**计算机图形学实验报告**

学号：16340054

姓名：戴馨乐

学院：数据科学与计算机学院

作业：Homework7

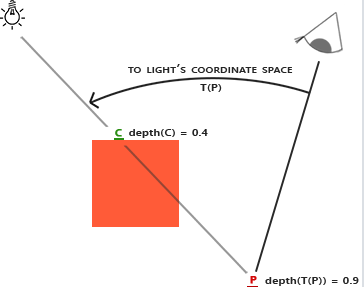
Basic：

1. 实现方向光源的Shadowing Mapping：
   * 要求场景中至少有一个object和一块平面(用于显示shadow)
   * 光源的投影方式任选其一即可
   * 在报告里结合代码，解释Shadowing Mapping算法
2. 修改GUI

**阴影映射 Shadow Mapping的思路：**

有一个光源和一些盒子，要从光源处沿着光线的方向能看到的就是亮的，看不到的就是在阴影处。那么找到光线到达的第一个物体，后面的自然都是在阴影下的。假如采用遍历很耗时，那么可以借用深度贴图：

平时透视图都是从观察者的角度出发的，这里从光源的透视图出发，来渲染这个场景，可以得到如下这种透视图：



从光源透视图的角度看出去，P点的深度大于C点，故P点应该在阴影中。用这种方式渲染出来的，就是深度贴图。

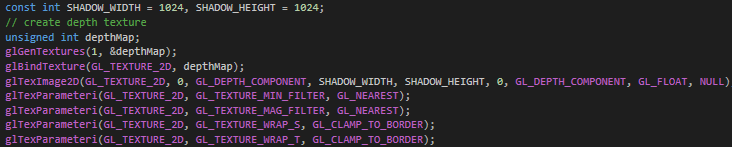
得到了深度贴图，将其存储为一个纹理，最后和绑定正常纹理那样，绑定到正常渲染的场景上就可以得到了带阴影的场景了。

**阴影映射 Shadow Mapping的详细步骤：**

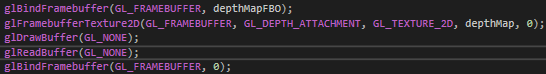
1. 生成深度贴图
   1. 深度贴图是从光的透视图出发，渲染的深度纹理，要将渲染的场景保存成纹理，需要用到帧缓冲：
      1. 生成帧缓冲对象



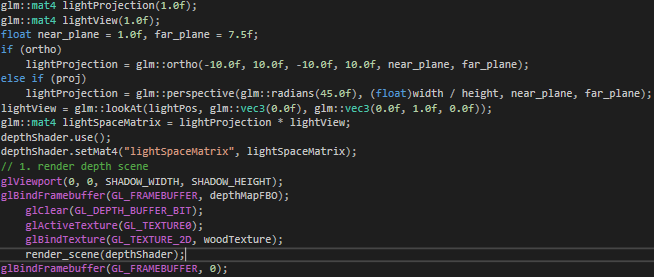
* + 1. 创建深度贴图的纹理对象，并设置相关的属性。其中，深度贴图的解析度不同于窗口大小，需要另外设置，这里设置解析度为

****

* + 1. 将深度贴图纹理作为帧缓冲的深度缓冲



这里，深度缓冲没有颜色缓冲，关注的只有深度缓冲，所以设置如下来告诉OpenGL这里不需要对任何颜色数据进行渲染

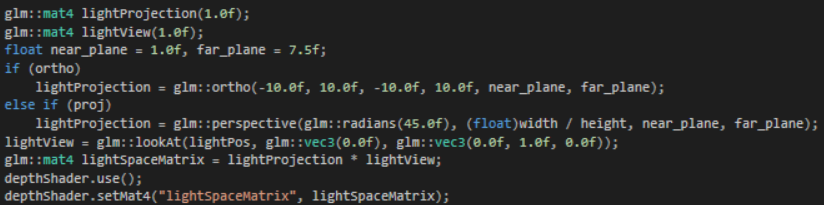
* 1. 在配置好帧缓冲之后，下一步需要在渲染循环中生成深度贴图。

可以看到，整个渲染过程分为：

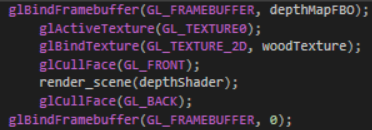
* + - 1. 改变视口大小并清理深度缓冲位



* + - 1. 因为要从光源的角度来看物体以生成深度缓冲，所以要使用从光源角度出发的透视矩阵和视角矩阵，最后可以得到将世界空间转换到光源空间的矩阵。这里默认使用的是正交投影。



* + - 1. 最后，绑定帧缓冲，然后渲染场景，就可以得到深度贴图纹理

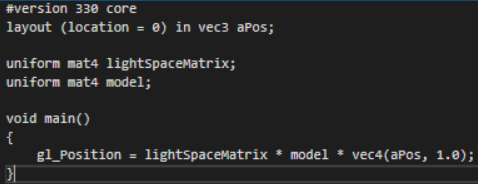


其中，生成深度贴图所用的Shader：

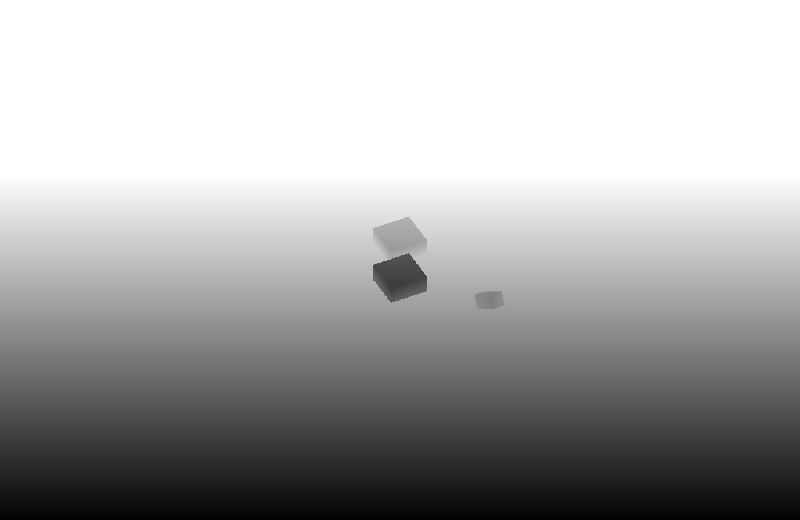
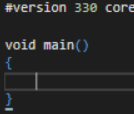


其中顶点着色器直接简单地将坐标转换空间，片段着色器由于没有需要颜色缓冲，所以什么事也不用做。

顶点着色器：



片段着色器：

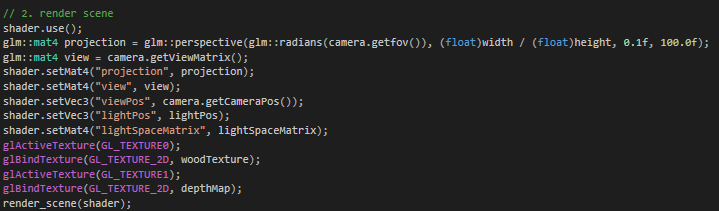
将深度贴图可视化之后：

可以看到，这里得到的就是从光源角度看下去的物体，只是没有了颜色缓冲仅仅有深度缓冲，这可以用来计算阴影。

1. 将深度贴图绑定到正常场景中

使用渲染场景的Shader来渲染：





这一段代码，就是简单地从摄像机的视角，来渲染物体，特别的是：



将深度贴图作为纹理绑定在场景上，相当于给原先的场景加上了阴影的效果。

要成功将阴影加上，还需要在顶点着色器和片段着色器上进行修改。

顶点着色负责：

1. 计算顶点的坐标



1. 为了计算光照，需要计算片段的位置，法向量



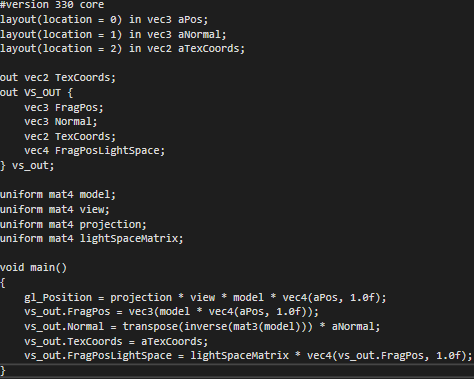
1. 为了加上纹理，需要传入纹理到片段着色器



1. 为了计算阴影，将片段的坐标进行光源空间的转换，因为深度贴图中阴影的计算就是在光源空间中的。



完整的代码如下：

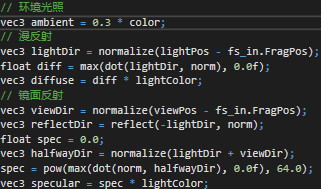


片段着色器负责：

1. 根据纹理计算物体的颜色



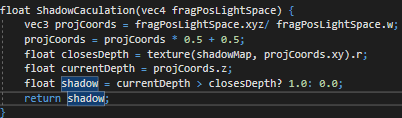
1. 计算光照



1. 计算阴影

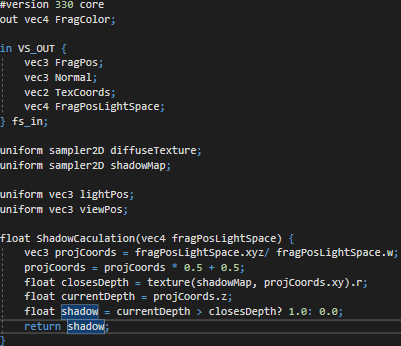
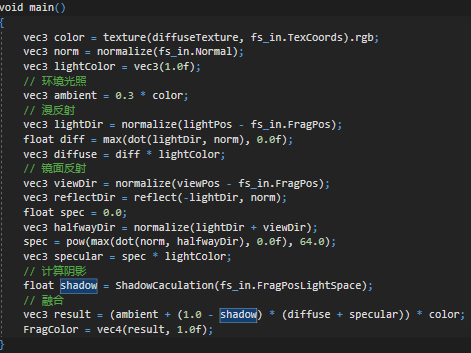


其中，阴影的计算想法很简单，就是计算光线所到达的最近深度以及当前这个点的深度，假如当前这个点的深度大，那么就是阴影，设为1.0；否则就是照亮的，为0.0。

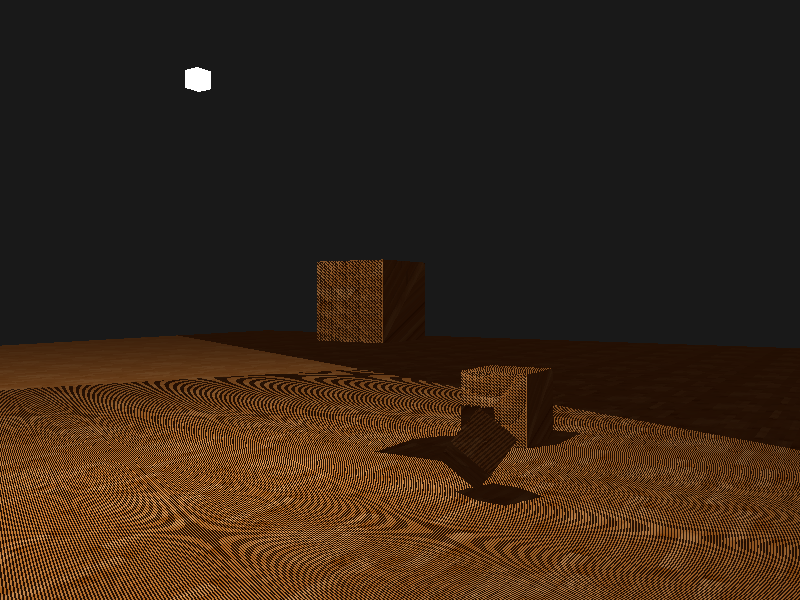


1. 综合光照和阴影得到最终的着色结果



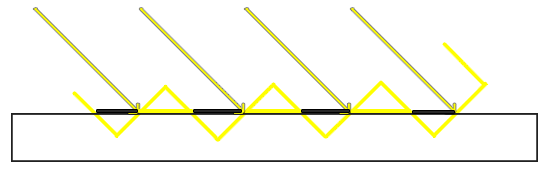
 完整代码如下：

运行结果：

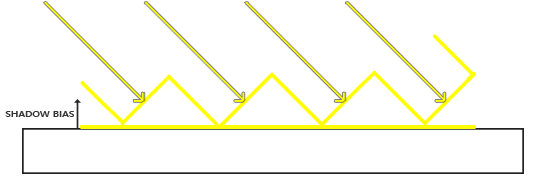


1. 解决阴影失真

这里，可以看到被光照亮的地方出现了一格格的阴影，很不真实。这是因为上述方法产生的深度贴图如下：



当光源远的时候，片元会从深度贴图的同一个深度值去采样，那么有些片元会被认为在平面下，有些认为在平面上，所以产生了参差交错的，不真实的阴影。解决方法很直观，将深度值整体上移就好了，如下：



那么，只需要在生成场景的shader的片段着色器生成阴影时候，增加一个偏移量bias，就可以解决了。为了这个偏移值可以适应不同角度的光线，所以使用光线的朝向的角度来改变：



结果如图：

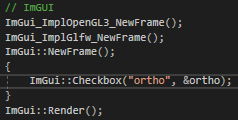
可以看到那种交错的不真实的阴影被去掉了。

**修改ImGUI：**

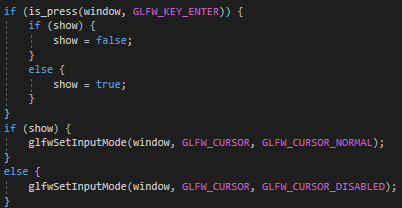
这里只需要调整是使用正交投影还是透视投影，所以只增加了一个Checkbox来选择



代码：



另外，由于为了更自由移动摄像头，所以隐藏了鼠标，这里需要调用出来选择，又需要隐藏起来移动摄像头，所以增加了可以修改鼠标状态的按键设置，这里设置为按下Space键，就可以显示或者隐藏鼠标。



Bonus：

1. 实现光源在正交/透视两种投影下的Shadowing Mapping

通过上面的ImGUI的设置来调整投影模式，如下：



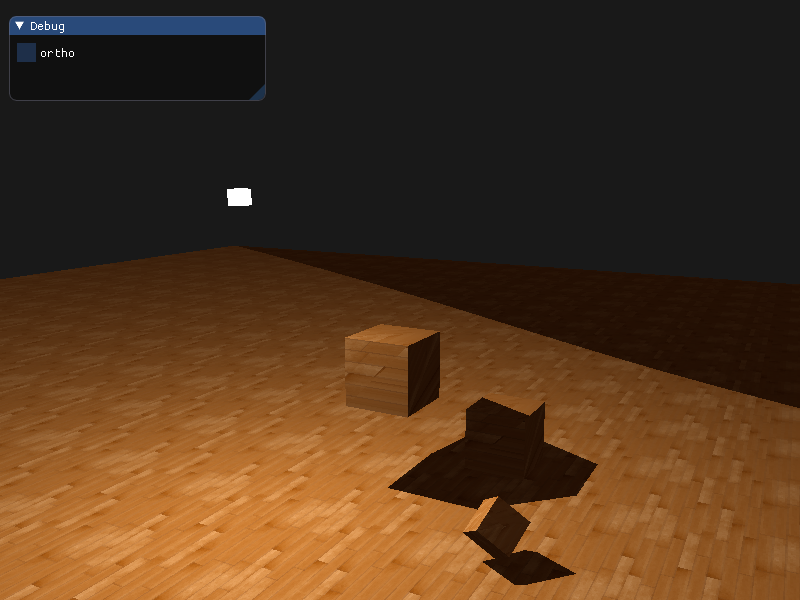
这样就可以修改了

对比如下：

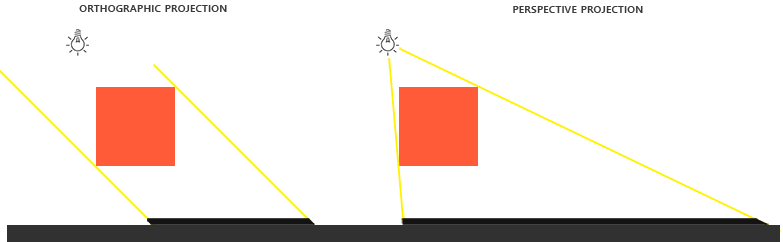
使用正交投影：



使用透视投影：



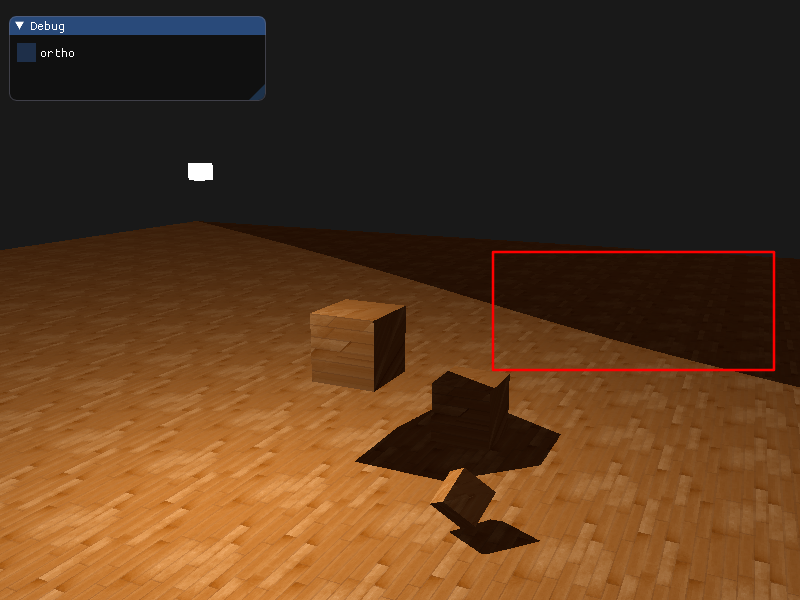
可以看到，正交投影的阴影面积比起透视投影要少。这是因为正交投影相当于平行光照射在物体上，而透视投影则是从光源出发的光线，对比如下：



从对比图看到，这和实验结果是相符的。

1. 优化Shadowing Mapping
   1. 解决采样过多问题

从上面实验图看到：



出现了一大块的阴影面积。而我们本来是有环境光照的，所以这些地方其实是对阴影采样过多了，超出光的范围一律视为处在阴影中，不管是不是真的。这个原因是深度贴图的环绕方式是GL\_REPEAT。解决办法是让超出深度贴图的坐标的深度范围都是1.0，这样它们就不会是被判断在阴影下了。

1. 设置深度贴图纹理：将模式设置为GL\_CLAMP\_TO\_BORDER，然后定义边框的深度borderColor，最后将这个设置为纹理的边缘的深度。



1. 修改片段着色器的阴影计算，将深度大于1.0的点，阴影都设置为没有阴影，即shadow值为0.0



效果：



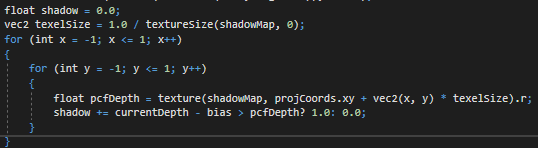
可以看到，远处的，不真实的阴影都被消除了。

* 1. 缓解阴影锯齿化的问题

由于图片是像素化的，而且深度贴图的解析度是固定的，那么必定会出现锯齿化，显得不真实。



虽然增加解析度可以缓解这个问题，但是通过另外一种方法PCF来解决会更加灵活。该方法是从深度贴图中多次采样，然后融合在一起，结果进行平均化，从而得到了柔和的阴影。这次是从一个的范围中进行采样融合。如下：



效果：



可能设置范围不够，虽然还有锯齿化，但是可以看到是有对周围进行采样融合，得到了柔和的阴影。