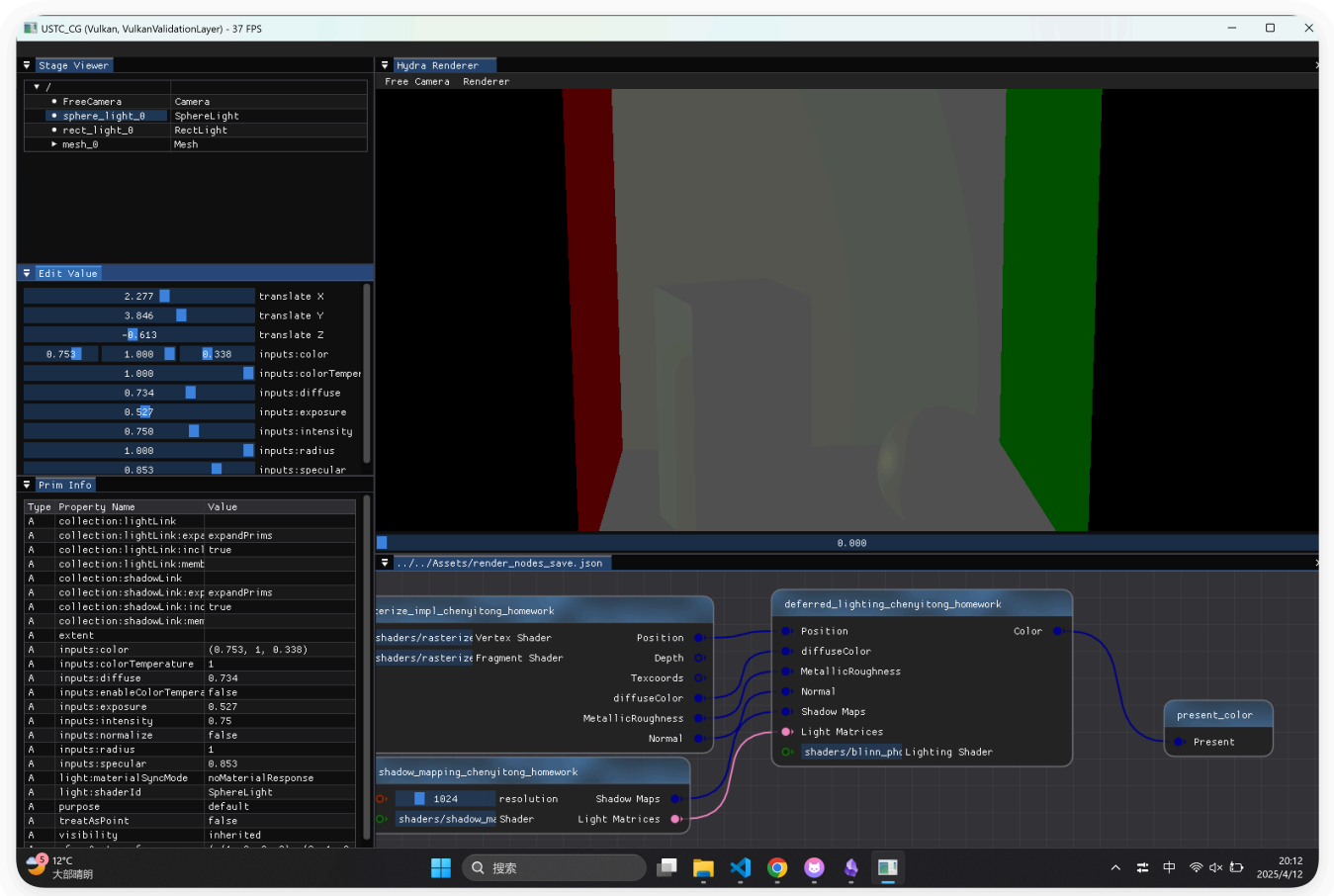
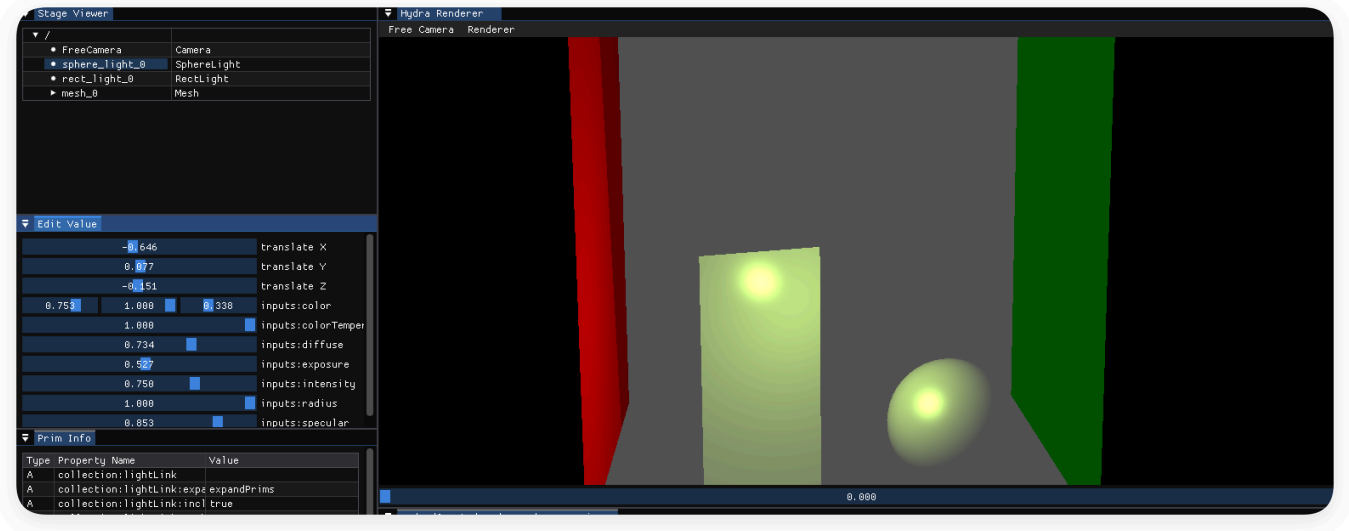
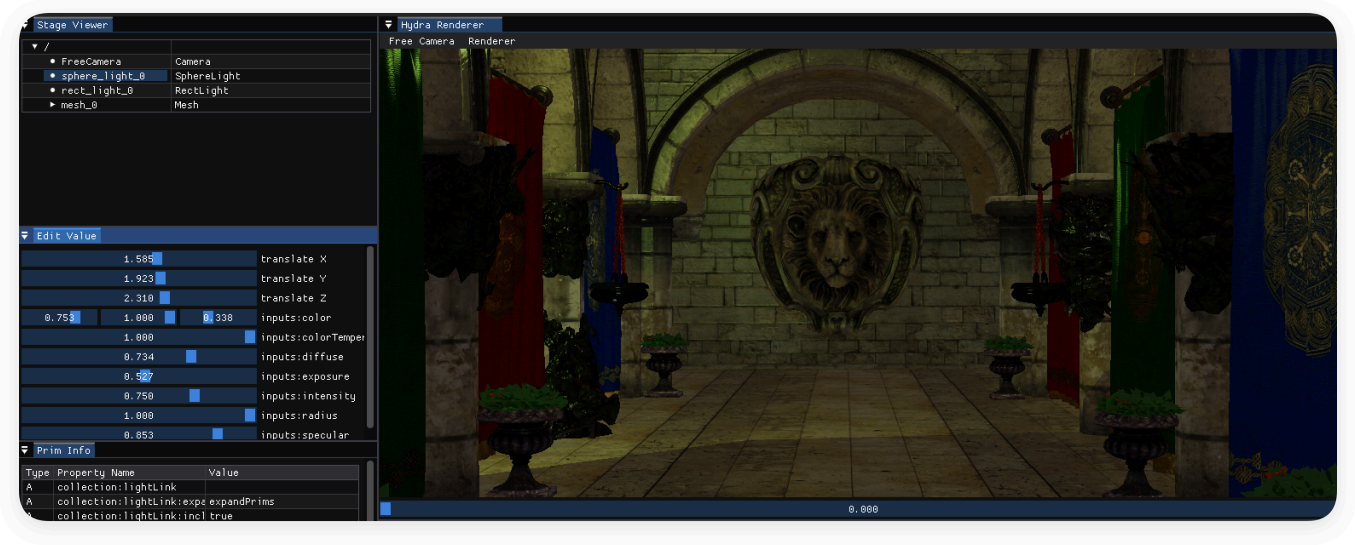
HW6 Report


USTC_CG (Vulkan, VulkanValidationLayer) - 37 FPS
Stage Viewer
FreeCamera Camera
sphere_light_0 SphereLight
rect_light_0 RectLight
mesh_0 Mesh
Edit Value
translate X
translate Y
translate Z
inputs:color
inputs:diffuse
inputs:exposure
inputs:intensity
inputs:radius
inputs:specular
Prim Info
Hydra Renderer
Free Camera Renderer
../../Assets/render_nodes_save.json
Vertex Shader
Fragment Shader
Position
Depth
Texcoords
diffuseColor
MetallicRoughness
Normal
deferred_lighting_chenyitong_homework
Color
Shadow Maps
Light Matrices
Lighting Shader
present_color
Present
shadow_mapping_chenyitong_homework
1024
resolution
Shader
12°C
大部晴朗
搜索
20:12
2025/4/12


Stage Viewer
FreeCamera Camera
sphere_light_0 SphereLight
rect_light_0 RectLight
mesh_0 Mesh
Edit Value
translate X
translate Y
translate Z
inputs:color
inputs:diffuse
inputs:exposure
inputs:intensity
inputs:radius
inputs:specular
Prim Info
Hydra Renderer
Free Camera Renderer



实验缺陷:

- 未能成功支持除了sphereLight以外的其他类型灯光的实现
- 某些角度的阴影部分有bug

一、 实验目的

本次实验旨在学习和实践 GPU 图形渲染管线中的核心技术，重点掌握：

1. 基于延迟着色 (Deferred Shading) 架构实现 Blinn-Phong 光照模型。
2. 理解并应用法线贴图 (Normal Mapping) 技术增强模型表面细节。
3. 实现阴影贴图 (Shadow Mapping) 技术，为场景添加实时阴影效果。
4. 熟悉 GLSL (OpenGL Shading Language) 编程，理解顶点着色器与片元着色器的作用。

二、 实现内容与方法

本次实验基于延迟着色管线完成。管线主要包含两个阶段：

1. **几何阶段 (Geometry Pass)**: 由 `rasterize_impl.fs` (片元着色器) 和对应的顶点着色器执行。渲染场景中的所有几何体，将其世界位置、深度、纹理坐标、反照率颜色、金属度/粗糙度以及应用了法线贴图后的世界空间法线等信息分别输出到 G-Buffer (一组屏幕大小的纹理) 的不同渲染目标 (Render Targets) 中。
2. **光照阶段 (Lighting Pass)**: 渲染一个覆盖全屏的四边形，在此阶段的片元着色器 (`blinn_phong.fs`) 中，从 G-Buffer 采样当前像素的几何信息，并结合光源信息计算最终的光照颜色。

- *[示意图：延迟着色流程图。包含两个主要阶段：1. 几何阶段（场景物体 -> 光栅化 (`rasterize_impl.fs` 等) -> G-Buffer纹理）；2. 光照阶段（全屏四边形 -> 读取 G-Buffer + 光源信息 -> 光照计算 (`blinn_phong.fs`) -> 最终颜色缓冲)]*

1. Blinn-Phong 着色模型实现 (于 `blinn_phong.fs`)

- **G-Buffer 采样**: 在 `blinn_phong.fs` 的 `main` 函数中，首先根据屏幕坐标 `gl_FragCoord.xy` 计算 UV，然后从 G-Buffer 对应的纹理中采样（这些纹理是由 `rasterize_impl.fs` 输出的）：
  - 世界坐标位置 (`fragPos`)：从 `position` 纹理采样。
  - 世界坐标法线 (`N`)：从 `normalMapSampler` 纹理采样（**注意：** 此纹理现在存储的是 `rasterize_impl.fs` 计算出的、已应用法线贴图的世界空间法线）。
  - 反照率颜色 (`albedo`)：从 `diffuseColorSampler` 纹理采样。
  - 粗糙度 (`roughness`)：从 `metallicRoughnessSampler` 纹理的 G 通道采样 (对应 `rasterize_impl.fs` 输出的 `metallicRoughness.y`)。
- **光照计算循环**: 遍历场景中的所有活动光源 (`light_count`)。
- **向量计算**: 计算光照所需的基本向量： `lightDir` , `viewDir` , `halfwayDir` 。
- **衰减计算 (Attenuation)**: 根据距离和光源参数计算光照衰减 `attenuation` 。
- **漫反射 (Diffuse)**: 使用 Lambertian 模型计算： `diffuse = lightColor * albedo * max(dot(N, lightDir), 0.0)` 。
- **高光 (Specular)**: 使用 Blinn-Phong 模型计算：

    - 根据 `roughness` 计算 `shininess`：`shininess = mix(256.0, 32.0, roughness * roughness)`。
    - 计算高光因子：`specFactor = pow(max(dot(N, halfwayDir), 0.0), shininess)`。
    - 计算高光颜色：`specular = lightColor * specFactor`。
- **环境光 (Ambient)**: 在循环外添加简单的环境光项：`totalLighting += ambientColor * albedo`。

2. 法线贴图 (Normal Mapping) (于 `rasterize_impl.fs`)

- **实现位置**: 法线贴图的应用逻辑在 *G-Buffer* 生成阶段的片元着色器 `rasterize_impl.fs` 中实现。
- ***G-Buffer* 输出**: 该着色器负责向 *G-Buffer* 的各个附件写入数据：
    - `layout(location = 0) out vec3 position`：输出世界坐标位置 `vertexPosition`。
    - `layout(location = 1) out float depth`：输出 NDC 深度 `clipPos.z / clipPos.w`。
    - `layout(location = 2) out vec2 texcoords`：输出纹理坐标 `vTexcoord`。
    - `layout(location = 3) out vec3 diffuseColor`：输出从 `diffuseColorSampler` 采样的颜色。
    - `layout(location = 4) out vec2 metallicRoughness`：输出从 `metallicRoughnessSampler` 采样的金属度(Z通道)/粗糙度(Y通道)值，并进行了 `.zy` swizzle。
    - `layout(location = 5) out vec3 normal`：输出**最终计算得到的世界空间法线**。
- **法线计算流程**:
    1. **采样法线图**: `vec3 normalmap_value = texture2D(normalMapSampler, vTexcoord).xyz;` 从法线贴图纹理中采样颜色值。
    2. **获取几何法线**: `vec3 normal = normalize(vertexNormal);` 获取并归一化传入的顶点法线（作为基础）。
    3. **计算 TBN 基向量**: 使用屏幕空间导数 `dFdx` 和 `dFdy` 动态计算切线 (Tangent) 和副切线 (Bitangent)：
        - `vec3 edge1 = dFdx(vertexPosition);`
        - `vec3 edge2 = dFdy(vertexPosition);`
        - `vec2 deltaUV1 = dFdx(vTexcoord);`
        - `vec2 deltaUV2 = dFdy(vTexcoord);`
        - `vec3 tangent = edge1 * deltaUV2.y - edge2 * deltaUV1.y;` (初始切线计算)
        - 包含了一个鲁棒性检查，并在切线接近零时尝试用另一种方式计算。
        - 使用 *Gram-Schmidt* 方法修正 `tangent`，使其与 `normal` 正交：`tangent = normalize(tangent - dot(tangent, normal) * normal);`
        - 计算 `bitangent`：`vec3 bitangent = normalize(cross(tangent,normal));`
    4. **转换切线空间法线**: 将从法线图采样到的颜色值 `normalmap_value` (范围 [0, 1]) 转换到切线空间的法线向量 `normalTS` (范围 [-1, 1])：`vec3 normalTS = normalize(normalmap_value * 2.0 - 1.0);`。
    5. **构建 TBN 矩阵**: 使用计算得到的 `tangent`, `bitangent`, 和几何 `normal` 构建从切线空间到世界空间的变换矩阵 `mat3` `TBN = mat3(tangent, bitangent, normal);`。
    6. **变换法线**: 将切线空间法线 `normalTS` 变换到世界空间：`vec3 modifiedNormal = TBN * normalTS;`。
    7. **输出最终法线**: 归一化 `modifiedNormal` 并将其作为最终的法线输出到 *G-Buffer* 的 `normal` 附件：`normal = normalize(modifiedNormal);`。
- *[示意图：切线空间示意图。显示一个表面点，上面有 N (几何法线)、T (切线)、B (副切线) 三个轴，以及从法线贴图采样的扰动法线 `normalTS` 如何相对于这个局部坐标系定义。TBN 矩阵将 `normalTS` 旋转到与世界坐标系对齐。]*
- **注意**: 在片元着色器中使用 `dFdx` / `dFdy` 计算 TBN 是一种常见技术，可以避免在顶点数据中存储或计算切线和副切线。但这种方法可能在 UV 映射不连续（如 UV 边界）的地方产生瑕疵。

3. 阴影贴图 (Shadow Mapping) (生成于 *Shadow Mapping* 节点，应用于 `blinn_phong.fs`)

- ***Shadow Map* 生成 (推断)**:
    - 为每个光源从其视角渲染场景深度到 `sampler2DArray` (`shadow_maps`) 的对应层。
- ***Shadow Map* 应用 (于 `blinn_phong.fs`)**:
    1. **坐标变换**: `fragPos` -> `fragPosLightSpace` (光源裁剪空间)。
    2. **透视除法**: `fragPosLightSpace` -> `projCoords` (NDC)。
    3. **边界检查**: 确保 `projCoords` 在 [-1, 1] 内。
    4. **UV 和深度转换**: NDC -> `shadowMapUV` ([0, 1]), NDC depth -> `currentDepth` ([0, 1])。
    5. **阴影偏移 (Bias)**: 计算动态 `bias` 防止自阴影。
    6. **深度采样**: 从 `shadow_maps` 对应层采样 `shadowMapDepth`。
    7. **深度比较**: `if (currentDepth > shadowMapDepth + bias)` 则 `shadow = 0.8`。
    8. **应用阴影**: `totalLighting += attenuation * (diffuse + specular) * (1.0 - shadow)`。

- *[示意图：Shadow Mapping 流程图。第一步：光源视角渲染，生成深度图层。第二步：相机视角渲染，在 `blinn_phong.fs` 中，片元位置变换到光源空间，采样对应层级的深度图比较深度，判断是否在阴影内，最终着色。]*

三、 实验结果

- **Blinn-Phong 光照**: 场景物体呈现了基于物理的漫反射和高光，光照强度随距离正确衰减。
- **法线贴图**: 通过 `rasterize_impl.fs` 应用法线贴图，模型表面（如砖墙、岩石）展现出丰富的凹凸细节，光照交互更真实。
- **阴影贴图**: 场景中物体能投射硬边阴影，增强了真实感和深度感。
- **可选任务**: PCSS 和 SSAO 写了部分代码， 但还存在部分bug。

四、 结论与讨论

本次实验成功地在延迟着色框架下实现了 Blinn-Phong 光照模型、法线贴图和阴影贴图技术。

- 通过 G-Buffer 生成 (`rasterize_impl.fs`) 和光照计算 (`blinn_phong.fs`) 的分离，实践了延迟着色的基本流程。
- 在 `rasterize_impl.fs` 中，具体实现了使用屏幕空间导数计算 TBN 矩阵并应用法线贴图的技术，将最终的世界空间法线存入 G-Buffer。
- 在 `blinn_phong.fs` 中，实现了从 G-Buffer 读取数据进行 Blinn-Phong 光照计算，并结合 Shadow Map 技术实现了多光源阴影效果。
- 实验加深了对 GLSL Shader 编程、渲染管线各阶段以及常用渲染技术的理解。