DigiPen Institute of Technology

CS300

Assignment 1: Loading and displaying geometry using OpenGL 4

Due date:

- Week 5, Friday, 23:59:00
- For late submission, please refer the course outline document.

Description

For your first programming assignment, you are to implement a GUI-driven application with the following requirements/features:

1. Scene Setup
   - Be able to load an OBJ file from disk. You will write the functionality to interpret the file data to extract information about vertices and faces from the file.
   - Once the file is loaded, apply appropriate transform to scale the model to the range [-1,1].
   - Center the object at the origin of the [-1,1] box.
   - Render 8 spheres, equally spaced on a circle around the object. The diameter of the spheres should be 0.1 units. Keep the spheres at sufficient distance (2.0-3.0 units) away from the model.
   - Render the following for full credit:
     - Central object (OBJ file – drawn using GL_TRIANGLES)
     - Spheres (Generate procedurally – drawn using GL_TRIANGLES)
     - Orbit of the spheres (drawn using line segments – GL_LINES)
2. Geometry Operations
   - We will use the provided OBJ files to test your program. **Make sure that your OBJ importer is able to read the supplied files.**
   - For face records with more than 3 vertices, use the "triangle-fan" approach discussed in class to create multiple triangles from the polygon.
   - Calculate the **per-vertex normal** from the face normal in the neighborhood of a vertex. Make sure that the vertex normal accumulation accounts for corner-cases

where there may be more than one face in the same plane. Ignore normals that are parallel to any previously accumulated normals in the vertex-normal computation.

- Use GL_LINES to draw the normal. For rendering a face normal, place it at the centroid of the triangle. **(How would you compute a face normal for a triangle?)**
- Compute the vertex normal by averaging the face normal of the faces that the vertex is a member of.
- **Your program should provide a menu-driven functionality to toggle between displaying the vertex normal, the face normal, or none.**
- Use suitable scale so that the normal vector does not appear too big wrt. the model.

3. **The rendering function must be completely implemented by the student, i.e. you must not use a third-party library to draw your shape.**
4. Rendering must be done using OpenGL rendering functions as discussed in class. Use Vertex Buffer Objects with the Index Buffer Objects for indexed mode rendering. (Implementation hint: Implement this functionality in a TriangleMesh class, with a VAOManager, and VertexBufferManager classes.)
5. The only "lighting" functionality that we will implement in this assignment is the ambient and diffuse lighting terms for a directional light source. Refer in-class discussion for the calculation of the individual terms.
6. **Additional Hints:**
   - The geometric information in the OBJ file can be translated directly into suitable arrays for specifying vertices and their attributes.
   - The topological information includes the indices that will populate the so-called "index array" (GL_ELEMENT_ARRAY_BUFFER) to be passed to the OpenGL server. **OBJ indices in the file begin at "1", OpenGL indices in the element array buffer begin at "0". Make sure your OBJ reader accounts for this change.**
   - Depending on face-based or vertex-based normal specification, you may have to reallocate the vertex information to account for redundant shared vertices between

multiple faces. Your code MUST support this functionality for all types of rendering calls.

```
Mesh *CreateSphere( float radius, int numDivisions )
{
    For theta: 0 → 360, stepsizeTheta = 360 / numDivisions;
           For phi: 0→ 180, stepsizePhi = 180 / numDivisions;
                  // Generate point on the sphere
         X = radius * sin( theta ) * cos( phi )
         Y = radius * sin( theta ) * sin( phi )
         Z = radius * cos( theta )

         Add point (X, Y, Z) to the sphere
         phi += stepsizePhi;
   theta += stepSizeTheta

   // Once all the sphere points are generated, create triangles
from adjacent points

      // Generate the index buffer that holds the indices of the
triangles

     // Create a new Mesh from the vertices and the indices
    Mesh *sphereMesh = new Mesh(…);

    return sphereMesh;
```

Suggested weekly breakdown of the tasks

In order to implement the assignment successfully, I would recommend the following checkpoints for each week:

Week	Functionality
1	Read. Yes, read the OpenGL references to understand the concepts and terminology. Use a paper-pencil model to sketch out any classes / data that needs to be developed. You may use the provided skeleton code as your reference if you wish.
2	Using the provided framework code, or by developing your own, render a single triangle by (a) hard coding the values in code, and (b) providing the same through an OBJ file. Your output must be identical in both cases.
3	Extend the OBJ input functionality to support larger files from the specified collection. Add functionality to support procedurally generated geometry.
4	Implement the Phong Lighting model from CS 250 for the OBJ files in the

	sample set.
5	Debug, document and repeat!

Assignment Submission Guideline

Please refer to the syllabus for assignment submission guideline. Failure to adhere to the submission guidelines might cause you to lose points.

Grade breakdown

Grade Item	Maximum Points Possible
Setting up an OpenGL application	**20**
Reading data from an OBJ file without errors	**20**
Implementing vertex and fragment shaders corresponding to Phong Lighting	**20**
Scene setup for the assignment is as per specification – Central object + rotating spheres	**10**
Face & vertex normals calculation and display	**10**
Miscellaneous issues (GUI controls, compilation or execution issues, etc.)	**20**
TOTAL	**100**