Graphics Programming II – Assignment #1 (Surface-space Normal Mapping) Due on 04/22/2009

This assignment consists of two largely separate parts. However, you will likely have to complete the first part before you will be able to implement the second part.

In the first part of the assignment you will implement a function that will generate simple surfaces of revolution. This function should have the following interface:

- Caller supplies a list of 2D points (x-y pairs) that define the outline of the object being revolved. You may assume that this defines a closed loop. That is, you may assume that point 0 and point N-1 have the same position. Your code should validate that at least 3 points are supplied.
- Caller supplies a 4x4 transformation matrix to transform the input points within the X-Y plane before revolving.
- Caller supplies the number of rotational steps. Your code should validate that at least 2 steps are requested.
- Caller supplies pointers to storage for output data. If any data pointer is `NULL`, your code should not generate that piece of data.
- Function generates the following data. 3-element data items (i.e., normals) may be padded to 4-elements. Be sure to document whichever you choose.
  - Positions - X, Y, Z floating point tuples
  - Normals - X, Y, Z floating point tuples
  - Tangents - X, Y, Z floating point tuples
  - Texture coordinates - S and T floating point tuples
  - Elements - Indicies used to draw the triangles.
- Function returns the number of verticies that were or would have been (in the case that all input pointers are `NULL`) generated.

The function should generate a series of triangle strips. You may either generate each rotational segment as a strip or each segment of the input data through the entire rotation as a strip. The triangle strips can be drawn in one of three ways. Extra credit will be given if you implement all three and measure the relative performance of each.

- `glMultiDrawElements` - This is the preferred method on most OpenGL 2.x implementations. This will allow drawing of the entire object with a single draw call.
- `glDrawElements` - This is a more straight forward way of drawing the object. It is probably easier to implement this method first.
- `glDrawElements` with primitive restart - This is the prefered method on OpenGL 3.0 or on Nvidia hardware. In this mode the lists of elements for each strip are separated by the index ~0. Primitive restart mode is then enabled (via `glEnable` with `GL_PRIMITIVE_RESTART_NV`) and the restart index must be set (via `glPrimitiveRestartIndexNV`). See the extension specification for `GL_NV_primitive_restart` for more details.

For the second part of the assignment you will render a normal-mapped object with a single point light source. The light source should orbit the object, and the object should viewable from multiple directions. The object can either rotate around its own center, or you can implement controls so that the user can change the view angle. Per-fragment lighting should be used with either Phong's or Blinn's lighting model.

The object should have the following textures of your choosing:

- Diffuse color texture map
- Normal map. Do *not* using mipmapping on the normal map unless custom mipmap generation is performed. Otherwise the pre-filtered normals will no longer have unit length.

- Extra credit will be given if a specular (aka gloss) map is also used. This is a texture that modifies the specular exponent of the lighting equation.

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).