计算机图形学

基于 C++实现基础光线追踪

姓名：白雪

学号：18051201

专业：计算机科学与技术

指导老师：陈小雕

自评：87.5

杭州电子科技大学

2020.5——2020.6

目录

一、需求分析

1. 光线追踪简介

在自然界中，光源发出的光线会不断地向前传播，直到遇到一个妨碍它继续传播的物体表面——把“光线”看作在一串同样路径中传输的光子流的话，在完全的真空中，这条光线将是一条标准的直线。但是实际上，由于大气折射、引力效应、材质反射等多种因素——在现实中，光子流实际上是会被吸收、反射和折射的——物体表面可能在一个或者多个方向全部或部分的光线，并有可能吸收部分光线，使得最终光线以种种形式，不同的强度，反射或者折射进入人的眼睛。而物理学中的光线追踪，指的就是一种通过种种方法对光线进行追踪的方法。不过，这一点在计算机图形学中却有所不同——作为三维计算机图形学中的特殊渲染算法，光线追踪的原理颇有把物理中“光线追踪”方法反过来用的意味——它通过将光的路径跟踪为图像平面中的像素并模拟其与虚拟对象的相遇来生成图像，从而产生高度拟真的光影效果，还可以轻松模拟各种光学效果（例如反射和折射，散射和色散现象（例如色差））——唯一的缺点，就是它相对较高的计算成本。

光线追踪广泛用于电影和电视节目所制作的计算机图形图像，但这是由于工作室可以利用整个服务器群（或云计算）的力量。即便如此，这可能是一个漫长而艰辛的过程，对于现有的游戏硬件来说，实时执行光线追踪的要求显得十分过分。相反，游戏主要是利用光栅化，一种渲染计算机图形的更快速方法。他能够把 3D 图形转换为 2D 像素以显示在屏幕上，但光栅话需要着色器描绘合理逼真的照明效果，最终的结果不如光线追踪那么自然或逼真。换句话说，光线追踪将

能带来光栅化所不具备的优点，但不会完全取代后者。不过，光线追踪将成为游戏开发者的又一项工具，而随着时间的推移它的重要性将日渐凸显。

微软表示，光线追踪将在“未来数年”获得更多光栅化所不具备的优点，包括全局照明。微软指出：“最终，光线最终可能完全取代光栅化，并作为渲染 3D 场景的标准算法。”

2. 开发意义

自从 2018 年 NVIDIA 推出 RTX“光线追踪”技术之后，游戏行业中对画面表现的追求就上升到一个新的高度。这项被誉为是 NVIDIA 在计算机图形学算法和 GPU 架构领域经过 10 年努力所取得的重要成果，可以让游戏画面的光影表现更加自然，呈现电影级画质的实时渲染。

在 Gamescom 2019 上，我们看到大量公布的新游戏将会支持光线追踪技术。同时英伟达也鼓励开发人员为自己的游戏中加入这项技术。比如《我的世界》就将拥有完全的路径跟踪照明功能，作为一款大热的沙盒游戏，光线追踪技术加入之后，可以说游戏的面貌彻底发生了改变。此外，已经确定支持光线追踪的游戏名单越来越长，无论未来如何，我们都可以确定会有更多的游戏开发商加入到光线追踪的阵营中来，可以说，让光线渲染更逼真和准确的光线追踪技术已经是下一代游戏画面的标准，光追技术已将成为游戏开发领域的一项基本技术。

实时光线追踪不仅能为游戏界带来一场变革，在其他领域也能发挥重要的作用。在建筑的 3D 效果图方面，在产品设计方面，在教学方面，光追技术都能够有所建树。由于有着广泛的应用前景，尤其是在建模渲染方面，光线追踪技术被视作 AEC 行业（建筑、工程以及施工行业）的“重要突破“。

3. 系统功能

本系统的主要功能包括：主要三维几何体的平移旋转，三种主要材质渲染，三种主要光源渲染，实时光线反射效果实现。

二、概要设计

代码上传至 GitHub： *https://github.com/baibaixue/my_raytracer*

系统文件主要结构：

命名空间 namespace：rt

第三方库：GLFW——用于创建 OpenGL 上下文，以及操作窗口。

App：基于 GLFW 对操作窗口函数进行封装。

Base：基本功能类：

主要含：Vector3——三维空间向量；

Color——ARGB 颜色；

Mathf——基本数学函数库。

Camera：Camera 类及其派生类 PerspectiveCamera 类——相机及透视相机。

Geometry：Geometry 类及其派生类——基本几何体实现；

Union 类——几何体集合。

Light：LightSample 类及其派生类——基本光源实现；

LightUnion 类——光源集合。

Material：Material 类及其派生类——基本材质实现；

Ray：Ray 类——光线类；

IntersectResult 类——光线追踪结果实现。

其他： initialize_scene——初始化场景物体及光源，用于测试；

Mainloop()——光栅化填充窗口；

Render——材质渲染及光环境渲染，光线反射；

交互函数——光标控制物体旋转，方向键及鼠标滚轮控制相机移动。

1. 系统架构图

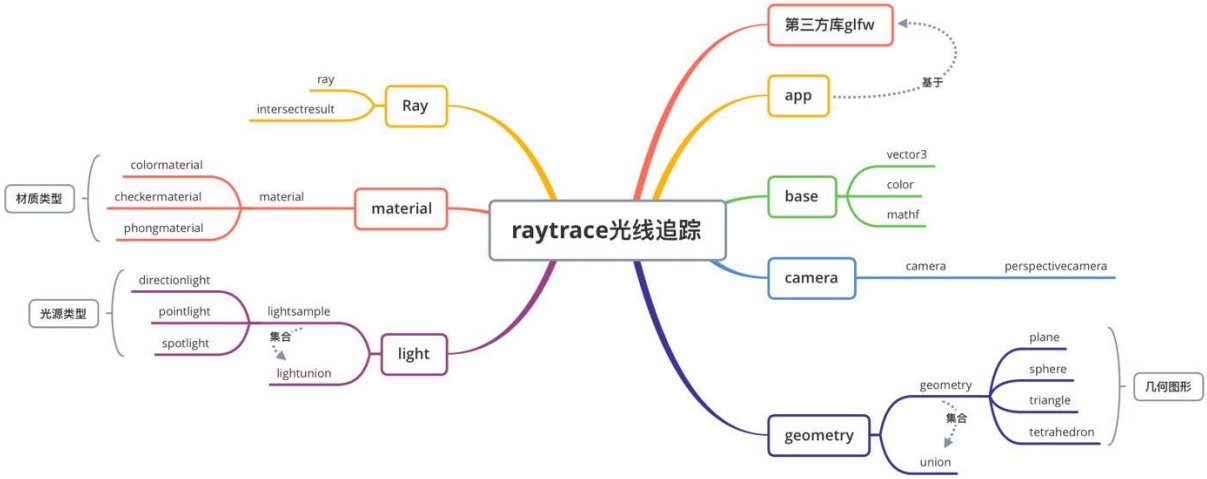


图 1：系统架构图

2. 系统处理流程

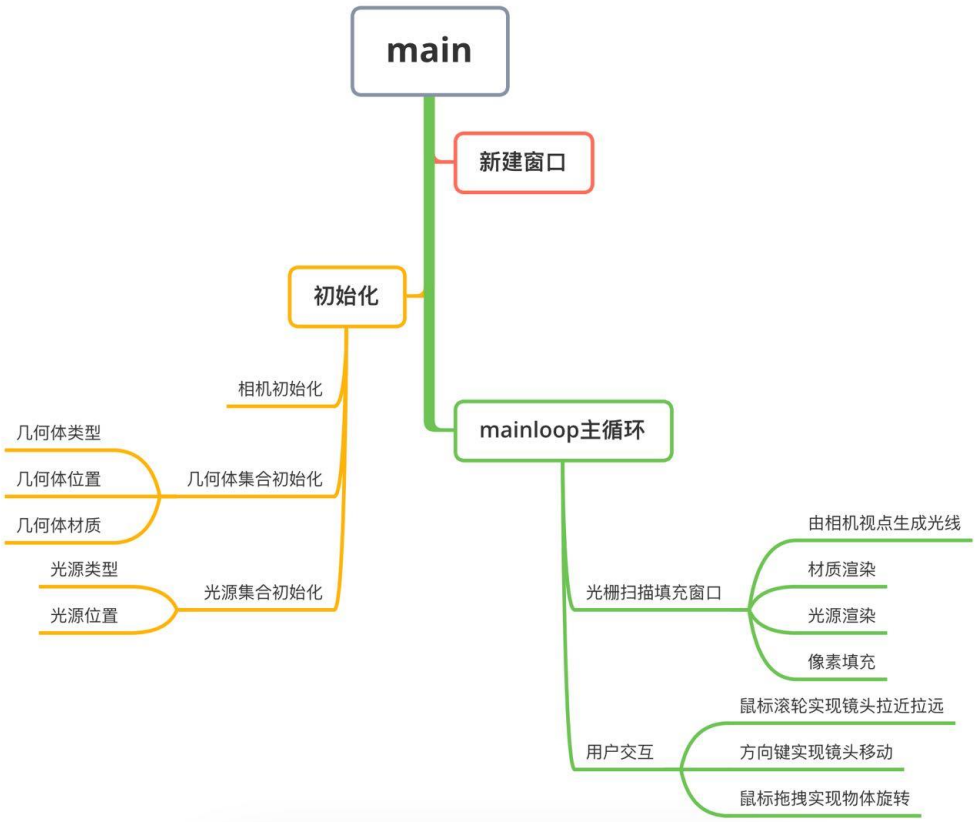


图 2：系统处理流程

三、详细设计

1. 基础函数库

- **Header：** 主要包含 c++常用头文件。
- **Vector3：** 空间三维向量;

  （x，y，z）对应空间三维向量坐标

  主要功能：

  - Length()：求向量模长

- SqrtLength()：求向量模长的二次方
- Normalize()：求单位向量
- Negate()：求反向量
- Add(Vector3)：两向量相加
- Subtract(Vector3)：两向量相减
- Mulitiply(float)：向量数乘
- Divide(float)：向量数除
- Cross(Vector3)：向量叉乘
- Dot(Vector3)：向量点乘

- **Color：** argb 颜色;

  a:透明度（尚未实装），r:红色 g:绿色 b:蓝色

  主要功能：

  - Add(Color)：两颜色相加
  - Multiply(float a)：颜色和实数相乘，颜色加深
  - Modulate(Color)：两颜色相乘

- **Mathf：** 基本数学函数库

  主要功能：

  - Abs()：取实数绝对值
  - Sign()：取实数符号
  - Approximately()：判断两实数是否近似相等
  - Floor()：向下取整，返回实数型
  - FloorToInt()：向下取整返回整数型

- Ceil()：向上取整，返回实数型
- CeilToInt()：向上取整，返回整数型
- Round()：四舍五入，返回实数型
- RoundToInt()：四舍五入，返回整数型
- Clamp()：区间控制
- Clamp01()：区间控制，区间为[0,1]
- Min()：取最小值
- Max()：取最大值
- Sqrt()：开方
- Pow()：求指数
- Exp()：求 e 的指数
- Log()：求对数
- Log10()：求 10 的对数
- IsPowerOfTwo()：是否能被 2 开方
- NextPowerOfTwo()：下一个能被 2 开方的整数
- Sin()：求正弦
- Cos()：求余弦
- Tan()：求正切
- Asin()：求反正弦
- Acos()：求反余弦
- Atan()/Atan2()：求反正切

2. 光线

- **Ray：** 光线;

  origin:光线起点，direction:光线方向

  主要功能：

  - GetPoint(float)：获得光线发射一段距离后的空间位置

    即：ray(光线)=o(光线起点)+t(光线发射距离)*d(光线发射方向)

- **IntersectResult：** 光线和物体相交情况;

  Is_hit：光线是否和物体接触

  Material：接触部位物体材质

  Distance：光线起点到物体接触点的直线距离

  Position：接触点位置

  Normal：接触平面法向量

3. 摄像机

- **Camera：** 相机。
- **PerspectiveCamera：** 透视摄像机，能表现近小远大的视觉效果;

  eye：相机视点

  front：相机向前的单位向量

  right：相机向右的单位向量

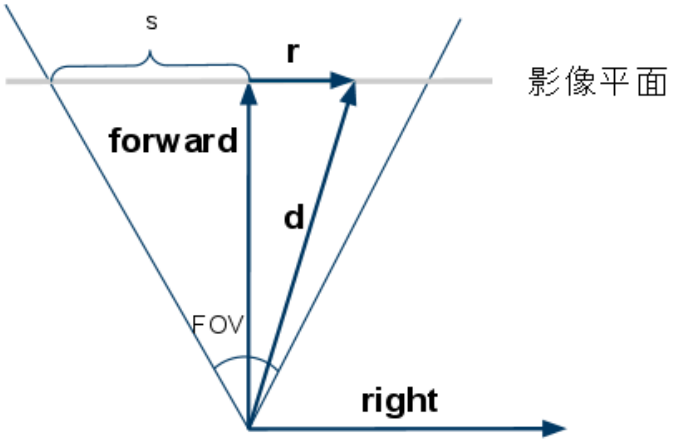
  fov：视野角度

  主要功能：

  - Initialize()：相机初始化

- GenerateRay()：生成到投影面(x,y)处的光线

透视摄像机：GenerateRay()实现原理：



摄像机的几个参数：透视投影从视点（eye），向某个方向观察场景，forward 和 right 分别为向前和向右的单位向量，fov 为水平和垂直方向的视野角度，即为观察的角度范围。由于视点是固定的，光线的起点不变，只须用取样坐标（sx,sy）计算其方向 d。

Fov 和 s 的关系：

$$\tan\frac{fov}{2} = s$$

4. 几何体

- **Generations：** 几何体接口；

  Material：继承属性，材质指针

  继承函数：

  - Intersect(Ray)：几何体与光线相交情况
  - Move_location(Vector3)：移动几何体位置
  - Turn_location(float,float)：几何体绕原点旋转

- **Plane：** 平面；

  normal：平面法向量

position：平面位置

d：空间原点到平面的最短距离

Plane：Insertect()实现原理：
一个无限平面在数学上定义为：

$$n \cdot x = d$$

n 为平面法向量，d 为空间原点至平面的最短距离，x 为平面上任意一点。
光线和平面相交的计算即为：
光线方向和平面法向量点乘的结果小于 0，即光线和平面存在交点，通过计算光线原点到平面的最短距离，即可得出平面和光线的相交情况。

- **Sphere：** 球体；

  center：球体中心

  radius：球体半径

Sphere：Insertect()实现原理：
球表面上的任意一点到球中心的距离为球的半径，用公式表示即为：

$$||x - c|| = r$$

这里 x 表示球上任意一点，c 为球中心，r 为球半径。
将光线 x=r(t) = o+t*d 带入公式，公式求解即为交点，方程化简得：

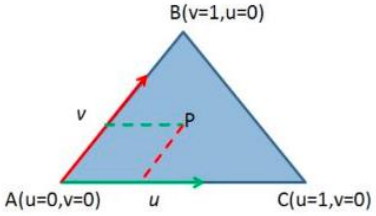
$$t = -d \cdot v \pm \sqrt{(d \cdot v)^2 - (v^2 - r^2)}$$

其中 d 为光线方向单位向量，v=o-c;
若根号内为负数，即相交不发生，且取最近的交点即正负号只需取负号。

- **Triangle：** 三角形面；

  a,b,c:分别为三角形的三个顶点坐标

Triangle：Intersect()实现原理



如图：对于三个顶点为 a,b,c 组成的空间三角形，对于三角形内任意一点，有如下参数方程：

$$P = (1 - u - v) * a + u * b + v * c$$

其中，u>=0,v>=0 u+v<=1;

将光线方程 r(t) = o+t*d 带入参数方程左部，化简，移项，整理可得：

$$[-d, b-a, c-a]\begin{bmatrix} t \\ u \\ v \end{bmatrix} = o - a$$

使用克莱姆法则求解线性方程组，可得：

$$\begin{cases} t = \dfrac{1}{P \cdot e1}(Q \cdot e2) \\ u = \dfrac{1}{p \cdot e1}(P \cdot T) \\ v = \dfrac{1}{P \cdot e1}(Q \cdot D) \end{cases}$$

其中：e1 = b-a，e2 = c-a，T = o-a，P = D×e2，Q = T×e1。

- **Tetrahedron：** 四面体；

  a,b,c,d：分别为四面体的四个三角形面

Tetrahedron：Intersect()实现原理：
四面体由四个三角形面组成，求其与光线的相交情况即为求每个三角形面与光线的相交情况，找出距离最近的交点。

- **Union：** 几何体集合；

  Generations：存放几何体指针的数组

  主要功能：

  - intersect(Ray)：返回光线和距离最近的物体的相交结果
  - Add(Generation*)：添加新的几何体

- move_location(Vector3)：移动集合中所有几何体位置
- turn_location(float,float)：旋转集合中所有几何体

5. 材质

- **Material：** 材质接口；

  reflectiveness：继承属性，材质反光度

  继承函数：

  - prototype(Ray,Vector3 position,Vector3 normal)：根据光线和材质的相交情况，返回像素值

- **ColorMaterial：** 颜色加深度渲染；

  color：材质颜色

> ColorMaterial：prototype()实现原理：
> 在颜色渲染的基础上进行深度渲染，即根据光线和物体交点的距离和相交范围的法向量，并映射到颜色深度范围即[0,255]。

- **CheckerMaterial：** 格子材质；

  scale：单位坐标内的格子数量

> CheckerMaterial：prototype()实现原理：
> 只凭借(x,z)坐标计算某位置的颜色为黑色还是白色。

- **PhongMaterial：** phong 高光材质；

  lightDir：光线方向

  lightcolor：光线颜色

  diffuse：漫射

  specular：镜射

shininess：光滑程度

Phong：prototype()实现原理:
实现简单的 phong 材质，只用全域变量设置了一个临时的光源方向(未来实现材质和光源的互动将结合环境光源进行渲染)，并且只计算漫射和镜射。
Phong 计算公式:

$$I = I_{pa}K_a + \sum(I_{pd}k_d cosi + I_{ps}k_s cos^n\theta)$$

其中，ka 为环境反射系数，kd 为漫反射系数，ks 为镜面反射系数。

6. 光源

- **LightSample：** 光线接口

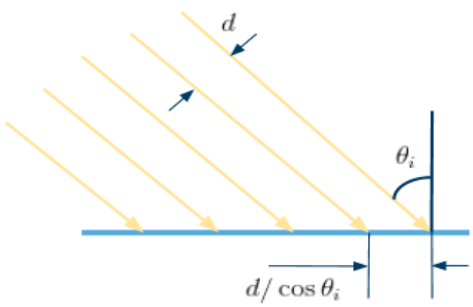
  L：光线方向

  EL：光辐射度（颜色）

  继承函数：

  - sample(Union,Vector3)：渲染辐照度
  - LightRender(Union , IntersectResult)：光照渲染

LightRender():实现原理:
光线渲染是在得到光源辐照度的前提下进行的，不同类型的光源得到辐照度的计算方法不同。
辐照度即平面每面积接收到的能量，平面法向量方向的面积，是光向量方向的面积的$\frac{1}{cos\theta_i}$倍，而辐照度则为其倒数，即为$\cos\theta_i$倍。
如图所示:



渲染辐照度之后，通过判断光源和物体的位置，返回光照渲染结果。

- **DirectionLight：** 平行光；

  direction：平行光方向

  shadow：是否存在阴影

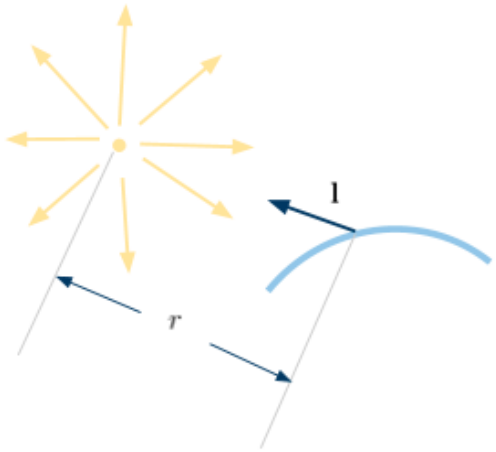
DirectionLight：sample()实现原理：
平行光的辐照度未发生衰减，即辐照度需要经过处理；判断物体和光线的相交情况，若物体和光线相交产生阴影，阴影处的辐照度为 0。

- **PointLight：** 点光源；

  position：点光源位置

  shadow：是否存在阴影

PointLight：sample()实现原理：



点光源是指一个无限小的点，向所有方向平均的散射光。对于点光源来所，接收到的能量和距离的关系，是成平方反比定律的：

$$E_L = \frac{I_L}{r^2}$$

当中 I 为辐射强度，当 r=1 时，辐射强度和辐照度相等。

$\frac{1}{r^2}$通常称为衰减系数。

- **SpotLight：** 聚光灯；

  intensity：光源强度

position：聚光灯中心位置

direction：聚光灯方向

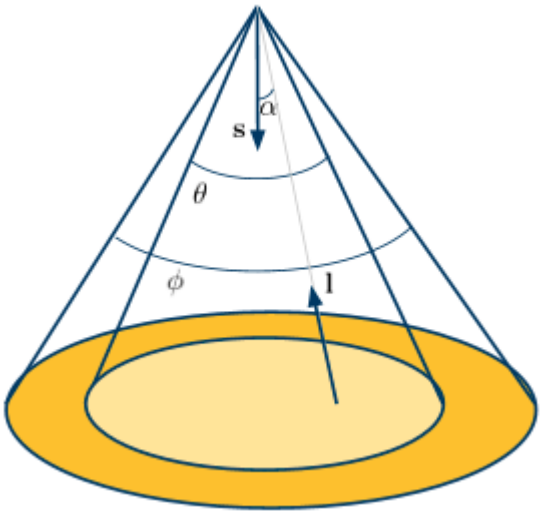
theta：聚光灯内圆锥顶角角度

phi：聚光灯外圆锥顶角角度

falloff：衰减系数

SpotLight：sample()实现原理：

聚光灯是在点光源的基础上，加入圆锥的范围。



聚光灯有一个主要方向 s，再设置两个圆锥范围，称为内圆锥和外圆锥，两圆锥之间的范围称为半影。内外圆锥的内角分别为θ和φ。聚光灯可计算一个聚光灯系数，范围为[0,1]，代表某方向的放射比率。内圆锥中系数为 1(最亮)，内圆锥和外圆锥之间系数由 1 逐渐变成 0。另外，可用另一参数 p 代表衰减(falloff)，决定内圆锥和外圆锥之间系数变化。方程式如下:

$$\text{spot}(\alpha) = \begin{cases} 1, & cos\alpha \geq cos\frac{\theta}{2} \\ (\frac{cos\alpha - cos\frac{\varphi}{2}}{cos\frac{\theta}{2} - cos\frac{\varphi}{2}})^{p}, & cos\frac{\varphi}{2} < cos\alpha < cos\frac{\theta}{2} \\ 0, & cos\alpha \leq cos\frac{\varphi}{2} \end{cases}$$

- **LightUnion：** 光源集合

  lights：存放光源指针的数组

主要功能：

- LightRender(Union,res)：集合中所有光源渲染结果
- Add(LightSample*)：添加光源

7. 用户交互

- **glfwSetScollCallback(glfwWindow,func)：**
  glfw 库函数实现鼠标滚轮控制镜头拉近拉远
  - 回调函数 func：scrollfun(glfwWindow,x,y)；
- **glfwSetKeyCallback(glfwWindow,func)：**
  glfw 库函数实现方向键控制镜头移动
  - 回调函数 func：key_callback(glfwWindow,key,scancode,action,mods)；
- **Course_Mouse_callback(glfwWindow,xpos,ypos)：**
  鼠标左键拖拽实现物体绕原点旋转

8. 其他

- **Init：** 初始化场景，包括初始化物体和光源，用于测试
- **Render：** 实现整体渲染

  主要功能：
  - getcolor(Union,LightUnion,Ray,maxReflect)：进行物体材质和光源渲染，maxReflect 为光线最多反射次数。

四、软件测试

1. 几何体及材质渲染测试

- 三垂直平面

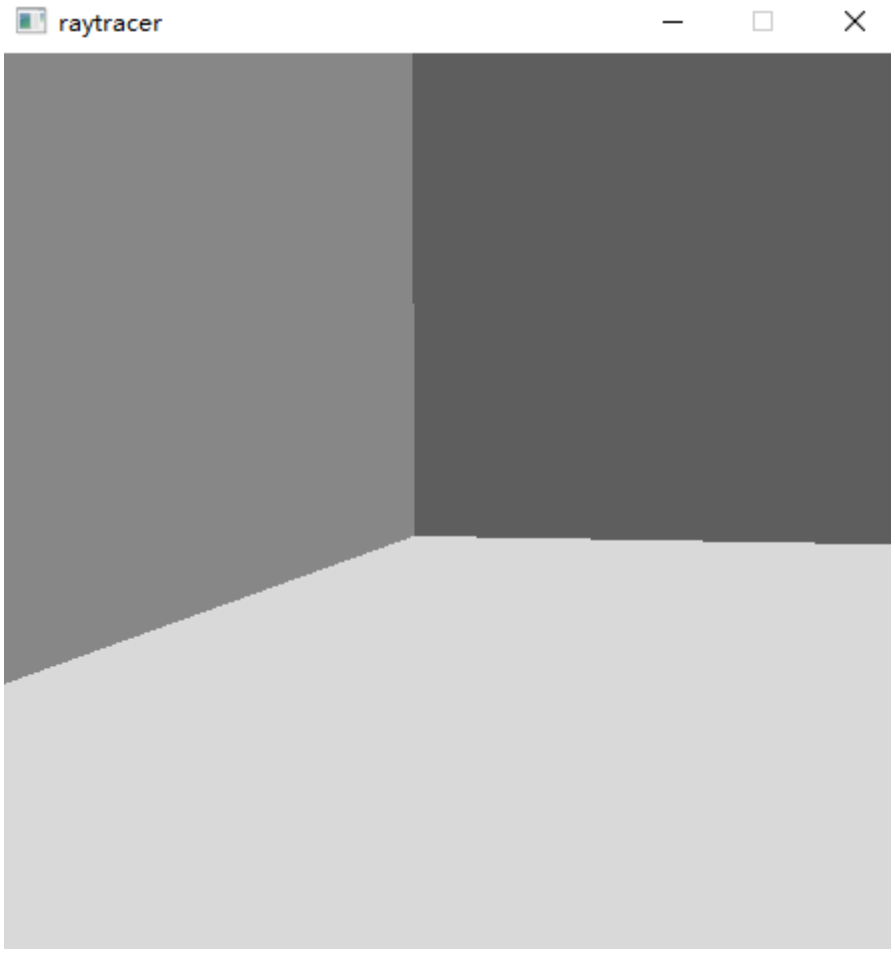


图 3：平面测试

- 球体

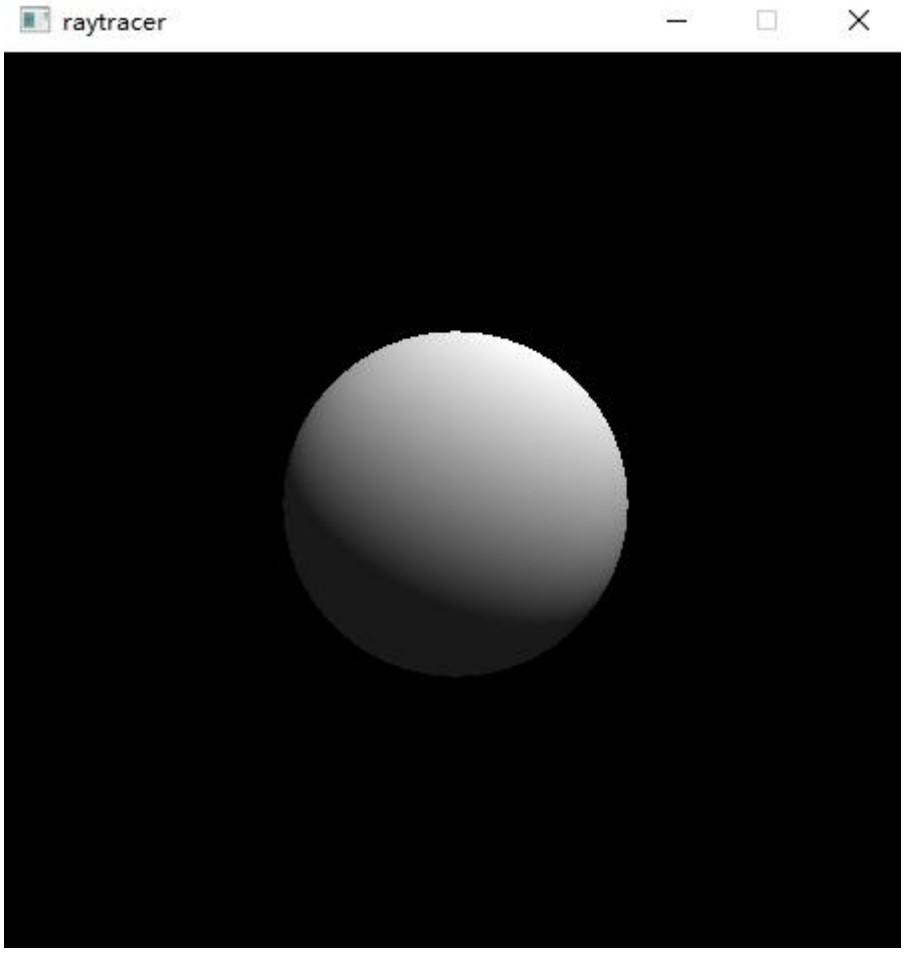


图 4：球体测试

- 平面：格子材质渲染，球体：phong 渲染

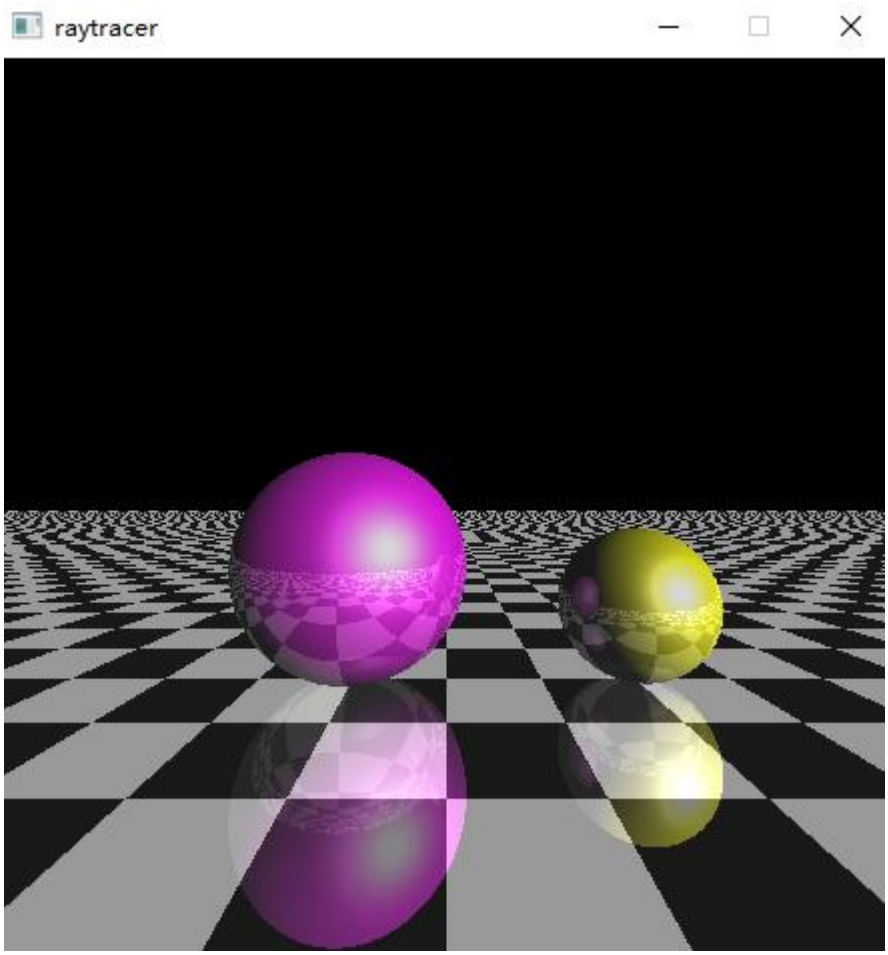
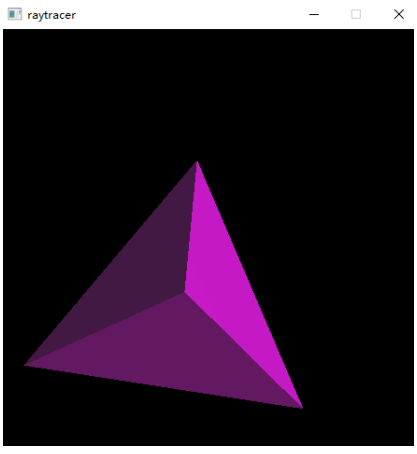
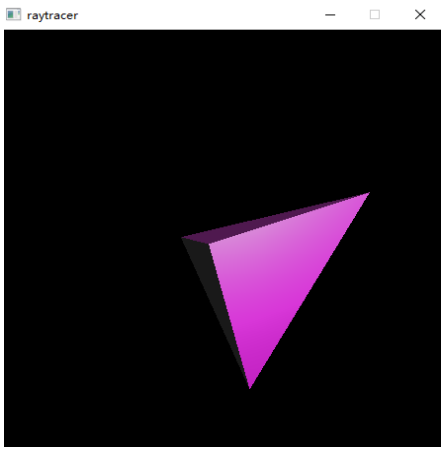


图 4：物体及材质渲染测试

- 四面体 phong 渲染





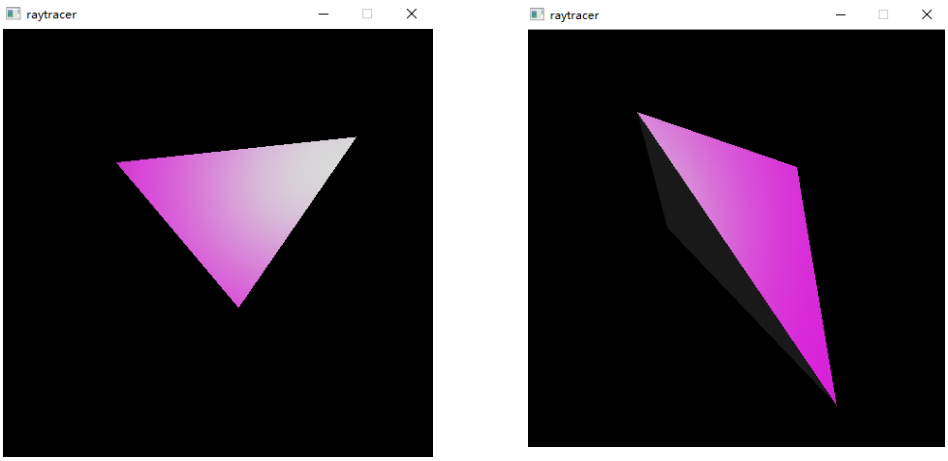


图 5-8：四面体 phong 渲染测试

2. 光源渲染测试

2.1 平行光

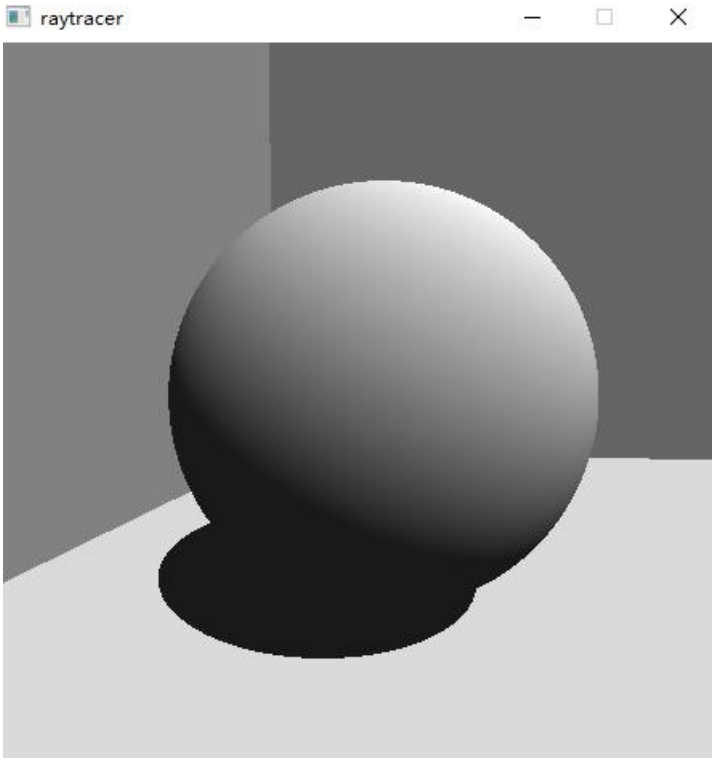


图 9：平行光测试

2.2 点光源

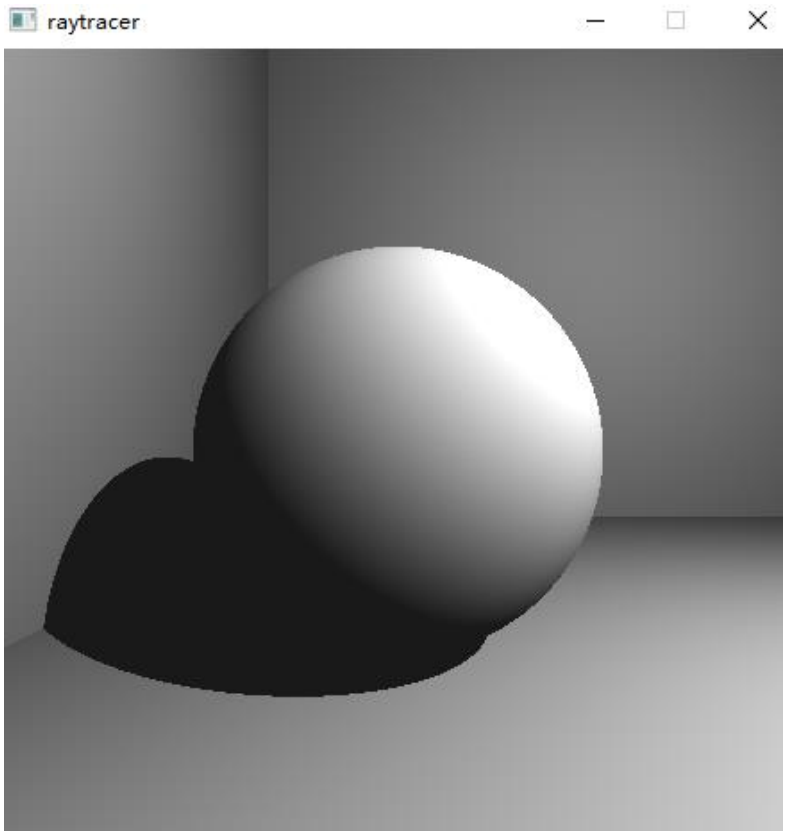


图 10：点光源测试

2.3 聚光灯

- 三原色聚光灯+点光源

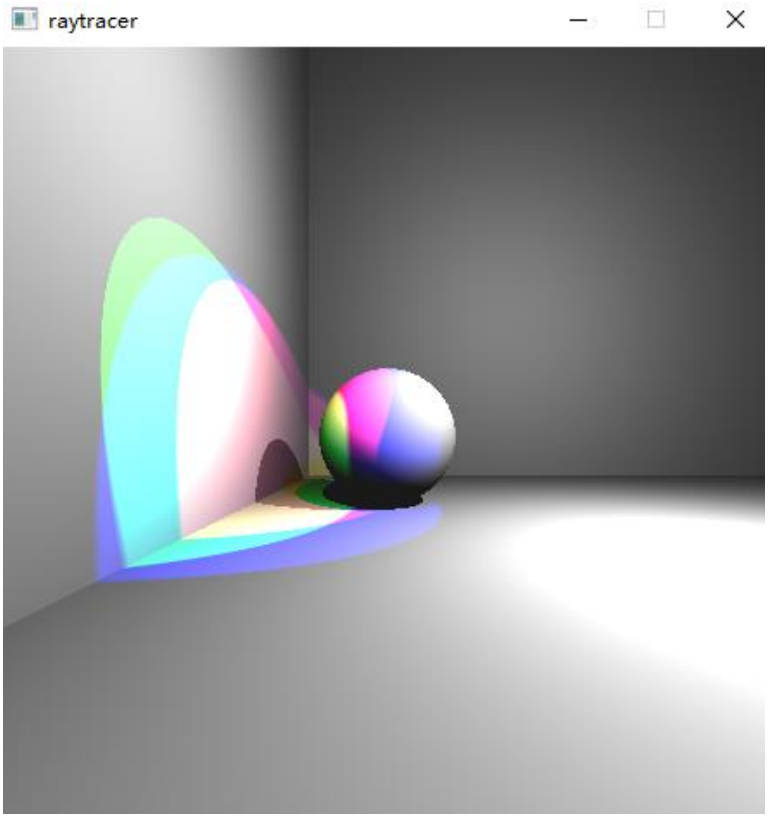


图 11：聚光灯测试

2.4 多个光源

- 采用 16 个点光源和 1 个平行光渲染，多重光源渲染能更好的体现出物体的立体感，但计算量大，渲染速度慢。

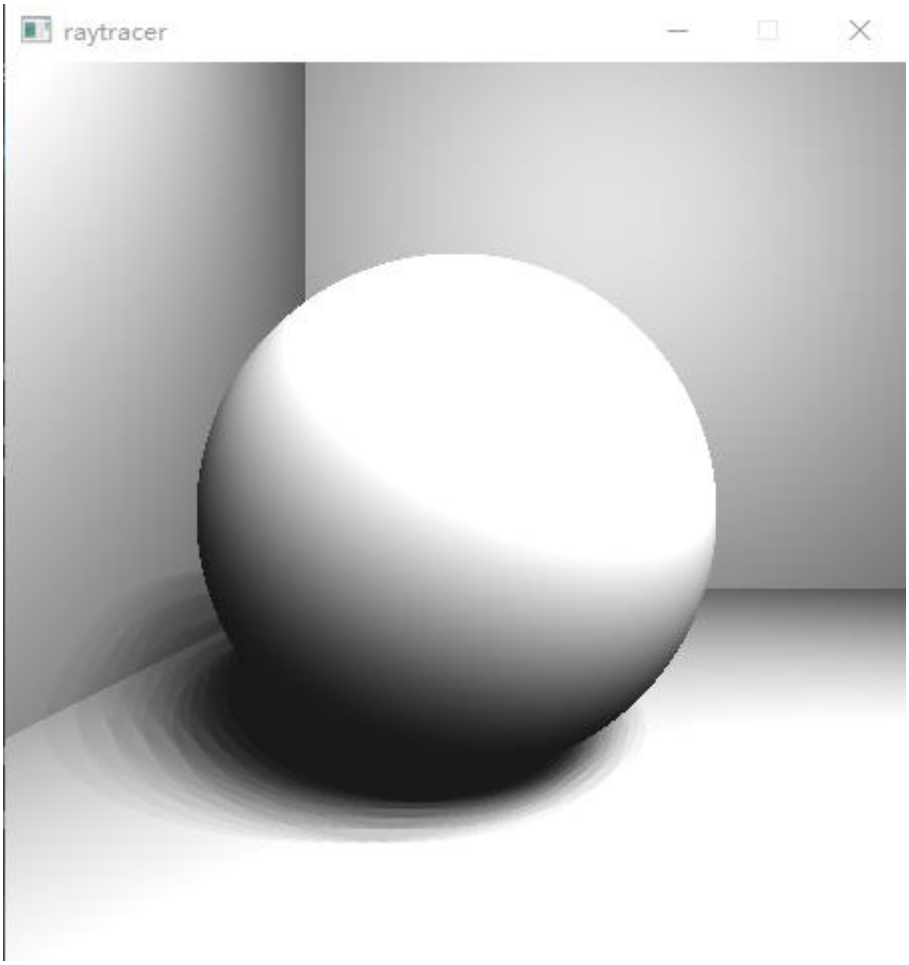


图 12：多光源测试

五、不足与改进

不足：

- 优化：虽然光线追踪需要大量运算量，但目前的程序仍可做进一步的优化，避免不必要的计算，提高速度。
- 实现的几何体种类，材质类型以及光源种类等较少。
- 代码规范：部分类库封装不够全面，内存分配不合理等。

改进：

- 优化算法，减少不必要的计算量。
- 代码规范：严格封装，正确管理内存分配等。

- 实现更多的几何体类型：由三角形面组成矩形，多面体，不规则图形等。
- 实现更多的材质类型：实现材质贴图，玻璃材质，橡胶材质，发光材质等
- 实现更多的光源类型：实现更加真实，更加完美地做到柔和阴影的面光源。
- 实现光源和材质间的互动：如折射，材质反射等。

六、 自评

- 选题(光线追踪)——90 分
- 实现功能(只实现部分基础功能)——83 分
- 代码规范(包括代码结构，封装完整性等)——85 分
- 文档撰写(包括补充文档，ppt 等)——92 分

总评：87.5 分

七、参考资料

[1]GLFW 官方文档.https://www.glfw.org/.2010

[2]Milo Yip.用 JavaScript 玩转计算机图形学（一）光线追踪入门.个人博客 https://www.cnblogs.com/miloyip/archive/2010/03/29/1698953.html.2010-03-29

[3]Milo Yip.用 JavaScript 玩转计算机图形学（二）基本光源.个人博客 https://www.cnblogs.com/miloyip/archive/2010/04/02/1702768.html.2010-04-02