学号	姓名	论文规范性（10）	问题分析与调研（30）	方案创新性（20）	实验结果分析与讨论（40）	结课论文总成绩（100）
21301108	文泽	6	21	17	35	79

存在格式问题；缺少对现有工作的梳理

北京交通大學

本科课程（论文）

简易的光线追踪

学　院：软件学院

专　业：软件工程

学生姓名：文泽

学　号：21301108

中文摘要

摘要：光线追踪算法由 Appel 在 1968 年提出，是一种基于真实光线传播模拟的计算机三维图形渲染算法。能实现较真实的光影效果。但是由于其庞大的计算量，一般用于离线渲染中。随着硬件技术的提升，已经推出支持实时光线追踪渲染的 GPU，如微软 DXR 和 NVIDIA RTX。光线追踪可简单分为正向光线追踪和反向光线追踪。正向光线追踪算法从光源位置跟踪光子穿过场景的路径。反向光线追踪则从观察者视角出发向场景发出光线，追踪光线的路径。

本次光线追踪实验，主要通过定义一系列物品、光、场景等结构，最终实现一系列功能函数，从而完成渲染。研究内容如下：(1)物质类实现。(2)摄像机，场景类实现。(3)光线与相交检测。(4)光线追踪 Trace。

1 引言

1.1 研究背景及意义

光线追踪，从算法名称上来考虑简单来说就是从视点开始发射一条射线通过视平面上的每一个像素点并不断进行光线与物体交叉判定，同时考虑反射折射等光学现象来渲染三维场景。算法的终止条件主要由是否碰撞到物体，衰减，追踪的深度决定的。因此单纯从算法思想上来讲并不复杂，但是细节上其实仍然有许多点值得留意学习。

本文旨在通过定义一系列物品、光、场景等结构，实现相关的功能函数，最终实现一个简易的光线追踪程序，并根据实现的程序做出一些分析。

1.2 论文结构安排

第一章：引言。该章节首先介绍了光线追踪研究背景与意义，最后介绍了本文的主要内容以及组织结构。

第二章：相关工作介绍及方法描述。该章节对实现过程中所涉及的关键技术进行了分析，包括材质定义，光线及相交点求交过程，场景与摄像机实现，光线追踪核心技术。

第三章：实验设置、实验结果与分析。涵盖一些实现的源代码介绍，及其对实验结果的展示。

第四章：结论。总结本次实验的收获与感悟。

2 相关工作介绍及方法描述

2.1 材质定义

2.1.1 材质分类

定义一个基类 Material，表示物体的材质。定义枚举类型将物体大致区分为三种类别：粗糙物体，反射面物体，折射面物体。

2.1.2 粗糙材质

本文实现的粗糙材质采用 Phong 模型求解，Phong 模型是针对局部光照的经验模型。在粗糙材质中，ka 表示环境光系数，kd 表示漫反射系数，ks 表示镜面反射系数。

```
struct RoughMaterial : Material
{
    RoughMaterial(vec3 _kd, vec3 _ks, float _shininess) : Material(t:ROUGH)
    {
        ka = _kd * M_PI;
        kd = _kd;
        ks = _ks;
        shininess = _shininess;
    }
};
```

2.1.3 反射材质

本文实现的反射材质应用了 Fresnel 公式，主要包含两个参数：n 表示折射率，kappa 表示物质的消光系数（光射入物体后的衰弱的快慢）。

```
struct ReflectiveMaterial : Material
{
    // n: 折射率
    // kappa：消光系数
    ReflectiveMaterial(vec3 n, vec3 kappa) : Material(t:REFLECTIVE)
    {
        vec3 one(x0:1, y0:1, z0:1);
        F0 = ((n - one) * (n - one) + kappa * kappa) / ((n + one) * (n + one) + kappa * kappa);
    }
};
```

2.1.4 折射材质

本文实现的折射材质同样应用了 Fresnel 公式，区别在于由于物体是具有折射性质的，说明光不会在物体内部衰弱，因此只包含了一个参数：n 表示折射率。

```
struct RefractiveMaterial : Material
{
    // n, 折射率
    RefractiveMaterial(vec3 n) : Material(t:REFRACTIVE)
    {
        vec3 one(x0:1, y0:1, z0:1);
        F0 = ((n - one) * (n - one)) / ((n + one) * (n + one));
        ior = n.x;
    }
};
```

2.2 光源，射线设计

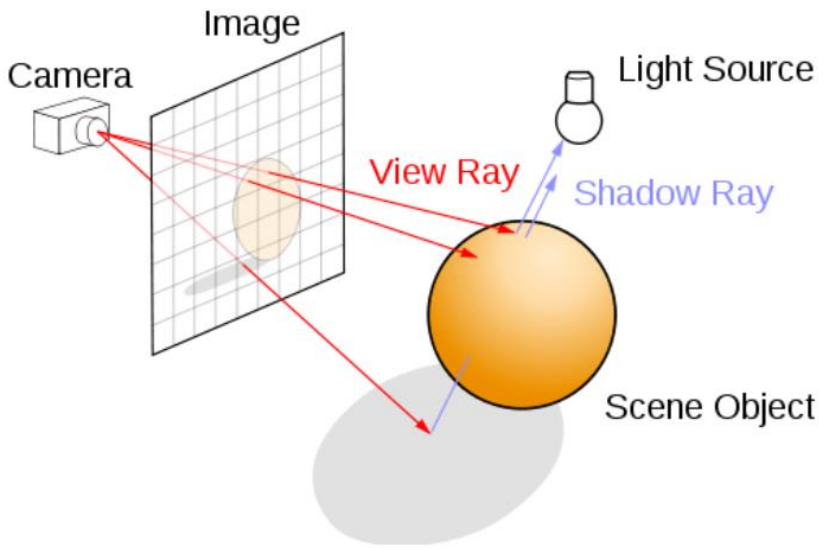
2.2.1 光源设计

光源主要包含两个参数：direction 表示光源的方向，le 表示光照的强度。

```
struct Light
{ // 定义光源
    vec3 direction;
    vec3 Le; // 光照强度
    Light(vec3 _direction, vec3 _Le)
    {
        direction = _direction;
        Le = _Le;
    }
};
```

2.2.2 射线

场景(Scene)中有一个摄像机，多个光源。追踪“光线”，实际上是从摄像机出发，穿过视窗上某一像素点的一条射线。追踪射线，检测与场景中所有物体是否有交。如果射线与场景中的物体均没有交点，场景渲染为环境光；如果有交点，追踪该射线并为交点位置着色。



光线追踪原理图

2.3 求交过程

2.3.1 光线与相交检测

光线定义为从某一原点 o，某一方向 d，延申出的一条直线。可以得出光线方程： p = o + t d ，t 可位于无限区间内，表示光线无限延申。

光线与场景中几何体的相交检测，可以看成在一个 3 维空间，一条直线与一个三维几何体是否相交的问题。

2.3.2 球体求交

联立直线方程与球方程：(1) p = o + t d (2) $|PP_0|$=R ；其中 P0 是球心坐标，R 为半径。(2)式两边平方，带入(1)式可得(3)，求解判别式 Δ 的正负即可知道光线与球体是否有交点。

（1）p = o + t d

（2）$|PP_0|$=R

（3）$(o^2 + 2 * o * t * d + t^2 * d^2) * p_0^2 = R^2$

2.3.3 平面求交

平面采用向量的表示方法（一个点与一条法线确定一个平面）可得平面方程为：$N(P - P_0) = 0$。

将直线方程：$P = o + td$ 代入后，求解 t 的表达式，可得：

$$t = \frac{N \cdot P_0 - N \cdot o}{N \cdot d}$$

当分母点乘为 0 时，表示平面与射线平行，此时不可能有交点；当求解的 t<0 时，表示交点位于射线起点后方，即摄像机后方，可排除；只有当 t>0 时，交点有效。

2.3.4 圆柱体求交

联立直线与侧面方程，求得可能的交点t_1、t_2；联立直线与上、下底面方程，求得可能的交点t_3、t_4，将可能的交点加入初始数组。若初始数组非空，则选择距离摄像机最近（非负且最小）的 t 作为交点；若初始数组为空，则没有交点。

2.4 摄像机，场景搭建

2.4.1 摄像机

场景中只有一台摄像机，用于模拟用户的视角。定义摄像机的位置，以及视窗的大小。同时不断修改摄像机的位置，模拟环绕，这样可以更好的展示实验的效果。

```
class Camera
{                                   // 用相机表示用户视线
    vec3 eye, lookat, right, up; // eye用来定义用户位置；lookat(视线中心)，right和up共同定义了视窗大小
    float fov;
```

```
    void Animate(float dt)
    {
        // 修改摄像机的位置（环绕）
        vec3 d = eye - lookat;
        eye = vec3( x0: d.x * cos(dt) + d.z * sin(dt), d.y, z0: -d.x * sin(dt) + d.z * cos(dt)) + lookat;
        set(eye, lookat, up, fov);
    }
};
```

2.4.2 场景搭建

在场景中，引入事先定义好的物体，光源，摄像机。物体包含：粗糙的球体，可反射与折射球体，粗糙圆柱体，可反射平面。渲染视窗上每个点的着色；调用摄像机的 Animate 函数，让视窗进行环绕。

2.5 光线追踪

2.5.1 捕捉漫反射

利用 Phong 模型计算视线与粗糙材质交点处的着色。漫反射表面粗糙，到各个方向反射的光线强度都相等。入射方向和表面存在一定的夹角，因此漫反射光强：

$$I_d = K_d * L_d * （\boldsymbol{l} \cdot \boldsymbol{n}）$$

在求解时，还需要求该射线与其他物体是否有交点，有交说明光源被遮挡，那么在用户看来就是阴影，其着色用环境光赋予即可。

2.5.2 镜面反射

反射材质与折射材质均会发生镜面反射（光线到达物体表面后，会分为反射光线和折射光线），对于反射光线，我们需要继续进行追踪，直到该光线与粗糙材质相交，或者到达预设的迭代深度（本文预设的迭代上限为 5）。

```
// 镜面反射
float cosa = -dot(ray.dir, hit.normal);
vec3 one(x0:1, y0:1, z0:1);
vec3 F = hit.material->F0 + (one - hit.material->F0) * pow(x:1 - cosa, y:5);
vec3 reflectedDir = ray.dir - hit.normal * dot(hit.normal, ray.dir) * 2.0f; // 反射光线R = v + 2Ncosa
vec3 outRadiance = trace(Ray(start:hit.position + hit.normal * epsilon, reflectedDir), depth + 1) * F;
```

2.5.3 折射光线

对于折射材质物体，我们根据其折射率，光线方向计算折射的效果。然后对计算结果进行处理，为折射物体表面的像素点进行着色。

3 实验设置及结果分析

3.1.1 实验设置

设置对应的类，功能函数，驱动 OpenGL 的渲染器，将设置好的场景渲染出来。过程大致划分为：

（1）定义工具类，用于实现初始化顶点着色器与片段着色器，其构造函数中生成了 VertexArray 与 Buffer 缓冲。

（2）调用初始化函数 onInitialization：设定好视窗、初始化场景、初始化着色器，并创建 gpu 进程。

（3）调用场景类 Scene 的 Render 函数，对每一个像素进行求解,根据结果获知需要用什么颜色给像素点赋值。

（4）利用绘图函数 Draw：将 Render 的求取结果绘制出来。

（5）显示函数 Display：由于视窗在进行环绕，所以需要显示函数不断更新当前最新的图像，并持续调用绘图函数将其绘制出来。

3.1.2 实验结果

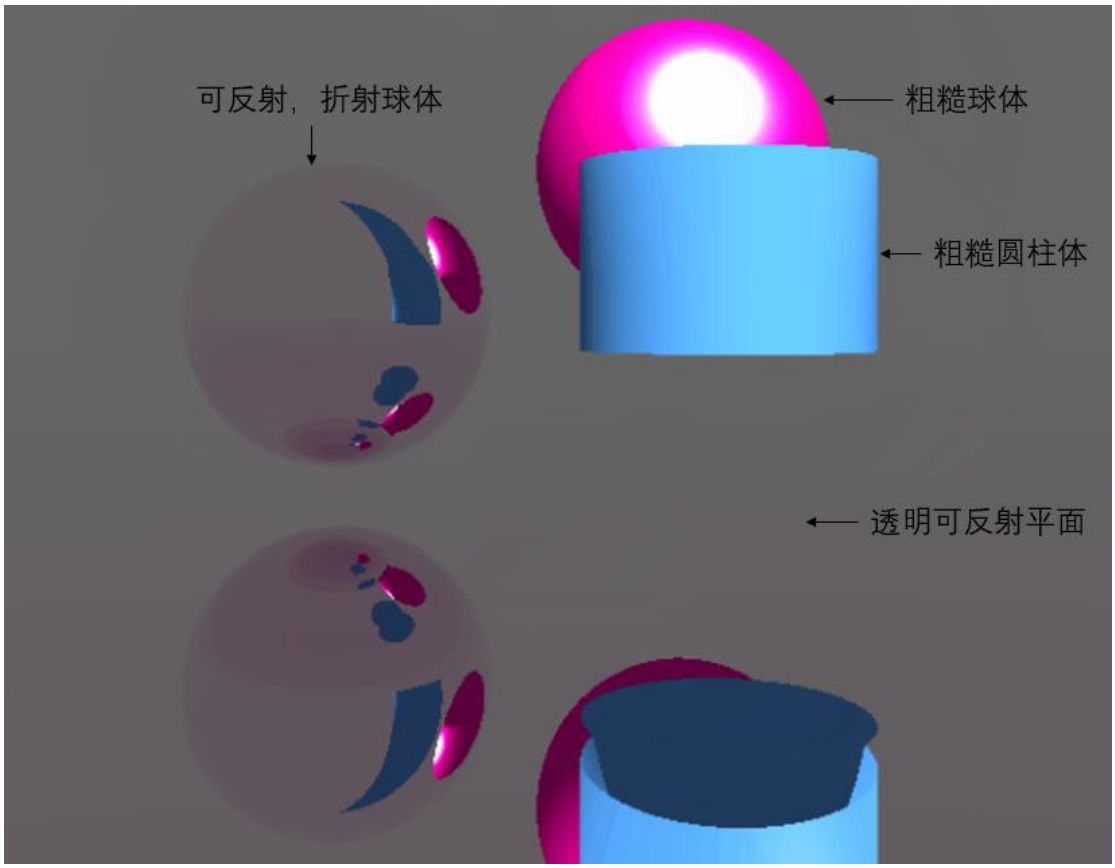


图 1 ：实验效果一

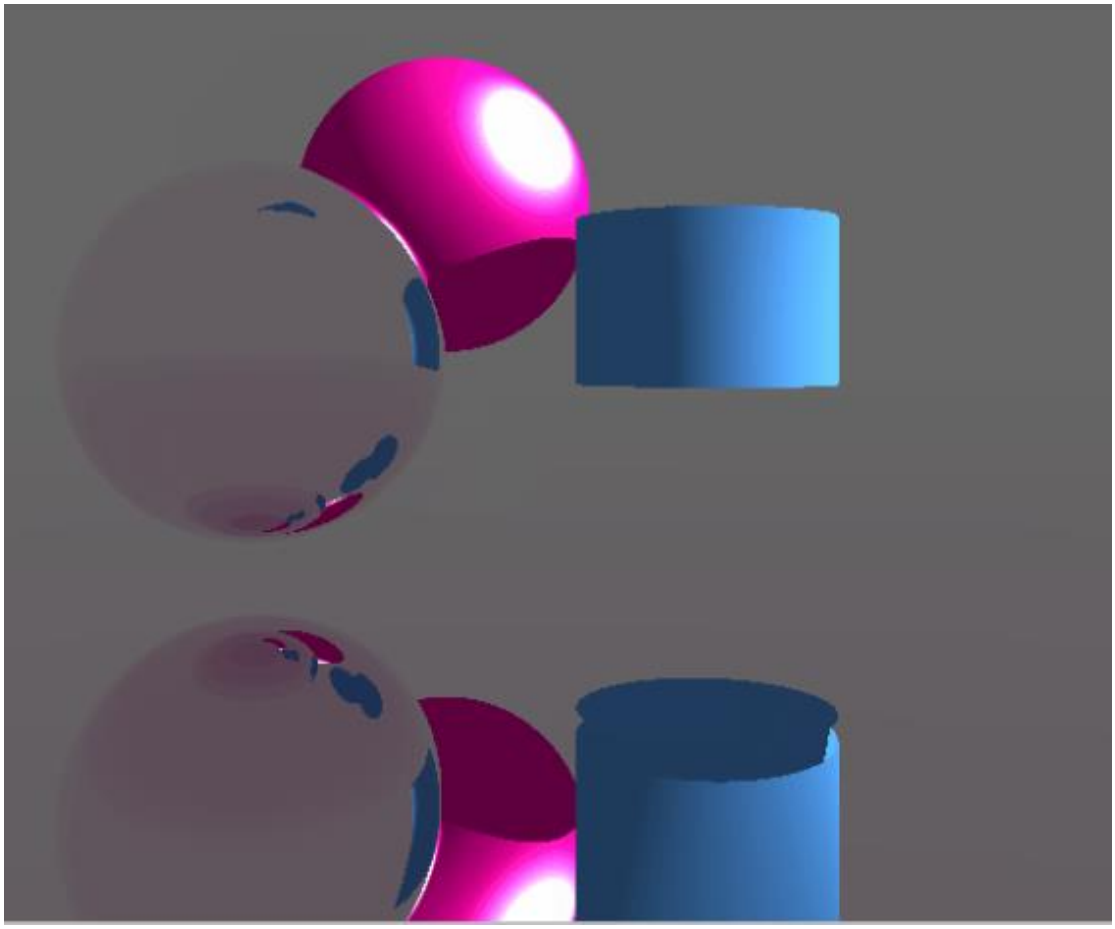


图 2 ：实验效果二

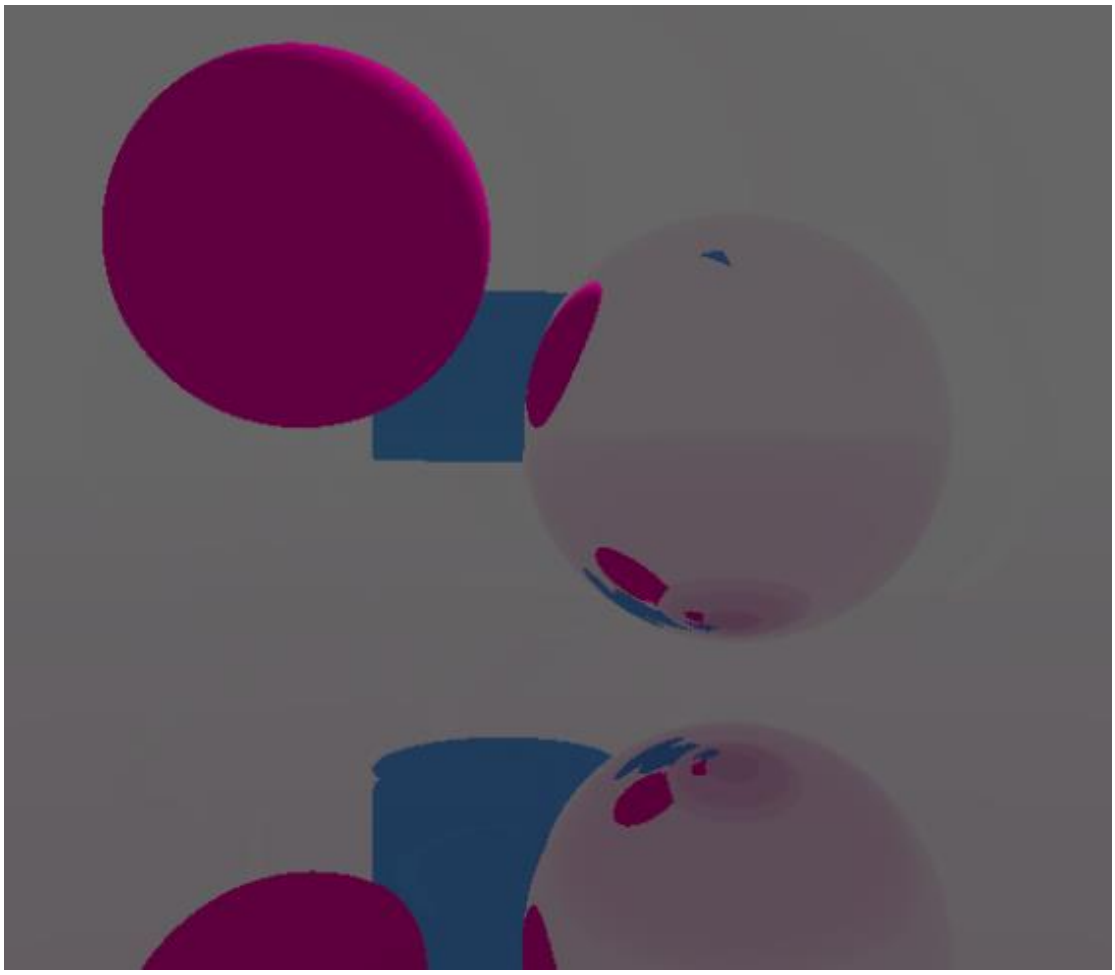


图 3：实验效果三

3.1.3 实验结果分析

场景中：透明可反射平面的效果很好，比较符合实际的镜面反射效果。粗糙圆柱体的模型创建和漫反射效果较差，与粗糙球体相比，并不能反映出点光源的位置。

4 总结

本次实验基于 OpenGL 库，实现了一个简易的光线追踪算法。该算法能够实现简单的漫反射，反射，折射光线的计算，并完成对像素点渲染着色，但效果并不是很好，需要后续不断地进行完善。

在实验过程中，掌握了模型创建，交点计算，光线设计，着色等知识，对计算机图形学的认识更进了一步。

参考资料

[1] GAMES101,闫令琪, 现代计算机图形学入门

[2] 极简光线追踪入门（CSDN-转载）， 音视频开发进阶

致　谢

感谢吴雨婷老师的教导!