Computer Graphics Homework 7

Basic

实现方向光源的Shadowing Mapping

cubes

> function : `renderCube()`

和之前的作业中一样，将顶点数组（每个面分割为两个三角形）与着色器绑定。这里根据VAO的ID判读那是否生成了顶点数组对象，如果没有则根据顶点数组生成对应的VAO和VBO，并且解析指针；如果已经初始化完毕，则直接绑定cubeVAO，并根据36个顶点绘制cube表面的12个三角形。

plane

平面同样先生成planeVAO和planeVBO并绑定缓存，解析指针，渲染时绑定，并绘制表示平面的两个三角形。

```
    glGenVertexArrays(1, &planeVAO);
    glGenBuffers(1, &planeVBO);
    glBindVertexArray(planeVAO);
    glBindBuffer(GL_ARRAY_BUFFER, planeVBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(planeVertices), &planeVertices,
GL_STATIC_DRAW);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid*)0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid*)(3 *
sizeof(GLfloat)));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid*)(6 *
sizeof(GLfloat)));
    glBindVertexArray(0);
```

场景渲染根据上面初始化的plane和cube通过改变model变换矩阵生成不同位置的plane和cube.

```
// Plane
glm::mat4 model = glm::mat4(1.0f);
glUniformMatrix4fv(glGetUniformLocation(shader.getShaderProgram(), "model"), 1,
GL_FALSE, glm::value_ptr(model));
glBindVertexArray(planeVAO);
glDrawArrays(GL_TRIANGLES, 0, 6);
glBindVertexArray(0);
// Cubes
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.0f, 1.5f, 0.0));
glUniformMatrix4fv(glGetUniformLocation(shader.getShaderProgram(), "model"), 1,
GL_FALSE, glm::value_ptr(model));
```

深度贴图

深度贴图主要用于计算阴影并将渲染结果存入纹理

1. 创建帧缓冲并进行2D纹理载入

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

2. 根据在**光的透视图**下的场景判断深度

> 注意这里深度贴图只需要存储场景的深度，即距离光源的远近，而不需要颜色缓冲
>
> ```
> glDrawBuffer(GL_NONE);
> glReadBuffer(GL_NONE);
> ```

3. 生成深度贴图
    1. 首先渲染深度贴图
    2. 之后使用深度贴图渲染场景

> 注意这里的引用需要改变视口参数来适应阴影贴图的尺寸

光源空间判断

在上文中加粗显示，对于深度的判断需要在光的透视图下进行计算。

这里的lightProjection为光源的投影方式（正交或者透视），lightView为从光源位置看向场景中央的视角。

于是光源变换矩阵这样形成。

渲染深度贴图

将物体放在光源视角下，即在其世界坐标系乘以$lightView * lightProjection$获得在光透视坐标系中的相对位置。并将这个结果矩阵传入顶点着色器文件进行变换。

将渲染好的深度贴图贴到四边形（cube和plane的表面）上，之后需要加入纹理进行贴图。

shadowing mapping算法

这个算法的实现在shadow_shader.fs中，首先要在顶点着色器中进行变换，一方面是从视角上的变换，另一方面是在光的透视空间的变换。将顶点着色器转换为世界坐标系的结果和顶点在光空间中的坐标位置。

面片着色器根据上次介绍的光照模型，并添加shadow判断变量表示该像素是在阴影内还是阴影外，如果是阴影部分，就只渲染环境光；如果不在阴影部分，就加上漫反射和镜面反射的渲染。

光照的公式如下：

$$lighting = (ambient + (1.0 - shadow) * (diffuse + specular)) * color$$

检查像素是否在阴影中

根据ppt中shadow mapping的方法，使用透视除法，将裁剪空间的坐标标准化，范围[-1,1]。之后，由于深度贴图的坐标范围为[0, 1]，因此通过如下公式进行映射

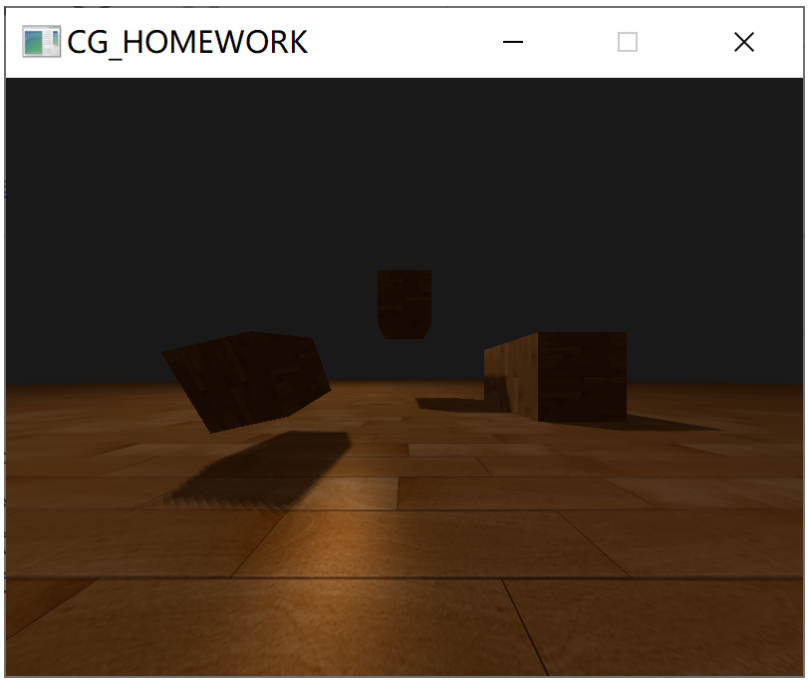
$$projCoords = projCoords * 0.5 + 0.5$$

之后从深度贴图中采样，得到光透视空间的最近深度，并将片元的当前深度与之比较，如果最近深度更近，则该点在阴影中；否则，该点不再阴影中。

使用带有纹理的贴图将木头的贴图载入。

> 这一步中注意需要加入一个cpp才能编译通过，可以参见纹理的章节

运行结果



Bonus

切换两种光的透视

通过int变量presp控制。

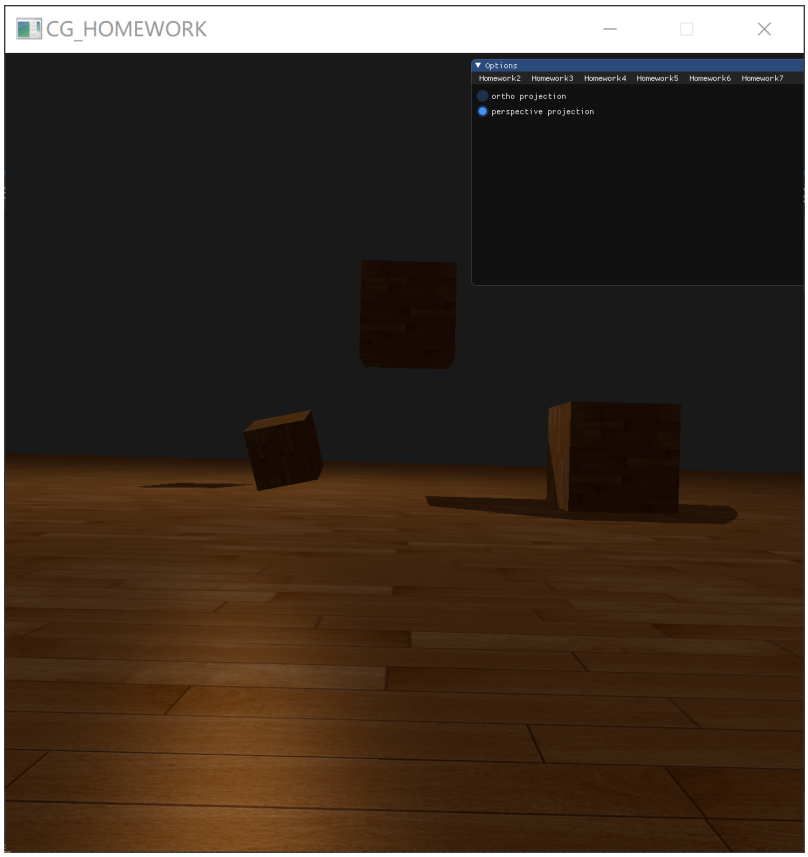
光源的显示控制：

```
if (persp == 1) {
    lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, near_plane, far_plane);
}
else {
    lightProjection = glm::perspective(45.0f, (float)windowWidth / (float)windowHeight, 0.1f, 100.0f);
}
```

菜单栏的控制：

```
void Homework7::imGuiMenuSetting() {
    if (ImGui::BeginMenu("Homework7")) {
        ImGui::MenuItem("Basic", NULL, &basic);
        ImGui::EndMenu();
    }
}
void Homework7::imGuiSetting() {
    if (basic) {
        // 选择正交和透视
        ImGui::RadioButton("ortho projection", &persp, 1);
        ImGui::RadioButton("perspective projection", &persp, 0);
    }
}
```

透视投影的运行结果

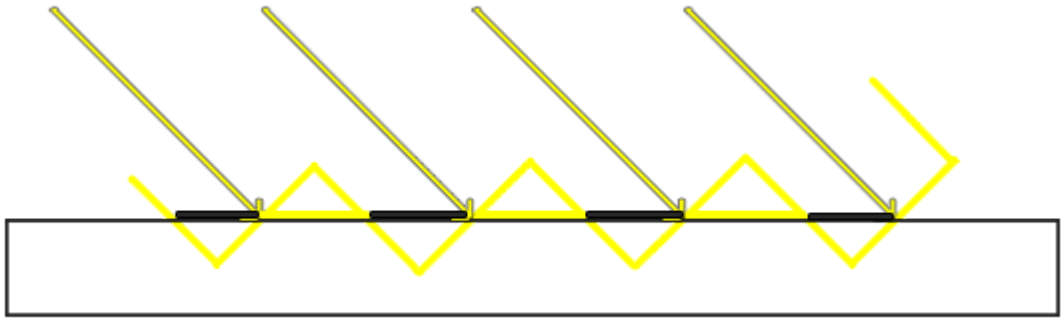


改进阴影贴图

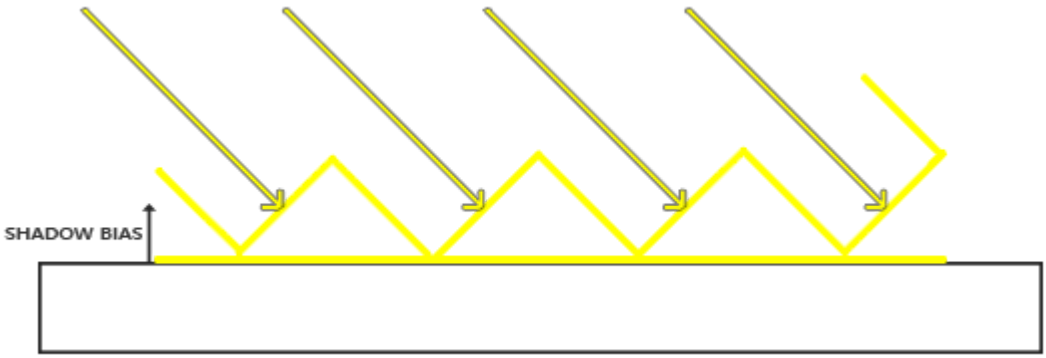
缝隙

阴影偏移，使得不会有片元被误认为在表面之前：

阴影偏移之前



阴影偏移之后



不直接将当前片元的深度和最近的深度比较，而将当前片元深度-0.005的偏移，这样与最近深度表面接触的那层像素不会产生断断续续的情况，代码实现：

```
float bias = 0.005;
float shadow = currentDepth - bias > closestDepth  ? 1.0 : 0.0;
```

但有时0.005的偏移仍然不能解决有阴影条纹状的问题，并且根据表面的法方向和光照方向。如果夹角偏大，则点乘结果偏小，$0.05 * (1.0 - dot(normal, lightDir))$偏大，因此这时的偏移量会相对大一些；但为了夹角过小的时候偏移量为0，使用max函数纠正：

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

悬浮

但是使用偏移会导致悬浮 - 另一个失真的问题，因为在判断的时候实际上对于物体表面进行了位移。

解决方法，正面剔除。

消除锯齿

通过从深度贴图中多次采样，产生柔和阴影，将多次结果平均。代码中采用从像素的八邻域中采样，类似于做一个均值模板的3*3模板的卷积。

代码中的实现：

```
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;
```

效果