CS 450: Assignment 05

Programming Assignments (95%)

- Copy src/app/Assign04.cpp and name it **src/app/Assign05.cpp**
    - Similar to before, make sure the shaders are loaded from the **shaders/Assign05** folder (instead of shaders/Assign04)
- Make a copy of the shaders/Assign04 folder and name it **shaders/Assign05**
- Modify **CMakeLists.txt** by adding the following lines to the end of the file:

```
add_executable(Assign05 ${GENERAL_SOURCES} "./src/app/Assign05.cpp")
target_link_libraries(Assign05 ${ALL_LIBRARIES})
install(TARGETS Assign05 RUNTIME DESTINATION bin/Assign05)
install(DIRECTORY shaders/Assign05 DESTINATION bin/Assign05/shaders)
```

- Make sure the sample configures, compiles, and runs as-is

Basic.vs

- Add uniform variable *viewMat* of type mat4
- Add uniform variable *projMat* of type mat4
- For gl_Position, apply the model, view, and projection transformations to objPos.
    - REMEMBER: RIGHT-TO-LEFT multiplication order!

Assign05.cpp

- **Add the following globals:**
    - A glm::vec3 to hold the camera position (e.g., eye)
        - *Default value:* (0,0,1)
    - A glm::vec3 to hold the camera's look-at point (e.g., lookAt)
        - *Default value:* (0,0,0)
        - **NOTE: This is NOT the direction the camera is facing! This is the point the camera is focusing on!**
    - A glm::vec2 to hold the last mouse position (e.g., mousePos)
        - You will get the initial value of this in the main() function later.

- Add the following function for generating a transformation to rotate around an arbitrary point and axis: **glm::mat4 makeLocalRotate(glm::vec3 offset, glm::vec3 axis, float angle)**
    - Generate transformation matrices (with glm) and form a composite transformation to perform the following IN ORDER:
        - Translate by NEGATIVE offset
        - Rotate *angle* around *axis*

                - **REMEMBER TO CONVERT *angle* to RADIANS!!!!**
            - Translate by offset
        - Return the composite transformation

- Add a mouse cursor movement callback: **static void mouse_position_callback(GLFWwindow* window, double xpos, double ypos)**
    - Get RELATIVE mouse motion
        - Subtract mouse position (xpos, ypos) from previous mouse position (global mousePos) → glm::vec2 relMouse
        - Use glfwGetFramebufferSize() to acquire the current framebuffer size
    - ***As long as the framebuffer has width and height greater than zero:***
        - Divide relMouse.x by current framebuffer width and relMouse.y by current framebuffer height to get scaled relative mouse motion
            - Make sure you do not do integer division!
        - Use relative mouse motion to rotate camera (use makeLocalRotate to get the appropriate matrix transformations):
            - RELATIVE X MOTION → rotate around GLOBAL Y axis
                - *Point to rotate:* lookAt
                - *Offset to rotate around:* eye
                - *Angle (degrees):* 30.0f * relative X mouse motion
                - *Axis:* glm::vec3(0,1,0)
            - RELATIVE Y MOTION → rotate around LOCAL X axis
                - *Point to rotate:* lookAt
                - *Offset to rotate around:* eye
                - *Angle (degrees):* 30.0f * relative Y mouse motion
                - *Axis:* cross product of camera direction and GLOBAL Y axis → LOCAL "X" axis
                    - *Camera direction:* lookAt - eye
            - NOTE: Both eye and lookAt are glm::vec3 values; in order to multiply by a glm::mat4, you will need to convert to and from glm::vec4:
                - *glm::vec3 to glm::vec4 (as point):*
                  glm::vec4 lookAtV = glm::vec4(lookAt, 1.0);
                - *glm::vec4 to glm::vec3:*
                  lookAt = glm::vec3(lookAtV);
    - Either way, store new current mouse position (xpos, ypos) in the global variable mousePos

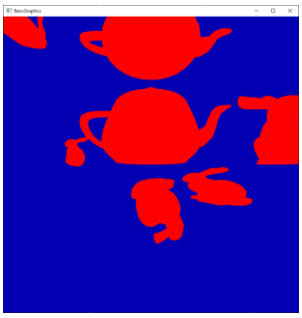
- **Add keys to your GLFW key callback function:**
    - **NOTE: For both camera direction and local X axis, NORMALIZE vectors before factoring in speed!**
    - If the action is either GLFW_PRESS or GLFW_REPEAT, add checks for the following keys:
        - GLFW_KEY_W
            - Move FORWARD in current camera direction
                - *Points to change:* lookAt, eye
                - *Camera direction:* lookAt - eye
                - *Speed:* 0.1
        - GLFW_KEY_S
            - Move BACKWARD in current camera direction
                - *Points to change:* lookAt, eye
                - *Camera direction:* lookAt - eye
                - *Speed:* 0.1
        - GLFW_KEY_D
            - Move RIGHT in LOCAL X direction (i.e., positive)
                - *Points to change:* lookAt, eye
                - *Movement axis:* cross product of camera direction and GLOBAL Y axis
                - *Speed:* 0.1
        - GLFW_KEY_A
            - Move LEFT in LOCAL X direction (i.e., negative)
                - *Points to change:* lookAt, eye
                - *Movement axis:* cross product of camera direction and GLOBAL Y axis
                - *Speed:* 0.1
- **In the main function:**
    - **Get the initial position of the mouse AFTER the GLFW window is created and GLEW setup:**
        - *double mx, my;*
        - *glfwGetCursorPos(window, &mx, &my);*
        - *mousePos = glm::vec2(mx, my);*
    - **Call glfwSetCursorPosCallback() to appropriately set the mouse cursor motion function**
    - **Hide the cursor**
        - *glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);*
    - **AFTER the creation of the shader program but BEFORE the rendering loop:**
        - Get the view matrix location using glGetUniformLocation()

        - Get the projection matrix location using glGetUniformLocation()

    - **INSIDE the drawing loop, AFTER the call to glUseProgram() but BEFORE the call to renderScene():**
        - Create a glm::mat4 for the **view matrix using glm::lookAt()**
            - Parameters “eye” and “center” should come from the global camera position (eye) and look-at point
            - Parameter “up” should be glm::vec3(0,1,0) (y axis)
        - Pass the **view matrix to the shader using glUniformMatrix4fv()**
        - Calculate the **aspect ratio** as the framebuffer width divided by height
            - NOTE: If either width or height are zero, set aspect ratio to 1.0. **Do NOT divide by zero!**
            - **Make sure to do FLOATING-POINT DIVISION!**
        - Create a glm::mat4 for the **projection matrix using glm::perspective()**
            - *FOV:* 90.0f degrees (IN RADIANS!)
            - *Aspect:* aspect ratio calculated before
            - *Near plane:* 0.01f
            - *Far plane:* 50.0f
        - Pass the **projection matrix to the shader using glUniformMatrix4fv()**

Screenshot (5%)

You should be able to rotate your view with the mouse and move forward/strafe left/backward/strafe right with the WASD keys. Remember that movement with the keys should be RELATIVE to your current view. For this part of the assignment, **upload ONE screenshot** of the application window when it first loads **bunnyteatime.glb**



Grading

Your OVERALL assignment grade is weighted as follows:

- 95% - Programming
- 5% - Screenshot