Mandelbrot Zooming

# Name: Sanchit Jindal

# Roll number: 200020120

I have implemented a Mandelbrot zooming animation for my CS293 project

* I am using SFML (Simple and Fast Multimedia Library) to produce the graphics
* I am also using the SFML boiler plate to compile the files which is provided at <https://github.com/andrew-r-king/sfml-vscode-boilerplate>
* To run the program, you need to have a SFML downloaded to a path where header files are checked, The SFML source code can be found at <https://www.sfml-dev.org/download/sfml/2.5.1/>
* You can simply build the file using build function of VS code
* The code is provided in main.cpp in “src”

# Implementation

1. The Naïve approach

For the most basic implementation for rendering a Mandelbrot set is to iterate over every pixel on the screen, calculating its escape time using the Mandelbrot function and colouring the pixel depending on the escape time

This method is not feasible to provide a good visualization as the process takes too long and is not able to build a simulation of Mandelbrot zoom

The worst case running time for this algorithm is O(w\*h\*i )

w is width of screen

h is height of screen

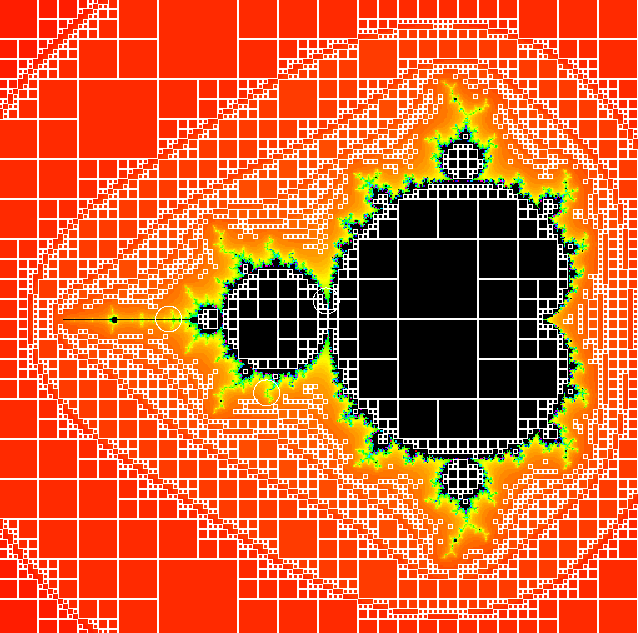
I is the maximum number of iterations allowed to the Mandelbrot function

1. Using Quad Trees

The process of rendering Mandelbrot sets can be made more manageable with the use of quad trees

Quad trees work as follows

* We define an area on the screen and check some pixels in the area for the escape times
* If all the pixels have the same escape time then we can assume that the whole area will have the same escape time pixels and hence can be filled by a single rectangle
* If all the selected pixels don’t have the same time then we divide the area into small parts and try again



As can be seen in the image the bigger square which cover a larger area  
are filled with a single rectangle instead of calculating all the values of all the pixels

This process reduces the processing time as it allows a large space to be processed by calculating the processing time of few pixels only and more detail is given to the boundary of the Mandelbrot set which is where most of the change occurs

We can select the pixels to be processed in an area in different ways

Two of them are:

* Selecting the corners, midpoint of edges and the centre of the area, this takes almost negligible pixels to process but the Mandelbrot set can become less sharp especially while zooming
* Selecting all the pixels on the border of the area and the pixel in the centre of the area, this processes much more pixels than the previous case and is thus slower but as the whole border is selected the image is sharper and there is no dithering as the Mandelbrot zooms