

# Comp770 – Spring 2007

Programming Assignment #2

Due 3/05/07 (before 11:59:59pm)

**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.

## Minimal requirements (worth 80%):

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. 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.
3. 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.
4. 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. Implement a faster scheme that performs incremental updates and that doesn't visit every pixel in the bounding box. Compare timings for both schemes. How much speedup did you get?
  - Linear interpolation and perspective-correct interpolation of parameter values. Provide controls for toggling between the two schemes.
  - Z-buffering.
  - Texture mapping using nearest neighbor filtering. Implement both `GL_CLAMP` and `GL_REPEAT` wrap modes. Modify the test program to include a textured primitive with texture coordinates outside the  $[0,1)$  range to test your implementation.
5. Create a simple scene on which to demonstrate your rendering. You can either extend the test program or create a new one. The scene doesn't have to be complex. A collection of polygons is fine. It just needs to show off the effects that you have implemented.
6. Provide a short write-up describing what you implemented and the approaches you took, as well as answering any of the questions related to items in this specification that you implemented.

## Extras:

- 1) (5 pts) Additional user clip planes. OpenGL has support for user clip planes that can be used to cut away part of an object to reveal its interior. Provide controls to rotate and translate your clip plane.
- 2) (5 pts) Add back face culling. The culling can be performed in `ClipPolygon()`, `TriangulatePolygon()`, or `RasterizeTriangle()`. Which function did you use? Why? What are the advantages and disadvantages of applying the culling in each method?
- 3) (5 pts) Per pixel lighting. Interpolate normals across the triangle. Apply the lighting equation to each pixel.
- 4) (5 pts) Bilinear interpolation. Improve texture mapping to use bilinear interpolation for texture look-ups
- 5) (5 pts) Mipmapping. The test program already generates mipmaps and `GLRenderer` reads them in. Extend your rasterizer to use mipmaps for texture filtering. Provide a special mode that displays colors representing the level in the mip map rather than the texture itself. This can be useful for debugging.
- 6) (10 pts) Summed-area tables. OpenGL does not support summed-area tables. Use the texture data provided by `GLRenderer` to create a summed-area table. Estimate the size of the bounding rectangle of the pixel in texture space similarly to mip maps. Provide a special mode that uses a color coding to show the size of the projected pixel in texture space. This can be useful for debugging.
- 7) (5 pts) Blending. OpenGL blending operations combine the incoming pixel with the pixel already in the frame buffer. The blending is controlled with the alpha value (read the documentation for more details). Render a small mesh with blending to make sure that you don't get double hitting.
- 8) (Variable) Some idea of your own. Be sure to consult with the instructor.

Note: Some of these functions that are also implemented in OpenGL (clip planes, back face culling, and blending). It might be useful to implement the functions first in OpenGL. For example, you can use `glClipPlane()` to `glEnable(GL_CLIP_PLANEi)` to get clip planes working. Then you can override the `GetState()` to read the clip plane state to be used in your own version. This can make debugging much easier because you know what your results should look like.

**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 instructor along with instructions for running your code.