CSC 305 Assignment 2 Part 2
Rendering with openGL

Due Monday, March 14, 2015

Minimum Requirements - Part 2 (20% of Assignment 2)

The minimum requirement of the second part of assignment 2 is to repeat the viewing and camera control task in the first part with openGL APIs and shaders. For this part, the model - view - perspective transform should be implemented in the vertex shader, instead of in the C++ code on CPU.

Features required:

1. Draw a cube with one solid colour around the centre of the rendering window, with openGL APIs and shaders. Construct the vertex position buffer and M-V-P transform matrices in the C++ code and upload these data on to the GPU. Then, transform the vertex position in the vertex shader and draw the cube in perspective.
2. Implement all the required camera controls in part 1 of the assignment, including rotation with left button and dolly-trolley with the right button.

At this point, you may notice that you cannot see the cube as a 3D shape, because all the surfaces of the cube are drawn in one colour. The solid colour cube appears even “less 3D” than the wireframe version. This example illustrates the necessity of using shading in the rendering, and the advanced requirements will guide you through the process of rendering a more realistic three-dimensional shape.

Advanced Requirements - (40% of Assignment 2)

- Rendering:
    1. Define a light source, adds Lambertian diffuse shading onto the cube surface, in addition to a constant ambient shading. The light source and surface normal information should be passed into the shader system as a vertex attribute channel, and the final pixel colour should be determined in the fragment shader. (3%)
    2. In addition to the diffuse shading, implement the specular shading in the fragment shader. Note that now you need to pass the viewing vector into the fragment

shader for the specular shading component. This completes the Phong shading model (4%).
    3. Read texture images from the disk and paste them onto the surface of objects in a natural, undistorted way (Cube 4%, Sphere 8%).
- Modelling and Animation:
    1. Create a second, smaller cube, which rotates around the center cube in a circular orbit and spins by itself, like the earth orbiting the sun (4%).
    2. Create a sky-box as the background of your scene and paste some cloud textures on to the sky-box. Textures on the skybox should be displayed as-is without shading, i.e., do not make specular light spots on the clouds. (2%).
    3. Use the program to create a smooth sphere to substitute the central cube. Procedurally generate the buffer of vertex position and normal with a polygonization algorithm for spheres. Do not input the vertex positions by hand or from an existing mesh file. Use relative more triangles (~200 will be enough) to get a smooth surface (10%).
- Interaction
    1. Implement the viewport mouse picking algorithm, such that when the user clicks an moving object in the scene (such as the secondary cube/sphere rotating around), the object expands (increase in scale) then shrinks to its original size (5%).

Notes

Writing program with APIs like openGL following its documentation is an important ability and you should practise it in this assignment. The API structure of openGL is complicated, and there is no point of memorizing the meaning and usage of every API functions. When writing this assignment, you should keep consulting the online openGL API documentation which can be found at:

https://www.opengl.org/documentation/

The official openGL WIKI gives tutorials and explanations on key concepts in openGL:

https://www.opengl.org/wiki/Main_Page

In addition to these, there are a plethora of openGL tutorials online. **If you used any code from online, you MUST refer it in your source code file, and identify which part is written by you with which part is from online.**

To read an external image file, you need an image reading library. On Windows and Mac, the programming framework comes with libpng, which saved the ray-tracing image for us in assignment 1. libpng is also capable of reading an external png format image file. The official openGL wiki lists several commonly used image libraries for you to pick from:

https://www.opengl.org/wiki/Image_Libraries