
Final Project Report – Path Tracing

王想 2100013146

1 Compile Environment

本实验选题为 PathTracing，参考闫令琪老师开设的《GAMES101 现代计算机图形学入门》课程实现算法。编译命令与可视计算与交互概论课程 lab 所提供代码框架相同。编译时在根目录依次执行命令 `xmake` 和 `xmake project -a x64 -k vsxmake ./build`, 若编译器不是 Visual Studio 则后一个命令需参考课程 lab 的 tutorial。编译器我使用 Visual Studio, OpenMP 库是 Visual Studio 自带的, 故无需另外安装其他第三方库, 若选用其他编译器, 需确保支持 OpenMP 库, 以支持多线程并行从而加速渲染。¹

2 Theory

PathTracing 的理论基础是渲染方程

$$L_0(p, \omega_0) = L_e(p, \omega_0) + \int_{\Omega} f_r(p, \omega_i, \omega_0) L_i(p, \omega_i) \cos \theta_i d\omega_i$$

上式中 $L_e(p, \omega_0)$ 是位置 p 的方向 ω_0 的自身发光的辐射率 (Radiance); $L_i(p, \omega_i)$ 是位置 p 的方向 ω_i 的入射光的辐射率, 包括光源和其他物体的反射光; $\cos \theta_i$ 是物体表面在位置 p 的法向与方向 ω_i 的夹角余弦值, $f_r(p, \omega_i, \omega_0)$ 是物体在位置 p 的双向反射分布函数 (BRDF)。BRDF 项由两部分组成, 一部分是漫反射项, 另一部分是镜面反射项。

$$f_r = k_d * f_d + k_s * f_s$$

其中, 漫反射项计算较为简单, 用 c 表示物体颜色, 则漫反射项 $f_d = \frac{c}{\pi}$; 镜面反射项则比较复杂, 现实中一般的镜面都并非完美的理想镜面, 需要用到微表面模型理论 (Microfacet Model), 根据 Cook-Torrance 模型, 镜面反射项

$$f_s = \frac{F(\omega_i, N)G(\omega_i, \omega_0, h)D(h, N)}{4(n \cdot \omega_i)(n \cdot \omega_0)}$$

上式中 ω_i 是入射方向, ω_0 是反射方向, h 是微平面法向, n 是宏观表面法向; FGD 这三项分别描述了物体镜面反射的相关属性: 函数 F 是菲涅尔项, 描述了物体表面在不同入射光角度下反射光线所占的比率; 函数 G 是几何函数, 描述了微平面自遮挡的属性; 函数 D 是法线分布函数, 代表了所有微观角度下微小镜面法线的分布情况。

求解上述渲染方程中的定积分采用蒙特卡洛积分

$$\int_a^b f(x) dx = \frac{1}{N} \sum_{i=1}^N \frac{f(X_i)}{p(X_i)}$$

¹对于部分编译环境, 若渲染时出现全黑现象请 Reset Scene, 之后便会正常运行。

上式中 X_i 为采样点, 其分布服从概率密度函数 $p(x)$, N 是采样次数, 将采样得到的值近似当成定积分的值。采样最简单的方法就是均匀采样, 但均匀采样收敛速度太慢, 需要较高的采样率才能得到较好的渲染效果, 所以通常会进行重要性采样。

利用蒙特卡洛积分将上述渲染方程转化为伪代码的形式就是

$$shade(p, \omega_0) = L_e(p, \omega_0) + \frac{1}{N} \sum_{i=1}^N \frac{f_r(p, \omega_i, \omega_0) shade(p_i, -\omega_i) \cos \theta_i}{p(X_i)}$$

这是一个递归程序, 随着递归深度的增加, 容易出现指数爆炸的问题, N^{depth} 只有在 $N = 1$ 时才不会产生这个问题, 所以这里采用 $N=1$, 但此时得到的近似定积分值显然不够准确, 所以需要在每一个像素上进行超采样从而获得足够准确的效果。递归的另一个问题是程序如何终止, PathTracing 算法在此引入俄罗斯轮盘赌, 设置一个固定概率 P_{RR} , 每次递归就产生一个位于区间 $[0,1]$ 的随机浮点数 f , 当 $f > P$ 时进行截断, 返回 0 向量。

由于 $shade(p, \omega)$ 方法中只有打到光源的光线才能算出对应的 L_e , 而采样的光线打到光源的概率通常不大, 很多采样的光线是无用的就浪费掉了, 所以采样时需要一部分直接对光源采样, 另一部分再对间接光采样, 从而加快收敛速度, 在更小的采样率 (ssp) 下获得更好的渲染效果。

3 Implement

依据上述理论, 计算 $shade(p, \omega_0)$ 算法步骤如下:

1. 在点 p 对光源进行均匀采样 ($pdf_light = 1/LightArea$), 如果中间没有物体阻挡, 计算直接光照 $L_dir = LightIntensity * f_r * \cos \theta * \cos \theta' / distance^2 / pdf_light$, 否则 L_dir 为 0 向量
2. 进行俄罗斯轮盘赌, 如果 $random_float > P_{RR}$ 则直接返回向量 L_dir
3. 在点 p 物体表面对反射光的方向进行重要性采样得到方向 ω_i
4. 向方向 ω_i 产生一条光线, 如果光线打到一个不发光的物体 q , 计算间接光照 $L_indir = shade(q, -\omega_i) * f_r * \cos \theta / pdf / P_{RR}$, 否则 L_indir 为 0 向量
5. 返回 $L_dir + L_indir$

cornell_box 中的光源是点光源, 为了适应对光源采样的需要, 我对其中的光源的处理进行了修改, 将其处理成面积光源进行采样。鉴于这一算法渲染时间过长, 我采取 BVH 数据结构进行光线求交加速, 并且在构建时采用 SAH 进一步提升性能; 并且调用 OpenMP 库实现多线程并行计算², 进一步加快渲染。

计算 BRDF 中的菲涅尔项时, 我同时实现了精确求解和 Schlick 近似求解菲涅尔项, 实验结果表明二者结果差异不大, 但 Schlick 近似计算量大幅减小, 故最终我采用 schlick 近似计算菲涅尔项, 公式如下:

$$F(\omega_i, N) = R_0 + (1 - R_0)(1 - \omega_i \cdot N)^5$$

$$R_0 = \left(\frac{\eta_1 - \eta_2}{\eta_1 + \eta_2} \right)^2$$

BRDF 中的几何函数我采用 Schlick-GGX, 记 h 为微观法向, α 为物体粗糙度, 则计算公式如下:

$$G(\omega_i, \omega_0, h) = G1(\omega_i, h) G1(\omega_0, h)$$

²当前采用 4 线程并行, 如需调整请修改 CasePathTracing.cpp 中第 108 行的 *num_threads* 参数

$$G1(\omega, h) = \frac{\omega \cdot h}{(\omega \cdot h)(1 - k) + k}$$

$$k = \frac{(\alpha + 1)^2}{8}$$

法线分布函数采用 GGX, 记 h 为微观法向, N 为宏观表面法向, α 为物体粗糙度, 则计算公式如下:

$$D(h, N) = \frac{\alpha^2}{\pi((N \cdot h)^2(\alpha^2 - 1) + 1)^2}$$

在采样时我采用多重重要性采样, 对于光源使用均匀采样, 对于漫反射材质使用 cosine-weighted 采样, 对于存在镜面反射的材质采用 BRDF 采样。记 ξ_1, ξ_2 分别为两个服从 $[0,1]$ 均匀分布的随机变量, 那么, cosine-weighted 采样的极坐标和对应的概率密度计算公式如下:

$$\theta = \arccos(1 - \xi_1), \quad \phi = 2\pi\xi_2$$

$$pdf = \frac{\cos \theta \sin \theta}{\pi}$$

BRDF 采样的极坐标和对应的概率密度计算公式如下:

$$\theta = \arccos \sqrt{\frac{1 - \xi_1}{\xi_1(\alpha^2 - 1) + 1}}, \quad \phi = 2\pi\xi_2$$

$$p_h = \frac{\alpha^2 \cos \theta \sin \theta}{\pi((\alpha^2 - 1) \cos^2 \theta + 1)^2}$$

$$pdf = \frac{p_h}{2(\omega_0 \cdot h)}$$

俄罗斯轮盘赌的概率 P_{RR} 会影响光线的反射次数, 光线停止反射时光线已经反射的次数 X 是服从 $p = 1 - P_{RR}$ 几何分布的随机变量, 其数学期望 $E(X) = \frac{1}{p}$, 本次实验中选用 $P_{RR} = 0.8$, 即光线反射次数的数学期望为 $\frac{1}{1-0.8} = 5$ 。

目前 lab3 的 RayTracing 框架中提供的场景除了 cornell_box 与 floor 外的几个场景所需的渲染时间极长, 即使使用 BVH 数据结构与重要性采样进行加速仍然无法在合理的时间内得出渲染结果, 而 floor 场景并不能体现 PathTracing 的优点, 故最终 Final Project 中提供的场景只有 cornell_box 这一个。并且, 此场景的光源强度、光源位置与物体材质相关数据我都进行了相应的调整, 与 lab3 中 RayTracing 的 cornell_box 场景不同。

算法实现效果如 Fig. 1 所示。在较低采样率时噪点较多, 随着采样率的提高, 噪点逐渐减少, 当 $ssp=256$ 时已经几乎没有噪点了, 并且相比光栅化和光线追踪而言, 渲染结果更加真实, 更贴合物理实际, 体现出来了磨砂金属材质的质感, 但是相应的缺点是渲染时间大幅度增加。

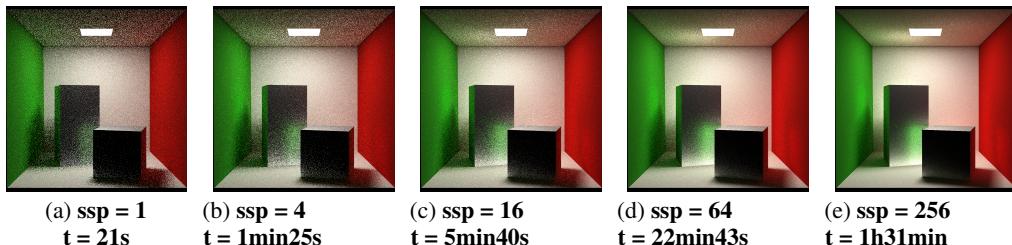


图 1: result

4 Analyse

4.1 Compare to RayTracing

为了更好地体现 color bleeding 的效果, 我将场景中的高矮盒子全部改成全漫反射材质, 以免反光的颜色干扰呈现 color bleeding 的效果。由于实现 PathTracing 时对场景进行了相应的调整, 基于控制变量原则, Fig. 2 中 RayTracing 的渲染结果是对场景做出相同调整后得出的。相比 RayTracing, PathTracing 最显著的提升是渲染效果更贴近现实, 然而, 更贴近物理真实的代价是计算量的增加, 也就是渲染时间的增加。Fig. 2 中 RayTracing 的渲染只需要不到 1 分钟, 但 PathTracing(ssp=64) 的渲染需要的时间却长达 23 分钟。

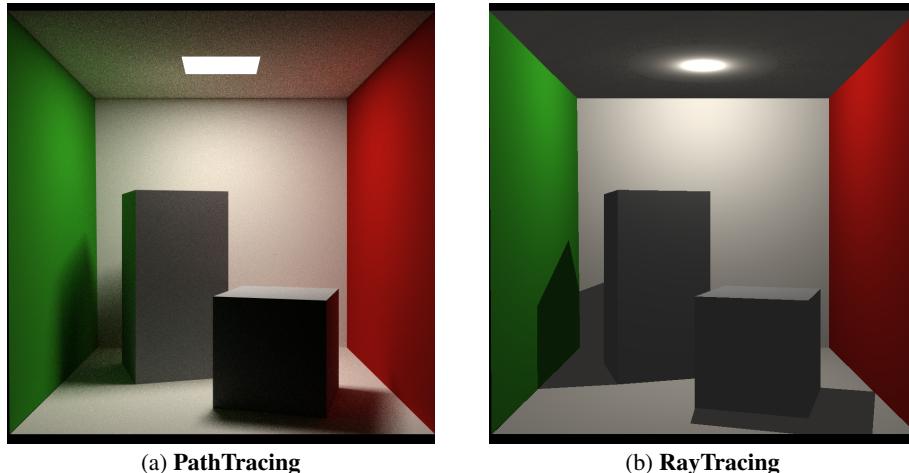


图 2: Overview

PathTracing 更贴近现实主要体现在两个方面, 一个是颜色渗透 (color bleeding) 现象, 一个是能够渲染出柔和平滑的影子。Fig. 3 展现了两种算法渲染效果的部分放大图, 可以明显看到 PathTracing 渲染结果中两个盒子的侧面分别呈现出对应的墙的颜色, 现实生活中我们也经常看到这种颜色渗透的现象, 但 RayTracing 渲染结果中看不到这种现象。

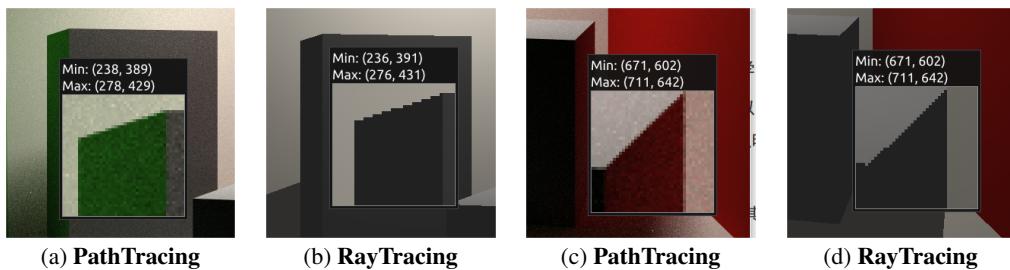


图 3: Color Bleeding

Fig. 4 展现了两种算法渲染效果的影子边缘的放大图, 可以明显看到 PathTracing 渲染结果的影子边缘是平滑的过渡, 影子更加柔和, 现实生活中我们看到的影子也都是这种柔和的影子, 但 RayTracing 渲染结果的影子边缘有明显的分界线, 影子十分尖锐。

4.2 Compare to UniformSampling

Fig. 5 展示了使用均匀采样和使用多重重要性采样的渲染结果 (ssp=16)。当场景中存在具有镜面反射的材质的物体时, 可以明显看到均匀采样的渲染效果难以体现镜面反射的现象。这是因为均匀采样在半球上均匀采样反射光线方向, 这个实际上只适合理想的全漫反射材质, 对于镜面反射材质, 光线不会在其表面均匀反射, 导致均匀采样的效果难以体现镜面反射的

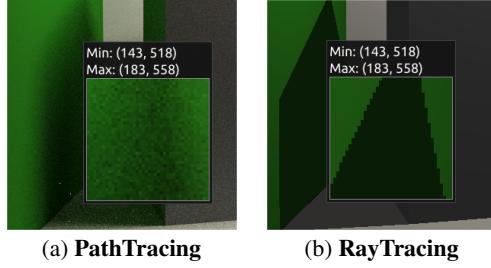


图 4: Shade

效果,看上去高矮盒子就像全部是漫反射材质。而 BRDF 采样则是利用 BRDF 和微表面模型进行的光线反射方向采样,更符合物理实际,所以渲染出来的效果也更好。

当场景中是全漫反射材质,均匀采样也不如 cosine-weighted 采样。因为当反射光线方向与物体表面法向夹角越靠近 90 度,反射出来的光线越可能难以击中光源而成为无效光线。均匀采样时会有很多反射光线浪费掉,导致影子与天花板比较黑。而 cosine-weighted 采样在这种情况下的概率密度较低,采样得到的反射光线有更高概率是有效光线,从而在全漫反射材质场景、相同 ssp 时,cosine-weighted 采样的渲染效果更好,更符合物理实际。

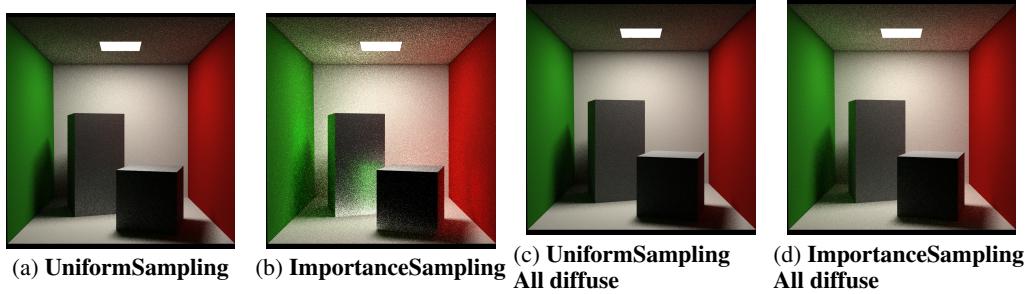


图 5: ssp=16

4.3 About result

就渲染结果而言,物理真实感相较于 RayTracing 已经有了大幅度的提升,但在渲染效率方面则仍有待提升。尽管已经使用了 BVH 和 SAH 建树来加速光线求交,实际上在这个场景中的速度提升并不明显,这是因为这个场景中的三角形本来也不多,在 lab3 的 RayTracing 的后面几个场景中效率提升才显著。但由于 PathTracing 算法计算量太大,渲染时间太长,即使我使用了多线程并行计算,并且在计算 BRDF 项时使用了一些近似计算以减小计算量,那些场景渲染时间仍难以接受,后续可以考虑使用 GPU 进行加速处理。