Graphics Programming II – Assignment #2
Due on 10/27/2010

In this assignment, you will implement per-fragment lighting, normal mapping, and environment mapping. This should be implemented in several steps.

- Modify the vertex shader to emit the light position and the eye position in tangent space. To verify the credibility of the results, emit each as the per-vertex color. What sorts of colors should you expect to see?
- Modify the fragment shader to perform lighting calculations in tangent space. Initially implement just simple diffuse lighting. Once that works, add either Blinn or Phong specular lighting.
- Modify the fragment shader to procedurally generate per-fragment normals. Use the texture coordinates supplied to the vertex shader (in the attribute `uv`) as the parameters to the texture procedure. This means that the `uv` attribute will need to be passed from the vertex shader to the fragment shader in a varying.
- Load a cubic environment map of your choosing. In the fragment shader, calculate the ideal reflection vector (using the GLSL built-in function `reflect`). Use this vector to sample from the environment map. In what coordinate space should this vector be calculated? *Note:* Code will be provided to load textures from disk.
- In the fragment shader, calculate the Fresnel factor (using Schlick's approximation). Use the Fresnel factor to combine the environment map with the calculated lighting value.

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).