光线追踪总结报告

2012011372 赖国堃

2014/07

目录

1 程序描述

本程序实现了光线追踪算法，使用了phong光照模型，并且实现了包括了抗锯齿，景深，kd树加速与多线程加速，以及蒙特卡洛光线追踪等渲染效果，支持读取obj 文件，并且提供对读入模型进行法向平滑，uv纹理贴图等功能。通过算法以及优化可以兼顾计算时间与场景的真实感。我是用Qt实现了整个程序，但是编译过程采取静态编译，可执行程序应该可以在各个平台运行。

2 程序类介绍

2.1 Class Ray

这个类表示光线，简单的来讲就是一条射线，其中包含射线的源点与方向。

2.2 Class object

这个类包含场景中的物体，记录的此物体材质所有信息，比如有颜色，漫反射率，折射率，纹理信息等,主要提供与光线求交操作，能够返回交点，颜色与法向量。由于场景中的物体主要包含了球与三角面片。这个类被以下两个类继承。

2.2.1 Class triangle

object类的子类，负责记录与计算有关三角形的信息。

2.2.2 Class sphere

object类的子类，负责记录与计算有关球的信息。

2.3 Class light

记录有关光源的信息，主要包括光源位置与光源颜色。

2.4 Class scene

这个类包含了场景的所有信息，包括场景中物体的个数与信息，还有场景中屏幕与摄像机的位置，还有一些景深参数的设定，还有光源点的信息。程序中构造各种类时，主要通过传递这个类的指针。

2.5 Class raytrace

根据给定的scene类的信息，对屏幕中的每个像素点进行着色，着色过程调用光线追踪追踪算法。

2.6 Class kdtree

利用scene类中的obj信息构造kdtree，提供操作是通过给定光线返回最先碰到的物体的操作。

2.7 Class thread

这个类继承自QThread类，用于实现多线程同时运行，在实际操作中我选择了16线程运行。

2.8 Class vec3f

基础类，表示3维向量，重载提供三维向量的基本运算。

2.9 Class simpleobject

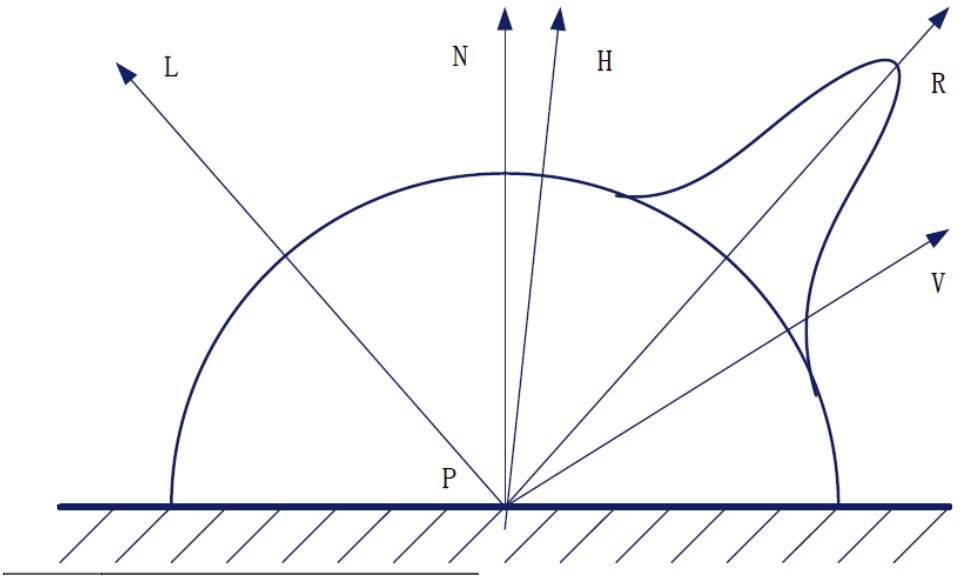
处理obj的读入问题，可以读入Vn（点法向量），与Vt（纹理向量）的信息。

3 程序算法描述

这里简单的阐述一下我实现的算法，和添加的优化，具体的效果，将在“程序亮点”部分给出。

3.1 phong模型

本算法主要采用光线追踪的框架，光照模型最初采用基础的phong模型。先给出phong模型的基本图示，



其中L是入射光；R是反射光；N是物体表面的法向量；V是视点方向。

phong模型将局部光照分为3 部分，第一部分为环境光照，其强度为

$$I_a = I_i * K_a$$

其中I_i与K_a 都是3维向量，前者为光源的颜色，后者为物体的环境光颜色，这一分量是由系统的环境光系数，以及物体颜色与物体材质的环境光系数组合而成。第二部分为漫反射光照，其强度为

$$I_d = I_i * K_d * (L \cdot N)$$

K_d分量同样由系统的漫反射系数，以及物体颜色与物体材质漫反射系数组合而成，L与N的意义见图示。最后一种是镜面反射光，其强度为

$$I_s = I_i * K_s * (R \cdot V)^n$$

其中K_s分量也由系统的镜面反射系数，以及物体颜色与物体材质镜面反射系数组合而成，而n为一个系统参数。最后这一点的由视点所看到的的光强就为

$$I = I_a + I_s + I_d$$

其中系统系数与物体系数均应取$(0, 1)$的数，这些系数可以根据实际情况进行调整获得最佳的效果。

3.2 阴影、反射与折射

3.2.1 阴影

阴影效果可以由一个点与光源连线是否中间是否被物体阻挡判断，如果此连线（即phong模型中的L）被阻挡那么此光源对这个点的亮度不起作用。

3.2.2 反射

可以通过入射光线以及交物体法向，计算出反射光线，然后进入下一层光线追踪，将反射光线的颜色乘上反射系数，再加到当前点的颜色上。

3.2.3 折射

利用入射光，交点法向，以及折射率，就可以通过折射定律，算出出射光，然后进入下一层光线追踪，然后将折射光线乘上折射系数累加到当前点上。

3.3 kd树

完成前面部分已经能够显示出一张不错的图片，但是由于求交耗费时间太大，所以比较难显示出一个较为复杂的模型，接下来就需要加入数据结构加快求交运算的速度。

首先对每个物体先构造一个平行于坐标轴的包围盒，然后利用包围盒建立一个kd树，我所采用的建树方法是将物体按某一维排序之后通过按中间的物体，分成左右子树分别包含左右两部分物体。在光线遍历树的时候，先观察里接近光源的子树得出最近交的物体，如果这个结果一定优于光源到另一个子树的距离那么就不遍历另一个子树。

这个算法在遍历是复杂度并不能保证总是O(logn)，但是实际效果十分出色。并且我对于一个物品序列的分割是直接取中点，如果在分割点的选取上添加估价函数应该还可以取得更好的效果。

3.4 抗锯齿

现在我们已经能够显示出比较复杂的物体了，但是我们可以发现物体的边缘充满了很多锯齿，一个粗暴的解决锯齿问题的方法是直接在屏幕的一个像素点中多取几个位置，然后将这个像素点的颜色取为这几个位置颜色的平均值。

这个方法效果很好但是速度有点慢，所以我们可以取那些颜色与周围像素点相差巨大的像素点做抗锯齿处理，这样就能取得很好的效果也不会减慢速度。

3.5 纹理

贴纹理的主要思想是将当前点的坐标，映射到二维纹理上，这样我们就能取得当前点贴上纹理之后应该有的颜色。

对于一个球来说，球坐标本身就是2维的，所以我们可以直接通过球坐标转换为二维平面坐标。

对于一个三角形来说，我们如果知道三角形的3个顶点对应的映射坐标，我们可以通过重心坐标系对纹理坐标进行插值取得当前点对应的映射坐标。对于三角形顶点对应的映射坐标，有的obj文件自带贴图的话，会在obj中给出每个三角形顶点对于的vt坐标（详细可以查obj文件格式）。对于没有自带vt坐标的模型，想贴上图的话，我们就可以自己定义一下vt坐标，只要保证顶点对应的映射坐标2维连续就能取得比较好的效果。

3.6 平滑法向

由于模型是由三角面片构成，所以比较粗糙的模型可以很明显得看出三角形面片。为了避免这种情况，我们希望求出的法向较为连续，最后的效果能像一个曲面，感觉光滑。我们可以假设每个顶点都有一个点法向，然后再三角形面片中的点同样可以通过插值方法求出其对应法向。

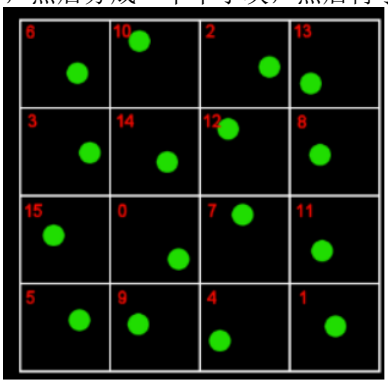
对于顶点法向计算有很多方法，我是通过一个顶点周围的三角形面片的法向做夹角的加权平均求得顶点法向。

3.7 Monte Calo渲染

这一部分主要是利用Monte Calo随机采样方法对光线追踪的效果进行优化。

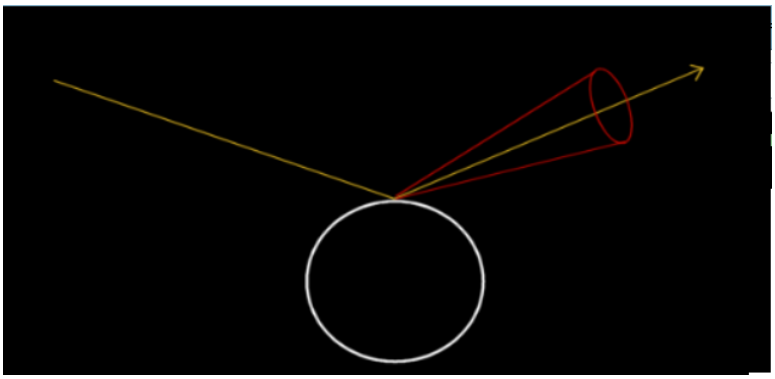
3.7.1 软阴影

由于在phong模型中的光源是点光源，所以最后显示的阴影的边界特别明显。但是在现实中点光源不存在，所以阴影的边界应该是模糊的，有一种渐变的感觉，所以我们可以假设光源有一定面积，然后通过随机采样，在这个面积中随机取一些点作为光源，计算阴影做加权平均值。具体我们可以将光源设为矩形，然后分成一个个小块，然后再小块中随机取点作为光源，如下图



3.7.2 反射

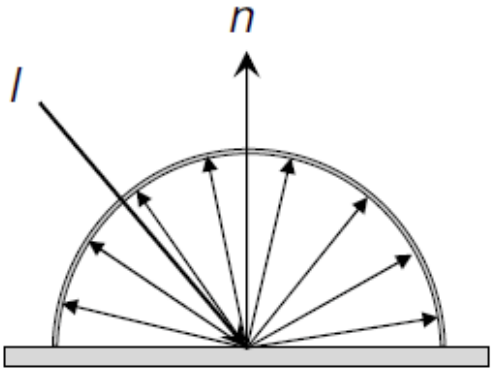
之前我们假设反射光只有一条射线，现在我们假设反射光是以反射射线为周的一个细小圆锥，如图



这样可以描述一个较粗糙的表面的反射效果。同样我们也是在这个圆锥中随机采样几个光线取均值，来描述这个圆锥返回的颜色值。

3.7.3 环境光

phong模型的对于环境光的处理是直接加上一个亮度，但是根据Lambertain模型，漫反射光应该是发散到四面八方的，如图



Lambertian

我们同样可以对这些光线进行随机采样，用这些光线效果的平均值取代原来phong模型中的环境光，这样获得效果就很有质感。

3.8 景深

对于景深的处理我采用了一个较为粗暴的方法，我利用物体到镜头的距离与镜头焦距的距离插值来定义一个模糊半径，然后根据距离的顺序进行插入，然后利用二维高斯模糊函数，进行模糊化处理。这样我取得了不错的效果，并且这个方法速度很快，可以根据已算好的图片进行快速绘制。

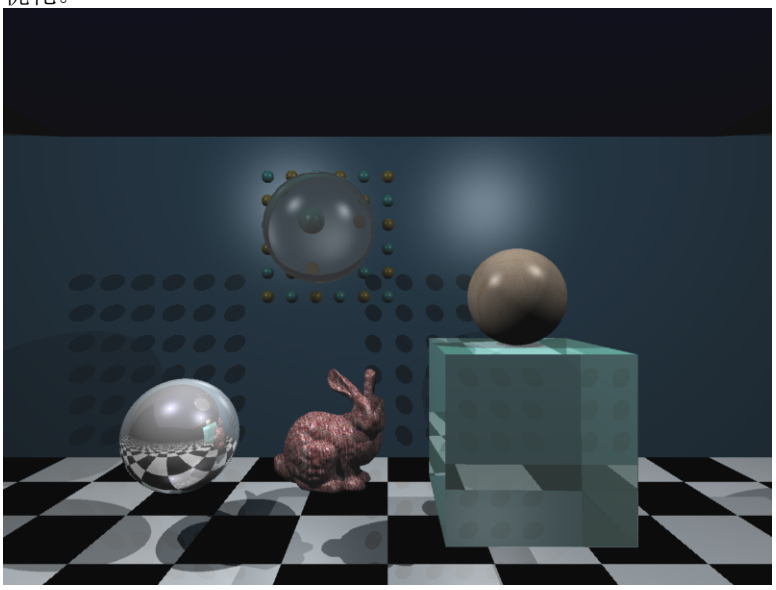
3.9 多线程

对于效果的算法优化就以上这些，最后我加了一个多线程加速，把整个屏幕分成16块，然后用16个线程同时运行，这样速度上能够快上不少。

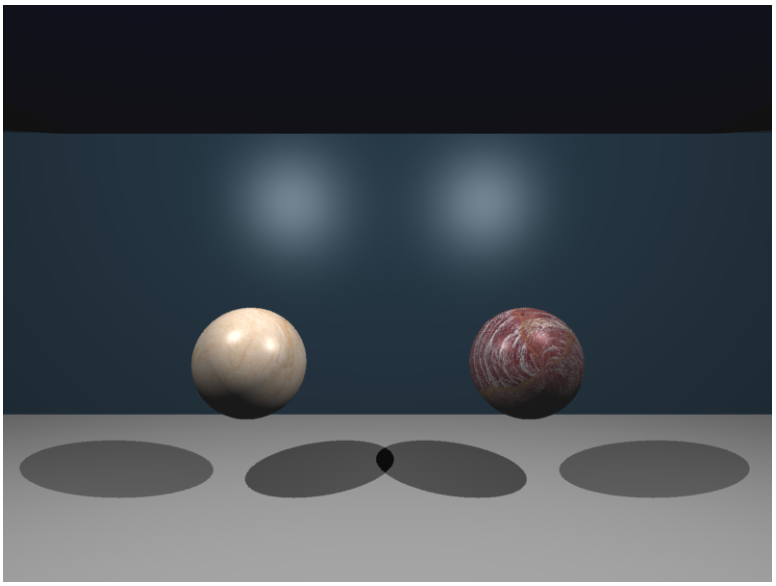
4 程序亮点

实现了大作业的基本要求和附加要求，并且拓展了很多渲染效果，使得绘制有较快的速度，较好的真实感，接下来给出一些图片，展示优化渲染的效果。

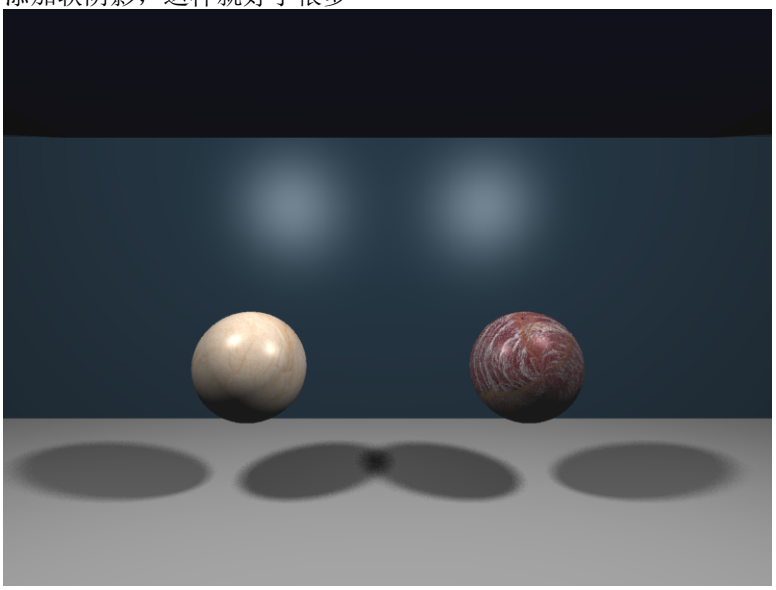
首先是没加渲染效果的一张图，实现了光照的phong模型，并且使用了kdtree加速与多线程加速，给普通物体表面贴上纹理，对其实现了4倍抗锯齿优化。



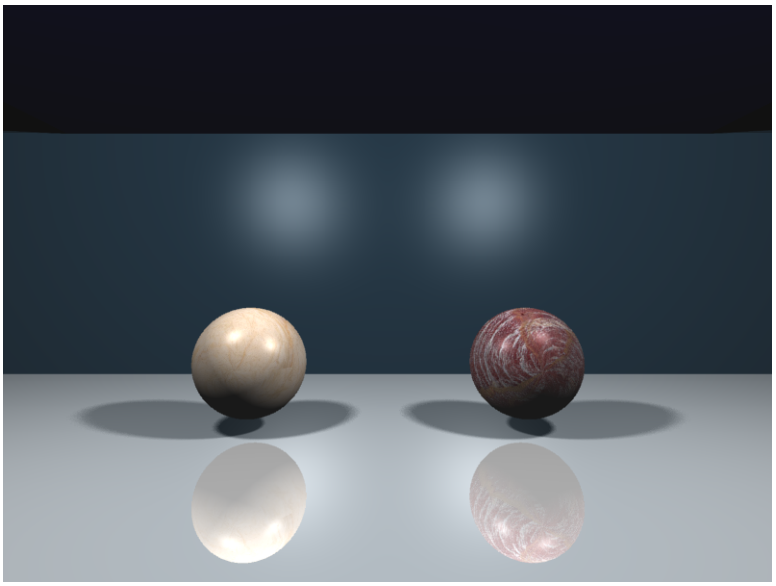
无软阴影，可以看到阴影边界相当明显



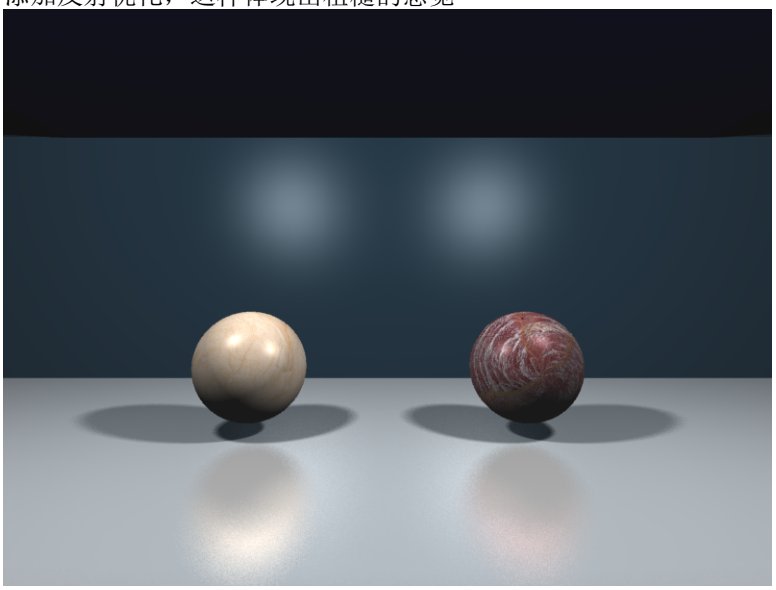
添加软阴影，这样就好了很多



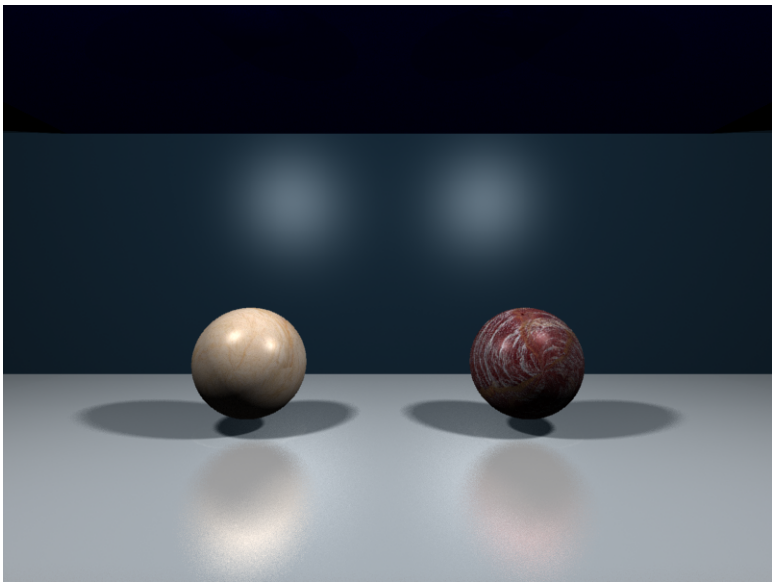
无反射优化，这样的反射感觉地板太光滑



添加反射优化，这样体现出粗糙的感觉

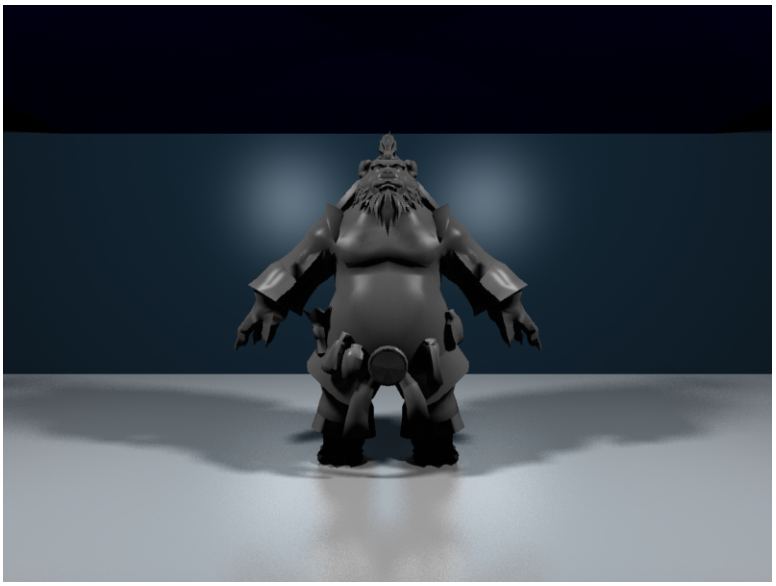


添加环境光优化，比之前的图明暗对比更好

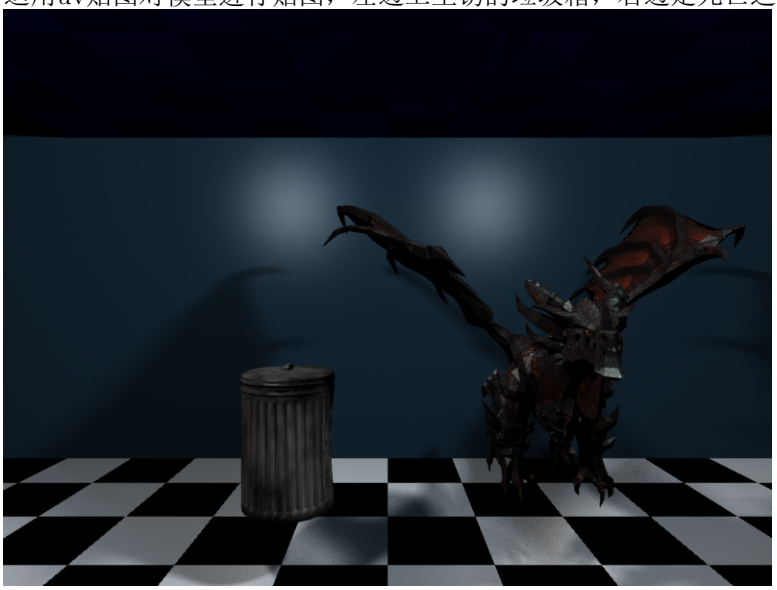


无平滑法向，明显肚子部分全是三角面片

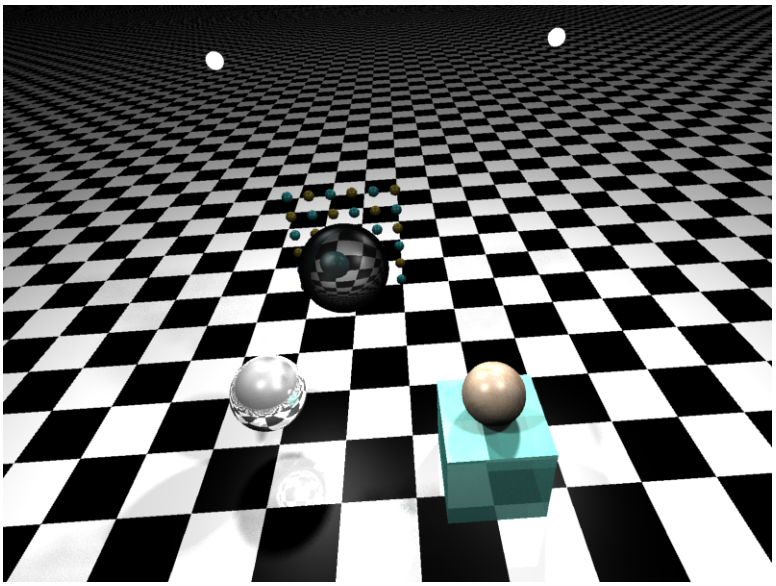
添加平滑法向，瞬间变光滑许多



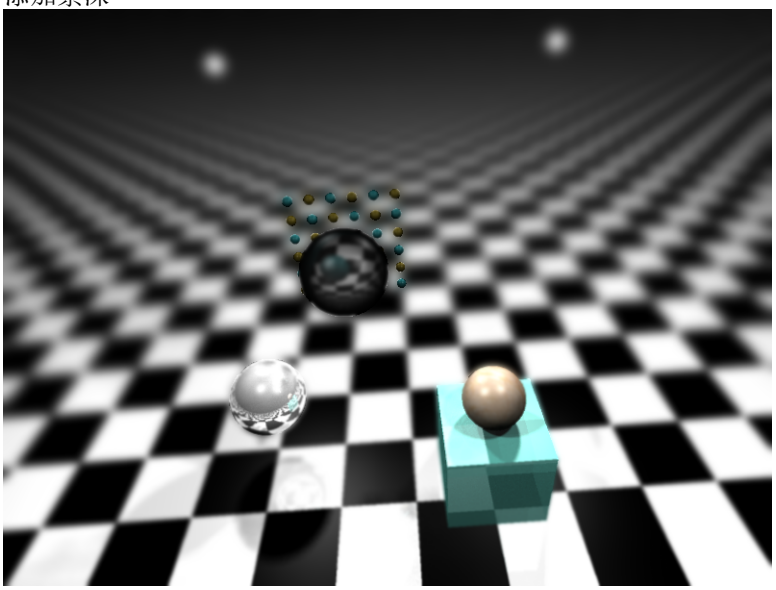
运用uv贴图对模型进行贴图，左边卫生锈的垃圾箱，右边是死亡之翼。



无景深



添加景深



5 程序运行结果

程序亮点中均是程序跑出的图片，在附件中还有一个10s的视频。

6 总结与收获

6.1 遇到的困难

本次试验中，遇到最大的困难时刚开始的时候感觉无从下手，当自己完成了phong模型之后，对于光线追踪的基本概念有了一定了解，之后就通过查阅各种资料添加各种优化。

在写程序的过程中调试也是个很艰辛的过程，特别是kd树部分耗费了接近1天时间找bug，并且在三角形求交等地方也要注意精度问题，每次在计算反射折射时，可以给光源移动一个微小的距离，使得光源不会再物体上，减小精度问题。

6.2 收获

我对图形学方向有着浓厚的兴趣，这次大作业虽然花了很多时间，但是在不断的优化自己的显示效果时，感觉相当有收获，也很有乐趣。通过这个学期的学习以及这个大作业，我初步对图形学有了一些了解，知道了一些图形学的研究方向。并且提高了自己的代码和debug能力。

7 参考资料

在最早比较无从下手的时候参考了这个网站

$http://www.flipcode.com/archives/Raytracing\_Topics\_Techniques-Part\_1\_Introduction.shtml$

感觉对于初学者还是相当有帮助。