1. Overview

In this assignment, you will use the OpenGL Shading Language (documentation at GLSL 3.30) to write your own fragment shader for illumination and shading of meshes.

Requirements

1. **Generate Triangle Meshes (TODO 1):**

   You will write classes to construct five geometric primitives: cube, ellipsoid, cylinder (with end caps), and torus. You should do so using the parametric definitions of each shape as discussed in lecture.

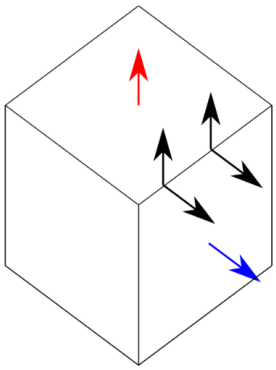
   1.1 Using a **Vertex Buffer Object (VBO):**

   A vertex buffer object stores data for the vertices of a triangle mesh, including positions, normals, colors, and texture coordinates. Following the example provided in DisplayableCube, generate and store the vertex data for your mesh in a VBO (self.vertices).

   If your primitives are drawn using a VBO only (no EBO), keep the call to self.vbo.draw() in your primitive's draw() function.

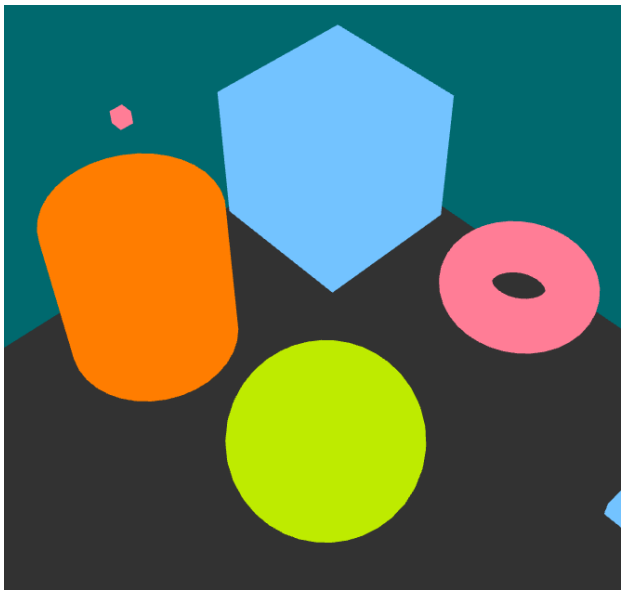
   1.2 Adding an **Element Buffer Object (EBO):**

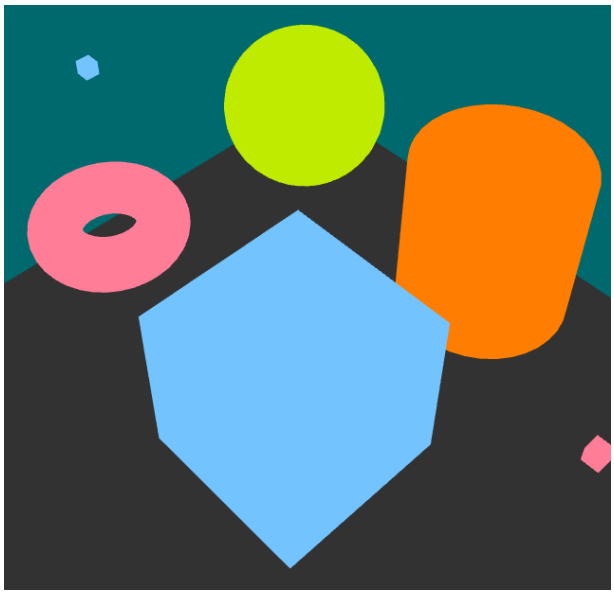
   An element buffer object allows vertex data to be referenced multiple times in the construction of a mesh, reducing redundancy in the VBO. Construct an EBO for each primitive (self.indices), referencing vertex data stored in a VBO. For full credit, **your VBO should not contain multiple copies of identical vertex data** (e.g. the poles of a sphere).

   Note: It may be necessary to store separate data for vertices that lie on sharp edges in order to capture discontinuous face normals. In this case, only the positions are identical (and not the normals). You may also need multiple copies of vertices for seamless texture mapping (TODO 6) as well.

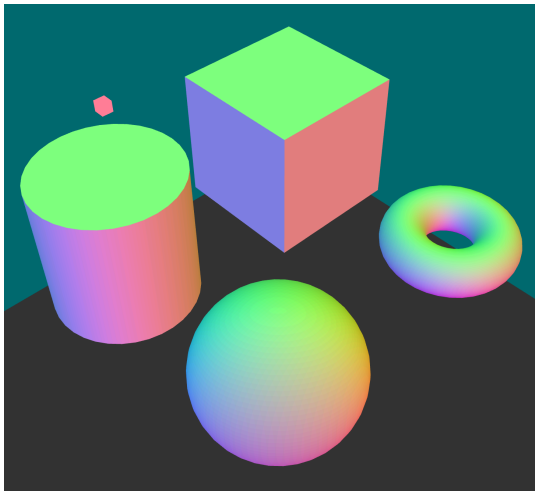
   To have OpenGL draw from the EBO, call self.ebo.draw() instead of self.vbo.draw() in your primitive's draw() function.



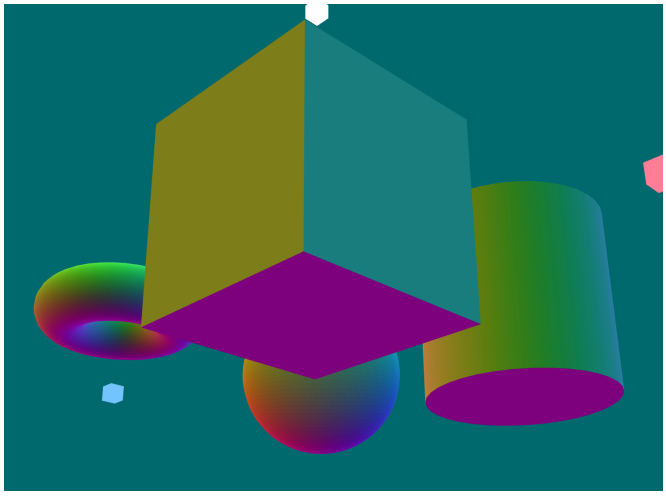


SceneOne after constructing Sphere, Torus, and Cylinder (“vertex” shading)

2. **Set Normal Rendering (TODO 2):** Implement a rendering mode that visualizes the vertex normals with color information, for debugging purposes. In the fragment shader, use the vertex normal as the vertex color (i.e. the rgb values come from the xyz components of the normal). Since the coordinates of the normal are in the range [-1, 1], you will need to offset and rescale them to create a valid RGB tuple in the range $[0, 1]^3$.



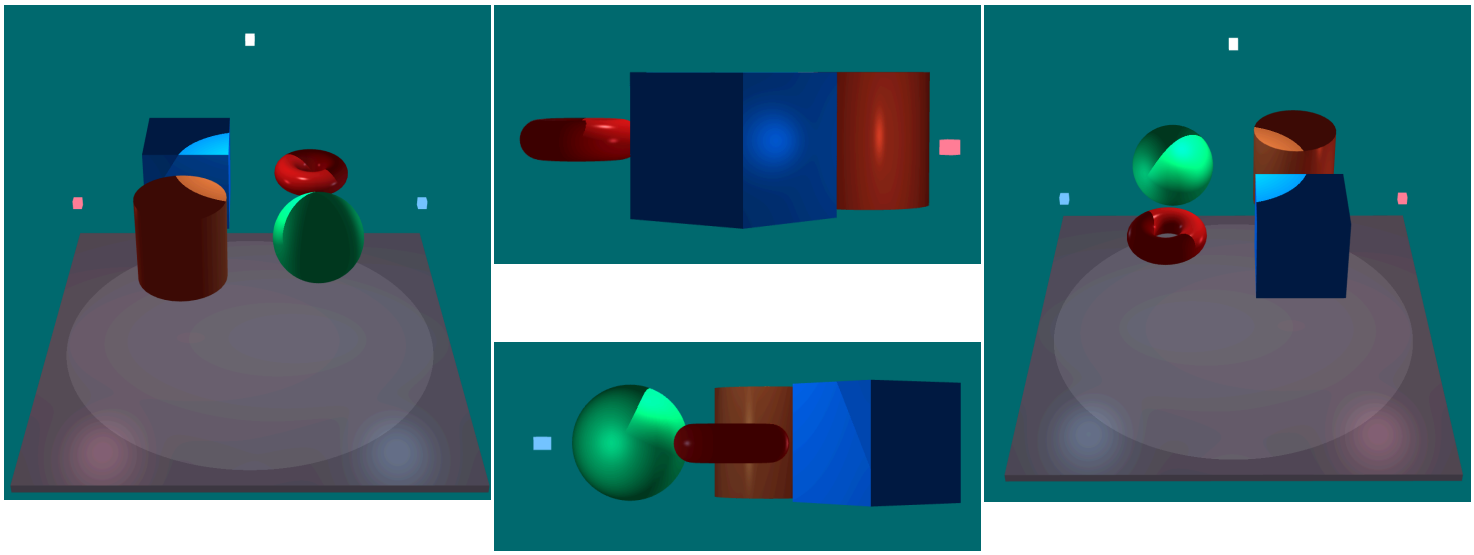
Normal rendering, viewed from top



Normal rendering, viewed from bottom

3. **Illuminate your meshes (TODO 3):** You’ll implement the missing parts in the fragment shader source code (FragmentShader.glsl). This part will be implemented in the OpenGL Shading Language. Your code should iterate through all lights in the Light array, computing the contribution from each light source. Use the lighting model from lecture to implement both **diffuse** and **specular** illumination. Incorporate a fixed amount of **ambient** illumination as a property of each scene (TODO 5).
4. **Set up lights (TODO 4):** Set up lights:

1. Use the Light struct (defined above) and the provided Light class to implement illumination equations for 3 different types of 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 the specular, diffuse, and ambient components with keys S, D, A.



SceneOne after completing TODO 3 & TODO 4. (NOTE: minor attributes like brightness might differ depending on your implementation of radial attenuation.)

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

    At least one scene must contain an infinite light source. (The default scene already has a spotlight).

    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).
    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)**. Use the modified normal in your lighting calculations. 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 skeleton program 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 step-by-step instructions, please check the environment setup 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 have prepared a video demo for you. PA4_Demo.mp4

We hope this can help you better understand your tasks and speed up your debugging process.

3. Submission (due @ 11:59 PM, Tuesday, 12/2)

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.