CSE 167 (FA 2022) Homework 1 – Due 10/5

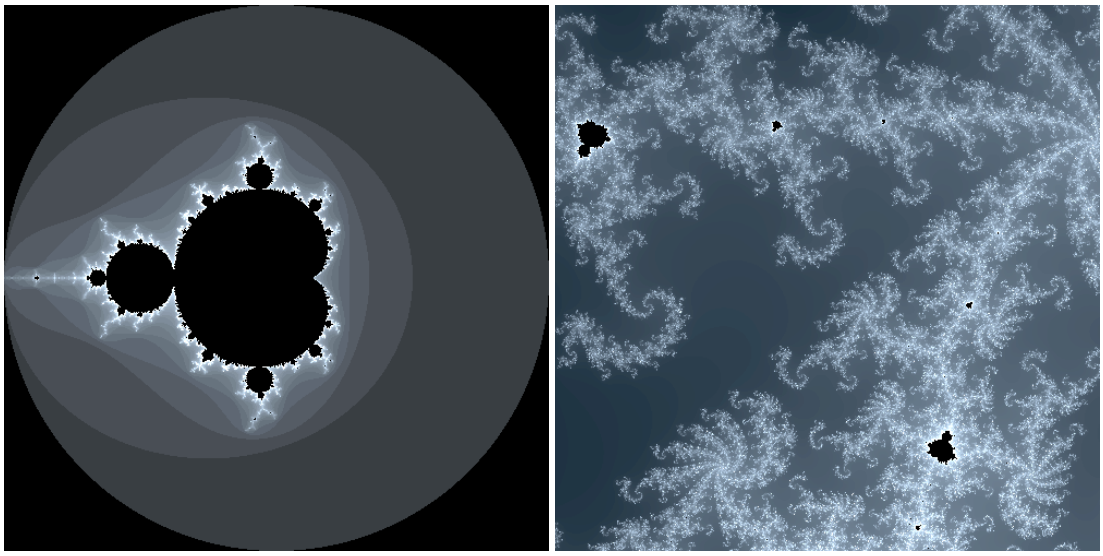

Figure 1 The Mandelbrot fractal as a whole (left); a zoomed-in shot of the fractal (right).

In this homework, we will write your own fragment shader for a cool mathematical visualization. Specifically, we will render the **Mandelbrot set**.[1][2]

The Mandelbrot set is a set in the complex plane. It can be defined with a very simple mathematical procedure, but the resulting picture of the set is visually stunning.

Mandelbrot set is a **fractal**. For a fractal, you can endlessly zoom into the picture and find a similar copy of the picture at a larger scale. For Mandelbrot set, the zoomed in versions of the picture are all slightly different depending on the scale and depending on where you zoom in. You can find infinitely many beautiful patterns by exploring different parts of the Mandelbrot set.

The Mandelbrot set is defined by the following procedure. Consider the iteration

Input: $c \in \mathbb{C}$;
1: $z_0 = 0 \in \mathbb{C}$;
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: $\quad z_{k+1} = z_k^2 + c$;
4: **end for**

For each fixed complex number c, we have a simple nonlinear iteration (the $(\cdot)^2$ in Line 3 makes it nonlinear). It turns out that for such a simple iteration, it is very difficult to predict where the iterant z_k is going. To demonstrate this quality, we visualize the behavior of the iteration depending on c. For c ranging over a square region in the complex plane, we generate a square picture, one value of c per pixel.

A standard aspect of the iteration behavior to quantify is "whether the iteration blows up." If c has a big magnitude, say $c = 10\mathbb{i}$, then we can see that the iterant seems to grow unboundedly. If c has a small magnitude, say $c = 0.1$, then the iteration stays bounded.

[1] Wikipedia reference: https://en.wikipedia.org/wiki/Mandelbrot_set
[2] Watch this video with excellent introduction: https://www.youtube.com/watch?v=NGMRB4O922I

Definition 1 The **Mandelbrot set** is defined by

$$M = \{c \in \mathbb{C} \mid \text{the above iteration stays bounded.}\} \tag{1}$$

If we observe $|z_k| \geq 2$ during the iteration, then the iteration is guaranteed to blow up. The circle centered at the origin with radius 2 is the circle of no return.

So, to make the visualization of the Mandelbrot set more informative, we color the pixel according to the number of iteration k it takes for $|z_k|$ to exceed 2. If the pixel is in the Mandelbrot set (in which case the iterant never has a norm exceeding 2) then we just color it with a default color.

In other words, we define a function

$$I : \mathbb{C} \to \{0\} \cup \mathbb{N} \tag{2}$$

with So $I(c)$ is 0 if c is in the Mandelbrot set (checked up to the max iteration number $n_{\max}$) and

Pseudocode for Mandelbrot shader

Input: $c \in \mathbb{C}, n_{\max} \in \mathbb{N}$;
1: $I(c) \leftarrow 0$;
2: $z \leftarrow 0 \in \mathbb{C}$;
3: **for** $k = 0, 1, 2, \ldots, n_{\max} - 1$ **do**
4: $\quad z \leftarrow z_k^2 + c$;
5: $\quad$ **if** $|z| > 2$ **then**
6: $\quad\quad I(c) \leftarrow k$;
7: $\quad\quad$ **break**;
8: $\quad$ **end if**
9: **end for**

otherwise $I(c)$ is the number of iterations it takes for z to pass the the circle of no return.

Programming 1.1 — 15 pts. Download HW1 skeleton code. You will only be modifying the file

`haders/Mandelbrot.frag`

```
#version 330 core

in vec2 canvas_coord;

uniform vec2 center;
uniform float zoom;
uniform int maxiter;

out vec4 fragColor;

// HW1: You can define customized functions here,
// e.g. complex multiplications, helper functions
// for colormap etc.


void main (void){
```

```
    vec2 c = center + zoom * canvas_coord;
    // HW1: Your implementation goes here. Compute
    // the value of the Mandelbrot fractal at
    // complex number c.  Then map the value to
    // some color.


    // HW1: Replace the following default color
    fragColor = vec4(0.5,0.5,0.5, 1.0f);
}
```

The parameter interface of this shader is already been implemented. The fragment's position relative to the square geometry is passed on from the vertex shader as `canvas_coord`. This coordinate is converted into the complex number c using a couple of uniform variables describing the center and the zoom. Specifically, each position $P \in [-1,1] \times [-1,1]$ corresponds to a complex number c by

$$c = \texttt{center} + \texttt{zoom} \cdot P \tag{3}$$

where the 2D vector (complex number) $\texttt{center} \in \mathbb{C}$ and the positive scalar $\texttt{zoom} \in \mathbb{R}_{>0}$ are uniform variables. Note that there is also the max iteration $n_{\max}$ as a uniform variable. You can control the center, zoom, and max iteration with keyboards.

Compilation

Compile the code skeleton with a similar process as HW0. If you needed to modify certain lines in HW0 to let things to run on your machine, then go ahead to do so for the main cpp files.

There are two projects within the code skeleton. One is `HelloSquare2` that should reproduce the same figure as HW0. `HelloSquare2.cpp` is an example of an object-oriented program contrasting the code from HW0. The main code for this HW1 is `Mandelbrot.cpp`. Running the executable `Mandelbrot` should produce a gray window.

Implementation

Implement the pseudocode above that evaluates $I(c)$. Use `c` as c and `maxiter` as $n_{\max}$. You can use `for` loops and `if` conditions similar to a C program.

You may want to write a customized function `vec2 cprod(const vec2 z1, const vec2 z2)` before the `void main(void)` in the fragment shader for the complex number multiplications.

Design a color function

Assign the fragment color according to the value of $I(c)$. You can design your own functions that would map the value of $I(c)$ (perhaps casted to float and divided by max iteration) to each channel of the color.

Uploads

1. Take a screenshot when all the parameters are at their default values (`center` $= (0,0)$ (*i.e.* $0 + 0\mathbb{i}$) and `zoom` $= 2$) so that we see the entire Mandelbrot set.
2. Zoom in and navigate to a favorite location and take a screenshot.
3. Upload your fragment shader `Mandelbrot.frag`.