计算机图形学作业（六）：画一个平面和一些立方体并实现阴影映射

引入依赖库

本次实验需要用到 OpenGL 官方的 Camera 库和 Shader 库。Camera 库内包含一个摄像机类，可以帮助我们快速地生成和使用摄像机，点击此处下载。Shader 库是内包含一个 Shader 类，可以帮助我们快速地声明和使用顶点着色器和片段着色器，我们只需新建顶点着色器和片段着色器代码文件，就可以在主函数中利用 Shader 的构造函数构造出着色器，并使用其中的函数控制着色器，点击此处下载

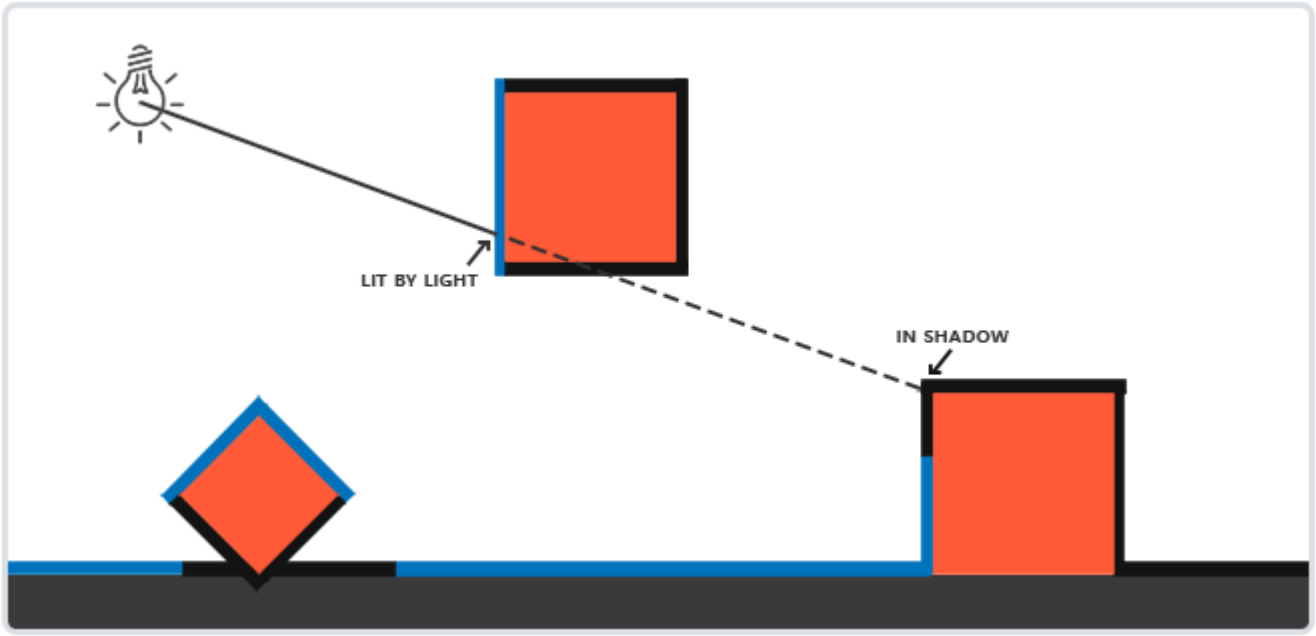
除此之外，本次实验需要用到加载纹理的第三方库`stb_image.h`，点击此处下载。下载之后，在项目中引入文件头文件`stb_image.h`，并新建一个`.cpp`文件，里面内容为：

```
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"
```

之后，我们便可以使用`stb_image.h`内置的函数加载纹理了。

阴影映射

原理



在深度测试中，在深度缓冲里的一个值是摄像机视角下，对应于一个片元的一个0到1之间的深度值。我们从光源的透视图来渲染场景，并把深度值的结果储存到纹理中，这样就能对光源的透视图所见的最近的深度值进行采样。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了。我们管储存在纹理中的所有这些深度值，叫做深度贴图或阴影贴图。

利用每个片元的深度值，我们可比较在同一条光线上的点谁距离光源更近，距离最近的点是可视的被光线照射的点，而距离较远的则被视为阴影。

深度贴图

第一步我们需要生成一张深度贴图。深度贴图是从光的透视图里渲染的深度纹理，用它计算阴影。因为我们需要将场景的渲染结果储存到一个纹理中，我们将再次需要帧缓冲。

首先，我们要为渲染的深度贴图创建一个帧缓冲对象：

```
GLuint depthMapFBO;
glGenFramebuffers(1, &depthMapFBO);
```

然后，创建一个2D纹理，提供给帧缓冲的深度缓冲使用：

```
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_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

把我们把生成的深度纹理作为帧缓冲的深度缓冲：

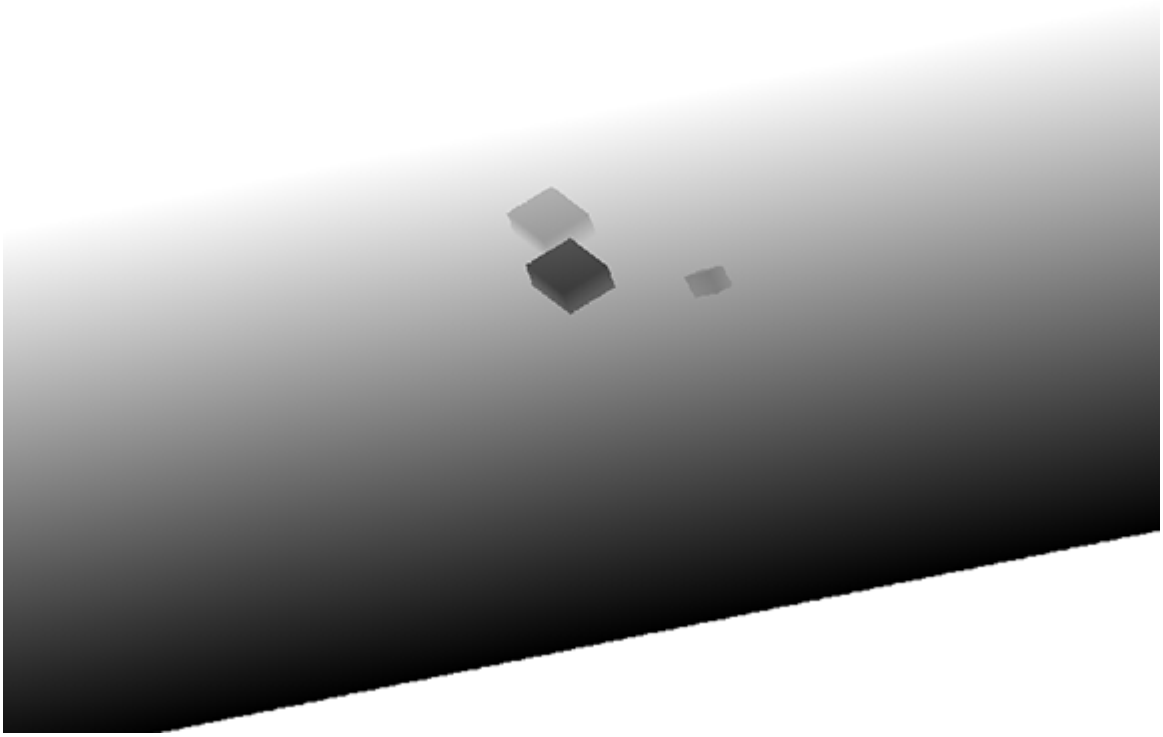
```
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D,
depthMap, 0);
glDrawBuffer(GL_NONE);
glReadBuffer(GL_NONE);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

合理配置将深度值渲染到纹理的帧缓冲后，我们就可以开始生成深度贴图：

```
// 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);
    ConfigureShaderAndMatrices();
```

```
    RenderScene();
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);
ConfigureShaderAndMatrices();
glBindTexture(GL_TEXTURE_2D, depthMap);
RenderScene();
```

最终效果如下：



渲染阴影

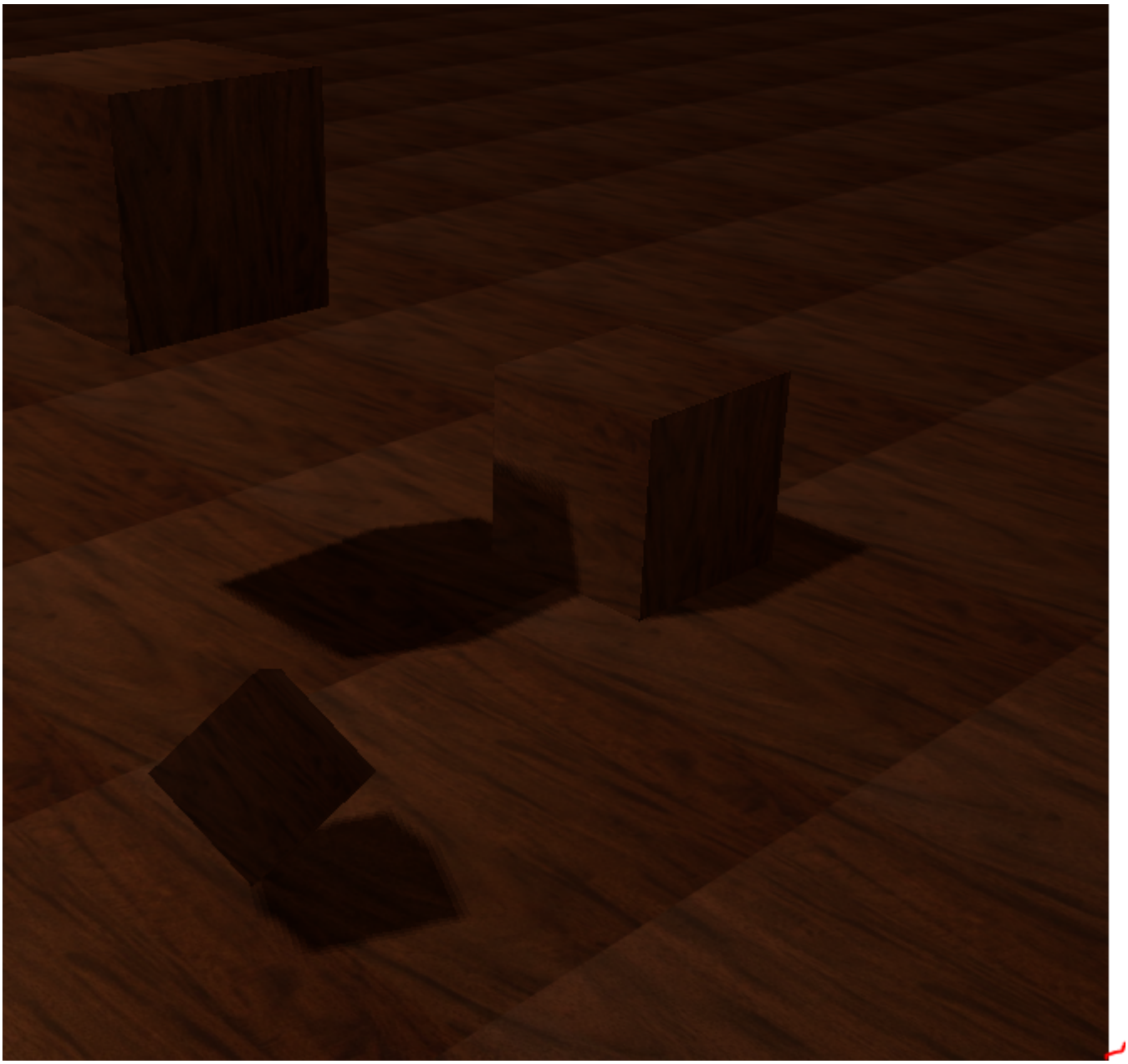
渲染阴影主要在像素着色器中进行，在片段着色器中检验一个片元是否在阴影之中，然后在顶点着色器中进行光空间的变换。着色器的代码在后面给出。

然后，在主函数中，设置着色器的各种属性：

```
shader.use();
glm::mat4 projection = glm::perspective(glm::radians(camera.Zoom),
(float)WINDOW_WIDTH / (float)WINDOW_HEIGHT, 0.1f, 100.0f);
glm::mat4 view = camera.GetViewMatrix();
shader.setMat4("projection", projection);
shader.setMat4("view", view);
// set light uniforms
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);
renderScene(shader);
```

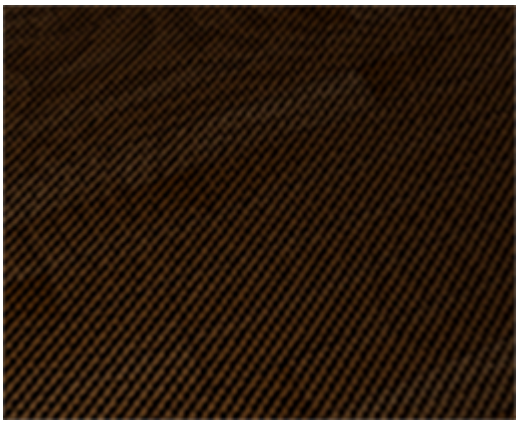
最终的效果如下：



阴影优化（加分项）

阴影失真

运行程序后，放大观察到有明显的线条样式：



为了解决这个问题，我们需要在片段着色器重稍微修改：

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

使用了偏移量后，所有采样点都获得了比表面深度更小的深度值，这样整个表面就正确地被照亮，没有任何阴影。

PCF

运行程序后，放大观察到锯齿现象：

我们要在片段着色器中稍微修改以下部分：

```
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;
```

以上解决方案叫做PCF，这是一种多个不同过滤方式的组合，它产生柔和阴影，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，我们就得到了柔和阴影。