ECE4893/8893 **Fall Semester, 2012**

Mandelbrot Set Display

Assigned: Oct 24, 2012 *Due: Nov 7, 2012, 11:59pm*

Introduction In this assignment we will use the OpenGL graphics API, and the *CUDA* GPU programming environment 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*, the use of CUDA are all given in the paragraphs below.

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

$M = \{c \in C | \lim_{n\to\infty} Z_n \neq \infty\}$

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 ∞, which would take quite a bit of computation. Luckily, we can make two simplifying assumptions.

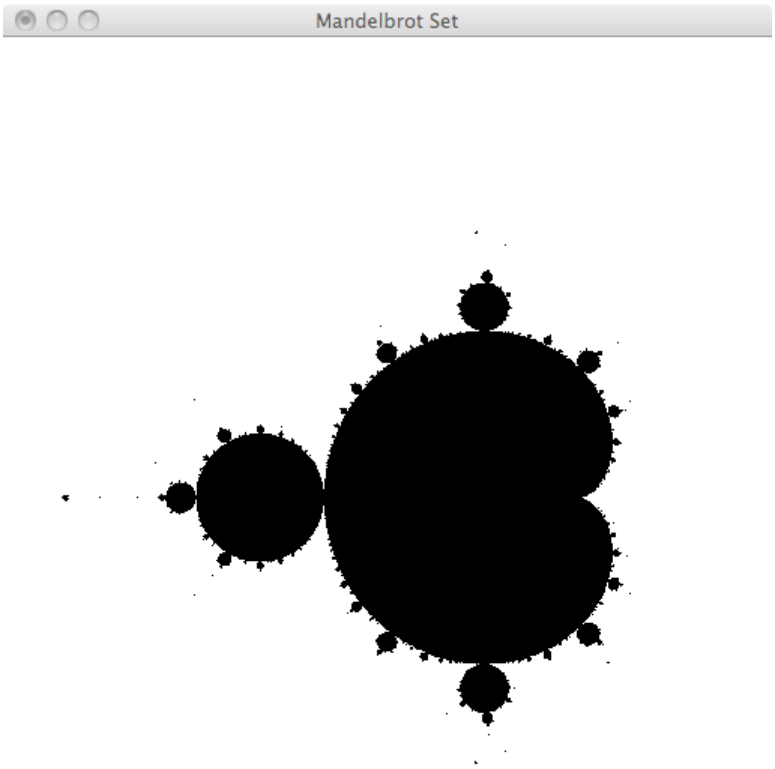
1. If the magnitude of any Z_n is greater than 2.0, then it can be proved that Z will eventually reach infinity, so if we ever get a Z_n for which the magnitude is greater then 2.0 we can stop iterating and claim that the point c is *not* in the *Mandelbrot Set*.

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*.

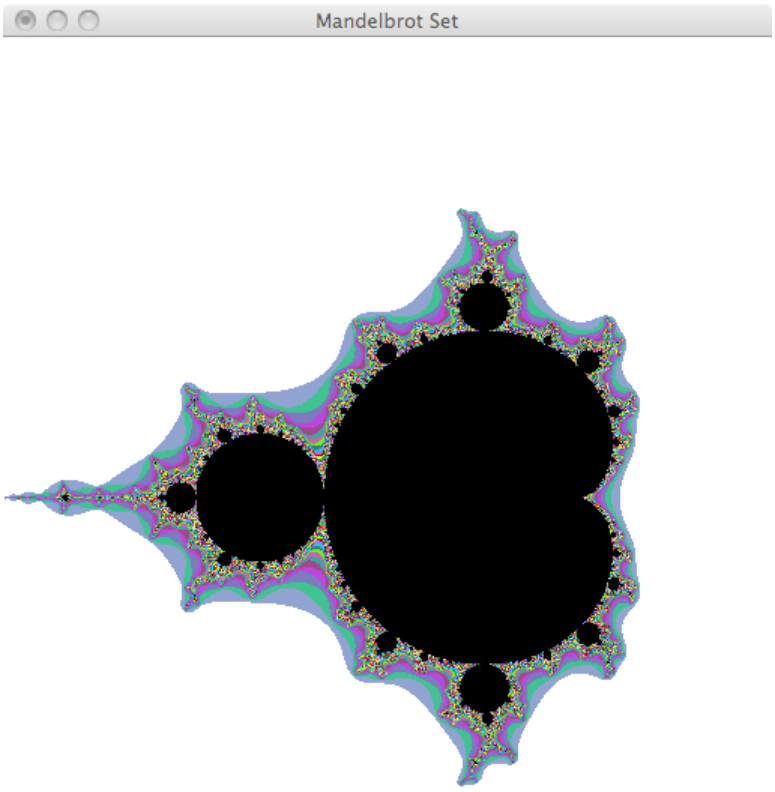
Displaying the *Mandelbrot Set* Start by creating a display window of 512 by 512 pixels. Then compute and display the *Mandelbrot Set*. Unfortunately, it appears in the above discussion of the math to compute the *Mandelbrot Set*, that we have to iterate all possible values of $c \in C$. Again we are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

$(-2.0, -1.2) < c < (1.0, 1.8)$

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, otherwise display a white pixel. Your result should be identical to that below.



Graduate Students. If the point is *not* in the *Mandelbrot Set*, 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 1999^{th} 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.



Next, enable the mouse clicks and movements in OpenGL and allow the user to select a rectangular region in the displayed *Mandelbrot Set* image. When the region is selected (ie. the mouse button released), recalculate the *Mandelbrot Set* using the minimum and maximum c range based on the selected region. *Graduate Students*, 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.

Using CUDA For your solution, use one CUDA thread per pixel and have the cuda thread calculate the iteration count for it's assigned pixel. Once all threads have completed the interation counts, then the main program can use OPENGL to paint the pixels of the appropriate color in the display window. Several cuda examples are included in the skeleton code directory.

Copying the Project Skeletons

1. **IMPORTANT NOTE** In addition to the jinx cluster, we also have access to the *pace* cluster for this work, as there are more systems in that cluster that support the *CUDA* environment than in the jinx cluster.
2. Log into `ece.pace.gatech.edu` or `jinx-login.cc.gatech.edu` using `ssh` cd .and your prism log-in name.
3. Copy the files from the ECE8893 user account using the following command:

   ```
   /usr/bin/rsync -avu /nethome/ECE8893/MBSet .
   ```

   If you are on pace, use the following command:

   ```
   /usr/bin/rsync -avu /nv/pec1/ECE8893/MBSet .
   ```

   Be sure to notice the period at the end of the above commands.
4. Change your working directory to `MBSet`

```
cd MBSet
```

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

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

6. Then edit `MBSet.cc` to create your code for the project You should specify "-X" on your `ssh` command (this is the default on some systems and not on others).

7. Build the binaries using the provided Makefiles for both jinx and pace clusters. These are Makefile.jinx and Makefile.pace. On the make command you specify which makefile to use by the "-f" switch, such as:

   ```
   make -f Makefile.jinx MBSet
   ```

8. On both clusters you need to use qsub to submit your job to a node with GPU hardware installed.

Turning in your Project. If running on jinx use the normal `riley-turnin` procedure. We don't have a turnin procedure on pace, so we will have to get support from the system administrator to copy your subdirectories to my locations.