Comp480

Objective: The purpose of these two assignments 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.

Programming Assignment #5

Due Apr.-15 (Wed.) (before 11:59pm)

The skeleton codes can show four different models when you click we click “F1”, “F2”, “F3”, and “F4”. The codes also support both hardware-based rendering using OpenGL and software-based rendering. By clicking “w”, you can see the wire-frame of the rendered images. If you click the space bar, it switches between the hardware or software rendering. However, many functions of the software-based rendering are not implemented. Note that the viewport area is shown in gray or dark violet when the hardware or software-based rendering is enabled respectively.
Also, you can toggle the lighting, clipping, triangulation, rasterization by clicking “1”, “2”, “3”, and “4” respectively.

We use the built-in software-version lighting function. So, start with your programming with the following configuration: “Software rendering:on - L(1):on C(2):off T(3):off R(4):off”

You need to implement the following functions in the software based rendering.

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. Then, turn on the clipping in your software rendering.
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. Then, turn on the triangulation in your software rendering.
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”
    - 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.
    - Then, turn on the rasterization in your software rendering.

Note that you can check your results with ones generated by OpenGL. This can be done by turning off each module in the software rendering.

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 along with instructions for running your code.

Programming Assignment #6

Due Apr.-27 (Mon.) (before 11:59pm)

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. Add a texture to a simple scene displayed when clicking the “F1”. Support texture mapping using nearest neighbor filtering.
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 along with instructions for running your code.