ECE 4122/6122 Lab 3: OpenGL with OBJ files and Multiple Objects

(100 pts)

Category: 3D Graphics
Due: Tuesday October 22th, 2023 by 11:59 PM

Objective:

To create a dynamic 3D graphics application using lighting, shading, model transformations, and keyboard inputs.

Description:

You can use the code from tutorial09_Assimp to create a C++ application that loads in the obj files for the 3D chess pieces and the chess board.

Modify the keyboard inputs so that (5 points each)

1) 'w' key moves the camera radially closer to the origin.
2) 's' key moves the camera radially farther from the origin.
3) 'a' key rotates the camera to the left maintaining the radial distance from the origin.
4) 'd' key rotates to camera to the right maintaining the radial distance from the origin.
5) The up arrow key radially rotates the camera up.
6) The down arrow radially rotates the camera down.
7) The 'L' key toggles the specular and diffuse components of the light on and off but leaves the ambient component unchanged.
8) Pressing the escape key closes the window and exits the program

Points grading:

- 30 Points – loading obj files and displaying the chess pieces.
- 30 Points – rotating and placing the chess pieces in the correct location and orientation
- 40 Points – keyboard control described above
- 5 pts extra credit the textures are applied correctly to the chess pieces and chessboard.
- 5 pts extra credit make a short, narrated video of your program in action. (include it in the zip file)

Turn-In Instructions

Create a Lab3 folder at the same level as the other tutorialxx _xx folders and place your code and the attached folders containing the obj file information in that folder.

Modify the top level CMakeLists.txt file to build your Lab3 application. You can make any needed changes to the files in the common folder to handle the keyboard input.

Zip up the folders **Lab3** and **common** into **Lab3.zip** and upload this zip file on the assignment section of Canvas. When the TAs replace the existing folders with yours and the code should compile and run correctly. You can also include a short, narrated video of your program in action.

Grading Rubric:

If a student's program runs correctly and produces the desired output, the student has the potential to get a 100 on his or her homework; however, TA's will look through your code for other elements needed to meet the lab requirements. The table below shows typical deductions that could occur.

AUTOMATIC GRADING POINT DEDUCTIONS PER PROBLEM:

Element	Percentage Deduction	Details
Does Not Compile	40%	Code does not compile on PACE-ICE!
Does Not Match Output	Up to 90%	The code compiles but does not produce correct outputs.
Clear Self-Documenting Coding Styles	Up to 25%	This can include incorrect indentation, using unclear variable names, unclear/missing comments, or compiling with warnings. (See Appendix A)

LATE POLICY

Element	Percentage Deduction	Details
Late Deduction Function	score – 0.5 * H	H = number of hours (ceiling function) passed deadline

Appendix A: Coding Standards

***Indentation*:**

When using *if/for/while* statements, make sure you indent **4** spaces for the content inside those. Also make sure that you use spaces to make the code more readable.
For example:

```
for (int i; i < 10; i++)
{
    j = j + i;
}
```

If you have nested statements, you should use multiple indentions. Each { should be on its own line (like the *for* loop) If you have *else* or *else if* statements after your *if* statement, they should be on their own line.

```
for (int i; i < 10; i++)
{
   if (i < 5)
   {
       counter++;
       k -= i;
   }
   else
   {
       k +=1;
   }
   j += i;
}
```

Camel Case:

This naming convention has the first letter of the variable be lower case, and the first letter in each new word be capitalized (e.g. firstSecondThird).

This applies for functions and member functions as well!

The main exception to this is class names, where the first letter should also be capitalized.

Variable and Function Names:

Your variable and function names should be clear about what that variable or function represents. Do not use one letter variables, but use abbreviations when it is appropriate (for example: “imag" instead of “imaginary”). The more descriptive your variable and function names are, the more readable your code will be. This is the idea behind self-documenting code.

File Headers:

Every file should have the following header at the top

```
/*
Author: your name
Class: ECE4122 or ECE6122 (section)
Last Date Modified: date

Description:

What is the purpose of this file?

*/
```

Code Comments:

1. Every function must have a comment section describing the purpose of the function, the input and output parameters, the return value (if any).
2. Every class must have a comment section to describe the purpose of the class.
3. Comments need to be placed inside of functions/loops to assist in the understanding of the flow of the code.