AAA633 Visual Computing (Fall 2022)
Instructor: Prof. Won-Ki Jeong
Due date: Oct 23, 2022, 11:59 pm.

Assignment 1: OpenGL Triangle Mesh Viewer (100 pts)

In this assignment, you will implement an OpenGL triangular mesh viewer using Phong shading. Here is the list of required functions you need to implement.

1. Triangular mesh file I/O functions (10 pts)

Several simple and complex triangular mesh data are given for this assignment. First step for the assignment is implementing I/O and data management functions for triangular meshes. The input data format is 'off', an ASCII text file format containing the list of vertex coordinates and per-face connectivity information. The format of off file is as follows:

```
OFF              : first line contains the string OFF only
#v #f 0          : total number of vertices, faces, and 0
vx1 vy1 vz1      : x/y/z coordinate for vertex 1
vx2 vy2 vz2      : x/y/z coordinate for vertex 2
...
#v_f1 f1v1 f1v2 f1v3  : # of total vertices and each index for face 1
#v_f2 f2v1 f2v2 f2v3  : # of total vertices and each index for face 2
...
```

All the example files provided for this assignment will be triangular meshes, so the total number of vertices per face is always 3. Below is an example of an off file containing 34835 vertices and 69473 triangles:

```
OFF
34835 69473 0
-0.0378297 0.12794 0.00447467
-0.0447794 0.128887 0.00190497
-0.0680095 0.151244 0.0371953
...
3 20463 20462 19669
3 8845 8935 14299
3 15446 12949 4984
```

Once you read an off file from the disk, you should store the mesh in memory using three arrays – vertex coordinates, vertex normals, and indices. Use these arrays as the inputs to your GL buffer objects. Note that per-vertex normal is not provided in the off file, so you need to compute it on your own (average neighbor triangle normal).

2. OpenGL viewer using per-pixel Phong illumination model (80 pts)

Once you load a mesh, you should be able to display and inspect the mesh using a

dedicated viewer (as shown in Figure 1). We will implement a triangle mesh viewer using OpenGL and shaders. The skeleton glut code is provided. You will need to add more OpenGL code so that you can visualize triangular meshes. The viewer should have the following functions:

- Triangle mesh rendering using vertex & index buffer objects. You must store vertex coordinates and vertex normals using VBO (GL_ARRAY_BUFFER), store vertex indices using IBO (GL_ELEMENT_ARRAY_BUFFER), and use glDrawElements() to render them (20 pts)
- Orthogonal and Perspective projections, which can be switched using keyboard interaction (10 pts)
- Phong illumination model using shaders, change shading parameters using keyboard interaction (30 pts)
- Virtual trackball - zooming, panning, and rotation using mouse interaction (20 pts)

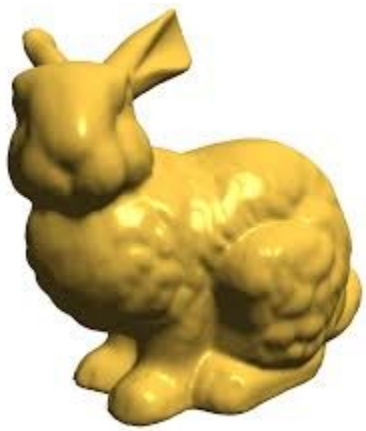

Figure 1. Example Phong shading of a bunny model using a single light source

You can use `glOrtho/glFrustum` for orthogonal and perspective projection. Use key 'o' for orthogonal and 'p' for perspective projection to switch between projection modes. Make sure you understand how those projections work.

You need to implement per-fragment shading technique using vertex and fragment shaders. You need to implement the Phong illumination model (Figure 2). I provide a part of Phong shader source code in the lecture notes, so you can freely use it as a starter.

For each light source

$$\mathbf{I} = \mathbf{I}_a + \mathbf{I}_d + \mathbf{I}_s$$
$$= \mathbf{k}_a \mathbf{L}_a + \mathbf{k}_d (\mathbf{l} \cdot \mathbf{n}) \mathbf{L}_d + \mathbf{k}_s (\mathbf{r} \cdot \mathbf{v})^{\alpha} \mathbf{L}_s$$

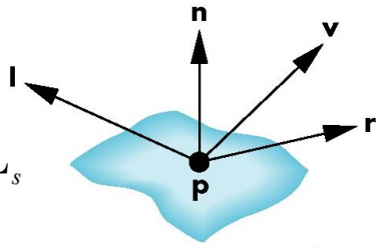

Figure 2. Phong illumination model

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.

Use at least two light sources with different colors (e.g., place one light on the right side and the other on the left side). Use your creativity to make a pretty image - try different ambient / diffuse / specula colors that you like the most.

You should implement virtual trackball functions. The trackball should work accurately. Make sure left-mouse is rotation, middle-mouse is panning (translation), and right-mouse is zooming. For panning, you need to apply translation on x/y plane. For zooming, you can move viewer's location (eye position) along the z-axis for perspective projection. You can use `glScale()` for zooming in orthogonal projection.

Tip : test your viewer code with a simple pre-defined GL example, like a cube or teapot (see below) provided by glut, before rendering a triangle mesh.

```
glutSolidTeapot(10);
```

3. Report (10 pts)

You need to write a report explaining what you did and learn in this assignment. Follow a standard report structure (introduction, method, result, conclusion, reference...), and include high-resolution figures in your report.

4. Etc.

The provided skeleton code is tested on a Windows PC and Microsoft Visual Studio. Use CMake to generate a solution file for Visual Studio. CMake is a platform-independent project generation tool (http://www.cmake.org/).

You should submit your **source code (not the project/solution files)**, the **report,** and **the best rendering image** in a single zip file. Everything must be self-contained, meaning that I should be able to unzip it, run CMake, and open using Visual Studio without any additional effort. The code must be compiled without additional external library setting. Make sure you compile your code in 64bit mode(x64) because the included freeglut library is compiled in 64bit mode.

Tip: Set the build directory different from the source directory when you run CMake to generate project file. By doing that, source files and project files are located in separate directories.

Good luck and have fun!