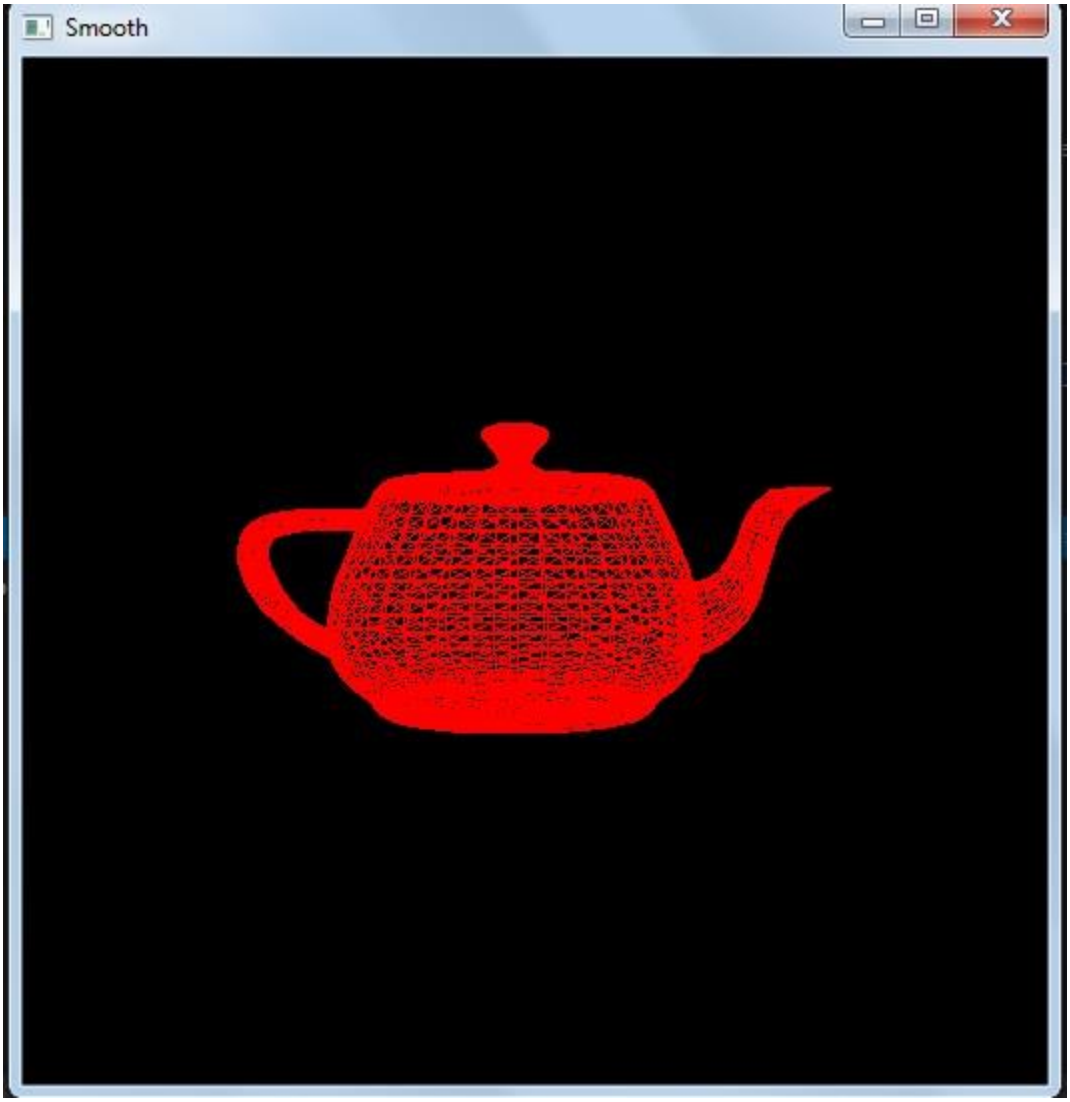
Report

Name: Garima Singh

UFID: 5197-5877

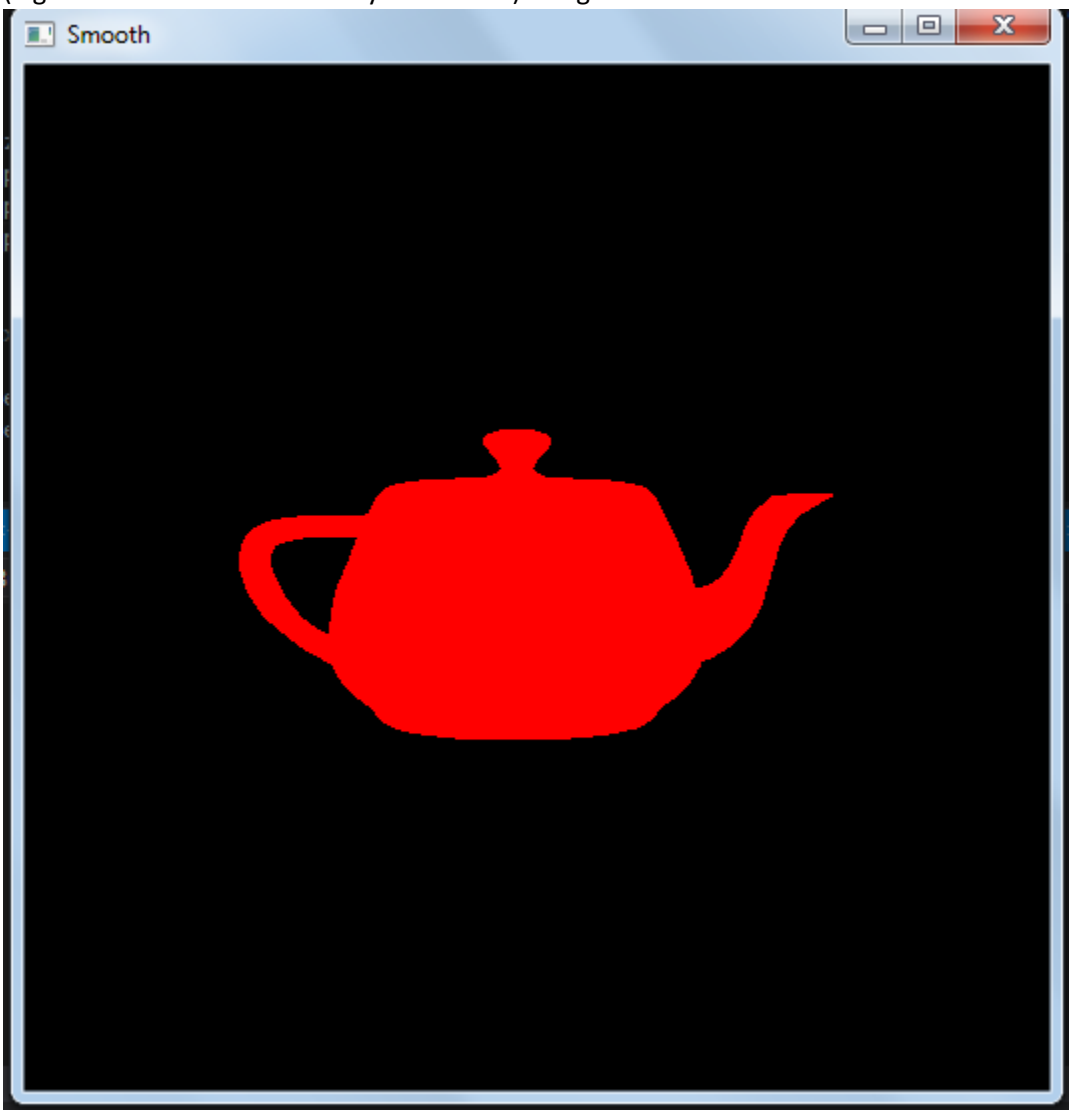
1) The first step I took was to extract the vertex information from the GLMmodel object in myDisplay() . I iterated over all groups in the model and further for every triangle in that group and used gluProject to get screen co-ordinates and z values for each vertex. I checked if this data was correct by using glBegin(GL_TRIANGLES) to draw all the triangles in the .obj files using black color. Using teapot.obj for testing (shading was clearly visible in it)

2) The next step was to implement line drawing algorithm. To test, I drew the triangles in the object files using my DrawLine1() function with red color. Used glDrawPIxels to draw by setting pixel values in an array win DrawLine1() (algorithm source: Wikipedia).

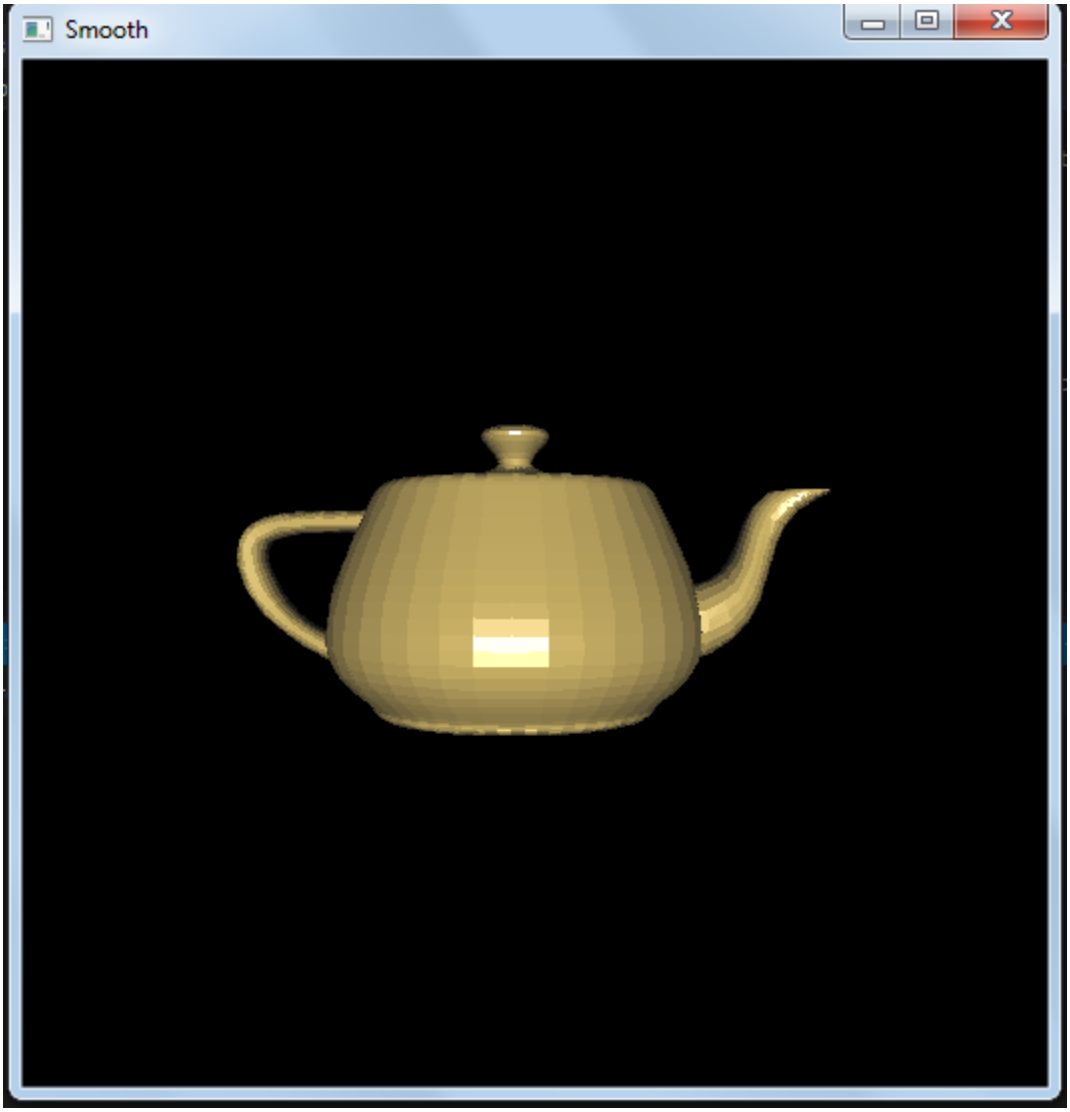


3) The next step was to implement the scan line algorithm. I faced some problems here because earlier I was implementing scan-line by drawing individual lines to fill the triangle. This would leave some gaps in the image and mess up the z- buffer values at those points. So finally I implemented the algorithm(scanFill3()) which calculates the barycentric co-ordinates of points within a bounding box to determine if it's in the triangle and then color it (using red as of now).

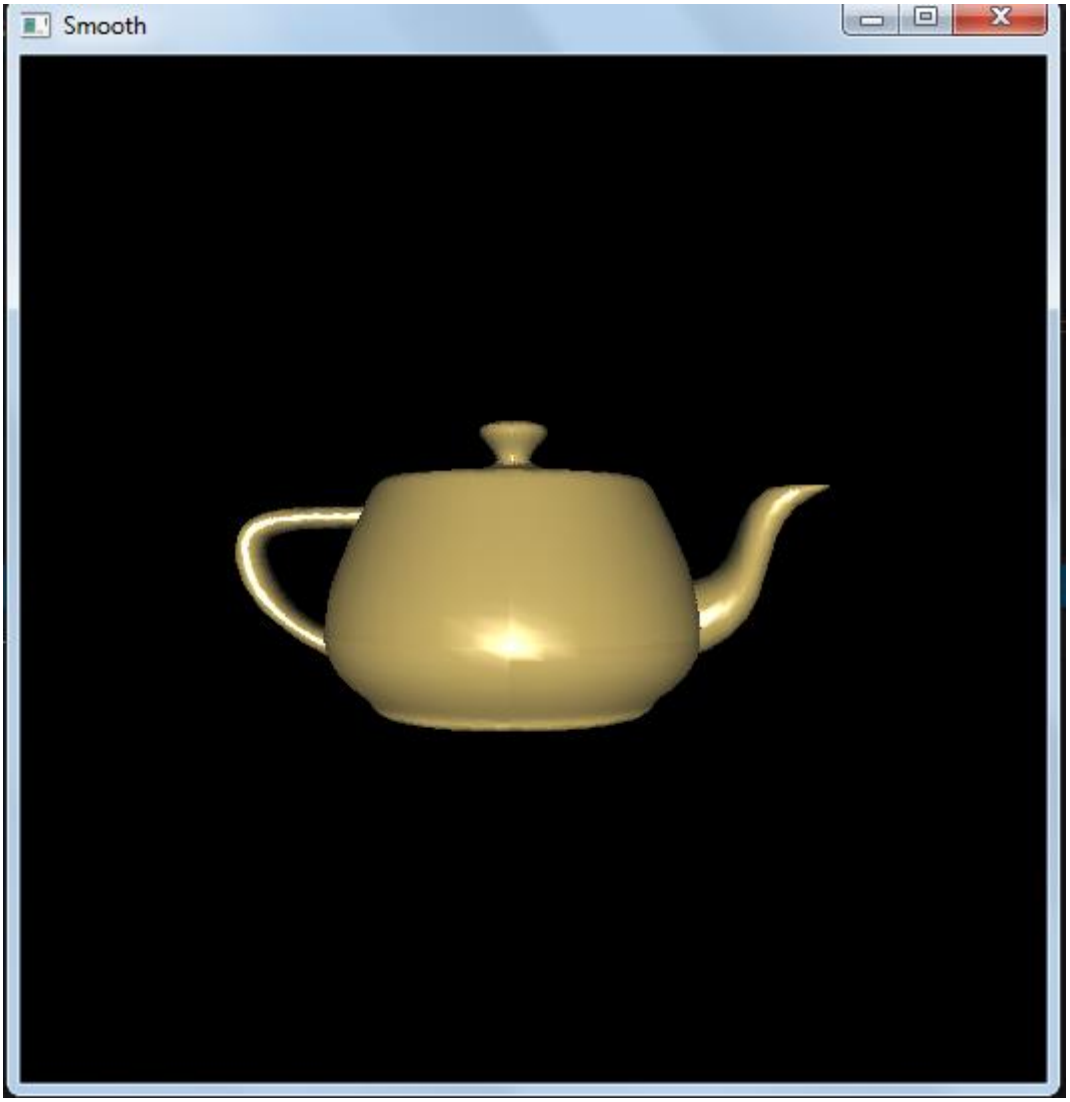
(algorithm source: Peter Shirley's textbook) using red color



4) Next was to implement z buffer which was easy to calculate because I already had barycentric co-ordinates for each point calculated in the scan –line algorithm. I messed up here because I was storing the z values in Int which would truncate it to 0 always which was giving me a transparent sort of effect. Rectified it to float later. Then I extracted material information(which is same for all triangles per group) and the facet normal per triangle and implemented flat shading (light direction and viewing direction are both (0,0,1) in smooth)



5) Finally I implemented Gourad shading by extracting the 3 vertex normal of the triangle and calculating color information for every vertex. I then pass these colors to the scanFill3 function which already has the barycentric co-ordinates for every point in the triangle. The color at every pixel is then calculated and colored appropriately. There is a significant smoothing effect with Gourad shading.



6) Implemented rotation as implemented in Nate Robbins code.

7) Updated normals by multiplying them with the model view matrix to adjust normal directions as they are rotated.

8) Put the functionality to toggle between Nate Robbins algorithm by pressing “y”

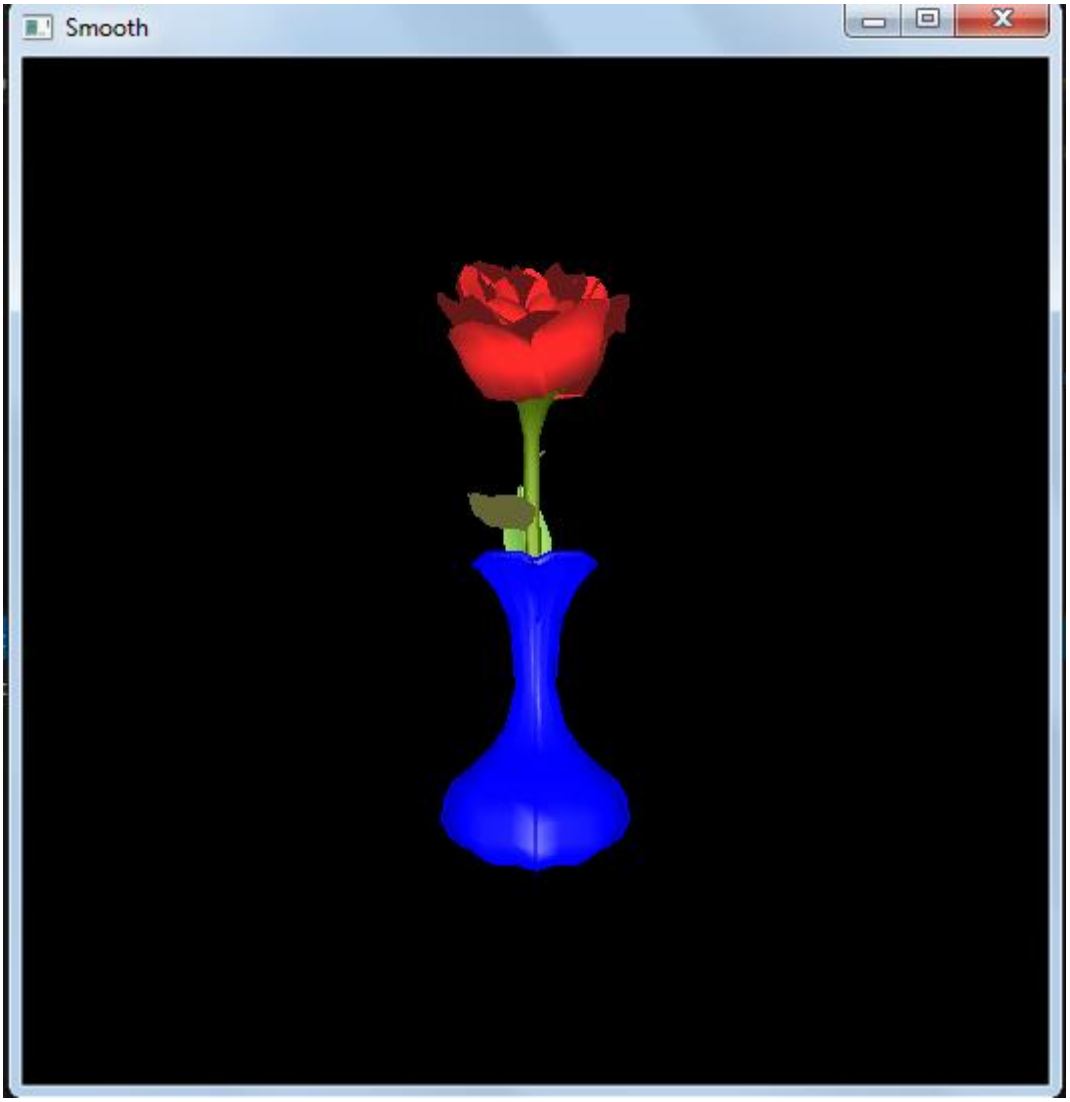
9) Put the functionality to toggle between flat shading and Gourad shading by pressing “g”.

10) To compare my implementation and Nate Robbins’, both looked similar to me. My rendering was fast enough (because of using bounding box) but during rotation it would take a little while if the no. of triangles to be rendered are a lot. Also, Nate Robbins’ code uses hardware to accelerate a lot of calculations whereas I am doing it in software.

11) Some other objects:

- al.obj

- rose+vase.obj


Smooth