

Mandelbrot Set

Ashvin Oli

December 18, 2020

Contents

1	Introduction	2
2	Implementation Details	2
2.1	Transformation Simple	2
2.2	Linear Mapping	2
2.2.1	Very important note on scaling	2
2.3	A bit of Complex algebra	3
2.4	Zooming in	3
3	Setting up SDL2	3
4	Building	3
5	Key Bindings	3
6	Code	3
6.1	Headers and function initialization	3
6.2	Main logic	6
6.3	Change Viewport Wrt Mouse position	7
6.4	Handle key presses	7
6.5	Paint white and allow drawing	8
6.6	Map Function	8
7	Output	9

1 Introduction

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

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

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

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-coordinate 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:

$$\begin{aligned} |c| > 2 &\implies |z_1| > 2 \\ &\implies |c| \leq 2 \end{aligned}$$

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

$$\begin{aligned} X &= (x - \frac{width}{2}) \times \frac{4}{width} \\ Y &= (y - \frac{height}{2}) \times \frac{4}{width} \end{aligned}$$

Note: I have scaled 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:

$$\begin{aligned} X &= output_{xmin} + \frac{output_{xmax} - output_{xmin}}{input_{xmax} - input_{xmin}} \times (x - input_{xmin}) \\ Y &= output_{ymin} + \frac{output_{ymax} - output_{ymin}}{input_{ymax} - input_{ymin}} \times (y - input_{ymin}) \end{aligned}$$

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 scaled 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 scaling 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

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

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 output_{max} and output_{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:

```
1 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

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <SDL2/SDL.h>
5
6
7 #define WIDTH 1000.0
8 #define HEIGHT 600.0
9 int MAX_ITER= 50;
10
11 double out_max_x;
12 double out_min_x;
13 double out_max_y= 2;
14 double out_min_y=-2;
15 int zoom_forever = 0;
```

16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66

```
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);
void white_paint_and_draw(SDL_Renderer**, 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_CENTERED,
    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 does some action
    int to_draw;

    //Clear using white color before going inside the loop and set to_draw to 1
    white_paint_and_draw(&renderer,&to_draw);

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

```

67 // offsets to zoom in or out or move image sidewise
68 float offset_x,offset_y;
69 while (!quit){
70     offset_x = (out_max_x - out_min_x);
71     offset_y = (out_max_y - out_min_y);
72     while (SDL_PollEvent(&event))
73     {
74         SDL_GetMouseState(&mouse_x,&mouse_y);
75         switch (event.type) {
76             case SDL_QUIT:
77                 quit = 1;
78                 break;
79             case SDL_MOUSEWHEEL:
80                 if(event.wheel.y > 0)
81                     // scroll down
82                 {
83                     printf("\r%-100s","Zooming in on mouse pointer. Wait for image to render!");
84                     fflush(stdout);
85                     offset_x /= 4;
86                     offset_y /= 4;
87                     MAX_ITER += 20;
88                 }else if (event.wheel.y < 0)
89                     // scroll up
90                 {
91                     printf("\r%-100s","Zooming out. Wait for image to render!");
92                     fflush(stdout);
93                     offset_x *=2;
94                     offset_y *=2;
95                     if (MAX_ITER >= 50) {
96                         MAX_ITER -= 10;
97                     }
98                 }
99             }
100         change_viewport_wrt_mouse(mouse_x,mouse_y,offset_x,offset_y);
101         white_paint_and_draw(&renderer,&to_draw);
102         break;
103     case SDL_KEYDOWN:
104         //if only designated keys are pressed than draw
105         if (handle_key_presses(event.key.keysym.sym,offset_x,offset_y,mouse_x,mouse_y))
106             white_paint_and_draw(&renderer,&to_draw);
107         }
108         break;
109     }
110 }
111
112
113 if (zoom_forever) {
114     //Decreasing and increasing values by certain Percenatage of the offsets for unifom sca
115     //And preveting the values to get reversed in sign.
116     out_min_y += offset_y*0.20;
117     out_max_y -= offset_y*0.20;

```

```

118         out_min_x += offset_x*0.20;
119         out_max_x -= offset_x*0.20;
120         SDL_SetRenderDrawColor(renderer, 255, 255, 255, SDL_ALPHA_OPAQUE);
121         SDL_RenderClear(renderer);
122         to_draw = 1;
123
124     }
125
126
127     //Draw pixels on the renderer
128     if (to_draw) {
129         to_draw = draw(&renderer, factor);
130         SDL_RenderPresent(renderer);
131         printf("\r%-100s", "Image Rendered! You may now zoom or pan.");
132         fflush(stdout);
133     }
134 }
135
136 //free resources
137 SDL_DestroyRenderer(renderer);
138 SDL_DestroyWindow(window);
139 SDL_Quit();
140 return 0;
141 }

```

6.2 Main logic

```

1  int draw(SDL_Renderer **renderer, int factor){
2      for (int x = 0; x < WIDTH; x++) {
3          for (int y = 0; y < HEIGHT; y++) {
4              // Mapping the screen with the limits.
5              // Same scaling has been done such that the image is centered on the screen.
6              double c_real = out_min_x + (out_max_y-out_min_y)/(HEIGHT)*x;
7              double c_img = map(y,0,HEIGHT, out_min_y,out_max_y);
8
9              double z_real = 0;
10             double z_img = 0;
11             int iter_count = 0;
12             while (pow(z_real,2)+pow(z_img,2) <= 4 && iter_count < MAX_ITER) {
13                 double temp_real = pow(z_real,2)-pow(z_img,2)+c_real;
14                 double temp_img = 2*z_real*z_img + c_img;
15                 z_real = temp_real;
16                 z_img = temp_img;
17                 iter_count++;
18             }
19
20             //If any number exits before reaching MAX_ITER then, it is not in the set. So colour
21             if (iter_count == MAX_ITER) {
22                 //printf("SELECT %.2f %.2f %d %d\n", c_real, c_img, x, y);
23                 //Draw with black

```

```

24         SDL_SetRenderDrawColor(*renderer, 0,0, 0, SDL_ALPHA_OPAQUE);
25         SDL_RenderDrawPoint(*renderer,x,y);
26     }else{
27         //Draw with custom shade
28         SDL_SetRenderDrawColor(*renderer, iter_count*factor*5,iter_count*factor, iter_count
29         SDL_RenderDrawPoint(*renderer,x,y);
30     }
31 }
32 }
33 return 0;
34 }

```

6.3 Change Viewport Wrt Mouse position

```

1 void change_viewport_wrt_mouse(int mouse_x,int mouse_y,float offset_x, float offset_y){
2     double mouse_x_mapped = out_min_x + (out_max_y-out_min_y)/(HEIGHT)*mouse_x;
3     double mouse_y_mapped = map(mouse_y,0,HEIGHT, out_min_y,out_max_y);
4     out_min_x = mouse_x_mapped - offset_x;
5     out_max_x = mouse_x_mapped + offset_x;
6     out_min_y = mouse_y_mapped - offset_y;
7     out_max_y = mouse_y_mapped + offset_y;
8
9 }

```

6.4 Handle key presses

```

1 int handle_key_presses(int keycode,float offset_x, float offset_y,int mouse_x,int mouse_y){
2     switch (keycode)
3     {
4         case SDLK_w:
5             //Move up
6             //Since y axis is inverted subtracting will take us to upper part of screen
7             printf("\r%-100s","Moving up. Wait for image to render!");
8             out_min_y -= offset_y/4;
9             out_max_y -= offset_y/4;
10            break;
11        case SDLK_s:
12            //Move down
13            printf("\r%-100s","Moving down. Wait for image to render!");
14            out_min_y += offset_y/4;
15            out_max_y += offset_y/4;
16            break;
17        case SDLK_a:
18            //Move left
19            printf("\r%-100s","Moving Left. Wait for image to render!");
20            out_min_x -= offset_x/4;
21            out_max_x -= offset_x/4;
22            break;
23        case SDLK_d:
24            //Move right

```

```

25     printf("\r%-100s", "Moving Right. Wait for image to render!");
26     out_min_x += offset_x/4;
27     out_max_x += offset_x/4;
28     break;
29 case SDLK_SPACE:
30     //Zoom in
31     printf("\r%-100s", "Zooming in on mouse pointer. Wait for image to render!");
32     change_viewport_wrt_mouse(mouse_x, mouse_y, offset_x/4, offset_y/4);
33     break;
34 case SDLK_f:
35     //Zoom forever
36     printf("\r%-100s", "Zooming forever on first mouse pointer location. Wait for image to");
37     zoom_forever = 1;
38     //Center the point under mouse pointer.
39     change_viewport_wrt_mouse(mouse_x, mouse_y, offset_x/2, offset_y/2);
40     break;
41 case SDLK_g:
42     //Stop Zoom forever
43     printf("\r%-100s", "Zooming forever stopped!");
44     zoom_forever = 0;
45     break;
46 case SDLK_c:
47     //Center the point under the mouse pointer.
48     printf("\r%-100s", "Centering the point under mouse pointer. Wait for image to render!");
49     change_viewport_wrt_mouse(mouse_x, mouse_y, offset_x/2, offset_y/2);
50     break;
51 default:
52     return 0;
53 }
54 return 1;
55 }

```

6.5 Paint white and allow drawing

```

1 void white_paint_and_draw(SDL_Renderer **renderer, int *to_draw){
2     SDL_SetRenderDrawColor(*renderer, 255, 255, 255, SDL_ALPHA_OPAQUE);
3     SDL_RenderClear(*renderer);
4     *to_draw = 1;
5 }

```

6.6 Map Function

```

1 double map(double input_value, double input_min, double input_max, double output_min, double
2     return output_min + (output_max-output_min)/(input_max-input_min)*(input_value-input_min);
3 }

```

7 Output



