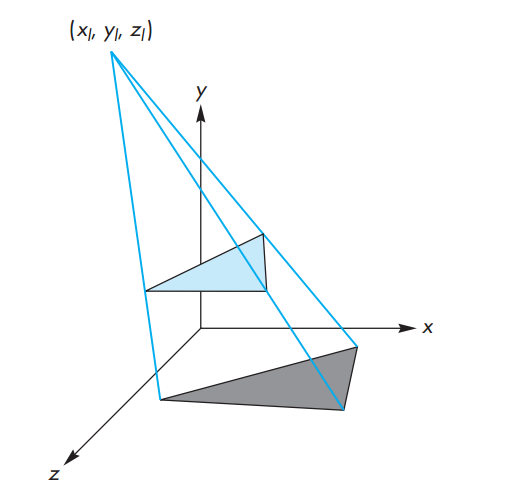
实验3.2 投影和硬阴影

1. 实验内容
2. 熟悉在OpenGL中实现正交投影变换。
3. 了解使用投影变换实现场景的硬阴影效果。
4. 理论背景
5. 投影和硬阴影

[注意]本节内容和书本上通过模-视变换矩阵来计算阴影投影矩阵方法不一样。

投影变换矩阵的一个应用就是生成简单的阴影。从物理上看，有光源才会产生阴影，如果光源投射的光线被物体遮挡，那么遮挡的部分被投影在地面上即产生了阴影。为了简单起见，我们假定阴影落在了地面上，即OpenGL坐标系下的 平面。

如下图所示，我们假定光源位置在 ，物体由三角形表示，投影平面上的黑色三角形区域即为阴影，我们称之为阴影多边形（Shadow Polygon）。



下面我们来推导阴影投影矩阵如下。假设三角形任意一个顶点坐标为 ，投影到投影平面之后的坐标为 ，因为该点在 平面上，所以 。根据比例关系可得如下公式：

求解可得，

同理可得和。

为了能够方便的通过矩阵表示出投影关系，我们将所有坐标设置在齐次坐标系下，那么投影关系就能表示成如下公式。

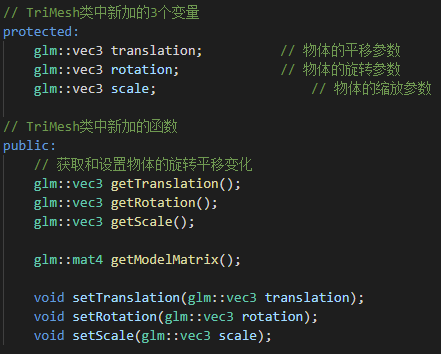


最后即可得到，

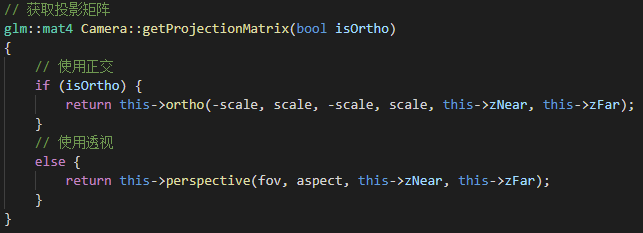
所以生成硬阴影的关键在于投影矩阵的求解。

1. 实验内容
2. TriMesh类和Camera类的内容补充

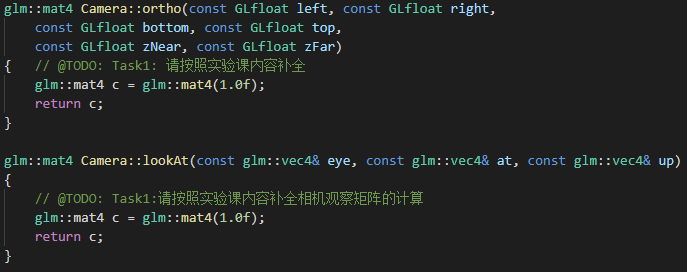
目前我们的TriMesh类只有顶点坐标等数据，但是没有记录物体的旋转、位移缩放的变量，这些也应该加进来。同时我们加上一个getModelMatrix计算物体的旋转位移矩阵。



相机实验的时候我们学习了如何计算相机的观察矩阵（viewModel）与投影矩阵（projectionModel）。为了更加方便使用这些矩阵，我们在Camera类里面再添加两个函数用于获取观测矩阵和投影矩阵。

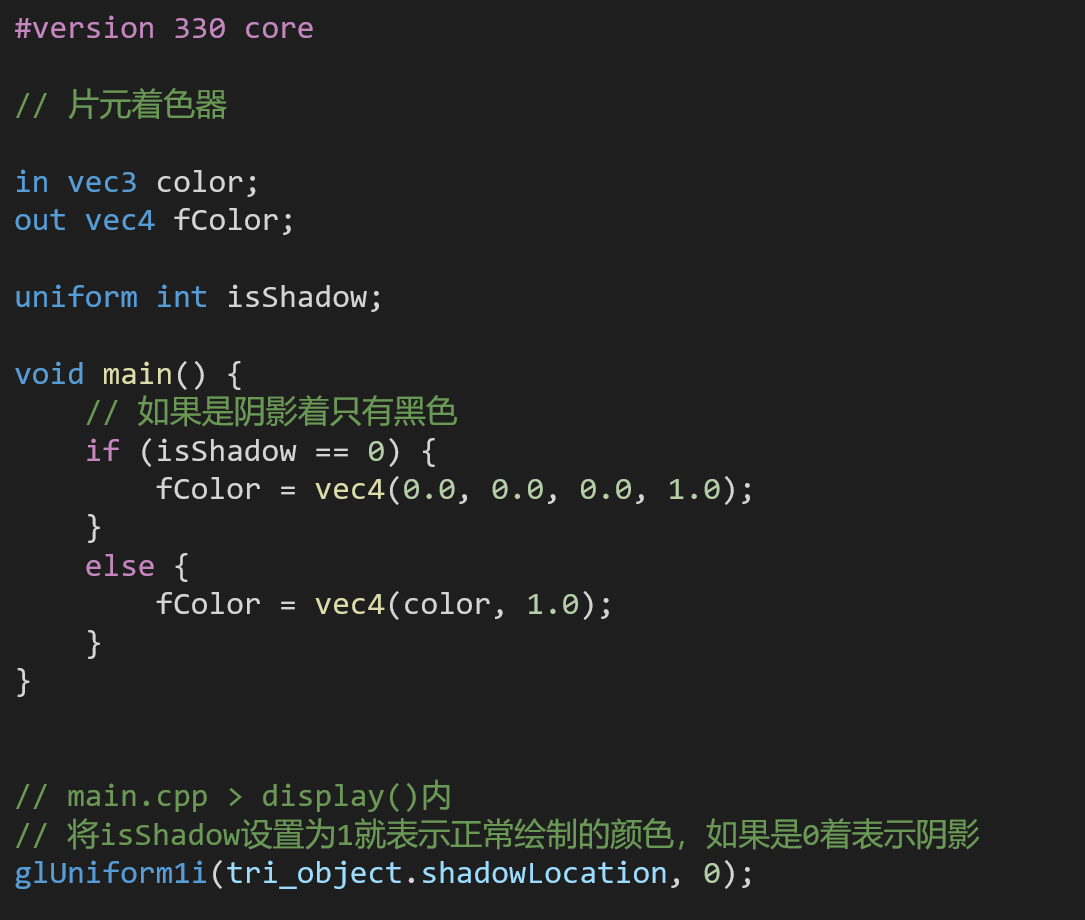


1. 实现lookAt与ortho函数。并将投影矩阵设置为正交投影。投影变换中实现的ortho函数将参数分别设置为左右裁剪平面-scale和+scale，上下裁剪平面-scale和+scale，远近裁剪平面zNear和zFar（请思考这些参数的意义是什么。）。将这些矩阵在程序中计算后传入顶点着色器中。



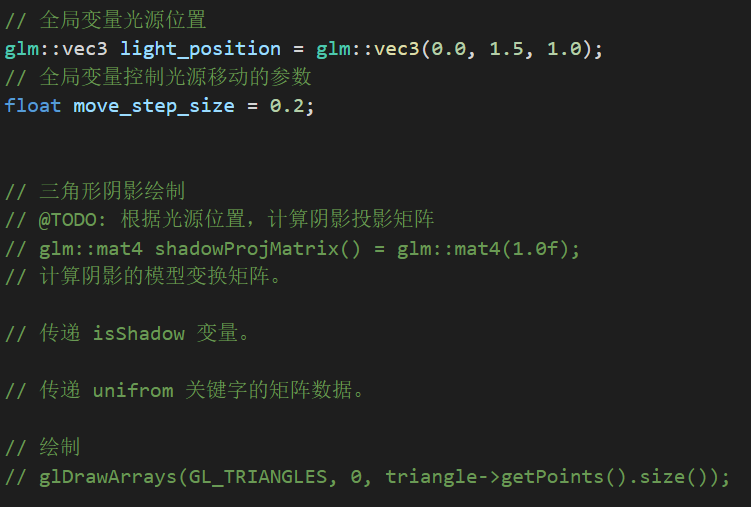
1. 计算阴影投影矩阵

我们定义一个全局变量light\_position表示光源位置，在参考程序中设置为(0.0,1.5,1.0)，继而参考上述理论说明部分计算阴影投影矩阵，更新模型视图矩阵，并将该矩阵传入到顶点着色器中，以执行对原始三角形顶点的投影变换。同时为了区分阴影和物体，我们加入一个isShadow变量判断用黑色还是物体的颜色。



1. 投影三角形的绘制

在绘制过程中，我们可以简单地将三角形绘制两次即可。第一次是按照常规方式绘制，第二次是使用了阴影变换矩阵之后对新的阴影三角形进行绘制。也就是说，第一次绘制时，将阴影投影矩阵设置为单位矩阵，而第二次绘制是计算出矩阵值之后变换三角形得到投影三角形。请参考代码中的display函数，这里需要将上述各部分进行整合，而得到最终的绘制结果。



1. 键盘控制相机和光源
2. 实验内容
3. 完成上述实验内容并实现对任意简单几何体的阴影变换。键盘控制光源与相机。在上述实验内容的基础上，可以尝试实现采用鼠标控制光源或者相机位置，并在鼠标点击之后计算阴影投影矩阵和绘制阴影。
4. 按照上述步骤完成之后会得到如下图所示的结果，通过键盘键能够观察旋转的三角形（红色）和投影生成的阴影（黑色）。

图片包含 图形用户界面

描述已自动生成