| 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） |
|---|---|---|---|---|---|---|
| 21301085 | 张佳硕 | 7 | 24 | 16 | 33 | 80 |

缺乏参考文献

北京交通大学

# 计算机图形学课程论文

# 基于 MATLAB 实现模型的光线追踪


学　　院：软件

专　　业：软件工程

学生姓名：张佳硕

学　　号：21301085



北京交通大学

2024 年 6 月 21

# 中文摘要

**摘要：**本论文研究了光线追踪技术的理论基础，并基于 MATLAB 实现了一个简单的光线追踪模型。论文首先介绍了光线追踪的基本原理，包括计算光线与物体的交点、光照效果的计算以及反射和折射的计算方法。接着，描述了如何在 MATLAB 中实现这些理论，并通过实验验证了该实现的效果。实验结果显示，光线追踪能够生成高质量的图像，模拟复杂的光影效果，验证了其在计算机图形学中的重要应用价值。

**关键词:** 光线追踪，MATLAB，计算机图形学，光照模型，反射，折射

# ABSTRACT

**ABSTRACT:** This thesis explores the theoretical foundations of ray tracing technology and implements a basic ray tracing model using MATLAB. Ray tracing is a technique for generating realistic images by simulating the path of light rays. The thesis first introduces the fundamental principles of ray tracing, including the calculation of intersections between rays and objects, computation of lighting effects, and calculation of reflection and refraction. Then, it details the implementation of these theories in MATLAB and validates the effectiveness of the implementation through experiments. The experimental results demonstrate that the ray tracing algorithm can generate high-quality images and simulate complex lighting effects, confirming its significant application value in computer graphics.

**KEYWORDS：** MATLAB, computer graphics, lighting model, reflection, refraction

# 目录

# 1 绪论

## 1.1 研究背景

《计算机图形学》课程结课论文要求针对计算机图图形学领域的任一技术方向展开调研，了解相关技术的研究现状并进行应用，并对实践结果进行讨论和分析。我感兴趣的计算机图形学领域是光线追踪，所以我对光线追踪这一技术的研究现状进行了了解，并使用 MATLAB 书写了一小段程序应用了光线追踪技术，最终对光线追踪有了更加深入的了解。

## 1.2 光线追踪在国内外的研究现状

渲染是将三维场景转换为二维图像的过程。渲染主要包括光栅化和光线追踪两种方式

### 1.2.1 光线追踪和光栅化

光栅化采用局部光照原理，根据光源照射物体的光照效果，将场景中的几何图元映射到图像的像素点上。光线追踪通过模拟光线的传播路径来生成逼真的图像。其基本原理是从观察者的视点出发，追踪每一条光线的路径，直到它们与场景中的物体相交。然后，根据光线与物体的交点计算出该点的颜色和亮度，考虑光源、反射、折射和阴影等因素。

### 1.2.2 研究现状

1979 年，Whitted 利用光线的可逆性，提出了光线追踪算法。这一算法模拟了真实的光照效果，通过在场景中光线与物体相交后，因反射、折射和阴影生成二次光线，并再次在场景中进行相交测试。如此迭代计算，直至光线的反弹次数达到预设的阈值。

光线追踪技术经过多年的发展，已逐渐摆脱传统光栅化渲染管线中的 Phong-Blinn 局部光照模型。现代光线追踪技术基本建立在基于 BRDF 函数的路径追踪(Path Tracing)及其优化方法上。此外，在一些高效团队作品中，还会通过构建全局三维有向距离场（Signed Distance Field, SDF）来实现光线追踪。

2012 年，NVIDIA 公司的工程师 Karras T 提出了一种最大化并行构建 BVH(Bounding Volume Hierarchy, 层次包围盒）的方法。该方法利用莫顿码的空间 Z 字型遍历顺序，通过固定的三次查询操作，实现了八叉树和 K-D Tree 的并行构建，具有极高的执行并

行性与指令并行性，是首个使用并行方式构建 BVH 的方法。

2014 年，德国拜罗伊特大学的学者 Guthe M 提出了一种深度优先的低延迟 BVH 遍历方式。该方法采用独特的四方向隐藏延迟的遍历方式，以硬件资源的牺牲为代价，进一步提高了运行的并行性。

2016 年，NVIDIA 公司的工程师 Binder N 和 Keller A 提出了一种无栈的 BVH 遍历方式。该方法针对满二叉树，具有固定的回溯时间，在系统稳定性和指令并行性方面具有诸多优势。

2018 年，NVIDIA 公司的工程师 Ylitie H 等人提出了一种压缩宽 BVH 的高效非相关性光线遍历方式，同时参考了 2012 年同公司提出的莫顿码遍历顺序优化方法，成为了目前 GPU 平台上最快的光线追踪技术。

2019 年，Intel 公司的工程师 Vaidyanathan K 等人设计了一种短栈的宽 BVH 遍历方式，将栈空间的使用量压缩到传统方法的 1/3 以下，解决了 CPU 平台光线遍历效率较低的问题，为 CPU 参与光线追踪技术的协同发展提供了保障。

由于国内缺乏世界领先的芯片企业，或相关企业起步较晚，国内在光线追踪这类高度依赖硬件设备的技术研究进展上几乎没有突出贡献。此外，由于国外功能强大的商用软件普及率较高，国内少有研究人员对光线追踪的关键技术进行深入研究，相关文献所体现的技术效果也与上述内容相差甚远。

## 1.3 本文的主要内容

本文的主要内容是通过 MATLAB 实现最基本的光线追踪渲染模型，旨在通过实践深刻理解光线追踪的理论基础及其实现过程。具体内容包括：

1）光线追踪的理论基础

2）MATLAB 实现简单的光线追踪

3）实验总结

# 2 基于 MATLAB 实现模型的光线追踪

## 2.1 光线追踪的理论基础

1）计算光线与物体的交点：

判断光线是否与场景中的物体相交以及交点位置。以球体为例其中c为球心，r为半径：

$$\|P - c\|^2 = r^2$$

光线的参数方程为，其中 o 为光线起点，d 为光线方向：

$$p(t) = o + td$$

将光线方程代入到球体方程得到：

$$|o + td - c|^2 = r^2$$

展开并整理：

$$(d \cdot d)t^2 + 2\big(d \cdot (o - c)\big)t + (o - c) \cdot (o - c) - r^2 = 0$$

展开并整理解方程得到光线和球体的交点，如果 t 为正表示光线与球体相交

2）光照效果的计算：

环境光：$I_{ambient} = k_a I_a$

反射光：$I_{diffuse} = k_d (L \cdot N) I_l$

镜面反射光：$I_{specular} = k_s (R \cdot V)^n I_l$

3）反射和折射的计算：

反射光线的方向由入射光线和表面法线决定，计算公式为：

$$R = D - 2(D \cdot N)N$$

根据斯涅尔定律（Snell's Law）计算折射光线的方向：

$$\eta_1 \sin\theta_1 = \eta_2 \sin\theta_2$$

## 2.2 MATLAB 实现光线追踪

1）定义图像参数和场景参数：

图像分辨率设置为 800x600。设置相机视场角（FOV）和图像宽高比。定义场景中的球体参数，包括位置、半径、颜色、镜面反射系数、反射率、透明度和折射率。定义光源的位置和颜色。

2）光线生成和方向计算：

对于图像中的每个像素，计算相应的光线方向。根据相机的视场角和图像宽高比，调整光线方向。

3）光线追踪函数：

trace_ray 函数递归计算光线在场景中的颜色，包括处理反射和折射。若达到最大递归深度，则返回背景色。

intersect_scene 函数计算光线与场景中物体的交点，返回最近的交点信息。

intersect_ray_sphere 函数判断光线与球体的交点，并计算交点位置。

compute_lighting 函数计算交点处的光照，包括环境光、漫反射光和镜面反射光。

reflect_ray 函数计算反射光线的方向。

refract_ray 函数计算折射光线的方向。

4）生成图像并显示：

遍历图像中的每个像素，调用 trace_ray 函数计算颜色值。使用 imshow 函数显示生成的图像。

## 2.3 核心代码

```
% 追踪光线函数
function color = trace_ray(origin, direction, spheres, light_position, light_color, depth)
    if depth <= 0
        color = [0, 0, 0];   % 超过递归深度返回背景色
        return;
    end

    [nearest_t, nearest_sphere] = intersect_scene(origin, direction, spheres);

    if isempty(nearest_sphere)
        color = [0, 0, 0];   % 没有交点返回背景色
        return;
    end

    % 计算交点和法线
    hit_point = origin + nearest_t * direction;
    normal = normalize(hit_point - nearest_sphere.center);

    % 光照计算
    local_color = compute_lighting(hit_point, normal, -direction, spheres, light_position, light_color,
nearest_sphere.specular);

    % 反射和折射处理
    reflection = [0, 0, 0];
    if nearest_sphere.reflective > 0
        reflected_dir = reflect_ray(-direction, normal);
        reflection = trace_ray(hit_point, reflected_dir, spheres, light_position, light_color, depth - 1);
    end

    refraction = [0, 0, 0];
    if nearest_sphere.transparency > 0
        refracted_dir = refract_ray(-direction, normal, 1.0, nearest_sphere.refractive_index);
        refraction = trace_ray(hit_point, refracted_dir, spheres, light_position, light_color, depth - 1);
    end
```

```
    % 组合颜色
    color = (1 - nearest_sphere.reflective) * local_color + nearest_sphere.reflective * reflection +
nearest_sphere.transparency * refraction;
end

% 场景交点计算
function [nearest_t, nearest_sphere] = intersect_scene(origin, direction, spheres)
    nearest_t = inf;
    nearest_sphere = [];
    for i = 1:length(spheres)
        [t1, t2] = intersect_ray_sphere(origin, direction, spheres(i));
        if t1 < nearest_t && t1 > 0
            nearest_t = t1;
            nearest_sphere = spheres(i);
        end
        if t2 < nearest_t && t2 > 0
            nearest_t = t2;
            nearest_sphere = spheres(i);
        end
    end
end

% 光线与球体的交点计算
function [t1, t2] = intersect_ray_sphere(origin, direction, sphere)
    oc = origin - sphere.center;
    a = dot(direction, direction);
    b = 2 * dot(oc, direction);
    c = dot(oc, oc) - sphere.radius^2;
    discriminant = b^2 - 4 * a * c;
    if discriminant < 0
        t1 = inf;
        t2 = inf;
    else
        t1 = (-b - sqrt(discriminant)) / (2 * a);
        t2 = (-b + sqrt(discriminant)) / (2 * a);
    end
end

% 光照计算
function intensity = compute_lighting(point, normal, view, spheres, light_position, light_color, specular)
    intensity = [0, 0, 0];

    % 环境光
    intensity = intensity + 0.1 * [1, 1, 1];

    % 光源方向
    light_dir = normalize(light_position - point);

    % 漫反射
    dot_nl = dot(normal, light_dir);
    if dot_nl > 0
        intensity = intensity + light_color * dot_nl;
    end

    % 镜面反射
    if specular > 0
        reflect_dir = reflect_ray(-light_dir, normal);
        dot_rv = dot(reflect_dir, view);
        if dot_rv > 0
            intensity = intensity + light_color * (dot_rv^specular);
        end
    end
end

% 反射光线计算
```

```
function reflected = reflect_ray(direction, normal)
    reflected = direction - 2 * dot(direction, normal) * normal;
end

% 折射光线计算
function refracted = refract_ray(direction, normal, eta1, eta2)
    eta = eta1 / eta2;
    cosi = -dot(direction, normal);
    sint2 = eta^2 * (1 - cosi^2);
    if sint2 > 1
        refracted = [0, 0, 0];   % 全反射
    else
        cost = sqrt(1 - sint2);
        refracted = eta * direction + (eta * cosi - cost) * normal;
    end
end
```

## 2.4 实验结果分析

通过 MATLAB 实现的光线追踪算法，可以看到程序生成了多个球体使用光线追踪渲染之后的场景。该实验验证了光线追踪算法在生成高质量图像和模拟复杂光影效果方面的有效性。

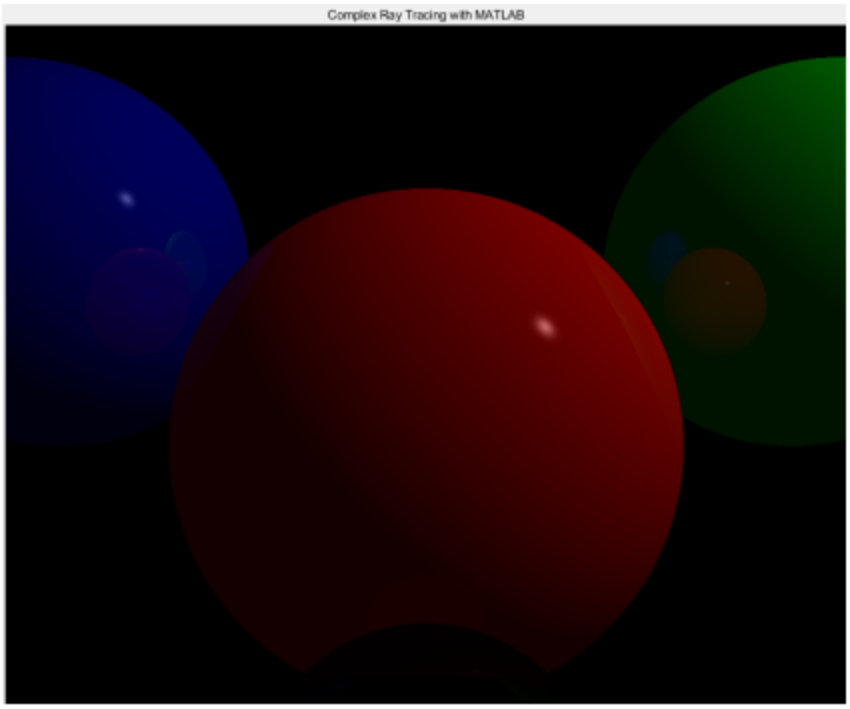
1）颜色和光影效果：

红、绿、蓝三个球体在光源照射下表现出明显的光影效果，能够看到高光反射点，这得益于较高的镜面反射系数（500）。

2）反射效果：

各个球体根据其反射率展示出不同程度的反射效果。例如，蓝色球体的反射率最高（0.4），反射出的环境颜色较多。

3）实验结果截图：



## 2.4 本章总结

本章介绍了光线追踪的理论基础。包括计算光线与物体交点、光照效果以及反射和折射的计算方法。具体内容涵盖了如何判断光线与球体的交点位置，以及根据 Phong 反射模型计算环境光、漫反射光和镜面反射光。

然后，详细描述了如何使用 MATLAB 实现一个简单的光线追踪模型。该实现包括定义图像参数和场景参数、光线生成和方向计算、光线追踪函数的递归调用、交点计算、光照计算、反射光线和折射光线的方向计算，以及最终生成图像并显示的过程。

通过实验结果分析，可以看到光线追踪算法在生成逼真图像方面的强大能力。为后续更负责的光线追踪渲染提供了基础。

# 3 总结

本论文详细探讨了光线追踪技术的理论基础和实际应用。通过对光线追踪原理的深入研究，我理解了光线与物体交点的计算、光照效果的模拟以及反射和折射的处理方法。

基于这些理论，我在 MATLAB 中实现了一个简单的光线追踪模型，并进行了实验验证。实验结果显示，该模型能够生成逼真且高质量的图像，有效模拟了光照、反射和折射等复杂光影效果，验证了光线追踪算法的有效性。红、绿、蓝球体在光源照射下表现出明显的光影效果和高光反射点，这些实验结果证明了光线追踪技术在生成逼真图像方面的强大能力。

本研究加深了我对光线追踪技术的理解，还展示了 MATLAB 作为工具在计算机图形学领域中的潜力和应用价值。光线追踪技术虽然计算复杂度高，但其生成的图像质量和逼真效果为其在计算机图形学中的重要地位提供了有力支持。

未来的研究可以进一步优化光线追踪算法，提高其计算效率，并探索其在更复杂场景中的应用。通过本次研究，我们对光线追踪技术有了全面的了解，为进一步研究和应用奠定了坚实的基础。我们相信，随着计算能力的不断提升，光线追踪技术将在更多领域得到广泛应用，并不断推动计算机图形学的发展。