

Fall 2004

22C:151 Introduction to Computer Graphics

Assignment 11 (Last Assignment)

Due: Thursday December 9th at 12:01am (Wednesday at midnight)

Goal: Experiment with vertex and fragment shaders to get a feel for the low level abilities of modern graphics cards.

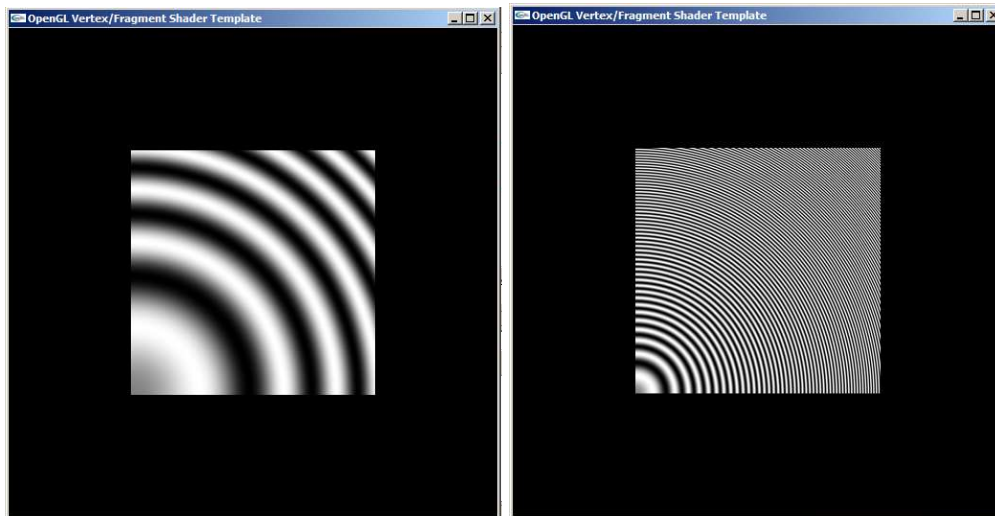
Download a copy of my “shader_template” program. Hopefully you should be able to compile it out of the box using the README instructions. However as long as the included executable works, you can finish this assignment. You do not need a compiler for this assignment, simply a text editor!

You will be modifying the two shaders, which are located in the files “testVertex.ARB.vsh” and “testFragment.ARB.psh” Since the precompiled executable loads these files, your shaders for this assignment must have the same filename (unless you change the OpenGL file to look in a different location)!

Problem 1 (10 points): Write a fragment shader that generates the sinusoidal function I showed in class. Use the fragment’s texture coordinate (“fragment.texcoord.xy”) for the x and y values in the following equation:

$$output_{color} = 0.5 * (1 + \sin(val * x^2 + val * y^2)) \quad (1)$$

The value val is passed in as a local parameter (i.e. you can access it via “program.local[0].x”), and you can change it via the ‘+’ and ‘-’ keys when you run the “shader_template” program. This should look like the following two examples (for $val = 20$ and 300.5 , respectively).



(Optional Extension for Problem 1): If you want to try some variations on this, “program.local[0].y” and “program.local[0].z” contain slightly offset values, so you can have a different pattern in the red, green, and blue channels. Also, another interesting function that produces similar results is:

$$output_{color} = 0.5 * x * y * (1 + \cos(val * x^2)) * (1 + \cos(val * y^2)) \quad (2)$$

Problem 2 (10 points): Now that you can see some cool Moiré effects on your quadrilateral, lets move the vertices around a bit in the vertex shader. Notice in the OpenGL program, that the one big quad in this program is actually made up of 400 smaller quads. For this problem, we’re going to use the vertex shader to manipulate the location of these quad vertices *before the quads are rasterized!!* This means, you’ll be changing the displayed geometry without recompiling the program.

Based on the *time* variable passed into the vertex shader as a local parameter (i.e., as “program.local[0].x”), change the location of the vertex as follows:

$$newObjSpacePos_x = oldObjSpacePos_x \quad (3)$$

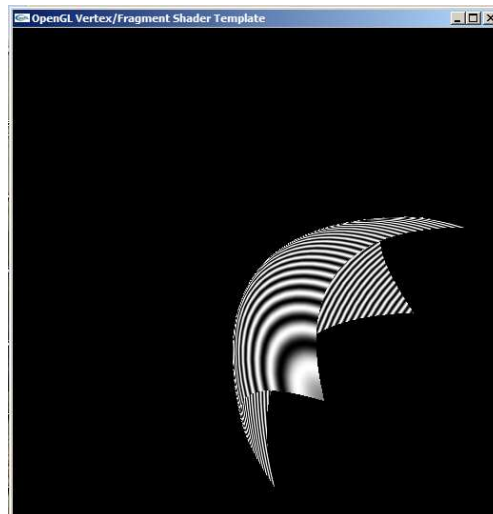
$$newObjSpacePos_y = oldObjSpacePos_y \quad (4)$$

$$newObjSpacePos_z = oldObjSpacePos_z + 5 * time^2 * (oldObjSpacePos_x^2 + oldObjSpacePos_y^2) \quad (5)$$

In your vertex shader, you can access all the values specified in the OpenGL program. For instance, your object-space vertex position is accessible in “vertex.position,” the texture coordinate is accessible in “vertex.texcoord,” the color is accessible in “vertex.color,” et cetera.

Two things you **must** do in your vertex shader: (1) Transform your new object space position into clip coordinates (using the combined modelview/projection matrix) and (2) Pass down your input texture coordinates so your new vertex shader will have access to them.

The results of this change are subtle – in fact you will not notice a difference when you first run your program. To see these changing vertex locations, you need to rotate your quad (using the left mouse button), and should see something that changes over time similar to this:



Extra Credit (Up To 5 points): Change the “shader_template” program to use either the Dragon or Buddha OBJ file instead of a quad. Write a fragment shader that uses the location of a light to create an “interesting” result on this model. This result should, of course, change as the light moves. The number of points received will depend on the coolness of your result. Note that this is probably harder than the rest of the entire assignment, but is here to reward those people who think shaders are neat and want to put more time into understanding them. (Alternatively, you can modify your existing OpenGL program instead of modifying my template.)

Make sure to update your web page with images of this assignment!!