Comp380

Programming Assignment #5

Due May.-31 (Fri.) (before 11:59pm)

Rep. TA: Sehwan Kim, (cs380ta@gmail.com)

Objective: The purpose of these two assignments is to let you 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 these assignments, 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.

Developing environment: Usage of Windows OS and Visual Studio (2008 or higher) is mandatory

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
     - You can check whether your Z-buffering implementation works well or not by moving the rectangle in the scene with F2 toward Z-direction (by pressing page-up, down keys)
   - Perform back-face culling and toggle back-face culling with a key map "B"
     - You should print the number of triangles actually rendered to the console to check whether your back-face culling implementation works correctly or not.
   - 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.

Tips

1. Your software renderer should work on the perspective mode with key 'p', and object translations (arrow keys), and rotations ('[' , ']')

Deliveries:

1) Binary and source codes of your solutions (Include a README.txt that specifies the files you made/changed)
   - Please change the file extension of your binary from ‘exe’ to ‘aaa’ or something. If not, your submission will be sent back due to gmail policy.
2) Provide a readme file that contains a) your compiler & development environments (e.g., MS Visual 10), and b) your OS that tests your binary files
3) Submit your files by sending them to TA, cs380ta@gmail.com

Policies: Everyone must turn in their own assignment. You can collaborate with others, but any work that you turn in should be your own.