**计算机图形学Homework7**

15331416 赵寒旭

**目录**

[**1. 运行结果** 2](#_Toc513075358)

[1）初始界面 2](#_Toc513075359)

[2）在立方体表面显示阴影 2](#_Toc513075360)

[3）在平面上显示阴影 3](#_Toc513075361)

[**2. 实现思路** 3](#_Toc513075362)

[**2.1 阴影渲染** 3](#_Toc513075363)

[1）光源空间变换 4](#_Toc513075364)

[2）创建深度贴图 4](#_Toc513075365)

[3）以光源视角渲染场景深度信息 5](#_Toc513075366)

[4）生成阴影 6](#_Toc513075367)

[**2.2 阴影优化** 7](#_Toc513075368)

[1）阴影偏移 7](#_Toc513075369)

[2）采样过多 8](#_Toc513075370)

[3）PCF 8](#_Toc513075371)

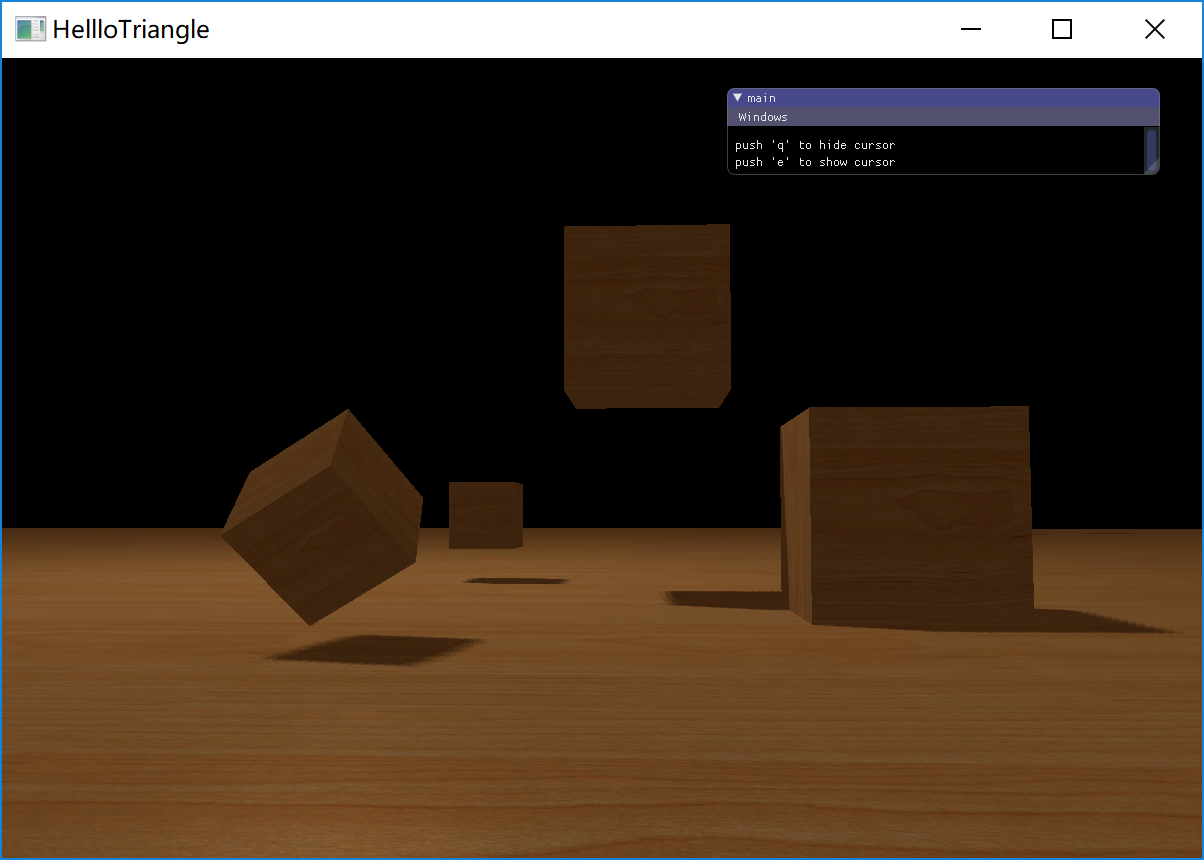
**1. 运行结果**

光源投影方式：正交投影

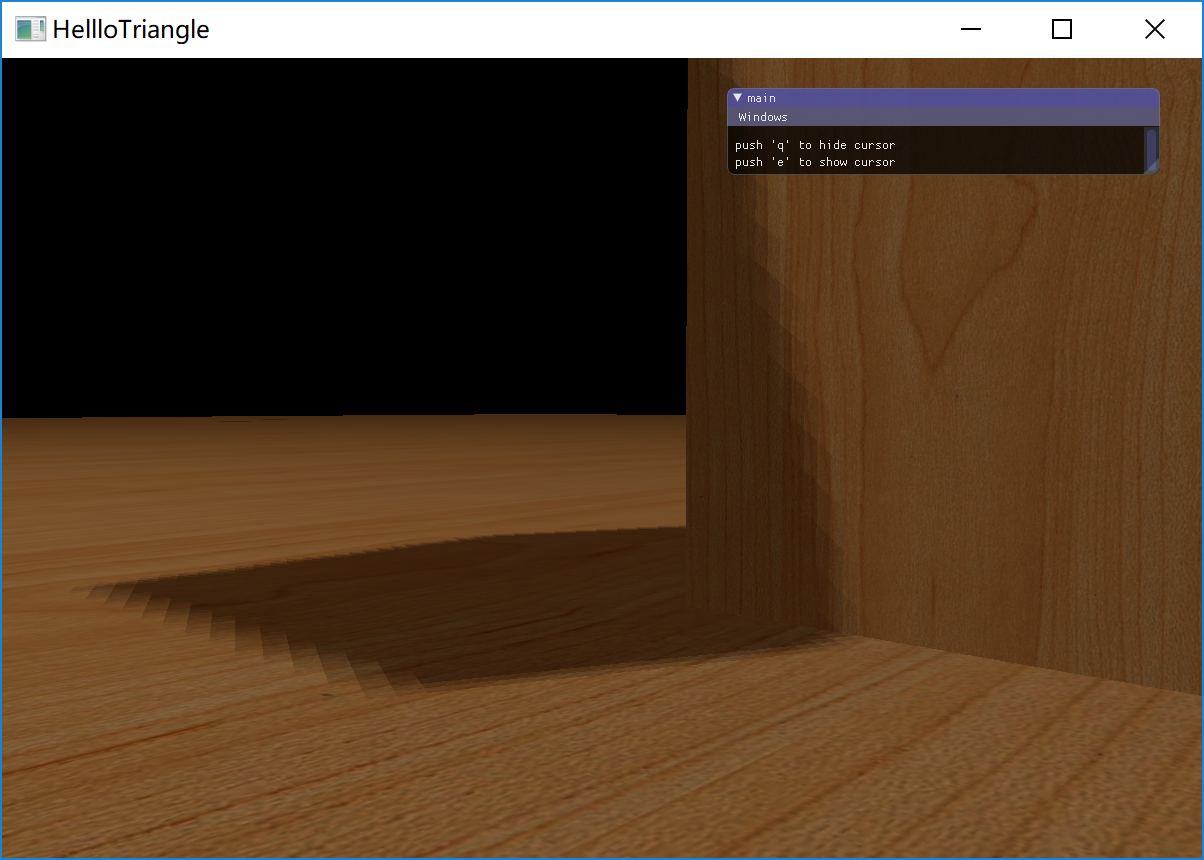
场景：4个立方体，一块平面

交互方式：WASD控制前左后右移动，鼠标控制视角，Q键隐藏鼠标指针，E键显示鼠标指针。

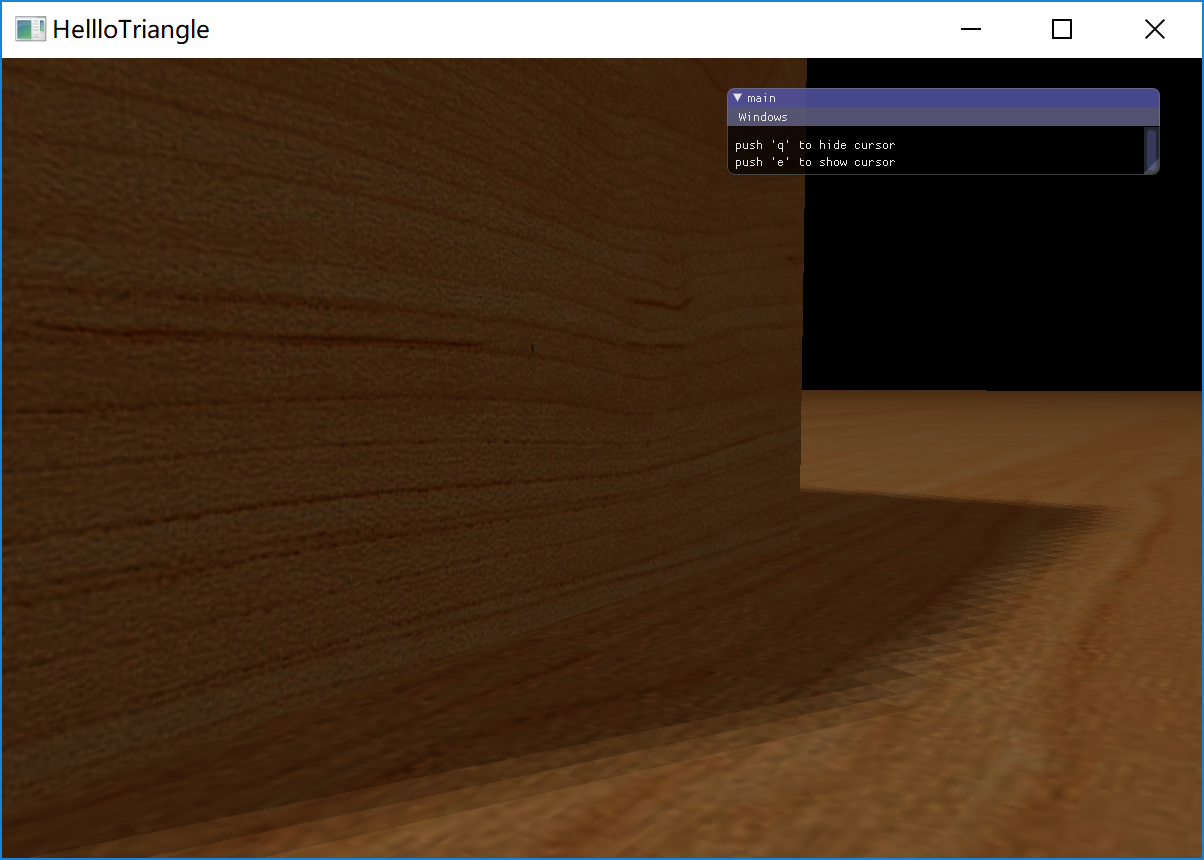
**1）初始界面**

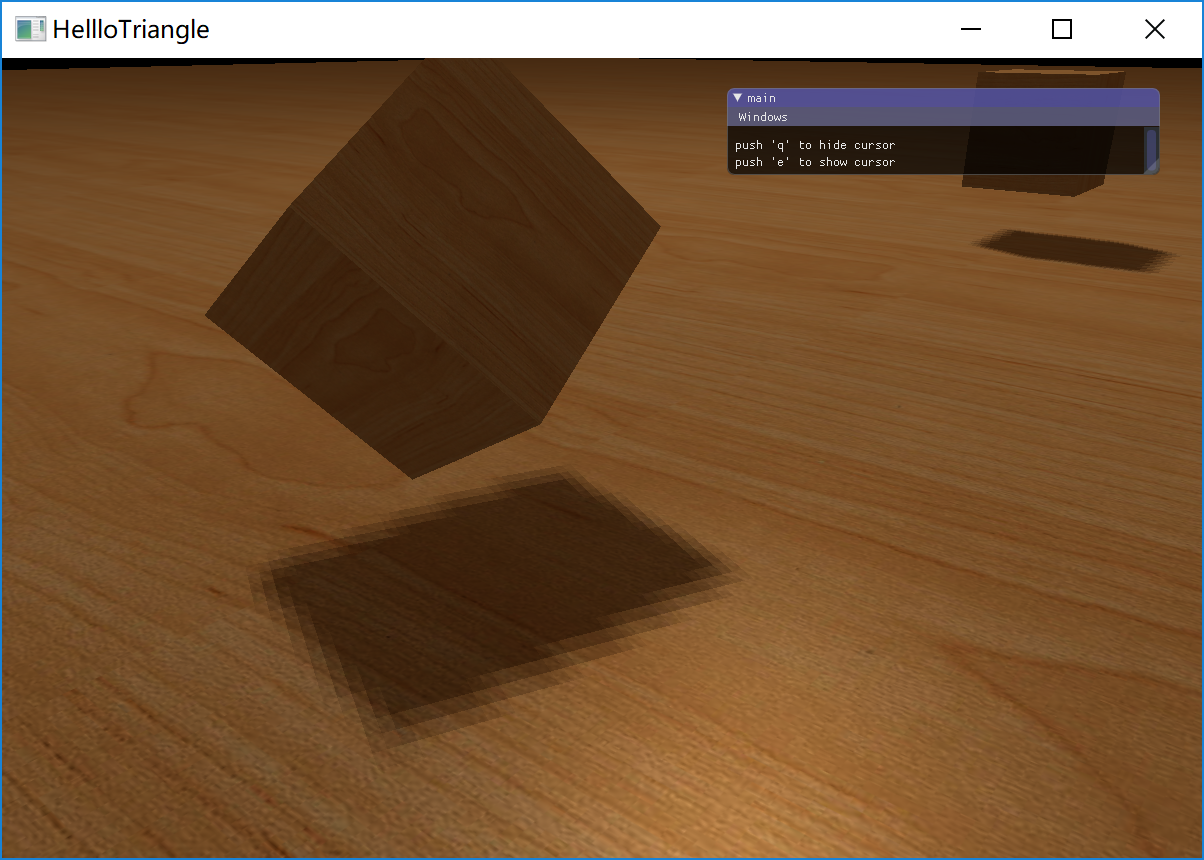


**2）在立方体表面显示阴影**



**3）在平面上显示阴影**





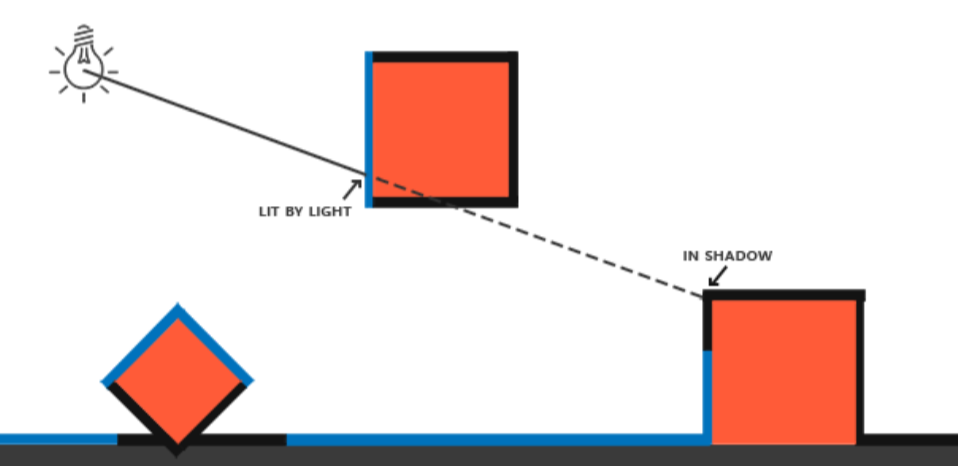
另有视频在doc文件夹下。

**2. 实现思路**

**2.1 阴影渲染**

本节重点：结合代码解释Shadowing Mapping算法。

首先简单描述以下阴影渲染的总体思路和基本原理。



阴影映射即以光的位置为视角进行渲染，所有能看到的表面都被点亮，而看不到的地方认为是被阴影覆盖。如上图所示，标记为蓝色的部分是被光线直射的地方fragment，黑色部分表示应被渲染为带阴影的fragment。

对上图射线来说，射线第一次击中物体的点作为**最近点**，如果射线上某一点点比此点距起点更远，这个点就在阴影中。

为了避免性能的消耗，我们借助深度缓冲得到摄像机视角下目标场景的深度值。

我们从光源的透视图来渲染场景，将得到的深度值结果储存到纹理中，对光源透视图所见的最近深度值进行采样，最终使深度值显示从光源的透视图下见到的第一个片元。

即我们得到了从光源出发所有最近点的深度值，并把它们统称为**深度贴图（depth map）**。

具体的阴影渲染过程，可以按照以下两个基本步骤来进行。

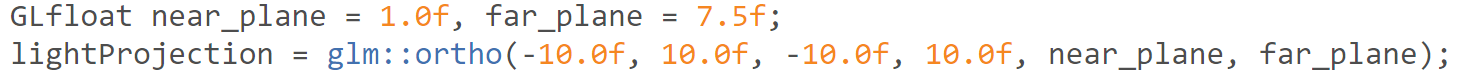
① 以光源视角渲染场景，得到深度图（DepthMap），并存储为texture。

② 以camera视角渲染场景，使用Shadowing Mapping算法（比较当前深度值与在DepthMap Texture的深度值），决定某个点是否在阴影下。

**1）光源空间变换**

我们使用一个投影矩阵和一个观察矩阵组合而成的T变换将世界坐标转化到光源的可见坐标空间。

使用正交投影矩阵lightProjection。



创建观察矩阵lightView来变换每个物体，将其转移到光源视角下，使从光源位置看向场景中央。



结合投影矩阵和观察矩阵，我们得到了光空间的变换矩阵lightSpaceMatrix，作用是将场景中的世界坐标转换为光源空间下的坐标，**此时每个fragment坐标的z分量即对应它在此光源下的深度**。



**2）创建深度贴图**

我们要将光源视角下的渲染结果存储到深度贴图中，首先需要创建一张深度贴图。

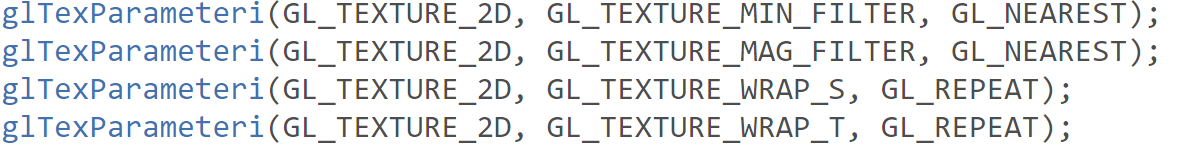
（1）创建一个帧缓冲对象



（2）创建一个2D纹理

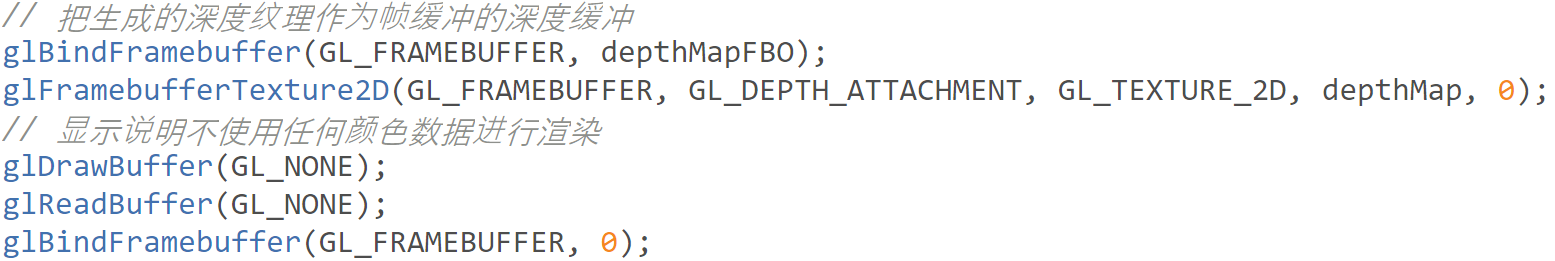
因为**只需要考虑深度值**，指定纹理格式为GL\_DEPTH\_COMPONENT。同时设置深度贴图的解析度即纹理的宽高分别为1200和800。





（3）把生成的深度纹理作为帧缓冲的深度缓冲

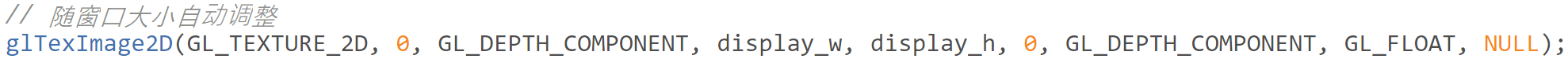
深度贴图仅需要深度值，显式告知OpenGL不使用任何颜色数据进行渲染。

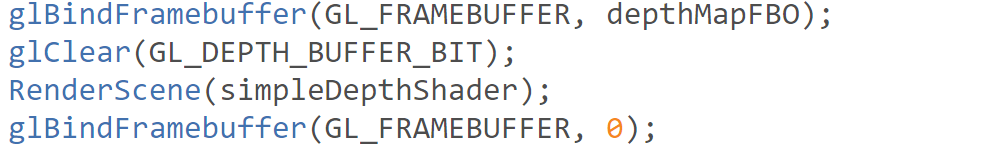


**3）以光源视角渲染场景深度信息**

在渲染循环中：

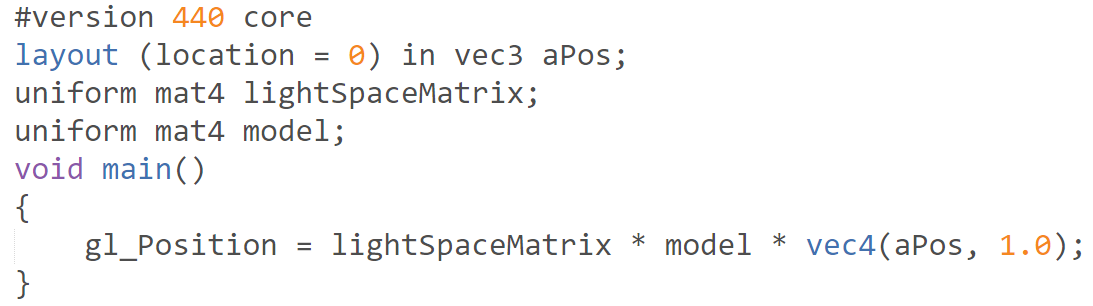






（1）顶点着色器（顶点变换到光空间着色器）

将顶点通过空间变换矩阵lightSpaceMatrix变换到光空间中。



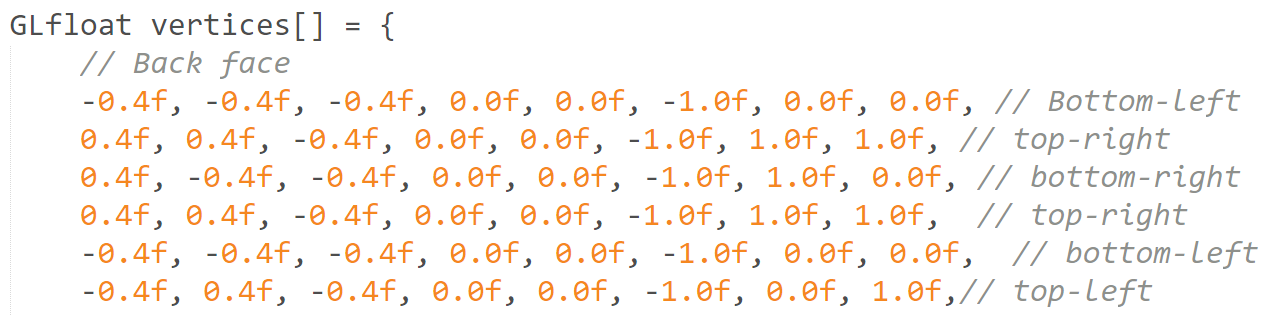
（2）片段着色器（（顶点变换到光空间着色器）

只需要深度缓冲，不需要考虑fragment颜色，着色器可以直接置为空。

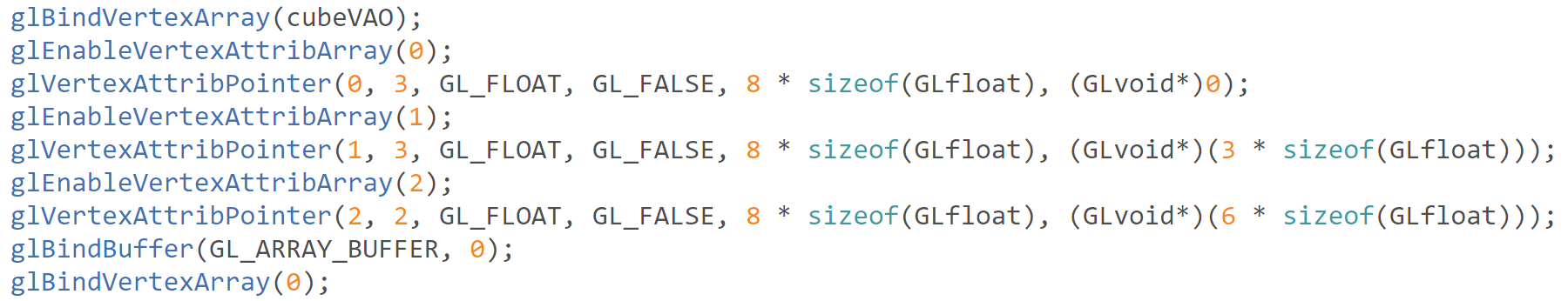


（3）RenderCube函数

初始化顶点坐标数组：以一面为例，分别是六个顶点（正方形由两个三角形组成）的坐标，法向量，纹理坐标。

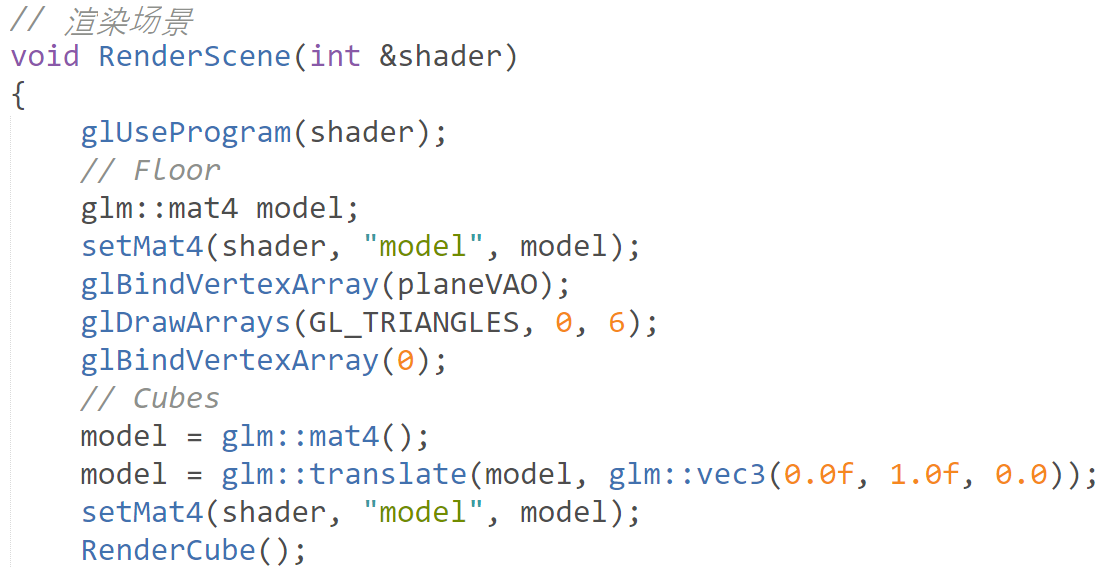


链接顶点属性：（对应顶点着色器输入）



（4）RenderScene函数

渲染场景。



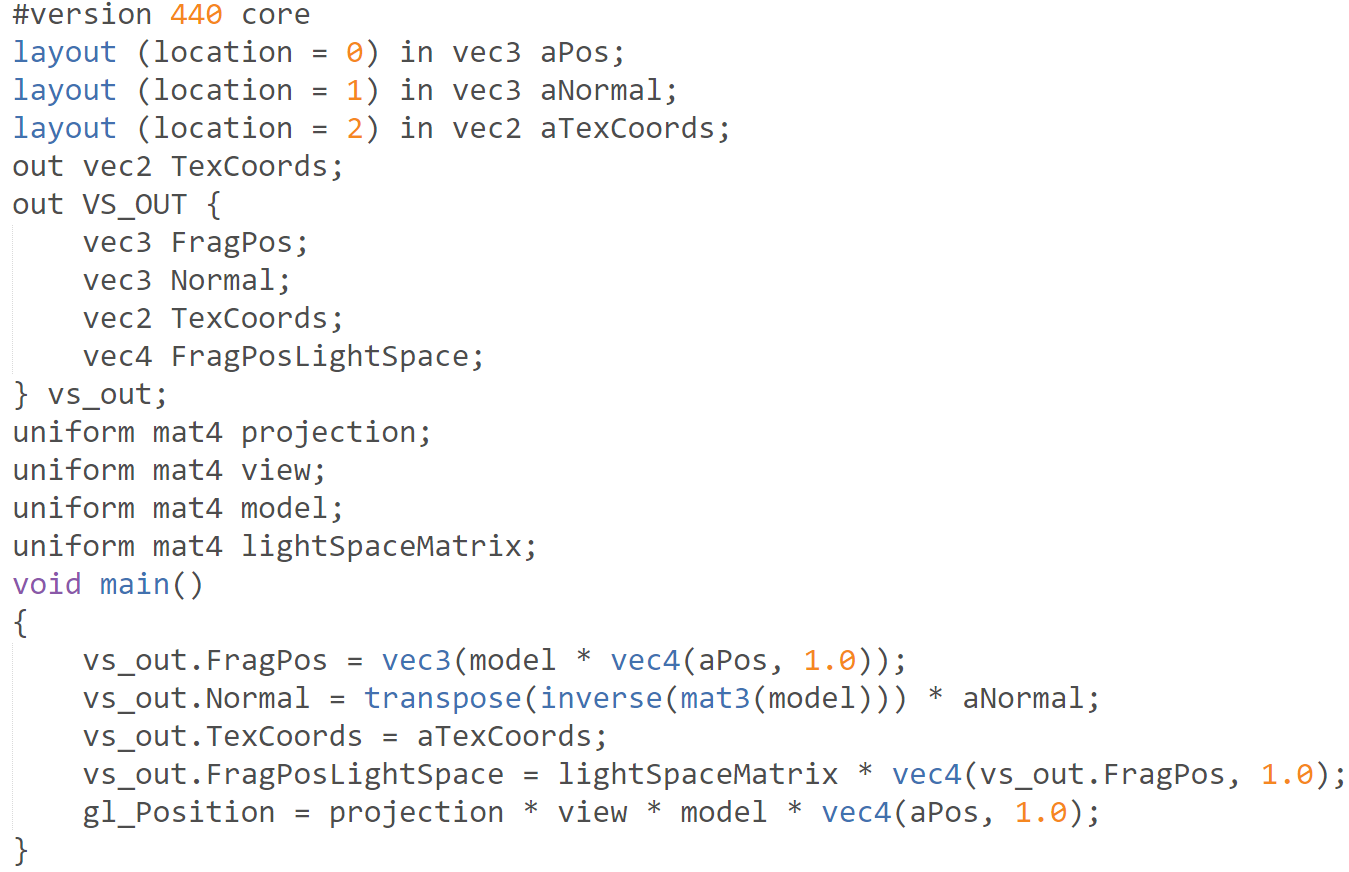
激活传入的着色器，用此着色器分别渲染floor和cube。

**4）生成阴影**

（1）顶点着色器（主着色器）

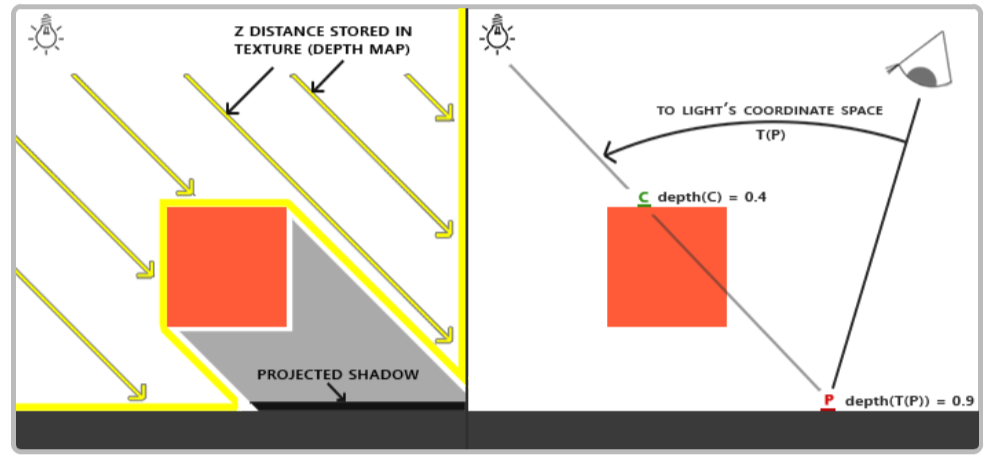
进行光空间的变换。

用空间变换矩阵lightSpaceMatrix把坐标从世界坐标转到光空间坐标，其余变换同普通顶点着色器。



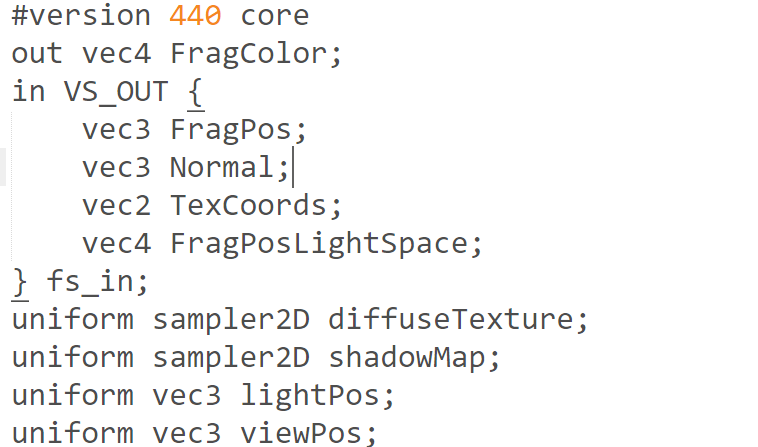
（2）片段着色器（主着色器）

首先介绍根据深度贴图决定一个fragment是否在阴影中的方法。

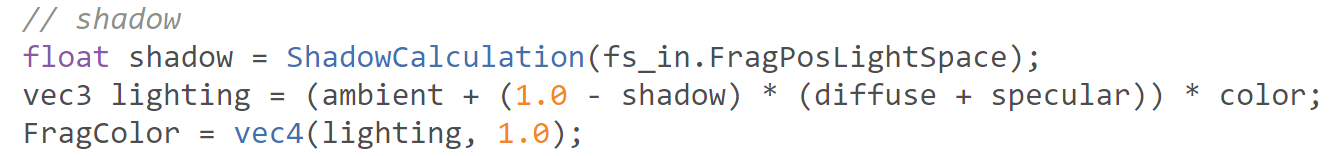


平行光下，考虑右图p点，渲染点P处fragment时，将P坐标转到光空间坐标中，z坐标对应P在此光源下的深度（此处为0.9）。使用P在光源的坐标空间的坐标，可以索引深度贴图，获得光视角中最近点的深度，此处为C点，深度0.4。索引深度贴图的结果0.4小于P的深度，可以确定点P在阴影中。

此处片段着色器相比于普通着色器增加了阴影计算。



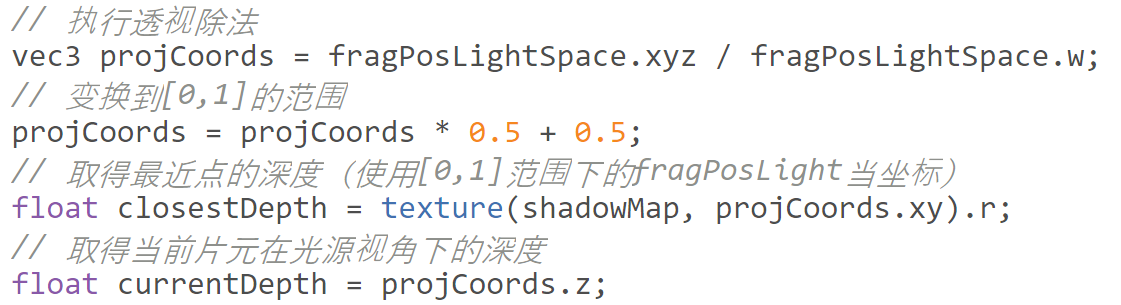
考虑到环境光照是固有的，即使是阴影部分也会有这部分的光照，我们将它独立出来加和，计算出的阴影因子不对它起作用。



shadow计算值为0时，没有阴影，光照正常。

shadow计算值为1时，diffuse和specular分量权重置0，只有环境光照。

为计算阴影因子shadow，我们使用ShadowCalculation函数：



① 执行透视除法，将裁切空间坐标的范围[-w,w]转到[-1,1]。

② 为和深度贴图坐标范围匹配，把projCoords变换到[0,1]的范围。

③ 得到光源视角下最近的深度closestDepth

④ 得到fragment在光源视角下的深度currentDepth

⑤ 比较closestDepth和currentDepth，若currentDepth值更大，fragment就在阴影中shadow = 1.0，反之光照正常，shadow = 0.0。

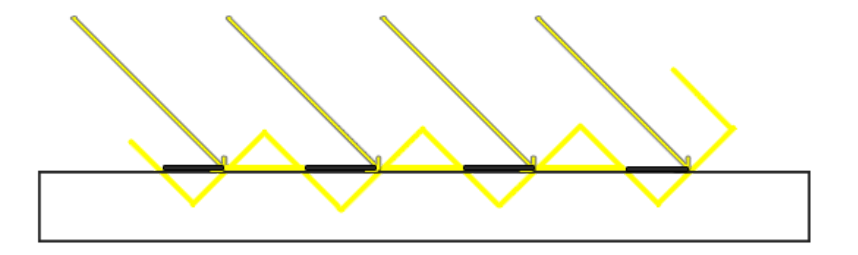
**2.2 阴影优化**

此部分的优化都在主着色器中的片段着色器里完成。

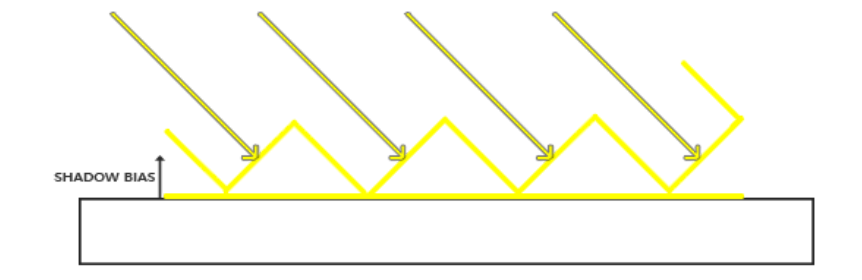
**1）阴影偏移**

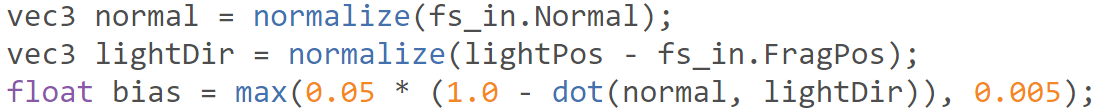
针对阴影失真（Shadow Acne）带来的不真实感，我们使用阴影偏移（Shadow bias）对表面的深度应用一个偏移量，避免fragment被错误地认为在表面之下。

光线方向与表面形成夹角时，深度贴图在一个角度下渲染，多个fragment会从同一个斜坡的深度纹理像素中采样，有些在表面上侧，有些在表面下侧，部分fragment被认为在阴影之中，会产生条纹。



应用偏移量后，所有采样点都获得了比表面深度更小的深度值，不会有fragment被认为在表面之下，没有任何阴影。



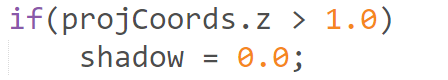


最后用currentDepth减去bias值得到最终深度值。

**2）采样过多**

默认把光的视锥不可见的区域认为是在阴影中，会使中心区域之外的部分区域显示错误的阴影效果。

当一个点的深度值超过光的远平面，就把shadow置为0，认为没有阴影。



**3）PCF**

此方法用于柔和阴影。从深度贴图中多次采样，每个采样点的计算结果可能都有不同，把多次结果做平均后再为阴影因子shadow赋值。

通过对纹理坐标进行偏移，确保每个新样本来自不同的深度值。此处采样得到9个值，后对计算结果求平均。

