**实习四**

# 1.如何编译代码

将给出的头文件cpp文件覆盖原工程。

参考命令行运行，但是请保证存在Result\_Picture的文件夹。为了方便查看而将图片全部保存在此子文件夹里。

# 2.合作

无。

# 3.参考资料

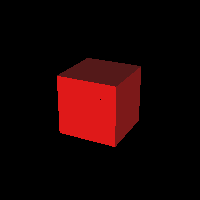
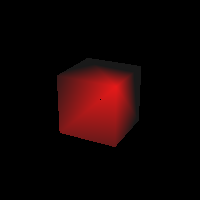
计图PPT，上课所学知识。

# 4.存在已知问题

（1）由于cube是使用三角形拼接而成。因此虽然法向量是用权重计算而成，但是在拼

接口处却会明显有分割。左图是使用加权法向量生成的图片，右图是最开始只有基础漫反射的单独面法生成的图片。发现右图接缝处并不太明显。反而加权法向量比较明显。

因此考虑对于拼接处需要使用当前边的相邻三角形的平均法向量来进行计算。



（2）我的世界（场景8）里的鸡的翅膀部分呈现阴暗，显示不清楚，一开始猜想是贴图

错误，于是做出了一个没有纹理的图，发现翅膀这一面确实比较暗。

因此考虑应该是光照的问题，可能需要在漫反射处进行改正，或者调节系数。



# 5.附加

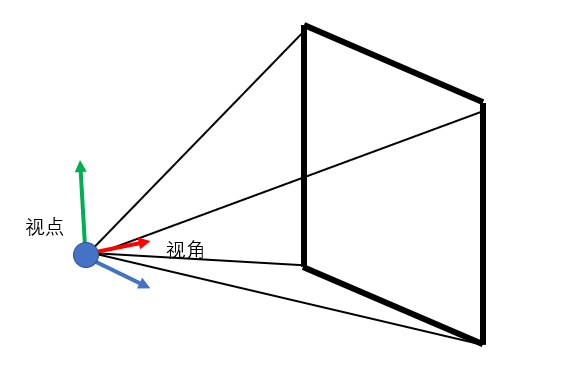
无

# 6.心得与分享

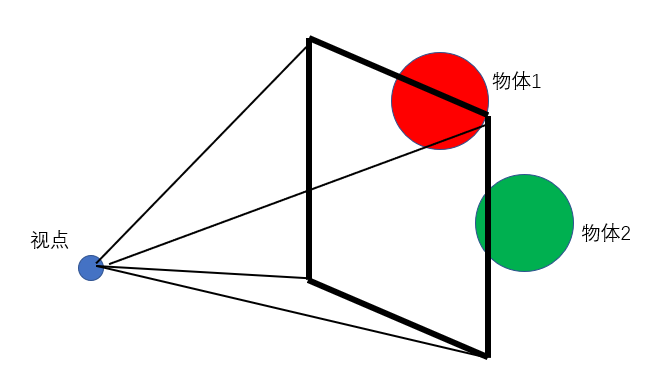
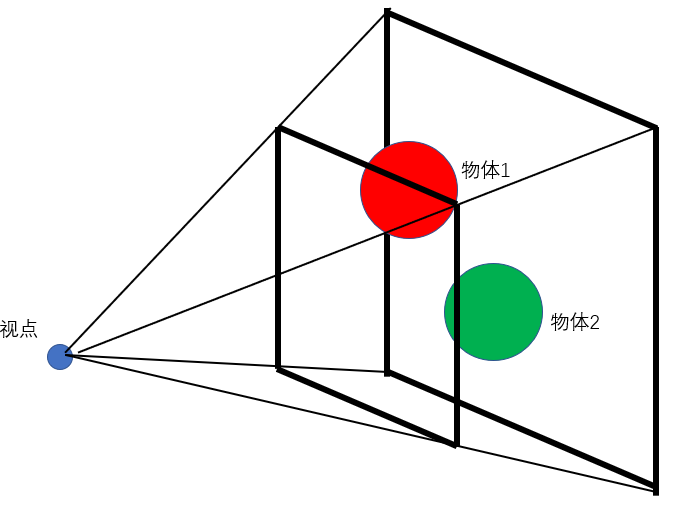
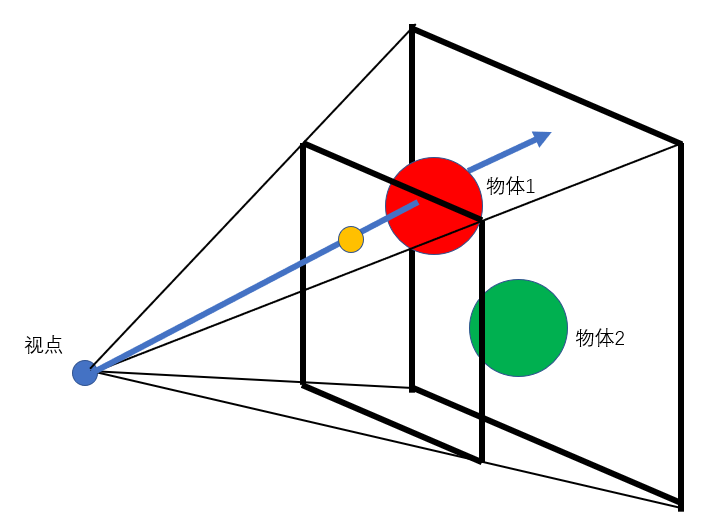
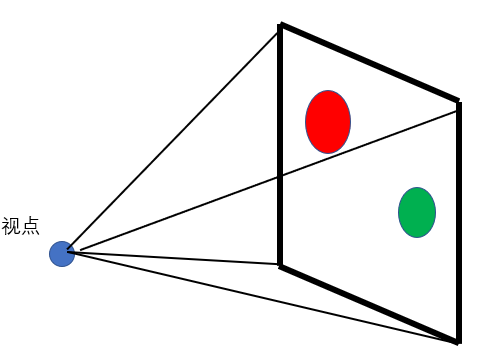
## 6.1透视投影

对于本题中所说的透视投影中虚拟平面不考虑距离很是困惑，于是首先自己模拟了一遍，发现确实如此。

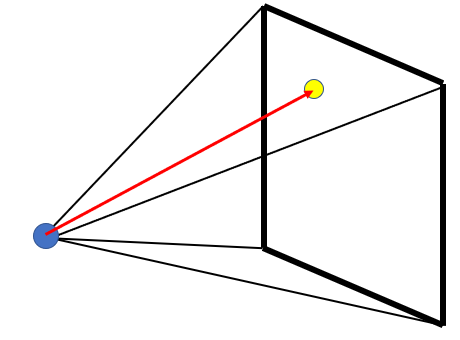
对于给定一个透视投影相机，如下图。给出一个视点，以及三个维度确定视点方向，给定视角表示范围。



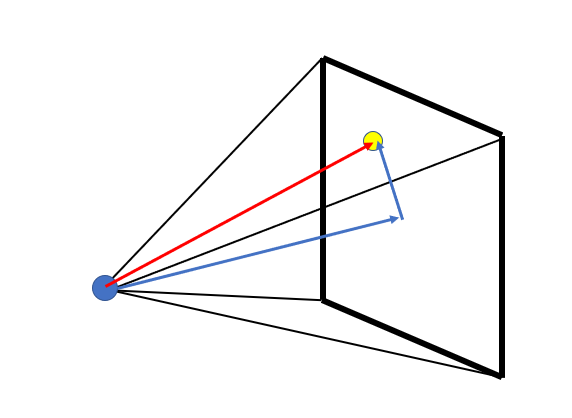
之后如果有物体就需要制定一个虚拟平面，然后可以假定能够将物体以一个台体框起来，然后在每个像素点射出一条射线最终得到一个投影面。

每条射线ray都会以当前视点为起点（图一蓝点），以虚拟屏幕上的一个点为终点（图二黄点）然后做出一条向量（红色箭头）。那么这条向量的normal的单位向量就是我们需要求的ray的方向。之后也就是需要用这条光线对物体进行求交。

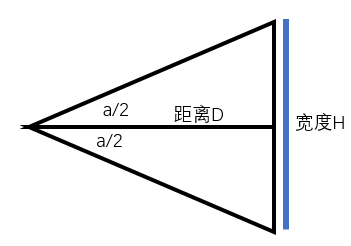
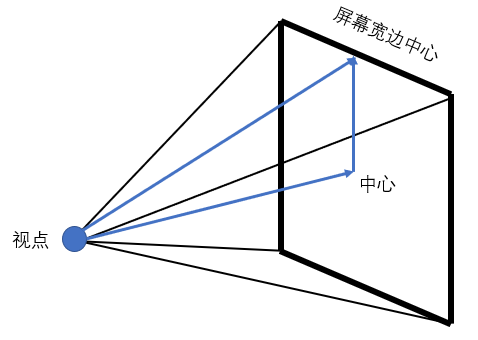


对于求法。因为虚拟屏幕是矩形的，那么我就以矩形屏幕的中心点作为分割。如图所示。就是假设当前视点所看的方向正对虚拟屏幕中心，即向量垂直于虚拟平面，那么最终向量就是视点到屏幕中心的向量，加上屏幕中心到当前屏幕点的向量，两个向量的和。



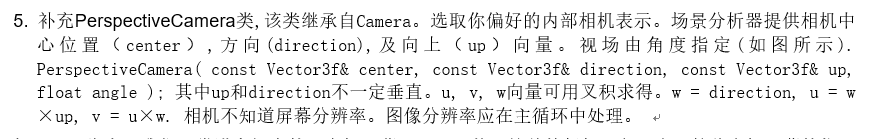
由于在代码里，已经有了这个视点的方向即direction，而且有视点位置。现在所需就是屏幕上的点的三维坐标，以及屏幕中心的三维坐标了。

先求虚拟屏幕中心的三维坐标。由于给出了屏幕的视角，但是没有给出虚拟屏幕距离，而且没有懂为什么与距离无关，那么我先假设视点与虚拟屏幕的距离设为D大小为10个单位。给出了视角a，那么如下图一所示，以视点，中心点，屏幕宽边的中心点做一个直角三角形。三角形视点处的角度为a/2，,因为如右图，两个三角形时对称的，可以知道是一半的大小。然后又假设了距离D，那么可知：tan（a/2）=（H/2）/D。因此H=2\*D\*tan（a/2）。如果有视角比例那么也可以做出虚拟屏幕的高。这里就假设宽高相等。

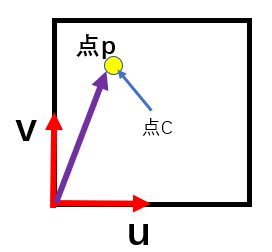
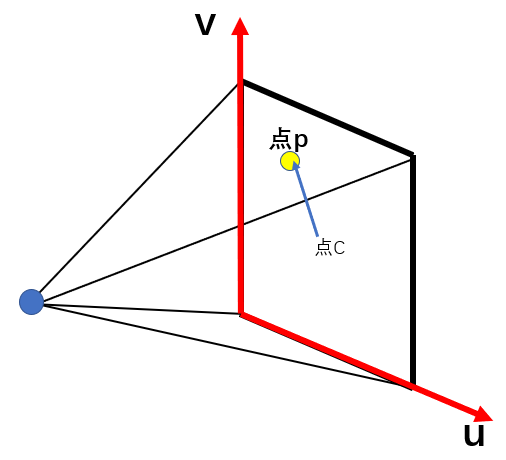


求屏幕的点的坐标。可知如果是在main函数里去双重循环求坐标的话，那么只能传进来一个二维坐标，看透视相机的参数也正好有一个二维坐标。当然因为在虚拟屏幕已经求出中点和宽度，那么只需进行几何计算就可以得出屏幕点的三维坐标。

方法是以虚拟屏幕左下角为一个原点。给虚拟屏幕做出u和v两个坐标轴，u和v的做法要参考文档中给出的方法。



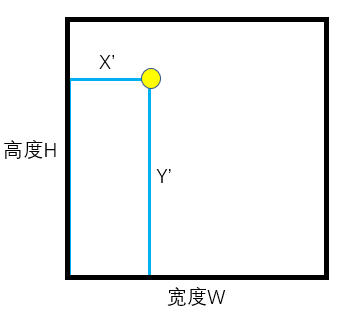
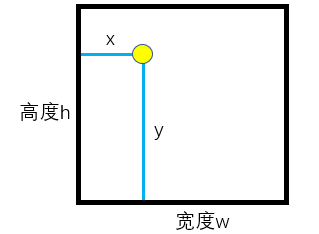
那么我们可以根据原点和点P（当前的屏幕点）得出一个向量（紫色）。因为我们已知u，v是两个三维的单位向量，这样才能定义好坐标轴。我们也知道P的二维坐标是（x，y）。那么OP的三维坐标就是u（Xp-Xo）+v（Yp-Yo）。对于原点的坐标因为我们知道中点C的三维坐标，那么根据比例就可以得到原点o的坐标。这样我们最终可得向量cp的坐标了。cp=op-oc。



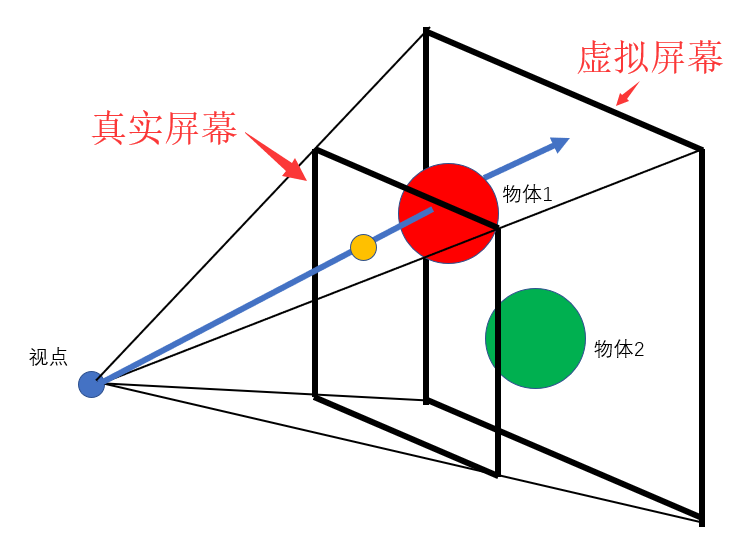
但是我们在main函数里传的坐标只是当前真实屏幕的坐标，而main里并不知道虚拟屏幕有多大，这就无法将两个二维点进行对应了。

因此考虑采用传入比例的方法。我们在main函数知道当前真实屏幕的宽度，那么我们就需要传入一个比例。然后在相机内部根据比例进行正确的计算，得出真正的二维坐标点。然后用二维坐标点去计算三维坐标。

如下图，假设左图是真实屏幕（较小），右图是虚拟屏幕（较大）。我们在main函数里知道当前点坐标（x，y），和真实屏幕宽w和高h。之后传入一个比例坐标（ x/w ，y/h ），就可以在透视投影相机里得到这个比例了。然后在相机里内知道虚拟屏幕的宽W和高H，那么就可以得到相机内的相对二维坐标（x’,y’）=(x/w\*W,y/h\*H)。



综上所述，我们就可以得到一个ray光线的方向了。当然我的同学之前问我，如果虚拟屏幕比真实屏幕大的话，就无法看到物体了。如下图，如果虚拟屏幕是物体的后面，而真实屏幕过小的情况。给出的解释：给出虚拟屏幕只是方便我们进行几何运算，求出光线的方向。最后我们还是要将光线的方向去正则化的，所以无论设的距离D有多大，最终求的都是相似三角形，最后都会进行正则化，自然距离也就不相关了。而且对于“看到”物体这一步是在求交算法处进行的，这一步只是生成光线。

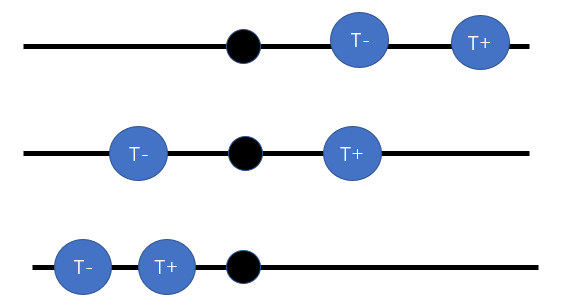


## 6.2物体求交

具体公式都在PPT里给出。

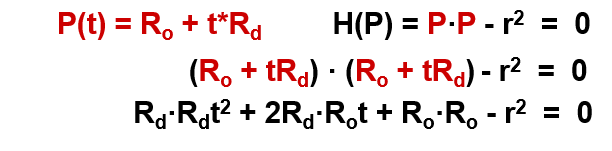
（1）对与球体的计算

需要用到判别式。而且在对transform求交时，需要注意a的值，改成Rd的平方。然后根据判别式的值分为三种情况，然后对于判别式大于0的情况又分为三个情况：较小的t比tmin大，较小的t比tmin小，但较大的t比tmin大。较大的t比tmin小。以下t的取值显而易见。



计算部分参考公式，需要注意这是以原点为球心的形式。对于普遍方法需要再给出

一个球心坐标，（P-C球心）\*（P-C球心）=r\*r

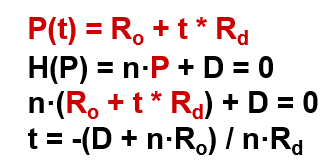


求法向量主要是采用几何法，已知当前交点与球心坐标，因此秩序将这个向量正则化便得到了法向量。

（2）对于plane平面求交时

需要注意虽然平面截距式为ax+by+cz+d=0，然后np+d=0。但是我们计算的公式使用的是np=d，因此在构造函数应将d取负。

计算方法直接参考公式即可。



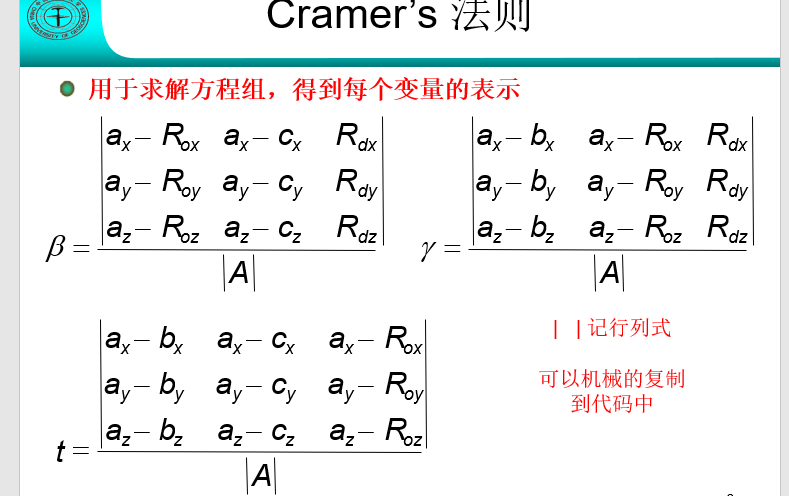
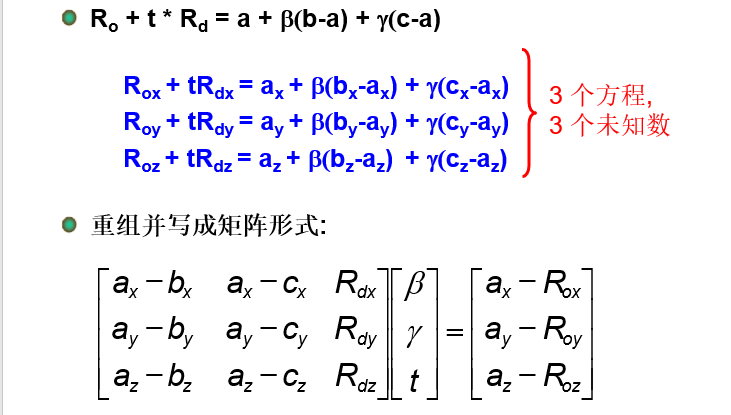
平面法向量在构造时已经给出，因此很方便求解。

（3）三角形求交。

使用重心求解，可直接参考公式。

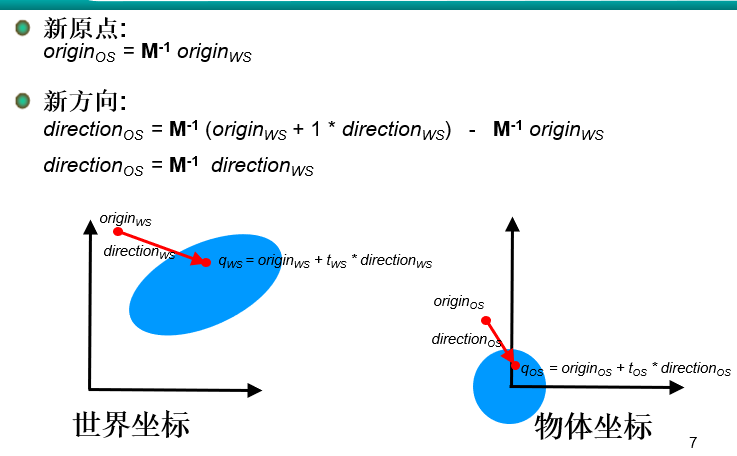
最后如果满足条件需要将三个normal带入normal=a\*n0+b\*n1+c\*n2。



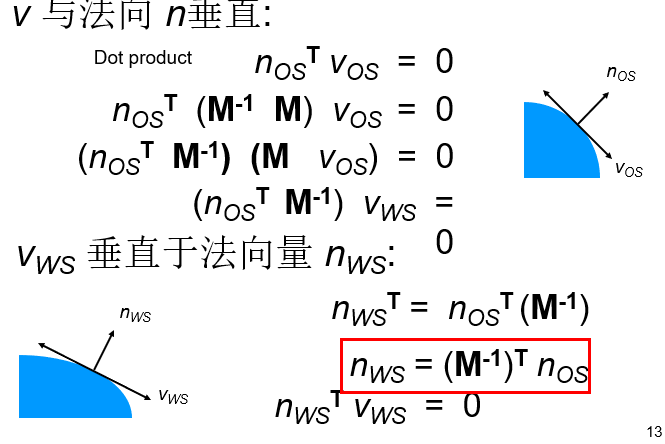


（4）transform求交。

查看了输入文件的格式，只需要将光线进行逆变换即可，矩阵已经在解释器里给出。因此直接用变换的光对原物体进行求交即可。当然这里采用了不正则化方向的方法。因此只是在对球体求交时的a改成Rd的平方。返回的t值直接可用，很方便。



对于法向量的计算，需要严格按照公式进行计算，对切向量进行变换的方式。

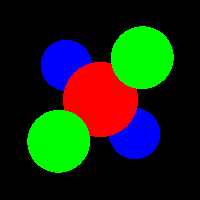


左图是求法向量时方向向量没有正则化的不正确的情况，右图是正确情况。

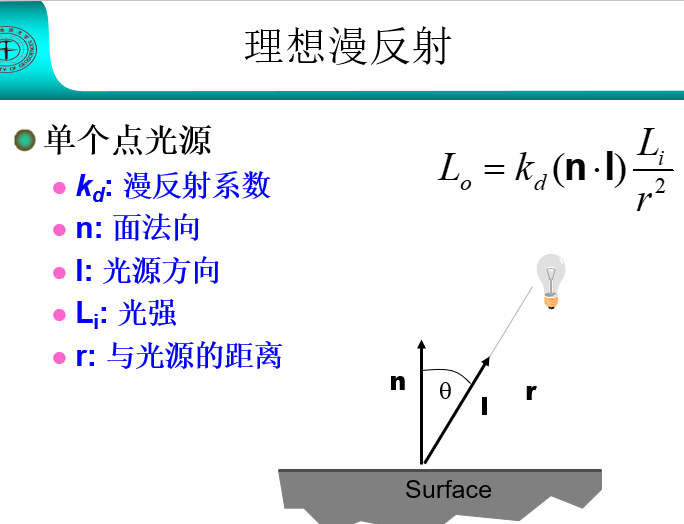
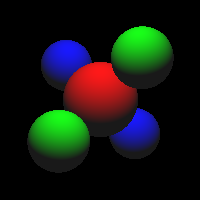


## 6.3光照

（1）最开始只是采用一个最理想的漫反射，即角度任意漫反射值相同，发现并不太真实。公式就是将颜色设为Material的diffuseColor。

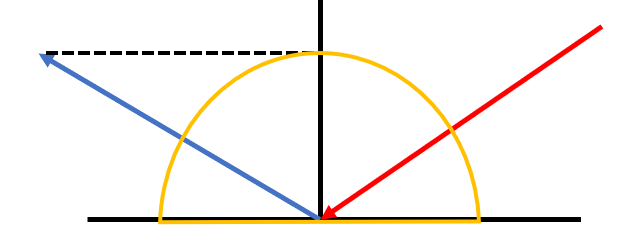


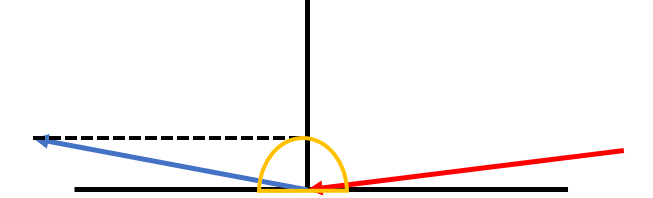
（2）之后进行改善。使用phong的角度漫反射光照。

这个地方花了一段时间进行理解，为什么公式是这个样子。这里我普通的情况是图一，因为当前是理想化的，那么所有方向的反射光都一样，那么我们可以很方便的计算垂直方向的光的大小，即光线乘以cos（a），然后其他方向又都相同，那么呈现的就是一个半球的形状了。

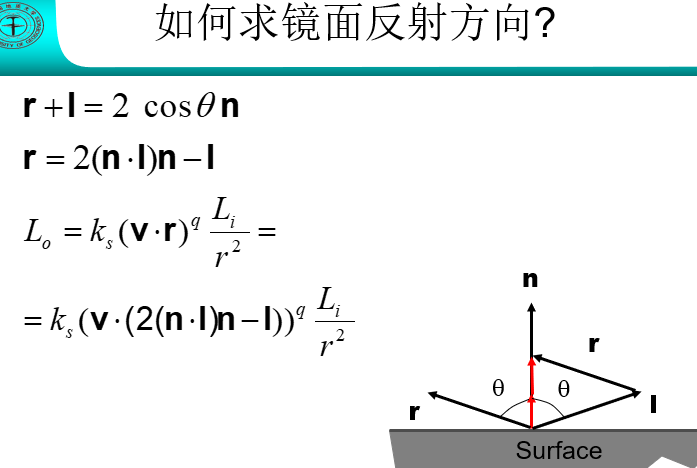
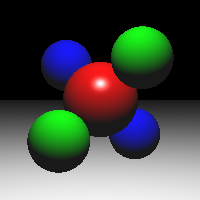
而如图二，如果倾角太大，即快要与表面平行，那么不应该有很大的漫反射的光线，而对于最初的方式，不管倾角多少漫反射值相同，这也就可以解释生成图片的差异了。





（3）使用phong的不用求反射光线的公式求出高光反射。

需要注意在代码中，要先对direction以及normal等向量正则化，不然会得出错误结果。

## 6.4纹理映射

在Mesh里已经设置了texCoords的三个二维的纹理坐标。我们只需要在三角形求交里运用求法向量的相同方式求出纹理坐标的权重即可。使用hit的setTexCoord函数，设置纹理的二维重心坐标。

然后在Material里的颜色设为当前材质的texture，二维坐标使用hit的texCoord得到。需要注意如果有材质是是应该将漫反射的系数进行替换，而不是全部替换。

## 6.5总结

本题目做起来还是比较有成就感的，每一题也不是太难，因为基本上是一步一步实现了课堂ppt里讲的内容，逐步完善。而且一步一步生成更加真实的图片，会感到比较有趣。本题可能是需要在透视投影处进行一定的思考，因此确实前期思考比较重要，然后把自己的正交投影思想讲给了许多同学更加有成就感，并且加深了印象。对于求交以及求法向量的方面主要是参照课堂的公式即可，这能够更深的理解公式，当然也顺便进行了复习。。。最后的纹理映射比较坑，就是不知道如何使用函数进行传递，之后把代码反复读了一段时间才明白做法。