本节课，先要实现一个光线投射器，然后最终实现一个递归的光线追踪器；

**一、开始**

运行样例解决方案，使用如下命令行：

a4 -input ../data/scene01\_plane.txt -output out1.png -size 200 200

（尴尬的是，没有提供windows上的解决方案- -）

**二、需求总结**

* 光源和着色

需要实现一个点光源和Phong氏反射模型

* 平面、三角形和变换节点

使用面向对象的技术，使得光线追踪器更灵活和更可拓展。一个泛型的Object3D类将作为所有图元的父类，你的工作是需要实现特化的子类。

* 递归光线追踪和阴影

光线投射器工作之后，你需要在Renderer里面递归调用方法，并且需要决定投射的阴影光线的可见性。

**三、开始代码**

* ArgParser ，可以访问命令行参数

将命令行参数 存到parser里面的一些string字段里

* SceneParser，从文本文件中读取，并连接图片或者.obj文件来解析特定的场景。

你需要需要完成一些构造函数，以及Group::addObject方法，这些方法会在解析器中调用。记得看一看数据的内容和格式，来了解场景是如何指定的。

（1）ParseFlie()方法，解析了透视相机、背景、光、材质、物体组等数据。

（2）需要实现Camera类 Camera.cpp是空的（好像不用实现）；

（3）一个方法，判断从文本文件中读取的字符是否与预期值相等，使用assert(!strcmp(xx, "xxx")); 因为strcmp(xx,xx)==0表示两个字符串相等，那么!0自然是true了。

（4）使用fscanf从输入流中读入数据。参考readInf/Float以及其他read方法。

（5）parserLight里面包含两种光源解析方法，parseDirectionalLight()和parsePointLight()

（6）parserObject方法，包含了各类物体的解析方法，比如parserGroup，parserSphere，parserPlane等，对于三角形和网格的解析相对复杂，要注意下。

* Image类用来初始化和编辑图像的RGB值。这个类也包含了存储简单png图像文件的方法。最终的产品需要接受场景的文本文件、计算正确的像素颜色，并存储为png。基本在Renderer类里实现。

（1）image类里面存储了宽度、高度以及一个vector<vector3f>data字段，data用来存储每一个像素的颜色；

（2）加载图片仍然使用了stbi库的stbi\_load方法。

（3）有三个方法,loadPNG,savePNG和compare方法，compare方法是将两张图片每个像素点的颜色差值（绝对值）,作为一张新图片的像素值。

* 我们提供给你了一个Ray类和一个Hit类来操控相机光线和他们的交点（在Ray.h中）。光线类包含一个源点和一个方向向量，Hit类存储了最近交点（以时间t表示，通过光线公式可计算点的位置）以及其法线，光线参数t，和指向交点物体材质的指针。

可以使用cout来输入一个ray。

* Hit的数据结构必须用一个很大的t值来初始化使用（std::numeric\_limit<float>())，计算交点的方法会通过存储最近的t和交点物体材质来更新它。

（1）使用了std::numeric\_limits<float>::max()作为最大值，这个值是浮点型的最大值，因为时间可能会是一个非常大的值。

（2）可以使用cout来输出hit的t和法线。

**四、实现步骤**

总流程如下：main函数解析命令行参数和场景，并将该信息传递给渲染器。在那里，您将根据场景的对象，通过光线投射/跟踪计算输出图像中每个像素的值，使用您将编写的相交和着色方法。

（1）命令行参数传递了场景文本文件的文件名，然后存储在ArgParser里面的Input\_file字段里。

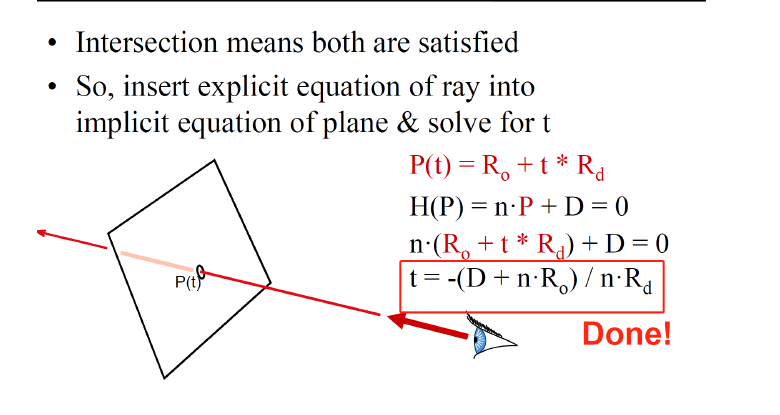
（2）渲染器renderer构造的时候，传递ArgParser对象存到\_args字段里，并且将该对象里面的Input\_file字段存到\_scene字段里（通过inpu\_file初始化构造的一个SceneParser）。

1. 检查抽象类Object3D，你不能直接创建一个抽象类的示例，但是可以通过初始化它的子类来使用抽象类。它有一个可以计算给定光线与物体是否相交的方法，并且存储了一个表示物体材质类型的指针；检查继承自Object3D的sphere类，并实现intersect()方法。sphere我们已经替你实现好了，你需要实现其他子类。

（1）计算球体和光线相交时，将光源移动到球体的局部坐标系中，相当于球体位于当前坐标系的原点，然后计算光线上的点到球心的距离，就可以使用点P的点积，即P·P了，P点积P在代码里面可以表示为Vector3f的absSquared()方法。

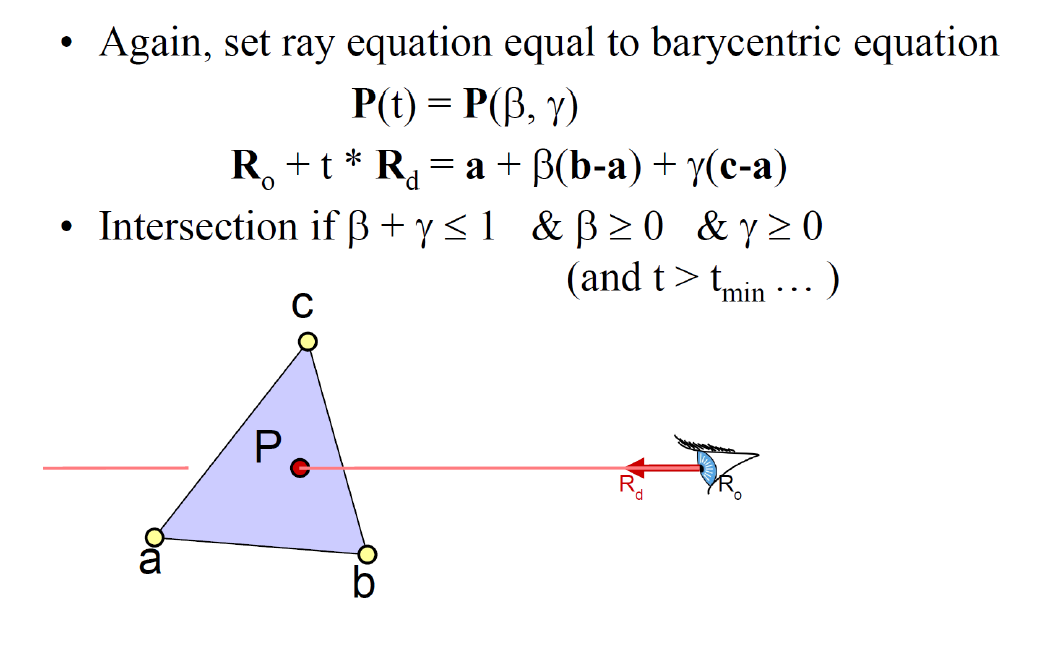
（2）intersect方法，需要通过公式计算，物体和光线是否相交，以及判断交点的位置是在相机前or相机后，然后取合适的根（t)，然后更新hit中最近的交点，设置相应的法线、材质等信息。

（3）Plane的相交计算

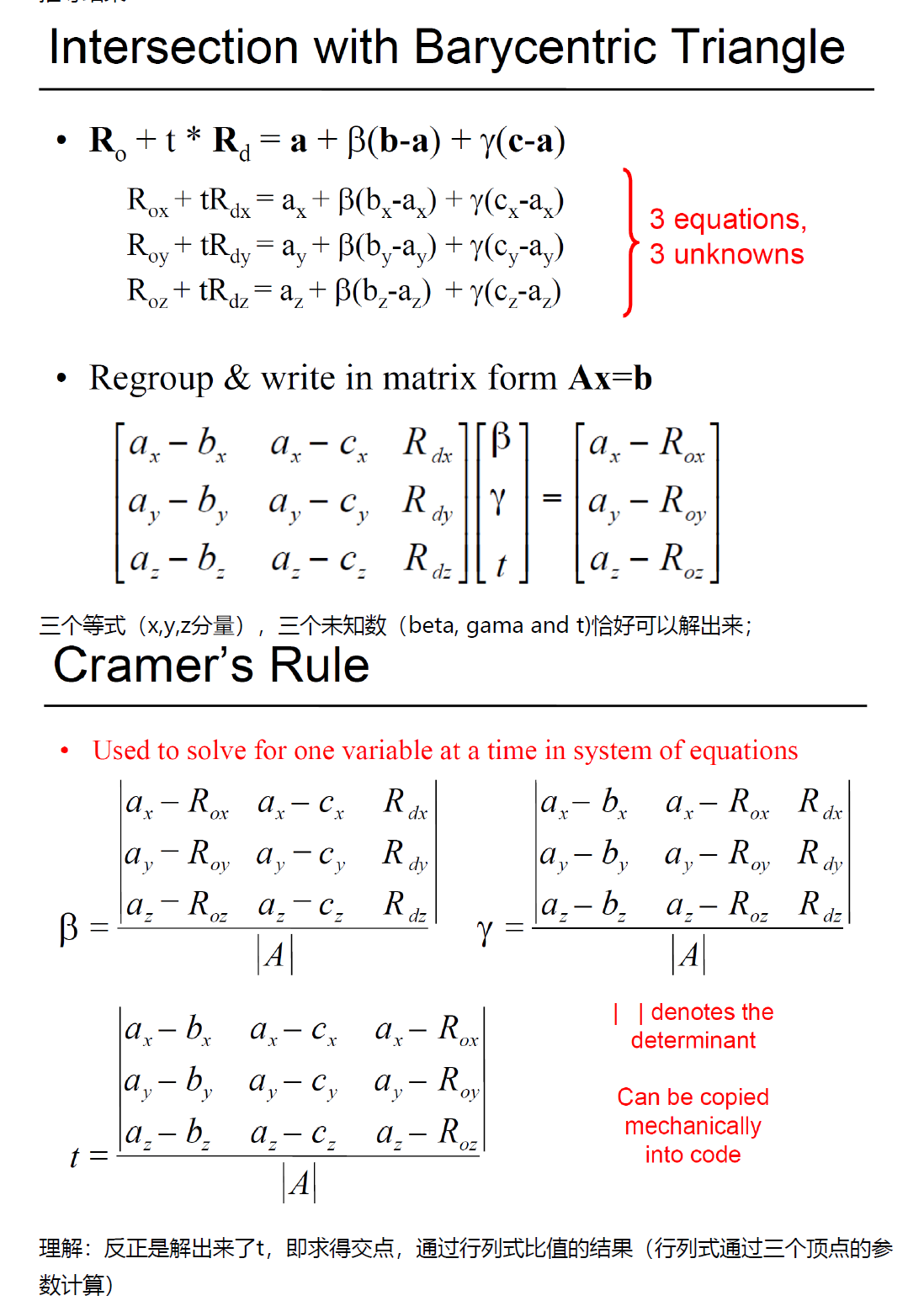


套用公式，记得光线的方向是没有长度的，应该是正交化之后的向量

（4）三角形的相交，使用重心坐标系，套用公式：



得出的别塔和伽马如果满足上述条件，则存在交点，使用公式计算：



下面第8点的建议是以矩阵的形式来表达和计算，更简洁

并且法线也应该通过重心坐标系的公式来进行插值计算。

2. 检查stater代码中的其他部分。

* 在Renderer::render()方法中，提供了一个光线追踪器的"每像素”循环，Render()对每条相机光线调用traceRay方法，你个工作是实现这个traceRay()方法。
* 看一看PerspectiveCamera，已经为你实现了，但是你要理解这个相机做了啥。

（1）在构造相机的方法里面，传递了相机中心，相机的朝向方向，一个辅助的世界向上up向量，和相机的视角大小（radians)；

（2）由于需要将屏幕定位标准尺寸[-1,1]，所以给定了相机视角大小，可以通过三角函数反推相机距离屏幕的位置是多少。即D = 1/tan(randians/2)；（实际上是将屏幕摆放在离相机多远的位置）

（3）计算屏幕中心的位置 然后再根据像素点的横纵偏移量 求得这条光线（想象两个三维向量相加）

* 在开始下一个步骤之前，看一看DirectionalLight，熟悉一下getIllumination()光源方法的接口。

3. 实现点光源

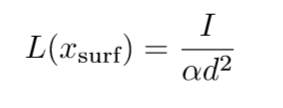
首先要实现PointLight::GetIllumination()方法，这个方法获取空间中的一个点，并返回：

(a) 场景点到点光源的方向向量（正交化后的）；

(b) 该点的光照强度（RGB)；

(c) 从场景点到点光源的距离；

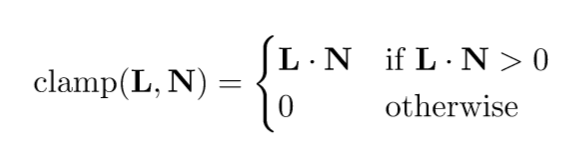
点光源距离场景点的距离为d，其光照强度xsurf是



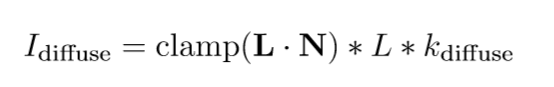
I是光源颜色，分母是距离平方。但是有时候距离平方的倒数曲线会过于陡峭，所以我们将其乘上一个衰减因子a（阿发）；

4. 实现漫反射着色

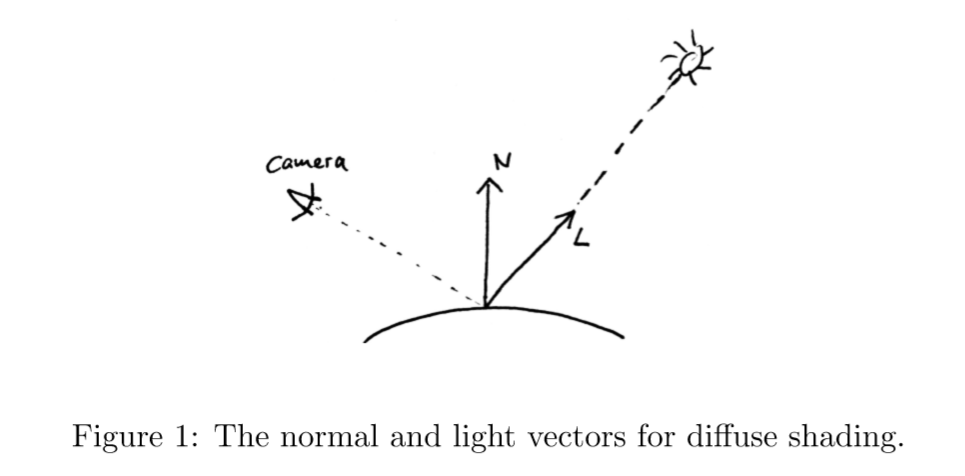
在Material::shade()方法中，实现Phong氏着色模型，给定光线方向L和法线N，我们需要计算插值的点积来作为漫反射的着色。（所谓插值的点积，如下：）



还需要乘上漫射系数(漫射的rgb值)kdiffuse和光照强度L，所以漫反射光照公式为：



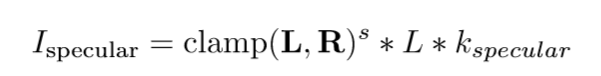
每个颜色通道各自计算。



#注意：漫射不考虑视线方向，只要光线（光源与物体的连线）未被遮挡，都会得到一个漫射值，当然需要乘上一个系数。

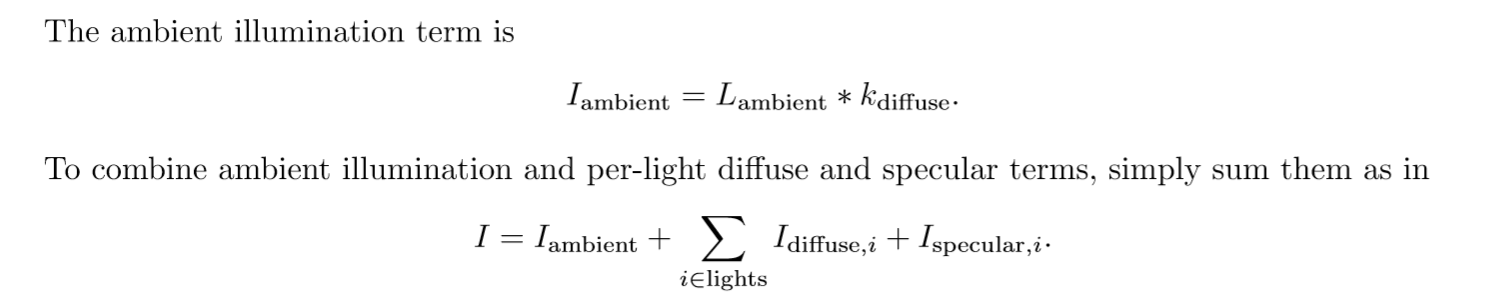
5. 实现冯氏着色模型中的镜面反射分量。镜面反射的强度依赖于shininess s, 表面到眼睛的光线E， E的全反射光线R， 光线方向L和法线方向N。

镜面反射公式为：



插值的点积和散射公式类似，但是这里使用了视线的反射光线R和光线方向L的叉乘，当相机移动的时候，这可以产生镜面的高光。同时，我们将插值的点积乘上了一个power s（幂次方）。shininess s的值越高，表示高光越密集，表面看起来更闪耀，更小的s则会使得表面更哑光。

6. 连接漫射射、镜面光和环境光。环境光照公式和总光照公式为：



I就是最终要写入帧缓冲的像素值。

7. 填充Plane。一个无限的平面图元继承子Object3D.使用你自己选择的表示法，但是构造函数一共视线在starter里。d是从原点的偏移值，意味着平面的公式是：P·n = d。视线intersect方法，确保更新存储在Hit里面的法线，以及交点距离t和颜色。你现在应该可以正确地渲染场景1的球体和平面。

8. 填充Triangle。同样继承自Object3D. 构造函数持有3个顶点，以及每个顶点的法线和材质。选择自己的方法来实现光线-三角形的相交计算。你将需要重心坐标系来插值三角形平面上的法线方向。（要解决3x3的线性空间，你可以以Ax=b的形式表示它并且使用Matrixf::inverse()来解决这个系统）

9. 填充Transform。Object3D的另一个子类。Transform存储了一个指向孩子节点的指针。同时也存储了一个4X4的转换矩阵M。此矩阵将“变换”节点的子节点从本地对象坐标移动到世界坐标。

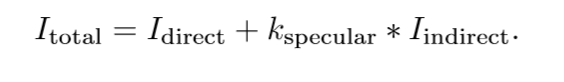
对于复杂的子对象（例如具有多个顶点的网格），每当我们要追踪射线时，都无法将整个对象移到世界空间中。 相反，将射线从世界空间移动到对象空间要便宜得多。 这是您必须实现的。

一旦发现交点，交点的发现是在物体的局部坐标系中的，你必须将法线从局部坐标转换到世界坐标，记住，当转换法线方向时，需要使用逆-转置矩阵（即先取逆，再取转置）。

参考Lecture2 part5

10. 对场景1-5运行你的光线投射器。恭喜，您已经创建了射线投射器！ 现在是时候将其扩展到光线跟踪器，以投射引起反射和阴影的辅助光线，以便我们可以在场景6中渲染闪亮的兔子并在场景7中渲染拱形。

11. 支持反射。traceRay()对于镜面材质将是递归的。 最大递归深度作为命令行参数-bounces max bounces传递。 在示例解决方案上使用此参数。 请注意，即使最大反射次数= 0，我们也会有效地进行光线投射（因为没有递归）。 通过将射线从当前的交点发送到全反射的方向，来实现反射材质的镜像反射。 这与冯氏模型镜面公式中的R方向相同。 使用修改后的递归深度，通过对traceRay的递归调用来跟踪二次射线。 将通过反射射线看到的颜色乘以镜面材质反射的颜色添加到为当前射线计算的颜色。 如果直接照明（环境，漫射，镜面反射）为*I*direct，则直接和间接照明的总强度为：



如果光线没有击中任何东西，只需通过SceneParser :: getBackgroundColor（dir）返回背景色。 实施反射后，您应该能够渲染反射兔子（场景6）。

12. 支持阴影。

要计算投射阴影，您需要将光线从表面点发送到每个光源。 如果报告了一个相交点，并且相交点比到光源的距离更近，则当前表面点处于阴影中，并且忽略该光源的直接照明。 请注意，阴影光线必须发送到所有光源。 回想一下，您必须将射线原点稍微移离表面，或将tmin等效设置为某个 esiplon。 祝贺您，有了阴影和反射，您现在可以正确渲染场景7。

**五、测试案例**

您的作业将通过运行入门发行版中提供的脚本来评分。 下面。 确保射线投射器产生等效的输出。 在开发过程中，可以使用以下命令测试渲染器。 入门发行版中的脚本显示了如何使用其他参数，例如-depth和-normals。

光线投射：

./a4 -input ../data/scene01 plane.txt -output 01.png -size 200 200

./a4 -input ../data/scene02 cube.txt -output 02.png -size 200 200

./a4 -input ../data/scene03 sphere.txt -output 03.png -size 200 200

./a4 -input ../data/scene04 axes.txt -output 04.png -size 200 200

./a4 -input ../data/scene05 bunny 200.txt -output 05.png -size 200 200

a4.exe -input ../data/scene01\_plane.txt -output ../output/1.png -size 200 200

a4.exe -input ../data/scene02\_cube.txt -output ../output/2.png -size 200 200

a4.exe -input ../data/scene03\_sphere.txt -output ../output/3.png -size 200 200

a4.exe -input ../data/scene04\_axes.txt -output ../output/4.png -size 200 200

a4.exe -input ../data/scene05\_bunny.txt -output ../output/5.png -size 200 200

**800：**

a4.exe -input ../data/scene01\_plane.txt -output ../output/1.png -normals ../output/1n.png -size 800 800 -depth 0 30 ../output/1d.png

a4.exe -input ../data/scene02\_cube.txt -output ../output/2.png -normals ../output/2n.png -size 800 800 -depth 0 30 ../output/2d.png

a4.exe -input ../data/scene03\_sphere.txt -output ../output/3.png -normals ../output/3n.png -size 800 800 -depth 0 30 ../output/3d.png

a4.exe -input ../data/scene04\_axes.txt -output ../output/4.png -normals ../output/4n.png -size 800 800 -depth 0 30 ../output/4d.png

a4.exe -input ../data/scene05\_bunny\_200.txt -output ../output/5.png -normals ../output/5n.png -size 800 800 -depth 0 30 ../output/5d.png

光线追踪：

对于光线跟踪，我们必须启用递归反射。 通常四次反射就足够了。 零反弹意味着仅是相机光线，根本没有递归。

a4.exe -input ../data/scene06\_bunny\_1k.txt -output ../output/6.png -normals ../output/6n.png -size 800 800 -depth 0 30 ../output/6d.png -shadows -bounces 4

a4.exe -input ../data/scene07\_arch.txt -output ../output/7.png -normals ../output/7n.png -size 800 800 -depth 0 30 ../output/7d.png -shadows -bounces 4

**六、提示**

•增量调试。 一次实现和测试一个图元。 一次测试一个着色附加组件。

•使用较小的图片大小以加快调试速度。 200×200像素通常足以意识到某些问题。 使用更高的分辨率（例如800x800）调试几何体和阴影的细节（我们将根据高分辨率图像进行分级）。

•与往常一样，请尽可能多地打印调试所需的信息，例如射线的方向矢量，命中值等。

•使用assert（）检查函数前提条件，数组索引等。请参见cassert。

•为避免段错误，请确保不要尝试访问超出图像宽度和高度的像素样本。 边界上的像素将具有裁剪的支持区域。

**七、额外提升**

注：此处为自己选择实现的一些效果

1. 抗锯齿处理 使用抖动采样和高斯模糊

根据ray-tracing in one weekend的做法，是在生成一条光线的时候，对相机原点的位置随机产生一个抖动的偏移，具体就是产生一个单位球体（圆盘）上的一个单位向量，然后将相机原点移动到这里，再去发射光线（射到同一个像素），相当于一个像素有多条临近的光线射向，采样到的物体也不是一个点，而是一个中心区域内的几个点，最后平均下来，就可以得到平滑的效果（这个过程/算法也称为高斯模糊）；

!记错了，这个是实现景深效果的模拟，抗锯齿处理，只需要对一个像素点的方格，进行进一步的细分采样，累计像素值再平均即可，进行细分的目标，是循环遍历时那个屏幕的偏移值。

测试：

1. 超采样抗锯齿

a4.exe -input ../data/scene01\_plane.txt -output ../output/1\_anti.png -normals ../output/1n\_anti.png -size 800 800 -depth 0 30 ../output/1d\_anti.png -jitter -filter 3 -shadows

a4.exe -input ../data/scene02\_cube.txt -output ../output/2\_anti.png -normals ../output/2n\_anti.png -size 800 800 -depth 0 30 ../output/2d\_anti.png -jitter -filter 3 -shadows

a4.exe -input ../data/scene02\_cube.txt -output ../output/2\_anti.png -normals ../output/2n\_anti.png -size 800 800 -depth 0 30 ../output/2d\_anti.png -jitter -filter 3 -shadows

a4.exe -input ../data/scene04\_axes.txt -output ../output/4\_anti.png -normals ../output/4n\_anti.png -size 800 800 -depth 0 30 ../output/4d\_anti.png -jitter -filter 50 -shadows

a4.exe -input ../data/scene05\_bunny\_200.txt -output ../output/5\_anti.png -normals ../output/5n\_anti.png -size 800 800 -depth 0 30 ../output/5d\_anti.png -jitter -filter 50 -shadows

a4.exe -input ../data/scene06\_bunny\_1k.txt -output ../output/6\_anti.png -normals ../output/6n\_anti.png -size 800 800 -depth 0 30 ../output/6d\_anti.png -shadows -bounces 4 -jitter -filter 50 -shadows

a4.exe -input ../data/scene07\_arch.txt -output ../output/7\_anti.png -normals ../output/7n\_anti.png -size 800 800 -depth 0 30 ../output/7d\_anti.png -shadows -bounces 4 -jitter -filter 50 -shadows

2. 折射

a4.exe -input ../data/scene01\_plane.txt -output ../output/1\_refract.png -normals ../output/1n\_refract.png -size 800 800 -depth 0 30 ../output/1d\_refract.png -bounces 4 -jitter -filter 10

a4.exe -input ../data/scene01\_plane.txt -output ../output/1\_anti.png -normals ../output/1n\_anti.png -size 800 800 -depth 0 30 ../output/1d\_anti.png -jitter -filter 3

a4.exe -input ../data/scene06\_bunny\_1k.txt -output ../output/6\_refract.png -normals ../output/6n\_refract.png -size 400 400 -depth 0 30 ../output/6d\_refract.png -bounces 10 -jitter -filter 10 -shadows

a4.exe -input ../data/scene07\_arch.txt -output ../output/7\_refract.png -size 200 200 -shadows -bounces 4 -jitter -filter 3

**作业笔记：**

1. Material类，存储3个字段，diffuseColor, specularColor和shininess，有一个shade方法，用于该材质的着色。

2. 在Camera.h 里面 增加了一个屏幕到相机的距离字段\_d，这样就不需要在generateRay方法里面，每次都要计算一次tan值了；

3. 环境光的计算 是判断不处于阴影之中才计算 还是说无论是否在阴影中 都要计算这个环境光？

（应该是只计算一次，只要视线与物体相交，无论该交点在所有光源方向上是否产生阴影，都会有一个基础的环境光，即这个环境光的计算是在遍历阴影光线的循环之外的）