CS 450: Assignment 04

Setup

- Copy src/app/Assign03.cpp and name it **src/app/Assign04.cpp**
- Replace “Assign03” in the application name and window title with “**Assign04**”
- Replace the name "Assign03RenderEngine" with "**Assign04RenderEngine**"
- Make a copy of the vulkanshaders/Assign03 folder and name it **vulkanshaders/Assign04**
- Modify **CMakeLists.txt** by adding the following line to the end of the file:
    - CREATE_VULKAN_EXECUTABLE(Assign04)
- Make sure the program configures, compiles, and runs as-is

Assign04.cpp

- Add the following includes:
    - **#include "VKUniform.hpp"**

- Add the following to the **SceneData** struct:
    - 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)
    - A glm::mat4 to hold the current view matrix
        - *Default value:* identity matrix (glm::mat4(1.0f))
    - A glm::mat4 to hold the current projection matrix
        - *Default value:* identity matrix (glm::mat4(1.0f))

- Create a struct to hold vertex shader UBO *host* data: **UBOVertex**
    - Add one field for the view matrix: **alignas(16) glm::mat4 viewMat**
    - Add one field for the projection matrix: **alignas(16) glm::mat4 projMat**

- 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 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 (sceneData.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:* sceneData.lookAt
                - *Offset to rotate around:* sceneData.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:* sceneData.lookAt
                - *Offset to rotate around:* sceneData.eye
                - *Angle (degrees):* 30.0f * relative Y mouse motion
                - *Axis:* cross product of camera direction and GLOBAL Y axis → LOCAL "X" axis
                    - *Camera direction:* sceneData.lookAt – sceneData.eye
            - NOTE: Both sceneData.eye and sceneData.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(sceneData.lookAt, 1.0);
                - *glm::vec4 to glm::vec3:*
                  sceneData.lookAt = glm::vec3(lookAtV);
    - Either way, store new current mouse position (xpos, ypos) in sceneData.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:* sceneData.lookAt, sceneData.eye
        - *Camera direction:* sceneData.lookAt - sceneData.eye
        - *Speed:* 0.1
    - GLFW_KEY_S
      - Move BACKWARD in current camera direction
        - *Points to change:* sceneData.lookAt, sceneData.eye
        - *Camera direction:* sceneData.lookAt - sceneData.eye
        - *Speed:* 0.1
    - GLFW_KEY_D
      - Move RIGHT in LOCAL X direction (i.e., positive)
        - *Points to change:* sceneData.lookAt, sceneData.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:* sceneData.lookAt, sceneData.eye
        - *Movement axis:* cross product of camera direction and GLOBAL Y axis
        - *Speed:* 0.1

- Modify **Assign04RenderEngine**:
  - Add the following protected instance variables:
    - **UBOVertex hostUBOVert**
      - Holds HOST vertex shader UBO data
    - **UBOData deviceUBOVert**
      - Holds DEVICE vertex shader UBO data
    - **vk::DescriptorPool descriptorPool**
      - Memory manager for descriptor sets
    - **vector<vk::DescriptorSet> descriptorSets**
      - List of descriptor sets
  -

- Please note that the following are already defined in VulkanRenderEngine:
    - **VulkanInitData vkInitData**
        - Initial setup data, like the device and physical device
    - **const int MAX_FRAMES_IN_FLIGHT**
        - Maximum number of frames-in-flight
    - **unsigned int currentImage**
        - Current image (about to be) in flight
    - **VulkanPipelineData pipelineData**, which contains:
        - **vector<vk::DescriptorSetLayout> descriptorSetLayouts**
            - This is set during ***VulkanRenderEngine's initialize(),*** through a call to ***createVulkanPipelineData()*** and finally ***getDescriptorSetLayouts()***
- Add to: **initialize()**; after the successful call to VulkanRenderEngine::initialize(params), do the following:
    - **We will be creating a SEPARATE SET of items for each frame-in-flight.**
        - **UBOData struct → already contains lists**
        - **List of descriptor sets**
    - Create **deviceUBOVert (instance variable)** using the function **createVulkanUniformBufferData()**
        - Use the device and physicalDevice from vkInitData
        - Size should be sizeof(UBOVertex)
        - Frames-in-flight should be MAX_FRAMES_IN_FLIGHT
    - **Create the descriptor pool (instance variable):**
        - Create a **vector of vk::DescriptorPoolSize objects** and add one:
            - Type is vk::DescriptorType::eUniformBuffer
            - Descriptor count is (UBO count * frames-in-flight)
                - For now: 1*MAX_FRAMES_IN_FLIGHT
        - Create the **vk::DescriptorPoolCreateInfo**
            - Use setPoolSizes(poolSizes) to set the pool sizes
            - Use setMaxSets(MAX_FRAMES_IN_FLIGHT)
        - Create the actual **descriptorPool** using **vkInitData.device.createDescriptorPool(poolCreateInfo)**
    - **Create the descriptor sets (instance variable):**
        - Create a local **vector of vk::DescriptorSetLayout objects** and add one per frame-in-flight:
            - Only ONE DescriptorSetLayout object was needed during pipeline creation (same for all frames)
                - Thus, getDescriptorSetLayouts() returns a list of ONE DescriptorSetLayout

        - Now, however, we are creating SEPARATE DescriptorSet objects (one for each frame) that all use the same layout
            - Therefore, add the first element of pipelineData.descriptorSetLayouts to this local list of DescriptorSetLayouts, once for each frame-in-flight
      - Create a **vk::DescriptorSetAllocateInfo** object
        - Use setDescriptorPool(descriptorPool)
        - Use setDescriptorSetCount(MAX_FRAMES_IN_FLIGHT)
        - Use setSetLayouts(localLayoutList)
      - Create the actual **descriptorSets** using **vkInitData.device.allocateDescriptorSets(allocInfo)**
    - For each frame-in-flight *index*:
      - Create a **vector of vk::WriteDescriptorSet objects → writes**
      - Create a **vk::DescriptorBufferInfo object → bufferVertexInfo**
        - Use setBuffer(deviceUBOVert.bufferData[index].buffer
        - Use setOffset(0)
        - Use setRange(sizeof(UBOVertex))
      - Create a **vk::WriteDescriptorSet object → descVertWrites**
        - **Use setDstSet(descriptorSets[index])**
        - **Use setDstBinding(0)**
        - **Use setDstArrayElement(0)**
        - **Use setDescriptorType(vk::DescriptorType::eUniformBuffer)**
        - **Use setDescriptorCount(1)**
        - **Use setBufferInfo(bufferVertInfo)**
      - Add **descVertWrites** to **writes**
      - Update the descriptor sets with **vkInitData.device.updateDescriptorSets(writes, {});**
  - Add to: **~Assign04RenderEngine():**
    - Destroy the descriptor pool:
      - **vkInitData.device.destroyDescriptorPool(descriptorPool);**
    - Clean up the UBO device data:
      - **cleanupVulkanUniformBufferData(vkInitData.device, deviceUBOVert);**
  - Override: **vector<vk::DescriptorSetLayout> getDescriptorSetLayouts()**
    - Create a **vector of vk::DescriptorSetLayoutBinding objects → allBindings**:
      - One **vk::DescriptorSetLayoutBinding** for the vertex shader UBO:
        - binding = 0
        - descriptorType = vk::DescriptorType::eUniformBuffer
        - descriptorCount = 1
        - stageFlags = vk::ShaderStageFlagBits::eVertex
        - pImmutableSamplers = nullptr

        - Create a vk::DescriptorSetLayout using **vkInitData.device.createDescriptorSetLayout(vk::DescriptorSetLayoutCreateInfo({}, allBindings));**
        - Return a vector only containing the one vk::DescriptorSetLayout

    - Create a new method: **virtual void updateUniformBuffers(SceneData *sceneData, vk::CommandBuffer &commandBuffer)**
        - Copy view matrix and projection matrix from sceneData into appropriate fields of hostUBOVert
        - Invert Y for the projection matrix:
            - **hostUBOVert.projMat[1][1] *= -1;**
        - Copy UBO host data into the CORRECT device UBO data:
            - **memcpy(deviceUBOVert.mapped[this->currentImage], &hostUBOVert, sizeof(hostUBOVert));**
        - Bind the CORRECT descriptor sets:
            - **commandBuffer.bindDescriptorSets(vk::PipelineBindPoint::eGraphics, pipelineData.pipelineLayout, 0, descriptorSets[currentImage], {});**
    - Add to: **recordCommandBuffer()**
        - RIGHT before the call to renderScene(), call:
            - **updateUniformBuffers(sceneData, commandBuffer);**

- **In the main function:**
    - **Get the initial position of the mouse AFTER the GLFW window is created:**
        - *double mx, my;*
        - *glfwGetCursorPos(window, &mx, &my);*
    - *sceneData.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);*
    - **INSIDE the drawing loop, BEFORE the call to renderEngine->drawFrame(&sceneData):**
        - Set the value of **sceneData.viewMat using glm::lookAt()**
            - Parameters "eye" and "center" should come from sceneData.eye and sceneData.lookAt
            - Parameter "up" should be glm::vec3(0,1,0) (y axis)
        - 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!**

- Set the value of **sceneData.projMat using glm::perspective()**
  - *FOV:* 90.0f degrees (IN RADIANS!)
  - *Aspect:* aspect ratio calculated before
  - *Near plane:* 0.01f
  - *Far plane:* 50.0f

shader.vert

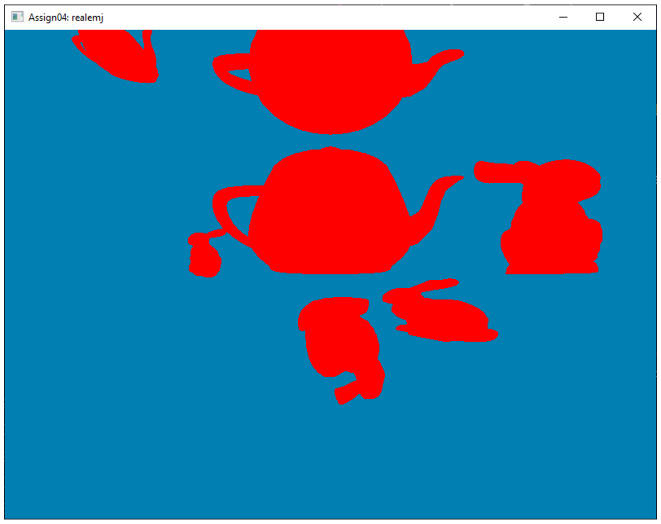
- Add the appropriate UBO struct
- Make sure that gl_Position receives the input vertex position multiplied by the model, view, and projection matrices
  - REMEMBER: RIGHT-TO-LEFT multiplication order!

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 the screenshot, you will load **bunnyteatime.glb** and take a screenshot when the application first loads: **Assign04.png**.

Copy the image to the **screenshots/** folder.



Grading

Your OVERALL assignment grade is weighted as follows:

- 95% - Programming
- 5% - Screenshots