Ritesh Sharma

1/18/2015

Project #3

CS 551

OREGON STATE UNIVERSITY

This was one of the interesting and difficult of all the project I worked on for this class. It took me a while to understand the concept of Z-buffer and Shading model. Since I implemented most of the routine in my last project, it was a continuation of the same. This time I had to implement z-buffer rendering. To fully understand the concept of z- and think of the data structure buffering it took me almost 2 hours. The steps for rendering using Z-buffer is as follows:

Step 1: Grab the vertices of a polygon
Step 2: Calculate the bounding box for the object made up of polygons
Step 3: Loop through the bounding box and take one pixel at a time
Step 4: Check whether the pixel lies inside the polygon, if it is interpolate Z values
Step 5: Check the interpolated Z values with the Z buffer set by the user, if it is less than value of Z buffer draw the pixel with interpolated Z values on the screen

Inorder to check whether the pixel lies inside the triangle of not, I have used barycentric interpolation approach. In this approach, we divide the triangle taking centroid into three triangles of equal area initially. Then again the area is calculated using new pixel and checked with the sum of the area of all the triangles. If the area is greater than or less than 1, then the pixel lies outside the triangle.

To implement step1-step5 given above took me 2 hours. It was straightforward to implement and I was able to reuse my old code from previous project.

The most difficult task for this project was to understand the shading model. After going through the class lecture and the book by peter shirley, I had a fair enough idea to implement the shading model for this project. It took me 2 days to understand the concepts and to think about the datastructure and come up with an efficient algorithm. From the lecture, I realised that the illumination model is based upon a relatively simple equation:

$$c = c_r * (c_a + c_l (n \cdot l)) + c_l * c_p * (h \cdot n)^p$$

where,

c	The final color of the pixel.
c_r	The color and intensity that the surface reflects, commonly called the diffuse color.
c_a	The color and intensity of ambient light in the room.
c_l	The color and intensity of a bright light, such as a point or directional lights.
c_p	The color and intensity of highlights that the surface reflects, or the specular color.
p	The fall-off rate of the highlight, or the shininess.
n	The normal of the surface at that point.
l	A vector pointing from the surface to the light generating c_l.
e	A vector pointing from the surface to the camera
h	A vector pointing from the surface that is halfway between e and l. $\| e + l \|$

The first term, $c_r * c_a$, takes the diffuse surface color and multiplies it by the ambient term to generate the ambient color of the object before any light is calculated from other bright light sources.

The second term, $c_r * c_l (n \cdot l)$, is calculating the amount of bright light hitting the surface based upon the angle between the normal of the surface and the angle from the surface to the bright light. When the angle is small, there is large amount of light reflected from the surface where as when the angle is large, there is very small amount of light reflected from the surface.

The final term, $c_l * c_p * (h \cdot n)^p$, determines the highlight on the surface. When looking at a surface, if the vector from the surface to the eye directly matches the angle of the vector from the surface to the light reflected about the normal, then we typically see some sort of glare or highlight.

Flat Shading is a type of shading when same color is used to color every pixel on the face of a polygon where as Smooth shading is a type of shading when linear interpolation between the vertices is used to color the face of the polygon. In Smooth shading, normals of each vertex is precomputed by averaging the normals of each polgon that connects to that vertex. Smooth Shading is also known as Gourad Shading.

Implementation of the above equation took me an entire day to work out the physics behing the phong shading model. To come up with efficient data structure took me 2 hours. For the implementation I first took single light source for both point and directional light and then moved to muliple light source. I also tested my results on different models which were downloaded from http://www.prinmath.com/csci5229/OBJ/index.html

The results are given below:

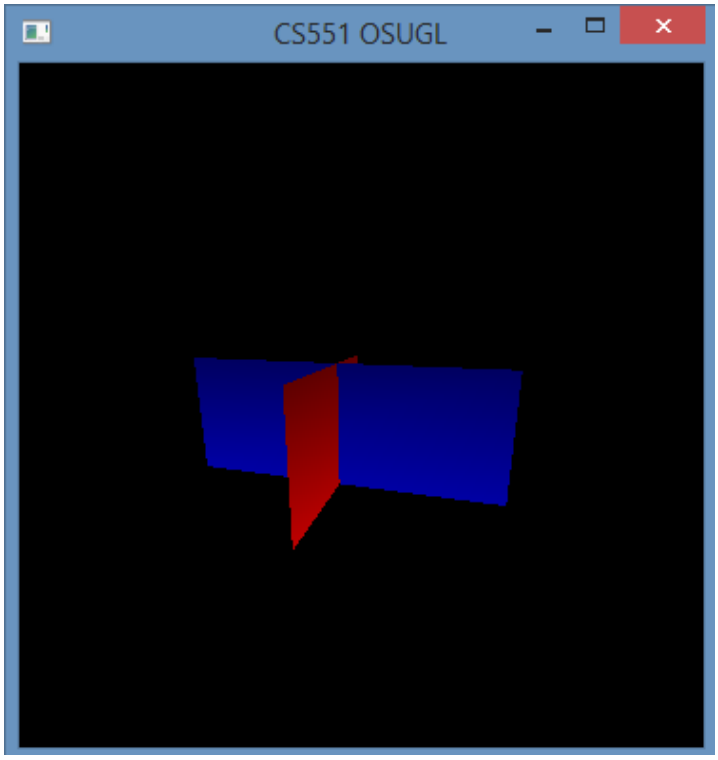

Figure 1. Test case 1 with point light source

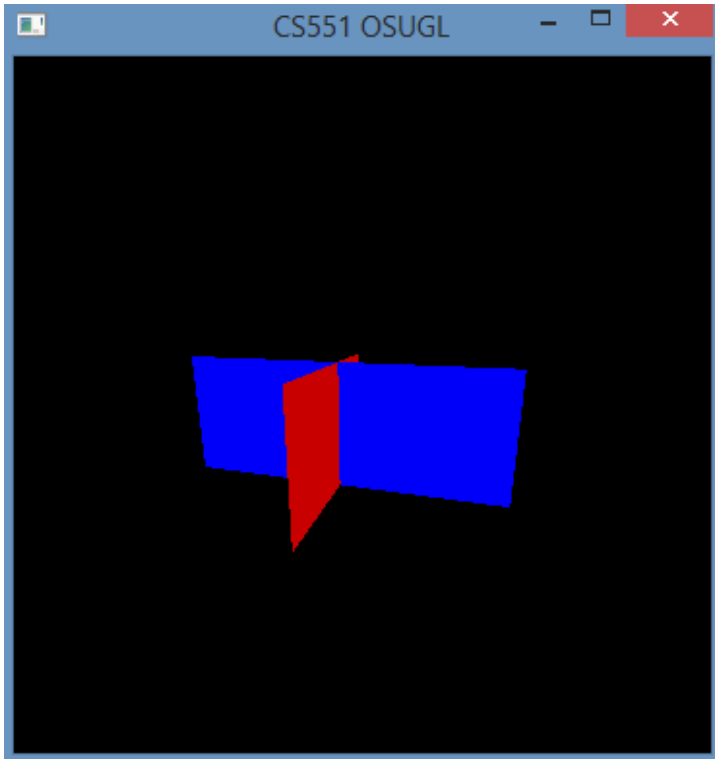

Figure 2. Testcase 1 with directional light source

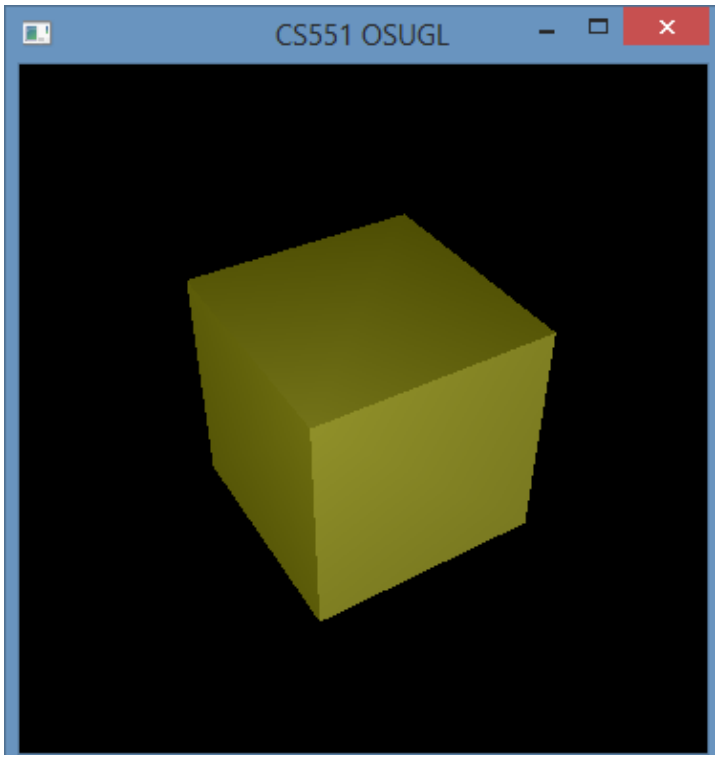

Figure 3. Testcase 2 with single light source and smooth shading

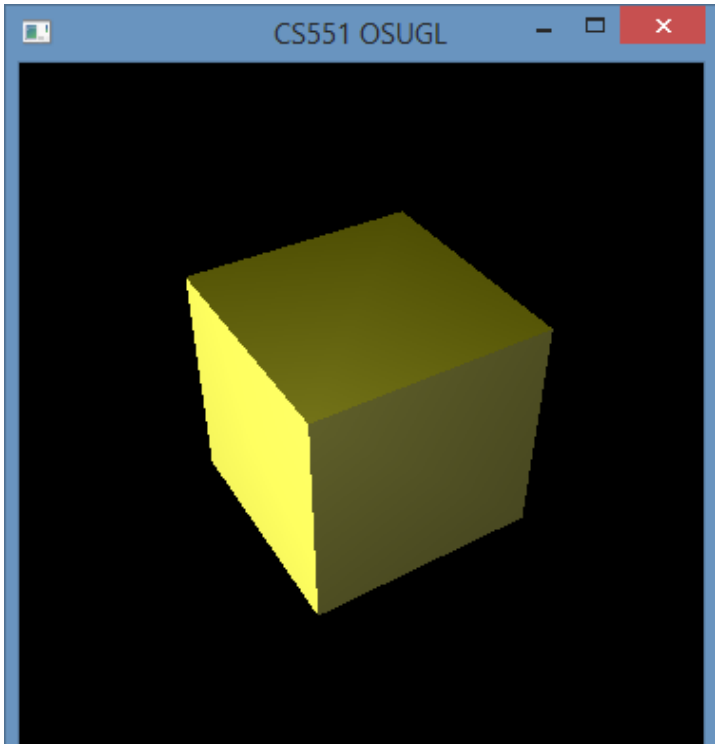

Figure 4. Testcase 2 with two light source and smooth shading

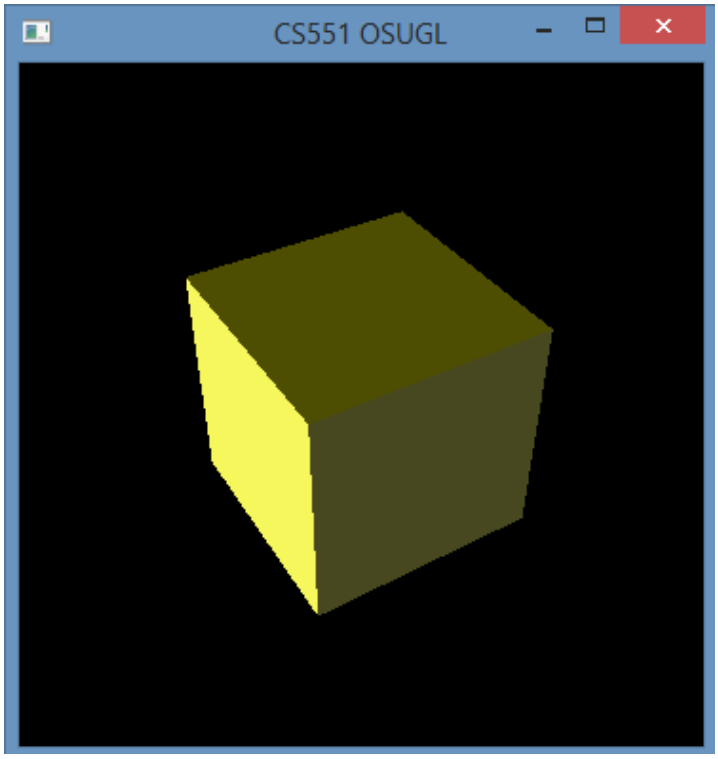

Figure 5. Testcase 2 with two light source and flat shading

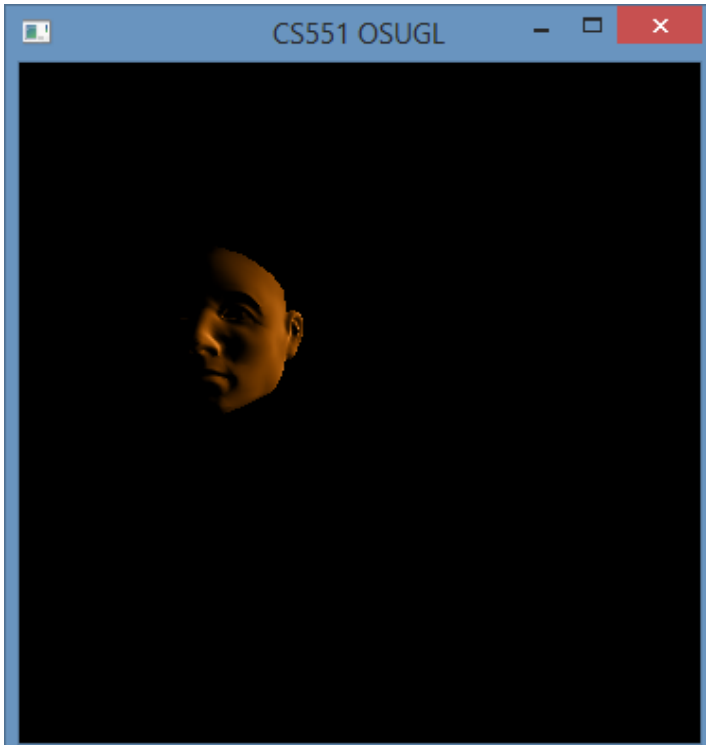

Figure 6. Testcase 3 with single light source and smooth shading

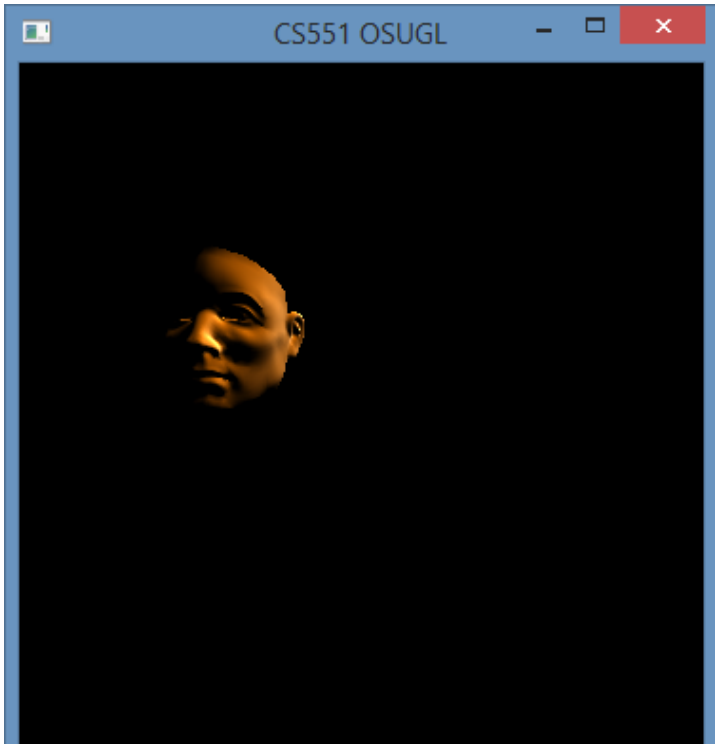

Figure 7. Testcase 3 with multiple light source and smooth shading

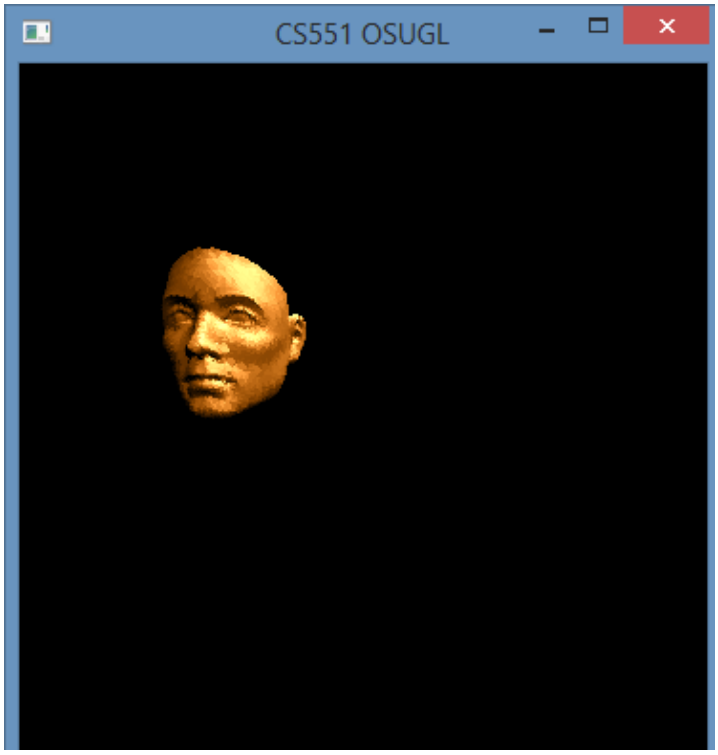

Figure 8. Testcase 3 with multiple light source and flat shading

I was amazed to see the artifacts when I used flat shading. It can be seen in fig. 8 above. But after going over the concepts again I realized that the farthest I move from the object, the artifacts start reducing and I get the better color on the object.

As far as algorithms and data structures, I think there are more efficient ways to store the data than the stack which I created for the matrices and vertices. Also generating the intermediate values using dot product and other multiplication can be reduced to make the system to work faster.