Graphics Programming I – Assignment #2 (Simple textured world)

In this assignment, you will implement a 3D world that a user can navigate using the keyboard.

- Navigation
  - Allow the user to move forward, backward, turn left (i.e., rotate the view), turn right, pitch up, and pitch down using the keyboard.
  - Allow the user to return to the starting position and view orientation using single key press.
- World Objects
  - Implement a ground plane. At least some of the other objects should be positioned to rest on the ground plane.
  - Implement multiple objects positioned on the world. Some objects may be static (i.e., not animated). There must be at least one "compound animated" object. That is, one object that moves relative to another object that is also moving. The stack of cubes in assignment #1 is an example of this.

    Objects must be implemented using some sort of high-level data structure. This structure should track information about the object (e.g., position, orientation, vertex data, texture object, etc.), and should implement methods to draw the object. It should also implement a method that, given a time delta, will update the position of the object based on its animation parameters.
  - Implement a sky box or sky cylinder. We will discuss this briefly, but searching the Internet will guide your way.
- Object Rendering
  - Implement at least two light sources in the world. One should be a directional light (representing the sun) and the other should be a *spot* light. The spot light should be positioned above the user and should move with the user. This should simulate a miner's headlight.
  - All objects in the world, including the ground plane, must be plausibly lit.
  - All objects should, to varying degrees, reflect the texture from the sky box. The type of reflection mapping implemented will depend on the method use to render the sky box / sky cylinder.
  - All objects should have their own base texture. This texture may be applied using any of the techniques discussed in class *except* reflection mapping. There is already one reflection map!

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).