Discussion 2
CSE 167

Outline

- Starter code walkthrough
- Project 1 part 4: Centering and Scaling
- Submission tips

Starter Code

- .h/.cpp files
  - main - Rendering loop
  - shader - Shader loading
  - Object - Rendered object base class
  - Cube - Sample spinning cube object
  - **Window** - Scene setup and event handling
  - **PointCloud** - Loads and renders .obj files
- Shader files
  - shader.vert - vertex shader
  - shader.frag - fragment shader

You will primarily be modifying these files

Starter Code - main

1. Set up window, callbacks, and settings
2. Initialize shaders
3. Initialize objects
4. Render Loop
   - alternate between updates and draws
5. Clean up

```
int main(void)
{
    // Create the GLFW window.
    GLFWwindow* window = Window::createWindow(640, 480);
    if (!window)
            exit(EXIT_FAILURE);

    // Print OpenGL and GLSL versions.
    print_versions();

    // Setup callbacks.
    setup_callbacks(window);

    // Setup OpenGL settings.
    setup_opengl_settings();

    // Initialize the shader program; exit if initialization fails.
    if (!Window::initializeProgram())
            exit(EXIT_FAILURE);

    // Initialize objects/pointers for rendering; exit if initialization fails.
    if (!Window::initializeObjects())
            exit(EXIT_FAILURE);

    // Loop while GLFW window should stay open.
    while (!glfwWindowShouldClose(window))
    {
            // Main render display callback. Rendering of objects is done here. (Draw)
            Window::displayCallback(window);

            // Idle callback. Updating objects, etc. can be done here. (Update)
            Window::idleCallback();
    }

    // destroy objects created
    Window::cleanUp();

    // Destroy the window.
    glfwDestroyWindow(window);

    // Terminate GLFW.
    glfwTerminate();

    exit(EXIT_SUCCESS);
}
```

1

2

3

4

5

Starter Code - Window

- Compiles and loads shader programs
  - If shaders fail to load, ensure shader files are at the locations given by these relative paths

```
bool Window::initializeProgram() {
        // Create a shader program with a vertex shader and a fragment shader.
        shaderProgram = LoadShaders("shaders/shader.vert", "shaders/shader.frag");

        // Check the shader program.
        if (!shaderProgram)
        {
                std::cerr << "Failed to initialize shader program" << std::endl;
                return false;
        }

        return true;
}
```

Starter Code - Window

- Initialize objects once upon starting up

```
bool Window::initializeObjects()
{
        // Create a cube of size 5.
        cube = new Cube(5.0f);

        // Create a point cloud consisting of cube vertices.
        cubePoints = new PointCloud("foo", 100);

        // Set cube to be the first to display
        currObj = cube;

        return true;
}
```

- Remember to deallocate!

```
void Window::cleanUp()
{
        // Deallcoate the objects.
        delete cube;
        delete cubePoints;

        // Delete the shader program.
        glDeleteProgram(shaderProgram);
}
```

Starter Code - Window

- idleCallback() updates all objects
  - e.g. spinning the cube/point cloud, morphing the point cloud
- displayCallback() renders the objects

```
void Window::idleCallback()
{
        // Perform any necessary updates here
        currObj->update();
}

void Window::displayCallback(GLFWwindow* window)
{
        // Clear the color and depth buffers
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        // Render the objects
        currObj->draw(view, projection, shaderProgram);

        // Gets events, including input such as keyboard and mouse or window resizing
        glfwPollEvents();

        // Swap buffers.
        glfwSwapBuffers(window);
}
```

Starter Code - Window

- Use to respond to keyboard presses

- Key Codes:
  - https://www.glfw.org/docs/latest/group__keys.html

```
void Window::keyCallback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
        /*
         * TODO: Modify below to add your key callbacks.
         */

        // Check for a key press.
        if (action == GLFW_PRESS)
        {
                switch (key)
                {
                case GLFW_KEY_ESCAPE:
                        // Close the window. This causes the program to also terminate.
                        glfwSetWindowShouldClose(window, GL_TRUE);
                        break;

                // switch between the cube and the cube pointCloud
                case GLFW_KEY_1:
                        currObj = cube;
                        break;
                case GLFW_KEY_2:
                        currObj = cubePoints;
                        break;

                default:
                        break;
                }
        }
}
```

Starter Code - PointCloud.cpp

- Constructor lines 34-50
  - We will learn more about these in Project 2

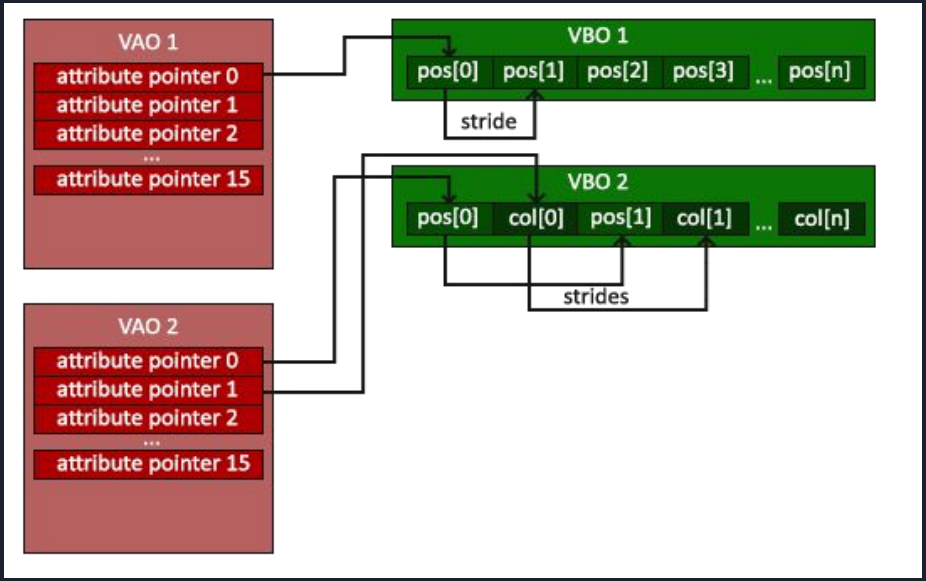
```
// Generate a Vertex Array (VAO) and Vertex Buffer Object (VBO)
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);

// Bind VAO
glBindVertexArray(VAO);

// Bind VBO to the bound VAO, and store the point data
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec3) * points.size(), points.data(), GL_STATIC_DRAW);
// Enable Vertex Attribute 0 to pass point data through to the shader
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), 0);

// Unbind the VBO/VAO
glBindBuffer(GL_ARRAY_BUFFER, 0);
glBindVertexArray(0);
```

- shader.vert line 9

```
layout (location = 0) in vec3 position;
```

Starter Code - PointCloud.cpp

- Vertex Buffer Object (VBO) holds data about each point
  - position, normals, color, etc.
- Vertex Array Object holds VBOs for a single rendered object



Starter Code - PointCloud.cpp

```cpp
void PointCloud::draw(const glm::mat4& view, const glm::mat4& projection, GLuint shader)
{
        // Actiavte the shader program
        glUseProgram(shader);

        // Get the shader variable locations and send the uniform data to the shader
        glUniformMatrix4fv(glGetUniformLocation(shader, "view"), 1, false, glm::value_ptr(view));
        glUniformMatrix4fv(glGetUniformLocation(shader, "projection"), 1, false, glm::value_ptr(projection));
        glUniformMatrix4fv(glGetUniformLocation(shader, "model"), 1, GL_FALSE, glm::value_ptr(model));
        glUniform3fv(glGetUniformLocation(shader, "color"), 1, glm::value_ptr(color));

        // Bind the VAO
        glBindVertexArray(VAO);

        // Set point size
        glPointSize(pointSize);

        // Draw the points
        glDrawArrays(GL_POINTS, 0, points.size());

        // Unbind the VAO and shader program
        glBindVertexArray(0);
        glUseProgram(0);
}
```

Starter Code - PointCloud.cpp

```
glUniform3fv(glGetUniformLocation(shader, "color"), 1, glm::value_ptr(color));
```

- glGetUniformLocation() gets the location of a uniform variable in the specified shader
- glUniformXXX() specifies a value for the uniform variable

```
uniform vec3 color;
```

- accessing a uniform variable (line 8 of shader.frag)

Part 4 - Centering

- Traverse through points and find max/min in each dimension
- Shift all points off center point
  - Method 1: Loop through and adjust each point
  - Method 2: glm::translate() to create transformation matrix

Part 4 - Scaling

- Find maximum distance of points to center of model
- Scale object up/down off some scale factor (can be found through trial and error)
  - Method 1: Loop through points and adjust to scale
  - Method 2: glm::scale() to create transformation matrix

Demo of Part 4

Extra Credit

- Maximum 10 points
- You can complete all of them for fun, but you will only get credit for up to 10 points

Submission Tips

- Project 1 outline updated w/ submission instructions
- What to turn into canvas:
  - Video demonstrating functionality
  - Zipped up source code and executable
    - (.cpp, .h, executable, shaders)

Submission Tips - Video

- Windows
  - Recommended to use OBS Studio
- Mac
  - Quicktime

Submission Tips - Extra

- Include comment on functionality implemented
  - Ex 1: I've done the base project with no issues. No extra credit.
  - Ex 2: Everything works except an issue with x, I couldn't get y to work properly.
  - Ex 3: Sections 1-2 and 4 work.
  - Ex 4: The base project is complete and I did z for extra credit.

Submission Tips

- Don’t wait until the last minute!
- Save extra time for figuring out how to create a video

Any questions?