Graphics Programming II – Assignment #1
Due on 01/26/2010

In this assignment, you will tesselate and render a set of Bézier patches. The patches are to be tesselated to a triangle mesh consisting of NxN vertexes with per-vertex positions and normals. In a future assignment per-vertex tangents will also be needed. The tesselation parameter, N, should be configurable at compile-time or run-time (your choice).

A single fixed-position light source and simple per-vertex lighting should be used. Locking the light at some known position, such as the camera position, should simplify the shaders needed for this assignment. Per-fragment lighting and complex light sources will be the topic of future assignments.

At the very least, the patch object should rotate around the three principle axes so that all orientations can be viewed. A more sophisticated user interface, such as using the arcball routines in the provided GLU3 library, would be preferable.

There are three main programming elements to this exercise:

- Evaluation of positions and normals on a Bézier surface.
- Orderly generation of data into vertex buffers.
- Generation of indices for rendering with `glDrawElements`.

For simplicity the program for this assignment tesselates all patches to the same number of triangles. In addition to the program, write a couple short paragraphs that answer the following questions:

- Why is this fixed tesselation inefficient?
- What parameters could be taken into consideration to select different levels of tesselation for some patches?
- Why are those parameters good choices?

Patch data will be provided in class and on the course website.

Criteria	Excellent	Good	Satisfactory	Unacceptable
Completion	Program correctly implements all required elements in a manner that is readily apparent when the program is executed. User interface is complete and responsive to input. Program documents user interface functionality.	Program implements all required elements, but some elements may not function correctly. User interface is complete and responsive to input.	Program implements most required elements. Some of the implemented elements may not function correctly. User interface is complete and responsive to input.	Many required elements are missing. User interface is incomplete or is not responsive to input.
Correctness	Program executes without errors. Program handles all special cases. Program contains error checking code.	Program executes without errors. Program handles most special cases.	Program executes without errors. Program handles some special cases.	Program does not execute due to errors. Little or no error checking code included.
Efficiency	Program uses solution that is easy to understand and maintain. Programmer has analysed many alternate solutions and has chosen the most efficient. Programmer has included the reasons for the solution chosen.	Program uses an efficient and easy to follow solution (i.e., no confusing tricks). Programmer has considered alternate solution and has chosen the most efficient.	Program uses a logical solution that is easy to follow, but it is not the most efficient. Programmer has considered alternate solutions.	Program uses a difficult and inefficient solution. Programmer has not considered alternate solutions.
Presentation & Organization	Program code is formatted in a consistent manner. Variables, functions, and data structures are named in a logical, consistent manner. Use of white space improves code readability.	Program code is formatted in mostly consistent with occasional inconsistencies. Variables, functions, and data structures are named in a logical, mostly consistent manner. Use of white space neither helps or hurts code reability.	Program code is formatted with multiple styles. Variables, functions, and data structures are named in a logical but inconsistent manner. Use of white space neither helps or hurts code reability.	Program code is formatted in an inconsistent manner. Variables, functions, and data structures are poorly named. Use of white space hurts code reability.
Documentation	Code clearly and effectively documented including descriptions of all global variables and all non-obvious local variables. The specific purpose of each data type is noted. The specific purpose of each function is noted, as are the input requirements and output results.	Code documented including descriptions of most global variables and most non-obvious local variables. The specific purpose of each data type is noted. The specific purpose of each function is noted, as are the input requirements and output results.	Code documented including descriptions of the most important global variables and the most important local variables. The specific purpose of each data type is noted. The specific purpose of each function is noted.	No useful documentation exists.

This rubric is based loosely on the "Rubric for the Assessment of Computer Programming" used by Queens University (http://educ.queensu.ca/ compsci/assessment/Bauman.html).