**Introduction**  In this assignment we will use the OpenGL graphics API to compute and display a visual image of the *Mandelbrot Set*. Once the set is displayed, we will use the mouse to select a square region from the displayed set, and recompute the set using only the selected region of the complex plane. Details of the math to compute the *Mandelbrot Set* is given in the paragraphs below.

**The Mandelbrot Set**  The *Mandelbrot Set* is defined as the set of points in the complex plane that satisfy

\[
M = \{ c \in \mathbb{C} | \lim_{n \to \infty} Z_n \neq \infty \}
\]

where

- \( M \) is the set of all complex numbers in the *Mandelbrot Set*.
- \( C \) is the set of all complex numbers in the complex plane,
- \( Z_0 = c \),
- \( Z_{n+1} = Z_n^2 + c \).

From this description it appears we have to iterate \( n \) over all values from 0 to \( \infty \), which would take quite a bit of computation. Luckily, we can make two simplifying assumptions.

1. If the magnitude of any \( Z_n \) is greater than 2.0, then it can be proved that \( Z \) will eventually reach infinity, so if we ever get a \( Z_n \) for which the magnitude is greater then 2.0 we can stop iterating and claim that the point \( c \) is *not* in the *Mandelbrot Set*.

2. If we have iterated 2,000 times and still not found a \( Z_n \) with a magnitude greater than 2.0, we can again stop iterating, but this time claim the \( c \) is in the *Mandelbrot Set*. The value 2000 was chosen arbitrarily but seems to work pretty well for this assignment.

**Displaying the Mandelbrot Set**  Start by creating a display window of 512 by 512 pixels. Then compute and display the *Mandelbrot Set*. Unfortunately, it appears in the above discussion of the math to compute the *Mandelbrot Set*, that we have to iterate all possible values of \( c \in C \). Again we are lucky that we can limit the possible range of the \( c \) value as follows:

\((-2.0, -1.2) \times (1.0, 1.8)\)

This still seems to be an infinite number of possible \( c \) values. To again avoid infinite processing, simply select 512 discrete \( c \) values in both the real and imaginary ranges above, resulting in 512 * 512 total unique \( c \) values. For each \( c \), simply iterate the \( Z \) values as shown above. It the \( c \) is in the *Mandelbrot Set*, display a black pixel at the appropriate point in the display window. If the point is *not* in the *Mandelbrot Set* (ie you found a \( Z_n > 2.0 \)), display the pixel in a color of your choosing. However, the chosen color must be a function of the number of iterations taken to find that \( Z \) will be infinite. For example, if on the 1999th iteration you found that the magnitude of \( Z \) was greater than 2.0, then the color painted must be the same for all \( c \) values that required 1999 iterations. Your result in this case will be similar to that below.

**USE 16 separate threads to compute the Mandelbrot Set.**
Graduate Students Only. Next, enable the mouse clicks and movements in OpenGL and allow the user to select a square region in the displayed Mandelbrot Set image. When the region is selected (i.e., the mouse button released), recalculate the Mandelbrot Set using the minimum and maximum $c$ range based on the selected region. Limit the selected region to be square, not rectangular. Also, again for grad students only, maintain a history of the displayed sets and allow for a keypress of $b$ to return to the previous set.

Copying the Project Skeletons

1. Copy the files from the ECE6122 user account using the following command:

   ```bash
   /usr/bin/rsync -avu /nethome/ECE6122/MBSet .
   ```

   Be sure to notice the period at the end of the above commands.

2. Change your working directory to MBSet

   ```bash
   cd MBSet
   ```

3. Copy the provided MBSet-skeleton.cc to MBSet.cc as follows:

   ```bash
   cp MBSet-skeleton.cc MBSet.cc
   ```

4. Build the binaries using the provided Makefile.

5. Run your program normally:

   ```bash
   ./MBSet
   ```

   This should open an X-window on your laptop with the proper Mandelbrot set display.
**Using CUDA**  You may want to use CUDA and GPU’s for this project to add a skillset to your resume; I will grade the CUDA submissions for a small number of extra credit points. Be aware that there are not very many jinx systems with GPU’s so you might have trouble getting access to one.

1. **Log in to the cluster using** jinx-login.cc.gatech.edu.

2. **Use the qsub command to gain access to one of the jinx systems with attached GPU card**
   
   qsub -l nodes=1:gpu

3. After a hopefully short wait you will get access to one of the GPU-Enabled systems, and you can edit and compile your program as needed. Entering `exit` will exit the GPU node back to jinx-login.

4. There is a Makefile and skeleton code in /nethome/ECE6122/MBSet-Cuda that you can copy as usual.

**Turning in your Project.**  Use the normal turnin-ece6122 procedure as always.

If you implemented a CUDA version then turn in two different solutions. First the MBSet as above. In addition also turn in the CUDA version from a directory called:

MBSet-Cuda

You will have to create this directory and put all MBSet code there including the CUDA code.