Project 9: Advanced Shaders 1

CST-310 Computer Graphics

Angel Velazquez, Kyler Harden

December 1, 2024

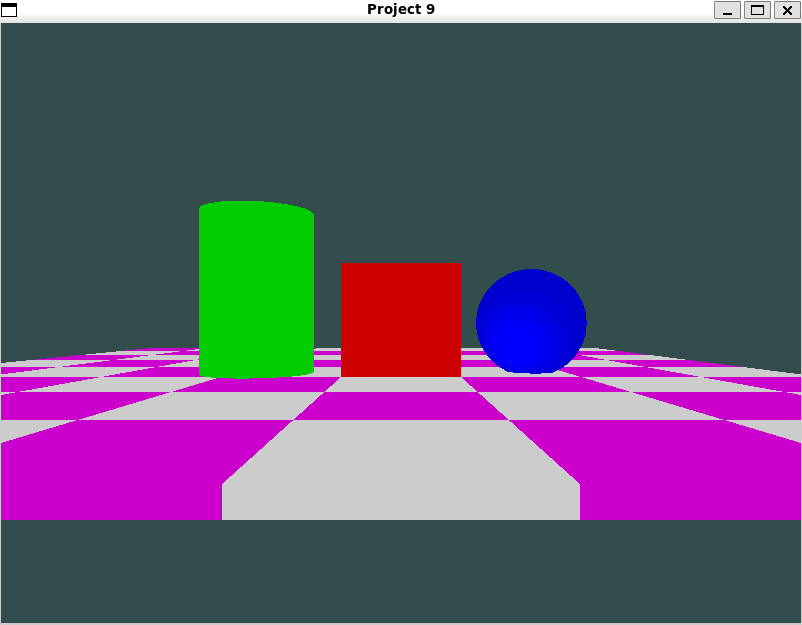
Documentation

Theoretical Background:

The goal of this project is to become more familiar with shaders and rendering in a 3d world. In order to achieve this we are going to create a Cylinder, a square, and a sphere on a checkerboard pattern floor. We would also like to manipulate the camera in order to move around the scene and be able to verify that the shapes are being correctly rendered in 3D and zoom in to look for any mistakes or gaps between the object and the floor.

Screenshots:

The following screenshot is of the code execution that correctly displays the objects in order from left to right: green cylinder, red square, and blue sphere.



**Concepts and Tools**

1. OpenGL Graphics Pipeline:

• Vertex Processing: Transforms vertex data from local to screen space.

• Fragment Processing: Handles per-pixel operations like color and lighting.

• Shaders: Vertex and fragment shaders are written in GLSL to handle transformations and lighting calculations.

2. Lighting and Camera:

• Implements Phong lighting with a single light source.

• Camera system allows free movement and rotation, controlled via keyboard inputs.

3. Data Representation:

• Vertices for geometric shapes are represented using arrays.

• Models (sphere and cylinder) are imported from external .obj files.

4. Graphics Libraries:

• GLFW: Manages window, context, and inputs.

• GLEW: Handles OpenGL extensions.

• SOIL: Loads textures.

• GLM: Provides mathematical operations like matrix transformations.

**Mathematical Concepts**

1. Transformations:

• Translation:

• Moves objects to different positions in the scene.

• Scaling:

• Adjusts the size of objects.

• Perspective Projection: :

• Projects 3D objects into 2D screen space.

2. Lighting Model (Phong Reflection):

• :

• Ambient: Constant base color.

• Diffuse: Depends on the angle between light and surface normal ().

• Specular: Reflection highlights ().

3. Camera:

• View Matrix: Defines the camera’s position and orientation.

• Uses glm::lookAt:

**Programming Concepts**

1. Shaders:

• Vertex Shader: Applies transformations (model, view, projection matrices).

• Fragment Shader: Computes lighting and color.

2. Uniforms:

• Pass dynamic data like transformation matrices, lighting information, and camera position to shaders.

3. Buffers:

• Vertex Buffer Object (VBO): Stores vertex data.

• Vertex Array Object (VAO): Configures vertex attributes.

4. Rendering Pipeline:

• Clear Buffers → Use Shaders → Bind Textures → Draw Objects → Swap Buffers.

**Aesthetic Decisions**

1. Color Scheme:

• Checkerboard alternates between purple and white.

• Cube is red, cylinder is green, and sphere is blue to provide clear visual differentiation.

2. Lighting:

• White light enhances the vividness of object colors.

3. Layout:

• Checkerboard forms the ground plane, objects are placed above for a structured scene.

**Flowcharts for Objects**

Checkerboard Logic

1. Loop over an 8x8 grid.

2. Alternate square colors based on .

3. Use transformations for each square:

• Translate to grid position.

• Scale to form flat tiles.

4. Draw each square.

Cube Logic

1. Translate the cube to the desired position.

2. Pass lighting and material properties to the shader.

3. Draw using the VAO and vertex data.

Cylinder Logic

1. Load cylinder model from .obj.

2. Translate, scale, and rotate the model to fit the scene.

3. Use cylinder-specific shader for lighting.

4. Draw the model using its Draw() function.

Sphere Logic

1. Load sphere model from .obj.

2. Translate and scale the model.

3. Use sphere-specific shader for lighting.

4. Draw the model using its Draw() function.

**Detailed Algorithm**

Main Loop

1. Calculate deltaTime.

2. Process user inputs (do\_movement).

3. Clear buffers.

4. Compute camera matrices (view, projection).

5. Render each object:

• Checkerboard: Loop through tiles and draw.

• Cube: Transform and draw.

• Cylinder: Transform and draw.

• Sphere: Transform and draw.

6. Swap buffers.

**Flowchart (Textual Representation)**

1. Start

2. Initialize GLFW and GLEW

3. Create window

4. Set up VAO/VBO

5. Load shaders

6. Main Loop:

• Input Handling

• Update Camera

• Clear Buffers

• Render Checkerboard:

• Loop over tiles

• Compute transformations

• Draw tile

• Render Cube

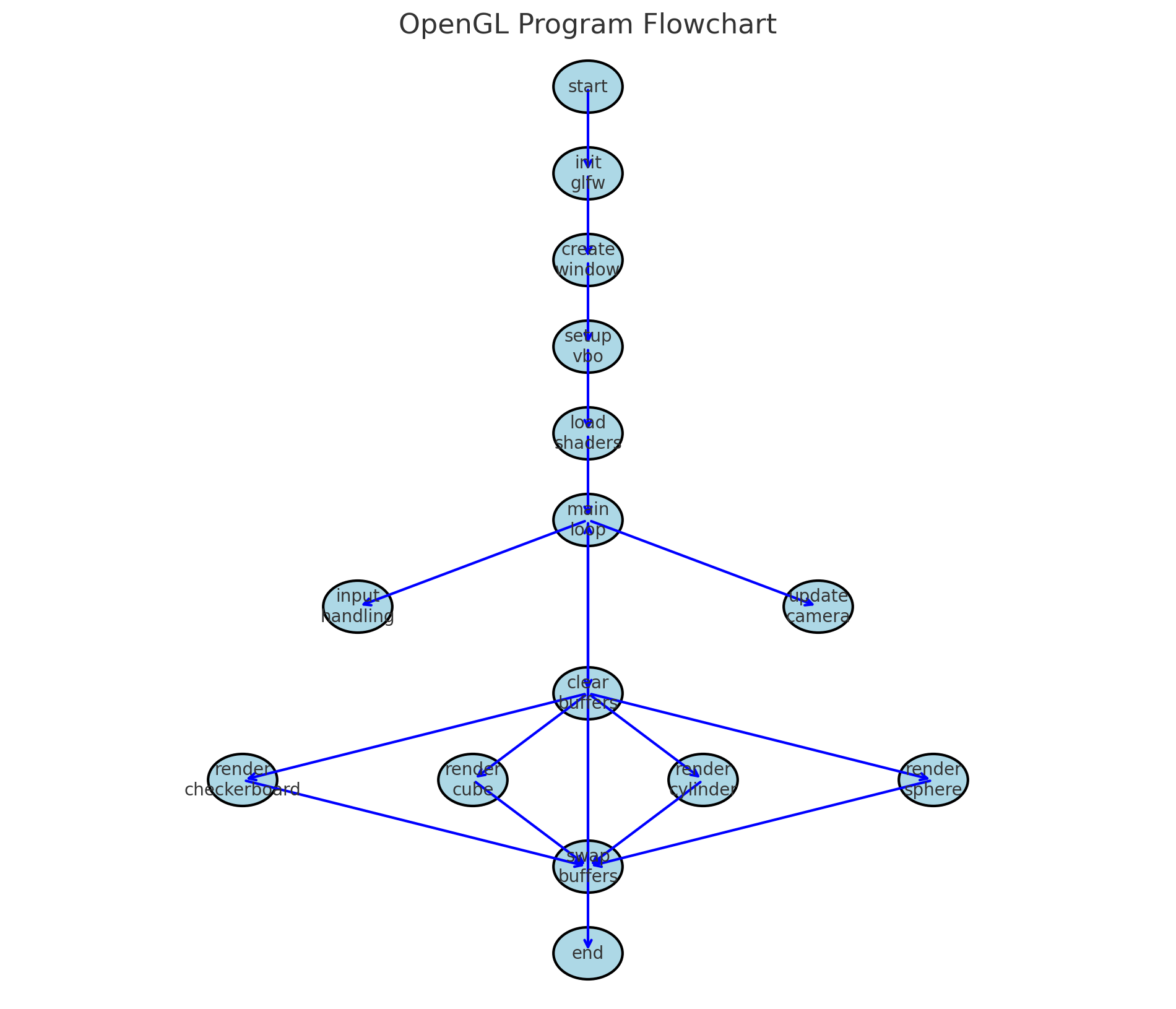
• Render Cylinder

• Render Sphere

• Swap Buffers

7. Clean up resources

8. End



Video Link: https://www.youtube.com/watch?v=7Dx8bn2Qyiw