科目	学号	姓名	邮箱
计算机图形学	16340294	张星	dukecheung@foxmail.com

Basic

题目一

实验要求

> 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

实验思路

渲染阴影，从摄像机的角度看，阴影就是遮挡的效果，从光源的角度看，阴影就是自己看不到的地方，所以我们可以使用之前的深度缓冲，将摄像机的视角转为光源的视角，然后使用*z-buffer*就可以获取深度信息了。题目要求的是**方向光源**，方向光源的位置为无穷远处，但是为了实现视角的转换，我们需要指定光源方向上的一个点，并将摄像机视角转换到光源视角。

这里我使用了正交投影，首先声明几个矩阵：

```
GLfloat nearPlane = 1.0f, farPlane = 7.5f;
glm::mat4 lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, nearPlane, farPlane);
glm::mat4 lightView = glm::lookAt(lightPos, glm::vec3(0.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glm::mat4 lightSpaceMatrix = lightProjection * lightView;
```

lightView 位于光源方向上，看向场景原点，指定up向量，然后乘投影矩阵，将这二者的的乘积作为参数传入vertexShader，也可以分别传入，作为view和projection矩阵，再乘model矩阵，即可将坐标空间转换到光源的视角内了。

```
const char *depthVertexShader = "#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";
```

为了使用深度信息，需要生成一张**深度贴图**：首先生成一个帧缓冲对象，然后创建纹理，这里需要用到之前的纹理部分的知识：

```
GLuint depthMapFBO;
glGenFramebuffers(1, &depthMapFBO);
const GLuint SHADOW_WIDTH = 1024, SHADOW_HEIGHT = 1024;

GLuint 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);
```

这里的函数很多，我仔细复习了之前的教程，做以简单说明：

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
```

这个函数主要是为了解决观察者离纹理距离不同而产生的种种问题，对纹理进行**过滤**，例如上述代码，第一个参数表示为2D纹理，第二个参数说明是缩小时的操作，第三个参数作用是该纹理点取最近的纹理像素作为它的样子。

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
```

第一个参数同上，第二个参数涉及上课所讲知识，纹理的坐标为S,T，wrap则是渲染方式，第三个参数的意思是超出坐标范围，应用我们所指定的像素颜色。边缘颜色与设置如下：

```
float borderColor[] = { 1.0f, 1.0f, 1.0f, 1.0f };
glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, borderColor);
```

为了使效果更加明显，我使用了教程中所给的图片资源：

```
unsigned int texture = loadTexture("wood.png");
glBindTexture(GL_TEXTURE_2D, texture);
```

但是如何将这些贴图贴到正方体上，需要更多的坐标属性：纹理坐标，即s，t，以接收者平面坐标为例：

```
float planeVertices[] = {
    // positions            // normals         // texcoords
    25.0f, -0.5f,  25.0f,  0.0f, 1.0f, 0.0f,  25.0f,  0.0f,
    -25.0f, -0.5f,  25.0f,  0.0f, 1.0f, 0.0f,   0.0f,  0.0f,
    -25.0f, -0.5f, -25.0f,  0.0f, 1.0f, 0.0f,   0.0f, 25.0f,

    25.0f, -0.5f,  25.0f,  0.0f, 1.0f, 0.0f,  25.0f,  0.0f,
    -25.0f, -0.5f, -25.0f,  0.0f, 1.0f, 0.0f,   0.0f, 25.0f,
    25.0f, -0.5f, -25.0f,  0.0f, 1.0f, 0.0f,  25.0f, 25.0f
};
```

为了实现这一目的，需要一个额外的**Shader**，顶点着色器处理纹理坐标，然后片段着色器贴到我们需要渲染的正方体上。顶点着色器如下，它需要将纹理坐标传递给片段着色器：

```
const char *quadVertexShader = "#version 330 core\n"
"layout (location = 0) in vec3 aPos;\n"
"layout(location = 1) in vec2 aTexCoords;\n"
"out vec2 TexCoords;\n"
"void main()\n"
"{\n"
"   TexCoords = aTexCoords;\n"
"   gl_Position = vec4(aPos, 1.0);\n"
"}\0";
```

然后片段着色器结合坐标和之前的纹理贴图，渲染片段颜色：

```
const char *quadFragmentShader = "#version 330 core\n"
"out vec4 FragColor;\n"
"in vec2 TexCoords;\n"
"uniform sampler2D depthMap;\n"
"void main()\n"
"{\n"
"   float depthValue = texture(depthMap, TexCoords).r;\n"
"   FragColor = vec4(vec3(depthValue), 1.0);\n"
"}\n\0";
```

最后进行阴影渲染：在这里需要进行计算，若一个片段在阴影内，则只渲染环境光，否则都要，顶点着色器与之前的并无太大差别，关键在于片段着色器的计算部分：

```
"   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"
"   float shadow = currentDepth > closestDepth  ? 1.0 : 0.0;"
```

需要将片段的坐标转为深度空间的[0,1]，然后计算出当前的深度值，与离光最近的深度值作比较，大于，则不在阴影中，否则处于阴影。

实验结果

见result.gif。

题目二

实验要求

> 修改GUI

实验思路

本次试验中并无太多可以修改的地方，所以我只设置了切换正交投影和透视投影的Checkbox。同时为了能够点击选择框，取消了鼠标的作用，不能够左右上下调整角度。

```
ImGui_ImplOpenGL3_NewFrame();
ImGui_ImplGlfw_NewFrame();
ImGui::NewFrame();
ImGui::Begin("Shadow Mapping", &ImGui, ImGuiWindowFlags_MenuBar);
ImGui::Checkbox("Pers_Orth", &Pers_Orth);
ImGui::End();

if (Pers_Orth) {
    lightProjection = glm::perspective(glm::radians(camera.getZoom()), (float)width /
(float)height, nearPlane, farPlane);
}
else {
    lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, nearPlane, farPlane);
}
```

实验结果

见result.gif。

Bonus

题目一

实验题目

> 实现光源在正交/透视两种投影下的Shadow Mapping

实验思路

为了体现出更加直接的效果，我将三个立方体摆成一排，光源处于左上方。

与教程不同的是，我仅通过调整了投影矩阵就实现了透视投影，而调整shader中的代码没有效果。投影矩阵如下：

```
glm::mat4 lightProjection = glm::perspective(glm::radians(camera.getZoom()), (float)width /
(float)height, nearPlane, farPlane);
```

正交投影矩阵如下：

```
glm::mat4 lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, nearPlane, farPlane);
glm::mat4 lightView = glm::lookAt(lightPos, glm::vec3(0.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glm::mat4 lightSpaceMatrix = lightProjection * lightView;
```

两者除了投影矩阵，其余部分均相同。

实验结果

题目二

实验题目

> 优化Shadow Mapping

实验思路

- 阴影失真：为了解决渲染出交替线的问题，这是因为对阴影部分判断失误所导致的，解决办法是将所有的深度值添加偏移量，这样就可以进一步精确对阴影的判断，从而避免产生这种问题。

```
"   float bias = max(0.05f * (1.0f - dot(normal, lightDir)), 0.005f);\n"
```

- PCF：解决了放大时的锯齿现象，这与之前所提到的解析度有关系，不可避免，但可以适度降低。可以通过多次采样，然后取均值的方法来降低锯齿现象，更为平滑。类似于数字图像处理中的均值滤波器。

```
"   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"
```

实验结果

见result.gif。