1. Overview

In this assignment you will use OpenGL Shading Language (GLSL) to write your own vertex and fragment shaders to compute illumination and shading of meshes.

Requirements

1. **Generate Triangle Meshes (TODO 1):**
    1. Use Element Buffer Object (EBO) to draw the cube. The cube provided in the start code is drawn with Vertex Buffer Object (VBO). In the DisplayableCube class draw method, you should switch from VBO draw to EBO draw. To achieve this, please first read through VBO and EBO classes in GLBuffer. Then you rewrite the self.vertices and self.indices in the DisplayableCube class. Once you have all these down, then switch the line vbo.draw() to ebo.draw().
    2. Generate Displayable classes for an ellipsoid, torus, and cylinder with end caps. These classes should be like the DisplayableCube class and they should all use EBO in the draw method.
       PS: You must use the ellipsoid formula to generate it, scaling the Displayable sphere doesn't count

2. **Set Normal Rendering (TODO 2):** As a visual debugging mode, you'll implement a rendering mode that visualizes the vertex normals with color information. In Fragment Shader, use the vertex normal as the vertex color (i.e. the rgb values come from the xyz components of the normal). The value for each dimension in vertex normal will be in the range -1 to 1. You will need to offset and rescale them to the range 0 to 1.
3. **Illuminate your meshes (TODO 3):** You'll implement the missing part in the Fragment shader source code. This part will be implemented in OpenGL Shading Language. Your code should iterate through all lights in the Light array. Use the illumination equations we learned in the lecture to implement components for
    1. Diffuse
    2. Specular
    3. Ambient
4. **Set up lights (TODO 4):** Set up lights:
    1. Use the Light struct which is defined above and the provided Light class to implement illumination equations for 3 different light sources.
        - Point light
        - Infinite light
        - Spotlight with radial and angular attenuation
    2. In the Sketch file Interrupt_keyboard method, bind keyboard interfaces that allow the user to toggle on/off specular, diffuse, and ambient with keys S, D, A.
5. **Create your scenes (TODO 5):**
    1. Test scene: We provide a fixed scene (**SceneOne.py**) for you with preset lights, material, and model parameters. This scene will be used to examine your illumination implementation, and you should **NOT** modify it.
    2. Create **2 new scenes** (can be static scenes). Each of your scenes must have
        - at least 3 differently shaped solid objects
        - at least 3 different materials
        - at least 2 lights
        - all types of lights should be used
    3. Provide a keyboard interface that allows the user to switch between scenes, toggle on/off each of the lights in your scene: Hit 1, 2, 3, 4, etc. to identify which light to toggle.

6. **Texture Mapping (Required for 680, Extra Credit FOR 480) (TODO 6/BONUS 6)**
    1. Set up each object's vertex texture coordinates(2D) to the self.vertices 9:11 columns (i.e. the last two columns). Tell OpenGL how to interpret these two columns: you need to set up attribPointer in the Displayable object's initialize method.
    2. Generate texture coordinates for the torus and sphere. Use "./assets/marble.jpg" for the torus and "./assets/earth.jpg" for the sphere as the texture image. There should be no seams in the resulting texture-mapped model.

Extra Credit

7. **Normal Mapping (BONUS 7)**
    1. Perform the same steps as Texture Mapping above, except that instead of using the image for vertex color, use it to modify the normals. Use the image ("./assets/normalmap.jpg") for both the sphere and the torus.
    2. Compute the tangent and bitangent vectors for the sphere and torus vertices and append them to the self.vertices 12:18 columns. Then, pass to the vertex shader by modifying the VAO.
    3. Build the TBN matrix for each point in the vertex shader, and replace its normal in the fragment shader by **TBN @ normalized(normalmap[u, v].rgb)**. Continue to the lighting with this modified normal. For more details, you can read here.

Programming Style

For any modified or newly added source file, you should include a brief explanation about what you did in this file. Your code should be readable with sufficient comments. You should use consistent variable naming and keep reasonable indentation.

README

With each assignment, please upload a README text or markdown file with the following information:

- your name
- any collaborators that you spoke to
- the class
- the assignment number
- a summary of the code and implementations you have written for the assignment

For the last point, you should outline the method you used to solve the problem, describe the variables and data-structures used in the program, any error conditions which might be encountered in the solution, and how these error conditions are handled by the solution.
If these details are described in the comments above the relevant functions or files, you may be brief here. You may call this file "READMEStudent" or something analogous to differentiate it from the README in the skeleton code.
Please include a mention of any resources that you consulted while completing your assignment.

2. Resources

2.1 Starter code

A Python Program skeleton, PA4, which includes basic classes, methods, and main pipeline is provided for you. You are expected to complete the parts marked with TODO. Comments in the skeleton code will help guide you in writing your subroutines.

2.2 Environment Setup

Installing the appropriate programming environment should be covered in a lab session. For more step-by-step instructions, please check the environment set up guidelines.

2.3 User Interface

The user interface to the program is provided through mouse buttons and keyboard keys.

Provided in starter code:

- **Left Mouse Dragging**: Rotate the camera
- **Middle Mouse Dragging**: Translate the camera
- **Scroll Up/Down**: Zoom in/out the scene

To implement:

- **A**: Toggle Ambient light
- **D**: Toggle Diffuse light
- **S**: Toggle Specular light
- **N**: Toggle Normal rendering
- **Left Arrow**: Go back to the last scene
- **Right Arrow**: Next Scene
- **1,2,3, ...**: Toggle lights in the current scene

2.4 Video Demo

We prepared a video demo for you (link TBA). We hope this can help you better understand your tasks and speed up your debugging process.

3. Submission (due @ 11:59 PM, Tuesday, 12/3)

3.1 Source Code

Your program's source files are to be submitted electronically on Gradescope. The code you submit should conform to the program assignment guidelines.

3.2 Demo

Part of your grade for this programming assignment will be based on your giving a short demo (5-10 minutes) during the CS480/680 scheduled labs following the assignment due date. You will be expected to talk about how your program works.

4. Grading

Students may earn a maximum of 10 points extra credit.

Requirements	CS480	CS680
Generate Triangle Meshes: Ellipsoid, Torus, and Cylinder with end caps	15	15

Implement EBO for defining your meshes	10	10
Generate normals for your meshes, and implement normal visualization	5	5
Illuminate your meshes with diffuse, specular, and ambient components	25	25
Support 3 different light types (point, infinite, spotlight)	15	15
Create 2 new scenes (in addition to test scene given in starter code)	20	20
Texture mapping (sphere, torus)	10 (extra)	10
Normal mapping	5 (extra)	5 (extra)
Programming Style	10	10

5. Code Distribution Policy

You acknowledge this code is only for the course learning purpose. You should never distribute any part of this assignment, especially the completed version, to any publicly accessible website or open repository without our permission. Keeping the code in your local computer or private repository is allowed. You should never share, sell, gift, or copy the code in any format with any other person without our permission.