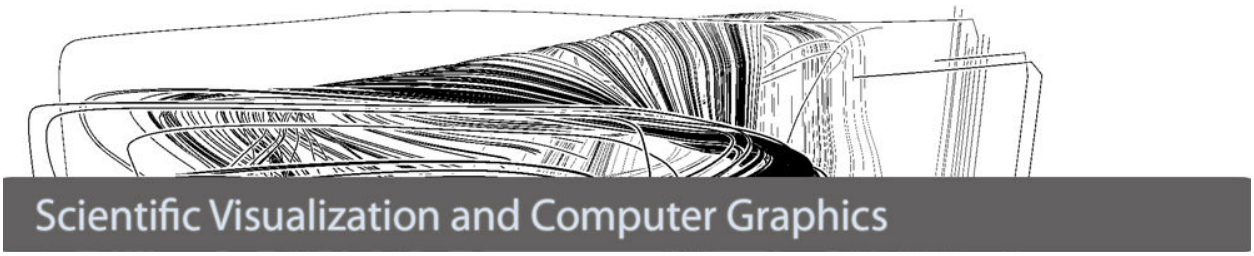


CSCI 3090 Assignment Four

GPU Programming: Vertex and Fragment Shaders (6%)

Due: April 8, 2014 (11:59pm)

Introduction

This assignment gives you the opportunity to try your hand at GLSL programming. All the questions in this assignment are built upon starter code courtesy of Daniel Vogel.

BONUS marks are available to a maximum of 20%.

Notes on Starter Code

MeshViewer.cpp draws several objects – a sphere, cube, teapot, and cylinder. Press "," to change between models. Use + and – to move the camera in and out. "s" will reduce the number of facets in the sphere (it's made by subdivision of a tetrahedron). "S" will add facets. Drag the mouse to rotate the object. Press spacebar to toggle between moving the object and moving the light. Adjustables U, V, W will scale the displayed mesh. Adjustables X, Y, Z will translate it.

Toggle a visualization of the vertex normals by pressing "n".

Part One (10%) Phong Shading (Lighting in Fragment Shader)

Starting with `MeshViewer.cpp`, `vshader.glsl`, and `fshader.glsl` move the specular and diffuse lighting component calculations from the vshader into the fshader. Moving the specular part was activity 2.1 of Lab 7 – you may copy and extend your solution from Lab 7 if you completed it.

This will require that you:

- Add the light and material properties to the fragment shader
- Pass the normal N and the half-vector H, and the light vector L from the vertex shader to the fragment shader (Hint: make them "out" from the vertex shader and "in" to the fragment shader. They will be interpolated across the primitives automatically.)

- Move the specular and diffuse light calculations to the fragment shader and add it to the final gl_FragColor

SUBMIT: In your assignment report, past a screenshot of your new specular highlights under "Part One". Submit your updated shaders as `phong-vshader.glsl` and `phong-fshader.glsl`.

Extend your new shaders for the remaining questions.

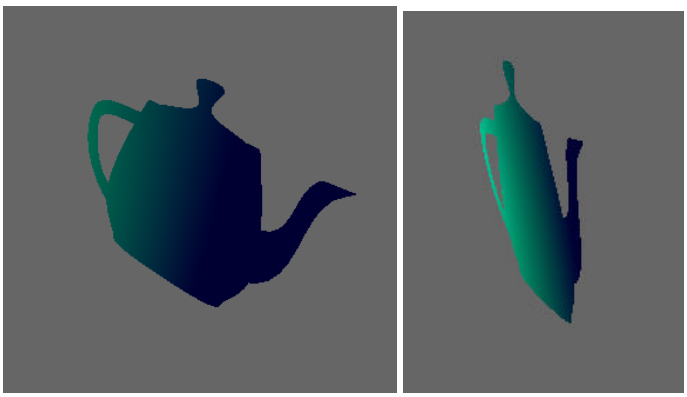
Part Two (20%) Flatten Shader

Vertex shaders are used to modify vertex locations. Recall that components of vectors in GLSL can be accessed with swizzling. In this part of the assignment, extend your vshader to create 'flattened' renderings of the objects. This will require you to set the z coordinate of all vertices to 0 *before transformations are applied*. If the object becomes flattened, the input normal vectors will no longer be valid. You should reset the normal vectors to be appropriate for a flat object. Check to make sure lighting behaves as expected on your result.

Note that `in` parameters are cannot be changed. Thus, you may need to initialize a new `vec4` based on an existing one. See the OpenGL API documentation for the `vec4` constructor information.

In the given code, `vPosition` is the input vertex position in local coordinates from the application program, `vertexPos` is the vertex position in world coordinates, and `gl_Position` is the output position of the vertex in screen coordinates. Choose carefully which one you will modify.

You should get something which looks like this:



SUBMIT: In your assignment report put screenshots of your flattened teapot under "Part Two"

Save your new vertex shader as `flatteapot-vshader.glsl` and submit it with your assignment submission.

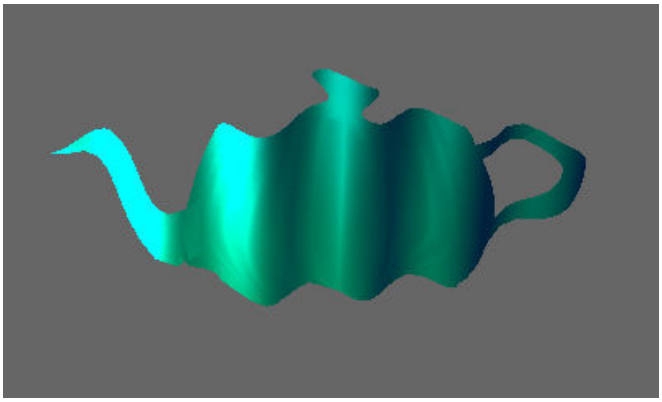
Part Three (20%) Wave Shader

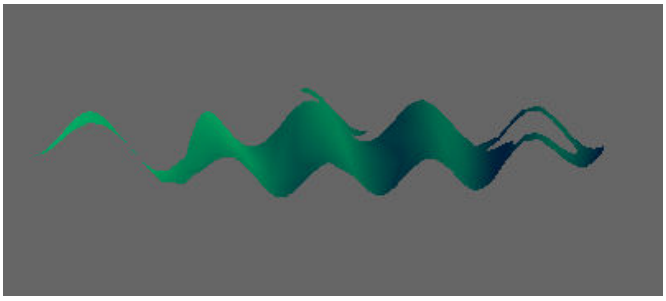
Modify your flatten shader to create a 'wave' shader which modulates the z component of the vertices with the following function: `z = sin(5.0 * x) * 0.25.`

The challenge for this part is to correctly calculate the new normal vectors so the lighting appears correct. Note that we are creating a sine wave in the x/z plane using coordinates `(x, y, sin(5x)*0.25)`. That is, the value of y doesn't affect the wave.

Some hints: The derivative with respect to x will give us a tangent vector to the wave. (Calculate this derivative). Given your tangent vector in the x/z plane is this: `(tx, 0, tz)`, then the normal is: `(-tz, 0, tx)`.

Your resulting teapot should look like this:





SUBMIT: In your assignment report put your formula for the normal vector and screenshots of your wavy teapot under "Part Three"

Save your new vertex shader as `wavyteapot-vshader.glsl` and submit it with your assignment submission.

Part Four (50%) Toon Shader

Cel shading or toon shading is a non-photorealistic cartoon or comic book style rendering. Edges are highlighted, and lighting is produced using a step function. In this part, you will create a cel shading of the Utah teapot. An example rendering, from Wikipedia is:

The rendering process is:

1. Turn off back face culling, and render the teapot in solid black, extending the vertices along their normal vectors (in a vertex shader), making the teapot slightly larger.
2. Turn on back face culling and render the teapot undistorted using a 'toon shader'

This works because the back faces will be deeper in the scene and larger than the front faces, thus the front faces will render in front using the depth buffer, but black edges will show.

Step 1 notes: Back face culling can be toggled using `glEnable(GL_CULL_FACE)`.

Step 2 notes: *This step is probably easier – try it first.* There are many ways to create a toon shader. You can find different implementations online. Perhaps the easiest way is to threshold the diffuse and specular values with a *ramp function*. For example, you could try the following pseudocode in your shader:

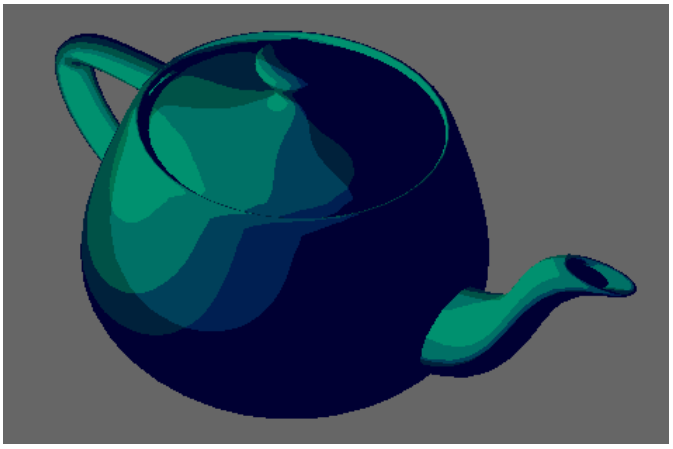
```
float s = max(dot(H,N), 0)  // this is the dot product part of the specular lighting
calculation

if (spec > 0.6) {
       spec = 0.9;
} else if (spec < 0.6 && spec > 0.3) {
      spec = 0.4;
} else if (spec < 0.3 && spec > 0.1) {
      spec = 0.1;
} else {
       spec = 0;
}

specular = pow(spec, material.specularExp) * material.specular * light.specular
```

A similar function can be used for diffuse lighting.

The following image was produced using the given material and lighting properties, and with **only step two of the above process** (i.e. the step function, without the edges).



Note that your result will look better if you also accomplish step one. You can use as many or as few steps in your ramp functions as you want. You may also modify the lighting and material properties to get a rendering you like and which highlights your achievements.

You will need to make changes to the application code as well as the shaders. You may need more than one shader program. You can compile multiple programs and choose between them using `glUseProgram( program )` where "program" is an index received from `loadAndInititalizeShaders()`.

SUBMIT: Prefix all files with "toon", e.g. "`toon-MeshViewer.cpp`" and "`toon-vshader1.glsl`". Submit application code and all shaders. Include screenshots of your results in your report under "Part Four". You may also submit a Windows executable.

Bonus (20%): Bump Mapping

Extend textmap.cpp and its related shaders to achieve bump mapping. This will require you to:

- Add phong illumination to the texture shaders (5%)
- Import and use a second texture (the bump map) to modify the xyz components of the normal vectors according to the rgb components of the bump map texels. Don't forget to renormalize your normal (15%)

A brick wall colour map and bump map are provided, but you may also download and use textures online.

Submit your source code and put images showing how the lighting is affected by the bump map into your report.

Assignment Tips

When you compile the VC project, the new EXE will be placed in a Win32/Debug folder. It is best to open the Solution (.sln) for the entire set of OpenGL/basic tutorials, and modify them as needed. This will ensure that compiling and linking works correctly. If you open an individual project (.vcproj) you will encounter errors.

General Assignment Guidelines

This is an individual assignment. While you may discuss your progress with your classmates, each member of the class is expected to hand in their own work.

Please include this statement on the cover page of your report:

"I, <insert name>, certify that this work is my own, submitted for CSCI 3090U in compliance with the UOIT Academic Integrity Policy."

In this course, you MAY NOT review, copy or otherwise use any graded academic assignments from prior semesters or other sections of this course, in written, electronic, or verbal form, used in whole or part, including formatting of any assignment.

Submission Guidelines

Please hand in a brief report (Word or PDF) describing your solution, including images produced by your programs, answering any questions posed in the assignment handout, and confirming the academic integrity statement. This report should be well-formatted, clearly organized, and use proper grammar and spelling. You must also submit well-formatted source code which can be compiled by the instructor. **Submit all files in a single zip archive.**

If you know there are errors in your calculations, please describe the problems to receive partial marks (e.g. "By drawing the normal vectors with lines I see that they are incorrect, thus the shading is also incorrect."). Suggest what you might change to fix the problem.

Any high-resolution images you produce may be submitted separately. A low-resolution version should be embedded in your report.