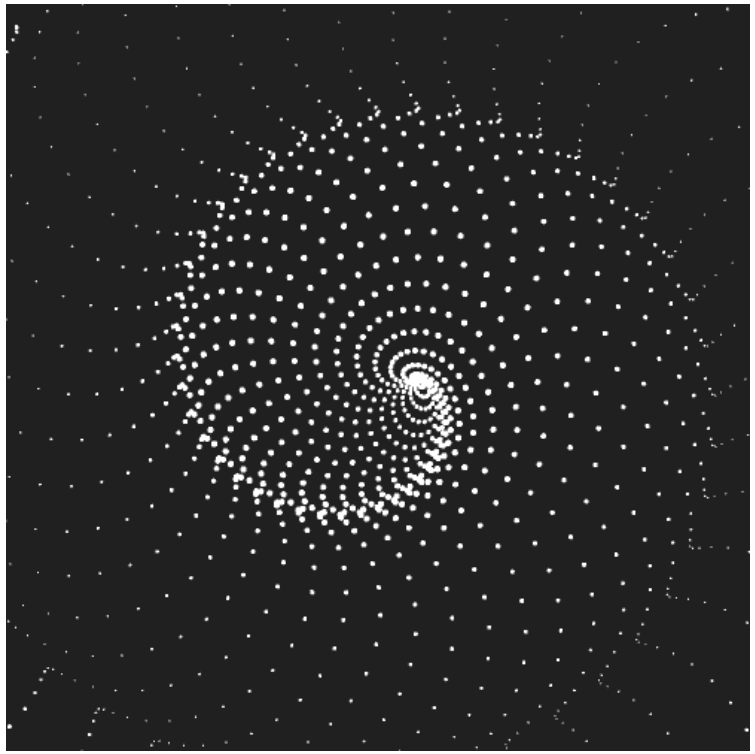
Mandelbrot Set

Paleti Krishnasai

CED18I039

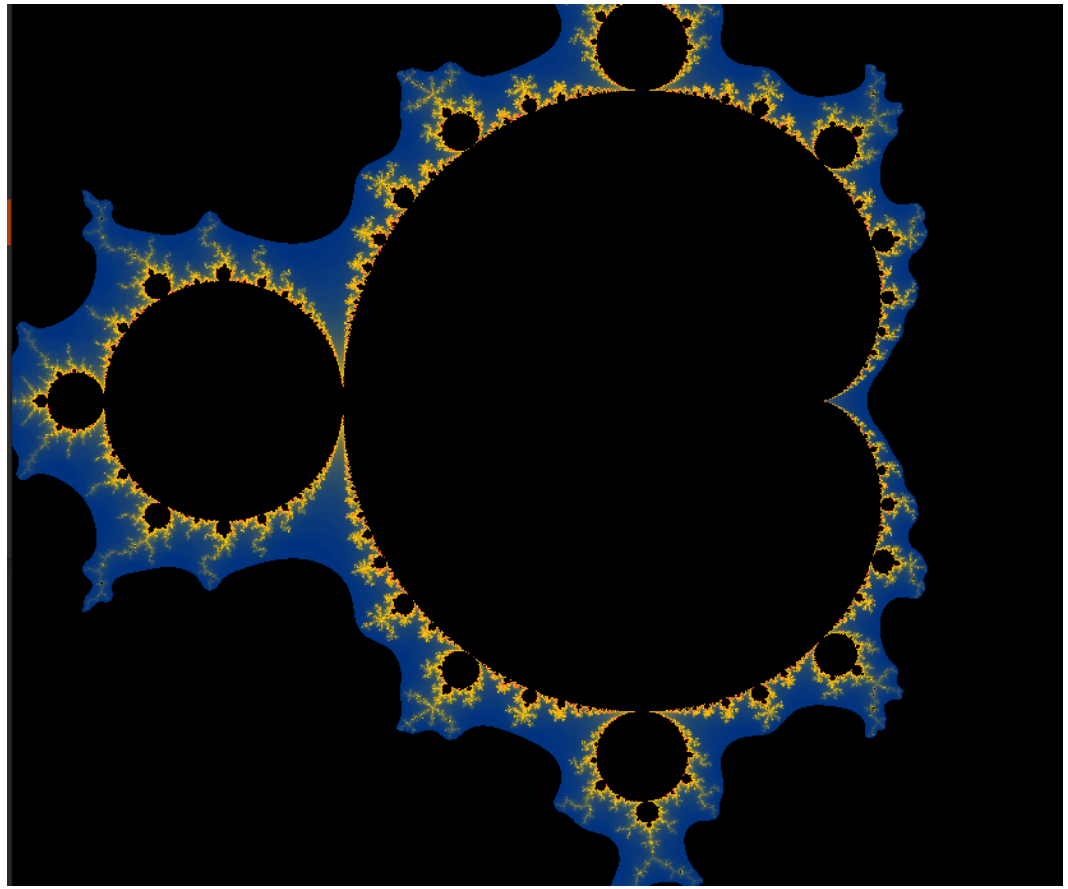
Fractals
[look closely]

- An Iterative pattern that displays some level of self-similarity.
- The simple answer is that a fractal is a shape that, when you look at a small part of it, has a similar (but not necessarily identical) appearance to the full shape. Take, for example, a rocky mountain. From a distance, you can see how rocky it is; up close, the surface is very similar. Little rocks have a similar bumpy surface to big rocks and to the overall mountain.
- Finite area … Infinite Perimeter!!



Mandelbrot Set

- The Father of fractal geometry.
- It is a set of points in the complex plane, the boundary of which forms a fractal.
- Quadratic recurrence equation $Z_{n+1} = Z_n^2 + c$, c is a complex number
- Stability over iteration of Zero.
- Take a complex number c. Feed it to the function $f(z) = z^2 + c$, where the initial value of $z=0$. Take the result of this (let's call it z_1) -and plug it back into the function. Repeat with z2, z3,...ad infinitum. Is the modulus of $z\infty$, its distance from the origin of the complex plane, below 2? If yes, then c is in the Mandelbrot set!



- To give an example, let's choose $c = i$.
- $z_1 = 0^2+i = i$
- $z_2 = z_1^2 + i = -1+i$
- $z_3 = z_2^2 + i = -i$
- $z_4 = z_3^2 + i = 1+i$
- $z_5 = z_4^2+i = 3i$
- .
- .
- .
- *So on and so forth, here we can see that modulus of z_5 is 3, which >2 , hence not in Mandelbrot set.*

Algorithm

- 1.) Calculate $z_{n+1}=z_n^2+c$
- 2.) If $|z_{n+1}| > 2$ then stop and return n.
- 3.) If $|z_{n+1}| <= 2$ go to *1*. and repeat .
- 4.) We must implement some sort of maximum number of iterations above which we interrupt the algorithm and assume the number is in the set. If we don't, we simply keep iterating forever!

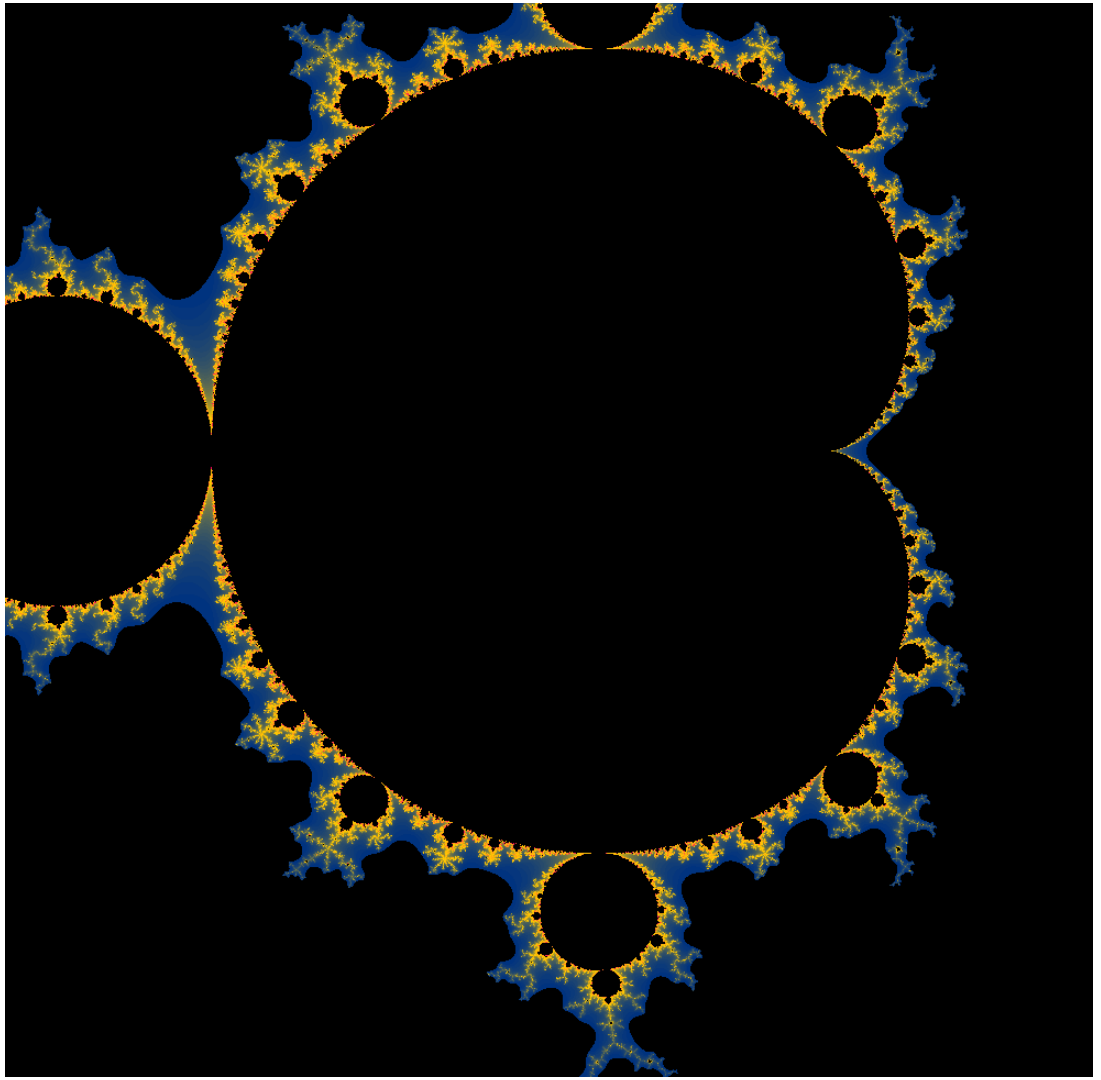
Implementation

- Create a simple array of floats vertices that will represent the coordinates of the four corners of the rectangle filling the screen, and an array of integers indices which indexes the above vertices defining two triangles.
- implement two quick functions. One will allow us to resize the window. The other one measures the time it takes to render a frame. Each render loop we increment the frame count, and each second, we print out the time it took to render one frame on average, in milliseconds.
- In the main() function we first set up a window in preparation for our rendering

- Once a window is set up, we make the data representing our rectangle available to the GPU. This is done via vertex array objects (VAOs), which in turn rely on vertex buffer objects (VBOs) and element buffer objects (EBOs).
- A VAO is created – it represents all data related to the vertices to be drawn. Instead of passing around custom structs or large arrays, we simply refer to different VAOs when we want to draw the corresponding vertices. We copy the coordinates of these vertices into a buffer: the VBO.
- Set up shaders. The corresponding class Shader is implemented in the files “Shader.h”
- 1) erase whatever was shown last frame, 2) update the FPS counter, 3) use shader 4) render the rectangle that represents the background, on which the shader rasterizes the mandelbrot set.

Shaders

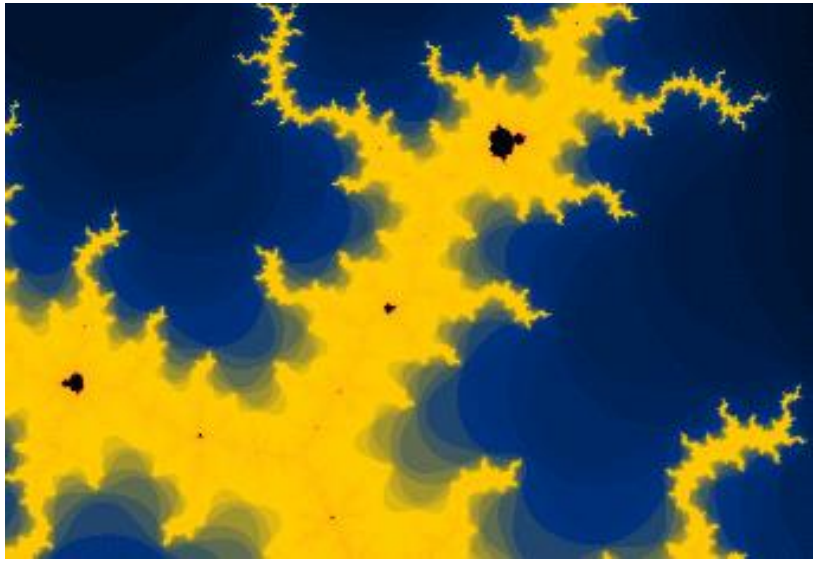
- Shaders are files written in a C-like language that get executed by the graphical processing unit. OpenGL uses a pipeline of several shader programs, each of which takes care of a different aspect of rendering the desired scene.
- **The vertex shader** is the first to be executed. It operates on each vertex. Vertex shaders can move vertices around or change their properties. We only render two triangles that form a rectangle filling the view.
- **The fragment shader** is executed for each fragment (or pixel) on the scene and returns the color of that pixel.
- Use the position of the fragment in the rendered view as the coordinates in complex space that we will feed to our Mandelbrot algorithm. The fragment shader starts with telling the GPU that we want to take the position of the fragment as an input and output the final color value. Then, we define a function that returns the number of iterations of the Mandelbrot set algorithm.

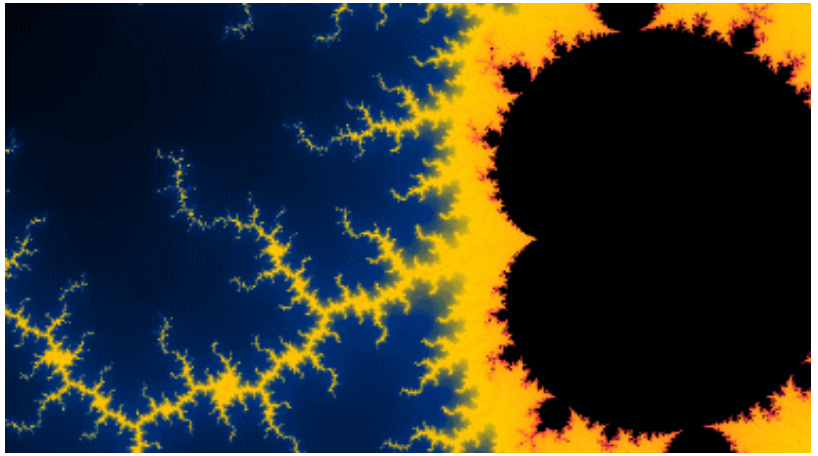


Each of the tiny black portions over the yellow is the inceptive effect of the mandelbrot set.

Controls

- implement a simple control functionality that allows the viewers to zoom in and out, as well as to move around the complex number plane. The actual implementation might seem counter-intuitive at first. Instead of moving the camera, it is usually the world itself that is moved around with respect to a static camera.
- The two triangles acting as a backdrop for our fractal will remain static. What we will change instead is the subset, the rectangle, of the complex number plane, that we feed to our Mandelbrot algorithm. For example, if we want to move right, we simply increase the real part of the value evaluated in the algorithm.
- LEFT_CTRL = zoom in
- LEFT_SHIFT = zoom out
- Arrow keys to maneuver





Zoomed in output of the code
(time to compute : 45 mins (approx))

Instructions To Execute The Program

(Code only tested on linux – ubuntu 18.04,20.04)

- The Code folder named CED18I039 contains a subfolder (Shaders), a main.cpp file and a Shader.h header file.
- Ensure CED18I039 is the working directory.
- Execute the below command in the terminal of the working directory.

  g++ main.cpp -o mandel -lGL -lGLU -lglut -lGLEW –lglfw

- The Code is extremely computationaly intensive and is heavy on the GPU usage. Each zoom in iteration takes a lot of time to compute with the differing range of colors.

FIN