CSE471 Introduction to Computer Graphics (Spring 2015)
Instructor: Prof. Won-Ki Jeong
Start date: April 6, 2015, Due date: April 26, 2015, 11:59 pm.

Assignment 2: GLSL Shaders (100 pts)

In this assignment, you will learn how to use GLSL shaders. The following 2 sections describe special render effect you need to implement.

1. Per-fragment smooth shading using Phong illumination model

You need to implement per-fragment shading technique using vertex and fragment shader. You need to implement Phong illumination model. I will provide a part of Phong shader source code in my lecture notes, so you can use it to start. Make sure that you implement this shader using your mesh viewer you implemented for assignment 1 so that you can check out the result easily.

The important differences from the source code provided in the lecture note are as follows:

1. You may not use pre-defined shader variables for matrix, such as `gl_ModelViewProjectionMatrix` and `gl_NormalMatrix`. Instead, you should calculate matrices on your own (using the matrix class provided in blackboard) and pass those to your shader using `uniform` variable.

2. You may not use pre-defined shader variables for light and material properties, such as `gl_LightSource, gl_LightModel,` and `gl_FrontMaterial`. Instead, you should define light and material variable on your own and pass those to your shader using `uniform` variable.

3. Use the following keys to change diffusion, ambient, and specular parameters interactively (k_a, k_d, k_s in Phong equation):

1 or 3 : decrease/increase diffusion parameter (k_d)
4 or 6 : decrease/increase ambient parameter (k_a)
7 or 9 : decrease/increase specular parameter (k_s)
- or + : decrease/increase shininess parameter (alpha in Phong equation)

Make sure k_a, k_d, and k_s are between 0 and 1. You do not need to

implement the attenuation term.

Figure 1. Example Phong shading using a single light source.

2. Cartoon shader

In this share, you need to implement piecewise-linear shading that mimics cartoon-like rendering with silhouette as shown in Figure 2.

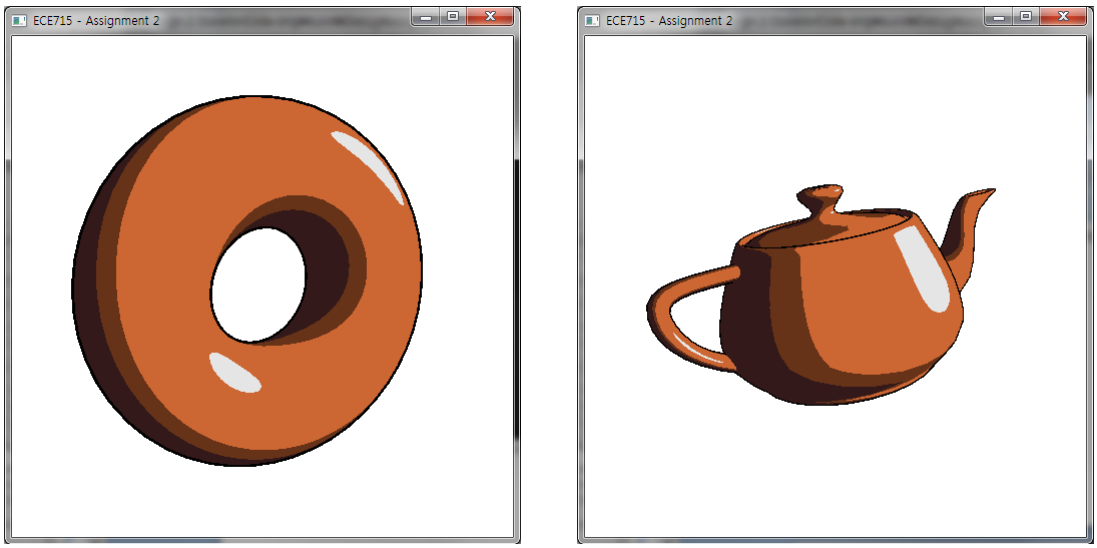

Figure 2. Example of cartoon shading with silhouette of Torus and Teapot models

The above shading effect consists of two render passes – silhouette render pass and cartoon render pass

Silhouette render pass Cartoon render pass

Figure 3. Silhouette & Cartoon Render passes

2.1 Silhouette render pass

To render the silhouette of the object (as shown Figure 3 left), you can draw the back-face polygons of the slightly larger object in black and then draw the front-face polygons of the original sized object in white. To make the object slightly larger, you can move each vertex along its vertex normal in your vertex shader. The silhouette thickness can be controlled by how much each vertex is moving along the vertex normal direction. Implement a keyboard callback so that pressing + increases (- decreases) the silhouette thickness interactively.

You can render front-faced or back-faced triangles only by using glCullFace().

2.2 Cartoon render pass

For this effect, you need to compute the angle between the surface normal and the light direction, and then assign a constant color value for a certain angle range (the higher the angle, the darker the color). The number of ranges can be interactively chosen using the number key (from 2 to 9). Then your shader will assign different shades of the color.

You already know how to compute per-fragment surface normal from the previous phong shader, so this will be easy to implement. Make sure that you normalize light direction and normal vectors in your fragment shader.

2.3 Combine silhouette & cartoon shading

Since you combine silhouette rendering with cartoon rendering, you need to draw only back-faced polygons with black color (first render pass), and then front-faced polygons can be drawn using cartoon rendering (second render pass).

You need to make two shader programs (one per each render pass), and render the object twice by switching between the programs.

3. Etc.

Test your code with glut 3D models first (glutSolidTorus(), glutSolidTeapot(), etc). Note that glutSolidTeapot() has a bug so the vertex normal is pointing inside the model (so you should use glFrontFace(GL_CW) to invert the orientation). Once it works fine, then try with triangular meshes (make sure per-vertex normal is calculated correctly).

Implement keyboard callback 'p' so that by pressing 'p' toggles between phong shader and cartoon shader. Note that some keys are assigned to both phong and cartoon shader (such as numeric keys), so you need to distinguish the keyboard callback correctly depending on the current rendering mode.

Submit a report describing your code and implementation details.

Score breakdown (total 100)

1. Phong shader (25)
   - 1.1. Correctness of per-pixel Phong illumination implementation (15)
   - 1.2. Ability to change diffuse/ambient/specular/shininess parameters interactively (10)
2. Cartoon shader (50)
   - 2.1. Correctness of silhouette rendering (15)
   - 2.2. Ability to interactively change the silhouette thickness by pressing + or - (10)
   - 2.3. Correctness of cartoon rendering (15)
   - 2.4. Ability to change cartoon shading ranges interactively using number keys (10)
3. Common (15)
   - 3.1. Ability to correctly render triangular meshes with shaders (5)
   - 3.2. Toggle between phong and cartoon shader by pressing p (5)
   - 3.3. Use the mesh viewer with virtual trackball function from assignment 1 (5)
4. Report (10)
   - 4.1. Quality of report (writing, structures, etc) (5)
   - 4.2. Including snapshot of each result (5)