CG Programming III – Assignment #2 (shadow textures)
Due on 04/15/2008

In this assignment you will be required to implement shadow textures. To show shadow textures in action, render a scene with *at least* two objects and a single light source. The light source and the objects must move in such a way that object can be both receivers and casters. One way to do this is to have multiple objects arranged in a plane with a light source orbiting them.

- Draw *multiple* objects in the scene. These can either be simple solids (e.g., the torus or sphere) or models loaded from disk.
- Include a single light source in the scene.
- Implement shadows using shadow textures.
  - Render the scene, from the point of view of the light, to the screen. Clear the screen to the color of the light, then draw the objects in black.
  - Render the scene, from the point of view of the light, to a framebuffer object. Draw that framebuffer object to the screen as a single quad. The output of this step *should* be the same as the output of the previous step.
  - Render a single object, from the point of view of the light, to a framebuffer object. Be sure to calculate the view frustum to maximize the object's size in the FBO. Draw that framebuffer object to the screen as a single quad.
  - Using projective texturing, apply the texture generated in the previous step to each object farther away from the light than the object rendered to the texture.
  - Generate a shadow texture for each object in the scene. When rendering an object in the final scene (from the eye's point of view), apply the shadow texture for all other objects. Be sure to apply the near-plane test mentioned in the lecture notes to prevent anti-shadows!

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).