CG Programming II – Assignment #2 (enhanced env mapped specular) Due on 02/05/2008

Starting with the code from assignment #1, add the following features:

- Add a normal map to the object. The normal map is to be implemented in surface-space.

  normal maps can either be found on-line by searching or by converting a gray scale "height map" to a normal map using Nvidia's Photoshop plug-in.

  http://developer.nvidia.com/object/photoshop_dds_plugins.html

- In assignment #1 the object rotated around it's center, but not other motion was implemented. Add the ability for the user to move around the object. Fix the point that is viewed at the center of the object, but allow the user free movement using the arrow keys. Using `gluLookAt` is advisable.

- Implement prefiltered specular environment maps.
  - Prefiltered environment maps are to be implemented using framebuffer objects.
  - While the application is running, pressing the 's' should decrease the specular exponent by some small, fixed value. Likewise, pressing 'S' should increase the specular exponent by some small, fixed value. This implies that it must be possible create an initial prefiltered environment at program initialization and update that environment map later. This means that the original, unfiltered environment map and the current prefiltered environment maps must both be kept in memory.
  - The number of samples from the environment used to create the prefiltered environment map should be selectable at compile-time.

- Implement irradiance maps. The irradiance map should be created in a similar manner as the prefiltered environment map.

For extra credit, improve the the quality of the prefiltered environment maps and the irradiance map making multiple passed. In the first pass sample the most important neighbor texels. In the successive passes, sample gradually more distanct neighbor texels until the weight falls below some preset threshold.

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).