真实感渲染框架报告

软博21 沈冠霖 2021312593

1.基本情况

1.1 开发环境

- Windows 10 系统
- Visual Studio 2022
- C++语言，Win32开发框架
- OpenCV 4.5.5
- Eigen 3.4.0

1.2 运行方式

直接使用Visual Studio编译运行src中的项目解决方案，或者运行bin文件夹中的.exe文件即可。

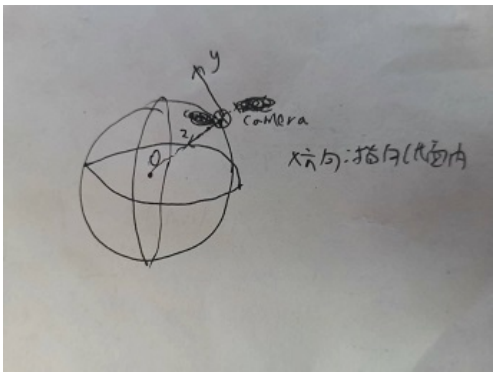
在场景中，使用键盘前后左右键进行相机平移，使用鼠标上下左右拖动进行相机旋转，使用鼠标滚轮进行视角缩放。注意视角缩放和平移一定要一次只滚动滚轮/点击键盘一次，而且场景漫游每次需要等待一秒左右等待场景刷新完毕。

也可以查看doc文件夹里的演示视频。

2.相机模型

我实现了透视投影，并且通过将相机限制在一个球上的方法实现了包括平移、旋转、缩放的场景漫游，具体细节如下。

首先是相机坐标系和世界坐标系的坐标转换，也就是相机外参的设置。我设定相机位于一个球的上半部分，相机朝向（也就是相机坐标系下z轴正方向）为指向球心的射线，相机坐标系下x轴，y轴正方向分别为逆时针方向，斜向上方向，示意图如下：



假设球心坐标为$O=(O_x,O_y,O_z)$，球的极坐标表示为(R,θ,ϕ)(其中θ为0到π之间)。

则有相机坐标系下，球心（原点）坐标为(O_x,O_y,O_z)，相机坐标为$(Rcos\theta cos\phi+O_x,Rsin\theta+O_y,Rcos\theta sin\phi+O_z)$。

x轴的坐标为$(-sin\phi,0,cos\phi)$，y轴的坐标为$(-sin\theta cos\phi,cos\theta,-sin\theta sin\phi)$，$z$轴的坐标为$(-cos\theta cos\phi,-sin\phi,-cos\theta sin\phi)$。

综上所述，从相机坐标系到世界坐标系的变换矩阵为 $\begin{bmatrix}-sin\phi & -sin\theta cos\phi & -cos\theta cos\phi & Rcos\theta cos\phi+O_x\\0 & cos\theta & -sin\theta & Rsin\theta+O_y\\cos\phi & -sin\theta sin\phi & -cos\theta sin\phi & Rcos\theta sin\phi+O_z\\0 & 0 & 0 & 1\end{bmatrix}$

这样，也能便于实现包括平移、旋转、缩放的场景漫游。我设定平移是改变球心坐标O的位置，旋转是修改θ（上下旋转）和ϕ（左右旋转），缩放是修改球半

我通过绑定$KeyDown$，$KeyUp$等键盘事件实现了平移，$MouseWheel$这个鼠标滚轮事件实现了鼠标滚轮缩放，

通过绑定$MouseMove$，$ButtonDown$，$ButtonUp$等鼠标事件实现了鼠标拖动旋转。

之后是像素坐标到相机坐标的变换，也就是相机内参。我设定$S=W=H=300$，$f_x=f_y=u_0=v_0=S/2=150$。

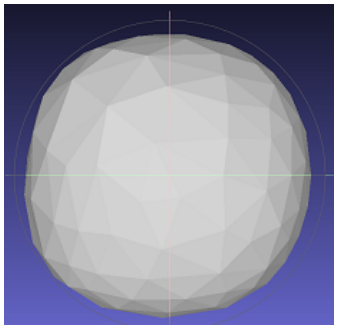
根据透视投影模型，有$\frac{u-u_0}{f_x}=\frac{x}{z},\frac{v-v_0}{f_y}=\frac{y}{z}$。

因此，对于像素坐标为(u,v)的射线Ray，我定义其在相机坐标下的起点为$(0,0,0)$，方向为$(\frac{u-u_0}{f_x},\frac{v-v_0}{f_y},1)$。

这样，再利用相机外参就能得到对应的世界坐标下的起点位置和方向，可以与世界坐标系下的物体进行求交，求局部光照、反射、透射等操作了。

3.物体模型

我实现了.ply和.obj文件的读取，以及.obj文件的纹理贴图。我的场景中一共有四个物体模型，包括一个有纹理的小动物.obj模型（shiba.obj），一个无纹理的黄色斯坦福兔子模型（bunny.ply），一个无纹理的半透明浅灰色球体(cube.ply)，一个无纹理的反光地板模型（board.ply)。我使用双线性插值算法进行纹理贴图。几个模型示意图如下：



为了方便后面光线跟踪算法的求交，我使用包围盒+八叉树的方法组织每个物体模型。对于每个物体模型，我首先求出其包围盒，并且在包围盒内部建立八叉树，让八叉树每个节点存储其x，y，z坐标上下界及其内部的面片（只要面片有一个点在八叉树节点内部就算）。由于这种方法会导致包围盒角落的面片遗漏，我限制了八叉树的深度和其每个节点存储面片的最小值。

在求交过程中，我让每条光线先和物体的包围盒求交，然后再递归进入子节点，和每个子节点的包围盒相交，如果交集不为空，则在这个子节点递归求交。如果是叶子节点，就和里面的所有面片求交。和单个面片求交算法我使用Moller-Trumbore算法实现。

4.光照模型

我实现了Phong模型和光线跟踪算法。对于光照模型，我设定在场景的y轴正上方有均匀的，来自无穷远处的平行光，以简化模型，具体公式如下：

$$I = I_a K_a + I_p K_d (L \cdot N) + I_p K_s (R \cdot V)^n$$

其中I_a为环境光颜色$Ambient$，K_a为物体每个面片的$Ambient$属性。

K_d为物体每个面片的$Diffuse$属性，K_s为物体每个面片的$Specular$属性。

L为面片中心到光源的向量，因为光源在y轴无穷远，$L = (0, 1, 0)$。

N为面片法向量，R为反射光线向量，$R = 2N(N \cdot L) - L$。

V为视线方向，也就是相机到视点中心的单位向量。

我对于每个点分别求解，然后使用$Gouraud$插值求得交点的颜色。

对于光线跟踪，我首先利用相机模型，对每个要求解的像素生成W*H条光线，并将其起点和方向转换到世界坐标系下，然后逐条光线进行光线跟踪。我将光线分为四类：起始光线，阴影光线，反射光线，折射光线，这四类的处理方法各有不同。

对于起始光线，反射光线和折射光线，需要递归处理，我的处理流程如下：

```
Color RayTracing(ray, depth, weight)
{
    if(depth > MaxDepth || weight < MinWeight)
    {
        return black; //递归终止准则
    }

    if(ray.type == INIT) //起始光线
    {
        intersection = GetIntersection(exclude = NULL); //和所有模型都要求交
    }
    else
    {
        intersection = GetIntersection(exclude = last); //其他两种光线，不考虑上次其相交的模型
    }
    if(intersection == NULL)
    {
        return black;
    }
    shadow = GetShadow(ray, intersection, depth, weight); //生成竖直向上的阴影光线
```

```
    reflection = GetReflection(ray, intersection); //生成反射光线
    refraction = GetRefraction(ray, intersection); //生成透射光线
    I_l = PhongModel(ray, intersection); //局部光强
    I_shadow = JudgeShadow(intersection, shadow, depth, weight); //判断是否在阴影
    I_s = RayTracing(intersection, reflection, depth + 1, weight * K_s); //递归求反射光强
    I_t = RayTracing(intersection, refraction, depth + 1, weight * K_t);// 递归求透射光强
    return I_l * I_shadow + K_sI_s + K_tI_t;
}
```

对于阴影光线，不需要递归，只需要求其传播方向上是否有其他物体即可。如果有不透明的物体，则说明之前的交点位于阴影中，需要设置颜色为黑色。如果有半透明物体，则需要减弱之前交点的光照强度。如果只有透明物体或者没有任何物体，可以直接按照phong模型求出这个交点的光照强度。

这张图片可以近似说明我的光线跟踪算法和光照模型的正确性。

首先，从光照模型上来看，由于光线都来自正上方，因此小动物的下方偏暗，上方偏亮，而且小动物颜色也和纹理贴图的颜色类似，说明光照模型正确。

其次，可以看出，小动物、兔子这两个实心物体下方的地板都有对应的黑色阴影，而半透明的球下方的地板有对应的深灰色阴影，说明阴影部分实现正确。

其次，地板上能看到小动物和球的倒影，说明反射部分实现正确。

最终，实心的小动物能够遮挡兔子的一部分（兔子也是实心的，变换视角也可以看到它遮挡其他物体），而半透明的球只会让其后面小动物变暗，不会完全遮挡，说明透射部分实现正确。

5.总结

这次我基本实现了相机模型，物体模型，光照模型和光线跟踪的相应算法，但是完成的并不够完美。首先，由于要实现场景漫游，我需要加快渲染的速度，这就导致我不能使用太过于复杂的物体模型，整个场景的面片数量只有一千多，也影响了场景的效果。其次，由于ply和obj文件的复杂性和各种可能，我的读取算法并不能读取任意种类的ply和obj模型，比如我只能支持三角面片的相关模型，要求模型必须提供法向，并且obj的材质方面仅仅支持物体的ambient, diffuse和specular三种设置和纹理。最终，我使用的也仅仅是简单的phong模型和光线跟踪算法，在真实感方面不如BRDF等更好的模型和算法。要想继续改进，需要在数据结构、求交算法上进一步改进，并且引入cuda等并行加速算法提高运算速度，以及使用更复杂多样的物体模型和更具有真实感的算法。