1 Introduction

For this program use either Code::Blocks on Windows or Linux, or XCode on the Mac, zip all the project into one zip file and upload to the assignment page of the MyClasses site for this class. In this exercise you need to update the DoxyGen documentation. Fully document all new data members and methods to any of the classes, and update the documentation for anything that has changed. Of course, include your name as an author on any files that you edited. There is a Windows executable for the program for you to run and experiment with.

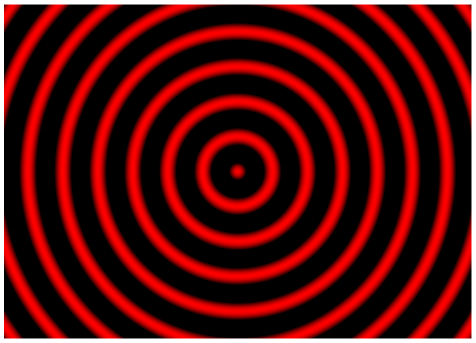
2 Exercise: Fragment Shader Play

This program is to get some experience with the GLSL shading language syntax and command library. The program is built on the ShaderPlay example from the class examples that are on-line. Make the following changes to the ShaderPlay example.

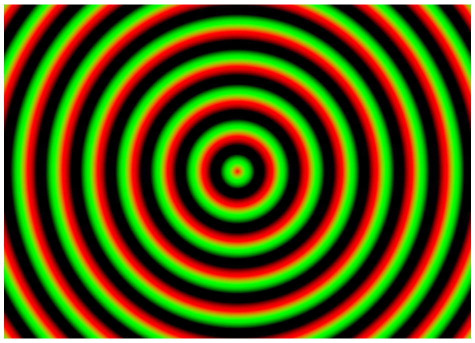
1. In the Box class, remove all color data and methods. Remove the colors array, any access methods to the colors and remove the loading of any color to the graphics card. The only data to be loaded to the graphics card is the vertex data.
2. In the graphics engine,
   (a) Remove the array of boxes and have only one box in the program. Change the accessor functions accordingly.
   (b) When the screen is resized have the box resize itself so that it covers the entire screen. What this does is it produces a fragment for each pixel on the screen, so even though there is some geometry on the screen (two triangles) the coloring is essentially geometry free.
3. In the UI class, remove the keyboard state processing and all key options except for the F10, M, A, and R options. You will be adding int the number keys to make the shader mode selections.
4. You will be leaving the vertex shader as the aspect ratio shader. In the fragment shader add in the following screen visuals. These should be lined to the UI and graphics engine via a uniform integer variable in the fragment shader.

   Number 0 Selected: When the 0 key is hit the shader should produce a blank screen.

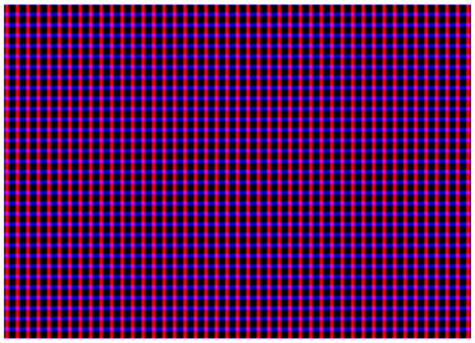
   Number 1 Selected: When the 1 key is hit the shader should produce the following static image.



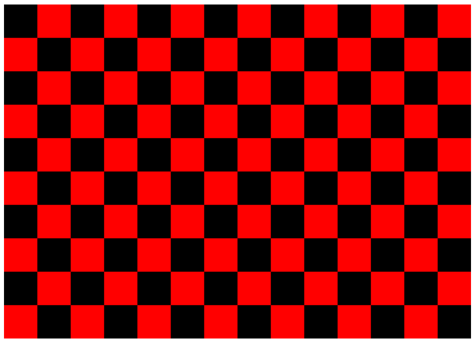
   Number 2 Selected: When the 2 key is hit the shader should produce the following static image.



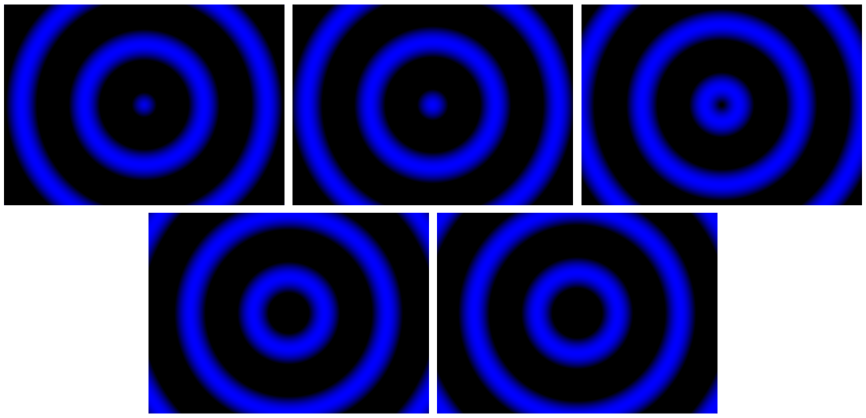
Number 3 Selected: When the 3 key is hit the shader should produce the following static image.



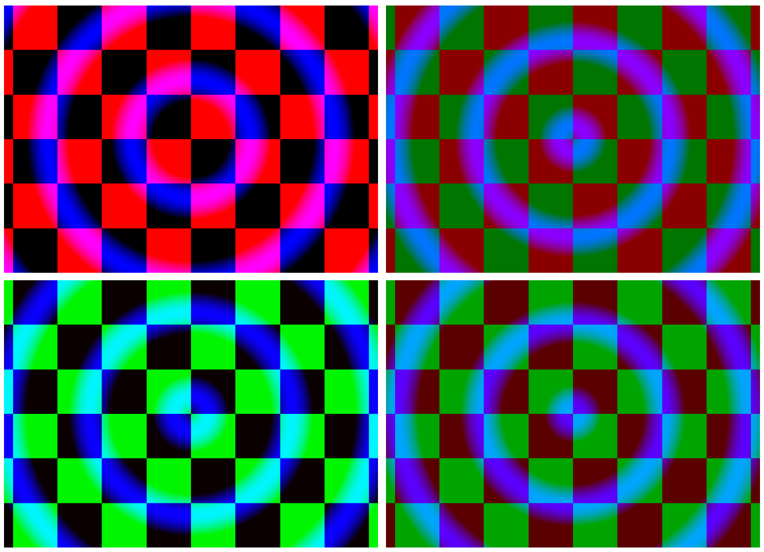
Number 4 Selected: When the 4 key is hit the shader should produce the following static image.



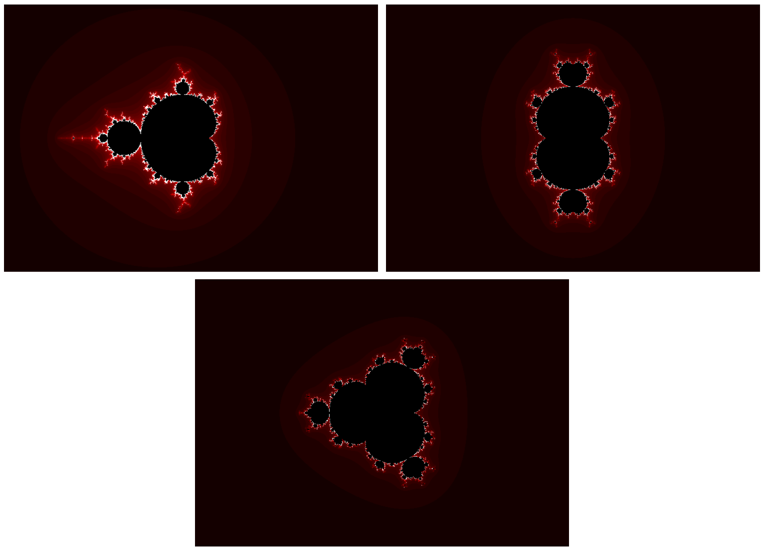
Number 5 Selected: When the 5 key is hit the shader should produce the following dynamic image. Blue circles growing larger from the center. For the animation, add in another uniform variable for the current time and have the CPU send the time to the graphics card on each frame.



Number 6 Selected: When the 6 key is hit the shader should produce the following dynamic image. Blue circles growing larger from the center over a checker board that the red squares fade in and out to black and the black squares fade in and out to green.



Numbers 7, 8, and 9 Selected: This one is extra credit and does not need to be implemented. The first is the Mandelbrot set and the other two are from the same family of fractals.



Here is how they are created. Take a point on the screen (a fragment in our case) and take its (x, y) position. We will think about the point (x, y) as being the complex number $x + yi$ where $i = \sqrt{-1}$. Let us call this complex number c, so $c = x + yi$. Now we start with another complex number q that is set to the origin, $(0, 0)$ and we do the iterative calculation $q = q^2 + c$ over and over again up to 100 times. As you can see from the equation, the result of the $q^2 + c$ calculation is stored back into q for the next iteration. This process is creating a sequence of complex number values at each iteration. One of two things will happen, first every value of q will have length less than or equal to 4 or there will be an iteration where the length of q is greater than 4. If the length of q ever exceeds 4 then there are some theorems that state the the sequence will simply get larger and larger and in the limit approach infinity. The sequences that stay inside the radius 4 circle are called the prisoners and those that march off to infinity are the escapees. If we have a prisoner we color the fragment black. If we have an escapee we note the first iteration where the length of q is larger than 4, call it t. If $t > 25$ we color the fragment white and if $t \leq 25$ we color the fragment the (r, g, b) color $(t/25, 0, 0)$. For complex numbers the formulas for addition and multiplication are as follows,

$$\begin{aligned}(a+bi)+(c+di) &= (a+c)+(b+d)i \\ (a+bi)(c+di) &= (ac-bd)+(ad+bc)i\end{aligned}$$

The algorithm above will produce the first image, the Mandelbrot Set. The other two images are produce by replacing $q = q^2 + c$ with $q = q^3 + c$ and $q = q^4 + c$.