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）
21271104	陶俊杰	6	20	16	36	78

没有相关工作介绍，没有参考文献

基于光线追踪实现真实的反射效果

摘要

本论文旨在展示通过 OpenGL 实现光线追踪技术，以模拟真实的反射效果。本论文详细描述了实现光线追踪的步骤和方法，并展示了其在示例场景中的应用效果。实验结果表明，采用光线追踪技术可以显著提升图像中的反射细节和真实性。

1. 引言

光线追踪（Ray Tracing）作为一种先进的图像渲染技术，能够模拟光线在三维空间中的传播路径，从而生成高度逼真的视觉效果。相比传统的光栅化渲染方法，光线追踪在处理反射、折射、阴影和全局光照等方面具有显著优势。这种技术通过追踪光线与物体表面及其相互作用的过程，实现了对光线行为的高度精确模拟，能够呈现出近乎真实的场景。然而，其高计算复杂度使得实时应用面临较大挑战。光线追踪需要大量的计算资源来处理每个光线与场景中所有物体之间的交互，这导致了计算量的急剧增加，使得在过去很长一段时间内，实时光线追踪只能停留在理论和实验室阶段。

近年来，随着硬件性能的提升和算法优化，实时光线追踪逐渐成为可能。现代图形处理单元（GPU）的计算能力不断增强，专门为光线追踪设计的硬件加速器也应运而生，大大提高了光线追踪的处理速度。此外，各种优化算法的开发和应用，例如基于层次包围体（Bounding Volume Hierarchy, BVH）的加速结构和降噪算法（Denoising Techniques），进一步降低了计算负担，使得实时渲染不再是遥不可及的目标。这些技术进步不仅提升了图像渲染的效率，还扩大了光线追踪的应用范围，从影视制作、建筑可视化，到电子游戏和虚拟现实，光线追踪正在逐步改变着各个行业的视觉表现形式。

2. 相关工作介绍

本论文通过 OpenGL 实现了基本的光线追踪算法，构建了一个包含多个几何体和光源的简单场景，在该场景下进行了光线追踪算法的应用，并与使用光线追踪算法之前的效果进行对比，以验证其效果和可行性。

3. 方法描述

3.1. 光线求交

本项目仅考虑了光线分别与球面和三角形面相交的情况。计算光线与球面相交时，联立求解光线与球面的方程，得到交点的坐标。计算光线与三角形面相交时，仍然联立方程求解，得到交点的坐标之后，还需额外判断交点是否在三角形内，本项目使用了重心法来进行判断，假设三角形的三个顶点分别为 A、B、C，光线与三角形的交点为 P，再求解 AP = u * AB + v * AC，若 u ≥ 0、v ≥ 0、u + v ≤ 1，则交点在三角形内，否则不在。

3.2. 光线衰减

本项目考虑了光线的距离衰减和阴影衰减。

计算距离衰减时，不对平行光源进行衰减计算，仅对点光源进行衰减计算，使用的衰减公式如公式 3-1 所示。

$$A_j^{dist} = min\left\{1, \frac{1}{a_j + b_j r_j + c_j r_j^2}\right\} \qquad (3-1)$$

在本项目中，公式 3-1 所使用的具体参数为：

$a_j = 0.25, b_j = 0.003372407, c_j = 0.000045492$。

计算阴影衰减时，对于平行光，只需判断监测点处光源是否被遮挡。对于点光源，在判断光源是否被遮挡后，还需判断监测点是否超过了点光源的照射范围。

3.3. 光线追踪

首先，基于 Blinn-Phong 反射模型实现了光源直射光强计算函数：先计算光线与物体交点的位置和法向量，再初始化光照结果为自发光和环境光之和，接着对所有的光源进行遍历，将距离衰减和阴影衰减的衰减因子进行点乘，得到真正的衰减因子，将该光源的漫反射成分和镜面反射成分之和与衰减因子进行点乘，得到该光源对光照结果的贡献，并加到光照结果之中。

然后，设计并实现了光线追踪函数：首先计算光线与所有物体的交点中里离起点最近的点，若无交点则返回光强为 0，即为黑色；若有交点，则在交点处调用前面实现的光源直射光强计算函数，得到通过光源直射造成的光强，再将反射光线和折射光线作为该流程的输入进行递归计算，得到反射光线与折射光线造成的光强，将三个光强相加得到最终的光强，并作为结果返回。

其中，在递归计算时，若光线与物体没有交点，或光线射到了背景，或超过给定的递归深度时，会结束递归并返回计算结果。

4. 实验设置

4.1. 实验环境

本项目参考了主流的光线追踪技术，并通过 C++进行了实现。此外，实验依赖于 GLFW 和 GLAD 来初始化和管理 OpenGL 上下文，并使用 GLM 库进行向量和矩阵运算。下面是具体的环境列表：

操作系统：Windows 11

开发工具：Visual Studio 2022

图形库：GLFW 3.4

OpenGL 加载器：GLAD 3.3

数学库：GLM 0.9.8.5

4.2. 场景设计

为了测试光线追踪算法的效果，本项目设计了一个简单而典型的场景，如图 4.1 所示。

场景包括以下元素：

一个位于下方的蓝白相间的网格平面，完全不具有反射能力，用于提供一个清晰的反射参考。

两个悬浮在空中的球体：颜色分别为绿色和蓝色，具有一定的反射能力，用于展示多个镜面物体之间相互的反射效果。

两束平行光源：一束自上向下的白色平行光源，用于模拟日光效果；另一束从摄像机左上侧打向蓝色小球的暗白色平行光源，用于稍微照亮下面平面上由上方球体遮挡造成的阴影。

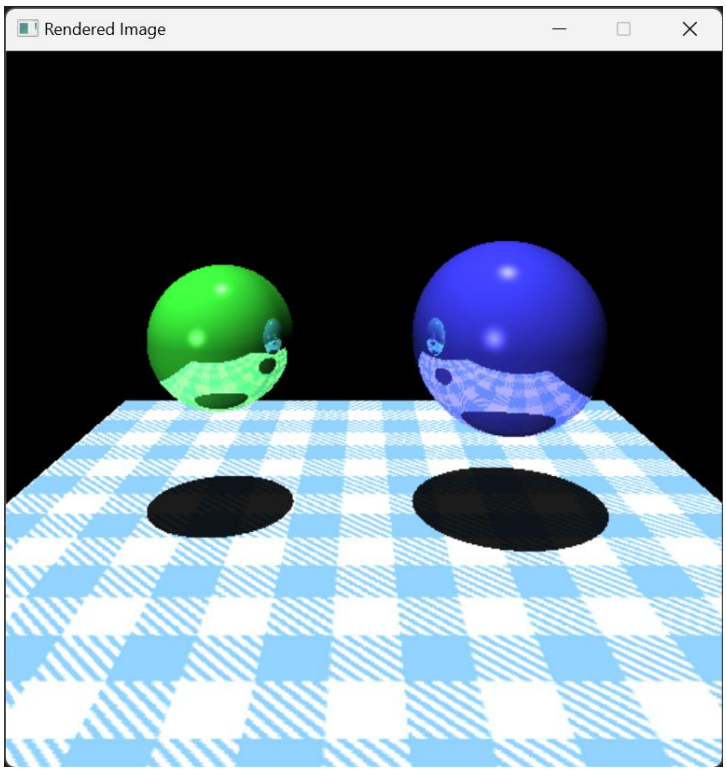

图 4.1 本项目设计的场景

5. 实验结果与分析

5.1. 前后对比

本项目通过对比开启和未开启光线追踪的图像效果，来验证光线追踪技术在反射模拟中的优势。如图 5.1 所示，左侧为未开启光线追踪的效果，右侧为开启光线追踪后的效果。

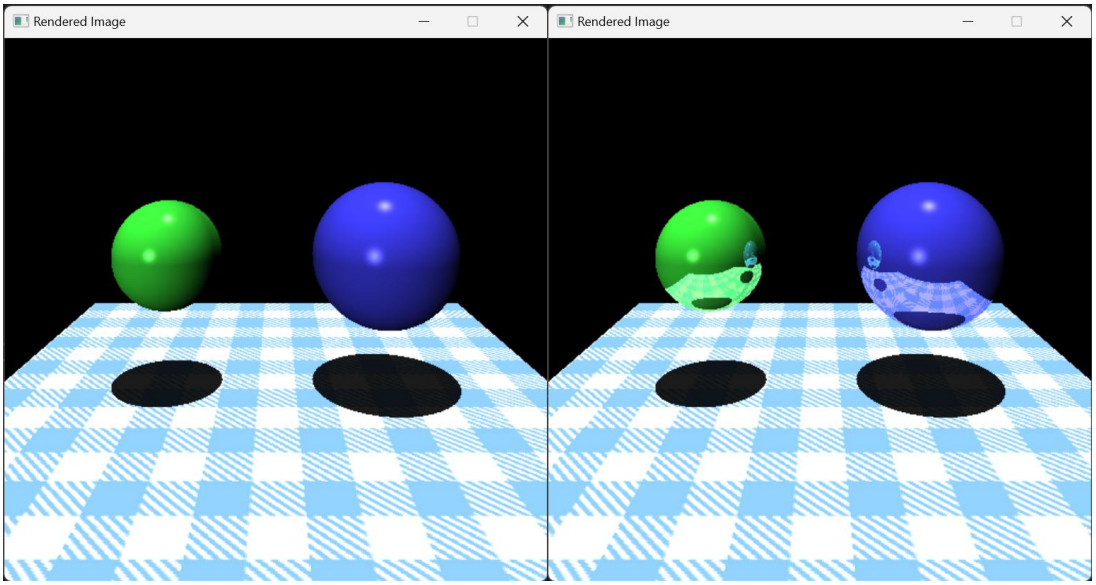

图 5.1 开启光线追踪前后对比

可以看到，未开启光线追踪时，场景中的物体表面显得缺乏真实感，图像整体较为平面化，不存在反射效果。而在开启光线追踪后，上述问题得到了显著改善，具体表现为：上方的两个球体表面呈现出明显的反射效果，能够看到下方的蓝色网格和另一个球体的反射，仔细观察，在球体上的的另一个球体的反射内部，还有着这样的反射，呈现出一种万花筒的效果。

5.2. 性能分析

本项目测量了不同递归深度下的渲染时间，发现随着递归深度的增加，计算时间呈指数增长。当递归深度达到 4 层以上时，渲染时间有显著的增加。这主要是由于每层递归都会生成新光线，并需要计算这些光线与场景中所有物体的交点。

6. 结论

6.1. 总结

本论文展示了如何通过 OpenGL 实现光线追踪技术，以模拟真实的反射效果。通过详细描述光线求交、光线衰减以及光线追踪的实现步骤和方法，我们成功地在一个简单场景中应用了这一技术。实验结果表明，相较于传统的光栅化渲染方法，光线追踪技术在处理反射、阴影和全局光照方面具有显著优势。

具体表现为：

显著提升反射细节和真实性：开启光线追踪后，场景中的反射效果更为真实，物体表面的光照处理更加细腻。

细腻的阴影处理：光线追踪技术在阴影的生成和处理上表现出色，能够准确模拟光源与物体之间的遮挡关系。

多重反射效果：光线追踪能够模拟复杂的多重反射情况，提升图像的层次感

和真实感。

6.2. 限制与未来工作

尽管光线追踪技术优势明显，但本项目在性能方面仍存在一些局限。由于仅使用 CPU 进行计算，当递归深度较大时，渲染速度明显减慢，难以实现实时渲染。为了进一步提高性能，未来可以利用 GPU 的并行计算能力：通过 CUDA 或其他 GPU 编程技术，将光线追踪算法移植到 GPU 上执行，可以显著提升渲染速度，实现实时光线追踪。