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）
21301004	陈俊杰	8	25	16	36	85

北京交通大学

《计算机图形学》期末课程论文

基于光线追踪技术的计算机图形学渲染研究

The Research on Computer Graphics Rendering Based on Ray Tracing Techniques

学　　院：软件学院

专　　业：软件工程

学生姓名：陈俊杰

学　　号：21301004

指导教师：吴雨婷

北京交通大学

2024 年 6 月

中文摘要

本文探讨了计算机图形学中的光线追踪渲染技术。光线追踪是一种模拟光线传播路径以生成高度逼真图像的方法，能够自然地实现阴影、反射和折射等复杂光照效果。

为了深入研究光线追踪的应用和效果，我实现了一个基于光线追踪的渲染项目，并通过与传统的光栅化技术进行对比，分析了两者在图像质量、渲染性能和适用场景方面的差异。

实验结果表明，光线追踪在表现真实感和复杂光照效果方面具有明显优势，特别是在处理反射和折射时，其效果远超光栅化技术。然而，光线追踪在实时渲染中的性能表现不及光栅化，需要更多的计算资源和时间。本文通过具体实例和实验数据展示了光线追踪的优缺点，并讨论了其在不同应用场景中的潜力和挑战，为进一步研究和应用提供了参考。

关键词： 计算机图形学；光线追踪；渲染技术；图像质感

目　　录

1 引言

1.1 研究背景

计算机图形学是现代计算机科学的重要分支，广泛应用于影视制作、游戏开发、虚拟现实等领域。随着计算机硬件性能的不断提升，渲染技术也在不断发展，以追求更高的图像质量和真实感。在众多渲染技术中，光线追踪（Ray Tracing）因其能够自然模拟光线的传播路径和光照效果，成为研究的热点之一。

1.2 研究目的与意义

本文旨在通过实现一个基于光线追踪的渲染项目，深入探讨光线追踪技术的应用和效果，并与光栅化技术进行对比，分析两者在图像质量、渲染性能和适用场景方面的差异。通过实验结果和数据分析，本文将展示光线追踪在生成高质量图像方面的优势，同时讨论其在实时渲染中的挑战和潜力，为未来研究和应用提供参考。

1.3 论文组织结构

本论文各章节主要内容如下：

第二章：介绍光线追踪的相关研究工作。

第三章：详细描述项目实现的方法，包括环境贴图加载、材质和球体定义以及光照效果的实现。

第四章：介绍实验设置，包括实验环境、工具和步骤。

第五章：呈现实验结果与分析，通过具体实例对比光线追踪和光栅化的效果。

第六章：总结与展望，总结本文的主要发现，并展望后续的进一步研究与改进方法。

2 相关工作介绍

光线追踪是一种基于物理光学的图像生成方法，通过模拟光线从光源到视点的传播路径，计算光线与场景中物体的交互，生成逼真的图像。该技术可以实现准确的阴影、反射、折射和全局光照效果，因此在静态场景的高质量渲染中具有显著优势。

近年来，真实感图形渲染更是受到了科研人员以及企业大众的青睐。在电影特效模拟、3D 游戏、科学计算可视化、自然景物仿真、计算机动画、虚拟现实系统以及多媒体教育等方面，真实感图形渲染都具有举足轻重的作用[1]。有关于光线追踪的理论研究与实践历史悠久，并且也有许多的相关理论研究与相关技术实现。

光照算法当中，最为经典、研究最为广泛的算法就是光线跟踪算法，它是基于 1968 年 Arthur Appel 提出的光线投射算法建立的。在传统的局部光照模型中，经典的是由 Lambert 提出的漫反射模型以及 Phong 提出的具有镜面反射效果的光照模型，该算法在计算上未考虑到模型表面的部分直接光照以及间接光照,无法真实模拟场景中的阴影以及光照效果。为了实现更好的光照效果，2012 年，Burleyl17 提出了一种可以广泛应用在不同材质表面的全局光照算法。但是该算法的求解成本较高,并且对于存储空间的需求量也很大[2]。

目前也已有许多较为完善的光线追踪器：如教育和科研领域经常使用到的 Mitsuba 渲染器，该渲染器是一款开源的渲染器。常用于模型商业渲染的 V-Ray 插件，目前已经集成到 3DMAX 和 SketchUp 两款工具当中。此外英伟达公司开发了的 Optix 图形库，该图形库基于英伟达公司的 CUDA 工具包，使得使用者能够通过编程，使光线追踪在英伟达所产的 GPU 中进行渲染[3]。

光线追踪在计算机 3D 图形处理中占有重要地位，且应用广泛，相关技术也逐渐适用于各行各业当中。光线追踪算法遵循真实世界中光线的传播、折射、衍射、吸收等过程的规律，可渲染出高真实感的 3D 场景。

3 方法描述

3.1 光线追踪算法基本原理

光线追踪算法是一种基于物理光学的图像生成技术，通过模拟光线在场景中的传播路径，生成高度逼真的图像。光线追踪基本原理示意图如图 3-1 所示，其基本原理总结如下：

从相机位置（或观察点）出发，向场景发射光线。每一条光线穿过图像平面的一个像素中心，进入三维场景，这些光线称为视线（View Rays），用于确定每个像素的颜色。判断每条视线是否与场景中的物体（如图中的球体）相交是关键步骤，这通常通过计算光线与几何体的相交点来完成。如果光线与物体相交，则计算交点的位置。在交点处，需要计算该点的光照情况，这时会发射阴影光线（Shadow Rays）从交点向光源出发，以检测该点是否被遮挡。如果阴影光线在到达光源前与其他物体相交，则该点在阴影中，否则它被光源直接照亮。随后，使用光照模型（如 Phong 或 Blinn-Phong 模型）计算交点的颜色，这包括考虑环境光、漫反射光和镜面反射光等因素。同时，通过计算反射光线和折射光线，可以模拟镜面反射和透明物体的效果。最终，将所有交点的光照计算结果综合，得到该视线对应像素的最终颜色。如果视线与场景中的任何物体都不相交，则该像素的颜色为背景色。

光线追踪算法的主要特点在于其能生成高度逼真的图像，特别是对于光影效果、反射和折射等复杂光学现象有出色的表现。然而，由于其需要大量的光线-物体相交计算，因此计算复杂度较高，通常需要较长的渲染时间。通过这些步骤，光线追踪算法能够模拟光在三维场景中的传播和交互，从而生成逼真的图像效果。

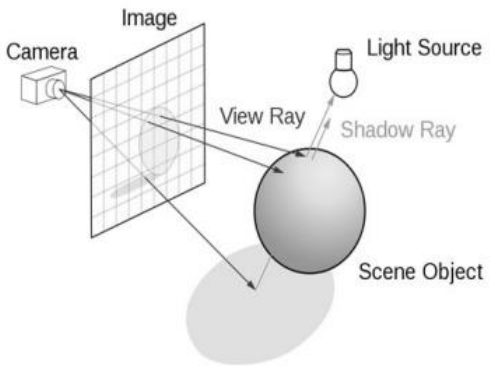

图 3-1 光线追踪原理示意图

3.2 光线追踪的实现

该部分将依次介绍光线追踪的实现过程。

3.2.1 光线生成

根据摄像机位置和视角生成初始光线。每个像素对应一条光线，从摄像机出发，穿过成像平面，进入场景中。

```
for (size_t j = 0; j < height; j++) {
    for (size_t i = 0; i < width; i++) {
        float dir_x = (i + 0.5) - width / 2.;
        float dir_y = -(j + 0.5) + height / 2.;
        float dir_z = -height / (2. * tan(fov / 2.));
        framebuffer[i + j * width] = cast_ray(Vec3f(0, 0, 0), Vec3f(dir_x, dir_y, dir_z).normalize(), spheres, lights);
    }
}
```

3.2.2 光线与物体的交点计算

计算光线与场景中物体的交点。如果光线与物体相交，计算交点位置、法向量和材质属性。

```
bool scene_intersect(const Vec3f &orig, const Vec3f &dir, const vector<Sphere> &spheres, Vec3f &hit, Vec3f &N, Material &material) {
    float spheres_dist = std::numeric_limits<float>::max();
    for (size_t i = 0; i < spheres.size(); i++) {
        float dist_i;
        if (spheres[i].ray_intersect(orig, dir, dist_i) && dist_i < spheres_dist) {
            spheres_dist = dist_i;
            hit = orig + dir * dist_i;
            N = (hit - spheres[i].center).normalize();
            material = spheres[i].material;
        }
    }
    return spheres_dist < 1000;
}
```

3.2.3 光照模型计算

实现 Phong 光照模型，计算光线与物体交点处的光照强度，包括漫反射、镜面反射和环境光照。

```
float diffuse_light_intensity = 0;
float specular_light_intensity = 0;
for (size_t i = 0; i < lights.size(); i++) {
    Vec3f light_dir = (lights[i].position - point).normalize();
```

```
5.      float light_distance = (lights[i].position - point).norm();
6.      diffuse_light_intensity += max(0.f, light_dir * N) * lights[i]
   .intensity;
7.      specular_light_intensity += pow(max(0.f, reflect(light_dir, N)
    * dir), material.specular_exponent) * lights[i].intensity;
8. }
```

3.2.4 递归反射和折射

通过递归调用光线追踪函数，实现反射光线和折射光线的计算，生成复杂的光照效果。

```
1. Vec3f reflect_color = cast_ray(reflect_orig, reflect_dir, spheres,
    lights, depth + 1);
2. Vec3f refract_color = cast_ray(refract_orig, refract_dir, spheres,
    lights, depth + 1);
```

4 实验设置

4.1 实验环境和工具

为了验证光线追踪在静态场景中的表现，如表 4-1、4-2 所示，本实验在以下环境和工具下进行：

硬件配置	
处理器	Intel(R) Core(TM) i9-14900HX
内存	16GB DDR5
显卡	NVIDIA GeForce RTX 4060 Laptop GPU
存储	1TB SSD

表 4.1 硬件配置

软件配置	
操作系统	Windows 11
开发环境	Visual Studio 2022
编程语言	C++
图形库	OpenGL
图像处理库	stb_image

表 4.2 软件配置

4.2 场景设置和参数选择

实验场景包含多个具有不同材质和光照属性的球体，以及一个环境贴图背景。

4.2.1 环境贴图

使用一张高分辨率的环境贴图（envmap.jpg），如图 4-1 所示，用于模拟背景环境光照。

图 4-1 envmap.jpg

4.2.2 材质定义

实验主要实现了象牙、玻璃、红色橡胶和镜子四种材质，材质定义的具体参数如下表 4.3 所示。

材质名	折射率	漫反射系数	镜面反射系数	环境光系数	颜色（RGB）	镜面指数
象牙	1.0	0.6	0.3	0.1	(0.4, 0.4, 0.3)	50
玻璃	1.5	0.0	0.5	0.1	(0.6, 0.7, 0.8)	125
红色橡胶	1.0	0.9	0.1	0.0	(0.3, 0.1, 0.1)	10
镜子	1.0	0.0	10.0	0.8	(1.0, 1.0, 1.0)	1425

4.3 材质定义参数表

4.2.3 光源设置

实验主要设置了三个光源，光源信息如下表 4.4 所示。

光源信息表		
光源名	位置	强度
光源 1	(-20, 20, 20)	1.5
光源 2	(30, 50, -25)	1.8
光源 3	(30, 20, 30)	1.7

表 4.4 光源信息表

4.2.4 球体定义

实验球体定义如下表 4.5 所示。

球体定义表			
球体名	位置	半径	材质
球体 1	(-3, 0, -16)	2	象牙
球体 2	(-1.0, -1.5, -12)	2	玻璃
球体 3	(1.5, -0.5, -18)	3	红色橡胶
球体 4	(7, 5, -18)	4	镜子

表 4.5 球体定义表

4.3 实验步骤

4.3.1 场景初始化

加载环境贴图，初始化材质和光源属性，创建球体对象。设置摄像机位置和视角。

4.3.2 光线追踪渲染

使用光线追踪算法渲染场景。遍历每个像素，生成从摄像机到像素中心的光线，计算光线与场景的交点和光照效果，生成最终图像。输出渲染结果为光线追踪图像（raytraced.jpg）。

4.3.3 光栅化渲染

使用光栅化算法渲染相同场景。遍历每个像素，生成从摄像机到像素中心的光线，计算光线与场景的交点和基本光照效果。输出渲染结果为光栅化图像（rasterized.jpg）。

4.3.4 结果分析

比较光线追踪和光栅化的渲染图像，从图像质量、渲染时间和性能等方面进行分析。总结两种渲染技术的优缺点，并讨论其在不同应用场景中的适用性。

5 实验结果与分析

5.1 实验结果展示

实验中，分别使用光栅化和光线追踪技术对相同场景进行了渲染，如图 5-1、图 5-2 所示，生成了两幅图像。

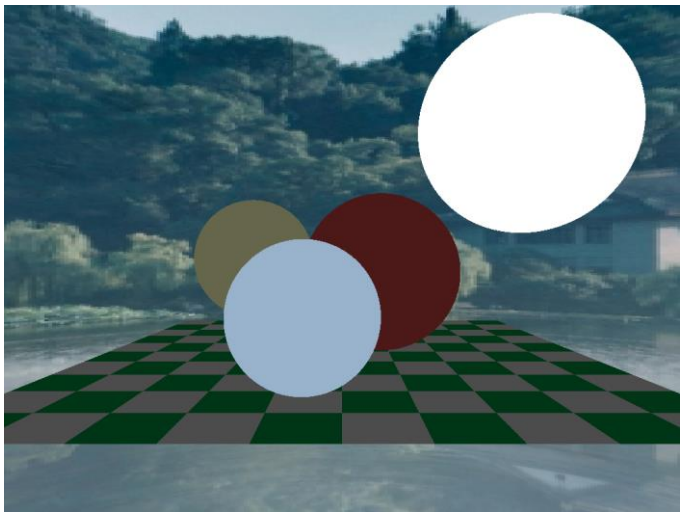

图 5-1 光栅化图像

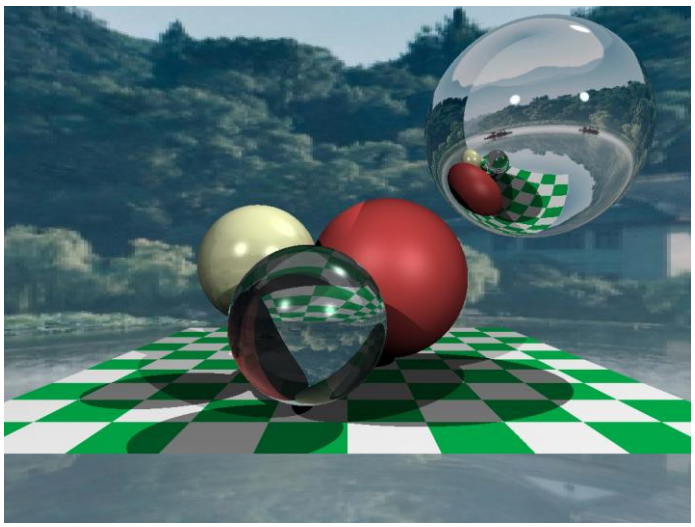

图 5-2 光线追踪图像

5.2 图像质量对比

从图像质量上看，光线追踪技术生成的图像在细节表现上显著优于光栅化技术生成的图像。

5.2.1 阴影效果

光线追踪技术能够生成自然的阴影效果，光源位置和物体遮挡关系在图中得到了真实的反映。而光栅化技术生成的图像阴影效果较为简单，不够真实。

5.2.2 反射效果

光线追踪技术能够模拟物体表面的反射效果，如图 5-2 中玻璃球和镜子球的反射效果非常逼真，甚至能够看到其他物体的反射影像。而光栅化技术缺乏这种反射效果，使得图像显得平面且缺乏真实感。

5.2.3 折射效果

光线追踪技术能够模拟物体内部的折射效果，图 5-2 中玻璃球展示了真实的折射和内部影像的变化，而光栅化技术无法实现这一点。

5.2.4 光照效果

光线追踪技术计算了每个像素的光照强度，包括漫反射和镜面反射，使得图像具有更高的对比度和光照细节。光栅化技术虽然也计算光照，但效果不如光线追踪细腻和真实。

5.3 渲染性能分析

在渲染性能方面，光线追踪技术的计算复杂度远高于光栅化技术，导致渲染时间明显更长。这也验证了以下结论：

光栅化渲染时间：光栅化技术通常在几秒钟内即可完成渲染，适用于实时渲染场景。

光线追踪渲染时间：光线追踪技术由于需要模拟大量光线的传播路径和交互，渲染时间通常需要几分钟甚至更长，视场景复杂度而定，适用于需要高质量图像，如电影制作、静态图像生成等场景。

6 总结与展望

实验结果表明，光线追踪技术在生成高质量图像方面具有明显优势，能够自然地模拟阴影、反射和折射等复杂光照效果。然而，其高计算复杂度和长渲染时间限制了其在实时渲染中的应用。光栅化技术虽然在图像质量上不如光线追踪，但其高效的渲染性能使其在实时渲染中具有广泛应用。

未来的研究可以着重于优化光线追踪算法，提高其渲染效率，使其在实时应用中更加可行。同时，结合光线追踪和光栅化的混合渲染技术，也是一种值得探索的方向，能够在保证图像质量的前提下，提高渲染效率。

参考文献

[1] 高园园. 基于光线追踪的真实感图形渲染 IP 的研究与实现[D]. 山东大学, 2015.

[2] 郭笑孜. 基于光线跟踪的实时全局光照算法研究与实现[D]. 陕西师范大学, 2021.

[3] 张文逍. 基于光线追踪的渲染系统的设计与实现[D]. 华中科技大学, 2016.

[4] John F. Hughes, Andries van Dam, Morgan McGuire, et al. Computer Graphics: Principles and Practice (4th Edition). Addison-Wesley Professional, 2013.