# 计算机图形学基础-大作业报告

——坐标变换+纹理映射+Blinn-Phong光照+阴影映射

姓名：魏少杭 学号：20373594 学院：人工智能研究院

*注：终端make up指令可以直接链接运行。如果想看光照结果而不看阴影映射结果，则将src中的main\_无阴影.cpp内容复制到main.cpp中即可。*

## 一、综述

本次作业主要应用OpenGL完成。完成作业过程中，学习了OpenGL的渲染流水线和一些库函数的用法。代码实现主要依赖glad、GLFW、glm、stb\_image.h等库和头文件。作业主要实现了顶点坐标变换、纹理映射、光照、阴影映射等。下面从这四个方面分别介绍作业完成情况。

代码结构中，全体流程均在main函数中实现。所有顶点着色器和所有片段着色器都以GLSL语言实现，并放在文件开头，在main中构建着色器时调用。

## 二、顶点坐标变换

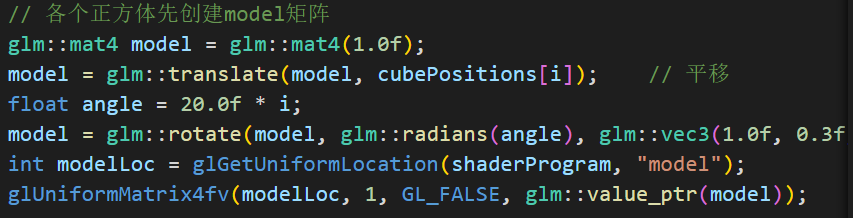
一个顶点通常包含x,y,z三维坐标，通常以四元组形式，经过局部空间-[model矩阵]->世界坐标系下的坐标-[视角矩阵view matrix]->视角空间-[projection矩阵]->剪辑空间-[viewport transform]->屏幕空间过程的变换。

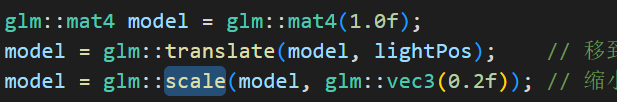
### 1.局部空间

物体初始所在的坐标空间即局部空间。初始时构建物体各个顶点即定义其局部空间坐标，通常为3维向量。

### 2.世界空间

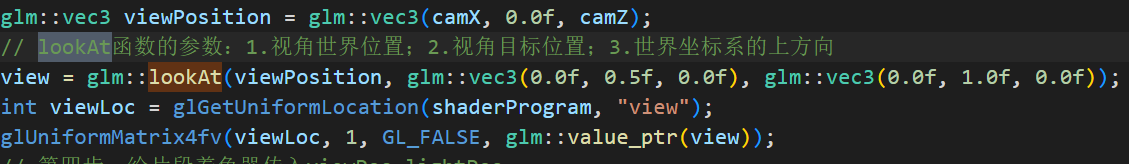
世界空间是所有物体共同存在的一个绝对空间，所有物体在这个空间中有自己的分布。由于局部空间不同物体的顶点可能位置有重合或很接近的情况，故需要用模型矩阵对各个物体的各个顶点坐标做变换，使得物体坐标从局部空间到世界空间。模型矩阵能够对物体进行位移、缩放、旋转等基本操作。在glm库下定义的各种变换函数可以实现这种变换。例如以下三种变换分别在glm中是glm::translate、glm::rotate和glm::scale。我们可以通过首先定义一个4\*4的对角阵作为model，然后model不断通过变换迭代得到完成平移和缩放的操作：





### 3.观察空间/视觉空间/摄像机空间

经常是通过一系列旋转、平移等操作，将物体从世界空间坐标转换到用户视野前方的空间坐标中。需要用view矩阵实现。glm::lookAt可以很方便地实现这个操作。



### 4.裁剪空间

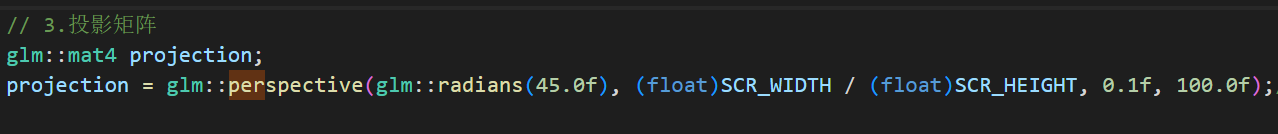
顶点着色器对坐标进行前面的变换之后，为了成像在一个屏幕的一定区域内，必须裁掉超出屏幕空间不可见的点。

由观察空间坐标转换到裁剪空间，需要用投影矩阵实现。投影矩阵主要包含两类：正交投影和透视投影。在本次实现过程中两类投影矩阵都有用到：

正交投影矩阵用glm::ortho实现，如下面例子，投影映射过程深度贴图部分由光源作为“相机”的坐标变换：

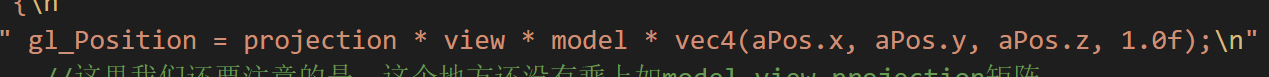


透视投影：glm::perspective实现。主要用于顶点着色器中进行透视投影，实现如人在观察事物时成像远近大小的逼真效果。45.0这个角度值是指的视野角度，是按照经验设定的；最后两项分别是屏幕空间中距离相机最近、最远的距离值。



进入裁剪空间之后，为了得到屏幕显示的图像，OpenGL通过在一个顶点着色器最后自动执行透视除法，将4D裁剪空间坐标变换为3D标准化设备坐标。最后通过glViewport函数指定视口大小，得到屏幕坐标。

在顶点着色器中，三次矩阵坐标变换如下图：



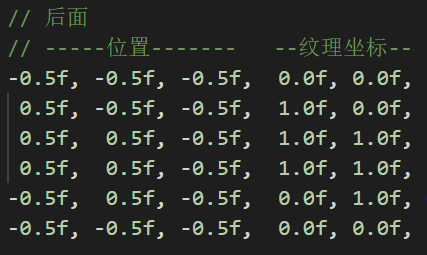
## 三、纹理映射



如图，可以对各个物体表面添加纹理图案。

### 1.纹理坐标

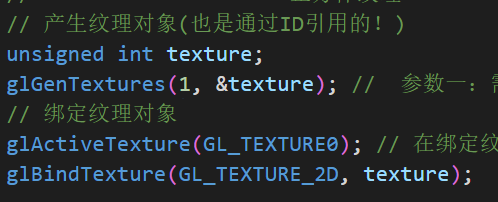
以正方体为例，对于一个正方体的OpenGL渲染过程中，每一个面由两个等腰直角三角形构成，且两个三角形斜边的两点相同。进入顶点着色器渲染前，需要定义这个正方形的顶点坐标、纹理属性。



如图所示，每行代表一个顶点的所有属性值，前三个我们定义为位置属性xyz坐标，后面两个定义为纹理坐标属性。纹理坐标与顶点坐标需要有一一对应的关系。2D纹理坐标是在x和y坐标上与顶点坐标对应。例如上图6行全为处于局部空间后面的顶点，而第一行顶点坐标值表示这个面的左下角，故纹理坐标（x、y均从0开始计数）也是对应的左下角。其他顶点以此类推，需要对每一个坐标做一个纹理坐标的一一映射。当然，如果形状比较复杂的物体，其顶点坐标与纹理坐标的映射关系需要具体使用函数来实现，此处为了使得后续着色器中光照的实现更方便，故使用简单正方体，直接手工标注即可。

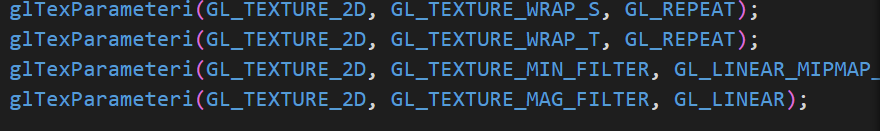
### 2.纹理映射与顶点着色器

步骤一：创建、绑定纹理对象



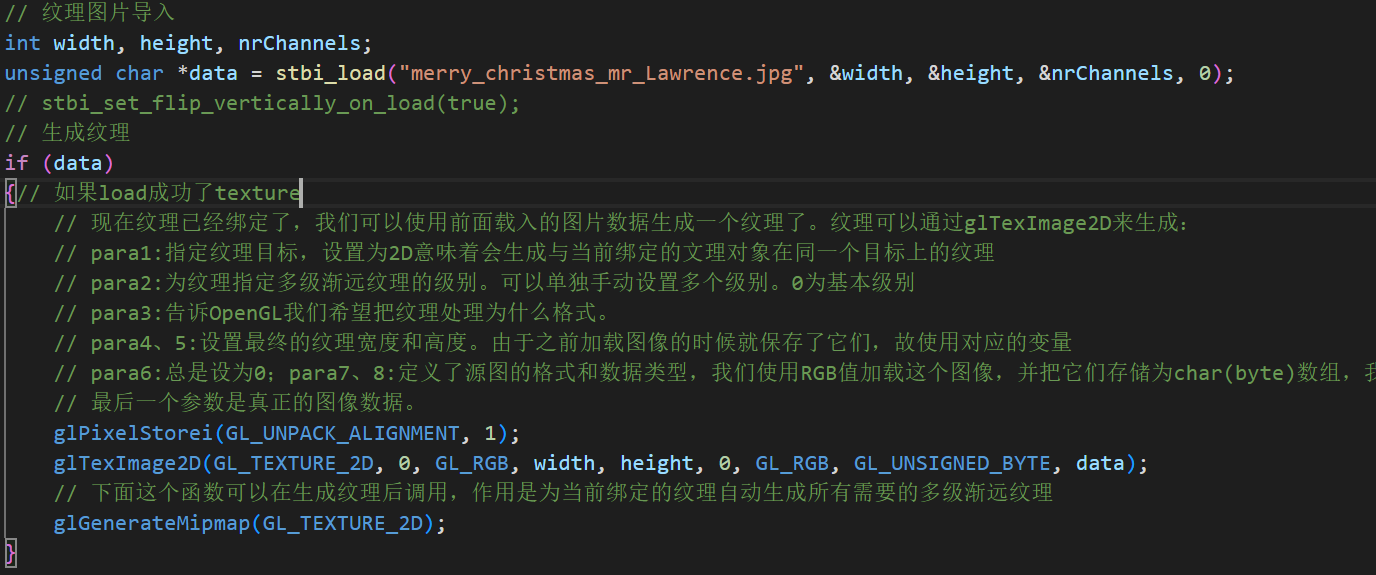
通过ID引用，使用glGenTexures函数生成纹理对象；绑定纹理对象使用glBindTexture。在绑定对象之前，可能需要激活纹理单元，这与是否有多个纹理对象有关。

然后为当前绑定的纹理对象设置环绕、过滤方式。



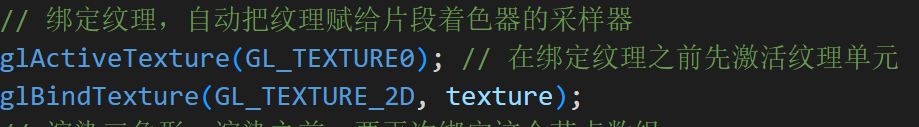
上图前两行表示纹理环绕方式设置。表示当纹理坐标设置在0,0到1,1范围之外时，分别绕着S、T轴（类似于X、Y轴）进行重复纹理。后两行表示图像进行放大和缩小时可以设置纹理过滤，通常有线性过滤、临近过滤等方式。在此处第三行表示设置当纹理被缩小时生成多级渐远纹理，根据距离远近调整解析度。

步骤二：导入纹理图片并生成纹理

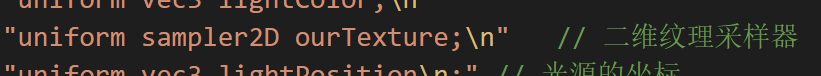


使用了stbi\_load库来将图片作为纹理样本。glTexImage2D能够将图片数据生成一个纹理，glGenerateMipmap能够实现上述的多级渐远纹理。

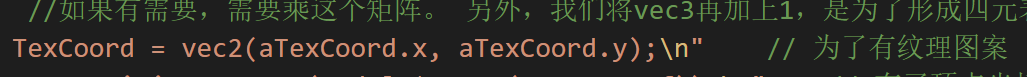
步骤三：渲染时送入着色器处理



渲染时，需要如上图进行激活纹理单元并绑定对应的纹理对象，绑定后自动送入对应纹理单元的片段着色器的输入参数中。



在片段着色器中，我们将之前的纹理对象作为二维纹理采样器。我们在片段着色器对输入的片段进行色彩生成时，还需要使用从顶点着色器送来的纹理坐标。



然后在片段着色器中使用内置的texture函数进行纹理采样，即得到纹理图案。



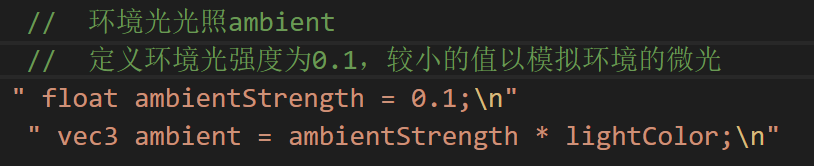
## 四、Phong、Blinn-Phong光照

### 1.Phong光照理解与实现

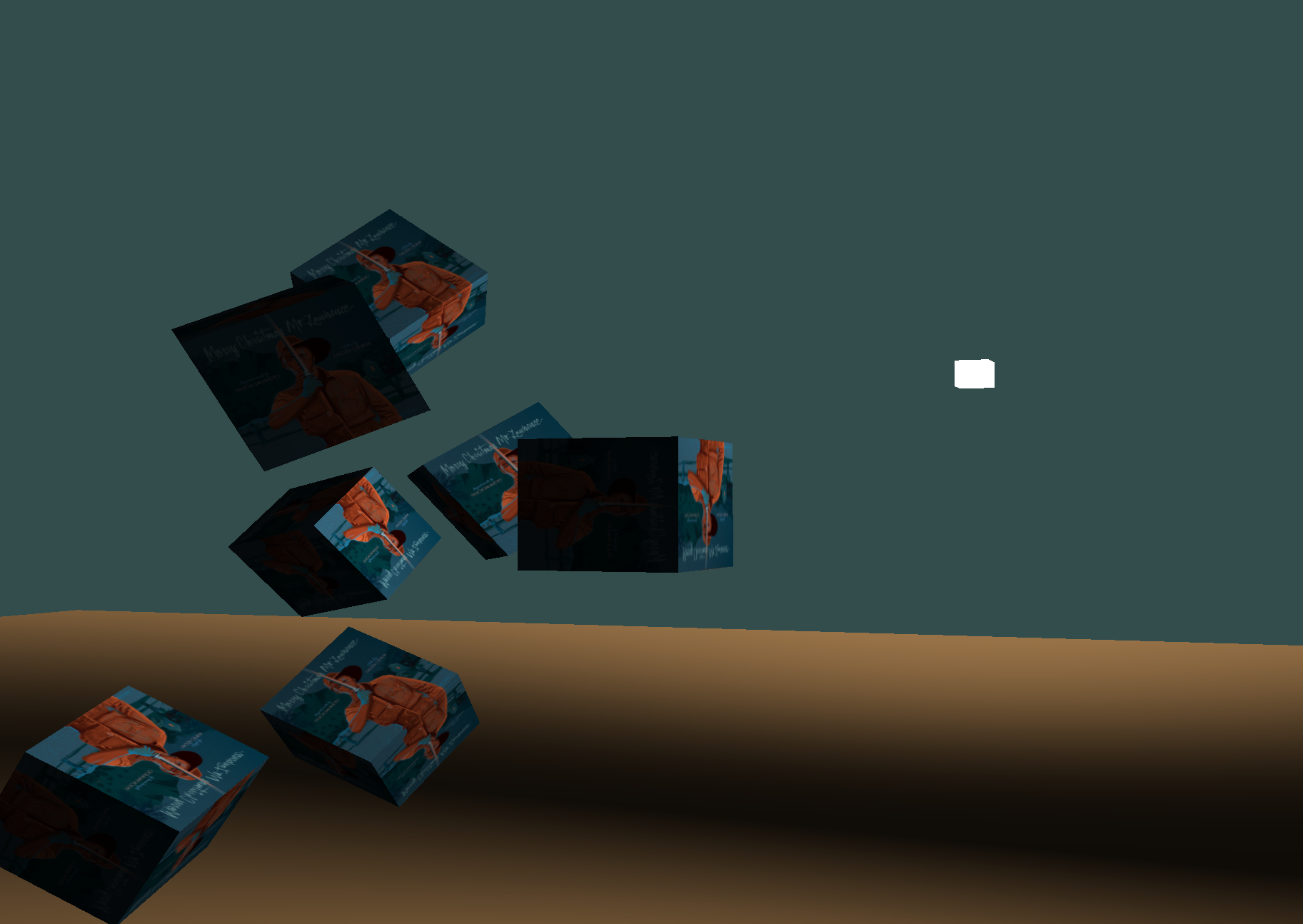
常见的光照模型Phong和Blinn-Phong光照的实现思路主要都产生于光的反射这个基本常识。

Phong光照由环境光、漫反射、镜面光照构成。

（1）环境光：周围环境本身会反射微弱的均匀的光。一般来说，环境光的大小受周围环境材质、光照强度等因素有关，是许多分散的不同光源的共同作用产生的。由于全局光照实现起来较为困难，在片段着色器中，以一定常量比例均匀弱化该空间内的光线亮度。



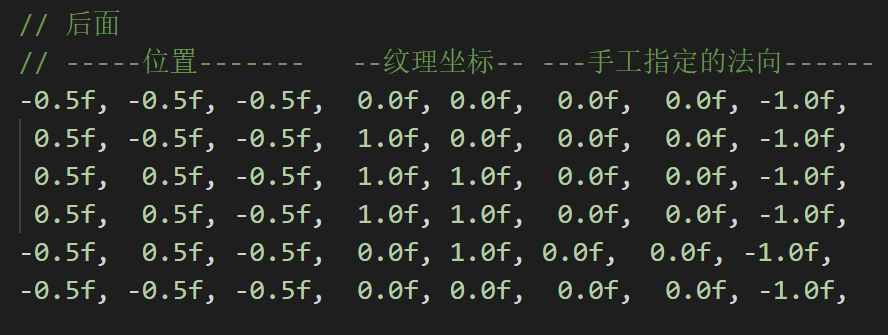
其效果如下图，可以看到，在完全背离光的面上仍然会有一点点亮度，这就是环境光的作用。



（2）漫反射光照

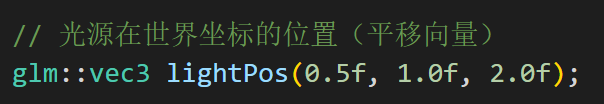
漫反射基本思路是计算物体表面的片段对应的法向量与光线方向夹角，夹角越小则说明该片段能够获得更多的亮度。因此计算的核心就是计算片段的法向量和光线的夹角。

之前我们定义了正方体作为物体，就是因为此处漫反射需要计算每一个片段的法向量较为复杂，为了尽可能简化法向量的计算而将精力集中在光照的实现上，我们可以在定义正方体的顶点集合时新增顶点的属性，即该顶点所在面的法向量。例如下图：



对于正方体的后面，若定义法向方向均为从物体内部指向表面以外，则其法向方向是沿z轴负半轴。标准化的法向长度为1。

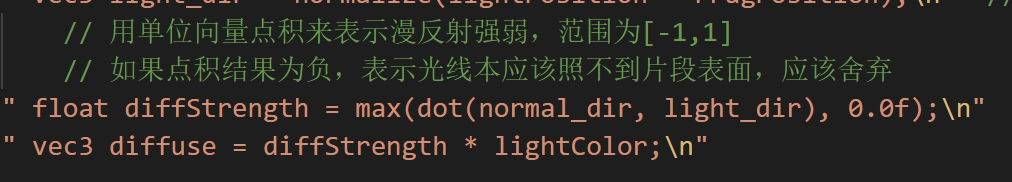
获得法向后，要计算光线，使用两点坐标之差即为一个向量的方法，再通过标准化，获得光线的单位方向向量。定义光源在世界坐标系中的坐标为全局变量三维坐标：



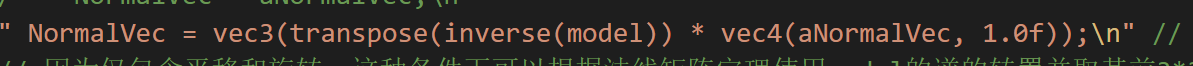
计算光线方向，是指的从片段到光源的方向。



然后使用单位向量点积，表示漫反射的强弱。因为点积越大则说明两个向量夹角越趋近于0，则光应该最大。我们通过标准化得到的单位向量再内积，是为了直接得到漫光照的强度比例因子。同时考虑到可能内积为负的情况，即光线根本不会反射的情况，所以要剪去这部分负值。进而得到了漫反射光的强度。



需要注意的是，由于我们之前定义的正方体可能会在顶点着色器中对其进行model变换，在这种情况下法线方向也是有变化的。但是向量直接进行model变换是不能得到正方体进行旋转后得到的法向。针对这个问题，通过查阅资料，得到了一个证明结论：**如果在model变换阶段不进行缩放操作的话，那么model这个4\*4矩阵仅需要求逆再求转置，然后与法向向量的4元向量进行乘积，得到的就是法向在物体进行了model变换后的方向。**在顶点着色器中需要这样操作：



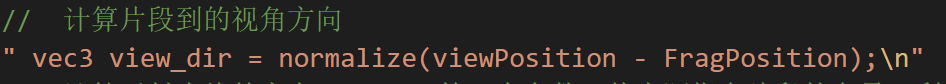
（3）镜面反射

镜面反射除了需要考虑光线方向、片段表面/法线方向，还需要考虑视线方向。如果视线方向与光线经过正反射得到的方向相同的话，则反射光强度最大。而镜面反射还与光源、物体片段本身属性有关，这个将用一个常量参与数学计算来表示其意义。我们定义这个镜面反射常量为128。这个常量能够很好地表现出镜面反射的特点。下面计算方向：

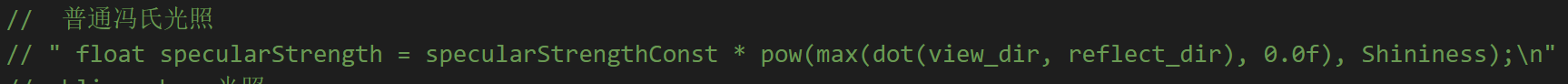
首先，计算反射光线：根据光线方向和法线方向，用OpenGL内置函数reflect计算反射方向。



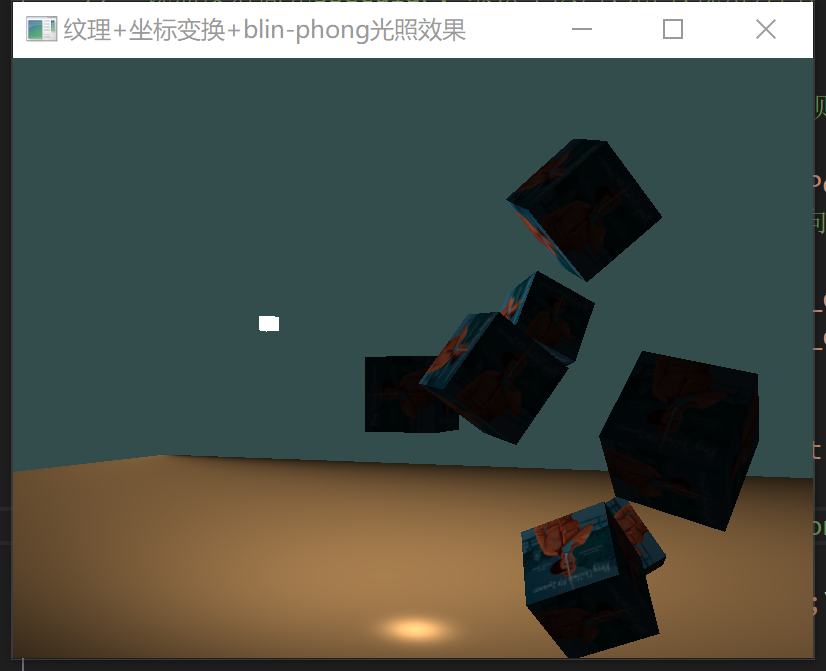
其次，视线方向：



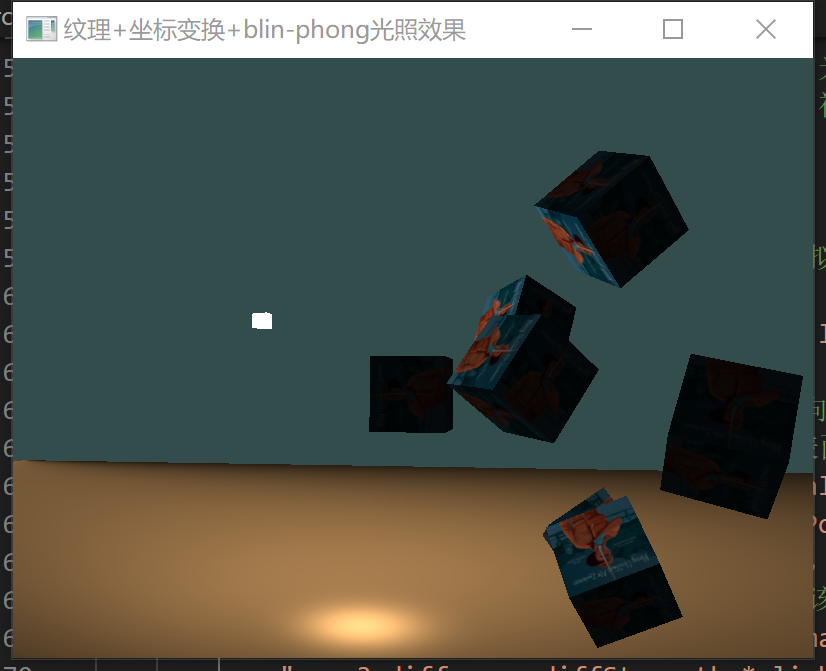
最后，与漫反射一样，用标准化的方向的内积作为反射光强比例因子，并保留非负部分。



冯氏光照效果如下：



可以看到，普通的冯氏光照在地面上镜面反射光十分明显。下面将调整参数，进一步观察规律。

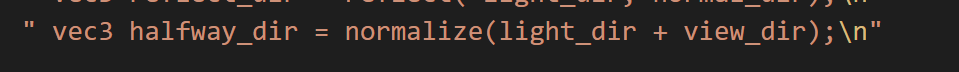


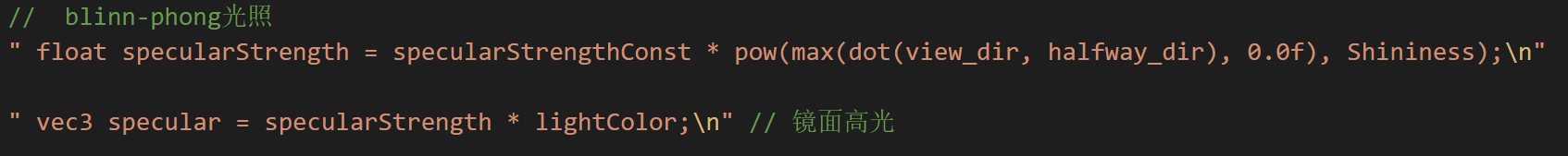
与第一张图（镜面反射常量为128）相比，第二张图（常量为32）的镜面反射区域强烈程度下降，光线更加柔和。

### 2.Blinn-Phong光照

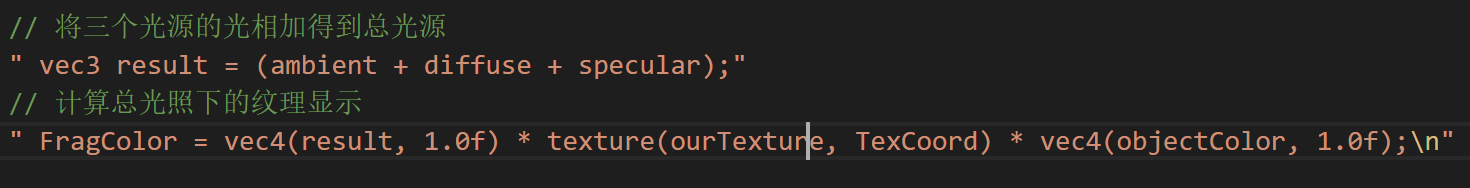
当Phong光照反光度很低时，会出现断层现象。主要问题是Phong算法计算镜面反射的时候，当光源在视角以下时，镜面反射的光线为0，而其他反光度又无法补充，所以会出现很黑的情况。

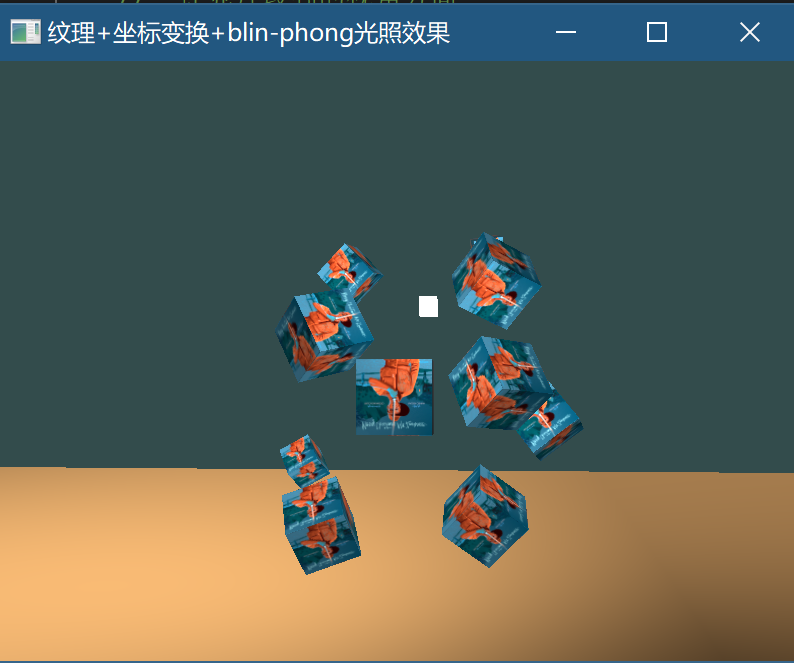
为了解决这个问题，Binn-Phong方法使用半程向量与法向的夹角取代反射光线与视线的夹角。半程向量是一个单位向量，其方向是光线方向和观察向量的中线方向。其计算可以由光线方向和观察向量相加（因为都是单位向量，相加后结果即为中线方向），再标准化，得到半程向量。如下图：

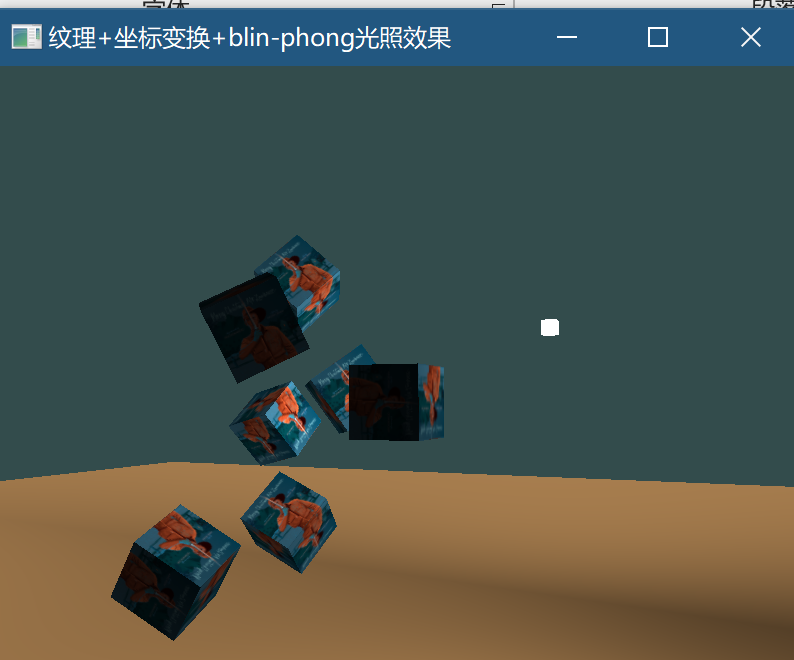




最后，将三种类型的光组合起来，得到光照效果。







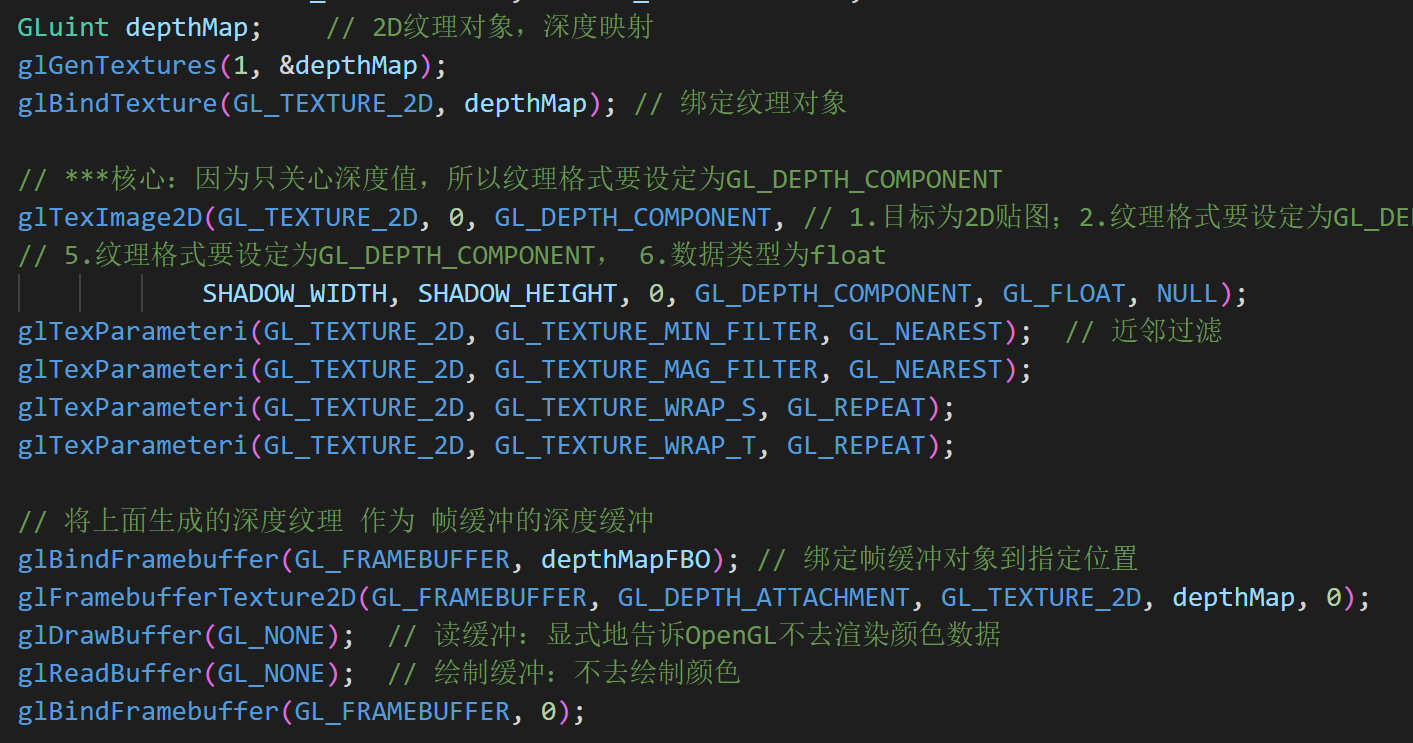
## 五、阴影映射

### 1.阴影映射的理解

以光源的视角看物体，一条光线射线上的最近片段可以被看见，其他片段均在阴影中。核心是找到从光线视角出发的各点的深度关系。主要由两步构成，第一步是渲染深度贴图，第二步是将生成的深度贴图用来判断是否某个片段是否应该处于阴影之中。

### 2.深度贴图

从光的透视图里面可以渲染出深度纹理。而这个纹理不包含颜色信息，仅包含深度值。具体操作时需要创建2D纹理对象depthMap，然后提供给帧缓冲的深度缓冲使用。

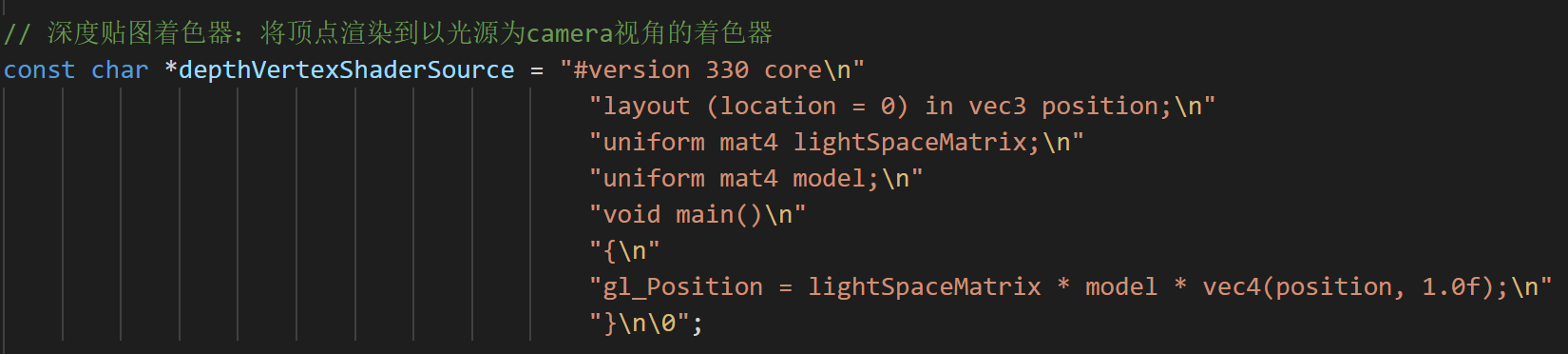


然后，在渲染阶段，我们需要首先渲染深度纹理贴图。由于这一步需要以光源为视角，所以需要计算世界坐标到光源坐标的变换矩阵。具体又分为计算光源的投影矩阵和光源作为视角的View矩阵。



渲染深度纹理贴图需要构建新的着色器：

顶点着色器：主要进行位置变换到光空间中。（片段着色器无操作）



在渲染了深度纹理贴图后，要使用原来的渲染物体的着色器来渲染立方体和地面。





而渲染正方体和地面的顶点着色器需要传入变换到光空间的矩阵，并输出片段在光空间的坐标。

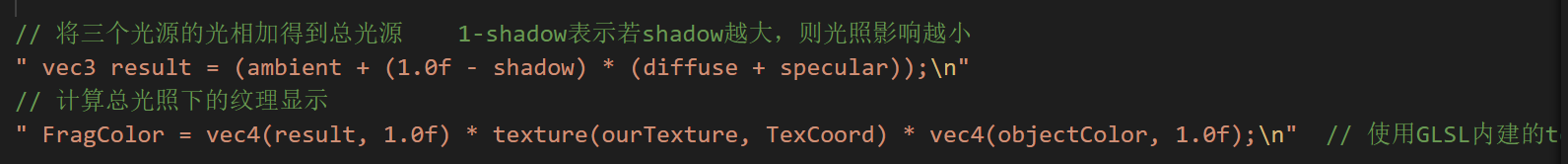


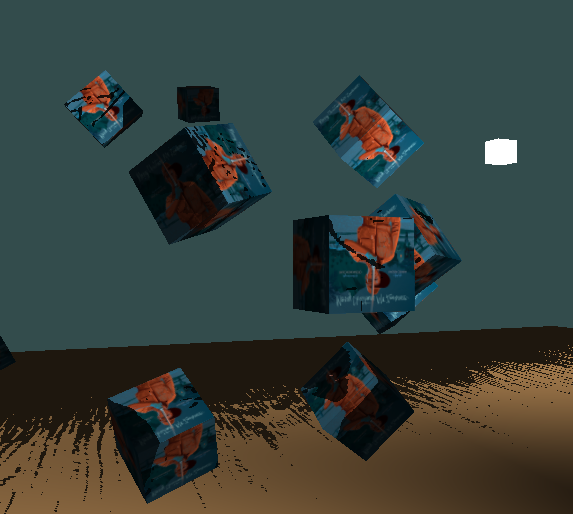
而片段着色器中需要传入刚刚得到的片段的光空间坐标。这个光空间坐标的作用是获取这个坐标的深度z。将该片段的深度z与之前渲染得到的深度纹理贴图（本质还是深度）映射值进行比较。若该片段深度比相同的屏幕空间x,y的最近深度更大，说明该片段不应显示，而是在阴影之中。若在阴影之中，可以设置阴影变量，进而控制漫射光、镜面反射光等。



计算获取阴影的代码如上。其中第一行意思是进行透视除法，这对透视投影来说是必须的，目的是让w位于-1到1之间。

获得了阴影之后，就可以设置阴影值对光源光照的影响。





阴影的效果图

## 六、展望与改进

在纹理映射方面需要尝试更多的纹理环绕方式、过滤方式，以获取更加美观、真实的效果。

光照部分不足之处有：1.环境光照由于没有设置真实环境下各个光源的计算，而是设置常量一笔带过，不够真实。2.整个光照部分仅考虑了反射角、观察角等而没有考虑光源距离带来的光照明亮度影响。

阴影映射的效果不好，其中比较核心的部分是深度纹理映射的投影矩阵的生成，难点在于要根据自己设定的形状、位置，来调整生成lightProjection矩阵的参数，如平截头体的左右坐标、顶部底部坐标、远近平面。因为这里使用的是定向投影，而没有用透视投影，可能因此地面、部分正方体等阴影效果并不是很好。可以进一步改进。