Graphics Programming I – Assignment #1 (Lit cube scene)
Part 1 due on 10/28/2009

In this assignment, you will implement a simple scene containing several lit, animated cubes. This assignment is divided into three parts. Each part is dues in successive weeks.

The first part requires only a single cube rotating in the scene.

- Implement a `vec4` class. This class should have methods that implement `add`, `sub`, `neg`, `mult`, `dot3`, `dot4`, `cross`, `magnitude`, and `normalize` methods. For `add`, `sub`, `neg`, and `mult` you may implement overloaded operators, but this is not required. You may also discover that you need additional operators while implementing the rest of the assignment.
- Using the `vec4` class, implement a `mat4` class. This class should have methods that implement `transpose`, `translate`, `scale`, `rotate_x_axis`, `rotate_y_axis`, `rotate_z_axis`, `look_at`, `perspective`, `mult` for a `vec4` times a `mat4`, `mult` for a `mat4` times a `vec4`, and `mult` for a `mat4` times a `mat4`. The rotation routines may be implemented by creating a routine to generate a rotation around an arbitrary axis. The `translate`, `scale`, `rotate_x_axis`, `rotate_y_axis`, `rotate_z_axis`, `look_at`, `perspective` should be non-class functions that return a `mat4`. These will likely be `friend` functions. You may also discover that you need additional operators while implementing the rest of the assignment.
- Implement a routine that creatss a buffer object and fills it with the vertexes of a cube. This data should be designed so that each face of the cube has a different color, and so that it can be used with `glDrawArrays`.
- Implement a display routine that will render the cube rotated by some angle.
- Implement an "idle" routine that will update the rotation angle based on the elapsed time. This routine could also generate the rotation matrix and update the vertex program's uniforms.

There are a lot of matrix and vector routines to write, but nearly all of them are very, very short. I *strongly* recommend implementing functions using other functions. For example, the matrix class should use an array of vectors, and vector multiplication and dot-products should be used to implement matrix multiplication. Implement a couple routines, then implement some simple test scaffolding. It will be very difficult to debug problems in more complex code (i.e., `look_at`) if you are not certain that the component routines are correct.

Do not spend a lot of time on these routines. In future assignments a more robust and feature-complete library will be available for your use.

Part 2 due on 11/04/2009

The second part requires several additions. Instead of a single cube, five cubes must be rendered. The cubes will start stacked in a column. Each cube will rotate around the edge with a positive X value that it shares with the cube below it. The should look like an arm bending. Each cube will repeatedly rotate from 0 to 45 degrees and back. At full rotation the top cube will be at the same level as the base cube. The five cubes will (roughly) form an arch.

Implement simple view frustum culling.

- Calculate a bounding sphere for each box. Transform the center of the bounding sphere by the model-view matrix.
- Calculate the plane equations for the camera-space view volume.
- Using the method described in the lecture notes to determine whether or not a sphere is inside the view volume.
- Do not render cubes associated with spheres that are outside the view volume.
- To test this, perform culling for a view volume that is much smaller then camera's actual view volume. Using half the actual field-of-view is a good choice.

Part 3 due on 11/11/2009

The third and final part of the assignment is to add lighting to the scene.

- Supply per-vertex normals
  - In addition to per-vertex position and color, specify per-vertex normals.
  - Create a new `attribute` in the vertex shader called `normal` and pass the per-vertex normals in through this attribute.
  - In addition to passing in the model-view-projection matrix, pass the upper 3x3 portion of the model matrix. Call this new matrix `normal_transform` in the vertex shader.
  - Transform the vertex normal by `normal_transform`. Question to think about: what "space" is the transformed normal in?
- Modify the vertex shader to perform per-vertex lighting.
  - Supply the position of a point light to the vertex shader in a uniform called `light_pos`. The point light should orbit the cubes around the (world-space) Z-axis. The point light should be 8 units from the origin.
  - Supply the direction of a directional light to the vertex shader in a uniform called `light_dir`.
  - Calculate the diffuse and specular lighting contributions for the point light.
  - Calculate the diffuse and specular lighting contributions for the directional light.
  - Combine the lighting from both lights with the vertex color. Pass the resulting color to the fragment shader in a varying called `color`.

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).