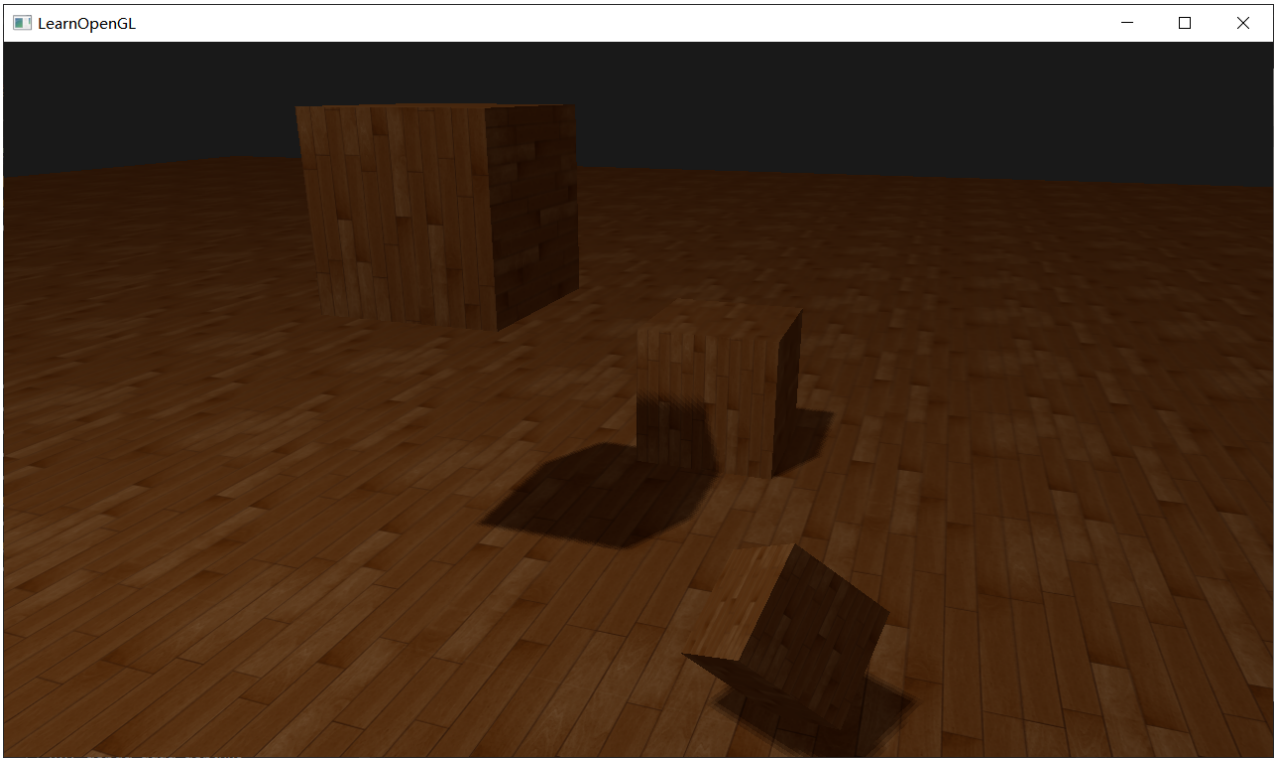
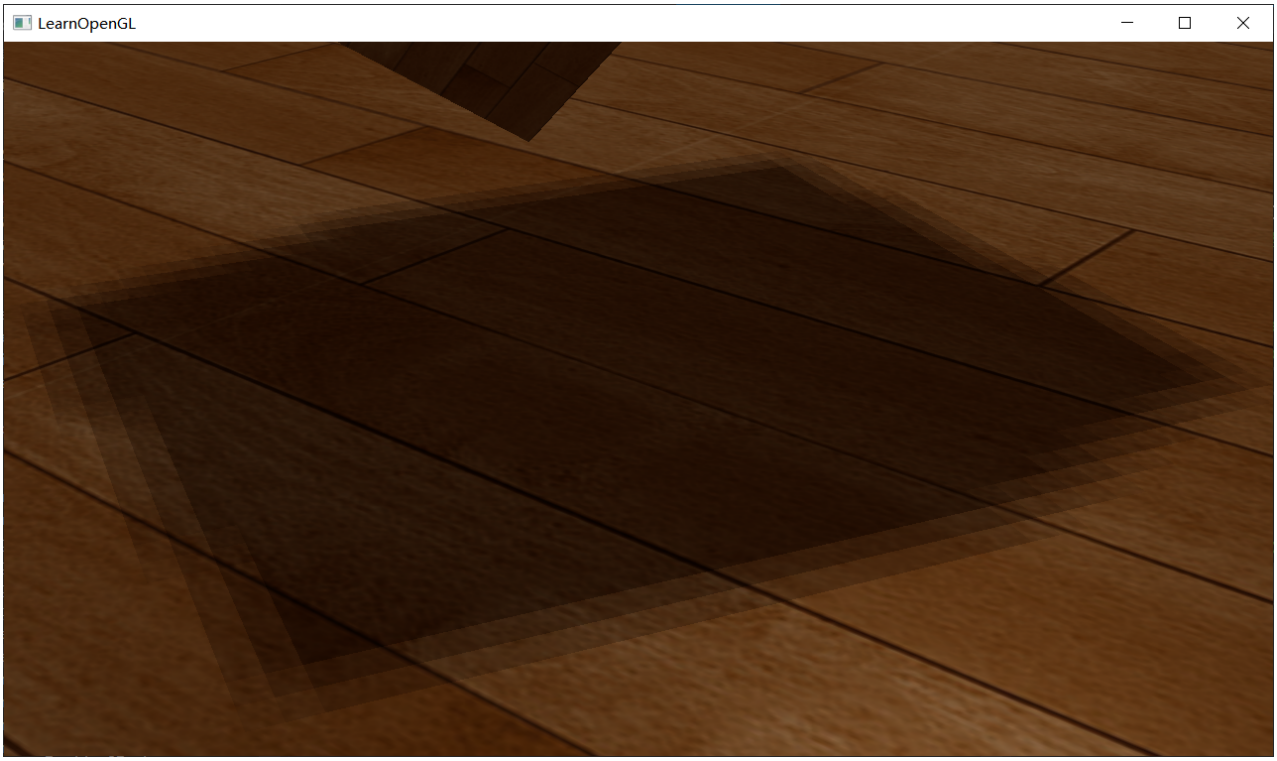
计算机图形学 作业七

一、实验结果

- Basic：
  - 实现 Shadow Mapping



- Bonus：
  - 参考 References 中的 PCF 方法来优化锯齿



二、代码实现

- Shadow Mapping 的基本思路是以光源为视角进行渲染，通俗一点说就是从光源的位置看，看不到的部分就是在阴影中，在 OpenGL 中实现的方法为先将物体坐标从世界空间转化到光源的观察空间，然后从光源的透视图来渲染场景，并将深度值的结果存在纹理中，这样纹理中的深度值就是从光源透视图下可见的第一个片元了，这个纹理可以称为深度贴图，关键代码如下：
- 生成纹理贴图：

```
// 生成深度贴图
glGenFramebuffers(1, &depthMapFBO); // 帧缓冲对象
glGenTextures(1, &depthMap);
glBindTexture(GL_TEXTURE_2D, depthMap); // 2D纹理
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[] = { 1.0, 1.0, 1.0, 1.0 };
glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, borderColor);
// 将生成的深度纹理作为帧缓冲的深度缓冲
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, depthMap, 0);
glDrawBuffer(GL_NONE); //
glReadBuffer(GL_NONE); // 显式地告诉OpenGL不需要使用颜色数据进行渲染
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

- 转换矩阵：

```
// 计算从世界空间坐标到光源处所见的空间的转换矩阵
glm::mat4 lightProjection, lightView;
glm::mat4 lightSpaceMatrix;
float near_plane = 1.0f, far_plane = 7.5f;
//lightProjection = glm::perspective(glm::radians(45.0f), (GLfloat)SHADOW_WIDTH /
(GLfloat)SHADOW_HEIGHT, near_plane, far_plane); // 透视投影
lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, near_plane, far_plane); // 正射投影
lightView = glm::lookAt(lightPos, glm::vec3(0.0f), glm::vec3(0.0, 1.0, 0.0));
lightSpaceMatrix = lightProjection * lightView;
simpleDepthShader.use();
simpleDepthShader.setMat4("lightSpaceMatrix", lightSpaceMatrix);
```

- 使用深度贴图渲染场景：

```
// 首先渲染深度贴图
glViewport(0, 0, SHADOW_WIDTH, SHADOW_HEIGHT);
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
glClear(GL_DEPTH_BUFFER_BIT);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, woodTexture);
renderScene(simpleDepthShader);
glBindFramebuffer(GL_FRAMEBUFFER, 0);

glViewport(0, 0, SCR_WIDTH, SCR_HEIGHT);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
shader.use();
glm::mat4 projection = glm::perspective(glm::radians(camera.Zoom), (float)SCR_WIDTH /
(float)SCR_HEIGHT, 0.1f, 100.0f);
glm::mat4 view = camera.GetViewMatrix();
shader.setMat4("projection", projection);
shader.setMat4("view", view);
shader.setVec3("viewPos", camera.Position);
shader.setVec3("lightPos", lightPos);
shader.setMat4("lightSpaceMatrix", lightSpaceMatrix);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, woodTexture);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, depthMap); // 使用深度贴图渲染场景
renderScene(shader);

debugDepthQuad.use();
debugDepthQuad.setFloat("near_plane", near_plane);
debugDepthQuad.setFloat("far_plane", far_plane);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, depthMap);
```

- Shadow Mapping 着色器：
  - 顶点着色器：

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoords;
```

```
out vec2 TexCoords;

out VS_OUT {
    vec3 FragPos;
    vec3 Normal;
    vec2 TexCoords;
    vec4 FragPosLightSpace;
} vs_out;

uniform mat4 projection;
uniform mat4 view;
uniform mat4 model;
uniform mat4 lightSpaceMatrix;

void main() {
    vs_out.FragPos = vec3(model * vec4(aPos, 1.0));
    vs_out.Normal = transpose(inverse(mat3(model))) * aNormal;
    vs_out.TexCoords = aTexCoords;
    vs_out.FragPosLightSpace = lightSpaceMatrix * vec4(vs_out.FragPos, 1.0);
    gl_Position = projection * view * model * vec4(aPos, 1.0);
}
```

- 片段着色器：

```
#version 330 core
out vec4 FragColor;

in VS_OUT {
    vec3 FragPos;
    vec3 Normal;
    vec2 TexCoords;
    vec4 FragPosLightSpace;
} fs_in;

uniform sampler2D diffuseTexture;
uniform sampler2D shadowMap;

uniform vec3 lightPos;
uniform vec3 viewPos;

float ShadowCalculation(vec4 fragPosLightSpace) {
    // perform perspective divide
    vec3 projCoords = fragPosLightSpace.xyz / fragPosLightSpace.w;
    // transform to [0,1] range
    projCoords = projCoords * 0.5 + 0.5;
    // get closest depth value from light's perspective (using [0,1] range fragPosLight as
coords)
    float closestDepth = texture(shadowMap, projCoords.xy).r;
    // get depth of current fragment from light's perspective
    float currentDepth = projCoords.z;
    // calculate bias (based on depth map resolution and slope)
    vec3 normal = normalize(fs_in.Normal);
    vec3 lightDir = normalize(lightPos - fs_in.FragPos);
```

```
    float bias = max(0.05 * (1.0 - dot(normal, lightDir)), 0.005);
    // check whether current frag pos is in shadow
    // float shadow = currentDepth - bias > closestDepth  ? 1.0 : 0.0;
    // PCF
    float shadow = 0.0;
    vec2 texelSize = 1.0 / textureSize(shadowMap, 0);
    for(int x = -1; x <= 1; ++x) {
        for(int y = -1; y <= 1; ++y) {
            float pcfDepth = texture(shadowMap, projCoords.xy + vec2(x, y) * texelSize).r;
            shadow += currentDepth - bias > pcfDepth  ? 1.0 : 0.0;
        }
    }
    shadow /= 9.0;

    // keep the shadow at 0.0 when outside the far_plane region of the light's frustum.
    if (projCoords.z > 1.0)
        shadow = 0.0;

    return shadow;
}

void main() {
    vec3 color = texture(diffuseTexture, fs_in.TexCoords).rgb;
    vec3 normal = normalize(fs_in.Normal);
    vec3 lightColor = vec3(0.3);
    // ambient
    vec3 ambient = 0.3 * color;
    // diffuse
    vec3 lightDir = normalize(lightPos - fs_in.FragPos);
    float diff = max(dot(lightDir, normal), 0.0);
    vec3 diffuse = diff * lightColor;
    // specular
    vec3 viewDir = normalize(viewPos - fs_in.FragPos);
    vec3 reflectDir = reflect(-lightDir, normal);
    float spec = 0.0;
    vec3 halfwayDir = normalize(lightDir + viewDir);
    spec = pow(max(dot(normal, halfwayDir), 0.0), 64.0);
    vec3 specular = spec * lightColor;
    // calculate shadow
    float shadow = ShadowCalculation(fs_in.FragPosLightSpace);
    vec3 lighting = (ambient + (1.0 - shadow) * (diffuse + specular)) * color;

    FragColor = vec4(lighting, 1.0);
}
```