CG Programming I – Assignment #4 (per-fragment lighting and multi-texture) Due on 11/27/2007

For this assignment, you will implement per-fragment lighting using texture combiners. This may be done either as a stand-along program or as an addition to the 3D world of assigment #3.

- Draw at least one object with per-fragment specular lighting.
  - Calculate per-vertex bitangent and normal vectors.
  - Draw the normal vector and bitangent vector.
    - It is sufficient to draw a simple line for each vector from the vertex. Each should be drawn in a different color. While this may seem like a silly requirement, it is a powerful debugging aid.
  - Calculate the per-vertex "surface-space" transformation.
    - This implicitly requires that the normal and bitangent vectors be transformed to world-space. Since these are vectors, the world-space translation is *not* applied, only the upper 3x3 portion of the matrix. Refer to the book for more details.
  - Calculate the per-vertex H vector.
    - This implicitly requires that the vertex positions and the positions of the light be transformed to world-space.
  - Draw the H vector.
    - It is sufficient to draw a simple line for this vector from the vertex. It should be drawn in a different color from the normal or bitangent. I cannot over emphasise how useful this is as a debugging aid.
  - Transform the per-vertex H vector to surface space.
  - Specify the transformed H vector as the vertex's color.
    - Remember: the components of the H vector have the range [-1, 1], but color components have the range [0, 1].
    - It should be apparent that traditional OpenGL lighting is *disabled*.
  - Create a normal map texture.
    - Either find a suitable normal map on the Internet *or* create a 1x1 texture with the color (0.5, 0.5, 1.0).
  - Configure the texture combiners to calculate N.H.
    - As described, this will use only one texture stage.
  - Create a gloss map.
    - Either find (or draw) a suitable gloss map on the Internet *or* create a 1x1 texture with the desired specular color.
  - Configure the texture combiners modulate the result of N.H with the value from the gloss map.
- The user should either be able to navigate around the object (i.e., the object is in the 3D world from assignment #3) or the object should rotate around its center.

The following inputs must be implemented. In addition, the program must, in some way, communicate to the user how to use it.

- Escape must terminate the program.
- Inputs must be implemented for movement (as in assignment #3) or an input must be implemented to pause the animation of the object.

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).