CS3388B, Winter 2023

Problem Set 7

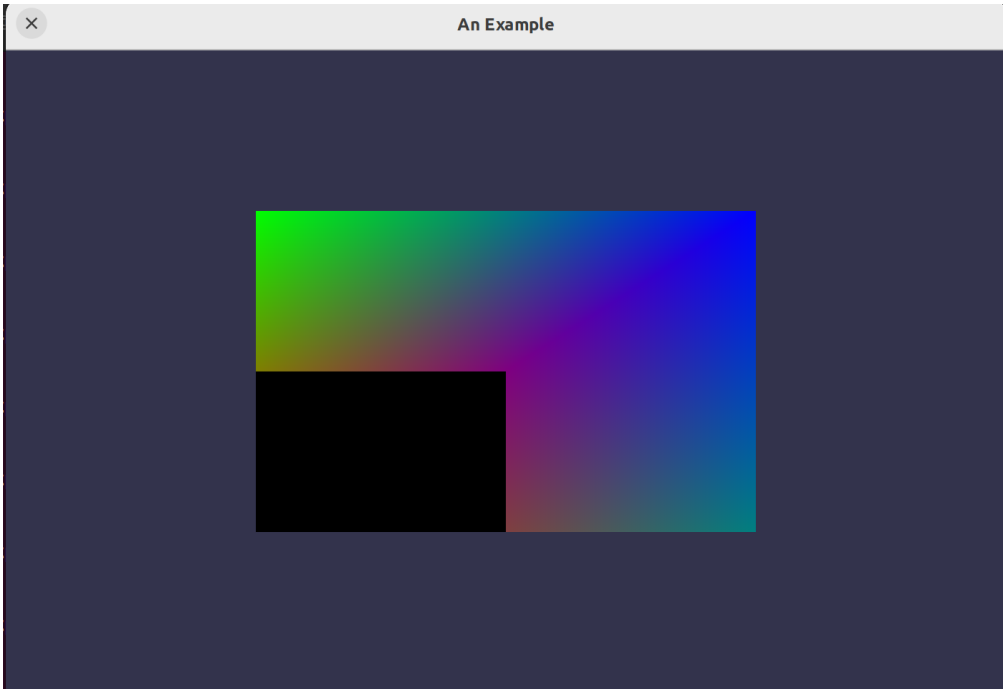
Due: March 12, 2023

In this problem set we are looking at shaders. Three C++ programs are provided to be modified. If you're using C++ throughout this course, modify the C++ code and submit the three modified source codes. If you're using Python, just submit the shader code (as a text file) which answers the questions.

Exercise 1. Exploring debugging with shaders.

In this exercise, we look to modify `P7-TODO.cpp` so that we get some visual feedback for the execution of a shader. First, implement a vertex shader that matches the vertex attributes specified in the render loop: position and color. Second, implement a fragment shader that uses an if/else to change a fragment's color. If the interpolated vertex color *received* by the fragment shader has an `r` value of greater than 0.5, return output black as the fragment's color. Otherwise, output the fragment's color as is.

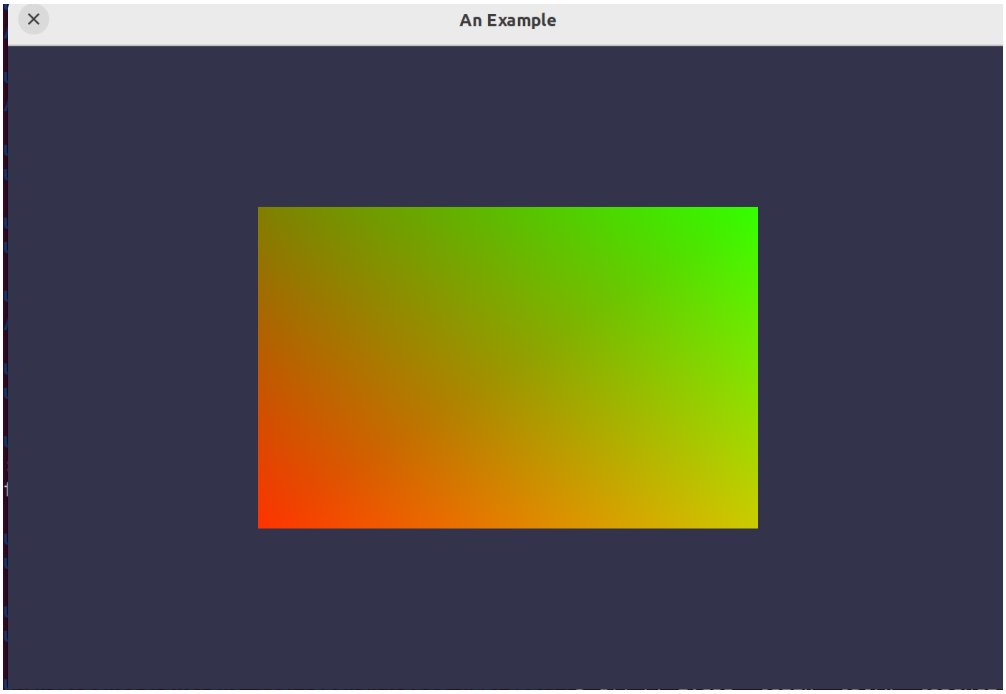
If done correctly, you should get the following:



Exercise 2. Exploring debugging with shaders, part 2.

In this exercise, we look to modify `P7-TODO2.cpp` so that we get more visual feedback for the execution of a shader. First, implement a vertex shader that matches the vertex attributes specified in the render loop: position and texture coordinates. Second, implement a fragment shader that uses the interpolated texture coordinates as the fragment's color. Use the texture coordinate's U value as the fragment's red color. Use the texture coordinate's V value as the fragment's green color. Set the fragment's blue color to 0, and alpha to 1.

If done correctly, you should get the following:



Exercise 3. Exploring textures with shaders.

In this exercise, we look to modify `P7-TODO3.cpp` so that it can handle multiple textures at once. In particular, that it can *blend* two textures together. First, implement a vertex shader that matches the vertex attributes specified in the render loop: position and texture coordinates. Second, implement a fragment shader that takes in texture coordinates, and outputs the fragment's color. The fragment shader should have 4 uniform variables: two sampler2Ds and two floats, named `tex1`, `tex2`, `blend1`, `blend2`. The fragment shader should use the texture coordinates to read two colors from the two textures, say `color1` and `color2`. Then, *blend* those two colors together so that the fragment's output color is `blend1*color1 + blend2*color2`.

If done correctly, you should get the following: