CG Programming III – Assignment #1 (Shadow Textures)
Due on 07/21/2009

For this assignment, you will implement a simple scene involving a moving object, a moving light source, and a ground plane. The object and light will cast a shadow onto the ground plane using a shadow texture. You do not need to implement complex lighting or other textures for this assignment.

You *must* use framebuffer objects to render the shadow textures.

I *recommend* implementing a light class that has the methods listed below. It might also be worthwhile to implement a more generic "viewer" class that implements these methods. Then the light class and perhaps a camera class could inherit from the viewer class.

- An `update_position` method that sets the position of the light in world space.
- A `get_position` method that gets the light's current position.
- A `get_view_transform` method that returns a viewing transformation matrix from the light's point of view. This method should be passed a point to look at and a nominal "up" direction.

In addition, I recommend implementing a function that generates a projection matrix from a distance and a radius. The distance and radius define a sphere some distance from the viewer along the Z axis. The resulting projection matrix should cause the sphere to fill the viewing area. When generating the shadow texture, assume the camera is looking at the center of the object. Calculate a bounding sphere from the object (you should be able to do this using simple trigonometry for simpler cubes and tori). Use this new function to calculate the projection matrix.

Finally, I recommend implementing a function called `texture_coordinate_scale_bias` that returns the appropriate scale and bias matrix to remap texture coordinates from the [-1, 1] range to the [0, 1] range.

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).