CIS565 GPU Programming and Architecture

Final Project Milestone II

Forward + with Vulkan

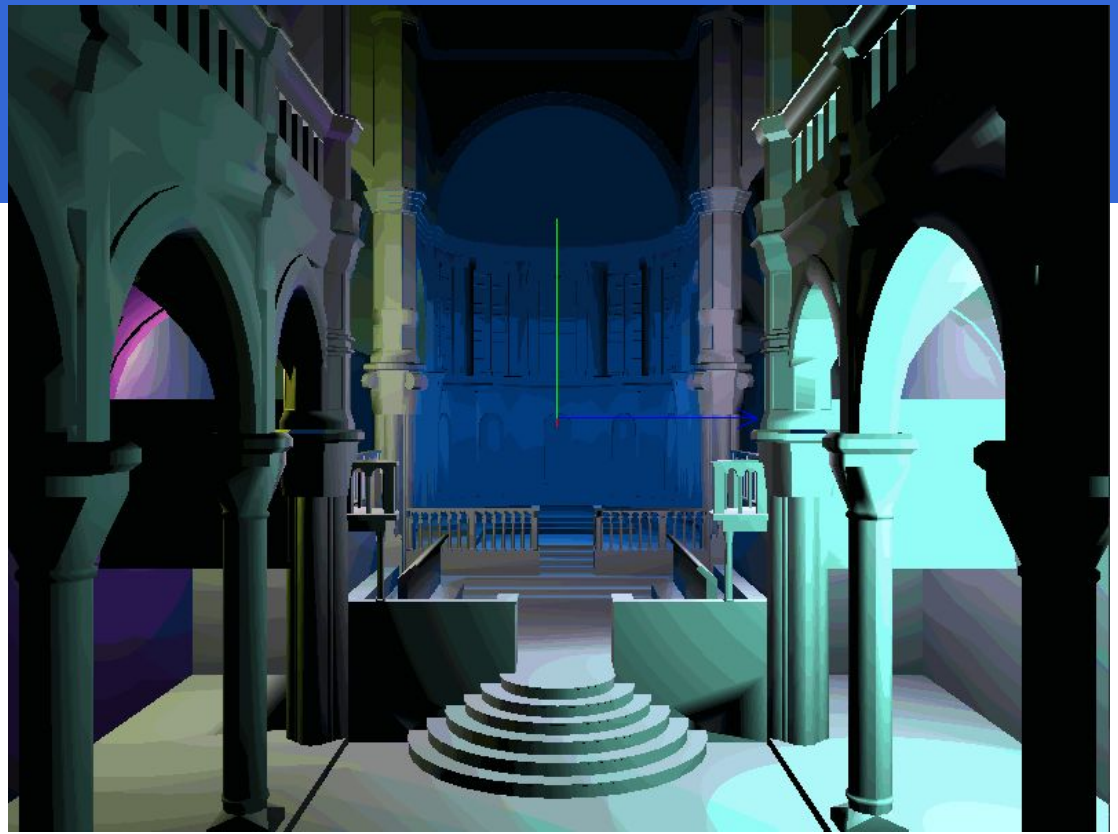
Zimeng Yang & Liang Peng

Instructor: Patrick Cozzi

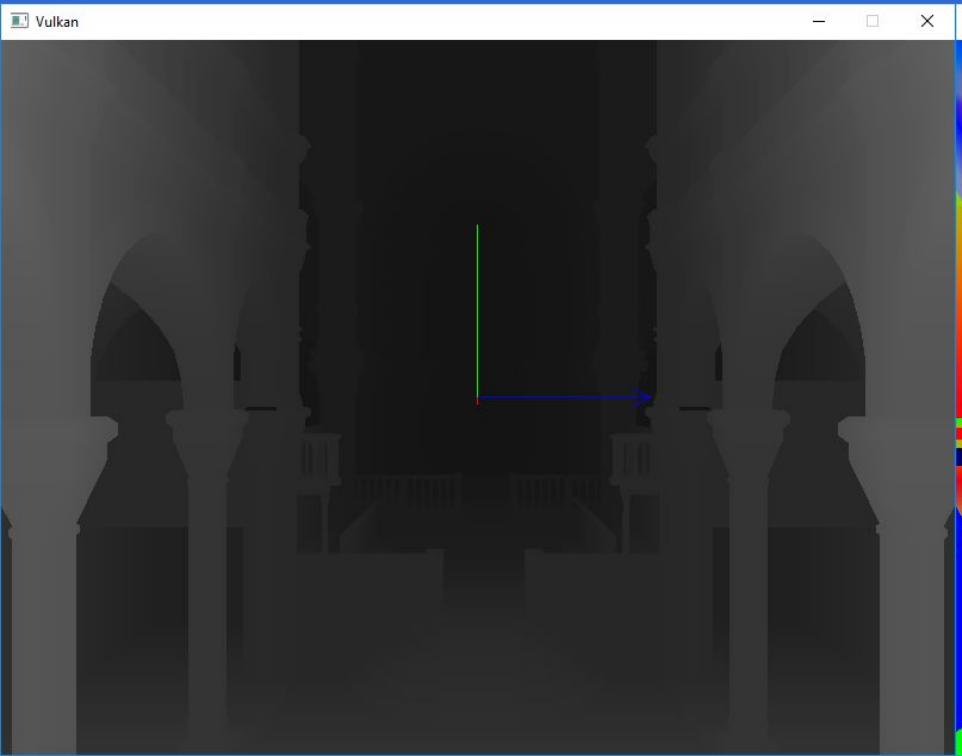
Finished Tasks

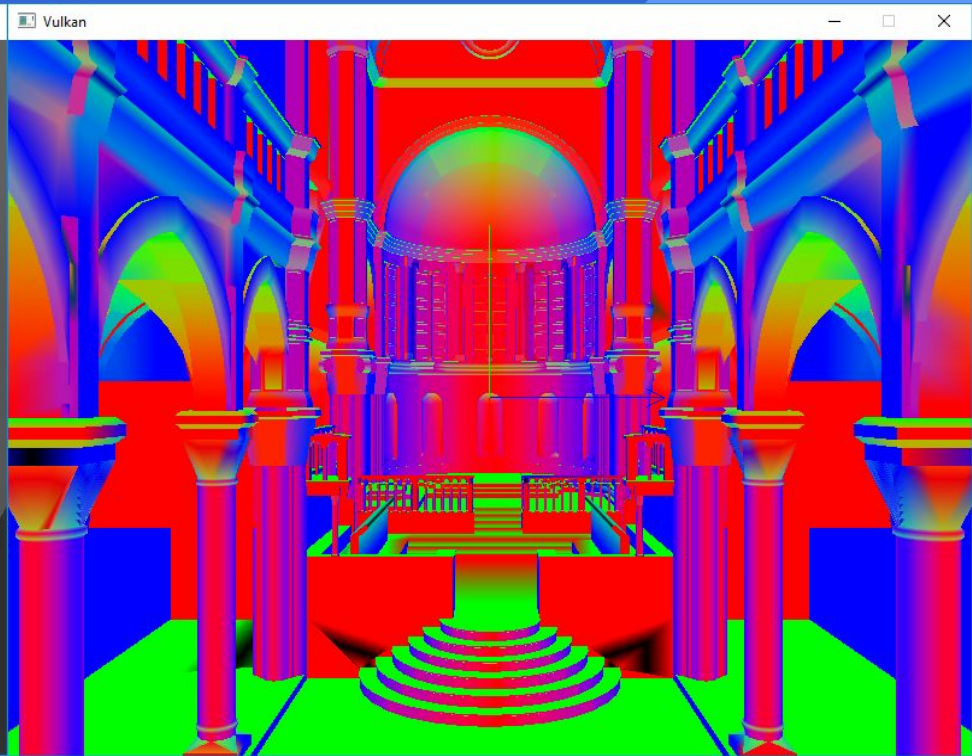
- Basix forward rendering with lighting
- Debug view: depth, normal
- Compute pipeline in progress

GIF



Screenshots





Working Code

```
//--------------------- update frag uniform buffer (lights)---------------------------------

for (int i = 0; i < fragLightInfos.numLights; ++i) {
        fragLightInfos.lights[i].pos.y += 0.0015f * ((float)lightMoveDirs[i] * 2.0f - 1.0f);

        if (fragLightInfos.lights[i].pos.y > 1.0f || fragLightInfos.lights[i].pos.y < -2.0f)
        {
                //fragLightInfos.lights[i].pos.y = 2.0f;
                lightMoveDirs[i] = !lightMoveDirs[i];
        }
}

vkMapMemory(device, fragLightsStagingBufferMemory, 0, sizeof(fragLightInfos), 0, &data);
memcpy(data, &fragLightInfos, sizeof(fragLightInfos));
vkUnmapMemory(device, fragLightsStagingBufferMemory);

copyBuffer(fragLightsStagingBuffer, fragLightsBuffer, sizeof(fragLightInfos));
```

```
layout(binding = 3) uniform LightInfos {
    LightInfo lights[8];
    int numLights;
} lightInfos;

// lighting
vec3 lightPos, lightDir, lightColor;
float dist, lightIntensity, NdotL, lightRadius;
for(int i=0;i<lightInfos.numLights;++i){
    lightPos = lightInfos.lights[i].pos.xyz;
    lightColor = lightInfos.lights[i].color.xyz;
    lightDir = lightPos - fragPosWorldSpace;
    lightIntensity = lightInfos.lights[i].pos.w;
    lightRadius = lightInfos.lights[i].color.w;

    dist = length(lightDir);
    lightDir = lightDir/dist;

    // attenuation
    float att = max(0, lightRadius - dist);

    NdotL = dot(normal,lightDir);

    finalColor += max(0, NdotL) * lightColor * att * lightIntensity;
}
```

Working Code

```
void createComputePipeline() {
    // shader module
    if (shaderModules.size() < 6) {
        shaderModules.resize(6);
    }

    // shader stage
    VkPipelineShaderStageCreateInfo shaderStageInfo = lo

    // descriptor set layout bindings
    VkDescriptorSetLayoutBinding bindings[2] = {};

    bindings[0].binding = 0;
    bindings[0].descriptorType = VK_DESCRIPTOR_TYPE_STOR
    bindings[0].descriptorCount = 1;
    bindings[0].stageFlags = VK_SHADER_STAGE_ALL;
    bindings[0].pImmutableSamplers = nullptr;

    bindings[1].binding = 1;
    bindings[1].descriptorType = VK_DESCRIPTOR_TYPE_STOR
    bindings[1].descriptorCount = 1;
    bindings[1].stageFlags = VK_SHADER_STAGE_ALL;
    bindings[1].pImmutableSamplers = nullptr;

    bindings[2].binding = 2;
    bindings[2].descriptorType = VK_DESCRIPTOR_TYPE_UNIF
    bindings[2].descriptorCount = 1;
    bindings[2].stageFlags = VK_SHADER_STAGE_ALL;
    bindings[2].pImmutableSamplers = nullptr;

    // descriptor set layout
    VkDescriptorSetLayoutCreateInfo descriptorLayoutInfo

    descriptorLayoutInfo.sType = VK_STRUCTURE_TYPE_DESCR
    descriptorLayoutInfo.pNext = nullptr;
    descriptorLayoutInfo.flags = NULL;
    descriptorLayoutInfo.bindingCount = 3;
```

```
    // descriptor set layout
    VkDescriptorSetLayoutCreateInfo descriptorLayoutInfo = {};

    descriptorLayoutInfo.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_C
    descriptorLayoutInfo.pNext = nullptr;
    descriptorLayoutInfo.flags = NULL;
    descriptorLayoutInfo.bindingCount = 3;
    descriptorLayoutInfo.pBindings = bindings;

    if (vkCreateDescriptorSetLayout(device, &descriptorLayoutInfo, nullptr
            computeDescriptorSetLayout.replace()) != VK_SUCCESS) {
        throw std::runtime_error("failed to create compute descriptor set
    }

    // pipeline layout
    VkPipelineLayoutCreateInfo pipelineLayoutInfo = {};

    pipelineLayoutInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_IN
    pipelineLayoutInfo.pNext = nullptr;
    pipelineLayoutInfo.flags = NULL;
    pipelineLayoutInfo.setLayoutCount = 1;
    pipelineLayoutInfo.pSetLayouts = &computeDescriptorSetLayout;

    if (vkCreatePipelineLayout(device, &pipelineLayoutInfo, nullptr,
            computePipelineLayout.replace()) != VK_SUCCESS) {
        throw std::runtime_error("failed to create compute pipeline layout
    }

    // compute pipeline
    VkComputePipelineCreateInfo pipelineInfo = {};

    pipelineInfo.sType = VK_STRUCTURE_TYPE_COMPUTE_PIPELINE_CREATE_INFO;
    pipelineInfo.pNext = nullptr;
    pipelineInfo.flags = VK_PIPELINE_CREATE_DISABLE_OPTIMIZATION_BIT;
    pipelineInfo.stage = shaderStageInfo;
    pipelineInfo.layout = computePipelineLayout;

    if (vkCreateComputePipelines(device, VK_NULL_HANDLE, 1, &pipelineInfo,
```

```
        nullptr, computePipeline.replace()) != VK_SUCCESS) {
        throw std::runtime_error("failed to create compute pipeline!");
    }

    // command pool
    QueueFamilyIndices queueFamilyIndices = findQueueFamilies(physicalDevic
    VkCommandPoolCreateInfo poolInfo = {};

    poolInfo.sType = VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO;
    poolInfo.queueFamilyIndex = queueFamilyIndices.computeFamily;
    poolInfo.flags = NULL;

    if (vkCreateCommandPool(device, &poolInfo, nullptr,
            computeCommandPool.replace()) != VK_SUCCESS) {
        throw std::runtime_error("failed to create compute command pool!");
    }

    // buffers
    createComputeBuffers();
}

void createComputeBuffers() {
    std::array<ComputeBufferObject, 2> computeBuffer = {};

    computeBuffer[0] = ComputeBufferObject{
        glm::vec3(1.f, 0.f, 0.f), glm::vec3(3.f, 0.f, 0.f), glm::vec3(.01f,
    computeBuffer[1] = ComputeBufferObject{
        glm::vec3(0.f, 3.f, 0.f), glm::vec3(0.f, 5.f, 0.f), glm::vec3(0.f,

    VkDeviceSize bufferSize = sizeof(ComputeBufferObject) * computeBuffer.s
    VDeleter<VkBuffer> stagingBuffer{ device, vkDestroyBuffer };
    VDeleter<VkDeviceMemory> stagingBufferMemory{ device, vkFreeMemory };

    createBuffer(bufferSize,
        VK_BUFFER_USAGE_TRANSFER_SRC_BIT,
        VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_COHEF
```

```
        VK_BUFFER_USAGE_TRANSFER_DST_BIT | VK_BUFFER_USAGE_VERTEX_
        VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT,
        computeBuffer1, computeBufferMemory1);

    copyBuffer(stagingBuffer, computeBuffer1, bufferSize);

    // compute buffer 2
    createBuffer(bufferSize,
        VK_BUFFER_USAGE_TRANSFER_DST_BIT | VK_BUFFER_USAGE_VERTEX_
        VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT,
        computeBuffer2, computeBufferMemory2);

    // compute uniforms buffer
    ComputeUniformBufferObject computeUniform = {
        8
    };

    bufferSize = sizeof(ComputeUniformBufferObject);
    stagingBuffer.replace();
    stagingBufferMemory.replace();

    createBuffer(bufferSize,
        VK_BUFFER_USAGE_TRANSFER_SRC_BIT,
        VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_H
        stagingBuffer, stagingBufferMemory);

    vkMapMemory(device, stagingBufferMemory, 0, bufferSize, 0, &da
    memcpy(data, &computeUniform, (size_t)bufferSize);
    vkUnmapMemory(device, stagingBufferMemory);

    createBuffer(bufferSize,
        VK_BUFFER_USAGE_TRANSFER_DST_BIT | VK_BUFFER_USAGE_UNIFORM
        VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT,
        computeUniformBuffer, computeUniformBufferMemory);

    copyBuffer(stagingBuffer, computeUniformBuffer, bufferSize);
```

Upcoming Milestones

- Milestone III
  - Update light position in compute shader
  - Compute frustum for each tile
  - Compute light list for each tile
- Final Presentation
  - Compute lighting in each tile
  - Performance Analysis

Thank you!