光线追踪算法的实现和思考

丁成 (学号 13307130299)

2013 年 12 月 3 日

摘要 本文介绍了光线追踪渲染算法原理、实现方法、真实模拟算法和实时渲染算法, 并提出了一些优化方法

关键词 光线追踪, 计算几何, 计算机图形学, C++, OpenGL, 实时渲染

1 引言

光线追踪 (Ray Tracing) 算法是一种基于真实光路模拟的计算机三维图形渲染算法. 相比其它大部分渲染算法, 光线追踪算法可以提供更为真实的光影效果.

此算法由 Appel 在 1968 年初步提出,1980 年由 Whitted 改良为递归算法并提出全局光照模型. 直到今天, 光线追踪算法仍是图形学的热点, 大量的改进在不断涌现.

基于对自然界光路的研究, 光线追踪采取逆向计算光路来还原真实颜色. 模拟的过程中涵盖了光的反射、折射、吸收等特性 (精确计算), 并辅以其它重要渲染思想 (进一步模拟). 其中包含了重要方法, 诸如冯氏光照模型 (Phong Shading)、辐射度 (Radiosity)、光子映射 (Photon Mapping)、蒙特卡罗方法 (Monte Carlo) 等等.

鉴于光线追踪算法对场景仿真程度之高, 其被普遍认为是计算机图形学的核心内容, 以及游戏设计、电影特效等相关领域的未来方向. 近年来由于硬件系统的迅速改良, 基于分布式、GPU, 甚至实时渲染的光线追踪算法也纷纷出现.

本文先给出光线追踪算法的基本框架结构和数学理论基础, 实现算法框架. 后介绍更多优化模型以及实现方法, 最后提出一些自己的改良算法, 并将其移植于 OpenGL 环境并实现简易实时移动渲染.

2 纵览

2.1 算法原理

从视点向屏幕上每一个像素发出一条光线, 追踪此光路并计算其其逆向光线的颜色, 映射到对应的像素上.

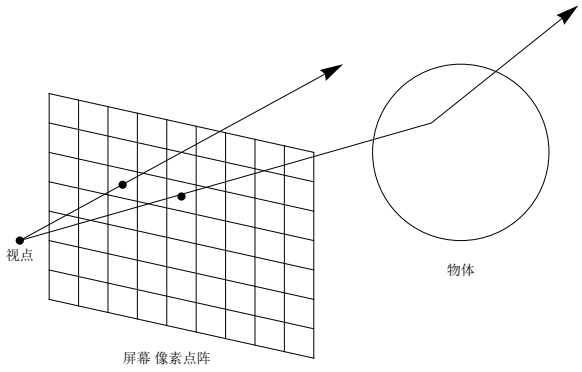

图 1: 光线追踪基本示意图

如图 1, 通过计算光路上颜色衰减和叠加, 即可基本确定每一像素的颜色.

2.2 框架

使用 C++ 实现完整的空间向量类, 包括相交检测、计算光线反射与折射. 对反射和折射后的光线递归计算颜色再叠加起来即可算得原光线颜色. (若光线未遇到物体则可返回背景色)

在递归的过程中, 光亮会不断衰减. 因此可以限制迭代的深度或者光亮小于特定值时停止迭代.

将对应于每一像素的光线颜色绘制到图片文件的相应坐标, 最后输出绘制所得图像.

基本 rayColor() 递归算法伪代码如下：

算法 1: 光线追踪递归框架

输入: 光线发出位置向量和方向向量
输出: 反向光颜色
function rayColor()

if *no intersection with any object* **then**
    return *background color*;
else
    obj ← find nearest object from the ray;
    reflect ray ← calculate reflect ray with *obj*;
    refract ray ← calculate refract ray with *obj*;
    main color ← the radiance of *obj*;
    reflect color ← rayColor(*reflect ray*);
    refract color ← rayColor(*refract ray*);
    return mix(*main color, reflect color, refract color*);

3 初步实现

对于任意几何体, 都需要建立特定函数 $f(\mathbf{x})$. 自变量 $\mathbf{x}$ 表示空间点的坐标, 函数

$$f(x)=\begin{cases}-1 & \text{点}\mathbf{x}\text{在几何体外部}\\ 0 & \text{点}\mathbf{x}\text{在几何体表面上}\\ 1 & \text{点}\mathbf{x}\text{在几何体内部}\end{cases} \tag{1}$$

3.1 计算交点

复杂几何体都可以近似划分为基本几何体, 一般只需要实现光线与球体或平面求交.

光线上的点坐标（以向量表示）可以写为

$$\mathbf{p}=\mathbf{s}+t\mathbf{d} \tag{2}$$

其中 $\mathbf{s},\mathbf{d}$ 分别是起始点坐标和方向向量.

因此交点可以表示为方程

$$f(\mathbf{p}=\mathbf{s}+t\mathbf{d})=0 \tag{3}$$

解出 t 值即可.

3.1.1 球体

球面方程为

$$|\mathbf{p}-\mathbf{c}|=r \tag{4}$$

$\mathbf{p}$ 为 (2) 式中坐标向量. 此时

$$f(\mathbf{p})=\mathrm{sgn}(r-|\mathbf{p}-\mathbf{c}|)=0$$

解出 t 值即可.

3.1.2 平面

设平面法向量的坐标为 $\mathbf{c}$, 法向量方向为 $\mathbf{N}$, 则有以下方程

$$(\mathbf{p}-\mathbf{c})\mathbf{N}=\mathbf{0} \tag{5}$$

同理也可解出交点位置.

3.1.3 其他几何体

可以将复杂曲面划分为许多个小平面（或三角形）的近似. 因此将复杂问题化归为光线与空间内多边形求交点.

首先求出光线与多边形所在平面的交点, 将此交点的空间坐标向量与该平面的两个基向量相乘即可得到其对应于此平面的二维坐标.

判断平面上点与多边形的位置关系可以使用经典的射线法.

3.2 反射光线

由反射定律, 入射光线与反射光线、法线共面, 且两条光线关于法线对称. 以上讨论的几种基本情况, 法线向量 $\mathbf{N}$ 是易求得的. 由几何关系可知

$$\mathbf{d}'-\mathbf{d}=2\mathbf{N} \tag{6}$$

其中 $\mathbf{d}'$ 是反射光线, 故可直接求得.

3.3 折射光线

首先通过折射定律判断是否存在折射, 如有则求出折射角. 将入射光线、折射光线、法线向量平移拼为三角形, 由正余弦定理即可求出折射光线向量.

3.4 Lambert 明暗模型

假设入射光线为 $\mathbf{d}$, 反射光线为 $\mathbf{d}'$, 入射点和光源的连线向量为 $\mathbf{l}$, 那么此处光亮可以定义为[1]

$$k=\cos<\mathbf{d}',\mathbf{l}>=\frac{\mathbf{d}'\mathbf{l}}{\|\mathbf{d}'\|\|\mathbf{l}\|} \tag{7}$$

此处反射颜色是原反射颜色乘以 k 之后的值.

[1] 根据亮度公式可近似推导

3.5 颜色

与传统 255 色不同, 这里颜色使用标准的全值域 RGB 模型, 会更便于计算. 将颜色表示为向量 (r, g, b), 其中 r, g, b 均为 0 到 1 之间的实数[2].

3.5.1 光照颜色

RGB 值为 $\mathbf{c_1}$ 的光线照射到 RGB 值为 $\mathbf{c_2}$ 的物体上, 反射或折射后的颜色为 $\mathbf{c_1} \bigotimes \mathbf{c_2}$,$\bigotimes$ 为笛卡尔积.

3.5.2 叠加颜色

RGB 值为 $\mathbf{c_1}$ 的光线和 RGB 值为 $\mathbf{c_2}$ 的光线叠加后颜色为 $\mathbf{c_1} \bigoplus \mathbf{c_2}$,$\bigoplus$ 为笛卡尔和. 叠加后大于 1 的 r, g, b 值直接赋为 1.

3.5.3 颜色衰减

RGB 值为 $\mathbf{c}$ 的光线照射在衰减度为 k 的材质表面后, 颜色表现为 $k\mathbf{c}$.

3.6 物体属性

3.6.1 反射比例（reflect）

表示与物体相交后, 有多少比例的光线产生了反射.

3.6.2 折射比例（refract）

表示与物体相交后, 有多少比例的光线产生了折射.

3.7 初步算法演示

应用了光源、球面、反射、漫反射、颜色叠加的基本光线追踪模型, 效果如右图所示：

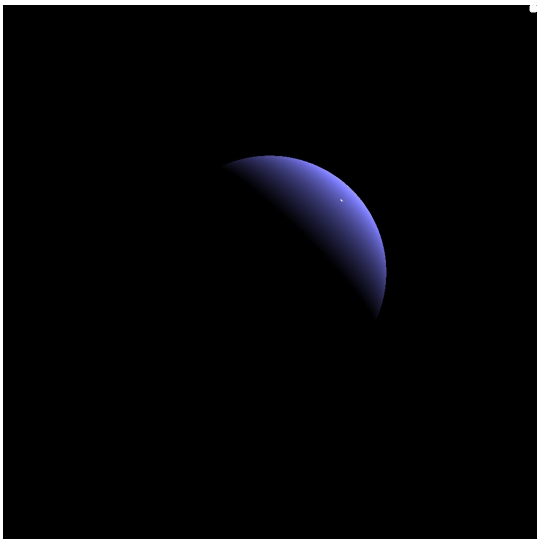

图 2: 初步算法效果

4 Phong 光照模型

4.1 原理

为了提供更为真实的渲染效果,Bùi Tuòng Phong 发明了 Phong 着色法. 这种方法将漫反射（diffuse reflection）和正反射（specular reflection）叠加在一起, 并给正反射加以高光, 使得“光晕”效果更明显.

使用 Phong 着色法需要给物体添加许多额外的物理参数：

4.2 Phong 模型中的物体属性

4.2.1 漫反射亮度（diffuse）

表示漫反射光的衰减比例.

4.2.2 反射亮度（specular）

表示正反射光的高光比例.

4.3 效果对比图

[2]一般的, 近似认为三原色光模式为线性且无偏移

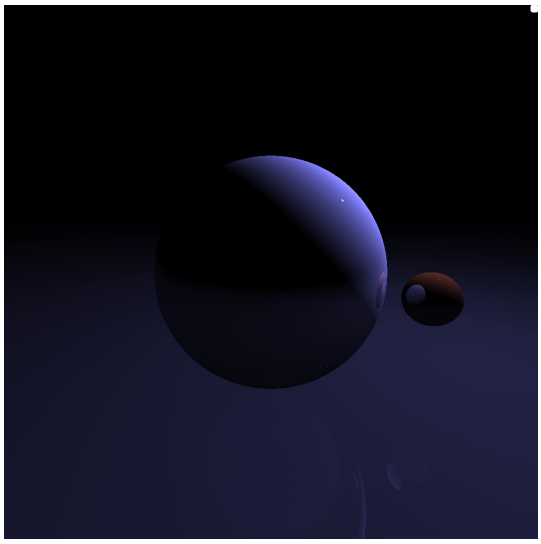

图 3: 球面及平面反射效果（普通）

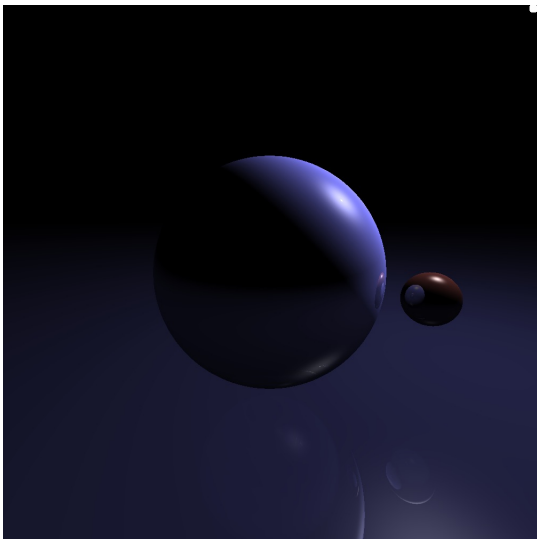

图 4: 球面及平面反射效果（Phong 着色）

4.4 参数

“光晕”在越接近正反射处表现得越明显, 因此高光函数应选取一个基于夹角余弦值的单调、且剧烈变化的函数. 有很多种模拟方法, 上图取了额外高光系数

$$s = \cos^{50}\theta \tag{8}$$

此时物体的表现光由以下组成

$$\begin{aligned} &obj.reflect * obj.specular * \cos^{50}\theta specularColor \\ &+obj.reflect * obj.diffuse * diffuseColor \\ &+obj.refract * refractColor \end{aligned}$$

5 阴影

阴影和高光一样, 也是让渲染增强真实感不可缺少的元素.

5.1 计算方法

将入射点和光源连线, 若与物体相交则可判断此处有阴影.

若判断有阴影, 则将 $obj.reflect, obj.refract$ 两个参数乘上一个小于 1 的倍数, 以产生变暗效果. 这里取值 $shadowRate = 0.4$.

5.2 阴影效果

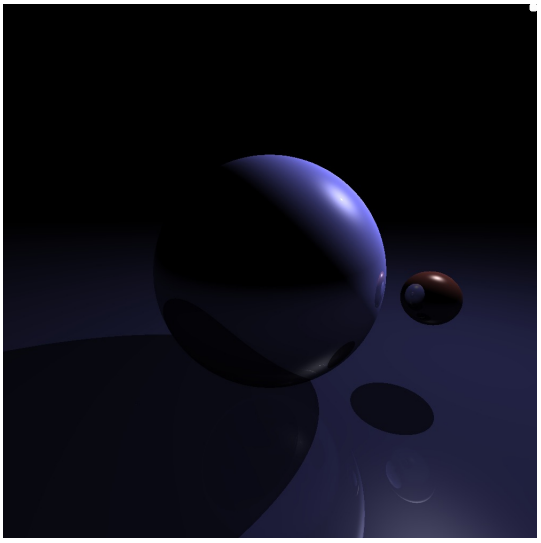

图 5: 普通阴影

5.3 软阴影、抗锯齿和漫反射

上面产生的阴影, 边缘太过尖锐, 没有现实中阴影的平滑. 软阴影有两种常见渲染方法.

5.3.1 采样

任何物体都不等价于一个质点, 光源亦如此. 光源放大来看就是一个球体, 因此阴影边缘的一个点可能被部分光源照到, 那么它就介于阴影和非阴影之间. 有一种思路是计算被光照的比例, 但难以实现. 目前被广泛应用的方法是“采样”.

我们随机在光源球体中取若干个点, 并每次将光源定于其中一点, 计算阴影. 这样计算若干次, 每一次由于光源位置稍有不同, 故阴影也不同. 将这些结果叠加起来, 根据概率知识, 很容易发现当取点数量达到一定时, 叠加起来的阴影越接近真实.

5.3.2 蒙特卡罗算法

蒙特卡罗算法是一大类随机算法的统称, 也是另一种接近真实的渲染方法. 在现实中不存在绝对平滑的物体, 所以不存在正反射. 利用这一点, 可以在每一次计算反射光的时候随机偏离一个角度. 如此随机很多次, 再将颜色平均起来就得到这一点的颜色. 蒙特卡罗算法既可以模拟计算漫反射, 也可以计算软阴影.

给物体添加附加属性“粗糙度”, 代表这个随机角的范围. 越粗糙的物体表面随机角越大.

5.3.3 软阴影和漫反射效果

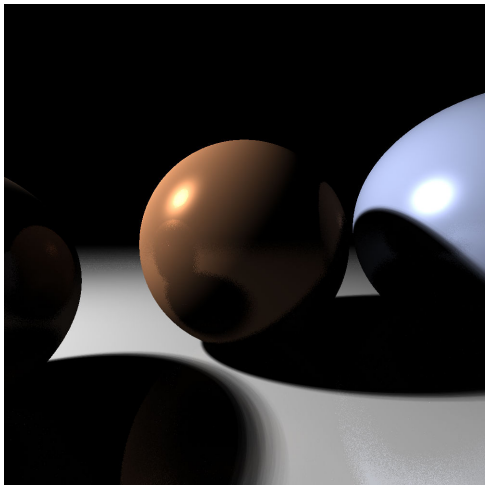

图 6: 采样与蒙特卡罗算法结合

6 镜头

观察发现, 靠近画面边缘的球体显示为椭球. 这是由于屏幕以斜面截取视线, 即所谓的广角镜头. 我们可以通过调整屏幕的形状（将平面映射到一个球面上）, 或者在镜头前加一个透镜.

同一场景, 调整镜头前后对比如下：

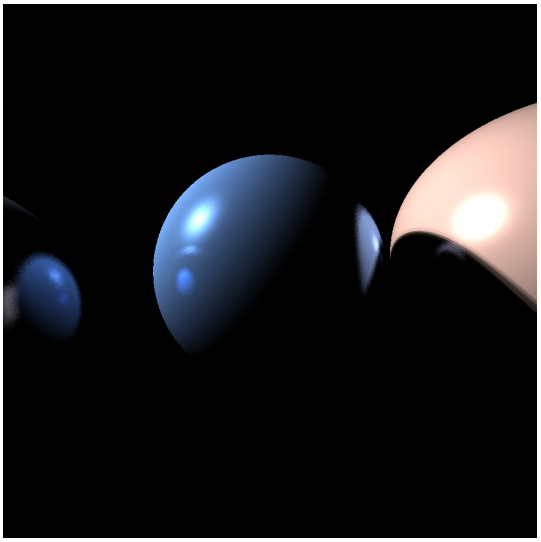

图 7: 广角镜头

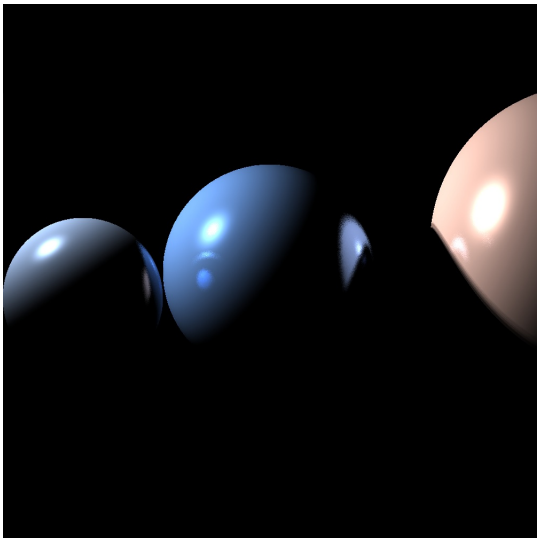

图 8: 鱼眼镜头

7 纹理

通过照射点的三维坐标, 可以还原出这个点对于此物体的坐标, 基于此就能给物体加上纹理. 例如对于平面有

$$\mathbf{x}' = \mathbf{n}\mathbf{x} \tag{9}$$

其中 $\mathbf{n}$ 是平面的基向量,$\mathbf{x}$ 是点的三维坐标,$\mathbf{x}'$ 是点相对于平面的坐标.

7.1 改进后的光线追踪效果图

再对物体的各种参数进行微调, 给平面添加纹理后得到效果图如下.

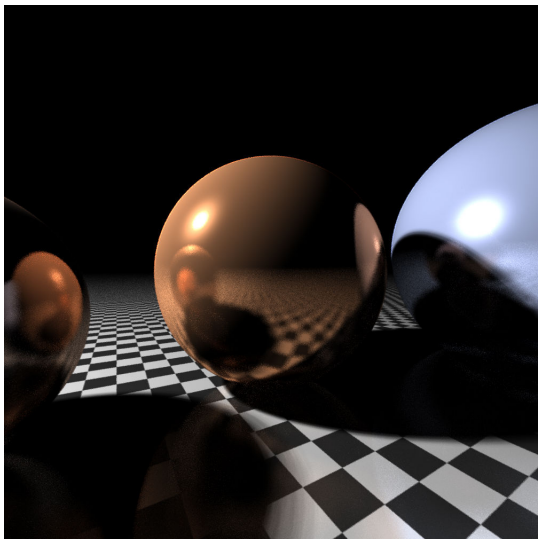

图 9: 添加纹理后的效果

类似的, 球面映射也可以通过极坐标（确定经纬度）来计算.

8 折射

8.1 折射定律

光线折射遵循斯涅耳定律

$$\frac{\sin\theta_1}{\sin\theta_2} = \frac{n_2}{n_1} \tag{10}$$

其中 θ_1, θ_2 分别为入射角、折射角,n_1, n_2 表示两种材质的折射率.

8.2 计算折射光线

通过常规解方程法计算折射光线太过繁琐. 因为向量之间便于计算夹角, 可以通过光学几何关系来确定折射光线.

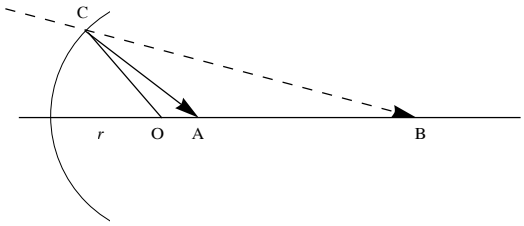

图 10: 计算折射光线

由正弦定理：

$$\frac{|OB|}{\sin\angle OCB} = \frac{r}{\sin\angle OBC} \tag{11}$$

$$\frac{|OA|}{\sin\angle OCA} = \frac{r}{\sin\angle OAC} \tag{12}$$

由折射定律：

$$\frac{\sin\angle OCB}{\sin\angle OCA} = \frac{n_2}{n_1} \tag{13}$$

又因为

$$\angle OCA + \angle OAC = \angle OCB + \angle OBC \tag{14}$$

结合上面四个方程就可解出折射光线.

8.3 一些问题

考虑折射（穿透）时, 往往会产生不止一个交点, 需要就发光点与物体位置关系进行讨论. 当光线存在于物体内部时, 折射率应该倒过来. 当折射角大于 90° 时应该不予考虑.

玻璃的折射率大致在 1.5, 也可以自己调整.

8.4 折射效果演示

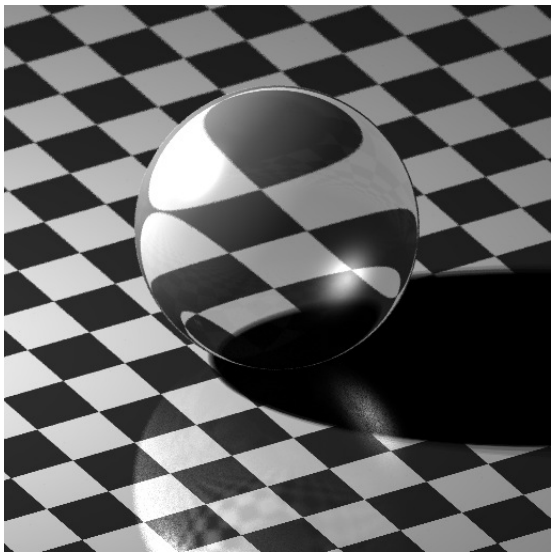

图 11: 折射效果图

9 景深

人眼和摄像机会因聚焦而产生景深的效果, 即在焦点外会产生模糊效果. 经过自己的思考, 下面通过两种方法实现模拟的景深效果.

9.1 叠加方法

将视点和视线方向随机扰动若干次, 但保证每一次均聚焦同一点, 再把绘制出来的图像叠加起来. 这样的优点是图像各处更为平滑, 缺点是需要大量“随机 -重绘”过程来增加真实感.

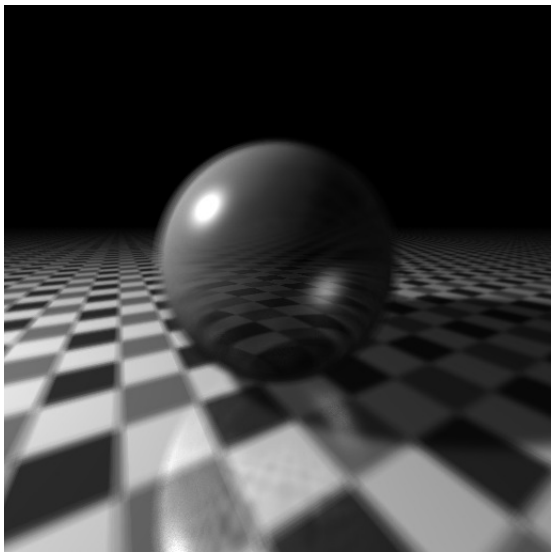

图 12: 叠加方法

9.2 蒙特卡罗方法

将每一条发出的视线方向上随机扰动一个角度. 关键点在于扰动角度的范围, 与视线和焦点的夹角正相关. 这样的概率分布明显是合乎逻辑的. 经过绘制实验, 发现经过一次绘制, 聚焦效果就已经十分显著. 缺陷是图像噪点多, 仍需进行稍许叠加方法计算.

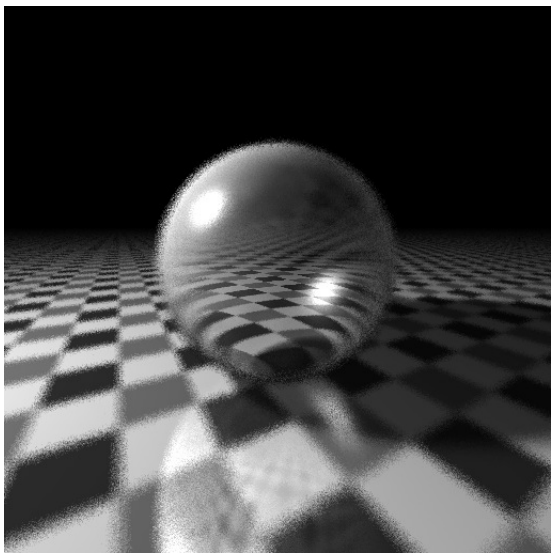

图 13: 蒙特卡罗方法

10 实时计算

在游戏等实际应用里, 对光线追踪算法的要求是能进行实时计算.

本文选取了移植于 OpenGL 库的实时光线追踪算法程序. 此程序可以随意移动视点和改变视线方向, 并实时输出绘制画面.

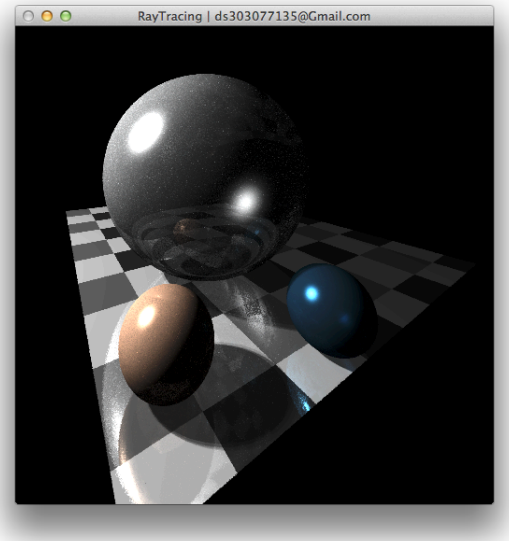

图 14: 基于 OpenGL 的实时光线追踪程序

算法大体部分没有改变, 主要不同的细节在于:

根据视点、视线的移动来调整重绘帧数;

直接利用基于 OpenGL 的光照系统、颜色系统、三维矩阵模型;

重绘的时候不必填白上一帧, 重绘像素点的顺序可以随机, 这样视觉上更有连续感.

11 效率

光线追踪因其计算量大, 算法效率的优化显得尤为重要. 下面列举出作者经过总结和思考得到的优化方法.

11.1 减少不必要的递归计算

当光线经过多次变向而减弱到一定程度, 大部分时候就可以直接忽略此光线而结束递归.

11.2 叠加方法添加概率模型

一般情况下, 只有在物体边缘处颜色会产生剧烈变化. 当使用随机扰动或采样法对某一位置计算时, 可以采用如下策略:

当经过数次计算, 发现此区域颜色变化不大时, 可以立即结束此处采样; 反之可以不断重复采样, 直到样本数量足够为止.

经过测试, 在渲染折射效果图 (本文图 11) 时, 通过直接重复采样计算软阴影耗时约 14 秒, 未修改其他地方, 经过优化采样概率模型, 耗时减少到约 6 秒, 而人眼无法观察出两结果的不同.

11.3 盒包围模型

光线追踪中, 很大一部分效率消耗在直线与三维图形交点计算里. 而许多光线不会遇到物体. 当物体数量多时, 可以通过盒模型来快速排除直线与几何形相离的情况.

盒包围模型采用了类似空间划分树的数据结构, 常用 BSP 或 K-D Tree 模型, 把空间分成许多立方体或球体, 每一部分完全包含了一些物体. 这些块与直线判断相交是极其容易的, 故可以直接排除与其内部物体的相交判断. 因此盒包围模型可以有效减少计算量.

11.4 基于硬件的优化

重写某些计算函数 (快速开根法等)、通过显卡 (GPU) 计算等等.

11.5 并行、分布式算法

利用多核 CPU, 将需要绘制的图像划分成许多部分, 并行计算或分布式计算.

12 展望

基本的光线追踪算法除基本光线效果 (反射、折射) 外, 仍不能实现许多更为真实、更复杂的物理效果. 例如绘制透明物体时, 光线产生的阴影下聚焦效果, 需要用到“光子映射”算法.

12.1 光子映射 (Photon Mapping)

此算法利用了光子在物体表面的概率分布模型.

12.2 辐射度算法

更为一般的“全局光照”模型, 需要用到全局渲染方程

$$L(x \to x') = k\left[E(x \to x') + \int_r \rho(x, x', x'')L(x', x'')\mathrm{d}x''\right]$$

计算光强分布函数 ρ（双向反射分布函数 BRDF）, 需要分别为不同材质选取不同的、较为复杂复杂的模型.

12.3 材质

与几何形不同, 例如流体、布质材料需要重新选取模型. 光线追踪算法不存在固定渲染模式, 是在实际应用中根据要求不断调整、重新修改的.

12.4 动态、自适应以及其他

用光线追踪实现不同频率光的色散、动态模糊、镜头光晕等等细节, 在概率计算部分加入自适应算法, 物体移动时实时计算包围盒等等.

参考文献

[1] Kevin Suffern, *Ray Tracing from the Ground Up*, A K Peters, Ltd, 2007.

[2] 沈一帆, 徐曼, 李宏宇, 彭源, 《计算机图形学》, 清华大学出版社, 2011.

[3] Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars, 邓俊辉 (译) 《计算几何：算法及应用》, 清华大学出版社, 2009.

[4] Dave Shreiner, The Khronos OpenGL ARB Working Group *OpenGL Programming Guide, Seventh Edition*, Pearson Education, Inc, Addison Wesley, Inc, 2010.

[5] 李静, 王文成, 吴恩华,《基于空盒自适应生成的动态场景光线跟踪计算》计算机学报, 32:1172–1182, 2009.

[6] 吴长茂, 张云泉, 郑海桥, 杨聪俐, 骆涛, 邱振戈, 谢金华,《基于大型场景的高精度成像并行光线追踪算法》, 2010.