光线追踪实验报告

周旭东

摘　要: 在本次光线追踪项目中，基于 GAMES101 框架实现了 Path Tracing 算法，并通过随机选择微表面来模拟毛玻璃表面的粗糙程度引入了毛玻璃和使用菲涅尔反射、漫反射和镜面反射的合理组合生成的金属材质效果，最终渲染出具有反射，折射，阴影等多种效果的逼真图像。为了评估渲染结果的质量，进行了多次渲染，采用不同的采样率（SPP）来观察图像逐渐收敛的过程。在 Cornell Box 场景中包含了交大校徽和图书馆模型，这使得渲染结果更加具有挑战性和真实感。本人通过这个项目，深入理解了光线追踪的核心概念，并成功将其应用于实际场景。实现毛玻璃和金属材质效果不仅拓展了渲染效果的多样性，也提高了场景的视觉真实感。这一成果使得本人对前沿光线追踪渲染有了深刻了解，在图形学和渲染技术方向的学术研究奠定了基础。

关键词: 光线追踪，路径追踪，康奈尔盒

Ray Tracing Experiment Report

Zhou Xudong

Abstract: In this ray tracing project, the Path Tracing algorithm was implemented based on the GAMES101 framework. The simulation of frosted glass surfaces was achieved by randomly selecting microfacets to emulate the roughness of the glass surface. Additionally, the project introduced the rendering of metal materials by employing a reasonable combination of Fresnel reflection, diffuse reflection, and specular reflection. The final renderings exhibited realistic images with various effects, including reflection, refraction, and shadows. To assess the quality of the renderings, multiple iterations were performed with different sample rates (SPP) to observe the gradual convergence of the images. The scenes included the logo of the university and a library model in Cornell Box, adding complexity and realism to the rendering challenges. Through this project, I gained a profound understanding of the core concepts of ray tracing and successfully applied them to practical scenes. The implementation of frosted glass and metal material effects not only diversified the rendering outcomes but also enhanced the visual realism of the scenes. This achievement deepened my insights into cutting-edge ray tracing rendering techniques and laid a solid foundation for academic research in the fields of computer graphics and rendering technology.

Key word: Raytracing, Path Tracing, Cornell Box

1 项目简介/Introduction

1.1 项目意义

光线追踪是计算机图形学和计算机图像学领域中一项重要的研究工作，具有深远的意义和广泛的应用。以下是光线追踪研究的一些意义：（1）逼真图像的生成： 光线追踪是一种能够生成高度逼真图像的算法。通过模拟光线在场景中的传播，光线追踪能够模拟真实光照、阴影、反射、折射等光学效果，生成更接近真实世界的图像。（2）视觉效果的提升： 在电影制作、游戏开发等领提高视觉效果。光线追踪使得图像具有更真实的光照和阴影效果，提升了观众的沉浸感和感知质量。这对于创造引人入胜的虚拟环境、影片场景或游戏场景至关重要。（3）艺术创作与虚拟现实： 光线追踪在艺术创作和虚拟现实领域发挥着关键作用。艺术家可以使用光线追踪技术来创造更具艺术感的场景和效果，而虚拟现实则借助光线追踪提供更逼真的虚拟体验。（4）产品设计与模拟： 在工程和产品设计中，光线追踪可以用于模拟光照对产品外观的影响。例如，对于汽车、建筑、珠宝等产品，光线追踪可以帮助设计师预测并优化产品的外观，提高设计质量。

在科研领域，有以下应用：（1）科学可视化： 在科学研究领域，光线追踪被用于可视化复杂的科学数据。科学家可以通过光线追踪技术更清晰地呈现数据，使得数据集更容易理解，发现隐藏在数据背后的模式和趋势。（2）计算机图形学研究： 光线追踪作为计算机图形学中的一个重要研究方向，推动了图形学的发展。通过研究光线追踪算法，研究者们不仅改进了图形学领域的渲染技术，还推动了计算机图形学与计算机视觉等领域的交叉研究。（3）计算机图形硬件发展： 光线追踪的需求促进了图形硬件的发展。实时光线追踪对于计算能力和硬件性能提出了挑战，激发了图形处理器（GPU）等硬件设备的创新和升级。

光线追踪的研究具有推动计算机图形学发展、提高视觉效果、应用于多个领域的潜力。其在科学、工程、艺术和娱乐等多个领域的广泛应用，使其成为计算机图形学研究中备受关注的前沿技术。

1.2 系统框架

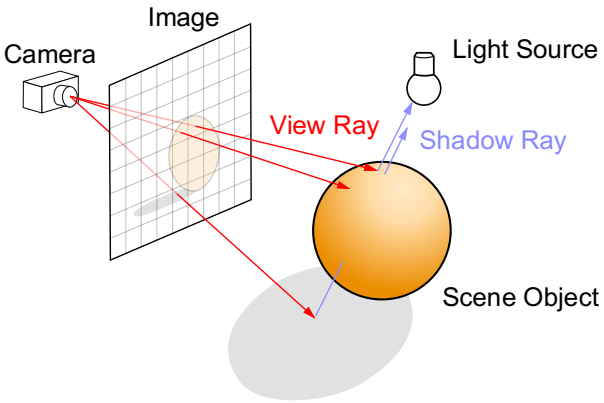


图 1 光线追踪基本框架[1]

本系统参考 Games101 [3] 光线追踪算法，采用 Path Tracing，其过程在参考图 1 基础上为：

1. **发射光线（Ray Generation）：** 从相机位置发射一条光线，每个像素对应一个初始光线。这条光线的方向通常是由相机到图像平面上的像素位置确定的。
2. **光线与场景的交点（Ray-Object Intersection）：** 对于每条光线，检查它是否与场景中的物体相交。这通常涉及与场景中的几何体（如球体、三角形等）进行求交操作。
3. **表面属性（Surface Properties）：** 对于光线与物体相交的点，获取该点的表面属性，如法线、颜色、纹理等。这些属性用于计算光照。
4. **发射新光线（Ray Scattering）：** 根据材质的属性（如反射率、折射率等），决定在该交点处发射新的光线。这可以是反射光线、折射光线，或者在漫反射材质上选择随机方向的光线。
5. **能量传递（Energy Transport）：** 对于发射出的新光线，递归地重复上述过程，从而模拟光线在场景中的传播。递归可能会在达到最大反射次数或遇到光源时终止。
6. **计算光照（Lighting Calculation）：** 在每次光线与物体相交时，根据光照模型计算该点的颜色。这通常涉及到光照方程的计算，包括漫反射、镜面反射、折射等。
7. **累积颜色（Color Accumulation）：** 在路径追踪的每一步，将计算得到的颜色值累积到最终的像素颜色中。这是一个迭代的过程，每次光线追踪过程都会为像素的颜色贡献一部分。
8. **图像输出（Image Output）：** 在所有像素的颜色计算完成后，将最终的颜色信息输出为图像。

2 计算方法/Methods

2.1 CastRay算法

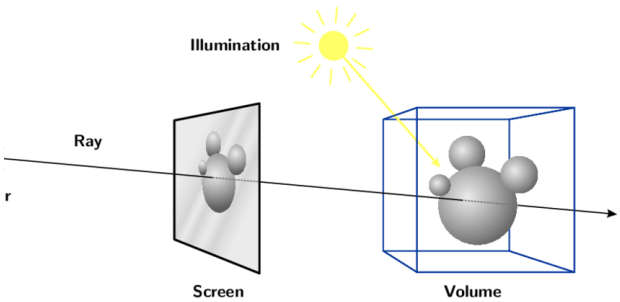


图 2 **Raycasting 模型[3]**

渲染器中涉及 CastRay，场景相交检测如图 2： 通过调用 scene.intersect(ray, culling)，获取光线与场景的相交信息。如果没有相交，即 !intersection，则返回场景的背景颜色。获取相交信息： 从相交信息中提取出着色点的位置 pos、法线 normal、观察方向 observer_dir 和材质

属性 mat_ptr。直接光照计算： 首先初始化直接光照贡献 i_direct 为零。然后对场景中的光源进行采样，计算从着色点到光源采样点的方向，并检查二者之间是否有阻挡物体。如果没有阻挡，进行光源的重要性采样和材质的重要性采样，并使用平衡启发式方法计算直接光照的贡献。间接光照计算： 初始化间接光照贡献 i_indirect 为零。通过俄罗斯轮盘赌（Russian Roulette）来控制递归深度。如果随机数小于轮盘赌的概率，进行间接光照的计算。对间接光源方向进行材质的重要性采样，并递归调用 cast_ray 函数获取间接光照贡献。返回结果： 将直接光照和间接光照的贡献相加，作为最终的颜色结果返回。

该算法通过直接光照和间接光照的计算，模拟了光在场景中的传播过程，考虑了光源、阴影、反射和折射等现象，以生成逼真的图像。算法中使用了重要性采样和俄罗斯轮盘赌等技术，以提高渲染效率和质量。

2.2 材质定义

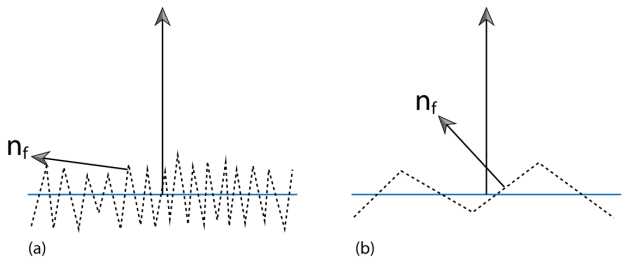


图 3 **微表面模型定义[4]**

首先定义微表面（Microfacet）模型如图 3，用于模拟材质表面微小的凹凸结构。主要包括以下功能：Microfacet 分布函数 (distribution)： 计算微表面的分布，描述凹凸结构的粗糙度。Fresnel-Schlick 反射率近似 (fresnel_schlick)： 使用 Schlick's 近似计算微表面的反射率。Microfacet 几何函数 (geometry)： 模拟微表面的遮挡和阴影效应。采样微表面法线 (sample_micro_surface)： 在微表面法线分布上进行重要性采样，生成微表面法线的随机方向。Microfacet 法线分布函数的概率密度函数 (pdf_micro_surface)： 为光线追踪的重要性采样提供微表面法线方向的概率密度。计算微表面法线方向 (outward_micro_surface_normal)： 根据入射和出射光线以及表面朝向，计算微表面法线的方向。反射微表面的 Jacobian (reflect_jacobian)：用于在重要性采样中调整反射方向的采样概率。折射微表面的 Jacobian (refract_jacobian)： 用于在重要性采样中调整折射方向的采样概率。在此基础上，定义毛玻璃和粗糙金属材质以呈现反射与折射效果。

增加以下结构进行判断：是否自发光（emitting）：根据材质的辐射强度判断是否自发光，返回辐射强度的平方是否大于零。自发光颜色（emission）：返回材质的自发光颜色。采样光线源方向（sample_ray_source_dir）：根据微表面模型采样微表面法线，然后使用反射计算光线源方向，用于追踪光线的反射路径。计算采样概率密度函数（pdf）：根据微表面法线和反射方向计算采样概率密度函数，用于重要性采样。计算贡献（contribution）：根据光线源方向、观察方

向和表面法线，结合微表面模型的分布等，计算出漫反射和镜面反射的贡献。

2.3 包围体层次结构求交

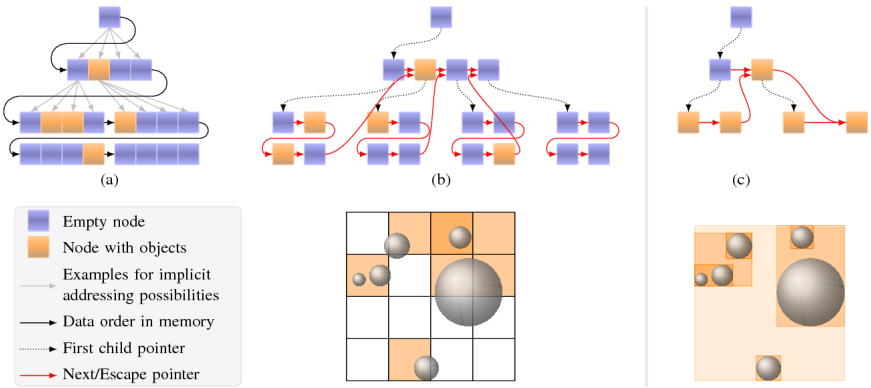


图 4 **BVH 求交模型[5]**

包围体层次结构（Bounding Volume Hierarchy，BVH）如图 4 的构建和射线求交实现的主要功能和流程为，BVH_tree 类定义：定义了 BVH_tree 类，包括默认构造函数和其他两个构造函数，其中一个用于从给定的对象列表构建 BVH。提供了求交和采样的功能。BVH_node 结构体定义：定义了 BVH_node 结构体，表示 BVH 树的节点。每个节点可以是内部节点（包含子节点）或叶子节点（包含一个对象）。每个节点存储了该节点的边界框（Bounding Box）以及节点的表面积。BVH_node 结构体具有递归的定义，其中左右子节点可以是新的 BVH_node 结构体。构建 BVH 的核心函数 recursive_build：通过递归构建 BVH 树。对于给定的对象列表，该函数在每个递归步骤中选择轴并在轴上分割对象。分割的方法有 Naive 方法和 SAH（Surface Area Heuristic）方法。使用轴对对象进行排序，然后递归地构建左右子树，直到满足递归结束条件。对于叶子节点，创建一个包含单个对象的节点。Naive 分割函数 naive_partition：采用 Naive 分割方法的一种实现，它直接将对象列表平均分割成左右两部分，然后递归构建 BVH。SAH 分割函数 sah_partition：采用 SAH 分割方法的一种实现，它计算在每个分割位置上的表面积，并选择最小代价的位置进行分割。求交函数 intersect：提供了用于在 BVH 树中寻找光线和对象相交的递归函数。从根节点开始，递归地遍历 BVH 树，检查光线和节点边界框是否相交。如果相交，进一步遍历左右子节点或叶子节点，直到找到相交的对象。采样函数 sample：提供了在 BVH 树中采样的递归函数。根据节点的表面积和阈值，以概率的方式选择是左子节点还是右子节点进行采样。

2.4 多线程渲染算法

在本地运行单线程光线追踪算法速度过慢，采用多线程加速算法，构成为——初始设置：在每个渲染线程中，首先计算了场景相关的参数，如视场角（fov）、屏幕宽高比、像素总数等。并行渲染循环： 算法通过总线程数和当前线程编号，将像素分配给各个渲染线程，以避免互斥锁的使用。每个线程负责渲染一部分像素，通过循环遍历这部分像素，计算它们的颜色。像

素采样： 对于每个像素，执行了多次采样（spp 次），每次采样都生成了主射线的方向。主射线方向的生成采用随机偏移，以模拟抗锯齿效果。生成的主射线通过调用 cast_ray 函数执行光线追踪，计算颜色贡献。光线追踪： 在 cast_ray 函数中，光线与场景相交，计算直接光照和间接光照。如果相交点的材质具有自发光属性，则直接采集该颜色作为贡献。结果合并： 每个像素的多次采样颜色累加后取平均值，最终得到该像素的最终颜色。进度更新： 如果当前线程是第一个线程（线程编号为 0），则在每行渲染完成时更新渲染进度。

3 工程处理过程/ Engineering process

3.1 模型数据预处理

普通 obj 文件带有材质，法线等多种信息。本系统中仅需要全三角形的顶点与面信息。故使用 python 制作预处理工具，算法流程为：打开文件：使用 open 函数打开指定路径的输入文件（OBJ 格式的三维模型文件）。遍历文件内容：通过 for 循环遍历文件的每一行。解析顶点信息：如果行以字母"v"开头，表示该行包含顶点信息。解析顶点坐标，将其乘以给定的缩放因子，然后添加到顶点列表中。解析三角形面信息：如果行以字母"f"开头，表示该行包含三角形面的信息。将包含法线和材质信息的面转换为只包含顶点索引的形式。如果面是多边形，则将其转换为若干个三角形，存储在三角形列表中。写入输出文件：使用 open 函数创建输出文件，然后按照 Wavefront OBJ 格式的规定，将缩放后的顶点和转换后的三角形面写入输出文件。

3.2 模型载入

初始化场景空间时用于从模型文件加载三角形网格并构建三角形网格的 BVH 树。下面是对这两个函数的详细描述：ad_triangles_from_model_file 函数：这个函数通过使用 objl 库加载模型文件。需要确保已经引入了该库。函数的参数包括模型文件名 file_name、三角形网格的材质指针 mat_ptr、缩放因子 scale_frac、平移向量 translation、绕定定轴旋转的旋转轴 rotation_axis 以及旋转角度 rotation_angle。通过 objl::Loader 对象加载模型文件，获取第一个网格信息 objl::Mesh mesh = loader.LoadedMeshes[0]。对每个三角形，依次处理每个顶点：应用缩放、平移和旋转操作，得到最终的顶点坐标。将三个顶点的坐标传递给 Triangle 类的构造函数，创建一个三角形对象。将创建的三角形对象指针存入 triangle_ptrs 向量中。最后，返回包含所有三角形对象指针的向量 triangle_ptrs。

TriangleMesh 类的构造函数：这个构造函数接受一个三角形对象指针的向量 triangle_ptr_list 和一个枚举值 BVH_tree::SplitMethod split_method。构造函数首先将三角形对象指针向量 _triangle_ptrs 初始化为传入的 triangle_ptr_list。然后，通过调用 transform_to_object_vector 将三角形对象指针向量转换为对象指针向量，并使用该向量构建 BVH 树 _bvh_tree。遍历 _triangle_ptrs 向量，检查是否有发光的三角形。如果有至少一个三角形是发光的，将 _emitting 标志设置为 true。最终，完成了对 TriangleMesh 对象的初始化。

这两个函数协同工作，load_triangles_from_model_file 负责从文件中加载三角形，并且将其进行缩放、平移和旋转，最后构建 Triangle 对象。TriangleMesh 构造函数则接收这些三角形对

象，构建 BVH 树，并检查是否有发光的三角形。

3.3 场景生成和渲染

主要场景生成和渲染经过以下几个过程：场景创建： 创建一个场景对象，设置图像的分辨率和相机的位置与视野角。常量定义： 定义一些常量，如每像素的采样次数和渲染中使用的总线程数。材质定义： 创建不同的材质对象，包括发光材质、漫反射材质、金属粗糙材质和玻璃材质。物体创建： 使用加载模型的函数，加载模型文件并为每个模型设置相应的材质、位置和旋转等属性，创建三角形网格和球体对象。物体添加到场景： 将创建的三角形网格和球体对象添加到场景中。BVH 树构建： 对场景中的所有物体（三角形网格和球体）构建 BVH 树以加速光线追踪。渲染器创建： 创建一个渲染器对象。场景渲染： 使用渲染器对场景进行光线追踪渲染，设置每个像素的采样次数和总线程数。输出：输出渲染完成的信息"Complete!"。

4 实验结果与分析/Experiment Results and Analysis

4.1 效果展示分析

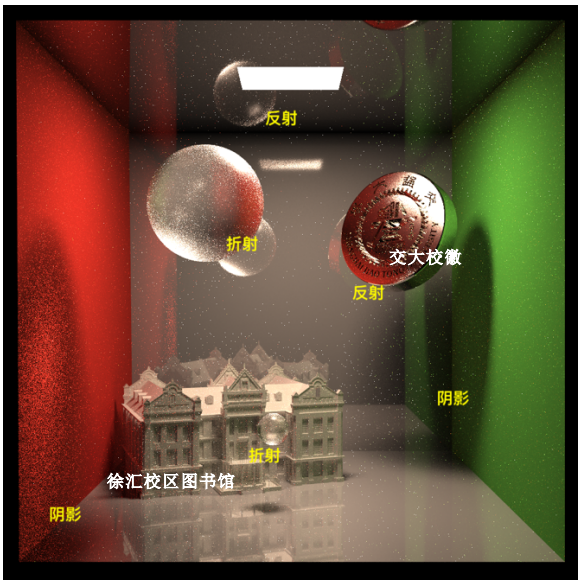
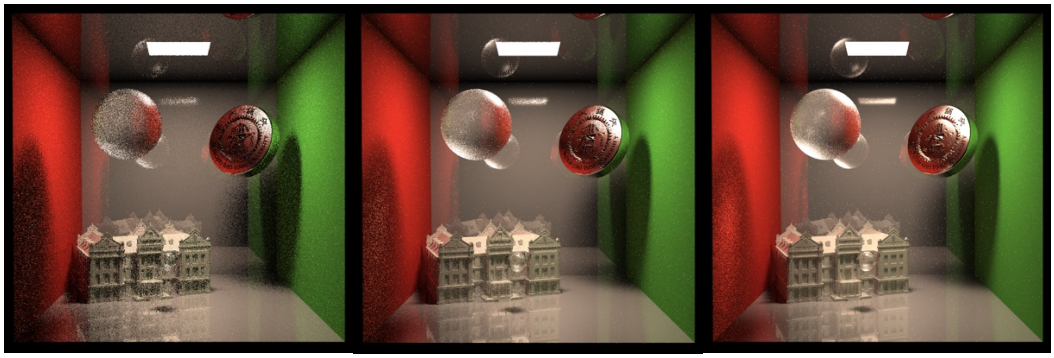


图 5 **渲染最终结果**

最终场景中由更换了墙面材质的 Coenell Box 与至于其中的金属材质交大校徽和漫反射材质交大徐汇校区图书馆模型组成。在渲染图 5 中可以清晰地看到光线追踪产生的各部分效果，包括反射:底部，背部，顶部墙面产生的镜面效果；交大金属校徽对红色（左面）和绿色（右面）墙面的精准反光。折射：左上角毛玻璃球折射出左面红色墙壁，中间偏下方在图书馆模型前的毛玻璃球也投射出其后的模型。阴影：从顶部光源向下，毛玻璃球对光源存在中部透过较多，边缘透过较少的特点，交大校徽则在角落呈现出很好的软阴影效果。

从上述分析看出，本次实验效果良好。

4.2 不同SPP渲染效果对比



(a)SPP=2　　(b)SPP=16　　(c)SPP=128

图 6 不同 SPP 渲染质量对比

图 6(a)-(c)展示了不同 SPP（Samples Per Pixel，每像素采样数）下渲染结果。从时间上来看，渲染用时和 SPP 几乎成正比关系；从效果上来看，随着 SPP 加大，画面噪点大幅缩减；交大校徽呈现的精度逐渐变好；毛玻璃球的表面越发光滑；阴影效果变浅但更加准确，透明球体带来的阴影展现更加透彻和精准。

5 特色与创新/Distinctive or Innovation Points

本项目在 GAMES101 的代码框架基础上进行了 obj 模型预处理，材质构建，CastTracing 算法构建，BVH 求交与多线程渲染等工作。在传统 Cornell Box 场景中加入反射材质和交大校徽，交大徐汇校区图书馆等特色模型，最终取得不错的渲染成果并进行了 SPP 横向结果对比。

在本项目的基础上，还可以进行的探索和尝试有化材质系统，支持更多种类的纹理映射和复杂的反射折射效果，实现更高效的光线追踪算法，以及实现更多的图形学特效，如运动模糊、景深等，以提升渲染画面的艺术效果。可以考虑将项目应用于虚拟现实（VR）和增强现实（AR）等领域，为用户提供更沉浸式的视觉体验。整合硬件加速技术，如 GPU 加速，进一步提高渲染性能，为元宇宙赋能。

References:

[1] Henrik, The ray-tracing algorithm builds an image by extending rays into a scene and bouncing them off surfaces and towards sources of light to approximate the color value of pixels

[2] Patric Ljung, Efficient Methods for Direct Volume Rendering of Large Data Sets

[3] DanielDFY, GAMES101, https://github.com/DanielDFY/GAMES101, 2020

[4] Aras Pranckevicius, Microfacet Models, pbr-book, 2018

[5] Feng Gu, Johannes Jendersie, Thorsten Grosch, Fast and Dynamic Construction of Bounding Volume Hierarchies Based on Loose Octrees, 2018

[6] Peter Shirley, Trevor David Black, Steve Hollasch, Ray Tracing in One Weekend, 2023-08-06