CSE471 Introduction to Computer Graphics (fall 2019)
Instructor: Prof. Won-Ki Jeong
Due date: Oct 13, 2019, 11:59 pm.

Assignment 1: OpenGL Triangle Mesh Viewer (100 pts)

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

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

Several simple and complex triangular mesh data are given for this assignment. First step for the assignment is provide I/O and data management class for triangular meshes. The input data format name is 'off', ASCII text file 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 input to your **buffer objects (vertex and index buffer objects)** and use **glDrawElements()** to render them. Note that per-vertex normal is not given in off file, so you need to compute it on your own.

2. OpenGL viewer (80 pts)

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

dedicated viewer (as shown in Fig 1). We will implement a triangle mesh viewer using OpenGL. 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 / Perspective projection (10 pts)
- Smooth / wireframe rendering (10 pts)
- Keyboard callback to switch between different projection and rendering modes (10 pts)
- Use multiple light sources and different light colors (10 pts)
- Virtual trackball - zooming, panning, and rotation using a mouse (20 pts)

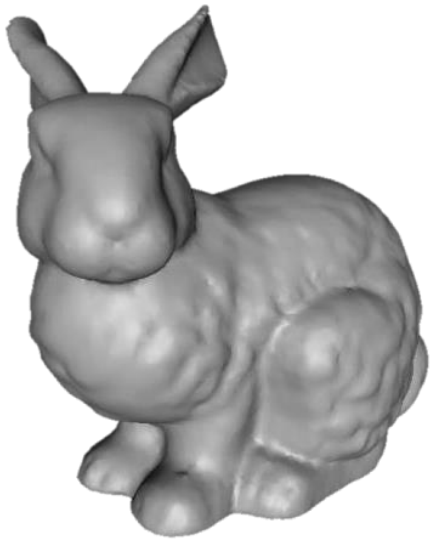

Figure 1 Smooth shading of the Bunny triangular mesh

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.

For smooth rendering, you need to use per-vertex normal (average neighbor triangle normal). Implement keyboard shortcut to switch between different shading modes : 's' is smooth shading, and 'w' is wireframe rendering.

Use at least two light sources with different colors (place one on right side and the other on 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 (will be discussed in the class). 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 have to scale it using glScale() for orthogonal projection.

Tip 1 : test your viewer code with a simple pre-defined GL example, like a cube or teapot provided by glut.

Tip 2 : `glutSolidTeapot()` has a bug – default surface normal vectors are pointing inside of mesh (meaning that treats clock-wise orientation as a front-facing triangle). In order to render correctly, you need to reverse it as below:

```
glFrontFace(GL_CW);     // reverse orientation
glutSolidTeapot(10);    // render teapot
glFrontFace(GL_CCW);    // return to correct orientation
```

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...). 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 (if that is needed, please modify the included CMakeLists.txt so that everything can be handled automatically by CMake). Note that I will not debug your code to compile and run, so it is your responsibility to make the code compatible to the skeleton code. Make sure you compile your code in 64bit mode(x64) because the included freeglut library is compiled in 64bit mode.

I will select top 3 best rendering images and show them in the class!

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!