CSE 5542, Lab Assignment 3
Introduction to GLSL, due on 03/08/2012 11:59PM

1 Goals

We will learn how to set up shader programs in OpenGL and how to use GLSL to write basic shaders.

2 Requirements

The SD_Cube_Simple3 example provided on the course website calculates transformation and projection by a set of matrix functions and it exports the matrices to the vertex shader using uniform variables. Based on this example, please implement the following steps.

2.1 Mesh (4 Points)

You are provided with a MESH.h file that can read/write wavefront .obj files, calculate vertex normals, and render the mesh using the original OpenGL fixed function pipeline. Please convert it into a vertex buffer object (VBO) version for rendering.

Instead of defining a vertex as a position vector and a color vector, you can define it as a position vector and a normal vector. During debugging, you may want to modify the original shader so that the whole mesh is rendered with the same hardcoded color. (Since there is no illumination yet, you cannot see the surface of the mesh but you can tell the shape from the silhouette.) The original MESH.h file defines vertex positions and vertex normals in two separate arrays. So you will need to modify this data structure. Make sure that you are using the same data type when transferring the data from the main memory to the GPU memory and when informing the vertex shader about the attribute variables. A wrong data type will cause the program to interpret the data erroneously.

2.2 Shaders (6 Points)

Please load the bunny.obj file and write three shading programs as follows. (If you cannot get Section 2.1 working, you can still work on this part by using the original cube instead.) Bind the three shaders to the 1, 2, and 3 keys respectively, so the user can browse from one shader to another.

2.2.1 Phong Shading (2 points)

Write a Phong shading program that calculates the illumination at the fragment level (rather than at the vertex level as the Gouraud shading method does). Using the following formula to calculate the illuminated color for each fragment as Figure 1a shows:

\[ c = k_d(L \cdot N) + k_s(M \cdot V)^n, \]

in which \( k_d \) and \( k_s \) are two coefficients that control the brightness of diffuse and specular reflection, and \( n \) controls the specular effect. \( N \) is the vertex normal coming from the input. \( L \) is the lighting direction from the vertex to the light source, \( V \) is the viewing direction from the vertex to the camera center, and \( M \) is: \( M = 2(L \cdot N)N - L \). The three vectors \( L \), \( N \), and \( V \) are normalized and they can be in any space. For simplicity, we can handle them in the eye space, in which the camera center is simply the origin. Please hardcode the location of a point light source in the eye space in the shader (somewhere close to the camera center), so that it follows the camera when the camera moves. You can also hardcode the \( k_d \) and \( k_s \) values in the shader. A result example is shown in Figure 1b.
2.2.2 Checkerboard Cut (2 points)

A checkerboard can be defined procedurally using two intermediate variables:

\[ x' = k(x - 2\text{floor}(x/2)), \quad y' = k(y - 2\text{floor}(y/2)), \]

in which \( x \) and \( y \) are the fragment coordinate values. If both \( x' \) and \( y' \) are less than 1, or both are greater than 1, they are in the same color. \( k \) is a coefficient that controls the size of the checkerboard. Using this method, please write a shader program to cut the bunny into pieces by using the \texttt{discard} command in the shader. Figure 1c shows an example when \( k = 40 \). For illumination, you can ignore specular reflection and use diffuse reflection only.

2.2.3 Wood Texture (2 points)

The wood texture is often generated procedurally in graphics using the following function:

\[ w = \sin \left( \sqrt{k(x^2 + z^2)} \right), \]

in which \( k \) is a coefficient that controls the frequency of the pattern and \( w \) is a variable goes from -1 to 1. Using linear interpolation, we can then determine the fragment color as:

\[ c = \left( \frac{w+1}{2} \right) \text{dark\_color} + \left( 1 - \frac{w+1}{2} \right) \text{light\_color}. \]

\( \text{dark\_color} \) and \( \text{light\_color} \) are the two grain colors and you can hardcode them into the shader. Figure 1c shows an example when \( k = 1000 \). For illumination, you can ignore specular reflection and use diffuse reflection only.

3 Bonus Credit (1 point)

Can you define the point light source in the world space instead, so when the camera moves, the light does not follow? Write it as another shader and bind it with the \texttt{4} key.

4 Submission Guideline

Please submit the source code (without the executable or object files) by email to our grader: Xiaoyin Ge (gex AT cse.ohio-state.edu). Use “CSE 5542 Lab 3” as the email subject.