

Fall 2009
22C:151 Introduction to Computer Graphics
Assignment 8

Due: Wednesday November 11th at 11:59pm

Goal: Experiment with OpenGL's GLSL vertex and fragment shaders.

Download a copy of my shader “template” programs. 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!

For Problem 1, you will be modifying the two shaders “test.vert.glsl” and “test.frag.glsl” that come with the shader template #1. For Problem 2, you will be modifying the shaders (with the same names) that come with shader template #2. Since the precompiled executables load these files, your shaders for this assignment must have the same filename (unless you change the C++ code to look in a different location)!

Note: These programs “intelligently” select the GL window resolution, so your program may either run at 512^2 or 1024^2 , depending on your screen resolution. All images were captured at 512^2 – Moiré patterns will take longer to appear at 1024^2 .

The two problems (each worth 10 points) are separated out on the following two pages.

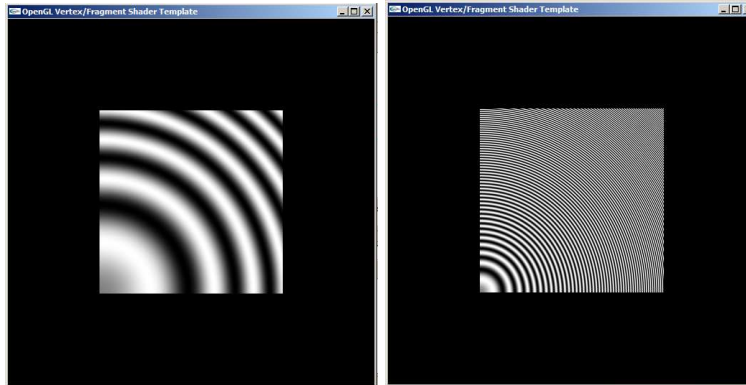
Make sure you read the *README* that comes with the program (and/or source) if you have problems running the program!

Problem 1 (10 points): Modify Shader Template #1 to exhibit the following behavior.

Part A (5 points): Compute the color in a fragment shader using the sinusoidal function:

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

Use the fragment's texture coordinate (“gl.TexCoord[0].xy”) for the x and y values and val is passed in as the uniform parameter “scaleValue” (i.e. you can access it via “scaleValue.x”), which can change via the ‘+’ and ‘-’ keys when running the program. This should look like the following two examples (for $val = 20$ and 300.5 , respectively).



Part B (5 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 uniform parameter “theTime” passed to the vertex shader, change the location of the vertex as follows:

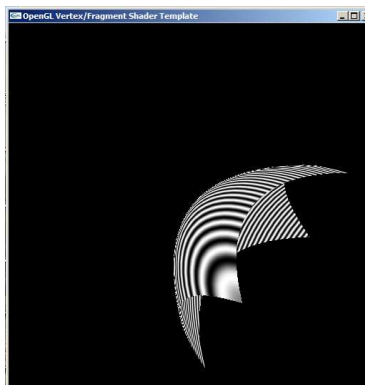
$$newObjSpacePos_x = oldObjSpacePos_x \quad (2)$$

$$newObjSpacePos_y = oldObjSpacePos_y \quad (3)$$

$$newObjSpacePos_z = oldObjSpacePos_z + 5 * theTime^2 * (oldObjSpacePos_x^2 + oldObjSpacePos_y^2) \quad (4)$$

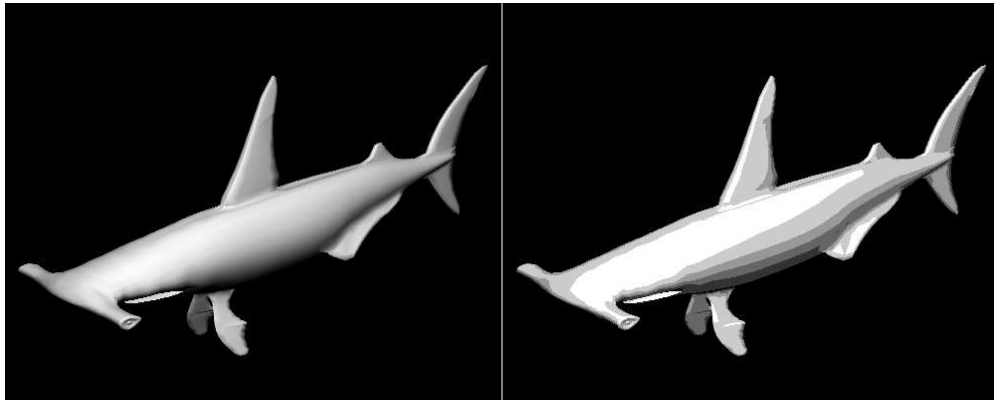
* Remember, two things you **must** do in your vertex shader are: (1) Transform your new object space position into clip coordinates (using the combined modelview/projection matrix) and (2) Pass down your input texture coordinates so your new fragment shader for Part A will have access to them.

* 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:



Problem 2 (10 points): Shader Template #2 defaults to showing the color of the shark model as the model's object-space surface normal. Modify template to exhibit the following behavior.

Part A (5 points): You will create a “cel-shaded” shark. The basic idea behind cel shading is that surfaces are still shaded in a semi-realistic way, where highlights move around intuitively as you move the light (light movement is done with the middle mouse button in the template). However, instead of intensity smoothly varying across the surface, it changes in discrete steps. This figure, for example, compares the shark with simple grayscale $\vec{N} \cdot \vec{L}$ as the color with a discretized “cel” shading based on the same value. Implement a similar cel shading by:

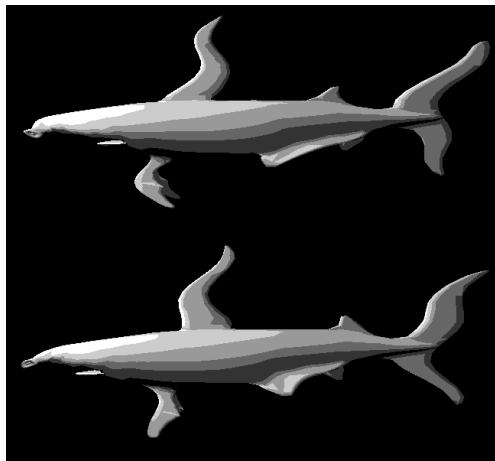


1. In the vertex shader, find the eye-space vector \vec{L} from each vertex to the light.
2. Transform the object-space object normal into the eye-space object normal \vec{N} .
3. Compute the dot product $\vec{N} \cdot \vec{L}$ and pass this value in `gl_FrontColor` to the fragment shader.
4. Discretize this value into 5 bins in the fragment shader. (E.g., `floor(5*($\vec{N} \cdot \vec{L}$)/5)`)

Part B (5 points): As in Problem 1, use the vertex shader to modify the position of each vertex prior to rasterization. Modify the object-space's x-coordinate prior to transformation into clip space, such that:

1. The model is still recognizably a shark.
2. The x-coordinate varies with time (you can use the uniform value “theTime” passed down from the OpenGL program).
3. The x-coordinate variation depends on the object-space y-coordinate. (I.e., some bits of the shark move right whereas bits, at different heights, move to the left).

The easiest way to achieve this is to use a sinusoidal variation in the x-coordinate. However, you will likely need to play with the scales to get an image similar to the following (images from 2 different times):



Extra Credit (Up To 5 points): Modify either the Shader Template #1 or #2 to create an “interesting” result. This result should change as a light moves around the scene. You may wish to modify the OpenGL program to pass additional information down to the shader to facilitate increased “coolness.” 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!!