AAA633 Visual Computing (Fall 2022)

2022021585 Jaeyoon Lee

Assignment 1: OpenGL Triangle Mesh Viewer

1. Introduction

In this assignment, I implemented a Phong shading and some keyboard and mouse interactions. By parsing the given off file, it was possible to get vertex coordinates and per-face connectivity information. Since per-vertex normal is not provided, I implemented a code which computes vertex normal by average neighbor tringle normal. Once the mesh loaded, it was performed to store vertex and index information in each VBO and IBO. After successfully rendered the triangular meshes, I implemented the keyboard interactions to switch the projection mode and to change shading parameters. Also I added the mouse interactions to zoom, pan, and rotate the object.

2. Method

A. Triangular mesh file I/O functions

To read an off file from the disk, I used the given function `textFileRead()` in textfile.cpp. By parsing each lines, it was possible to generate the array of vertex coordinates and per-face vertex indices. To compute the per-vertex normal, I imported Vector.h file and Vector.cpp file to use `VECTOR3D` class to calculate the vector operations. Once the array of vertex normal is generated, then it is concatenated to the vertex coordinates array as `vnArray`. The whole process of generating the array is written in the function `generateArray()`, and it is performed once in the `main` function after the `glewInit()`.

B. OpenGL viewer using per-pixel Phong illumination model

After all the vertices and indices are loaded into the designated arrays, it was performed to store the vertex coordinates and the vertex normals into the `GL_ARRAY_BUFFER`, and the vertex indices into the `GL_ELEMENT_ARRAY_BUFFER`. Since the texture data is not given, there was no need to store the texture coordinates into the `GL_ARRAY_BUFFER`. Instead, it was needed to set the stride as `sizeof(float)*6` in the `glVertexAttribPointer()` function, since the `vnArray` is aligned as 3 coordinates and 3 normals per

vertex. Those processes of loading buffers are performed in `bufferManager()` function, right after the `generateArray()` in the `main` function. After the binding buffer, it was rendered by using `glDrawElements()` in the `renderScene()`.

For the keyboard interactions, I wrote down the code into the `keyboard()` function. In the case of projections, as the most vertices are located around the position of (0, 0, 0), I set up the default camera location as (0.0, 0.0, 10.0). For the orthogonal projection, since it is not related to the camera position, I set the value as `glOrtho(-1,1,-1,1,-50,50)` to make the object located on the center. In contrast to the orthogonal projection, perspective projection is related to where the camera is looking at, so I set the value as `glFrustum(-0.1,0.1,-0.1,0.1,3,20)` to make the object located in the view frustum and look naturally. Those projection modes can be switched by using key ‘o’ and ‘p’.

In the case of Phong shading, I changed the code of vertex shader and fragment shader. For the vertex shader, I added the code `layout (location=1) in vec3 in_normal` to get the vertex normal information from the VBO. For the fragment shader, I used `uniform` value to get the shading parameters which are manipulated by the keyboard interactions. All the equations are same as Phong illumination model, but there are two light sources in the code so the fragment color is averaged. The keyboard interactions for changing shading parameters are also implemented in the `keyboard()` function.

For the virtual trackball, I used `glutMouseFunc()` function and `glutMotionFunc()` function. By `processMouse()` function, it can handle the last position where the mouse has clicked, and which mouse button is clicked. By `processDragMouse()` function, it can calculate how much value of the mouse position has moved after clicked. By using those functions, it was possible to move the position and direction of the camera. For zooming, it moves the camera position along the z-axis for the perspective projection, and `glScale()` function is used for the orthogonal projection. When the mouse moves to the right, it zooms in to the object, and vice versa for zooming out. For the rotation, the Cartesian coordinates of the camera position was converted as spherical coordinates to match the mouse movement and the camera movement. After the movement is applied to the camera position in spherical coordinates, it is reconverted to the Cartesian coordinates.

3. Result

As a result, I successfully implemented the rendering of triangular meshes and the interactions of mouse and keyboard. Following figures show the result of the rendering and interactions.

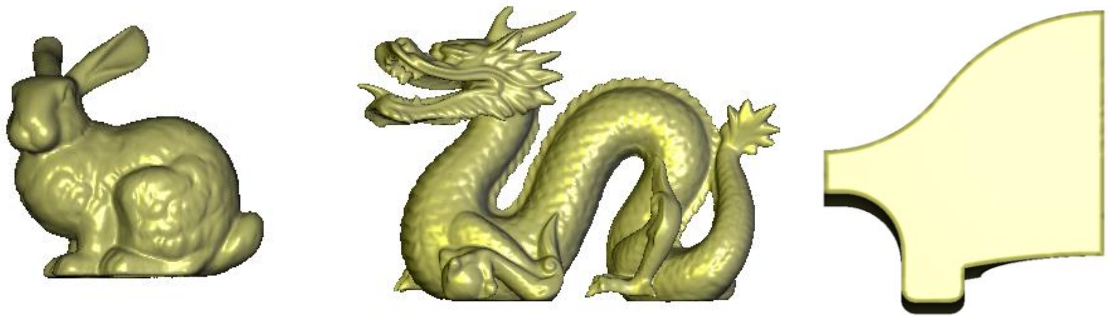

Figure 1. Rendered images of Phong shading

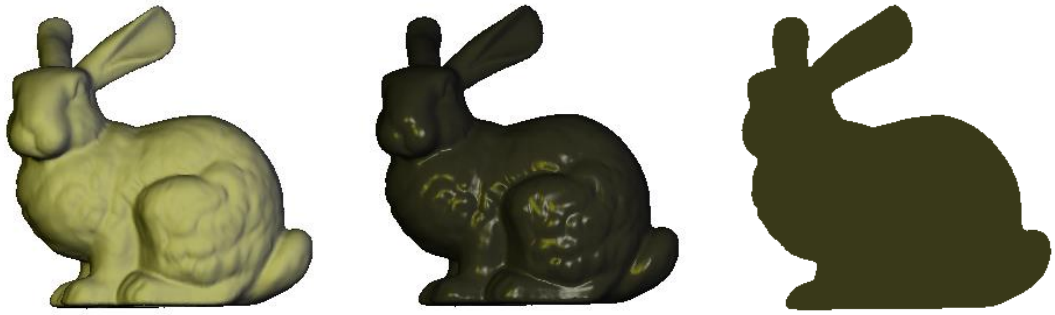

Figure 2. Changing shading parameters (specular, diffuse, ambient)

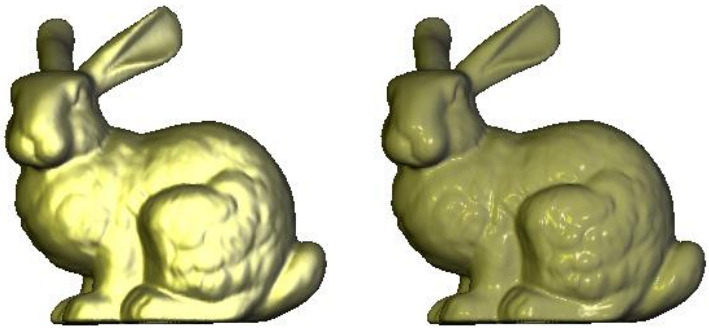

Figure 3. Changing shininess parameter

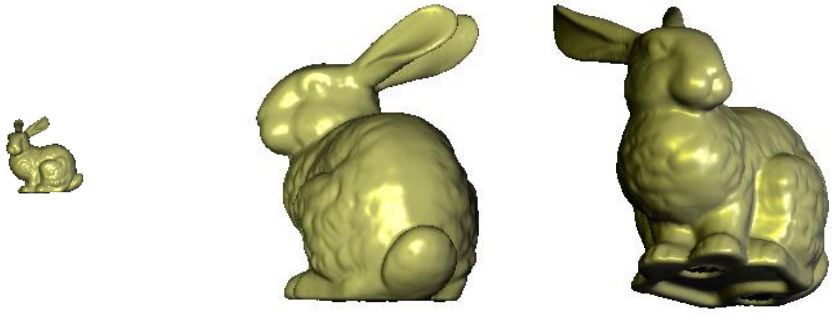

Figure 4. Virtual trackball – zooming, panning, and rotation

4. Conclusion

By conducting this assignment, I learned and understood general structures and pipelines of OpenGL. Some of interactions that I have implemented in the virtual trackball are not act naturally, but it still works well on simple movements. It was hard to check the difference between orthogonal projection and perspective projection, although it was implemented differently. I used two light sources, but one of them is a white light so it just looks as one light source.

5. References

https://heinleinsgame.tistory.com/category/OpenGL
https://tech.burt.pe.kr/opengl/opengl-tutorial/chapter-11
https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=nanapig94&logNo=221172431880
https://codingcoding.tistory.com/275
https://learn.microsoft.com/ko-kr/windows/win32/opengl/gllightfv
https://computergraphics.stackexchange.com/questions/151/how-to-implement-a-trackball-in-opengl