CS 450: Assignment 05

Setup

- Copy src/app/Assign04.cpp and name it **src/app/Assign05.cpp**
- Replace "Assign04" in the application name and window title with "**Assign05**"
- Replace the name "Assign04RenderEngine" with "**Assign05RenderEngine**"
- Make a copy of the vulkanshaders/Assign04 folder and name it **vulkanshaders/Assign05**
- Modify **CMakeLists.txt** by adding the following line to the end of the file:
    - CREATE_VULKAN_EXECUTABLE(Assign05)
- Make sure the program configures, compiles, and runs as-is

Assign05.cpp

- Add to the Vertex struct a field for the normal:
    - **glm::vec3 normal**

- Add to the UPushVertex struct a field for the normal matrix:
    - **alignas(16) glm::mat4 modelMat**

- Create a struct to hold data for a point light: **PointLight**
    - A field for the light's world position: **alignas(16) glm::vec4 pos**
    - A field for the light's view position: **alignas(16) glm::vec4 vpos**
    - A field for the light's color: **alignas(16) glm::vec4 color**

- Create a struct to hold fragment shader UBO *host* data: **UBOFragment**
    - A point light instance: **PointLight light**
    - How metallic the surface is: **alignas(4) float metallic**
    - How rough the surface is: **alignas(4) float roughness**

- Add to the **SceneData** struct:
    - A PointLight struct with an initial world position of (0.5,0.5,0.5,1.0) and color of (1,1,1,1)
    - A float metallic (default: 0.0f)
    - A float roughness (default: 0.1f)

- Add keys to your **GLFW key callback function**:
  - If the action is either GLFW_PRESS or GLFW_REPEAT, add checks for the following keys:
    - GLFW_KEY_1
      - Set sceneData.light.color to (1,1,1,1) (white)
    - GLFW_KEY_2
      - Set sceneData.light.color to (1,0,0,1) (red)
    - GLFW_KEY_3
      - Set sceneData.light.color to (0,1,0,1) (green)
    - GLFW_KEY_4
      - Set sceneData.light.color to (0,0,1,1) (blue)
    - GLFW_KEY_V
      - Decrease sceneData.**metallic** by 0.1f
      - **Clamp the value to 0.0f if it drops below that.**
    - GLFW_KEY_B
      - Increase sceneData.**metallic** by 0.1f
      - **Clamp the value to 1.0f if it goes above that.**
    - GLFW_KEY_N
      - Decrease sceneData.**roughness** by 0.1f
      - **Clamp the value to 0.1f if it drops below that.**
    - GLFW_KEY_M
      - Increase sceneData.**roughness** by 0.1f
      - **Clamp the value to 0.7f if it goes above that.**

- Modify **Assign05RenderEngine**:
  - Add the following protected instance variables:
    - **UBOFragment hostUBOFrag**
      - Holds HOST fragment shader UBO data
    - **UBOData deviceUBOFrag**
      - Holds DEVICE fragment shader UBO data

  - Add to: **initialize()**; after the successful call to VulkanRenderEngine::initialize(params), do the following:
    - Create **deviceUBOFrag (instance variable)** using the function **createVulkanUniformBufferData()**
      - Use the device and physicalDevice from vkInitData
      - Size should be sizeof(UBOFragment)
      - Frames-in-flight should be MAX_FRAMES_IN_FLIGHT
    - **CHANGE the following:**
      - Change the one **vk::DescriptorPoolSize object** so that the descriptor count is (2*MAX_FRAMES_IN_FLIGHT) = (UBO count * frames-in-flight)

- Inside of the loop for each frame-in-flight *index* that handles writing descriptor set info:
  - AFTER creating and adding the vk::WriteDescriptorSet for the vertex UBO, but BEFORE the call to updateDescriptorSets():
    - Create a **vk::DescriptorBufferInfo object → bufferFragInfo**
      - **Use setBuffer(deviceUBOFrag.bufferData[index].buffer**
      - **Use setOffset(0)**
      - **Use setRange(sizeof(UBOFragment))**
    - Create a **vk::WriteDescriptorSet object → descFragWrites**
      - **Use setDstSet(descriptorSets[index])**
      - **Use setDstBinding(1)**
      - **Use setDstArrayElement(0)**
      - **Use setDescriptorType(vk::DescriptorType::eUniformBuffer)**
      - **Use setDescriptorCount(1)**
      - **Use setBufferInfo(bufferFragInfo)**
    - Add **descFragWrites** to **writes**

- Add to: **~Assign05RenderEngine():**
  - Clean up the UBO device data:
    - **cleanupVulkanUniformBufferData(vkInitData.device, deviceUBOFrag);**

- Change: **vector<vk::DescriptorSetLayout> getDescriptorSetLayouts()**
  - Add another item to your **vector of vk::DescriptorSetLayoutBinding objects → allBindings**:
    - **vk::DescriptorSetLayoutBinding** for the fragment shader UBO:
      - **binding = 1**
      - descriptorType = vk::DescriptorType::eUniformBuffer
      - descriptorCount = 1
      - **stageFlags = vk::ShaderStageFlagBits::eFragment**
      - pImmutableSamplers = nullptr

- Override: **virtual AttributeDescData getAttributeDescData() override**
  - Create an instance of **AttributeDescData** struct → **attribDescData**
    - This is defined in VKMesh.hpp/cpp
  - Set attribDescData.**bindDesc** to:
    - **vk::VertexInputBindingDescription(0, sizeof(Vertex), vk::VertexInputRate::eVertex)**

    - Clear attribDescData.**attribDesc** and add instances of **vk::VertexInputAttributeDescription**:
        - **Location=0, binding=0, vk::Format:: eR32G32B32Sfloat, offsetof(Vertex, pos)**
        - **Location=1, binding=0, vk::Format:: eR32G32B32A32Sfloat, offsetof(Vertex, color)**
        - **Location=2, binding=0, vk::Format:: eR32G32B32Sfloat, offsetof(Vertex, normal)**
    - Return **attribDescData**

  - Change: **renderScene()**
    - Calculate the **normal matrix** as:
        - The glm::mat4…
        - Of the transpose…
        - Of the inverse…
        - Of the glm::mat3
          of sceneData->viewMat * tmpModel
    - Set the normMat field of your **UPushVertex** struct

  - Change: **updateUniformBuffers()**
    - Copy the light, metallic, and roughness values from the sceneData into the appropriate fields of hostUBOFrag
    - Copy UBO host data into the CORRECT device UBO data for the fragment data:
        - **memcpy(deviceUBOFrag.mapped[this->currentImage], &hostUBOFrag, sizeof(hostUBOFrag));**

- **Change: extractMeshData()**
  - In your loop that sets the vertex data:
    - Set the color to **yellow (1,1,0,1)** for all vertices
    - Set the normal using data from **mesh->mNormals[i]**

- **In the main function:**
  - **INSIDE the drawing loop, BEFORE the call to renderEngine->drawFrame(&sceneData):**
    - Update the light's view position with the current view matrix and the light's world position.
        - I.e., setting **sceneData.light.vpos**

shader.vert

- **BEFORE the main() function:**
    - Add a new input variable: **layout(location=2) in vec3 inNormal;**
    - Add **mat4 normMat** to the **PushConstants** struct
    - Add new output variables:
        - **layout(location = 1) out vec4 interPos;**
        - **layout(location = 2) out vec3 interNormal;**

- **In main():**
    - Set **interPos** to equal the vertex position in **view coordinates**
    - Set **interNormal** to equal the **mat3(pc.normMat)*inNormal**

shader.frag

- **BEFORE the main() function:**
    - Add new input variables:
        - **layout(location = 1) in vec4 interPos;**
        - **layout(location = 2) in vec3 interNormal;**
    - Add a constant float for **PI = 3.14159265359**
    - Add a struct to hold a point light:
        - **struct PointLight {**
          vec4 pos;
          vec4 vpos;
          vec4 color;
          };
    - Add the appropriate UBO struct **UBOFragment** with **set = 0** and **binding = 1**

- Add a function: **vec3 getFresnelAtAngleZero(vec3 albedo, float metallic)**
    - This function calculates the external reflection R_F at incoming light angle 0.
    - Parameter metallic is assumed to be between 0 and 1.
        - If 0 → insulator (e.g., plastic), and albedo is diffuse color
        - If 1 → metal, and "albedo" becomes the specular color
    - Start with vec3 F0 at vec3(0.04)
        - Good default value for insulators
    - Use mix() function to interpolate between default F0 and albedo:
      F0 = mix(F0, albedo, metallic);
    - Return F0

- Add a function: **vec3 getFresnel(vec3 F0, vec3 L, vec3 H)**
    - This function returns the Fresnel reflectance given the light vector and half vector, assuming a starting value of F0 (i.e., $R_F(0)$).
    - Compute the max of 0 and the dot product of L and H → cosAngle
    - Use the **Schlick approximation** to calculate the Fresnel reflectance (see slide 38 of the PBR slides).
    - Return the computed value.

- Add a function: **float getNDF(vec3 H, vec3 N, float roughness)**
    - This function returns the Microgeometry Normal Distribution Function (NDF) value (i.e., how many microgeometry normals are aligned for reflection).
    - Use the GGX/Trowbirdge-Reitz NDF (see slide 45 of the PBR slides).
    - Return the computed value.

- Add a function: **float getSchlickGeo(vec3 B, vec3 N, float roughness)**
    - This is a helper function for getGF() (see slide 50 of the PBR slides).
    - Calculate k as $(\text{roughness} + 1)^2 / 8$
    - Calculate dot(N, B) / (dot(N, B)*(1 – k) + k)
    - Return computed value

- Add a function: **float getGF(vec3 L, vec3 V, vec3 N, float roughness)**
    - This function returns the Geometry Function value (i.e., how many microfacets are NOT shadowed or masked (see slide 50 of the PBR slides).
    - Compute GL = getSchlickGeo(L, N, roughness)
    - Compute GV = getSchlickGeo(V, N, roughness);
    - Return GL*GV

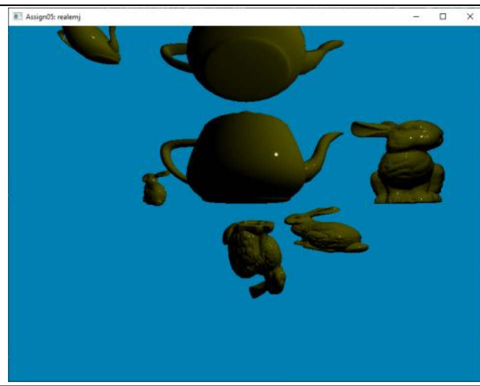
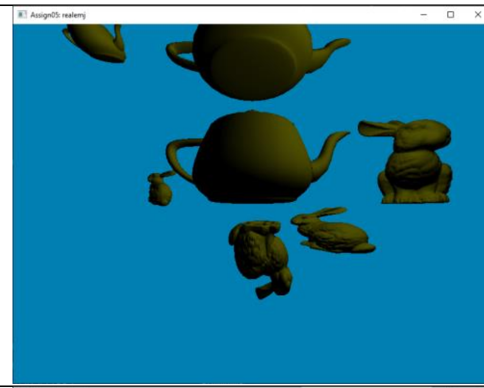
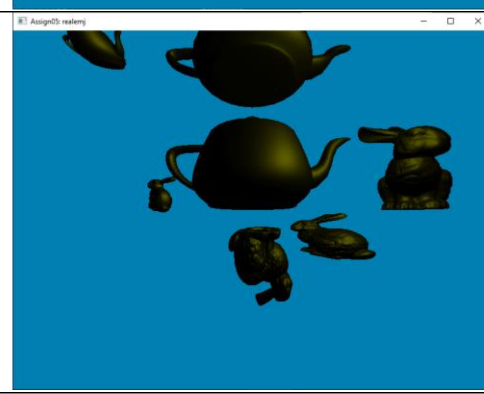
- **IN the main() function:**
    - Normalize interNormal → N
    - Calculate the light vector L as the **NORMALIZED** direction vector FROM **interPos** TO **ubo.light.vpos** (you will have to convert to vec3 at some point)
    - Set **vec3 baseColor = vec3(fragColor);**
    - Calculate the **normalized view vector V** (remember that **interPos** is in view space).
    - Calculate **F0** using **getFresnelAtAngleZero(baseColor, metallic)**.
    - Calculate the **normalized half-vector H**.
    - Calculate **Fresnel reflectance F** with **getFresnel(F0, L, H)**.
    - Set **specular color kS** to F.
    - Calculate the complete **diffuse color** as follows:
        - Set **diffuse color kD** to 1.0 – kS.
        - Multiply kD by (1.0 – metallic)
            - If metal → diffuse color does not exist.

- Multiply by baseColor.
- Divide by PI.

- Calculate the complete **specular reflection** (see slide 33 of the PBR slides) as follows:
  - Calculate **NDF** using **getNDF(H, N, roughness)**.
  - Calculate **G** using **getGF(L, V, N, roughness).**
  - Multiply kS by NDF and G.
  - Divide kS by **(4.0 * max(0, dot(N,L)) * max(0, dot(N,V))) + 0.0001.**
- Calculate final color as **finalColor** as **(kD + kS)*vec3(light.color)*max(0, dot(N,L)).**
- Set **outColor** to **vec4(finalColor, 1.0).**

Screenshots (5%)

Upload FOUR screenshots of the application window when it first loads **bunnyteatime.glb**:

	Roughness = 0.1	***Roughness = 0.7***
Metallic = 0		
Metallic = 1		

Use the following names and copy the images to the **screenshots/** folder:

- **Assign05_M0_R1.png**
- **Assign05_M0_R7.png**
- **Assign05_M1_R1.png**
- **Assign05_M1_R7.png**

Grading

Your OVERALL assignment grade is weighted as follows:

- 95% - Programming
- 5% - Screenshots