Spring ’11 CIS 441/541 Assignment 3 – 120/100 points possible – Due Friday, 6-3, 11:59 PM

The goal of this assignment is to provide experience rendering curved surfaces in 3D and experience with shader-based rendering. The exercises from the book are indented to provide practice working through problems involving texture mapping, shaders, and 3D curves and surfaces.


4. [5] Exercise 12.7, page 650. Answer in the form of a matrix equation which transforms interpolating control points q to Bezier control points p (e.g., $p = TRq$).

5. [10] Exercise 12.13, page 651. Answer in the form of pseudocode which would determine if any surface is planar based on its control points. Make no assumptions about the coordinate values of the control points.

6. [10] Create assignment3.c with an int main() function which creates a GLUT window with title “Assignment 3” and dimensions 800x600. GL capabilities should be identical to the previous assignment except when stated otherwise (i.e., use your previous assignment as a template for this assignment, but remove all previous rendering operations).

7. [10] Use glMap2f() and glEvalCoord2f() to render a Bezier surface patch using a set of 4x4 control points spaced uniformly between -0.5 and 0.5 in both the x and y directions. The z coordinates should initially be 0.0, but create an idle behavior to animate these coordinates in some interesting way (but do not alter the x and y coordinates). Create a material for your surface and enable GL_AUTO_NORMAL so that glEvalCoord2f() generates normals along with vertices.

8. [5] Add menu items to select a glPolygonMode() of either GL_FILL or GL_LINES and map the ‘f’ key to cycle between these options.

9. [15] Add menu items to specify 0, 1, 2, 3, 4, or 5 subdivisions for the surface. For 0 subdivisions, 4x4 distinct vertices should be used for rendering; 7x7 distinct vertices should be used for 1 subdivision, 13x13 distinct vertices should be used for 2 subdivisions, etc. Also map the ‘s’ key to cycle through the subdivisions options.

10. [30] Create a vertex shader which calculates and passes the normal, light vector, viewer vector, and half-angle vector to the pixel shader (see examples in book). Create a fragment shader which calculates a pixel color based on the input vectors and light/material properties.
using the Phong lighting equation (see examples in book). Light your surface using your vertex and fragment shaders. Add menu items to select Fixed-Function or Shader lighting and map the ‘I’ key to cycle between these options.

Note that you may need to do some additional system configuration to get your compiler to compile against OpenGL 2.1+ (which is when GLSL was introduced). This doesn’t seem to be an issue on OSX, but I needed to use the 32-bit version of GLEW (see link on course links page) to get this working on 64-bit Windows 7. The 32/64-bit discrepancy is interesting, but makes sense as I’m compiling a 32-bit OpenGL application with MSVC.

541 Only: Add timing functionality which periodically prints the average Frames Per Second (FPS) to the console. Write a short paper comparing the performance of fixed-function and shader lighting with respect to distinct vertex count. Support your observations and use a table and graph to illustrate your data.

The final result should look like:
11. [+20] Add a recursive function which renders the Bezier surface by explicit subdivision of the control points (see examples in book). That is, your function should use glVertex3f() and glNormal3f() to specify vertices instead of glMap2f() and glEvalCoord2f(). Add menu items to select between Hardware and Software surface rendering and map the ‘r’ key to cycle between these options.

Zip your source file(s) and problem-set solution document and upload the .zip file to Blackboard.