Comp480 – Spring 2008

Programming Assignment #2

Objective: The purpose of this assignment is to become familiar with the details of the rendering pipeline. A good understanding of the rendering pipeline is a great help in writing and debugging programs that use OpenGL and other graphics APIs. In this assignment you will implement the basic functionality required to rasterize polygons, as well as parameter interpolation for lighting and texturing. You will be provided with a framework in which you will write your implementation and a program that you can use to test it.

Part-1 requirements

Due 4/21/08 (before 12:00pm, mid-noon)

We will test the following functions with the included scenes that can be seen when we click "F1", "F2", "F3", and "F4". We will use the built-in software-version lighting function.

1. Clip polygons against view frustum in clip coordinates in the ClipPolygon() method. Use the Sutherland-Hodgman to clip against one frustum plane at a time. Remember to interpolate vertex attributes to the new point on a clipped edge.
2. Modify TriangulatePolygon() to turn the clipped polygons into triangles. Rasterization is easier to implement if the inputs are always triangles. You may assume that the input polygons are always convex.
3. Modify RasterizeTriangle() to do the following:
   - Triangle rasterization using edge equations. Evaluate the full edge equations at each pixel in the bounding box of the primitive.
   - Perform linear interpolation for colors assigned to each vertex.
   - Z-buffering.
   - Perform back-face culling and toggle back-face culling with a key map "B"

Part-2 requirements

Due 4/30/08 (before 11:59pm, right before the mid-night)

1. Implement the OpenGL lighting model including ambient, diffuse, and specular lighting. Modify the ComputeLighting() function to compute lighting at the vertices and store it in the vertex color.
2. Support texture mapping using nearest neighbor filtering. Use a simple scene (e.g., the scene that can be displayed by a key map of "F1") showing a textured primitive to test your implementation.
3. Provide linear interpolation and perspective-correct interpolation of parameter values. Provide toggling between the two schemes by typing a key "i".

Policies: Everyone must turn in their own assignment. You can collaborate with others, but any work that you turn in should be your own. Turn in your work by emailing an archived and compressed version of it (source and executable) to the TA (TaeJoon Kim) along with instructions for running your code.