Mandelbrot Set

Ashvin Oli

December 18, 2020

Contents

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

$\mathrm{X} = (x - \frac{width}{2}) \times \frac{4}{width}$
$\mathrm{Y} = (y - \frac{height}{2}) \times \frac{4}{width}$
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:
$\mathrm{X} = output_{xmin} + \frac{output_{xmax} - output_{xmin}}{input_{xmax} - input_{xmin}} \times (x - input_{xmin})$
$\mathrm{Y} = output_{ymin} + \frac{output_{ymax} - output_{ymin}}{input_{ymax} - input_{ymin}} \times (y - input_{ymin})$

2.2.1 Very important note on scaling

Applying above formula with different $output_{xmax}$ and $output_{ymax}$ distorts the image. So we need to make sure that x axis and y axis are equally scalled, while using the linear interpolation. So, the scale factor which is

$$\frac{output_{max} - output_{min}}{input_{max} - input_{min}}$$

has to be same for both. I have chosen scalling with respect to HEIGHT as height is less in my case, and this exact scaling has been applied to x-axis. The only important thing to notice is minimum value of x, which has been set that the entire width is in same scale to height, and by taking the half of the scaled value and making it negative, minimum x has been set, and likewise the max value is the positive half. The only time both max and min x will be utilized is when we are zooming. While zooming infinitely to the mouse pointer what I have done is: First center the point under the mouse pointer, and zoom in to the center infinitely.

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 Key Bindings

Use W,S,A,D to move. SPACE to zoom in. Srolling down and pressing SPACE do the same thing. Press F to zoom in forever in the point under the current mouse pointer, and G to stop zooming forever. To center the point right under the mouse pointer press C.

6 Code

6.1 Headers and function initialization

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


#define WIDTH 1000.0
#define HEIGHT 600.0
int MAX_ITER= 50;

double out_max_x;
double out_min_x;
double out_max_y= 2;
double out_min_y=-2;
int zoom_forever = 0;


int draw(SDL_Renderer **,int);
double map(double,double ,double, double, double);
void change_viewport_wrt_mouse(int,int,float,float);
int handle_key_presses(int,float,float,int,int);
int main(int argc, char *argv[])
{
 out_min_x = -2 * WIDTH/HEIGHT;
 out_max_x = 2 * WIDTH/HEIGHT;
 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 zoom level
  float zoom = 1;
  //Default value of to_render is true and is set true again when the user does some action
  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;
  // offsets to zoom in or out or move image sidewise
  float offset_x,offset_y;
  while (!quit){
    offset_x = (out_max_x - out_min_x);
    offset_y = (out_max_y - out_min_y);
    while (SDL_PollEvent(&event))
        {
        SDL_GetMouseState(&mouse_x,&mouse_y);
         switch (event.type) {
         case SDL_QUIT:
           quit = 1;
           break;
         case SDL_MOUSEWHEEL:
           if(event.wheel.y > 0)
             // scroll down
             {
               printf("\r%-100s","Zooming in on mouse pointer. Wait for image to render!");
               fflush(stdout);
               offset_x /= 4;
               offset_y /= 4;
               MAX_ITER += 20;
             }else if (event.wheel.y < 0)
             // scroll up
             {
               printf("\r%-100s","Zooming out. Wait for image to render!");
               fflush(stdout);
               offset_x *=2;
               offset_y *=2;
               if (MAX_ITER >= 50) {
                 MAX_ITER -= 10;
               }

             }
             change_viewport_wrt_mouse(mouse_x,mouse_y,offset_x,offset_y);
             SDL_SetRenderDrawColor(renderer, 255, 255, 255, SDL_ALPHA_OPAQUE);
             SDL_RenderClear(renderer);
             to_draw = 1;
             break;
         case SDL_KEYDOWN:
           //if only designated keys are pressed than draw
             if (handle_key_presses(event.key.keysym.sym,offset_x,offset_y,mouse_x,mouse_y)
               SDL_SetRenderDrawColor(renderer, 255, 255, 255, SDL_ALPHA_OPAQUE);
               SDL_RenderClear(renderer);
               to_draw = 1;
             }
             break;
         }

        }

    if (zoom_forever) {
      //Decreasing and increasing values by certain Percenatage of the offsets for unifom sca
      //And preveting the values to get reversed in sign.
        out_min_y += offset_y*zoom*0.20;
        out_max_y -= offset_y*zoom*0.20;
        out_min_x += offset_x*zoom*0.20;
        out_max_x -= offset_x*zoom*0.20;
        zoom *= 0.95;
        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%-100s","Image Rendered! You may now zoom or pan.");
          fflush(stdout);
        }
    }

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

6.2 Main logic

```
int draw(SDL_Renderer **renderer,int factor){
   for (int x = 0; x < WIDTH; x++) {
      for (int y =0;  y < HEIGHT; y++) {
      // Mapping the screen with the limits.
      // Same scaling has been made. This causes a slight problem. x and y might gain values
      // prevents distortion. And this is also the reason the image is not centered at the be
      // But the origin is to the left of screen.
        double smaller = WIDTH > HEIGHT ? HEIGHT:WIDTH;
        double c_real = out_min_x + (out_max_y-out_min_y)/(HEIGHT)*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;
}
```

6.3 Change Viewport Wrt Mouse position

```
void change_viewport_wrt_mouse(int mouse_x,int mouse_y,float offset_x, float offset_y){
  double smaller = WIDTH > HEIGHT ? HEIGHT:WIDTH;
  double mouse_x_mapped = out_min_x + (out_max_y-out_min_y)/(HEIGHT)*mouse_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;

}
```

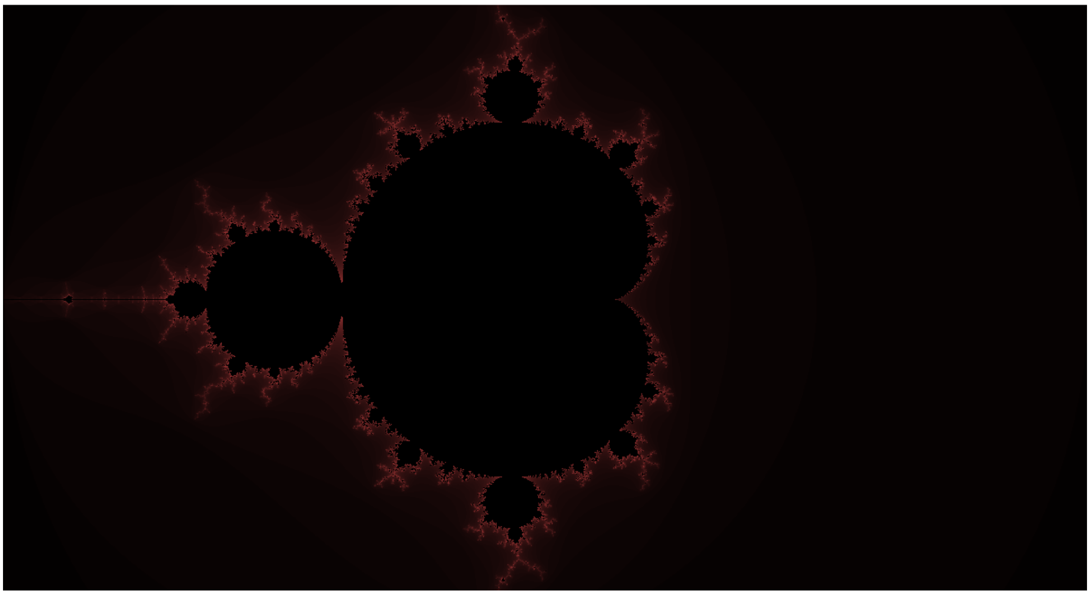
6.4 Handle key presses

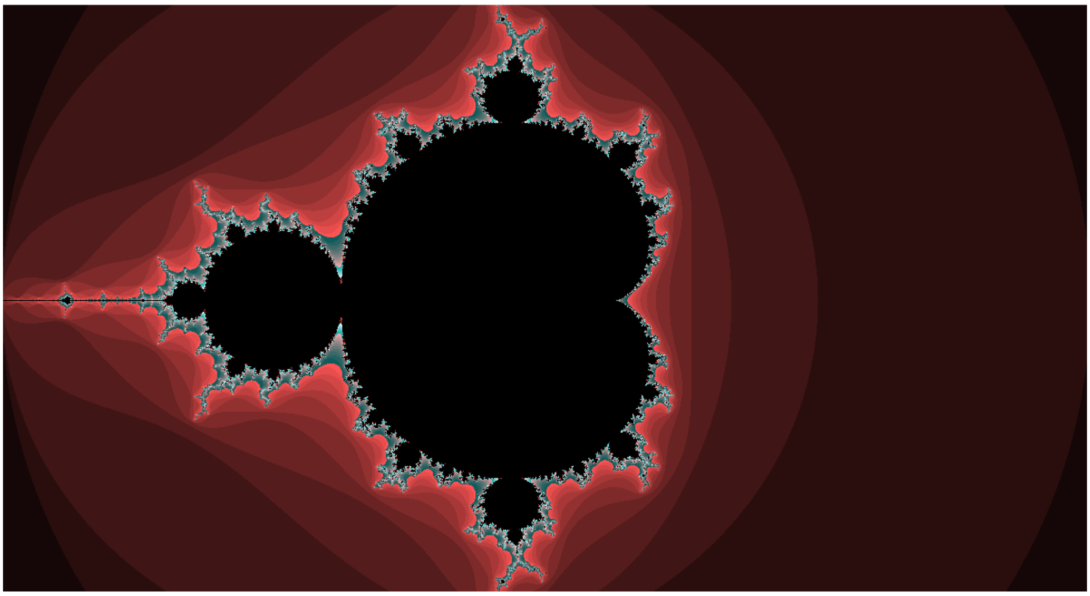
```
int handle_key_presses(int keycode,float offset_x, float offset_y,int mouse_x,int mouse_y){
   switch (keycode)
     {
     case SDLK_w:
       //Move up
       //Since y axis is inverted subtracting will take us to upper part of screen
       printf("\r%-100s","Moving up. Wait for image to render!");
       out_min_y -= offset_y/4;
       out_max_y -= offset_y/4;
       break;
     case SDLK_s:
       //Move down
       printf("\r%-100s","Moving down. Wait for image to render!");
       out_min_y += offset_y/4;
       out_max_y += offset_y/4;
       break;
     case SDLK_a:
       //Move left
       printf("\r%-100s","Moving Left. Wait for image to render!");
       out_min_x -= offset_x/4;
       out_max_x -= offset_x/4;
       break;
     case SDLK_d:
       //Move right
       printf("\r%-100s","Moving Right. Wait for image to render!");
       out_min_x += offset_x/4;
       out_max_x += offset_x/4;
       break;
     case SDLK_SPACE:
       //Zoom in
       printf("\r%-100s","Zooming in on mouse pointer. Wait for image to render!");
       change_viewport_wrt_mouse(mouse_x,mouse_y,offset_x/4,offset_y/4);
       break;
     case SDLK_f:
       //Zoom forever
       printf("\r%-100s","Zooming forever on first mouse pointer location. Wait for image to
       zoom_forever = 1;
       //Center the point under mouse pointer.
       change_viewport_wrt_mouse(mouse_x,mouse_y,offset_x/2,offset_y/2);
       break;
     case SDLK_g:
       //Stop Zoom forever
       printf("\r%-100s","Zooming forever stopped!");
       zoom_forever = 0;
       break;
     case SDLK_c:
       //Center the point under the mouse pointer.
       printf("\r%-100s","Centering the point under mouse pointer. Wait for image to render!"
       //Center the point under mouse pointer.
       change_viewport_wrt_mouse(mouse_x,mouse_y,offset_x/2,offset_y/2);
       break;
     default:
       return 0;
     }
   return 1;
}
```

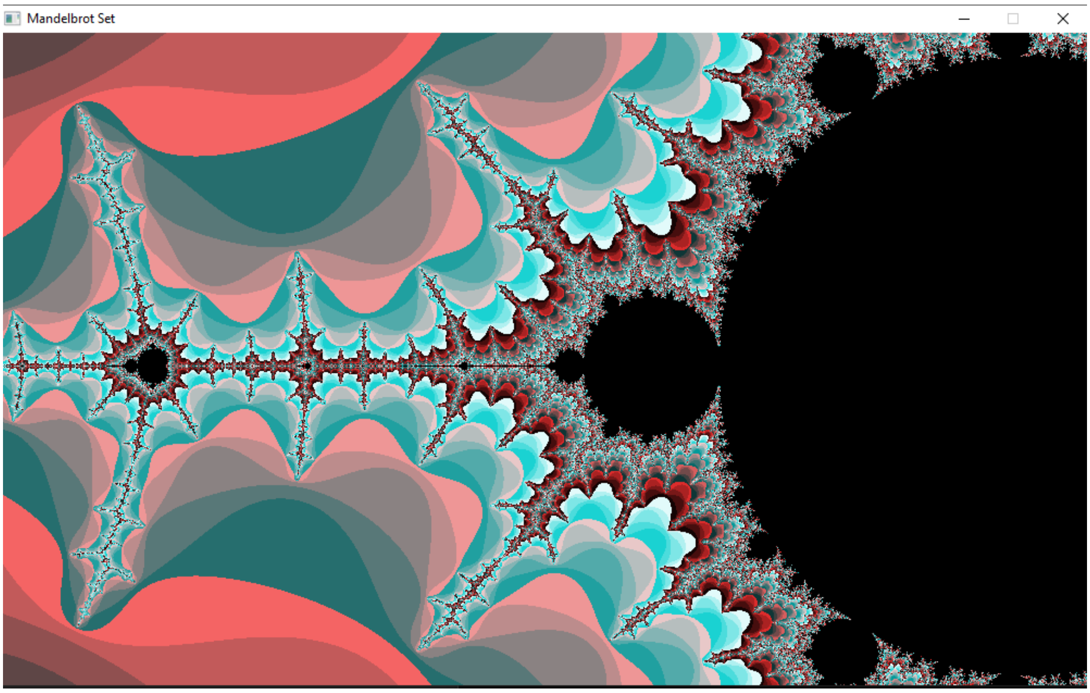
6.5 Map Function

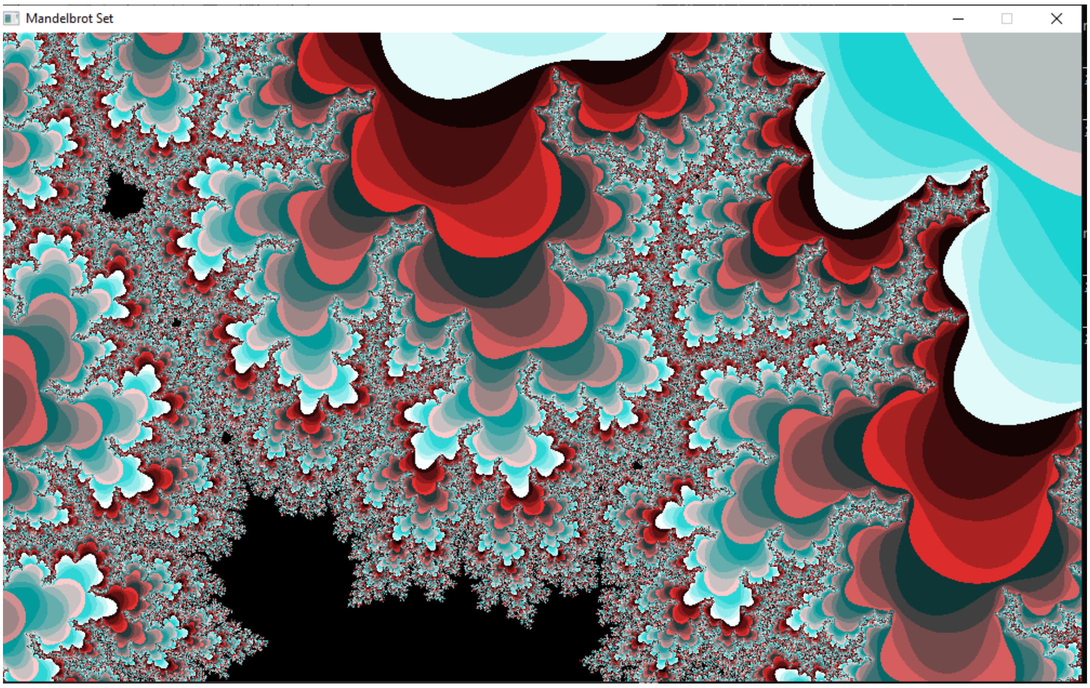
```
double map(double input_value, double input_min, double input_max, double output_min, double
  return output_min + (output_max-output_min)/(input_max-input_min)*(input_value-input_min);
}
```

7 Output






Mandelbrot Set


Mandelbrot Set