光线追踪算法实现

1 程序设计描述

1.1 光线生成算法

基于相机参数生成从视点出发穿过每个像素的光线，基于相机参数和像素坐标生成透视投影光线，模拟人眼成像原理，通过视场角(FOV)控制视野范围，光线方向通过像素坐标归一化计算得到。

通过 Camera.get_ray(x, y, width, height)方法实现：视口坐标系转换，将屏幕坐标(x,y)映射到 NDC 空间，并计算方向，参数化射线方程：

$$ray(t) = eye_{pos} + t \cdot dir, t \in [0, \infty)$$

其中，eye_{pos}为相机所在的位置，t 控制射线上的点距离相机的远近。

基于向量反射公式生成反射光线，使用法线向量归一化保证计算稳定性，添加 $1e-4$ 偏移量以防止自相交。

1.2 求交算法（Intersection Detection）

采用分层检测机制，通过空间划分优化相交测试效率。为每种几何体(球体、立方体、圆锥等)实现独立的光线相交检测，使用解析几何方法计算光线与几何体的交点，返回最近的交点(如果存在)。

逐个物体进行相交测试，记录最小 t 值的交点。返回结构包含交点距离 t、三维坐标 point、法线 normal。以球体为例，其隐式方程为：

$$||P - C||^2 = r^2$$

其中，P 表示三维空间中任意一点的坐标，C 为球体的中心点坐标，r 为球体的半径。将$P(t) = O + tD$代入隐式方程，得：

$$t^2 (D \cdot D) + 2t\big(D \cdot (O - C)\big) + (O - C) \cdot (O - C) - r^2 = 0$$

其中，O为光线起点，D为光线方向。解二次方程得 t 值，筛选正根，然后计算法线 N，公式如下：

$$N = \frac{P - C}{||P - C||}$$

通过调整 r, C, D 可以精确控制光线与球体的相交行为，进而影响渲染结果。

1.3 递归光线追踪策略

通过光线反弹模拟全局光照，采用深度限制控制计算复杂度。递归追踪反射和折射光线，设置最大递归深度防止无限循环，根据材质属性决定是否继续追踪。

1.4 光照模型计算（Blinn-Phong Model）

结合经验模型与物理特性，实现实时可用的局部光照模拟。实现 Phong 光照模型，包含环境光、漫反射和高光反射，支持多光源计算，考虑光线衰减。

1、环境光：场景级环境光与材质环境系数的乘积，模拟间接光照的近似解。

$$C_{ambient} = k_a \cdot I_a$$

其中，k_a 为材质环境光系数，表示材质对环境光的反射能力，为 RGB 向量。I_a 为场景环境光强度，表示场景中所有间接光照的总体强度。

2、漫反射项：采用 Lambert 余弦定律。

$$C_{diffuse} = k_d \cdot I_l \cdot \max(0, N \cdot L)$$

其中，k_d 为材质漫反射系数，表示材质对直接漫反射光的反射能力。I_l 为光源强度。N 为表面法线向量，L 为光源方向向量，$\max(0, N \cdot L)$是为了确保仅当光源在表面上方时计算漫反射。

3、阴影检测：发射阴影射线验证可见性。

4、镜面反射：使用 Blinn 半角向量优化，实现采用传统 Phong，Shininess 指数控制高光范围。

$$C_{specular} = k_s \cdot I_l \cdot \max(0, R \cdot V)^n$$

其中，k_s 为材质镜面反射系数，表示材质的高光反射能力。R 为反射向量，计算公式为$R = I - 2(N \cdot I)N$，V 为视线方向向量，n 为 Shininess 指数，控制高光范围和锐利度，$\max(0, R \cdot V)^n$ 为了确保仅当反射光朝向相机时计算高光。

1.5 主要功能函数说明

1、vector.py 文件：Vector3 函数实现三维向量的加减、点乘、叉乘、归一化等基础操作，支持所有几何体、光照、相机等模块的空间运算，为整个渲染系统提供

底层的空间代数支持。

2、material.py 文件：定义物体表面的光照属性（环境光、漫反射、镜面反射系数和高光指数），决定物体在不同光照下的颜色和高光表现。

3、geometry.py 文件：定义各种几何体（球体、立方体、平面、四面体等）及其与射线的求交算法，每个几何体通过 intersect 和 intersect_batch 函数来判断光线与物体的交点和法线。

4、scene.py 文件：管理所有物体和光源，负责递归追踪光线、计算像素颜色。add_object/add_light 函数负责添加物体和光源，trace_ray/trace_rays_batch 函数负责递归追踪主光线和反射光线，计算交点处的环境光、漫反射、镜面反射和阴影，实现核心的光线追踪算法和物理光照模型。

5、camera.py 文件：定义相机参数（位置、朝向、视场角等），并根据像素坐标生成主光线。计算每个像素在成像平面上的位置，通过 get_ray 方法，生成从相机出发、指向场景的光线。

6、ray_tracer.py 文件：主程序，负责场景搭建、渲染循环、进度显示和图片保存。

1.6 核心功能完成情况

1、三维场景构建（几何体放置）

完成情况：场景中已成功放置多个基本几何体，包括球体、立方体、四面体和平面（地板）。每个几何体的位置、大小、材质参数均可灵活设置。

效果：渲染结果中可见多个不同形状和材质的物体，空间关系正确。

2、光源布置

完成情况：场景中设置了四个点光源，每个光源有独立的位置。光源参数可调，支持多光源叠加。

效果：物体表面有明显的高光、阴影和明暗变化，光照自然。

3、光线与几何体求交

完成情况：所有几何体（球体、立方体、平面、四面体）都实现了射线求交算法。支持批量射线与物体的高效求交，返回交点、法线、距离等信息。

效果：每个像素的主光线都能正确判断与场景物体的最近交点，保证了渲染的准确性。

4、光照模型（环境光、漫反射、镜面高光）

完成情况：在交点处，综合计算了环境光、漫反射和镜面高光分量（Phong 模型），支持多光源叠加，材质参数可调。

效果：物体表面有真实的明暗、反射和高光效果，地板和物体均有自然的光照表现。

5、递归光线追踪

完成情况：实现了递归光线追踪，支持多次反射（可设最大递归深度），反射光线会继续与场景物体求交，叠加反射色彩。

效果：球体、地板等表面可见反射效果，增强了真实感。

6、整体渲染流程

完成情况：从相机发射主光线，批量追踪，递归处理反射，最终合成每个像素的颜色。渲染结果完整、无条带、无错位，图片输出正常。

2 运行情况

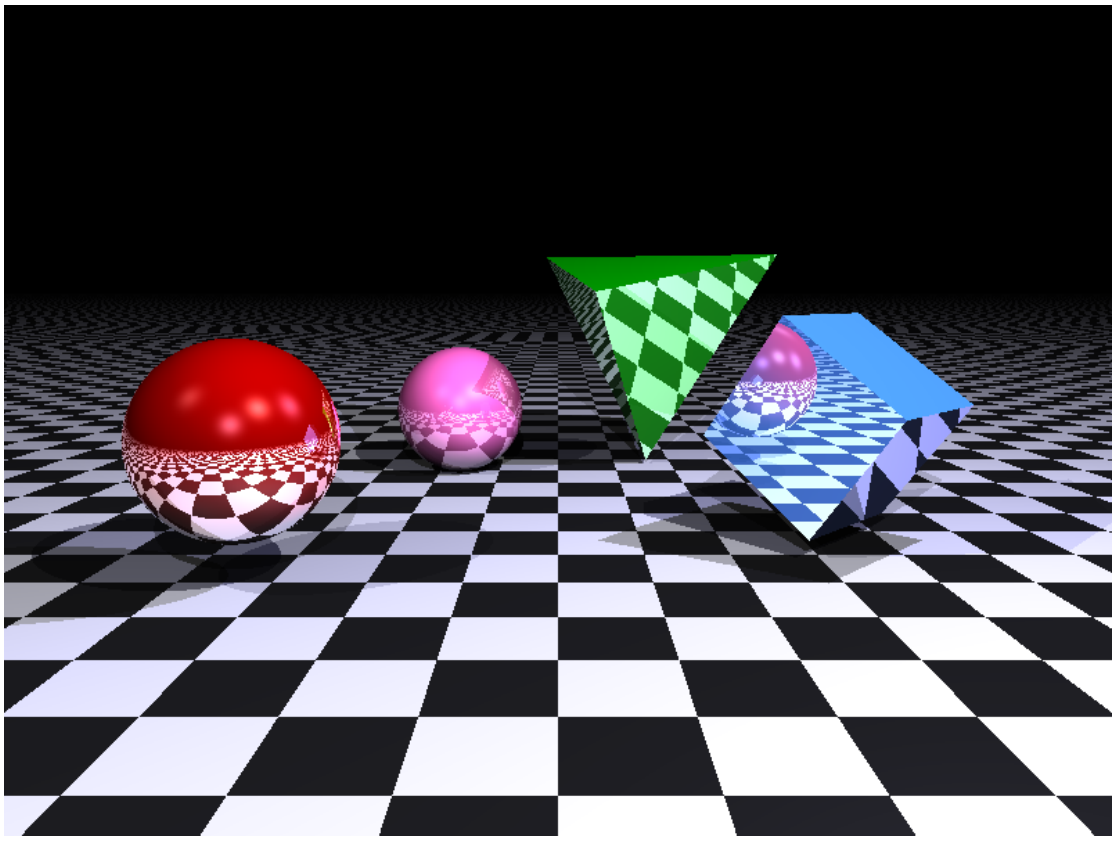

图2-1 运行结果图

成功加载 5 类几何体：红色/粉色球体、绿色棋盘格四面体、蓝色旋转立方体及黑白格子地板。材质系统正确解析 4 种材质属性（环境光、漫反射、镜面反射、光泽度）。4 组点光源完成空间分布，实现全局光照效果。

所有物体保持预设空间关系（图 2-1 运行结果图），红色球体（坐标-2.2,0,-4）与粉色球体（-1.1,-0.3,-8）呈现正确景深。旋转立方体（2.2,0,-5）的 45° Y 轴/35.26° X 轴旋转矩阵生效。四面体顶点坐标经旋转矩阵变换后，最低点保持 y=0.01 的悬空状态

球体表面呈现高光反射（shininess=50/30），立方体棱角处镜面反射强度符合预期（specular=[0.8,0.8,1.0]）。棋盘格材质 UV 映射准确，绿色四面体与蓝色立方体纹理无错位

3 总结及提升

3.1 程序优点分析

1、模块化架构设计：材质系统、几何体类（Sphere/Cube/Tetrahedron 等）、场景管理、相机模块解耦清晰，便于扩展新几何体或材质类型。使用面向对象编程（OOP）封装光线追踪核心逻辑，符合图形渲染管线设计范式

2、GPU 加速优化：全程使用 PyTorch 张量运算，充分发挥 GPU 并行计算能力（如旋转矩阵计算、颜色混合）。批处理渲染机制（batch_size=1024）有效减少 CPU-GPU 数据传输开销

3、物理真实感实现：实现环境光(ambient)、漫反射(diffuse)、镜面反射(specular)的 Blinn-Phong 光照模型。多光源系统（4 个点光源）增强场景层次感

4、工程化实践：进度条可视化(tqdm)提升渲染过程可观测性。自动保存输出路径（基于脚本目录）增强部署便利性。异常捕获机制保障渲染中断时的错误定位。

3.2 现存不足与挑战

1、性能瓶颈：逐像素循环渲染未完全发挥 GPU 并行性。

2、缺乏空间加速结构（如 BVH/KD-Tree），复杂场景渲染效率指数级下降。

3、功能局限性：仅支持基础几何体，缺乏三角形网格加载能力。材质系统未实现纹理映射、法线贴图等高级特性。缺少阴影生成、环境光遮蔽(AO)等全局光照

效果。

4、内存管理：全尺寸图像直接存储（800x600x3=1.44MB）虽小，但高分辨率场景可能触发显存限制。

3.3 开发难点与解决思路

1、几何体旋转处理

难点：几何体的旋转需保持局部坐标系一致性。

方案：通过矩阵乘法实现局部空间到世界空间的变换，采用局部坐标偏移确保旋转中心正确性。

2、渲染速度优化

难点：大尺寸图像渲染时 CPU 计算速度慢，需将计算转移至 CUDA。

方案：运用 Torch 将计算转为张量的性质存入 GPU 进行计算，但出现计算效率低的问题，逐步优化至高效率。

3、物理正确性验证

难点：确保光线-几何体相交计算的数学正确性。

方案：通过单元测试对比理论值（相交公式与解析解对比）。

3.4 优化提升方向

1、引入 BVH 加速结构：通过空间划分减少无效相交测试，预期提升复杂场景渲染速度 5-10 倍

2、实现延迟渲染，将着色计算推迟到可见性确定后，减少冗余计算。

3、添加 OBJ 格式加载器：支持三角形网格渲染，扩展场景复杂度

4、实现 PBR 材质系统：集成金属度(Metallic)、粗糙度(Roughness)参数，支持 IBL 环境光照。

5、工程优化：全流程 CUDA 化，将 Python 层循环移植到 CUDA Kernel，消除 CPU-GPU 同步开销。

6、动态分辨率渲染：根据负载自动调整渲染区域，平衡画质与帧率。

7、可视化增强：添加抗锯齿(TAA/MSAA)：通过时间/空间采样减少锯齿现象。

8、实现色调映射(Tone Mapping)：将 HDR 颜色映射到 LDR 显示空间。