COSE436 Interactive Visualization (fall 2021)
Instructor: Prof. Won-Ki Jeong
Due date: Oct 17, 2021, 11:59 pm.

# Assignment 1: WebGL Triangle Mesh Viewer (100 pts)

In this assignment, you will implement an WebGL 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
```

A basic text parser is provided in the skeleton file (see "off_reader()" in assign_1.js). You should modify the given skeleton parser to read .off file. Note that the given parser is a callback function of "파일선택" button in the browser. Once you select an off file, then the parser will be called automatically.

Once you read an off file from the disk, you should store the mesh in memory using three arrays – vertex coordinates (position), vertex normals, and indices. Use these arrays as input to your **buffer objects (vertex and index buffer objects)** and use

**gl.drawElements()** to render them. Note that per-vertex normal is not provided in the off file, so you need to compute it on your own.

**2. WebGL 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 WebGL. The skeleton Javascript and HTML code is provided. You will need to modify the given skeleton 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 gl.drawElements() to render them (20 pts)
- Orthogonal / Perspective projection (10 pts)
- Smooth / wireframe rendering (10 pts)
- Button callbacks to switch between different projection and rendering modes (10 pts)
- Use multiple light sources with different light colors (10 pts)
- Virtual trackball - zooming, panning, and rotation using a mouse (20 pts)

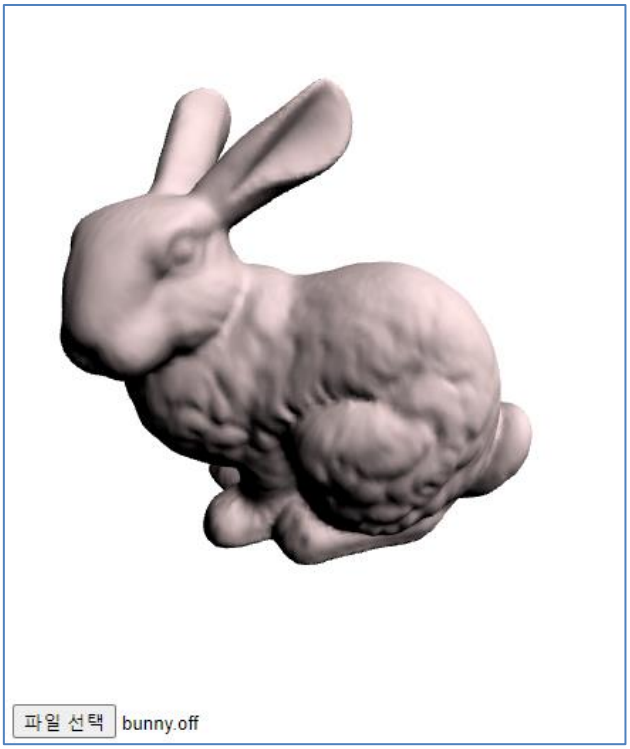


Figure 1 Smooth shading of the Bunny triangular mesh

You can use lookAt(), ortho(), and perspective() functions provided in MV.js for creating a viewing matrix and orthogonal/perspective projection matrix. Make sure you understand how those projections work.

For smooth rendering, you need to use per-vertex normal. You should average adjacent triangle normal vectors for each vertex.

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 colors that you like the most.

You should implement virtual trackball functions as 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 for orthogonal projection.

**Tip 1 :** Test your viewer code with a simple geometry, like a cube, for easier debugging.
**Tip 2 :** You will need many trial-and-errors before getting the image on the screen. You may want to use a debugger in the Chrome browser (press F12).
**Tip 3 :** You may use source code provided in our textbook (you can download from the class Blackboard).

**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 using a Chrome browser.

You should submit your **source code (HTML, JS) and** the **report** in a single zip file. Note that I will not debug your code to run, so it is your responsibility to make the code error free.

Good luck and have fun!