CG Programming II – Assignment #2

In this assignment you will need to implement a single rotating cube with an animated light source. Each surface of the cube should consist of a single quadrilateral (or two triangles). The each cube surface should have ambient, diffuse, and specular reflection present. All lighting must be calculated *per-fragment* by a GLSL fragment shader.

In addition to implementing standard lighting, the surfaces must also be bump mapped. The bumps must affect both the specular and diffuse reflection.

Required graphical elements:

- Rotating cube.
- Animated point light source.
    - Light source moves (e.g., orbits the cube).
    - Light source has constant color.
    - Light source is represented on the screen using a point. Setting the point size > 1.0 would be helpful, but is not required.
- Specular, diffuse, and ambient reflection performed in a GLSL fragment shader.
- Bump mapping implemented in the fragment shader.

Required inputs:

- Escape must terminate the program.
- A key sequence must be available to enable or disable specular reflection (e.g., pressing 's').
- A key sequence must be available to enable or disable diffuse and ambient reflection (e.g., pressing 'd').
- A key sequence must be available to pause the animation.

The recommended sequence to implement this assignment:

1. Implement a *basic* vertex shader that performs standard GL rotation and passed the input vertex color, supplied in one of the attributes, through to the fragment shader. Implement a *basic* fragment shader that simply copies the color supplied by the vertex shader to the output color.
2. Modify the vertex program to transform the vertex's normal and pass it to the fragment shader as a varying. Modify the fragment shader to use the *normal* from the vertex shader as the output color.
3. Modify the fragment shader to normalize the normal before using it.
4. Modify the fragment shader to perform diffuse shading using the normalized normal and the light. Also add ambient lighting.
5. Modify the fragment shader to perform specular shading.
6. Modify the fragment shader to look-up a texel from a normal map. Use this normal as the output color. The texture coordinates need to be passed from the vertex shader. This will *replace* the normal passed by the vertex shader.
7. Use the texel from the normal map to perform the shading implemented in steps #4 and #5.

Criteria	***Excellent***	***Good***	***Satisfactory***	***Marginal***	***Unacceptable***
Code Function	Program correctly implements all required graphical elements in a manner	Program implements all required graphical elements, but the operation of some	Program implements all required graphical elements in some fashion.	Program implements most required graphical	Most or all of the required graphical elements are

Criteria	*Excellent*	*Good*	*Satisfactory*	*Marginal*	*Unacceptable*
	that is readily apparent when the program is executed. Appropriate algorithms and data structures are used in the implementation.	elements may not be obvious. Appropriate algorithms and data structures are used in the implementation.	Algorithms and data structures are used that perform the required function, but may be less than ideal.	elements in some fashion.	missing or do not function correctly.
Code Mechanics	Program code is formatted in a consistent manner, variables and data structures are named in a consistent, logical manner. Code is commented adequately.	Program code is mostly consistent, but contains some occasion inconsistencies.	Program code is readable. Individual functions or code blocks show consistent formatting, but that formatting does not carry through the entire program.	Program code is not consistently formatted, but is still somewhat readable.	Program code is a mess and may be more suitable as an entry to the International Obfuscated C Coding Competition.
User Interface	The program is responsive to input. All required inputs are implemented, and the user is informed, by the program, what the inputs are. The program can be terminated by the user.	The program is responsive to input. All required inputs are implemented. Some of the inputs are documented by the program.	The program is unresponsive under some circumstances. All required inputs are implemented. Some of the inputs are documented by the program.	The program is unresponsive under some circumstances. Some of the required inputs are either not implemented or are not implemented correctly. Some of the inputs are documented by the program.	Many of the required inputs are either not implemented or are not implemented correctly. The program lacks documentation for the inputs.