CSE47101 Computer graphics  
20111554 Wonjin Choi

Assignment 1 Report

1. Introduction

In this assignment, I implement openGL code that receives data from file and makes meshes from it. Also, some functions that can be used to switch between several rendering modes and polygon modes. Panning, zooming, and rotation using mouse were key points of this assignment.

I created Mesh class to manage file input/output and to create vertices data from it. Several global variables are declared at first part of main.cpp and they are used in panning, zooming, or rotating the model. Following function is used for initial setting for buffer objects, lights, and view. I’ll explain in details about Mesh class and other important functions.

\* In the visual studio, assign\_1 must be set as startup project to debug the program.

1. Methods
2. Mesh class

First of all, I’ll explain about Mesh class. In a header file Mesh.h, the mesh class is defined. The variables input and output are used for streaming data from file. nVertices and nFaces are used to store total number of vertices and faces. Actually, I stored vertices data in vertices array and faces data in faces array. Then, I calculated the center of model. It will be used later for camera setting or rotating, etc. I called calculateVertexNormal function to calcaulate vertex normals and stored it in normalSmooth or normalFlat which will be used later according to the shading type.

Let me explain the function calculateVertexNormal. Inside the function, I used two 3-dimensional vectors to calculate normal. From index calculation, I could calculate normals for faces. For normalFlat, I just make it same for every vertices in a triangle. For normalSmooth, I went through faces and found consisting vertices and then added corresponding normalFace value. After accumulate all values of all vertices, I normalized each component.

1. Global variables

As written in source code, cx, cy, cz are camera position, ox, oy, oz are center position, and ux, uy, uz are up vector of camera. dx, dy, storeDx, storeDy are used for panning. storeDx and storeDy are for remembering last panning position so that successive panning can be done naturally. Following variables are for rotating. angle is rotation angle, and storeAngle has same role as storeDx or storeDy. rotateDirection is a vector where rotation is done. storeRotateDirection has same role as storeAngle, for successive rotation. For natural successive rotation, I needed to use quaternions [1]. dz, storeDz and zoom, storeZoom are used for zooming. isOrtho is used for checking what current view is. mouseX, mouseY, and mouseZ are used for recording mouse location when it’s clicked. mouseLeft, mouseMiddle, and mouseRight are used for checking whether they are pressed.

1. init function

Typical color and depth clearing and enabling depth test and normalization of normal buffer are done firstly. Then, I used GenBuffers, BindBuffer, and BufferData functions to create buffer objects for vertex buffer, normal buffer, and index buffer [2]. Next is light settings. I used 2 lights, one is at right and the other is at left side. Initial polygon mode and shade model are set as following, and initial view setting was done.

1. renderScene function

At first I cleared color buffer and depth buffer. For panning, I translated the model (storeDx + dx) in x direction and (storeDy + dy) in y direction. dx and dy are updated according to mouse motion, and storeDx and storeDy are previous dx and dy and it was stored when mouse middle button was released.  
For rotation, firstly I translated the model to the eye location, rotate it storeAngle radians so that the view is same as previous rotated view. Then I translated the model to its original position. The same thing was done to apply new rotation according to mouse motion.  
For zooming, if the current view is orthogonal view, I scaled it following similar method as rotation, except scaling was used instead of rotate. If the current view is perspective view, camera was moved in z direction according to mouse motion. Similarly, storeDz is used to recover previous view that is zoomed before.  
Finally, I enabled client states for vertex buffer, normal buffer and bind them, and draw them using index buffer. Lastly I swapped buffers so that current working buffer is drawn in the screen.

1. keyboard function

I used q, or Q, or ESC for exiting the program. O or o are used to change the view to orthogonal view. I changed several global variables as their initial values. P or p is used to change the view to perspective view. Again I changed the global values to their initial values except the camera position so that the view is properly set. Actually I optimized the camera position and frustum range for bunny model. I’m sorry that I didn’t check for other models. W and w keys are used to convert to wireframe. F or f keys are used to convert to flat rendering. S or s keys are used to convert to smooth rendering.

1. mouseButton function

Let’s start from left button. It’s for rotation. When it’s pressed, I calculated its normalized x and y coordinates. Also, I want to mention that I used circle-hyperbola model to implement trackball [3]. Anyway, I calculated z coordinate and set mouseLeft as true. When left button is released, I used a new quaternion to combine previous rotation and present rotation using two quaternions from mouseMotion function. And I made p2 same as p3 for successive combining and made q1 as its initial value for correct rotation. Then, by using properties of quaternion, I calculated the angle of combined rotation and rotation vector of combined rotation. mouseLeft was set false.  
Next is middle button. It’s for translation. When the middle button is pressed, I recorded its initial position. When its released, I accumulated the distance mouse moved for successive translation.  
Last one is right button, and it’s for zoom. When it’s pressed, I recorded its y coordinate. When it’s released, I accumulated change in y coordinate and previous zoom factor is multiplied by new zoom factor which is from mouseMotion function.

1. mouseMotion function

I used mouseLeft, mouseMiddle, mouseRight flags to check whether they are pressed or not. If left button is pressed, I calculated mouse cursor’s position and made 2 vectors to calculate the rotation angle and rotation direction. Then I used 2 quaternions for successive rotation.  
If middle button is pressed, I calculated relative distance of mouse movement. They are used in renderScene function to translate. 0.0005 is scale factor to prevent too fast movement.  
If right button is pressed, I calculated the ratio of current mouse y coordinate and previous y coordinate. It is used in renderScene function to zoom. It was for orthogonal view. For perspective view, I calculated relative y coordinate. It is used in renderScene function.

1. Result

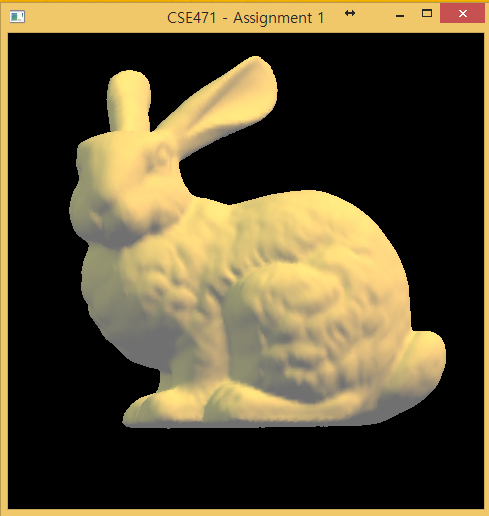


Figure 1. Orthogonal view



Figure 2. Perspective view

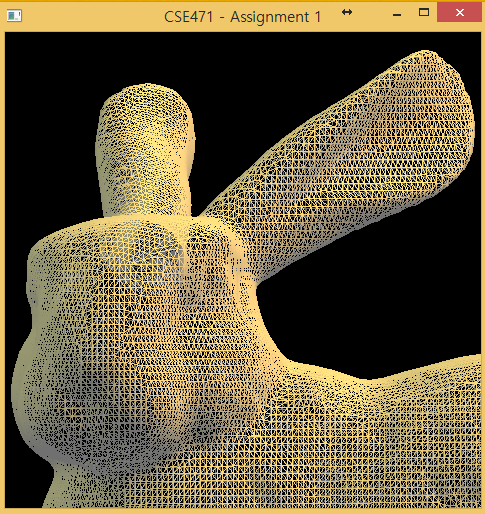


Figure 3. Wireframe, zoomed

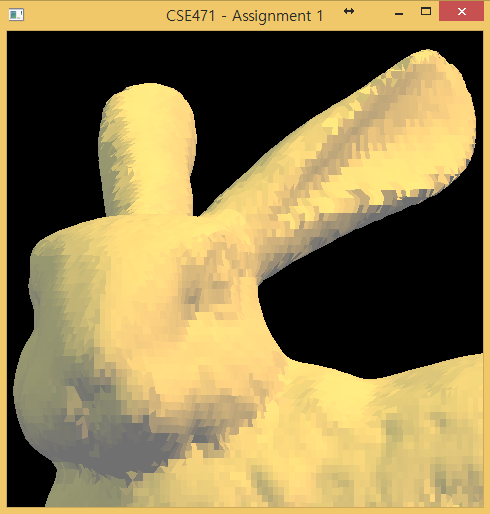


Figure 4. Flat rendering, zoomed



Figure 5. Smooth rendering, zoomed

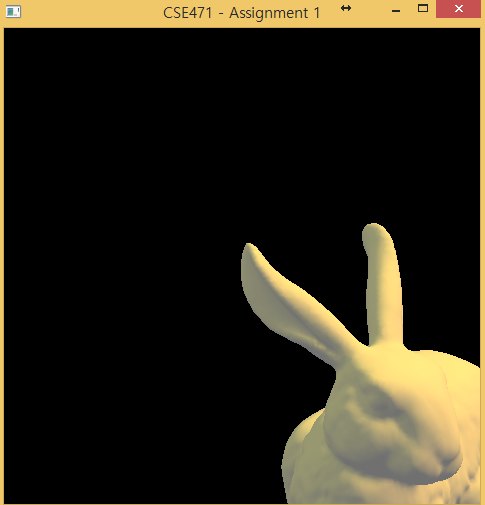


Figure 6. Rotated and translated

1. Conclusion

In assignment 1, I practiced basic openGL functions such as vertex buffer objects, orthogonal/perspective projection, flat/smooth/wireframe rendering, keyboard callbacks, light sources, and virtual trackball. There are some weird things remained like zooming in perspective view and I’m still a little confused about several coordinates. But, anyway, every features required for this assignment are implemented.

I found a bug, when I clicked inside the window and outside window alternatively, the model suddenly disappeared. I couldn’t figure out what happened.

1. Reference

[1] <http://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation>

[2] http://openglbook.com/chapter-3-index-buffer-objects-and-primitive-types.html

[3] https://www.opengl.org/wiki/Object\_Mouse\_Trackball