Computer Graphics Coursework 1

Deadline 4pm, 24/10/2016

Outline

The aim of the coursework is to modify the vertex and fragment shaders in the
provided OpenGL framework to implement transformations using the provided
model, view and projection matrices, and to implement Phong shading. The
object is rendered into a texture, which can then be modified in a post processing
fragment shader before being drawn onscreen.

Specification

The additions required are:

- Modify the vertex and fragment shader code (vertex.glsl, fragment.glsl) to:
    - Transform the vertices using the model, view and projection matrices
      provided
    - Transform the normals using the matrix provided
    - Calculate vectors from the transformed vertex to the light source and
      eye (the light and eye coordinates in the global coordinate system are
      provided, these should not be transformed).
    - Calculate Phong illumination, with an ambient, diffuse and specular
      component
    - Implement Phong shading, using interpolated vertex normals per
      fragment/pixel

You should not use Gouraud shading (calculating illumination once per
vertex and interpolating).

Implementing these methods will give you a good mark, however extra marks
will be awarded for:

- Using the texture provided in combination with the Phong shading in fragment.glsl
- Modifying the post processing shader in postfragment.glsl, to add effects. For ideas see lecture 4 (OpenGL).

Framework

The *cw1.tgz* file contains a framework with all of the necessary code to create a window, load an object file and render to it using OpenGL. You do not need to understand the OpenGL API calls to be able to complete the coursework. **The only files you need to edit are *vertex.glsl* and *fragment.glsl*, and optionally *postfragment.glsl* .**

Compilation

The program should compile on DICE machines running Scientific Linux 7. However older machines using OpenGL 2.1 will not run the coursework. To check the OpenGL version on a machine type

```
glxinfo | grep OpenGL
```

in the terminal. The Drill Hall computer lab machines (1.B30) should all be capable of running the program, and you can test this using the demo framework before making modifications.

A makefile is provided to build the program, and it can be compiled simply by typing

```
make
```

in the folder containing the makefile and source files. Note that the shader programs are not compiled until runtime, and so any errors in these will not be reported until the program is run.
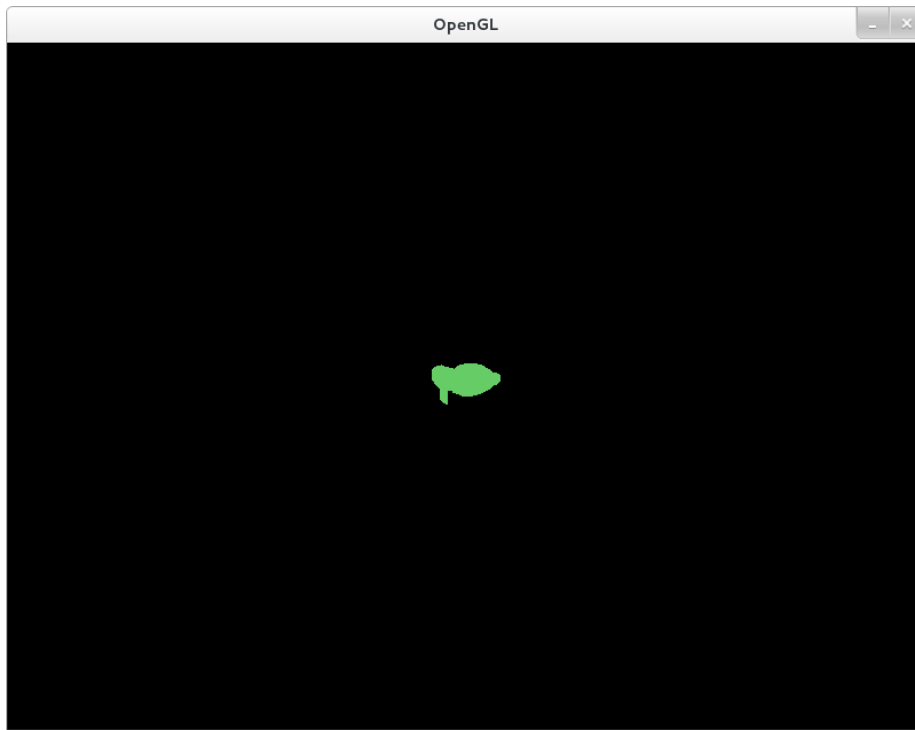
Running

The framework can be run using:

```
make run
```

or:

```
./demo1 models/bunny.obj
```

You should see something that looks like this before having modified the shaders:

Modifications

The vertex shader has some *uniform* variables as input that you should use to transform the vertices and normal vectors:

uniform mat4 model 4 by 4 model transformation matrix

uniform mat4 view 4 by 4 view transformation matrix

uniform mat4 proj 4 by 4 projection transformation matrix

uniform mat4 normTrans 4 by 4 normal vector transformation matrix

uniform vec4 eyePos Homogeneous coordinate of eye/camera

uniform vec4 lightPos Homogeneous coordinate of light source

Each vertex has position and normal vector attributes:

in vec3 position Vertex position

in vec3 normal Vertex normal

The vertex shader should output into the following variables:

vec4 gl_Position Built in variable, the final position of the vertex in screen coordinates

out vec4 lightVec Vector to light source

out vec4 eyeVec Vector to eye/camera

out vec4 normOut Vector of transformed surface normal

Once the transformations in the vertex shader have been correctly implemented the object should begin to rotate on the screen.

For the fragment shader, the GPU will automatically interpolate the outputs of the vertex shader between the three vertices of every triangle, calling the fragment shader with the interpolated values to draw each pixel within the triangle.

The fragment shader takes the three *out* variables above as input. There is also a *uniform sampler2D* provided that gives access to a texture, that you can use to add extra special effects (optional). Output should be stored in:

out vec4 outColour Vector of red, green, blue and alpha for each pixel, as floating point values between 0 and 1

Postprocessing

The object is rendered using vertex.glsl and fragment.glsl into a texture. Then two triangles that cover the screen are rendered using postvertex.glsl, with postfragment.glsl as fragment shader, having access to the texture into which the object was rendered. To modify the *postfragment.glsl* shader and add special effects, you have:

in vec2 pos Position of pixel to be drawn in screen space

uniform sampler2D depth Texture containing Z values for each pixel in the rendering of the object

uniform sampler2D colour Texture containing colour values for each pixel in the rendering of the object

The *postfragment.glsl* shader supplied with the framework simply looks out the corresponding pixel in the texture and outputs this to the screen.

Lighting calculations

Illumination and shading is covered in lecture 2. The relevant equations for the coursework are given below.

Normal vector transformation:

Special care must be taken when transforming normal vectors to a surface to make sure they remain orthogonal to the surface after transformation. For a 4 by 4 3D transformation matrix M applied to an object, the correct transformation to apply to the vertex normals is:

$M' = (\overline{M}^{-1})^T$

where $\overline{M}$ is the upper left 3 by 3 sub-matrix of M. See Shirley, 6.2.2 for more detail. This matrix is provided to you as *normTrans*.

Model, view and projection transformations:

The model transformation matrix takes vertex coordinates from their local coordinate system to the global coordinate system. Calculations of vectors from vertices to light sources and the eye should be performed in the global coordinate system, since the light and eye coordinates are given in global coordinates.

The final coordinate of a vertex on the screen is then calculated by first transforming the vertex coordinate from global coordinates to coordinates relative to the camera using the view transformation matrix, and finally by transforming coordinates relative to the camera using the projection matrix.

Diffuse reflection:

$I_{diff} = I_p k_d \cos\theta$

where θ is the angle between the normal vector of the vertex and the direction to the light source. I_p is the intensity of the light source, and k_d the diffuse reflectivity.

Specular reflection:

$I_{spec} = I_p k_s (\cos\alpha)^n$

where α is the angle between the reflected incoming light and the direction to the camera. I_p is the intensity of the light source, and k_s the specular reflectivity. n is the specular intensity.

If L is the normalised vector pointing towards the light source and N the (normalised) normal vector at the vertex, $\cos\theta$ can be calculated as:

$\cos\theta = L \cdot N$

To calculate $\cos\alpha$ using the unit vector V pointing from the vertex towards the camera we use:

$\cos\alpha = (2N(L \cdot N) - L) \cdot V$

Colour calculations:

Ambient lighting I_{amb} is simply a constant value. The values of $\cos\theta$ and $\cos\alpha$ should be set to 0 if they are negative. Then as an example to calculate the red, green and blue components of the vertex colour for a blue surface with a white specular reflection:

$r = I_{spec}$

$g = I_{spec}$

$b = I_{spec} + I_{diff} + I_{amb}$

Hints

Some things you may find useful to know:

- OpenGL colour values for the red, green and blue channels are floating point values in the range 0.0 to 1.0.
- To combine ambient, diffuse and specular lighting you can simply add the red, green and blue intensities for each together.
- You can find documentation on the OpenGL API calls used in the framework at: http://docs.gl

Submission

Coursework is submitted electronically using the Informatics generic electronic submission procedure:

```
submit cg 1 file1, file2, etc.
```

or

```
submit cg 1 directory
```

The program must be compiled and run on DICE, and you must submit your work through the above route. Work will not be accepted otherwise. If your submission does not run on DICE you risk being scored 0.

University policies

Late policy:

http://www.inf.ed.ac.uk/student-services/teaching-organisation/for-taught-students/coursework-and-projects/late-coursework-submission

Conduct policy:

http://www.inf.ed.ac.uk/admin/ITO/DivisionalGuidelinesPlagiarism.html