Programming OpenGL

Notes for CSE 470
Instructor: Ross Maciejewski

1 Project Guidelines

- Have fun, learn, be creative, and create something you are proud to show-off!
- No teamwork allowed! It is ok to discuss the projects with other students. You can use the discussion board on myASU (The TA will also try to answer your questions posted at myASU, but only sporadically); however your code must be your own. (You would be surprised how easy it is to detect copied code!). You cannot post code on the discussion board! (only very small code sections)
- Comment your code - you will be glad you did. You will be reusing code throughout the semester.
- You will upload your projects on myASU. Details about this will be announced in class.
- Late projects will lose points:
  10% for up to 24 hours too late.
  20% for up to 4 * 24 hours too late.
  The myASU timestamp will show when you uploaded your project.
  40% for one week late
  75% for two weeks late
- If you have **good reasons** that you cannot complete a project on time **and** you have written documentation, then we can make adjustments to due dates. However, you must notify us before the due date that you would like to discuss such an arrangement. Good reasons would be illness, family emergency, visiting a conference to present a paper, …
- All the University, Student Life and Fulton School academic integrity policies hold: *http://www.asu.edu/studentlife/judicial/integrity.html*

2 Upload your projects

How to upload a Project
This is a short guide for the successful upload of a project:

- Check the due date of the project on myASU
- Use myASU (digital dropbox) to upload your project. (Do not send the program as email attachment to the instructor or TA!)
- Your project should be packed in a zip file and named with the following naming convention:
  PX.Lastname.V1.zip
  (if you submit some updated files, use a different version)
  So my second project would be named
  P2.Maciejewski.V1.zip
  If I then found a last minute mistake and want to upload again I would use the name P2.Maciejewski.V2.zip
- If there is someone with the same last name in the course you can add your first name after the last name in the filename.
- The zip file should include all the source code and should be ready to compile. That means it should include your Visual C++ project files.
- The zip file should include an executable and the freeglut.dll. You have to make sure that the executable can be started directly by clicking on it. (from wherever the file is downloaded to). You might have to copy the exe file from the Release folder into the project directory.
- A short text file called “instructions.txt” – this text file should include the following:
  - a) instructions about how to navigate in your program, including the function of special keys (e.g. x exits the program)
  - b) special functions that you implemented
  - c) changes from the original specification that you negotiated with the instructor or TA to the original specification
  - d) other things you consider important
  - e) specify the libraries or code that you used and did not write yourself. You do not need to specify code that we provide on myASU.

3 Project 3

3.1 Purpose

This project has been designed for you to get experience with 3D objects and textures.

3.2 Overview

You will develop an application that reads a mesh data file and allows the user to interactively view it. Several options for the 3D viewer will be available. These will include rendering modes, different views and projections.

This program is a building block for the next program that will display 3D illuminated objects.

3.3 Required Functionality

Here are the elements that your program must have. Despite this "laundry list" of requirements, there is some room for creativity! Note: If the text suggests a menu item "Sound - Off" that would mean that "Off" is an entry in the submenu "Sound".

- **20 pts – Create a Scene: 20 pts -** The scene is drawn as a collection of building object (meshes). You should at least implement 3 buildings that are assembled through a collection of three types of basic primitives: *boxes, cones,* and *cylinders*. Each of the three buildings should consist of three or more basic primitives. There should also be a *ground plane* for the meshes to stand on. The four types of basic primitives (box, cylinder, cone and plane) should be generated procedurally. That means you write a respective functions that creates a mesh data structure describing the object(E.g. CreateBox(...)).
- **10 pts –Object Loader:** Use or implement a file loader to load several unique 3D objects (minimum of 2 objects). You can use additional meshes taken from the internet, designed by yourself, or additional meshes that we posted on blackboard.
- **10 pts** – **Mesh class:** You will implement and use a proper mesh class as discussed in the OpenGL lecture slides. You should create a mesh data structure for all elements of the scene. The simplest data structure stores the vertices of a mesh in a dynamic array = an STL-vector. A second STL-vector stores the indices or index triplets. Each triangle is defined by three indices. The indices in the index STL-vector denote the location of the vertices in the vertex STL-vector. A third STL-vector stores the normals. The use of STL is mandatory. In the next project you need to extend with per face normals, per vertex normals, and per vertex texture coordinates.
- **5 pts – Expand the Mesh Data Structure:** Your mesh data structure should be able to store per face normals, per vertex normals, and per vertex texture coordinates.

- **10 pts – Object rendering:** Write a rendering routine that renders all objects stored in the mesh data structure. The objects should have colors per vertex (can be assigned randomly). The menu items "Shading - Flat", "Shading - Smooth", “Shading – none” should switch between smooth shading, flat shading and displaying the scene only as a wireframe.
- **10 pts – Per Face Normal Vectors (+ Normal Visualization):** You need to implement a routine that computes per face normal vectors for each triangle in the scene. If the user selects the menu item "Normals - Per Face Normals" the scene is rendered using per face normals.
  If the user selects the menu item "Normal Visualization - On" and the application is in per face normal mode the per face normals should be visualized as lines attached to the middle of the triangles. The normals have to be visualized with some reasonable length so that they are easily visible. "Normal Visualization - Off" turns off normal visualization.
- **15 pts – Basic Per Vertex Normals (+ Normal Visualization):** To make shading smooth, you should compute a per vertex normal for each vertex in the scene. To compute a per vertex normal of a vertex v, you have to average the per face vertex normals for each triangle that shares the vertex v. You should implement two versions. One version simply averages the normals and the other version computes a weighted average, so that the normals are weighed according to the area of the incident triangles. (selected by "Normals - Per Vertex Normals" and "Normals - Per Vertex Normals Weighted"). If the user selects the menu item "Normal Visualization - On" and the application is in per vertex normal mode the normals should be drawn as line segments starting at the vertices.
- **10 pts – Navigation:** The camera is controlled as follows:
  You can go forward, backward with the cursor keys or rotate using the left and right keys. Use the a and s key to control the elevation over the ground plane (a real person cannot do that). The look at point moves with the camera and is a fixed offset away in the movement direction.
- **15 pts – Interaction (Graduate Student Requirement):** The user should be able to select an object with the mouse. When the mouse point touches an object, the object will highlight.

3.4 Grading

These points represent an initial weighting of the functionality. **Important:** You get full points for a *good* implementation of the required functionality. In principle you will get full points if you implement the requirements correctly, but there are many ways to do things according to specification that we might consider *unreasonable* or *undesired*. Especially we will deduct points if the design choices make it difficult to judge the program (E.g. you choose materials that are so dark that it hard to see anything, the objects move too slowly or too fast, the control of the program is extremely difficult.)

Note: we might change the weights slightly if a requirement is misunderstood by a majority of the class or some other special circumstance. I do not expect that to happen though.