Homework 7 - Shadowing Mapping

陈曦 16340036

Basic:

1.实现方向光源的Shadowing Mapping:

- **要求场景中至少有一个object和一块平面(用于显示shadow)**
- **光源的投影方式任选其一即可**
- **在报告里结合代码，解释Shadowing Mapping算法**

Shadowing Mapping算法思路：

对光源的透视图所见的最近的深度值进行采样，并把深度值的结果储存到纹理中。最终，深度值就会显示从光源的透视图下见到的第一个片元。

深度映射由两个步骤组成：首先，我们渲染深度贴图，接下来像往常一样渲染场景，使用生成的深度贴图来计算片元是否在阴影之中。

- **深度贴图：** 首先，我们要为渲染的深度贴图创建一个帧缓冲对象，然后，创建一个2D纹理，提供给帧缓冲的深度缓冲使用，然后把生成的深度纹理作为帧缓冲的深度缓冲，进行渲染

```
    // Configure depth map FBO
    const unsigned  int SHADOW_WIDTH = 1024, SHADOW_HEIGHT = 1024;
    unsigned  int depthMapFBO;
    glGenFramebuffers(1, &depthMapFBO);
    // - Create depth texture
    unsigned  int depthMap;
    glGenTextures(1, &depthMap);
    glBindTexture(GL_TEXTURE_2D, depthMap);

    glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, SHADOW_WIDTH, SHADOW_HEIGHT, 0,
GL_DEPTH_COMPONENT, GL_FLOAT, NULL);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER);
    float borderColor[] = { 1.0f, 1.0f, 1.0f, 1.0f };
    glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, borderColor);

    glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
    glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, depthMap, 0);
    glDrawBuffer(GL_NONE);
    glReadBuffer(GL_NONE);
    glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

```
// 1. 首选渲染深度贴图
glViewport(0, 0, SHADOW_WIDTH, SHADOW_HEIGHT);
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
glClear(GL_DEPTH_BUFFER_BIT);
RenderScene(simpleDepthShader);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
// 2. 像往常一样渲染场景，但这次使用深度贴图
glViewport(0, 0, SCR_WIDTH, SCR_HEIGHT);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, depthMap);
RenderScene(shaderProgram);
```

- **渲染至深度贴图：**

```
//定点着色器
const char *vertexSimpleDepthShaderSource = "#version 330 core\n"
"layout(location = 0) in vec3 position;\n"

"uniform mat4 lightSpaceMatrix;\n"
"uniform mat4 model;\n"

"void main()\n"
"{\n"
"    gl_Position = lightSpaceMatrix * model * vec4(position, 1.0f);\n"
"}\0";

//片段着色器
const char *fragmentSimpleDepthShaderSource = "#version 330 core\n"

"void main()\n"
"{\n"
    // gl_FragDepth = gl_FragCoord.z;
"}\n\0";
```

- **渲染阴影：**

```
//顶点着色器
const char *vertexShaderSource = "#version 330 core\n"
"layout(location = 0) in vec3 position;\n"
"layout(location = 1) in vec3 normal;\n"
"layout(location = 2) in vec3 aColor;\n"

"out VS_OUT{\n"
"    vec3 FragPos;\n"
"    vec3 Normal;\n"
"    vec3 AColor;\n"
"    vec4 FragPosLightSpace;\n"
"} vs_out;\n"

"uniform mat4 projection;\n"
"uniform mat4 view;\n"
"uniform mat4 model;\n"
"uniform mat4 lightSpaceMatrix;\n"

"void main()\n"
"{\n"
"    gl_Position = projection * view * model * vec4(position, 1.0f);\n"
"    vs_out.FragPos = vec3(model * vec4(position, 1.0));\n"
```

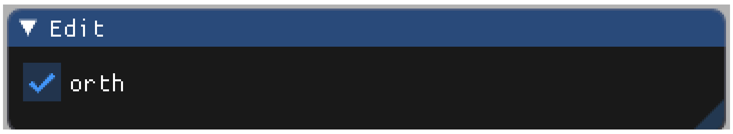
```
"   vs_out.Normal = transpose(inverse(mat3(model))) * normal;\n"
"   vs_out.AColor = aColor;\n"
"   vs_out.FragPosLightSpace = lightSpaceMatrix * vec4(vs_out.FragPos, 1.0);\n"
"}\0";


//片段着色器
const char *fragmentShaderSource = "#version 330 core\n"
"out vec4 FragColor;\n"

"in VS_OUT{\n"
"   vec3 FragPos;\n"
"   vec3 Normal;\n"
"   vec3 AColor;\n"
"   vec4 FragPosLightSpace;\n"
"} fs_in;\n"

"uniform sampler2D diffuseTexture;\n"
"uniform sampler2D shadowMap;\n"

"uniform vec3 lightPos;\n"
"uniform vec3 viewPos;\n"

"uniform bool shadows;\n"

"float ShadowCalculation(vec4 fragPosLightSpace)\n"
"{\n"
"   vec3 projCoords = fragPosLightSpace.xyz / fragPosLightSpace.w;\n"
"   projCoords = projCoords * 0.5 + 0.5;\n"
"   float closestDepth = texture(shadowMap, projCoords.xy).r;\n"
"   float currentDepth = projCoords.z;\n"
"   vec3 normal = normalize(fs_in.Normal);\n"
"   vec3 lightDir = normalize(lightPos - fs_in.FragPos);\n"
"   float bias = max(0.05 * (1.0f - dot(normal, lightDir)), 0.005f);\n"
    // PCF
"   float shadow = 0.0;\n"
"   vec2 texelSize = 1.0 / textureSize(shadowMap, 0);\n"
"   for (int x = -1; x <= 1; ++x)\n"
"   {\n"
"       for (int y = -1; y <= 1; ++y)\n"
"       {\n"
"           float pcfDepth = texture(shadowMap, projCoords.xy + vec2(x, y) * texelSize).r;\n"
"           shadow += currentDepth - bias > pcfDepth ? 1.0 : 0.0;\n"
"       }\n"
"   }\n"
"   shadow /= 9.0;\n"

"   if (projCoords.z > 1.0)\n"
"       shadow = 0.0;\n"

"   return shadow;\n"
"}\n"

"void main()\n"
"{\n"
"   vec3 color = fs_in.AColor;\n"
"   vec3 normal = normalize(fs_in.Normal);\n"
"   vec3 lightColor = vec3(0.4);\n"
"   vec3 ambient = 0.2 * color;\n"
"   vec3 lightDir = normalize(lightPos - fs_in.FragPos);\n"
```

```
"    float diff = max(dot(lightDir, normal), 0.0);\n"
"    vec3 diffuse = diff * lightColor;\n"
"    vec3 viewDir = normalize(viewPos - fs_in.FragPos);\n"
"    float spec = 0.0;\n"
"    vec3 halfwayDir = normalize(lightDir + viewDir);\n"
"    spec = pow(max(dot(normal, halfwayDir), 0.0), 64.0);\n"
"    vec3 specular = spec * lightColor;\n"
"    float shadow = shadows ? ShadowCalculation(fs_in.FragPosLightSpace) : 0.0;\n"
"    shadow = min(shadow, 0.75);\n"
"    vec3 lighting = (ambient + (1.0 - shadow) * (diffuse + specular)) * color;\n"

"    FragColor = vec4(lighting, 1.0f);\n"
"}\n\0";
```

2.修改GUI

```
        //创建ImGui
        ImGui_ImplOpenGL3_NewFrame();
        ImGui_ImplGlfw_NewFrame();
        NewFrame();

        Begin("Edit");
        Checkbox("orth", &orth);
        End();
```

菜单栏截图如下，初始界面默认选中正交投影，取消选中透视投影



Bonus:

1.实现光源在正交/透视两种投影下的Shadowing Mapping

```
    if (orth) {
        lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, near_plane, far_plane);
    }
    else {
        lightProjection = glm::perspective(45.0f, (float)SHADOW_WIDTH / (float)SHADOW_HEIGHT,
near_plane, far_plane);
    }
```

- **正交投影:**

- **透视投影：**

2.优化Shadowing Mapping (可结合References链接，或其他方法。优化方式越多越好，在报告里说明，有加分)

使用了PCF（percentage-closer filtering）来优化Shadowing Mapping，这是一种多个不同过滤方式的组合，产生柔和阴影，使它们出现更少的锯齿块和硬边。核心思想是从深度贴图中多次采样，每一次采样的纹理坐标都稍有不同。每个独立的样本可能在也可能不再阴影中。所有的次生结果接着结合在一起，进行平均化，就可以得到柔和阴影。一个简单的PCF的实现是简单的从纹理像素四周对深度贴图采样，然后把结果平均起来，textureSize返回一个给定采样器纹理的0级mipmap的vec2类型的宽和高。用1除以它返回一个单独纹理像素的大小，我们用以对纹理坐标进行偏移，确保每个新样本，来自不同的深度值。这里采样得到9个值，它们在投影坐标的x和y值的周围，为阴影阻挡进行测试，并最终通过样本的总数目将结果平均化。

```
    // PCF
"   float shadow = 0.0;\n"
"   vec2 texelSize = 1.0 / textureSize(shadowMap, 0);\n"
"   for (int x = -1; x <= 1; ++x)\n"
"   {\n"
"       for (int y = -1; y <= 1; ++y)\n"
"       {\n"
"           float pcfDepth = texture(shadowMap, projCoords.xy + vec2(x, y) * texelSize).r;\n"
"           shadow += currentDepth - bias > pcfDepth ? 1.0 : 0.0;\n"
"       }\n"
"   }\n"
"   shadow /= 9.0;\n"
```