp_real = pow(z_real,2)-pow(z_img,2)+c_real;
          double temp_img = 2*z_real*z_img + c_img;
          z_real = temp_real;
          z_img = temp_img;
          iter_count++;
        }

        //If any number exits before reaching MAX_ITER then, it is not in the set. So colour
        if (iter_count == MAX_ITER) {
          //printf("SELECT %.2f %.2f %d %d\n",c_real,c_img,x,y);
          //Draw with black
          SDL_SetRenderDrawColor(*renderer, 0,0, 0, SDL_ALPHA_OPAQUE);
          SDL_RenderDrawPoint(*renderer,x,y);
        }else{
           //Draw with custom shade
          SDL_SetRenderDrawColor(*renderer, iter_count*factor*5,iter_count*factor, iter_count
          SDL_RenderDrawPoint(*renderer,x,y);
        }
      }
    }
    return 0;
}
```

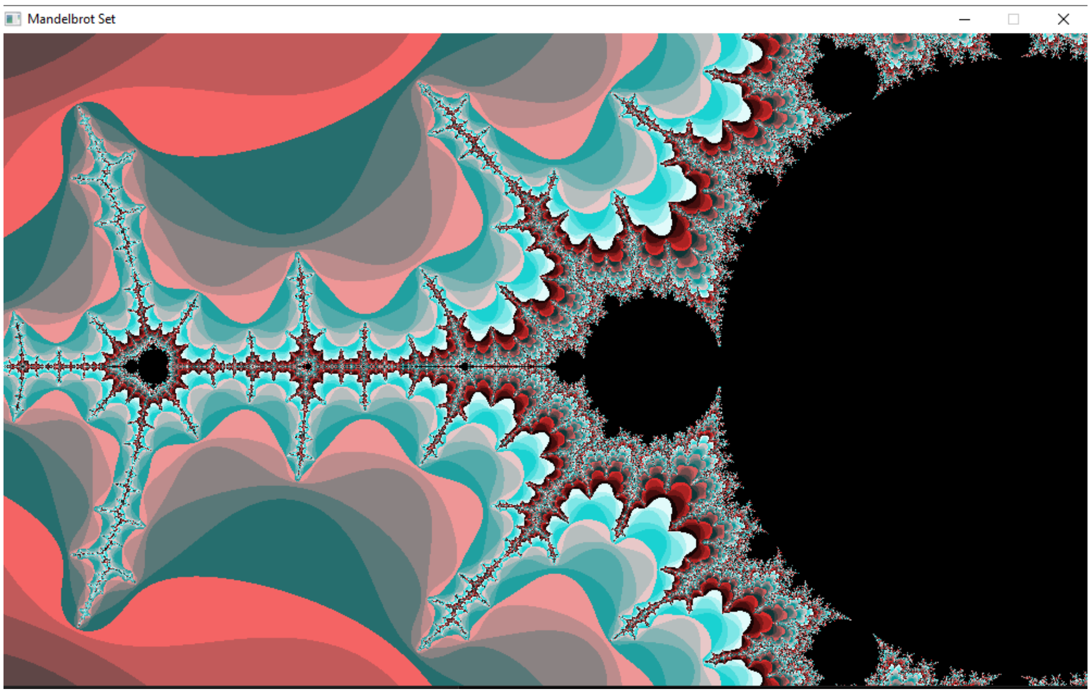
5.3 Map Function

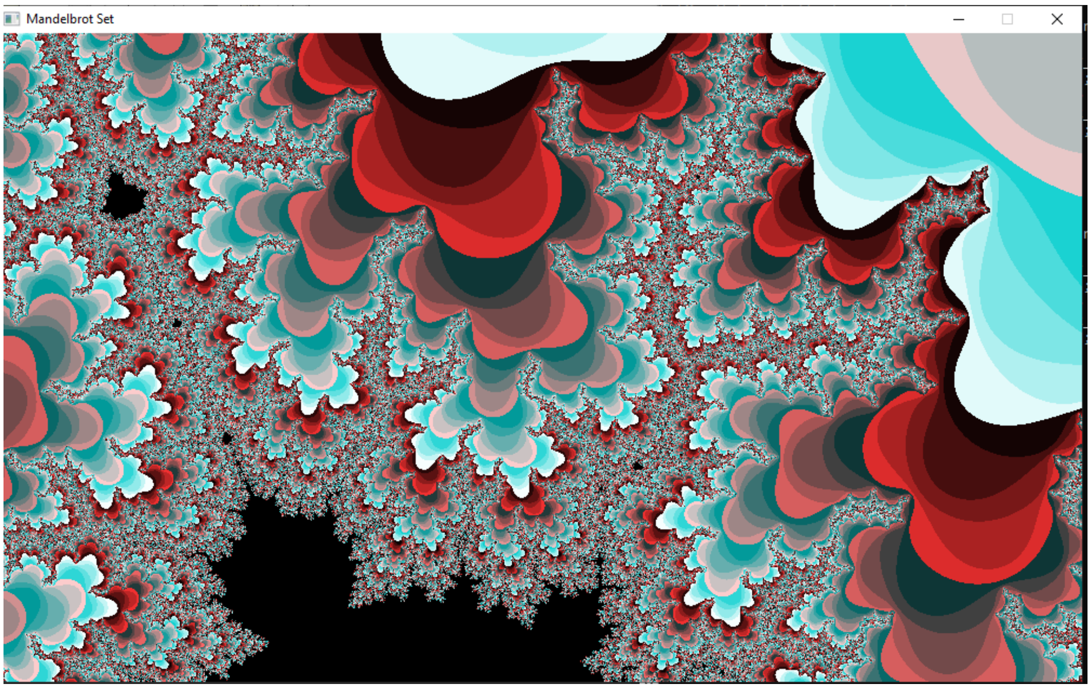
```
double map(double input_value, double input_min, double input_max, double output_min, double
  return output_min + (output_max-output_min)/(input_max-input_min)*(input_value-input_min);
}
```

6 Output






Mandelbrot Set


Mandelbrot Set