Computer Graphics Lab 9

Shader and texture in OpenGL

Lab 1: Introduction to OpenGL Shaders with Python

Objectives:

- Understand Basic Shader Structure
  - Understand the OpenGL shader pipeline
  - Learn GLSL (OpenGL Shading Language) basics
- Master Data Flow in OpenGL
  - Learn how data moves from CPU to GPU
  - Understand vertex attributes and uniforms
  - Grasp color interpolation concepts
- Create Basic 2D Shapes
  - Implement simple geometric shapes (square, triangle)
  - Understand vertex positioning in OpenGL
  - Work with color attributes

Required Files

Download the following files into your project directory:

1. first_shader.py (main program)
2. glApp folder containing:
   - PyOGApp.py
   - Utils.py
   - Graphics_Data.py
   - Square.py
   - Triangle.py
   - Mesh.py
   - Uniform.py
   - Camera.py
   - Transformation.py

Part 1: Understanding first_shader.py

Task 1.1: Run and Observe

Run first_shader.py and Observe what you see.

TA Check: _____

Task 1.2: Answer the Following Questions

1. What is the purpose of the vertex shader in this program? Explain how the position calculation works.
2. How does the color information flow from vertex shader to fragment shader?
3. Why do we need both VAO (Vertex Array Object) and VBO (Vertex Buffer Object) in modern OpenGL?
4. What is the role of the uniform 'translation' variable in the shader?
5. Explain why we use vec4 for gl_Position but vec3 for other attributes?

Part 2: Creating shader2.py

Task 2.1: Modify the Code

Create shader2.py by modifying first_shader.py to display a five-pointed star:

1. Replace the vertex data with:

```
position_data = [
    [ 0.0, -0.9, 0.0],  # Bottom point
    [-0.6,  0.8, 0.0],  # Upper left
    [ 0.9, -0.2, 0.0],  # Right
    [-0.9, -0.2, 0.0],  # Left
    [ 0.6,  0.8, 0.0]   # Upper right
]
```

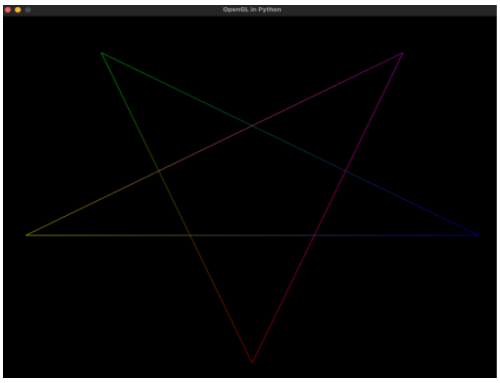
2. Add color data:

```
color_data = [
    [1.0, 0.0, 0.0],  # Red
    [0.0, 1.0, 0.0],  # Green
    [0.0, 0.0, 1.0],  # Blue
    [1.0, 1.0, 0.0],  # Yellow
    [1.0, 0.0, 1.0]   # Purple
]
```

3. Use GL_LINE_LOOP instead of individual shapes:

```
glDrawArrays(GL_LINE_LOOP, 0, self.vertex_count)
```

4. Expected Output



TA Check: _____

Lab 2 : 3D Projections in OpenGL with Python

Objectives

- Understand 3D transformation pipeline in OpenGL
- Implement perspective projection and camera systems
- Work with shader uniforms for transformation matrices

- Create interactive 3D scene visualization

Prerequisites

Required Files

Download the following files into your project directory:

1. Projections.py (main program)
2. glApp folder containing:
    - PyOGApp.py
    - Utils.py
    - Graphics_Data.py
    - Square.py
    - Triangle.py
    - Mesh.py
    - Uniform.py
    - Camera.py
    - Transformation.py
    - Axes.py

Part 1: Understanding the Code

Task 1.1: Shader Analysis

Study the vertex shader code:

```
#version 330 core
in vec3 position;
in vec3 vertex_color;
uniform mat4 projection_mat;
uniform mat4 model_mat;
uniform mat4 view_mat;
out vec3 color;
void main() {
    gl_Position = projection_mat * inverse(view_mat) * model_mat * vec4(position, 1.0);
    color = vertex_color;
}
```

Task 1.2: Answer the Following Questions

1. Explain the purpose of each transformation matrix:
    - projection_mat
    - model_mat
    - view_mat
2. Why do we use inverse(view_mat) in the transformation chain?
3. What would happen if we changed the order of matrix multiplication?
4. How does depth testing (GL_DEPTH_TEST) affect the rendering?
5. Explain how the camera movement system works in this implementation.

Part 2: Practical Implementation

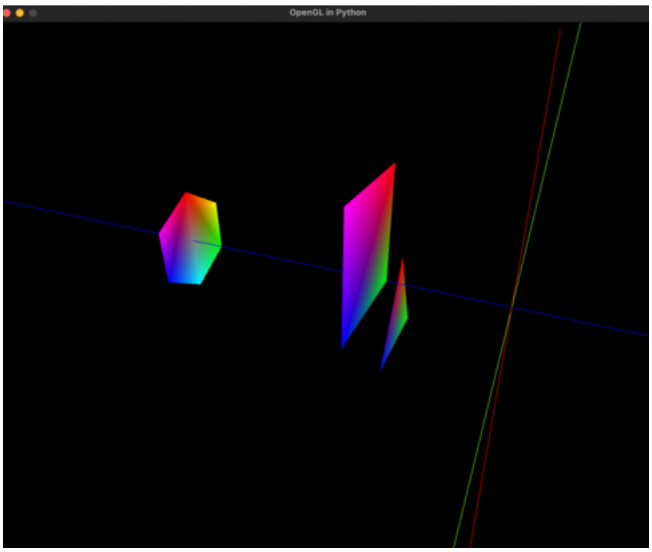
Task 2.1: Running the Base Program

1. Run Projections.py and observe:
    - Initial scene setup
    - Camera controls
    - Object positions
    - Coordinate axes

TA Check: _____

Task 2.2: Modifications

1. Create a Projections2.py by modufying Projections.py to include a new object
    - Create a new geometric shape in glApp
    - Position it in 3D space
    - Ensure proper coloring



TA Check: _____

Lab 3: 3D Objects with Lighting and Textures

Objectives

- Understand the basics of OpenGL shader programming
- Implement 3D object rendering with textures
- Apply lighting effects using fragment shaders
- Work with camera transformations in a 3D scen

Task 1:

1. Download and run “main.py”

TA Check: _____

2. Adding More Objects
    a. Create a new 3D object.
    b. Apply a new texture to the object

TA Check: _____