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）
21301082	王增超	8	25	16	33	82

计算机图形学期末课程论文
调研光线追踪技术

王增超
21301082@bjtu.edu.cn

摘要—光线追踪技术 [1] 作为计算机图形学中的一项革命性技术，以其高度逼真的渲染效果在电影制作、游戏开发和三维可视化等领域发挥着重要作用。本文旨在调研光线追踪技术的原理、发展历程以及在现代图形渲染中的应用现状，对该技术进行应用，并对实践结果进行讨论和分析。

文章首先介绍了计算机图形学的基本概念和光线追踪技术的基本原理，包括光线与物体表面的交互以及光线在场景中的传播方式。随后，本文介绍光线追踪技术的发展历程以及在现代图形渲染中的应用现状。最后，本文通过一个基于 OpenGL 的光线追踪渲染器，展示光线追踪技术的应用。

***Index Terms*—光线追踪，计算机图形学，渲染技术，OpenGL**

I. 介绍

计算机图形学是计算机科学的一个分支，它涉及到使用计算机技术和算法来创建、处理和显示图像和图形。光线追踪技术是一种模拟光线在三维场景中传播和交互的计算机图形学渲染方法，它通过以下步骤生成逼真的图像：首先从观察者位置发射光线，这些光线与场景中的物体相交并根据物体的表面属性进行交互，包括吸收、漫反射、镜面反射和折射。交互后，光线可能产生新的反射或折射光线，继续在场景中传播并可能与更多物体交互。为了模拟复杂的光照效果，如软阴影和间接光照，光线追踪还会考虑全局光照，即光线在场景中的多次反射。此外，技术如抗锯齿用于减少图像的锯齿效应，而路径追踪则通过随机采样光线路径来计算场景的最终颜色和亮度。

本文中利用 OpenGL 递归实现光线追踪技术 [2] 实现了一个基于 OpenGL 的光线追踪渲染器，用于渲染具有复杂光照特性的 3D 场景。

II. 相关工作

光线追踪技术最早可以追溯到 20 世纪 60 年代，当时数学应用组的科学家们发明了这项技术，最初用于模拟光线的传播路径，主要用于科学研究和模拟。在 20 世纪 70-80 年代，光线追踪开始被应用于计算机图形学，但受限于当时的计算能力，它主要用于生成静态图像和动画，而非实时渲染。到了 2008 年左右，英特尔尝试将光线追踪技术商业化，希望通过 CPU 进行光线追踪计算，但由于性能需求高和开发难度大，相关产品 Larrabee 最终未能成功上市。2018 年，英伟达发布了支持光线追踪的 RTX 系列显卡，这些显卡配备了专门的 RT Core 来加速光线追踪过程。同年，微软推出了支持光线追踪的 DirectX Ray Tracing（DXR）API，标志着实时光线追踪技术进入大众视野。

光线追踪技术与传统的光栅化渲染技术在实现光照效果时有本质的不同。光栅化渲染，也称为多边形渲染，是一种在 3D 图形中常用的技术，它通过将 3D 模型转化为 2D 屏幕上的像素来实现。这种方法依赖于预先计算好的光照贴图和简单的光照模型，可以快速渲染出场景，但在真实感方面有所欠缺 [3] [4]。

光线追踪技术在电影行业 [5] 和游戏行业 [6] 中被广泛应用于视觉效果的制作，特别是在科幻和神话类影片中，许多场景是通过电脑技术制作的。例如，迪士尼和皮克斯公司的动画电影《冰雪奇缘》和《超能陆战队》等都采用了光线追踪技术来实现逼真的光影效果。

现有的光线追踪算法主要集中在如何高效地模拟光线与场景中物体的相互作用。基本的光线追踪算法包括光线与几何体的相交检测、光照模型的计算以及反射和折射的递归处理。随着技术的发展，出现了多种优化技术，如使用空间分割结构来加速交点检测，以及利用现代 GPU 的并行计算能力来提高渲染效率。

III. 方法

本文主要实现了一个基于 OpenGL 的光线追踪渲染器。在 OpenGL 中递归实现光线追踪可以模拟出非

常逼真的光照效果。

A. 方法描述

场景和模型的加载是光线追踪渲染的第一步。本文使用了 Assimp 库，实现了对.obj 格式 3D 模型的读取，如图 1，将其转换为程序能够理解的格式，并赋予材质和纹理信息。并在代码中添加 Model 类负责存储模型的多边形网格、材质属性和纹理信息，并提供加载模型的方法。

```
// 读取 obj 模型
//
//
Model tree2 = Model();
tree2.translate = glm::vec3(5, 0, 2);
tree2.scale = glm::vec3(0.02,0.02,0.02);
tree2.load("models/blacksmith/blacksmith/blacksmiths.obj");
models.push_back(tree2);


Model car = Model();
car.translate = glm::vec3(10, 0, 2);
car.scale = glm::vec3(1, 1, 1);
car.load("models/bujiadi/uploads_files_5230360_bolide.obj");
models.push_back(car);

Model house2 = Model();
house2.translate = glm::vec3(5, 0, 15);
house2.scale = glm::vec3(0.02, 0.02, 0.02);
house2.rotate = glm::vec3(0, 0, 0);
house2.load("models/house+3/house 3/twnhousel.obj");
models.push_back(house2);



Model house = Model();
house.translate = glm::vec3(5, 0, -15);
house.scale = glm::vec3(0.02, 0.02, 0.02);
house.rotate = glm::vec3(0, 0, 0);
house.load("models/church/church/church.obj");
models.push_back(house);
```

图 1. 读取 obj 模型

本文使用 GLSL（OpenGL Shading Language）编写顶点着色器和片段着色器，通过 getShaderProgram 函数编译并链接成可执行的着色器程序，如图 2。设计 Mesh 类的 bindData 方法，将顶点数据上传到 GPU，为后续的光线-物体相交测试做准备。实际的相交测试在着色器程序中实现，通过计算光线与几何体的交点来确定光线是否击中物体。

通过在顶点着色器和片段着色器中实现 Phong 模型，计算每个像素的光照效果。在着色器程序中添加递归逻辑，模拟光线在物体表面的反射和穿过透明物体的行为。

为了增强视觉效果，实现纹理映射技术，将 2D 纹理映射到 3D 模型表面，本文采用 SOIL_load_image 函数加载纹理图像，并将其应用于模型表面。同时，使用 MIP

```
// 获取着色器对象
GLuint getShaderProgram(std::string fshader, std::string vshader)
{
    // 读取shader源文件
    std::string vSource = readShaderFile(vshader);
    std::string fSource = readShaderFile(fshader);
    const char* vpointer = vSource.c_str();
    const char* fpointer = fSource.c_str();

    // 容错
    GLint success;
    GLchar infoLog[512];

    // 创建并编译顶点着色器
    GLuint vertexShader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(vertexShader, 1, (const GLchar**)(&vpointer), NULL);
    glCompileShader(vertexShader);
    glGetShaderiv(vertexShader, GL_COMPILE_STATUS, &success);   // 错误检测
    if (!success)
    {
        glGetShaderInfoLog(vertexShader, 512, NULL, infoLog);
        std::cout << "顶点着色器编译错误\n" << infoLog << std::endl;
        exit(-1);
    }

    // 创建并且编译片段着色器
    GLuint fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fragmentShader, 1, (const GLchar**)(&fpointer), NULL);
    glCompileShader(fragmentShader);
    glGetShaderiv(fragmentShader, GL_COMPILE_STATUS, &success);   // 错误检测
    if (!success)
    {
        glGetShaderInfoLog(fragmentShader, 512, NULL, infoLog);
        std::cout << "片段着色器编译错误\n" << infoLog << std::endl;
        exit(-1);
    }
```

图 2. 着色器

贴图技术，为纹理的不同层次提供细节，提高渲染效率。在 OpenGL 的渲染循环中，采用 glut_display_func 函数，实现实时渲染流程。

B. 实验设置

硬件配置: 所有实验在配置有 AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz 处理器、NVIDIA RTX 3060 显卡的计算机上进行。

软件环境: 使用 Visual Studio 2022 作为开发环境，以及 C++ 标准库，GLEW（OpenGL 扩展加载库），FreeGLUT（OpenGL 实用工具库），GLM（OpenGL 数学库），SOIL2（图像加载库），Assimp（模型导入库），ImGUI（GUI 库）等库的支持

测试环境: 包含多个物体（如汽车、房子）的场景，测试基本的光线追踪效果和性能。

IV. 实验结果与分析

A. 实验结果

测试结果如图 3 所示

B. 实验分析

实验结果表明，光线追踪技术能够提供高质量的渲染效果，特别是在模拟复杂光照和材质方面。然而，在

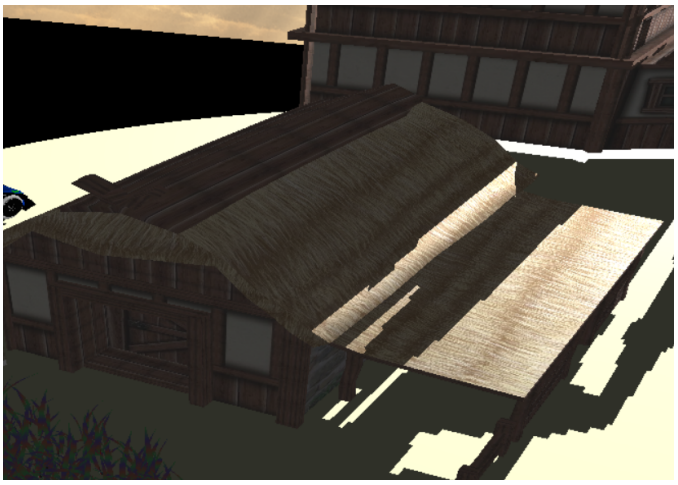

图 3. 在测试中，光线追踪技术成功地渲染出了真实的光影效果，但由于场景复杂，渲染时间达到 3 分钟。

处理复杂场景时，尽管光线追踪提供了高质量的渲染效果，但其计算成本也相应较高。

V. 讨论与展望

综上所述，基于 OpenGL 的递归光线追踪技术能够生成逼真的渲染效果，但同时也面临着性能和资源消耗方面的挑战。在未来我将会继续优化算法，进一步研究和开发更高效的光线追踪算法，研究适合实时渲染的光线追踪技术，同时改进资源管理策略，优化内存和计算资源的使用，以适应大规模场景的渲染需求。

参考文献

[1] 徐汝彪, 刘慎权, 光线追踪技术综述, 计算机研究与发展 27 (5) (1990) 9.

[2] Z. You, Fundamental ray tracing project based on opengl with acceleration, Journal of Physics: Conference Series 2646 (1) (2023) 012001. doi:10.1088/1742-6596/2646/1/012001.
URL https://dx.doi.org/10.1088/1742-6596/2646/1/012001

[3] 蔡至诚, 混合实时渲染关键技术研究.

[4] 符鹤, 谢永芳, Tbr 架构 gpu 中三角形高效光栅化., Journal of Image & Graphics 20 (4) (2015).

[5] 白苏琼, 程万里, 基于光线追踪及碰撞检测的游戏动画动态仿真模型研究, 自动化与仪器仪表 (05) (2024) 168–171+177. doi:10.14016/j.cnki.1001-9227.2024.05.168.

[6] 张健浪, 游戏的未来: 光线追踪技术步入实用化, 新电脑 31 (7) (2007) 52–56.