光线跟踪实验报告

基科 11 刘心宇 2011012135

2014 年 1 月 6 日

目　录

1　实验目的 **2**

2　实验内容 **2**

3　实验原理 **2**
3.1　三维场景描述 . . . 2
3.2　基本光线跟踪算法（Ray Tracing） . . . 2
3.2.1　Phong 模型 . . . 2
3.2.2　光线求交 . . . 3
3.2.3　阴影 . . . 3
3.3　软阴影（Soft Shadow） . . . 3
3.4　纹理（Texture） . . . 3
3.5　反锯齿（Anti-aliasing） . . . 3
3.6　景深（Depth Of Field） . . . 3
3.7　空间加速（BSP Tree） . . . 3

4　编程实现——文件结构和类设计 **4**

5　效果演示 **4**
5.1　Phong 模型 . . . 4
5.2　反射与折射 . . . 4
5.3　阴影与软阴影 . . . 5
5.4　反锯齿 . . . 5
5.5　纹理 . . . 6
5.6　obj 文件导入 . . . 6
5.7　景深 . . . 7
5.8　空间加速 . . . 7
5.9　视频演示 . . . 8

6　实验出现的问题 **8**

7　实验总结 **9**

参考文献 **10**

1 实验目的

用 C++ 实现光线跟踪算法。

2 实验内容

- 三维场景的描述
- 基本光线跟踪算法（Ray Tracing）
- 软阴影（Soft Shadow）
- 纹理（Texture）
- 景深（Depth Of Field）
- 空间加速（BSP Tree）

3 实验原理

3.1 三维场景描述

可以用参数化的方法描述最基本的几何体包括平面，球体和立方体等。对于复杂的三维物体，可以用专业的建模软件建模，生成 obj 文件，并导入。

3.2 基本光线跟踪算法（Ray Tracing）

我们之所以能看到物体，是应为光线从光源出发经过一系列的反射折射最终进入视点。光线跟踪算法采用逆向的方法，假设光线从试点出发，投射在场景中，计算颜色，并对反射和折射光线继续进行跟踪，将所得颜色按照系数与当前颜色叠加，最终完成绘制，是一种递归算法。虽然在理想情况下，光线可以在物体之间进行无限次的反射和折射，但在实际算法中，当光线的强度小于最小阈值或者没有与物体相交时，终止递归。光线跟踪算法可以实现其它算法来很难达到的效果，它作为一个有效的真实感图形绘制算法被广泛地使用。[1] 光线跟踪伪代码如下：

Algorithm 1: RayTracing

```
RayTracing (start, direction, weight, color)
{
    if (weight < MinWeight)
        color = black
    else
    {
        计算光线与所有物体交点中距离 start 最近的点 P;
        if ( 没有交点 )
            color = black;
        else
        {
            Ilocal = 在交点处采用局部光照模型计算出的光强;
            计算反射光线的方向 R;
            Raytracing (P, R, weight*wr, Ir);
            计算反射光线的方向 T;
            Raytracing (P, T, weight*wt, It);
            color = Ilocal + Ir + It;
        }
    }
}
```

3.2.1 Phong 模型

在计算交点处的局部光照时，采用 Phong 模型。Phong 光照模型可以表述为：由物体表面上一点 P 反射到视点的光强 I 为环境光的光强 I_e、理想漫反射光强 I_d 和镜面反射光 I_s 的总和，即

$$I = I_a K_a + I_p K_d (L \cdot N) + I_p Ks (R \cdot V)^n$$

其中 K_d 为漫反射系数，K_s 为镜面反射系数。图1显示了 Phong 模型中的几何量。

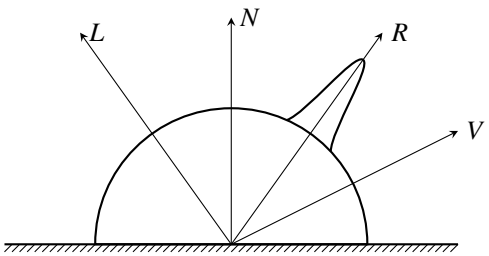


图 1: Phong 模型中的几何量

3.2.2 光线求交

针对不同的几何体，有不同的求交算法，在此不在赘述。

3.2.3 阴影

在计算局部光照模型时，判断图1中交点和光源的连线 L 是否与其他物体相交，如果相交，则该交点处在阴影区，否则不是。处在阴影区的点局部光强就是环境光，无漫反射和高光。

3.3 软阴影（Soft Shadow）

理想点光源，每一个交点只有是阴影和不是阴影两种状态，所以在阴影的边缘十分明显。但现实情况并非如此，为了实现软阴影的效果，可以将面光源看成若干个点光源，然后加权平均，所得到的阴影边缘会有渐变，更加真实。

3.4 纹理（Texture）

需要将物体表面的坐标和位图的坐标对应起来，也就是纹理坐标。由于选择的几何体都是参数化的描述，所以很容易建立纹理坐标，实现纹理映射。

3.5 反锯齿（Anti-aliasing）

消除物体边缘的锯齿效果，本次实验中采用最简单但并不高效的方法，用 4 倍采样，相当于高分辨率的图像缩小，会有效地减少锯齿。

3.6 景深（Depth Of Field）

景深是指相机对焦点前后相对清晰的成像范围。虽然透镜只能够将光聚到某一固定的距离，远离此点则会逐渐模糊，但是在某一段特定的距离内，影像模糊的程度是肉眼无法察觉的，这段距离称之为景深。

现实中景深通常由物距、镜头焦距，以及镜头的光圈值所决定。且焦平面前后的景深距离也并不相同。为了实现方便，我们假设景深与物距无关，且前后相同，因此用两个参数描述景深，焦距和光圈大小。

具体实现时，在做光线跟踪的同时，记录每一个像素的深度值（深度缓存）。然后用每一个像素的到焦平面的距离作为参数和光圈大小，做模糊处理。这个方法是基于深度缓存的后期处理，并不是对于透镜的模拟（可以用分布式光线跟踪算法），所以得到的效果可能并不那么真实，是一种近似处理。

3.7 空间加速（BSP Tree）

BSP 树是一种二叉树，用于加速光线与大量物体求交的过程。在树中每个叶节点上维护一个链表，存储该节点上的所有物体；同时在每个节点上存储一个 AABox(axel-aligned box)，表示该节点对应的空间区域。初始时树中只有根节点，且其上的链表为空，AABox 为整棵树所对应的子空间。

- **插入物体** 物体被插入到所有满足以下条件的叶节点中：从根到该叶节点的路径上的每个节点的 AABox 均与该物体相交。物体被插入一个叶节点时，如果该节点深度小于一个预设常数且该节点上已有的物体个数达到某预设常数时，先对节点进行分割再接着在子树上插入。

• **分割叶节点** 当某叶节点 U 上的物体个数达到某常数，且节点的深度未达某常数时，则需要对叶节点进行分割。分割方法为：为 U 分配两个子节点 V_1, V_2，将 U 的 AABox 沿着最长边等分成两个小的 AABox 并分别赋给 V_1, V_2；再使用上述插入算法，将 U 上原有的每个物体插入到以 U 为根的子树上，并清空 U 上的物体链表。

• **光线求交** 如果 U 是叶节点，则遍历 U 上的所有物体分别求出交点，其中距光源最近且在 U 的 AABox 内部的交点即为所求交点。
否则，先求出光线与 U 的两个子树 $V1, V2$ 的 AABox 的交点，不妨设与 V_1 交点的距离严格小于与 V_2 交点的距离 (如果不相交，则认为距离是无穷远)，或者两者距离相等但光源在 V_1 内部。此时，先递归求解光线与 V_1 子树中的物体的最近交点，如果有交点则为最终所求交点；否则再递归求解光线是否与 V_2 子树中的物体有交点。
需要特别注意的是，如果光线与 V_1, V_2 的 AABox 交点到光源距离相等但光源不在任何一个的内部，则应递归与两者上的物体都求交，再取最近者。

4 编程实现——文件结构和类设计

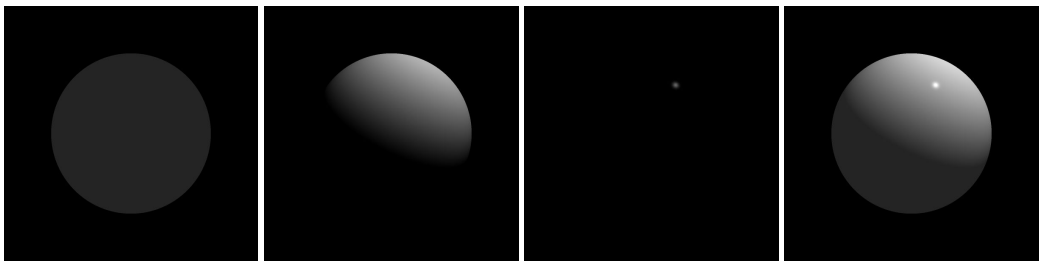
.\XY.h	全局函数定义，宏定义
.\XYClass.h	所以类型定义
.\BSPTree.cpp	AABox 和 BSPTree 类，实现空间加速
.\XYCamera.cpp	XYCamera 类，实现渲染，其中包含光线跟踪算法
.\XYLight.cpp	XYLight 和 XYLightSource 类，描述灯光
.\XYMath.cpp	XYVector3 类，实现向量运算
.\XYObject.cpp	简单几何体, 基类 XYObject，派生类 XYPlane,XYShpere,XYTriangle,XYBox
.\XYScene.cpp	XYScene 类，描述场景，物体的集合，处理求交，使用 BSPTree 加速
.\XYTexture.cpp	XYTexture 类，描述纹理
.\XYTime.cpp	计时模块
.\XY.cpp	其他辅助类，全局函数实现
.\main.cpp	测试用例

//网格模型解析（助教提供）
.\SimpleObject.h
.\Vec3f.h
.\SimpleObject.cpp
.\Vec3f.cpp

5 效果演示

5.1 Phong 模型

图 2中展示了 Phong 模型示例，即局部光照 = 环境光 + 漫反射 + 高光

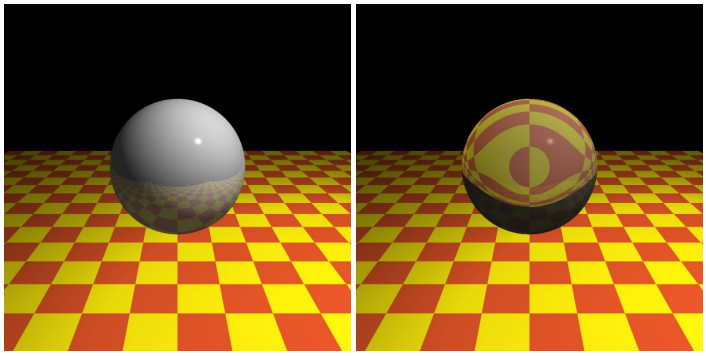


(a) 环境光　(b) 漫反射　(c) 高光　(d) 叠加

图 2: Phong 模型示例

5.2 反射与折射

运用光线跟踪算法求解全局光照，可以得到反射折射的效果，如图 3所示。



(a) 反射　　(b) 折射

图 3: 反射与折射

考虑可能会出现全反射的情况，如图 4所示。更多效果请见附件 tir.mp4。

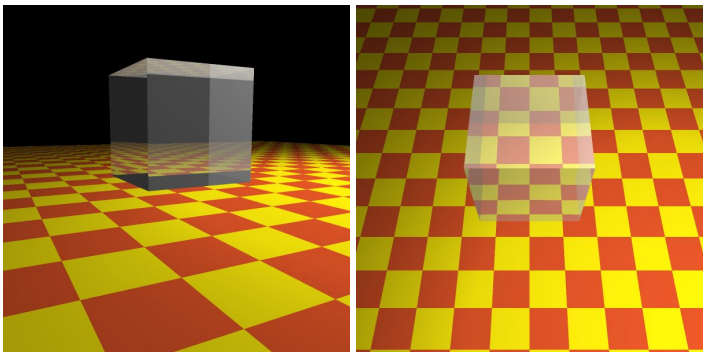
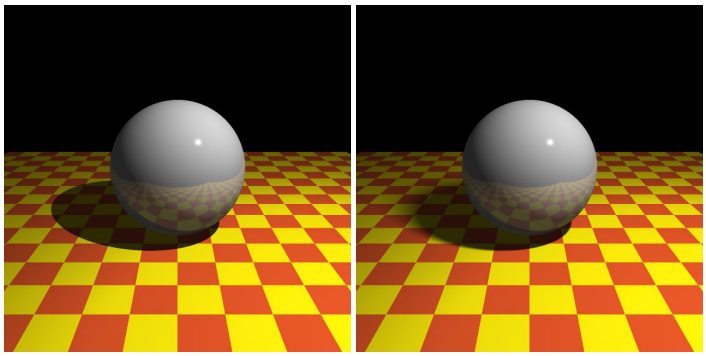


图 4: 反射与全反射

5.3 阴影与软阴影

在求解全局光照时，增加一次求交运算（见3.2.3），可以得到阴影效果。将面光源看成若干个点光源的集合（见3.3），就可以产生软阴影的效果，如图 5所示。

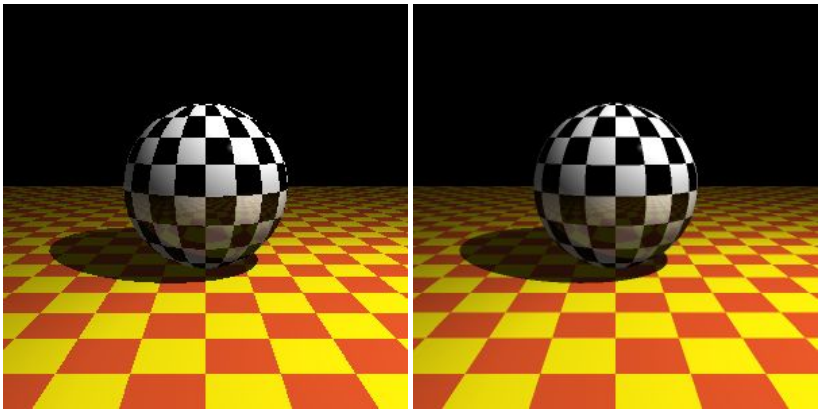


(a) 硬阴影　　(b) 软阴影

图 5: 阴影与软阴影

5.4 反锯齿

4 倍采样，得到反锯齿的效果, 如图 6所示。



(a) 1 倍采样　　(b) 4 倍采样（反锯齿）

图 6: 反锯齿

5.5 纹理

将物体上的坐标映射到位图上，如图 7所示。

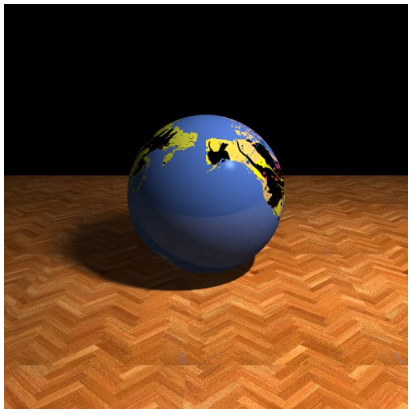
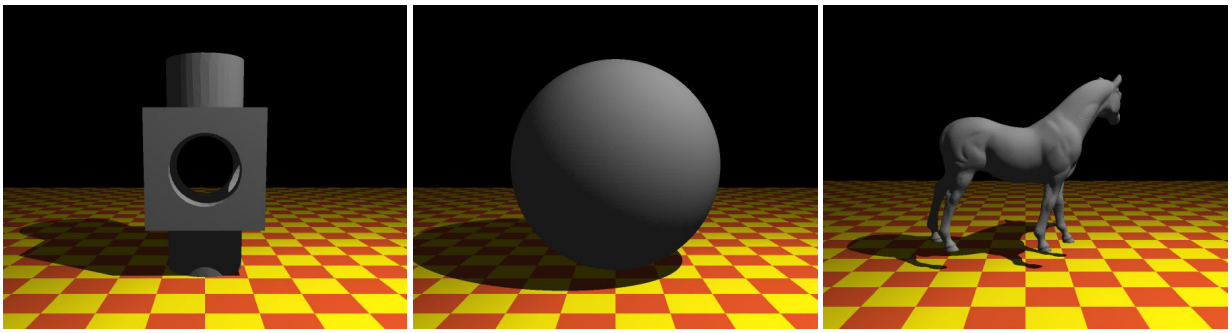


图 7: 纹理

5.6 obj 文件导入

解析 obj 文件，转化成网格模型，并渲染，如图 8所示。

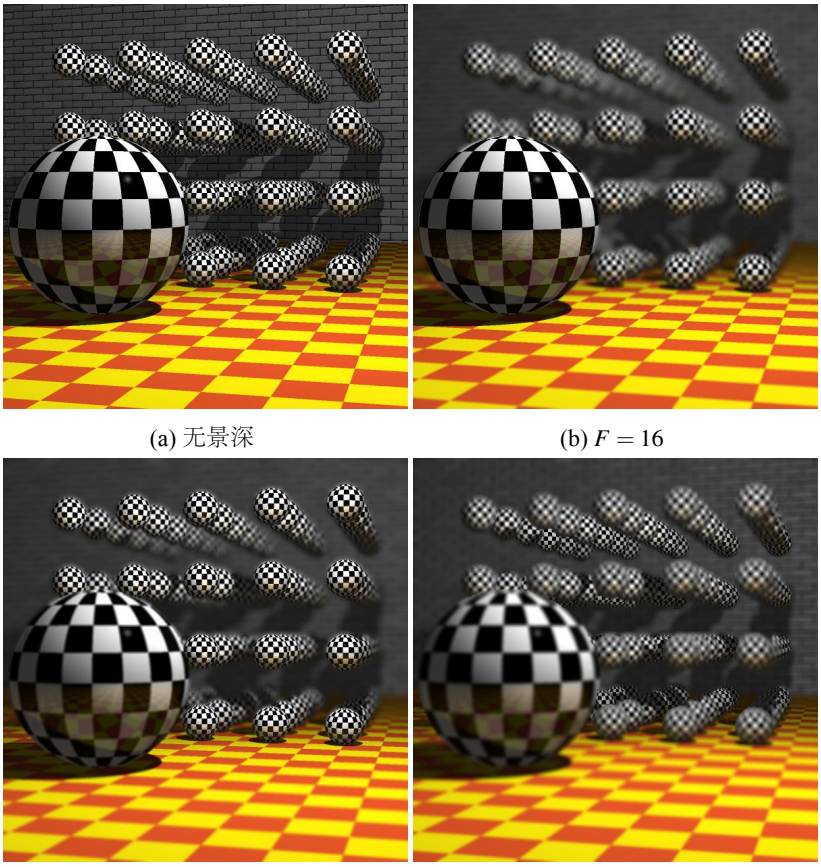


(a) block 4,272 Triangles　　(b) shpere 20,480 Triangles　　(c) horse 96,966 Triangles

图 8: obj 文件导入

5.7 景深

当光圈值固定的时候，改变焦平面的距离 F，实现不同的景深处理，效果如图 9所示。更多效果请看附件 dof.mp4。



(a) 无景深　(b) $F = 16$

(c) $F = 22$　(d) $F = 30$

图 9: 景深

5.8 空间加速

使用 BSPTree 对空间进行分割，提高求交的速度，可以将 BSPTree 对空间的划划分输出成 obj 文件，并查看，效果如图 10所示。

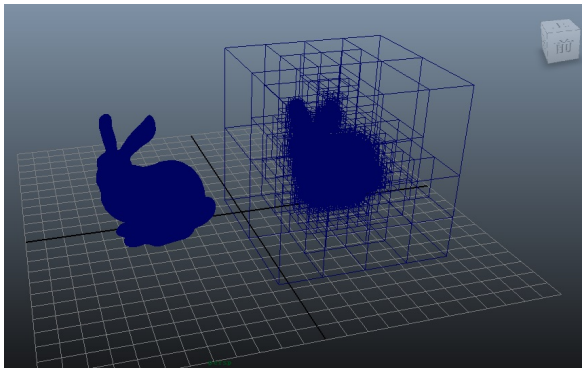


图 10: 空间加速 BSPTree

5.9 视频演示

综合了一些效果，如图 11，具体效果看附件 scene.mp4。

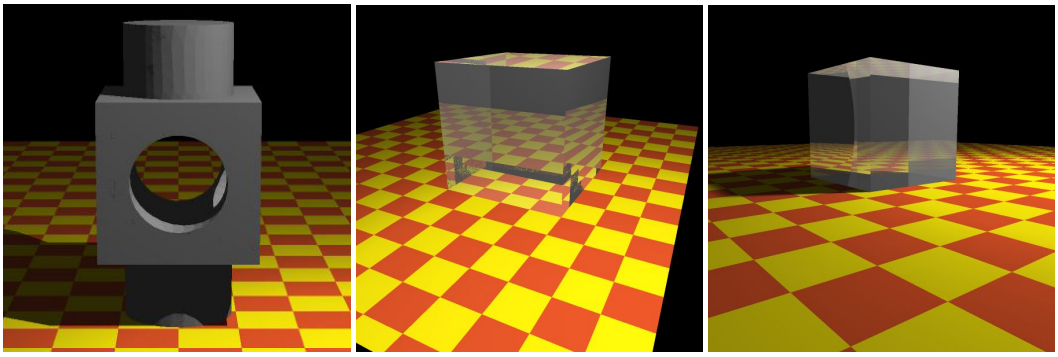
图 11: 场景

6 实验出现的问题

很多有问题的渲染结果并没有被保留。主要存在以下的一些问题:

- **法向量方向** 最开始只渲染了平面，颜色异常诡异，经过调试后，发现是没有考虑法向量的方向（有两个方向），导致漫反射光为负数，导致了渲染结果的异常。之后对所有模型考虑光照时，都考虑到了法向量方向。
- **颜色范围** 刚开始想加入多个光源，可是加入两个光源的时候，渲染结果的颜色也很诡异，然后当有多个光源的时候，将每个光源的光强降低，后来还是出现了很多问题，于是加入了颜色的范围，光强每个分量的大小只能为 $[0, 1]$。
- **软阴影** 最开始写软阴影的时候，考虑到效率的问题，首先判断一次求交，若处在阴影区，再对该店进行软阴影处理，即只对硬阴影区域内的点考虑软阴影，这样的话虽然阴影区有渐变的效果，但是最开始还是有一个突变，相当于原来是从 0 到 1 的跳变，现在是从 0 到 0.5 的跳变，然后渐变到 1。想了一些修改的方法，但效果并不是很好，而且会导致阴影区域缩小，最后还是采用每个点求阴影都把光源看成若干个点光源的方法，这样会导致软阴影的时候速度很慢。
- **景深** 利用的参数因该是该点到相机垂直方向的距离，而不是连线的距离。导致最开始的焦平面是弯曲的，由于没有意识到这个问题，因此花了一段时间去 debug。
- **BSPTree** 最开始分割的时候采用 XYZ 轮换的方式，发现还是会有一些不均匀，后来采用对最长的边进行分割。但有些 bug 好像还是无法避免，比如图 12(a)中物体中间的一些噪点，可能跟求交运算的误差有关，导致一些面片并没有被相交。将镜头后移动一些，问题就没有了，应该确实与浮点数误差有关。
- **浮点数误差** 这个一直是最头疼的地方，很多很多的 bug 都跟浮点数误差有关，比如在算全反射的时候，底面会有一些奇怪的黑影，如图 12(b)，后来发现是立方体和地面太近，折射进入，导致一会交在立方体，一会直接交在底面。将立方体稍微升高即可（在不透明的立方体好像不会存在这样的问题），效果可以见附件 tir.mp4。其中还有一个问题就是反射面的直线有形变，如图 12(c)，现在还没有解决，由于判断全反射的时候用到了三角函数和反三角函数，猜测可能跟其中的浮点数误差有关，以后有时间可以继续解决。在本次实验中，定义了全局的宏 EPSILON，用于误差控制，当两个数的差的绝对值小于 EPSILON，便认为两个数相同，可有效地减少误差带来的 bug。但 EPSILON 也不是越小越好，小到一定程度的时候会出现噪点。

- **obj 文件** 本次实验没有处理网格的平滑，有待以后继续解决。
- **其他问题** 整个渲染有镜头形变的感觉，在边缘处的球体并不那么圆，不是什么太大的问题，暂时还没有解决。还有就是编程本身的一些问题，很多 win32 api 不熟练，图形窗口，位图操作等。



(a) (b) (c)

图 12: 出现的问题

7 实验总结

本次光线跟踪实验解决了图形学中一个很根本的问题就是绘制问题，其中遇到了很多问题，基本上都已经解决，提高了编程的能力和对于真实感绘制的理解。还有一些改进的方法，分布式光纤跟踪，光束跟踪等，希望以后可以跟深一步的学习。最后感谢老师和助教的帮助。

参考文献

[1] 孙家广, 胡事民. 计算机图形学基础教程（第二版）. 清华大学出版社, 2009.