# 光线跟踪

高林

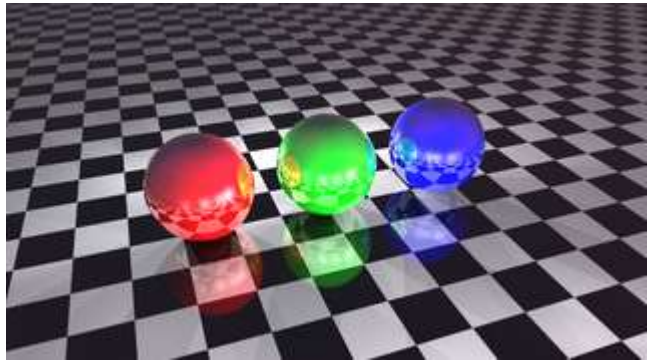
中国科学院计算技术研究所

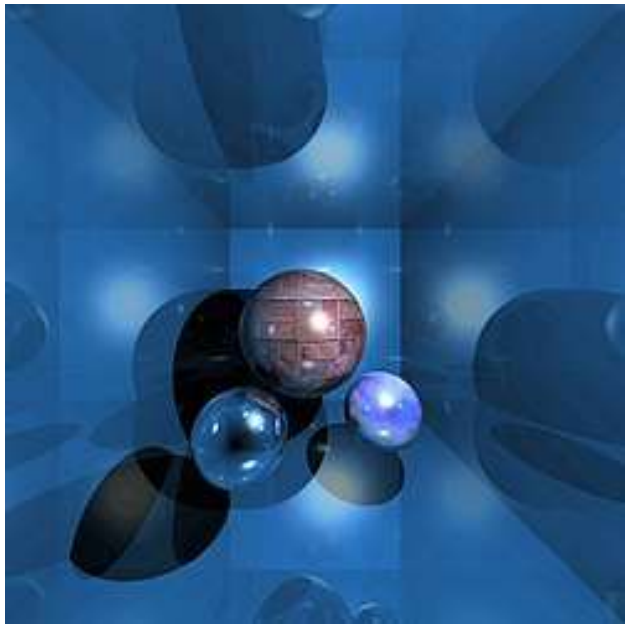
2023-04

# 什么是光线跟踪

- 一种用复杂的光线相互作用来绘制三维图形的渲染算法，追踪光线从来源开始照射到物体上，再由物体反射的光线“路径”

- 效果
  - 模拟真实世界的光源
    - 自然光、漫反射光、折射光、 …
  - 模拟真实世界的物体的材质
    - 镜面、透明、半透明、 …
  - 模拟真实世界的阴影效果
    - 软阴影、硬阴影、…





# 光线跟踪算法的背景

- 使用光线追踪技术，可以合成具有极高真实感的图像

  斯坦福大学的cs348b渲染竞赛

使用L系统对树木程序建模，使用元球对雪进行程序化建模，地下散射算法使雪具有逼真的外观

冰洞图像，物理模拟冰的融化和重新冻结，模拟由于融化的径流导致的冰的侵蚀

# 光线跟踪算法的背景

- CG与电影行业中，光线追踪被大量运用，渲染出令人震撼的画面效果

  Pixar公司RenderMan的R19版本以后转向RIS，基于光线追踪技术

  Disney公司动画部门的渲染器Hyperion，基于光线追踪，功能强大


# 光线跟踪算法的背景

- CG与电影行业中，光线追踪被大量运用，渲染出令人震撼的画面效果

  使用光线追踪技术来生成或强化特效，包括逼真的反射、折射和阴影效果

  使电影中的虚拟角色、火焰、烟雾和爆炸场景更加真实

# 光线跟踪算法的背景

- 游戏行业，光线追踪技术逐渐得到应用

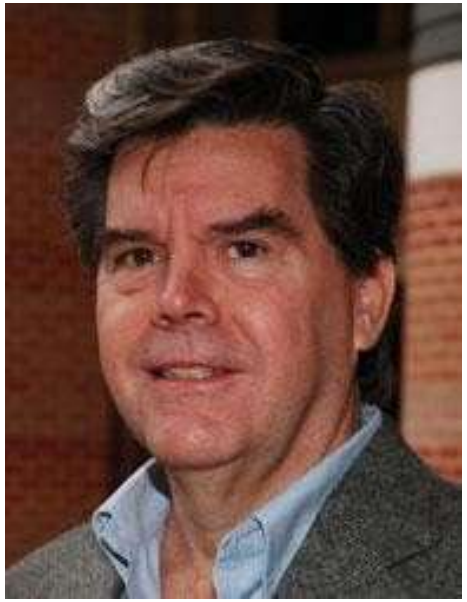
  Unity 2019.3正式支持实时光线追踪特性，部分3A游戏已加入实时光线追踪

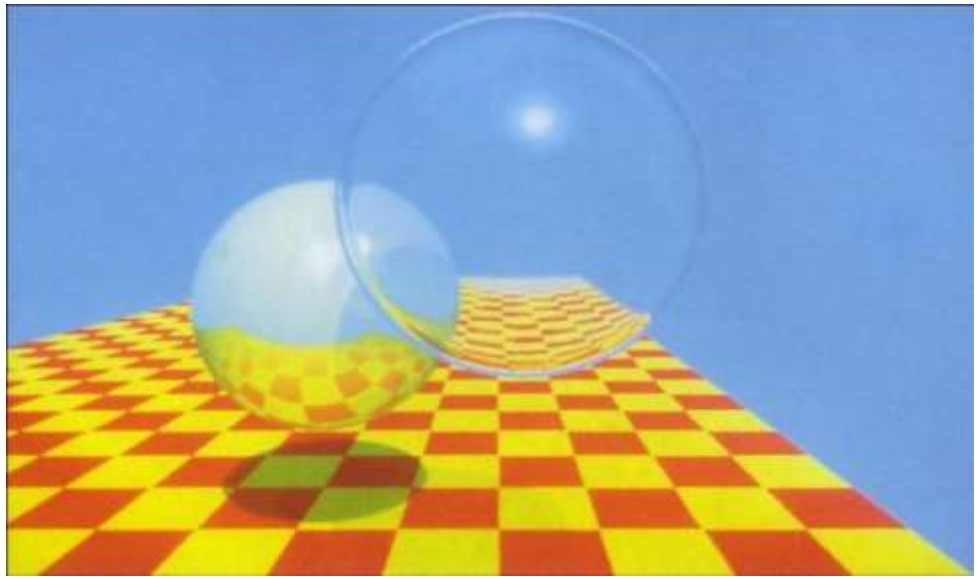

NVIDIA宣布了可加速硬件中光线追踪速度的新架构Turing


第一款搭载RTX实时混合光线追踪技术的游戏《战地5（Battlefield V）》正式面世

# 光线跟踪算法的历史

- Turner Whitted 于 1980 年首次提出一个包含光反射和折射效果的模型：Whitted 模型，并第一次给出光线跟踪算法的范例，是计算机图形学历史上的里程碑。
- Turner Whitted，An improved illumination model for shaded display, Communications of the ACM, v.23 n.6, p.343-349, June 1980.





# 光线跟踪算法的历史

- Turner Whitted 于 1978 年在北卡罗来纳州立大学 (NCSU) 获得 Ph.D.，之后来到贝尔实验室，提出了震惊世界的光线跟踪算法。Turner Whitted 一共只发表过19篇论文，其中光线跟踪算法是他的第一篇论文。
- Turner Whitted 于 2003 年当选美国工程院院士。


**Dr. J. Turner Whitted** (Print This)
Senior Researcher

**Primary Work Institution:** Microsoft Research
**Election Year:** 2003
**Primary Membership Section:** 05. Computer Science & Engineering

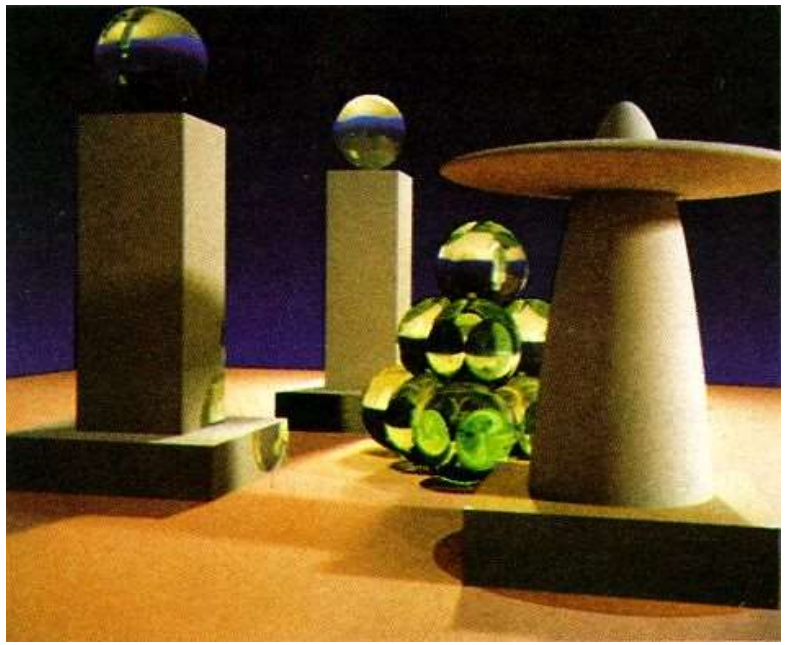
**Country:** United States
**State:** WA

**Member Type:** Member

**Election Citation:**
For contributions to computer graphics, notably recursive ray-tracing.

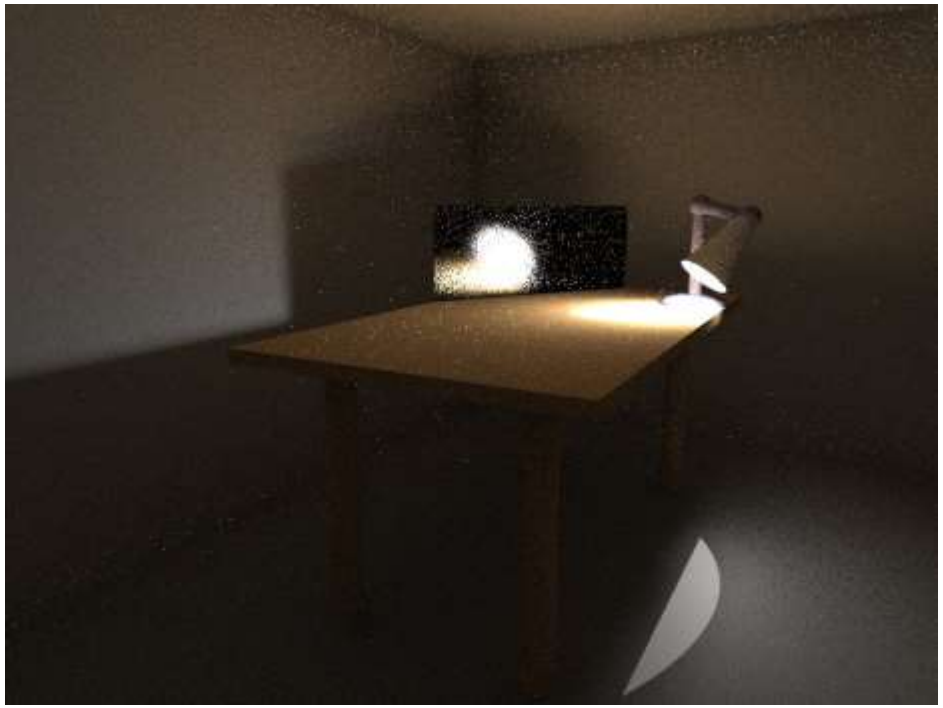
# 光线跟踪算法的历史

- 1984年，Cook 等人提出了分布式光线追踪（distributed or distribution ray tracing,DRT），使用了蒙特卡洛方法
- 1986年，Kajiya等人提出了渲染方程(rendering equation)，同时也提出了最原始路径追踪算法。



# 光线跟踪算法的历史

- 双向路径追踪：由Lafortune和 Willems提出，Veach则对双向路径追踪做了详细的描述。
- 梅特波利斯光照传输：1997年 Veach首次将最早应用在计算物理领域的Metropolis 采样方法引入图形学



# 光线跟踪算法的现状

- 离线光线追踪：光线追踪和颜色计算离线进行，即使设置了很高的光线追踪采样率，很多的反射次数，烘焙Lightmap的时间会变长，最终依旧可以渲染出很好效果。

- 实时光线追踪：就是随着摄像机视角的变动，后端需要实时发射追踪光线来重新计算光照信息，如果屏幕分别率很高，计算量很大，对GPU的性能要求很高

# 光线跟踪算法的现状


混合渲染管线（Hybrid Rendering Pipeline）

基于管线的混合渲染，指的是一条渲染管线中，会用到不同的计算承载方式，最终来完成渲染工作。计算承载方式包括光栅化、计算着色器、光线追踪着色器等

实时渲染(Real-time Rendering)：

反射渲染、环境光遮蔽、阴影渲染、透明阴影渲染、多光源处理、粒子系统渲染……

# 光线跟踪算法的现状

- 透明和半透明渲染实时渲染的发展近况：
  - 正确表示顺序无关的透明渲染（Order-independent Transparency，OIT）
  - 可变的粗糙度，折射和吸收（Variable roughness , refractions and absorption）
  - 多IOR过渡（Multiple index-of-refraction transitions）

# 光线跟踪算法的现状

- 多光源处理的实时渲染：
  - 基于加速结构的光源选择（Acceleration structure-based selection）
  - 基于重要性采样的光源选择（Light Importance Sampling）
- 全局光照的实时渲染：
  - 基于面元（Surfels）
  - 基于网格（Grid）
  - 基于探针（Probes）

# 光线投射算法

光线：

1.光以直线传播（大部分）

2.光线交叉时不会相互干扰（光子没有相互作用；忽略驻波效应）

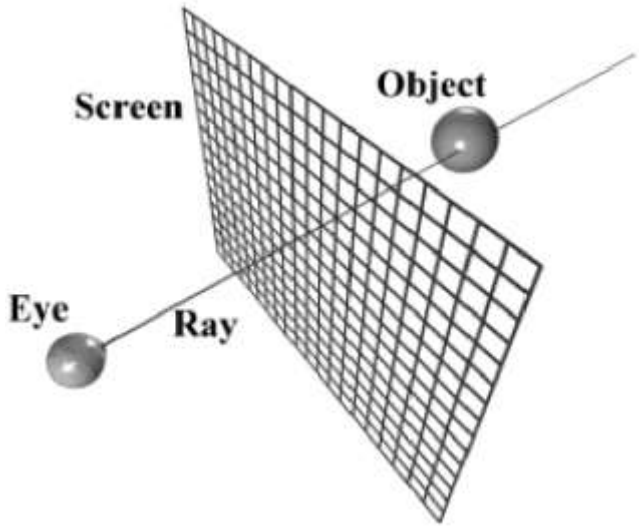
3.光线从光源传播到眼睛（在路径反转作用下，物理学性质是不变的）

# 光线投射算法

算法原理：

- 对屏幕上每一像素，执行下述3步操作：

  1. 从视点出发通过该像素中心向场景发出一条光线，并求出该条光线与场景中物体的全部交点
  2. 将各交点沿光线方向排序，获得离视点最近交点
  3. 依据局部光照明模型计算该交点处的光亮度，并将所得光亮度值赋给该像素

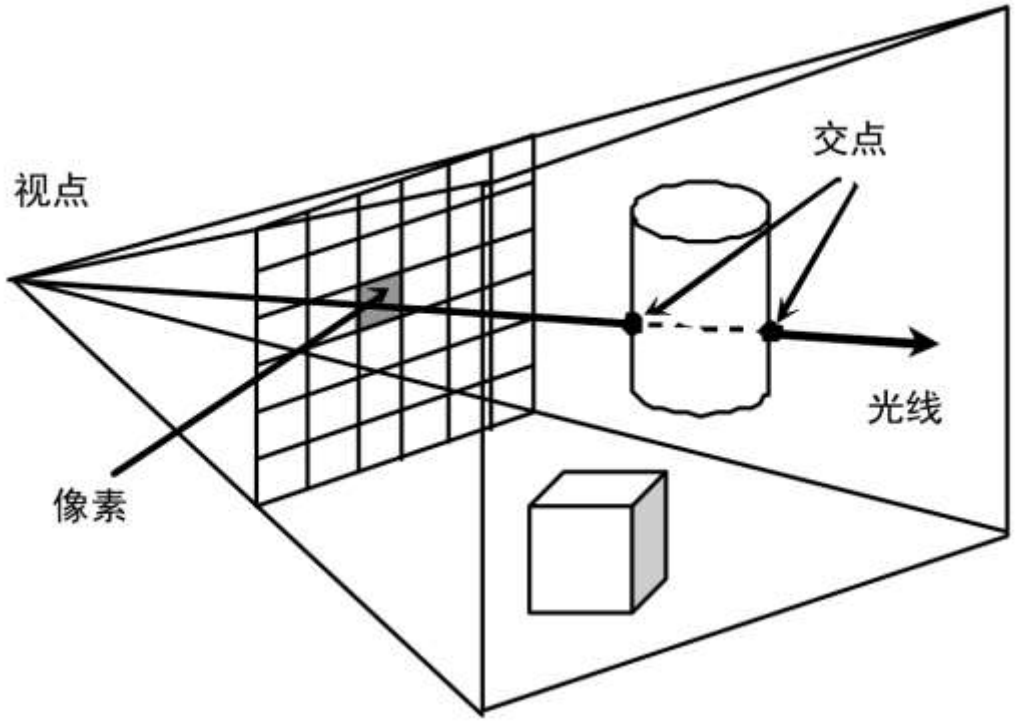



- 当所有屏幕像素都处理完毕时，即得到一幅真实感图形

# 光线投射

- 步骤1：

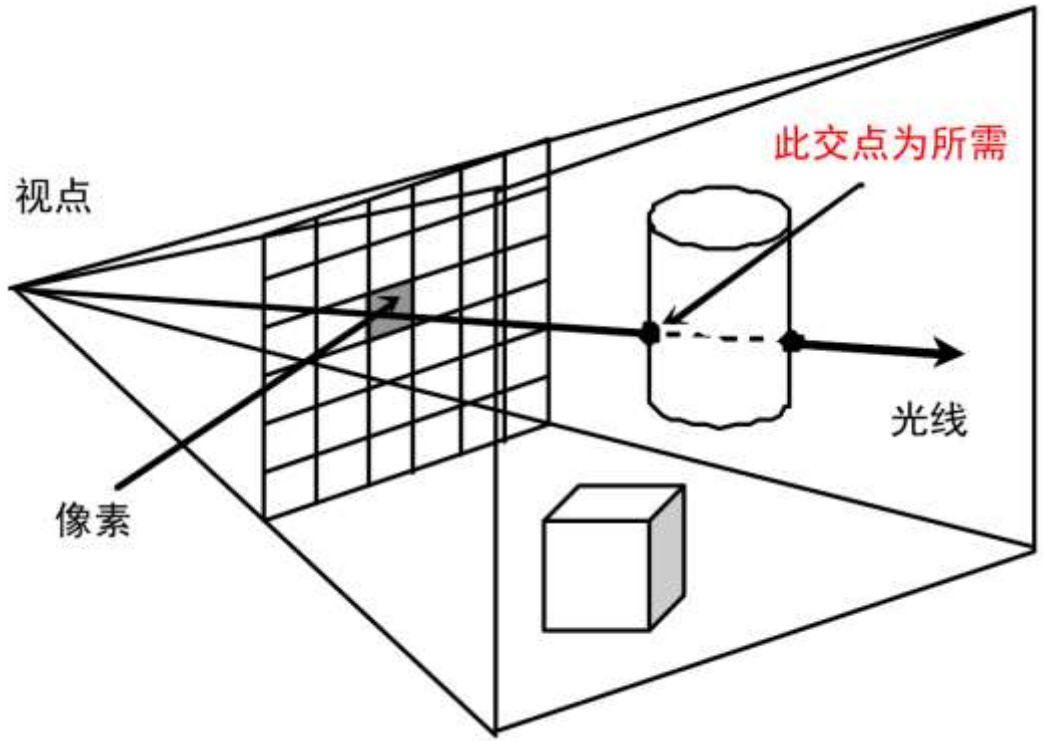
从视点出发通过屏幕上一像素中心向场景发出一条光线，并求出该条光线与场景中物体的全部交点



# 光线投射

- 步骤2：

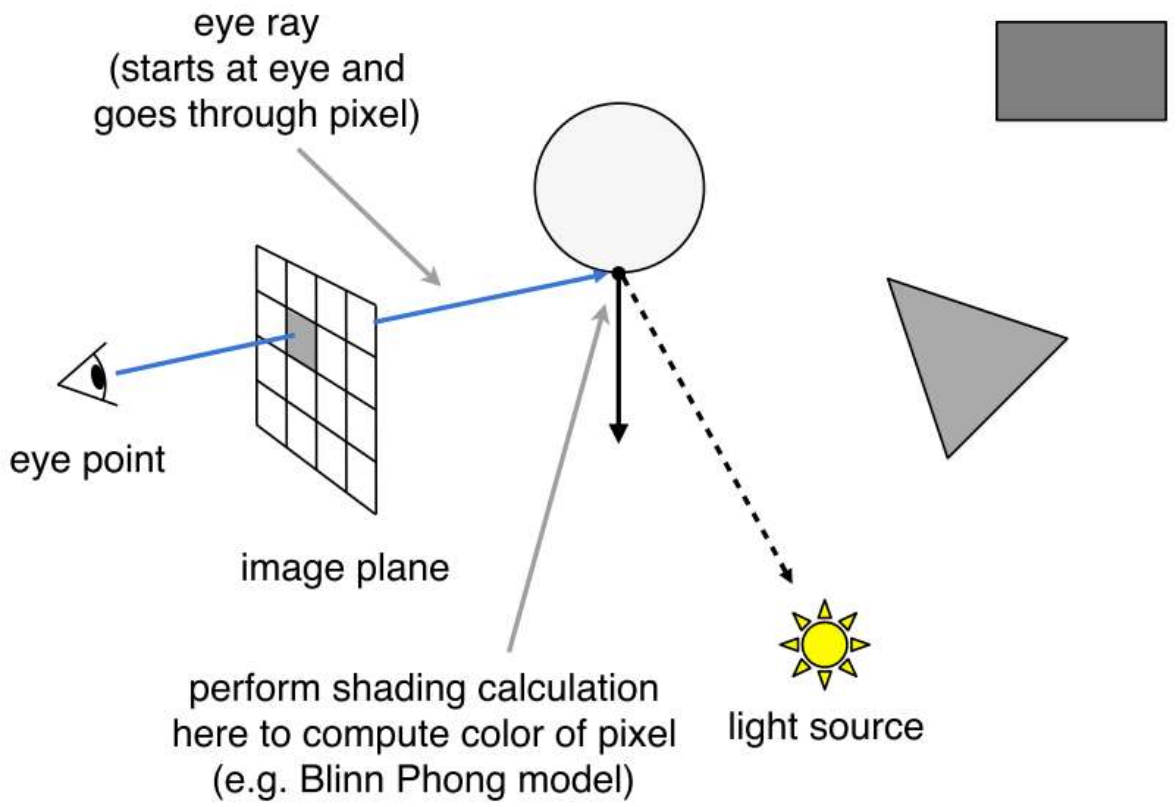
  将各交点沿光线方向排序，获得离视点最近交点



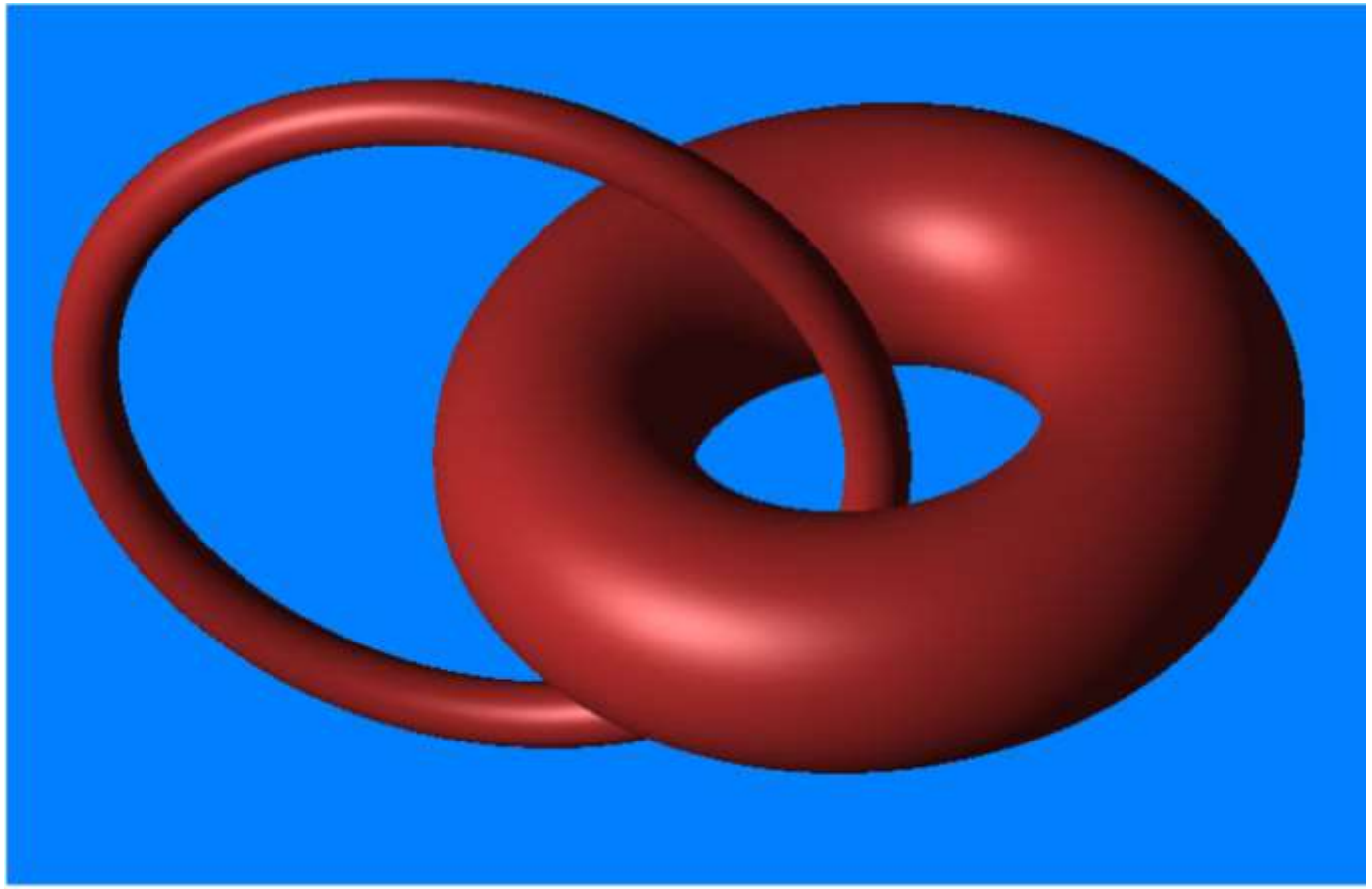
# 光线投射

- 步骤3：

  依据局部光照明模型计算该交点处的光亮度，并将所得光亮度值赋给该像素



# 光线投射

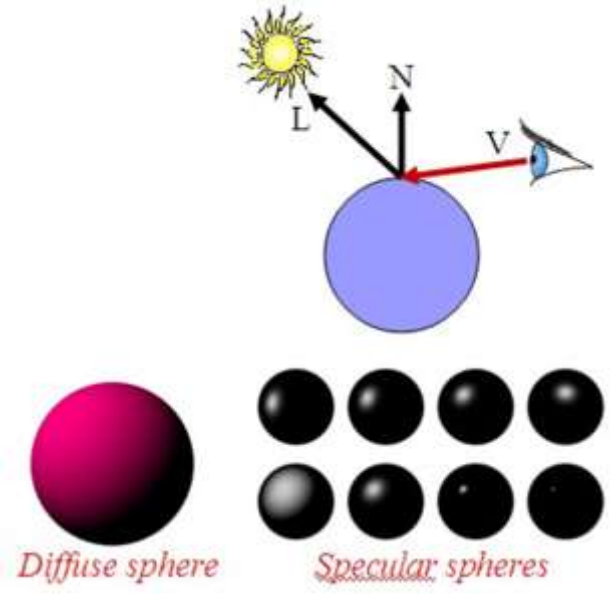


光线投射结果示例

# 光线投射算法

光线投射算法：

- 明暗效果仅仅由第一次相交的物体表面法向方向、材质、视点和光照方向、以及光照强度等因素共同决定。
- 光线投射并不考虑第二层以及更深层次的光线， 因此不具有阴影、反射、折射等效果。



# 光线投射算法

伪代码：

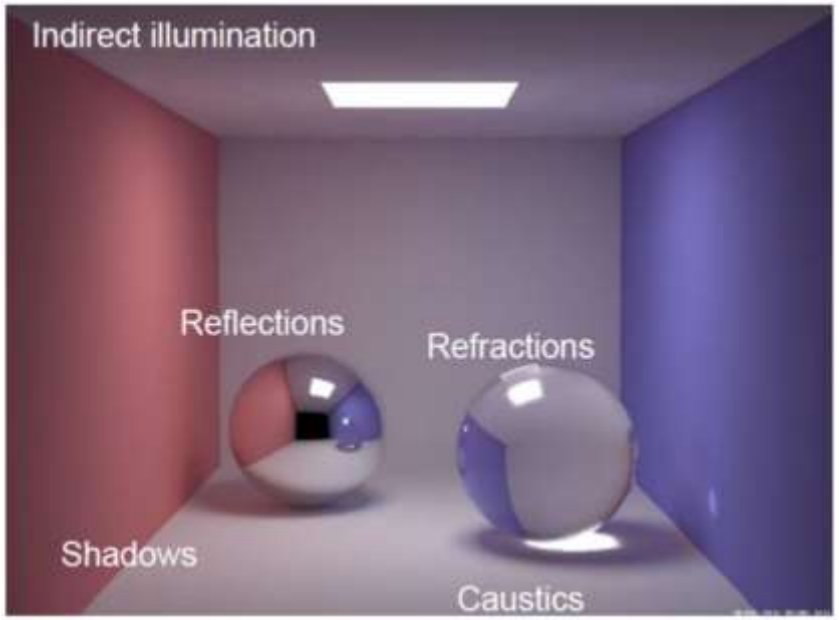
FOR each pixel

- Cast a ray and find the intersection point
- IF there is an intersection
  - color = ambient
  - FOR each light

    color += shading from this light (depending on lightproperty and material property)
  - return color
- ELSE
  - return background color

# 光线跟踪算法概述

- 光线求交 (Ray Intersection)、阴影 (Shadows)、透明和镜面反射 (Trans-parence and Specular Reflection)、纹理 (textures)
- 迄今为止最为成功的生成真实感图形算法之一：算法简单、生成的图形真实感强、计算量大
- 其前身是光线投射（Ray Casting）算法



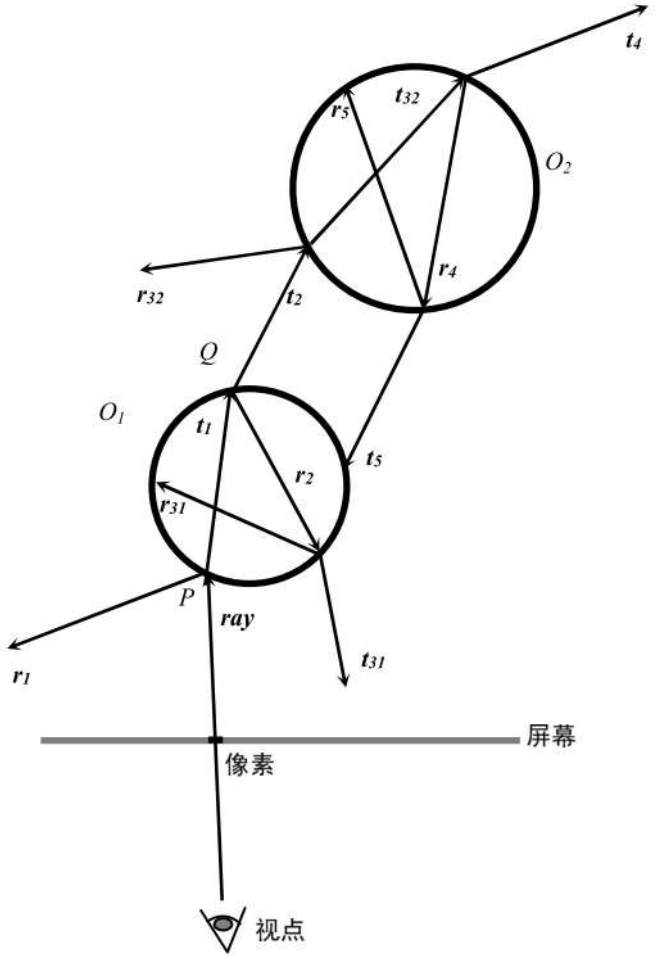
# 光线跟踪算法概述

- 为什么我们能看见物体?

光 (Light) 可以理解为一系列由光源发出、 经物体反复弹射而最终进入视点的光线 (Rays)。最终将光线 (Rays) 射入人们的眼中，使我们看到物体。

- 光线追踪:

逆向跟踪从光源发出的光经由物体之间的多次反射和折射后投射到物体表面，最终进入人眼的过程:



# 添加反射和折射效果

光线跟踪具有模拟物体表面镜面反射效果以及折射效果的能力：

– 首先，计算光线与场景中物体的最近的交点。

– 然后，计算光线在交点处被物体反射和折射所产生的新的光线的方向，新的方向由入射光方向、物体表面法向、及介质共同决定。

– 对新产生的光线 (反射光线和折射光线) 分别继续进行跟踪。

# 光线的反射

反射定律：

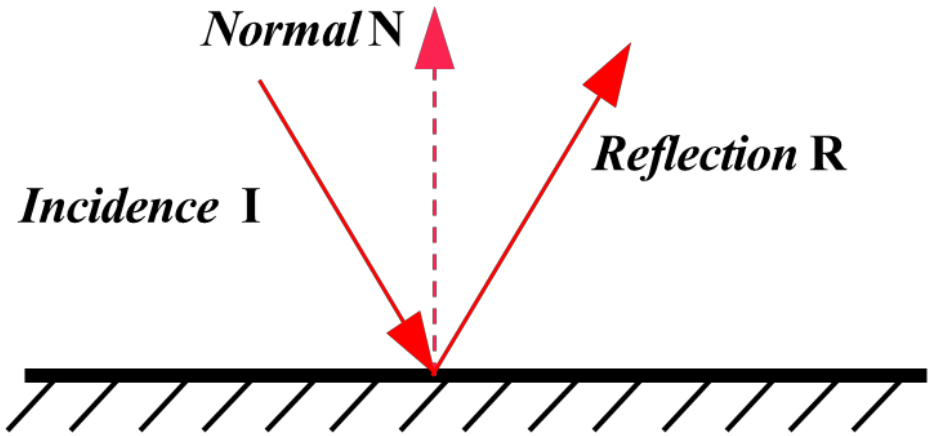
– 入射角 = 反射角

– **入射光线I**、**反射光线R**、以及**物体表面法向量N**位于同一个平面内。

反射光线的方向向量 $R$ 可以按如下公式计算：

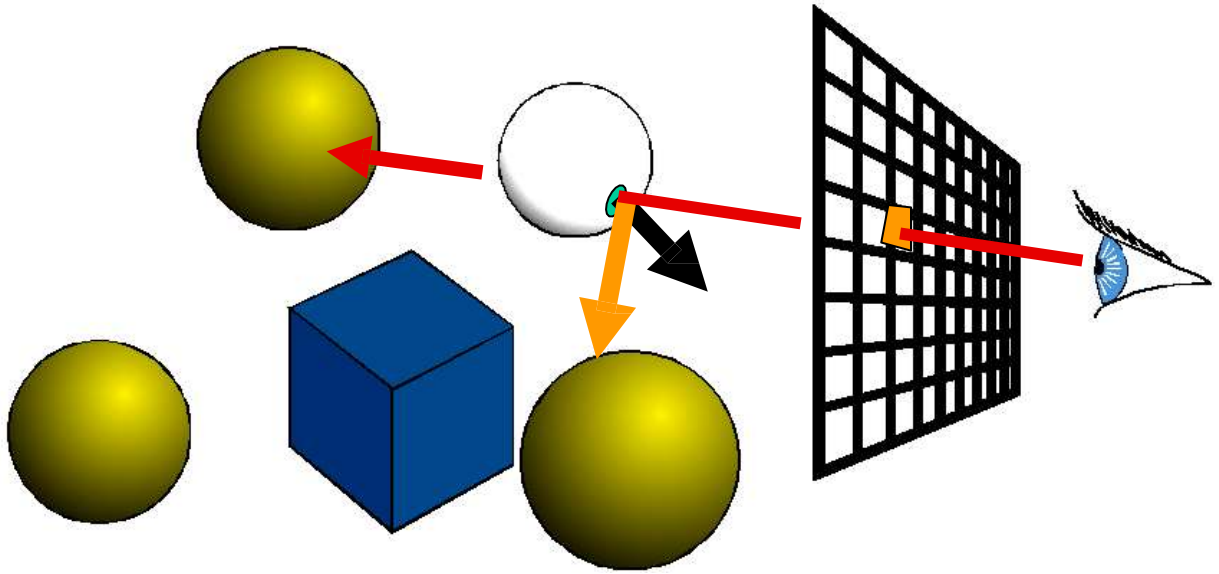
$$\boldsymbol{R} = \boldsymbol{I} - 2(\boldsymbol{I} \cdot \boldsymbol{N})\boldsymbol{N}$$

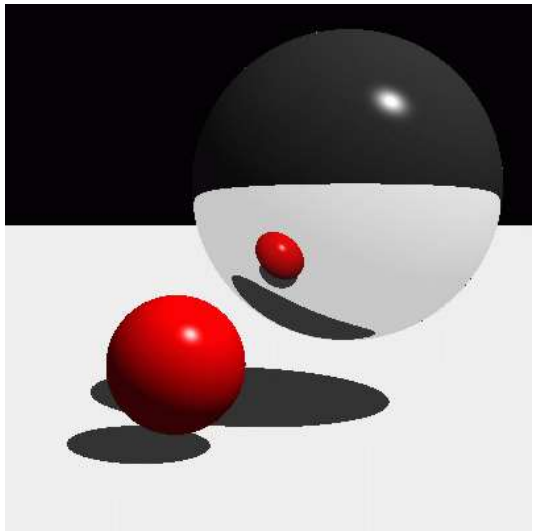
其中 $I, N, R$ 均为单位向量



# 光线的反射

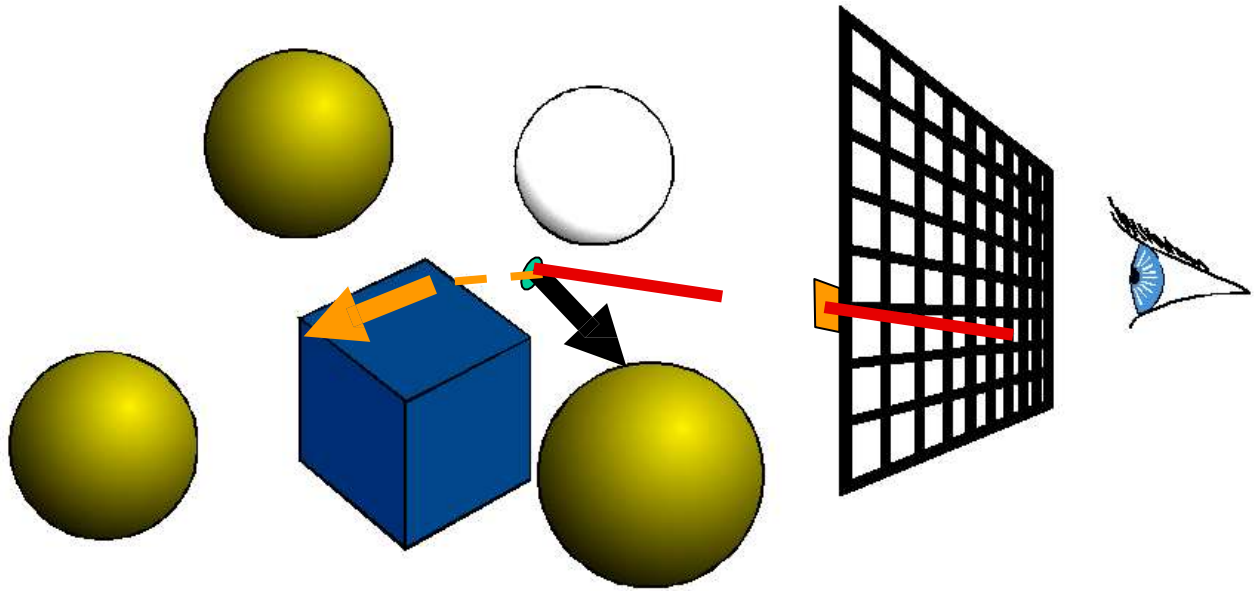
如图所示，反射光线的方向和视点方向，相对于法向方向呈对称关系：





# 光线的折射

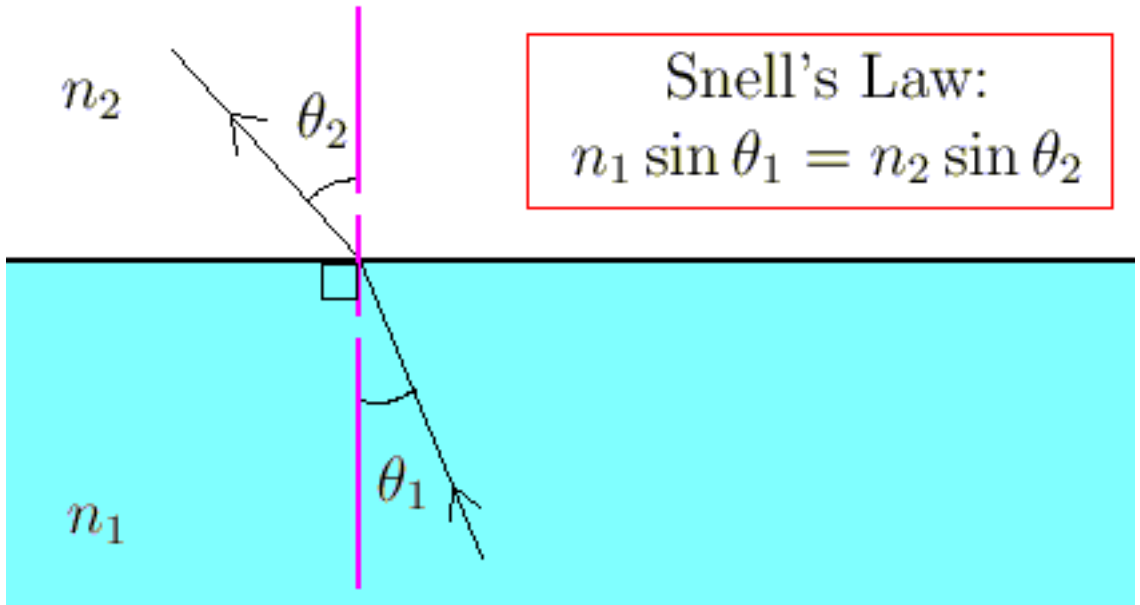
当光线由一种透明或半透明的介质进入另一种时，光的方向会由于介质密度的相对不同而发生偏折，也就是光的折射现象。



# 光线的折射

折射定律 (也叫 Snell 定律)：

入射角和折射角的正弦值之比是一个取决于介质的常数，这个常数被称为相对折射率。



# 光线的折射

求折射方向T

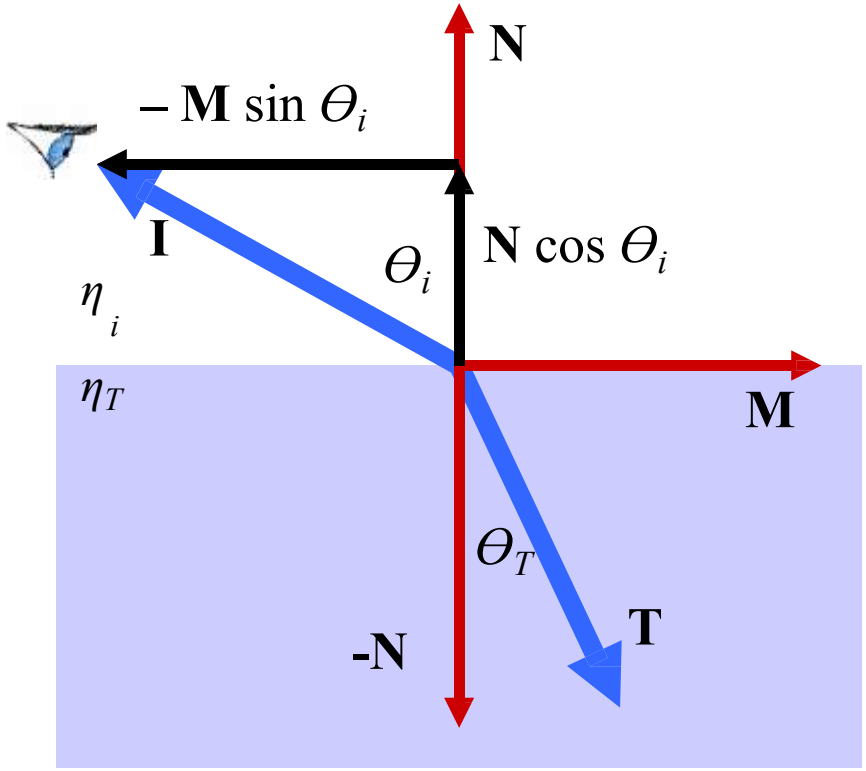
Snell 定律：

$\eta_i \sin\theta_i = \eta_T \sin\theta_T$

推导思路：

将T分解为平行于N的向量-N和垂直于N的向量M，求出他们的线性组合系数，即：

$T = a * (-N) + b * M$



# 光线的折射

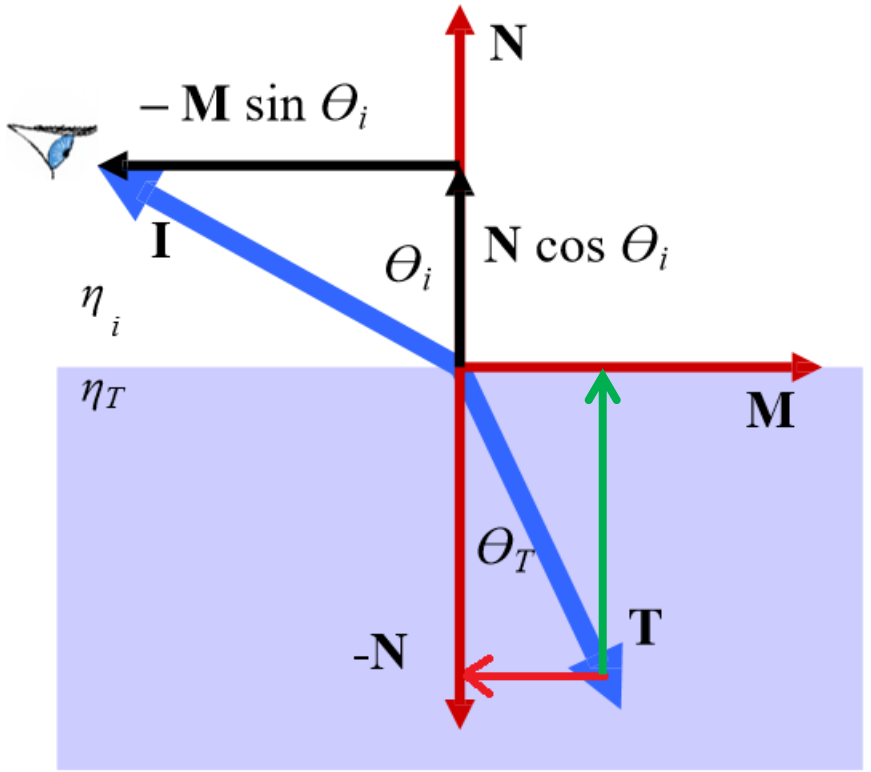
$$T = a * (-N) + b * M$$

**a** 就是向量T在向量-N上的投影长度，由T和-N都是单位向量，有：

$$a = cos\theta_T$$

**b** 就是向量T在向量M上的投影长度，由T和M都是单位向量，有：

$$b = sin\theta_T$$



# 光线的折射

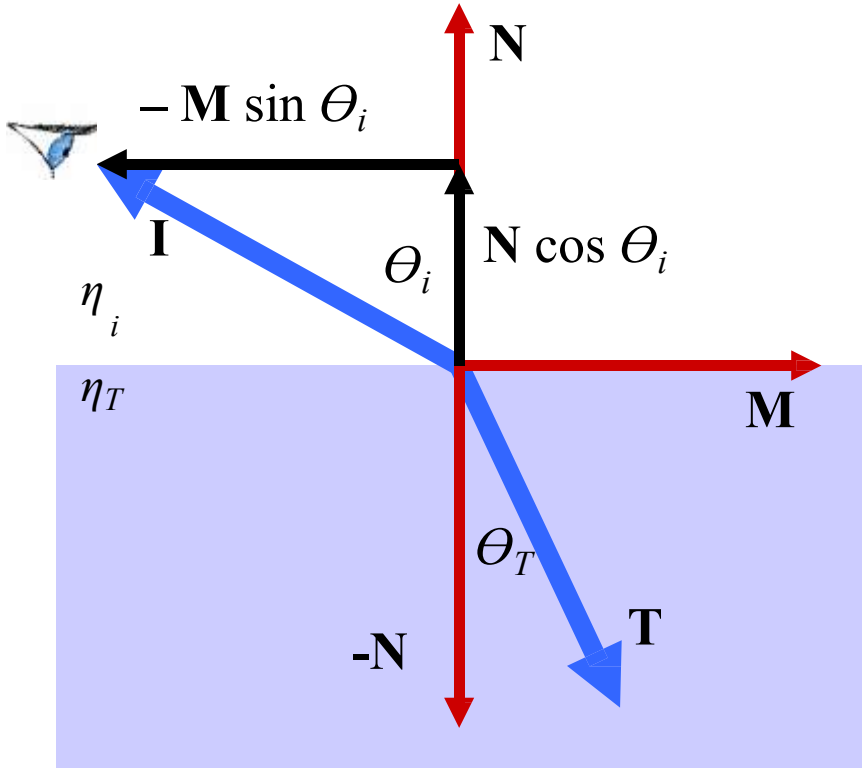
求M

观察右图可以得到一组M的公式：

$$(N \cdot I) * N + (-Msin\theta_i) = I$$

那么就可以得到M的表达式为：

$$M = [(N \cdot I) * N - I]/sin\theta_i$$



# 光线的折射

求T

snell定律，$\eta_i \sin\theta_i = \eta_T \sin\theta_T$

$$T = a * (-N) + b * M$$

$$= cos\theta_T * (-N) + sin\theta_T * M$$

$$= cos\theta_T * (-N) + (sin\theta_T / sin\theta_i) * [(N \cdot I] * N - I]$$

$$= cos\theta_T * (-N) + (\eta_i / \eta_T) * [(N \cdot I] * N - I]$$

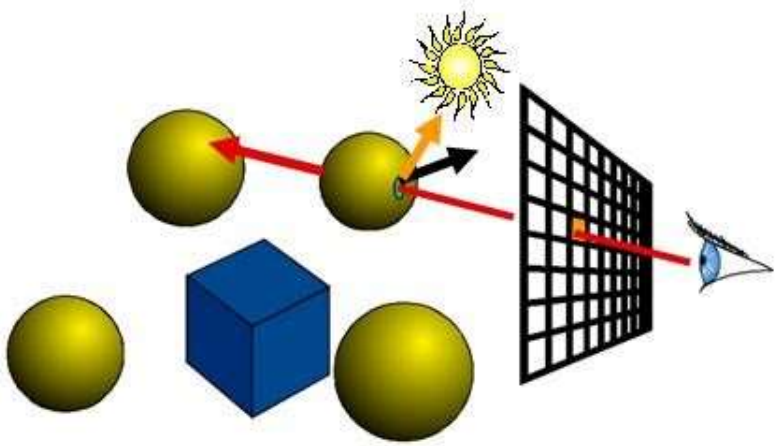
$$= -\frac{\eta_i}{\eta_T} * I + \left(\frac{\eta_i}{\eta_T}(I \cdot N) - \cos\theta_T\right) * N$$

# 光线跟踪算法流程

将显示缓存区看成是由空间中的像素组成的矩形阵列，人眼透过这些像素看到场景中的物体

对于每个像素 P 计算其色彩值：

1. 计算由视点连接像素 P 中心的光线 (Ray) 延长后所碰到的第一个物体的交点。
2. 使用局部光照模型 (如 Phong模型) 计算交点处的颜色值。
3. 沿交点处的反射和折射方向对光线进行跟踪。



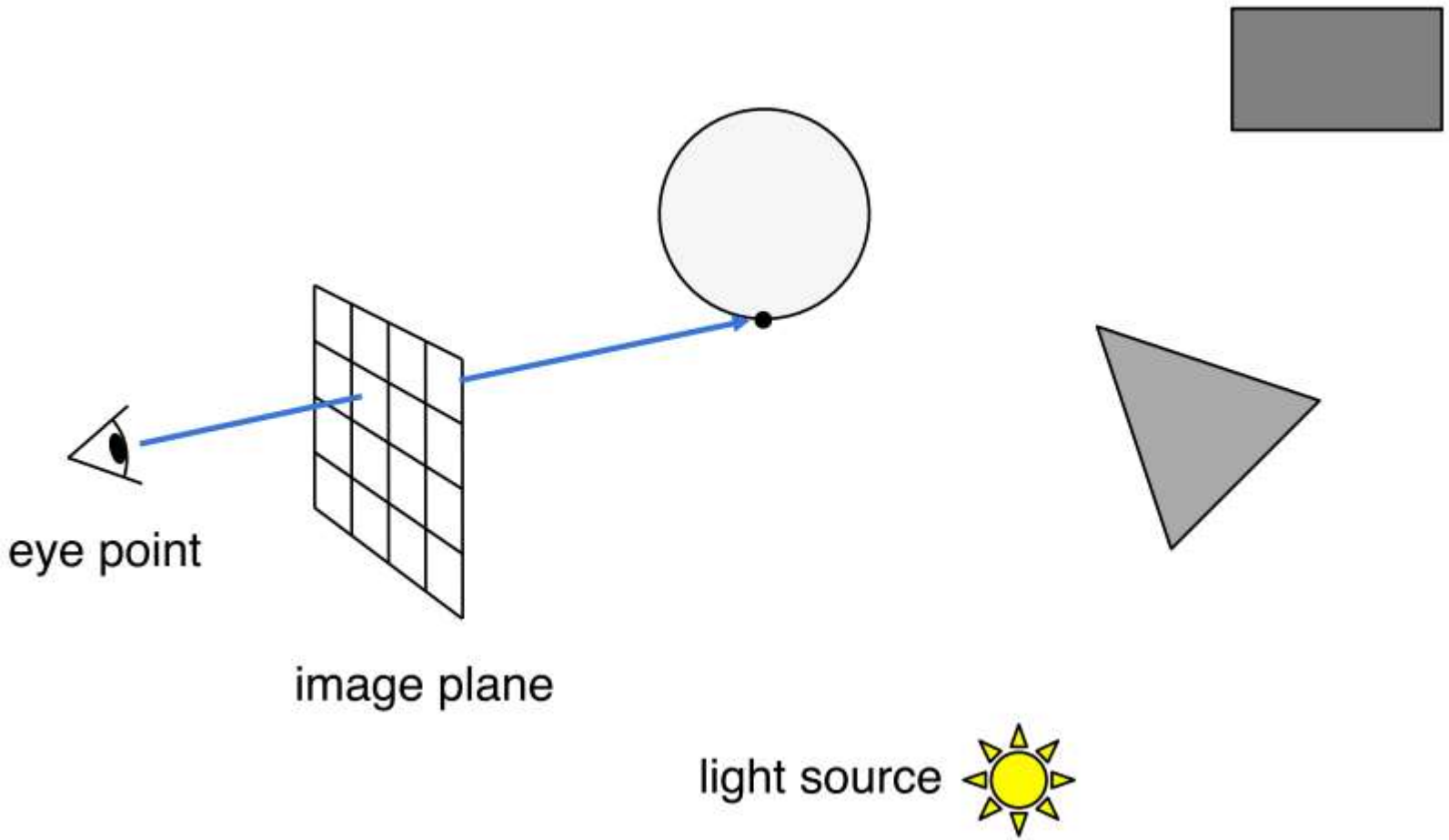
# 光线跟踪原理

- 跟踪从照相机中发出的黄线的话，每条黄线都可以产生出一系列secondary rays：一条反射光线，一条折射光线，并为每个光源产生一条阴影线。

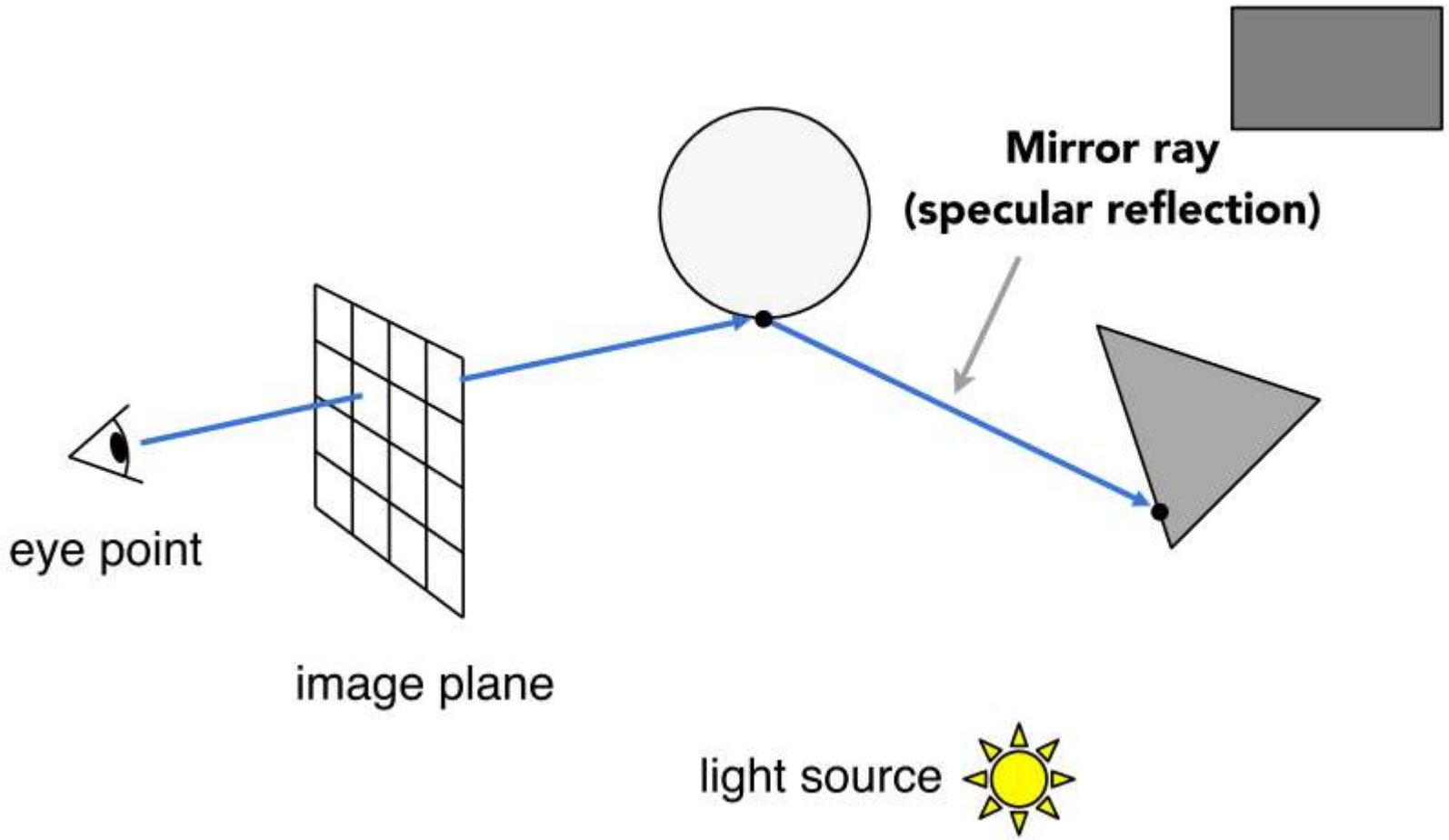
- 这些光线产生后（除了阴影线外）都可以被视为普通的光线。

- 这意味着一条反射光线可以再被反射和折射，这种方法叫做“递归光线跟踪”。每条新产生的光线都增加了它先前光线聚集的地方的颜色，最终每条光线都对最开始由primary ray穿过的像素点的颜色做出了自己的贡献。

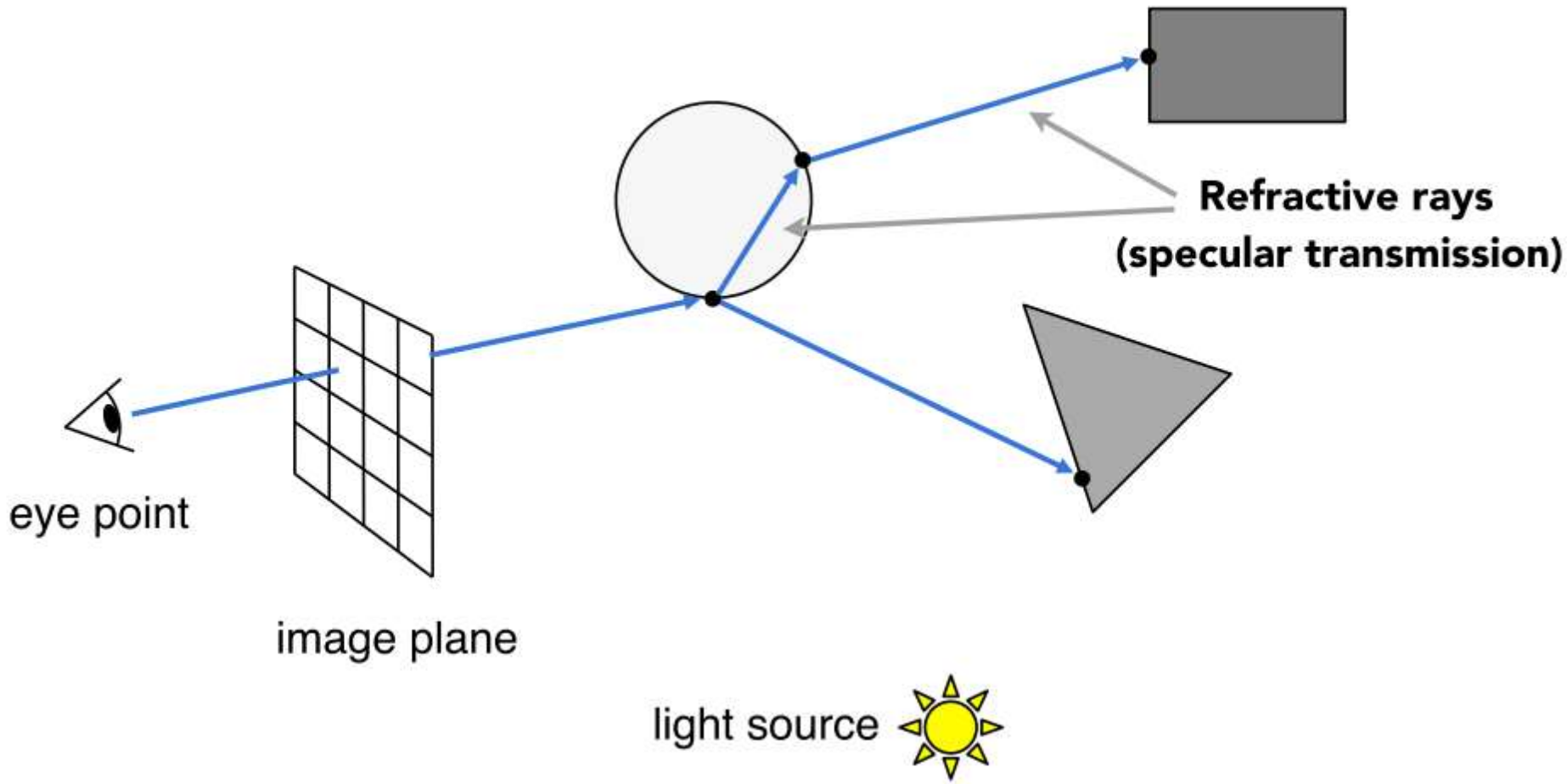
# 光线跟踪原理



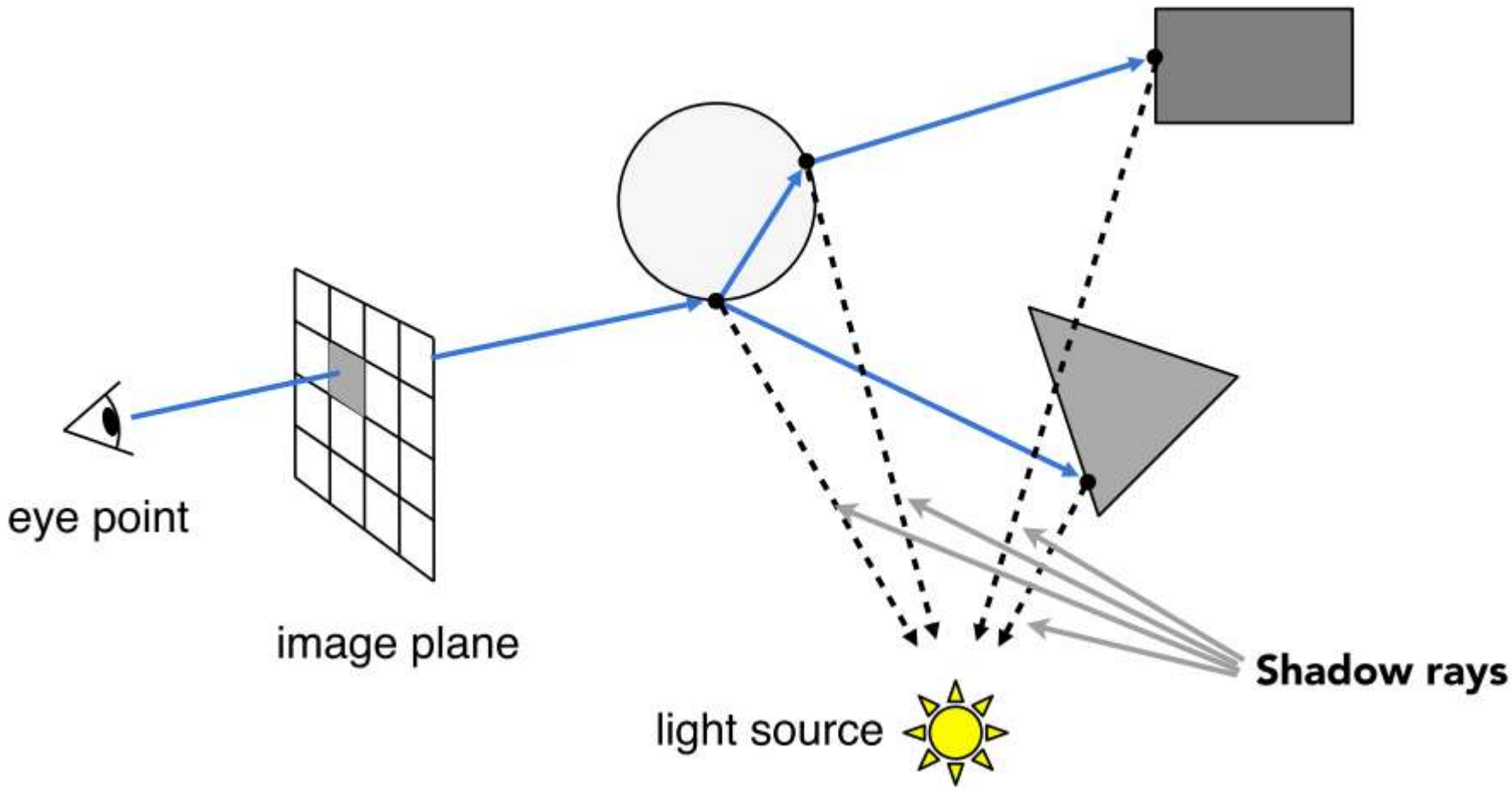
# 光线跟踪原理


Mirror ray
(specular reflection)
eye point
image plane
light source

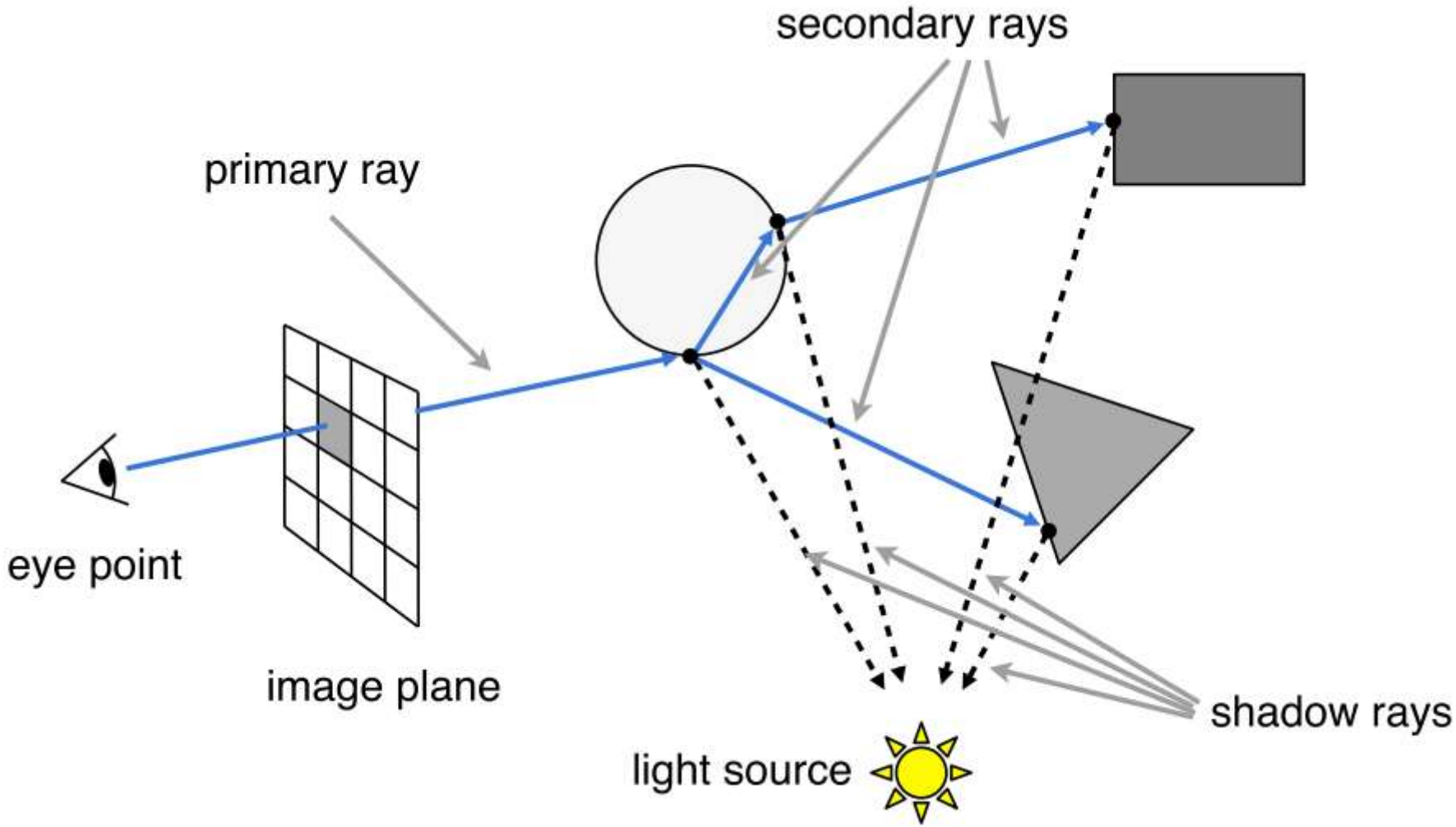
# 光线跟踪原理


Refractive rays
(specular transmission)
eye point
image plane
light source

# 光线跟踪原理

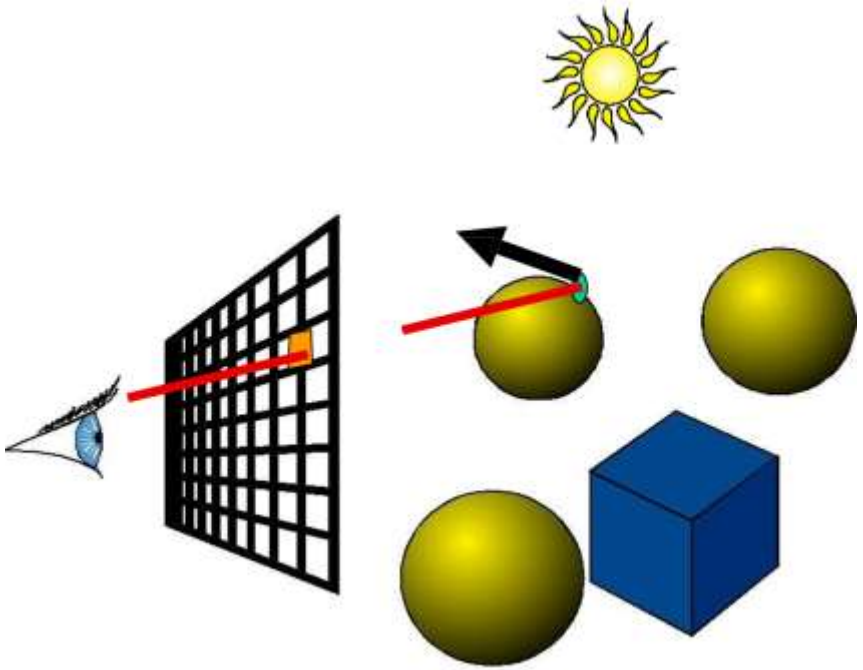


# 光线跟踪原理


secondary rays
primary ray
eye point
image plane
shadow rays
light source

# 光线跟踪算法流程

```
对图像中的每一个像素{
        创建从视点通过该像素的光线
        初始化   最近T 为无穷大，最近物体为空
        对场景中的每一个物体{
                如果光线与物体相交{
                        如果交点处的 t 比 最近T 小{
                                设置 最近T 为交点的 t值
                                设置 最近物体 为该物体
                        }
                }          //使用T，t来储存物体交点和视点的距离
        }                  //以上为对最近物体交点的计算
```



# 光线跟踪算法流程

如果 最近物体为空值{

用背景色填充该像素

}

否则{

对每个光源射出一条光线来检测是否处在阴影中

如果物体表面是反射面，生成反射光；递归

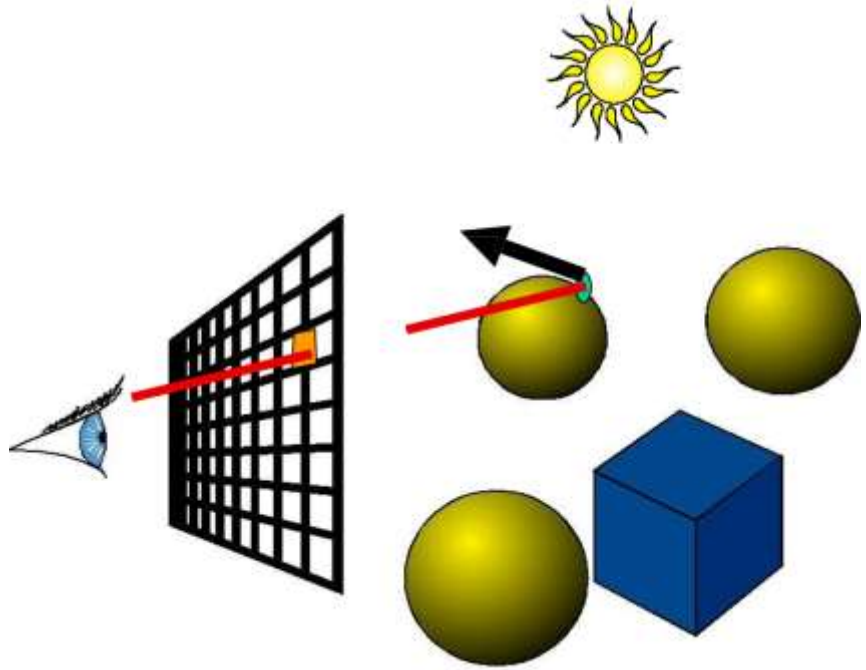
如果物体透明，生成折射光；递归

使用 最近物体 和 最近T 来计算着色函数

以着色函数的结果填充该像素

}

}



# 经典的光线跟踪算法

```
IntersectColor( vBeginPoint, vDirection) //输入参数，射线的起点，方向
{
        Determine IntersectPoint;
        Color = ambient color;  //赋初值，环境光
        for each light
                Color += local shading term;
        if( surface is reflective )  //如果物体可反射
                Color += reflect Coefficient * IntersectColor( IntersecPoint, Reflect Ray);
        if( surface is refractive ) // 如果物体可折射射
                Color += refract Coefficient * IntersectColor( IntersecPoint, Reflect Ray);
        return Color;
}
```

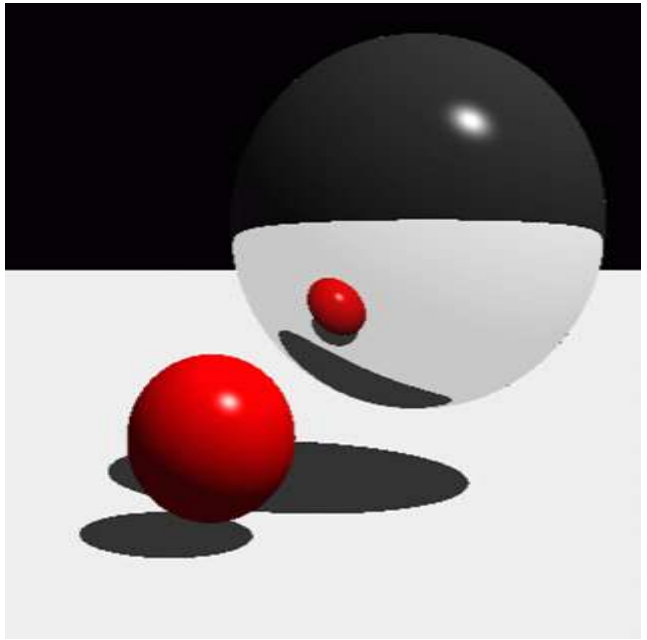
# 光线跟踪

- 光线跟踪递归过程中止条件

1. 光线与环境中任何物体均不相交，或交于纯漫射面
2. 被跟踪光线返回的光亮度值对像素颜色的贡献很小
3. 已递归到给定深度

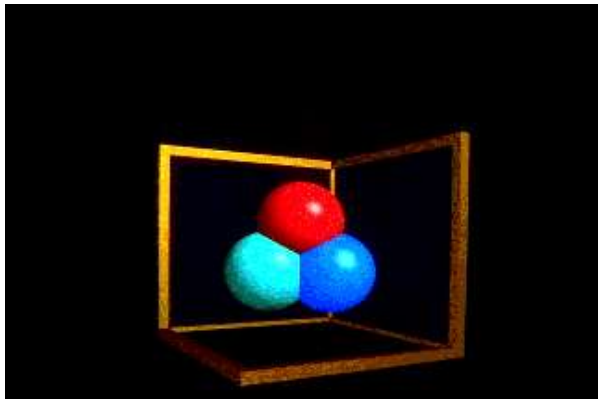
# 光线跟踪特征

- 通过光线跟踪，可以很容易地表现出例如阴影、反射、折射等引人入胜的视觉效果。
- 除了基本的几何形体，（例如球体、椎体、立方体等），光线跟踪容易适用于更复杂的物体表示方法，(例如多边形网格表示或者复合形体等)。

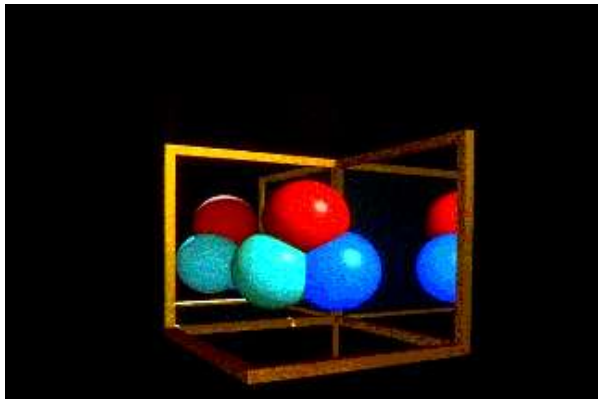


# 光线跟踪特征

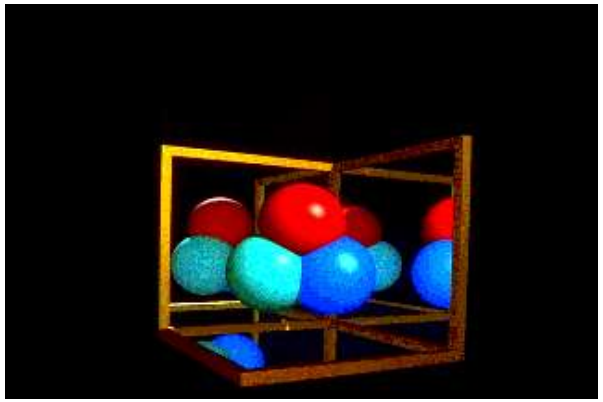
- 光线跟踪效果示例：



0 层递归



1 层递归



2 层递归

# 光线求交

- 光线的表示
- 光线与平面 (Plane) 求交
- 光线与三角形 (Triangle) 求交
- 光线与球面 (Sphere) 求交
- 光线与多边形（Polygon）求交

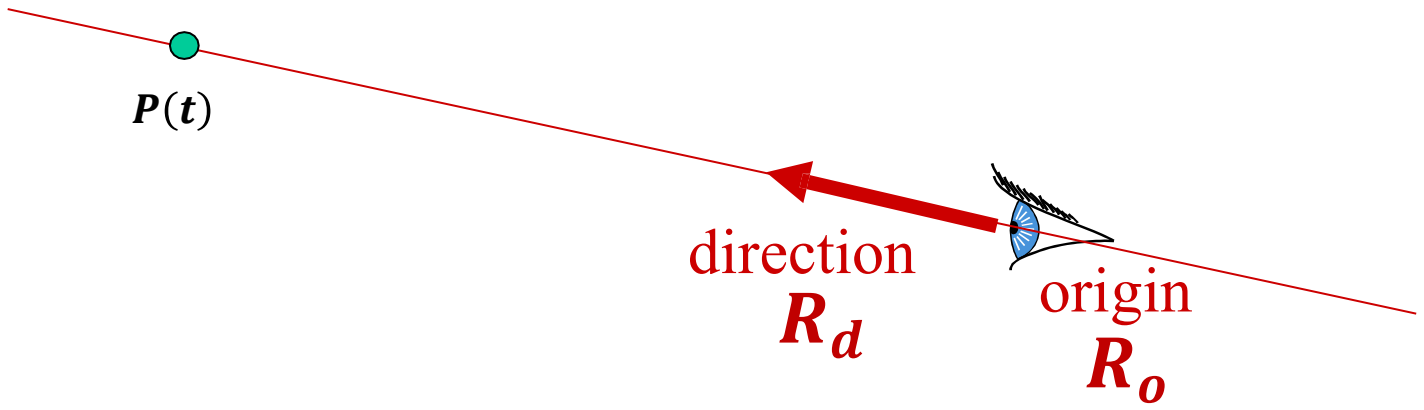
# 光线

光线 (射线) 的参数表示

$P(t) = R_o + t * R_d$

$R_o = (x_o, y_o, z_o)$是光线的源点；向量 $R_d = (x_d, y_d, z_d)$ 代表光线的朝向，通常来说 $R_d$ 是单位向量

参数$t$表示光线到达的位置：在光线的正方向上，参数$t$都是正数， $t > 0$



# 光线与平面求交

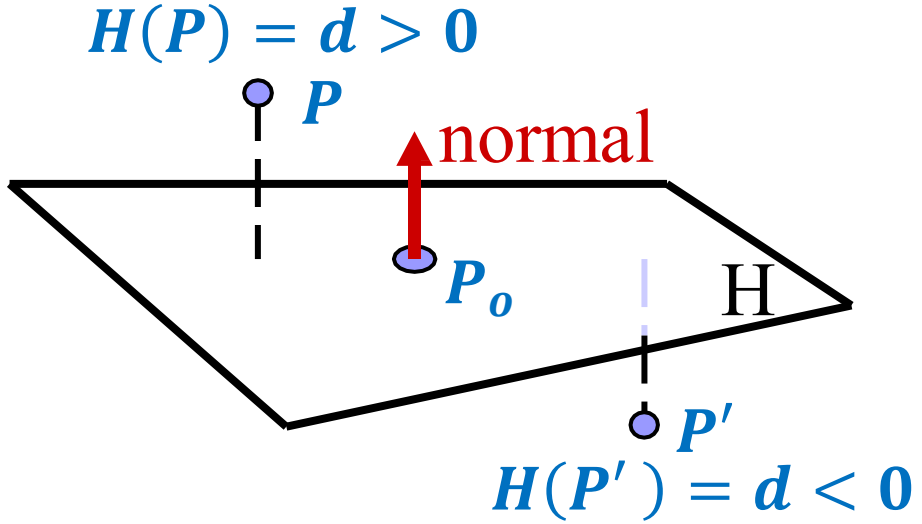
平面的表示：

显示表示：$P_o = (x_o, y_o, z_o),\ n = (A, B, C)$

隐示表示：$H(P) = Ax + By + Cz + D = 0$　　$H(P) = n \cdot P + D = 0$

点到平面的距离：

当 $\boldsymbol{n}$ 是单位法向量时，$\boldsymbol{P}$ 到平面 $\boldsymbol{H}$ 的距离就是 $\boldsymbol{H(P)}$ 。



# 光线与平面求交

隐示表示: $H(P) = Ax + By + Cz + D = 0$

$H(P) = n \cdot P + D = 0$

- 给定满足如下的平面方程:

$$n \cdot P + D = 0$$

如何计算一条光线与该平面的交点?

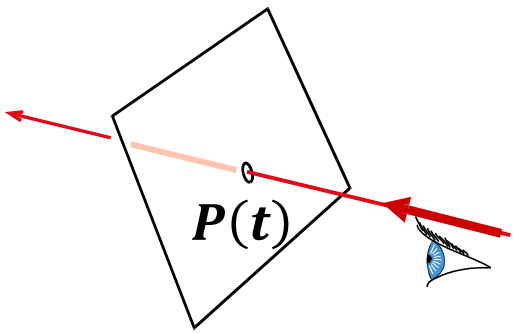
- 连列方程如下:

$$P(t) = R_o + t * R_d$$

$$n \cdot P(t) + D = 0$$

解得: $t = -(D + n \cdot Ro) / (n \cdot Rd)$

最后验算 $t > 0$



# 光线与三角形求交

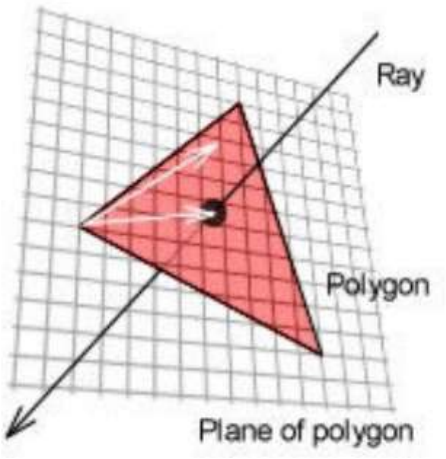
- 在实时的图形学中，基于三角形的几何表示 (三角面片) 十分常见：

  每个三角形可以使用三个顶点的坐标来表示。

- 常用步骤：

  1. 判断光线是否与三角形所在平面求交

  2.若交点存在，判别交点是否位于三角形内部



# 光线与三角形求交

- 重心坐标：

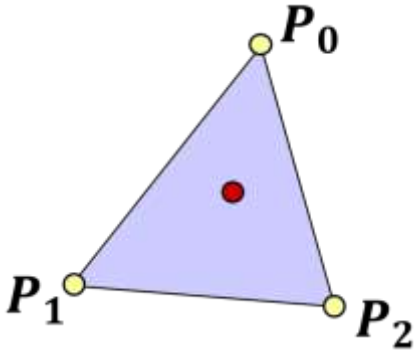
  三角形 $P_0P_1P_2$ 内的一点 $P$ 可以表示成：

  $$p = \alpha p_0 + \beta p_1 + \gamma p_2$$

  其中 $(\alpha, \beta, \gamma)$ 被称为重心坐标，它们满足：

  $$0 \le \alpha, \beta, \gamma \le 1, \alpha + \beta + \gamma = 1$$

  重心坐标有许多其他应用，例如纹理映射、法向插值、颜色插值，等等。



# 光线与三角形求交

我们将$\alpha + \beta + \gamma = 1$ 写成 $\alpha = 1 - \beta - \gamma$，有：

$$p = (1 - \beta - \gamma)p_0 + \beta p_1 + \gamma p_2$$

可以将光线与三角形求交描述成如下方程：

$$R_0 + tR_d = (1 - \beta - \gamma)p_0 + \beta p_1 + \gamma p_2$$

即

$$(R_d \;\; p_0 - p_1 \;\; p_0 - p_2)\begin{pmatrix} t \\ \beta \\ \gamma \end{pmatrix} = p_0 - R_0$$

这说明交点的重心坐标以及其在直线上的 $t$ 参数可以通过解一个线性方程组而得到。

# 光线与三角形求交

令：$E_1 = P_0 - P_1, E_2 = P_0 - P_2, S = P_0 - R_o$

根据 Cramer 法则，我们可以写出：

$$\begin{bmatrix} t \\ \beta \\ \gamma \end{bmatrix} = \frac{1}{det(R_d, E_1, E_2)} \begin{bmatrix} det(S, E_1, E_2) \\ det(R_d, S, E_2) \\ det(R_d, E_1, S) \end{bmatrix}$$

最后需要检查 $t > 0$ 且 $0 \leq \beta, \gamma \leq 1, \beta + \gamma \leq 1$ 以保证交点位于三角形的内部。

# 光线与球面求交

**数学求解：**

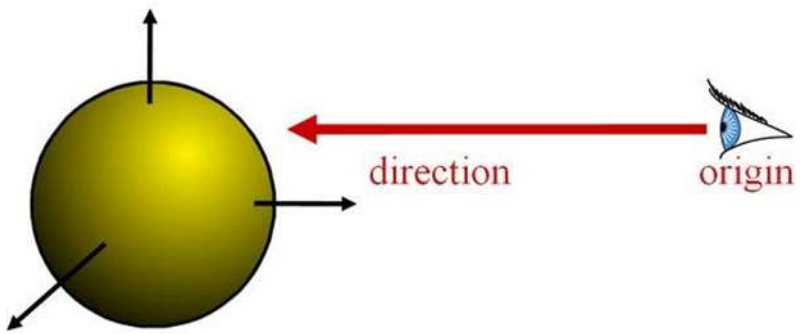
首先给出球面的数学描述：

给定中心点 (球心)$p_c$，半径 $r$，可以写出球面的隐式方程

$$f(P) = \|P - P_c\| - r = 0$$

求光线与球面的交点，相当于要求解如下的方程：

$$\|P - P_c\| - r = 0$$



# 光线与球面求交

将之前的方程进行化简：

$$\|P(t) - P_c\| - r = 0$$

$$\|R_o + tR_d - P_c\| = r$$

$$(R_o + tR_d - P_c) \cdot (R_o + tR_d - P_c) = r^2$$

$$t^2(R_d \cdot R_d) + 2t\big(R_d \cdot (R_o - P_c)\big) + (R_o - P_c) \cdot (R_o - P_c) - r^2 = 0$$

由于 $R_d$是单位向量：

$$t^2 + 2t\big(R_d \cdot (R_o - P_c)\big) + (R_o - P_c) \cdot (R_o - P_c) - r^2 = 0$$

记为： $t^2 + 2tb + c = 0$

解得： $t = -b \pm \sqrt{b^2 - c}$

# 光线与球面求交

之前的代数方法存在可能改进和加速的地方，例如，注意到我们实际上并不需要计算被挡住的第二个交点。

下面的几何方法则具有如下优点：

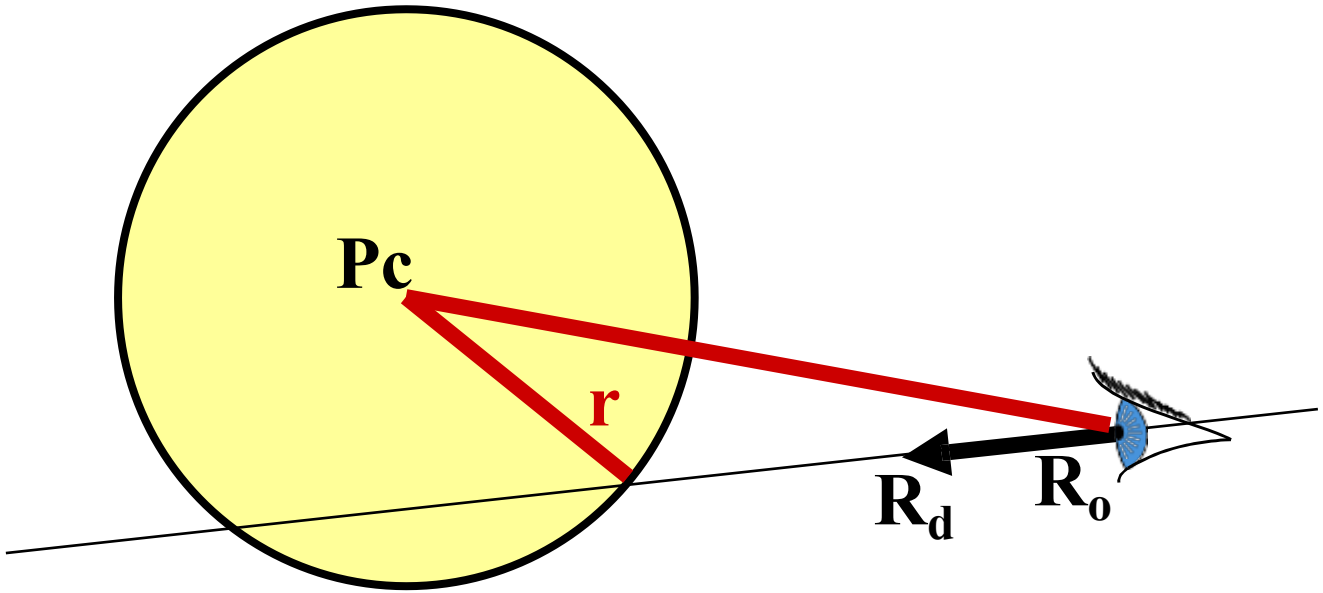
- 能快速判断光线与球面是否相交。
- 能快速判断光源是否在球体内部。
- 能快速判断光线的方向是朝向是远离球面。

# 光线与球面求交——几何法

几何方法：

计算光线起点到球心中心的向量$\vec{l}$

$$l = P_c - R_0$$



# 光线与球面求交——几何法

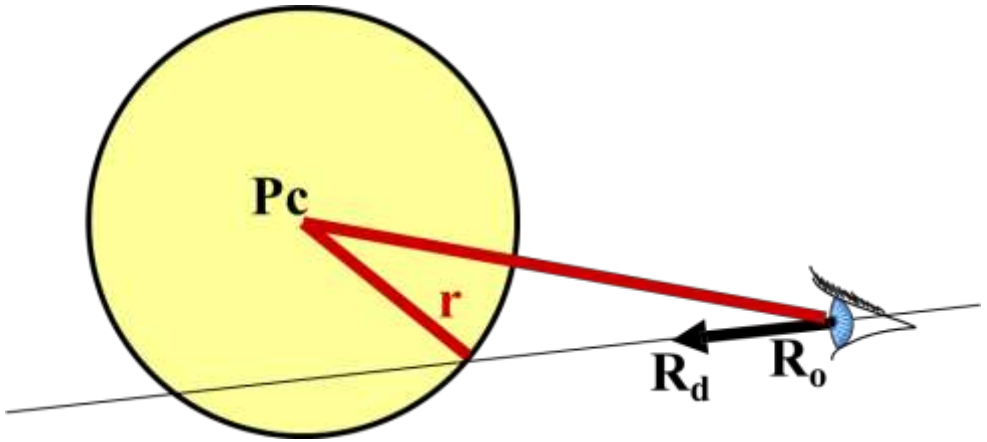
几何方法：

判断光线起点是否在球的内部

位于球的内部：$l^2 < r^2$

位于球的外部：$l^2 > r^2$

位于球上：$l^2 = r^2$



# 光线与球面求交——几何法

几何方法：

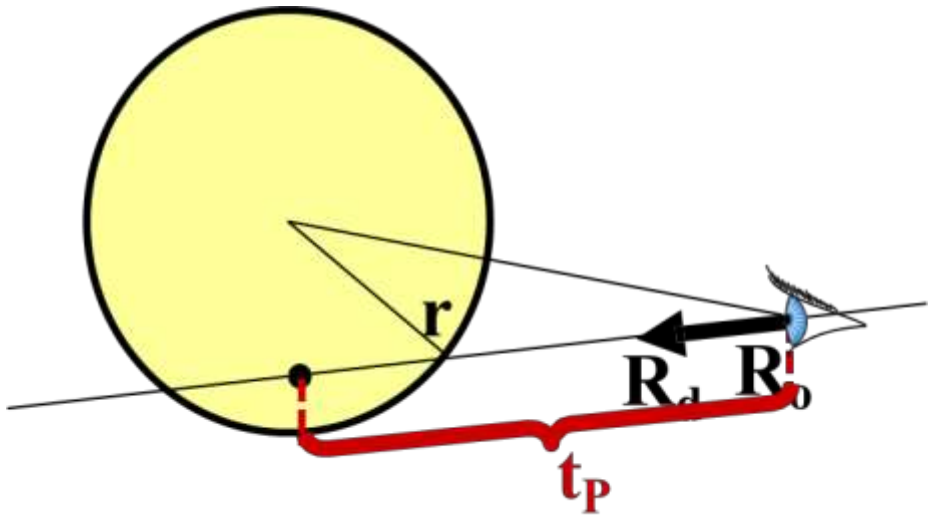
找到距离球心最近的点：

$$t_p = l \cdot R_d$$

If 起点位于球外 & $t_p < 0$ → 不相交

球位于光线起点的后方

光线与球不相交



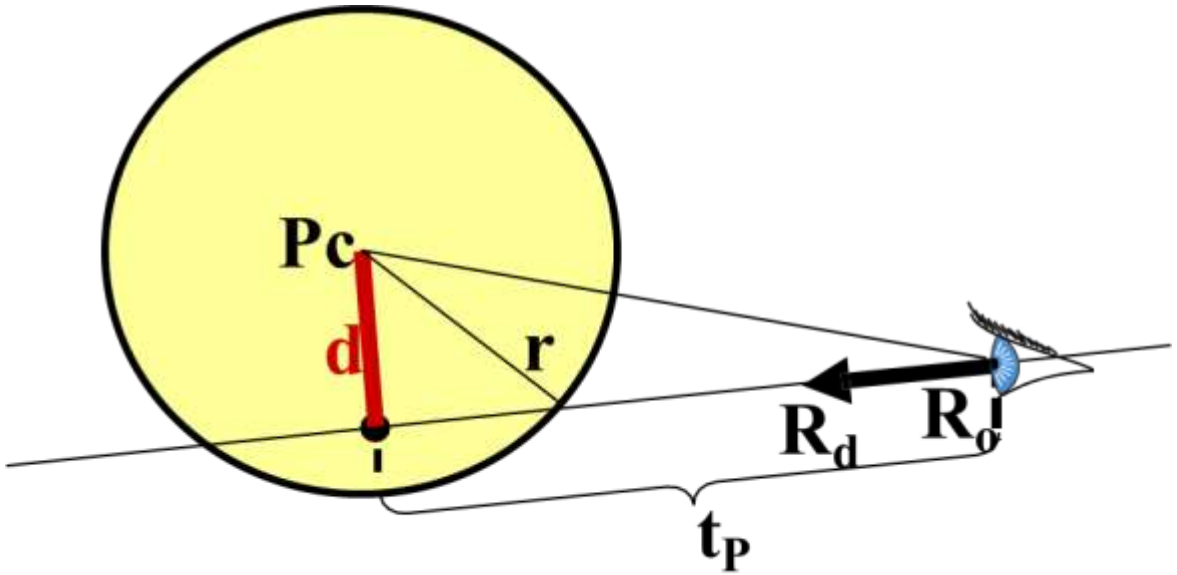
# 光线与球面求交——几何法

几何方法：

找到距离球心最近的点与球心的距离 d的平方：

$$d^2 = l^2 - t^2$$

If d > r → 不相交



# 光线与球面求交——几何法

几何方法：

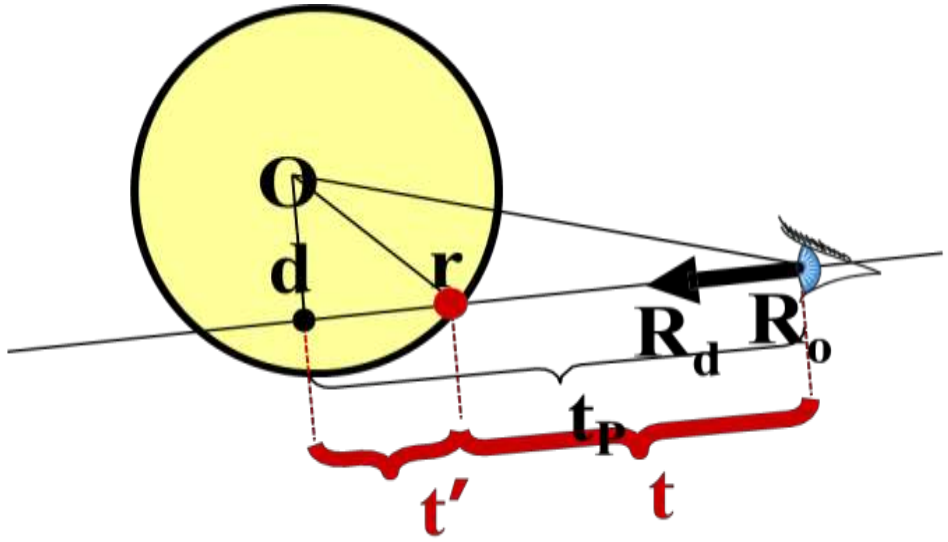
求最近点与相交点的距离：

$$t'^2 = r^2 - d^2$$

解为：

If 光线起点位于球外部 $\rightarrow$ $t = t_p - t'$

If 光线起点位于球内部 $\rightarrow$ $t = t_p + t'$



# 光线与多边形求交

尽管三角形是最为常用的基本渲染单元，但是光线与多边形的求交也是经常需要用到的。

一个$n$多边形可以表示成$n$个顶点的有序列表：$\{v_0, v_1, \cdots, v_{n-1}\}$，且该多边形的 $n$ 条边分别为$v_0v_1, v_1v_2, \cdots, v_{n-2}v_{n-1}, v_{n-1}v_0$。其中 $v_0, v_1, \cdots, v_{n-1}$ 位于同一平面，该平面可以表示为：

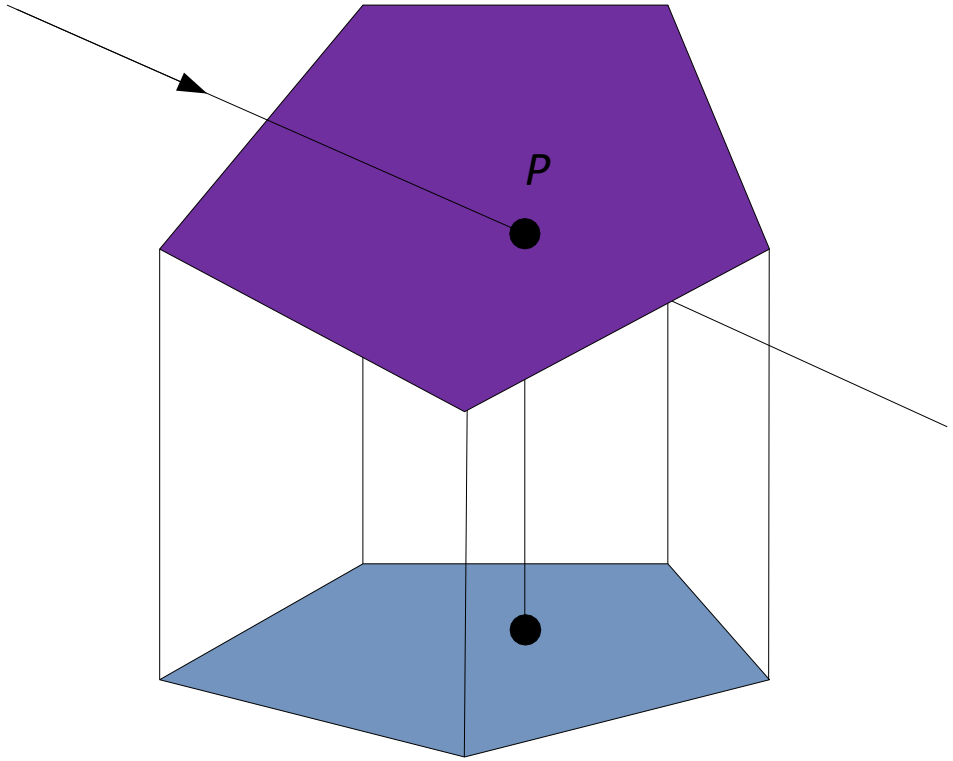
$$\pi_p\colon n_p \cdot x + d_p = 0$$

# 光线与多边形求交

我们首先计算光线与多边形所在平面的交点。

如果交点存在，我们需要判断这个交点是否位于多边形的内部。

为了进行这一判断，我们将交点以及多边形的所有顶点投影到 XY-, YZ-, ZX-平面中一个 (如图所示)，以方便后续处理。



# 光线与多边形求交

在二维情况下，判断一个点是否在一个多边形的内部，可以使用交点检测算法 (Crossing Test)。

交点检测算法基于如下 Jordan 曲线定理： 平面上一个点位于一个多边形的内部， 当且仅当由该点出发的任何一条射线都与多边形的边界有奇数个交点。交点检测算法也叫做奇偶检测算法。

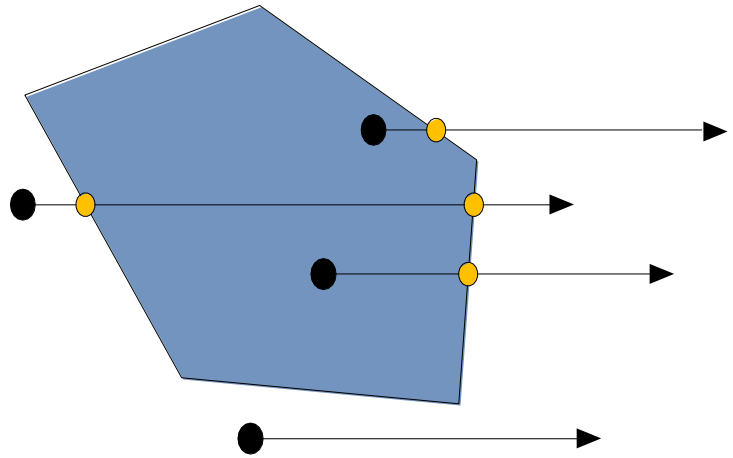
# 光线与多边形求交

交点检测算法如图所示：

多边形内的两个黑点分别与多边形边界相交1次。

多边形外的两个黑点分别与多边形边界相交0次和2次。

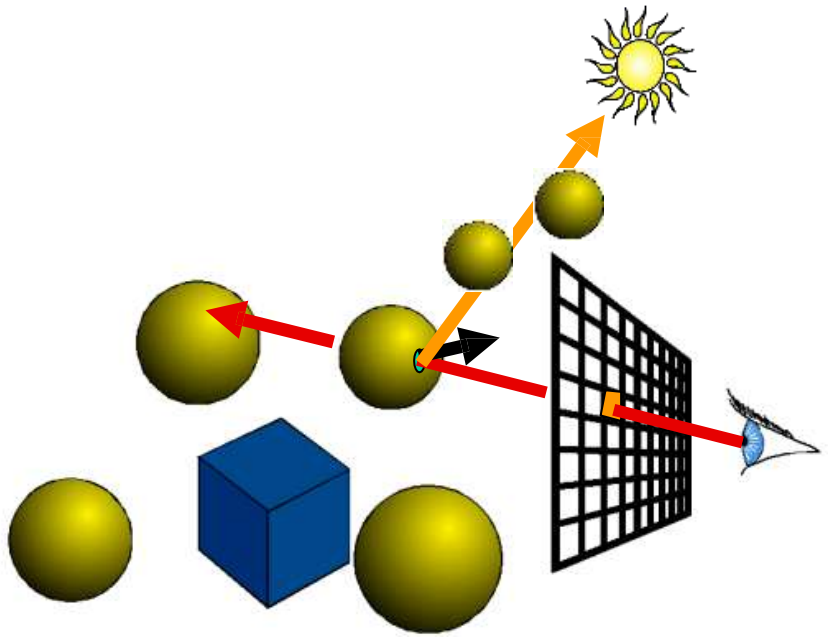
交点检测算法是所有不进行预处理的方法中最快的。



# 添加阴影 (Shadows)

如图所示，由交点向光源方向投射一道光线，如果这道光线与场景中物体有交点，则该交点属于阴影区域。

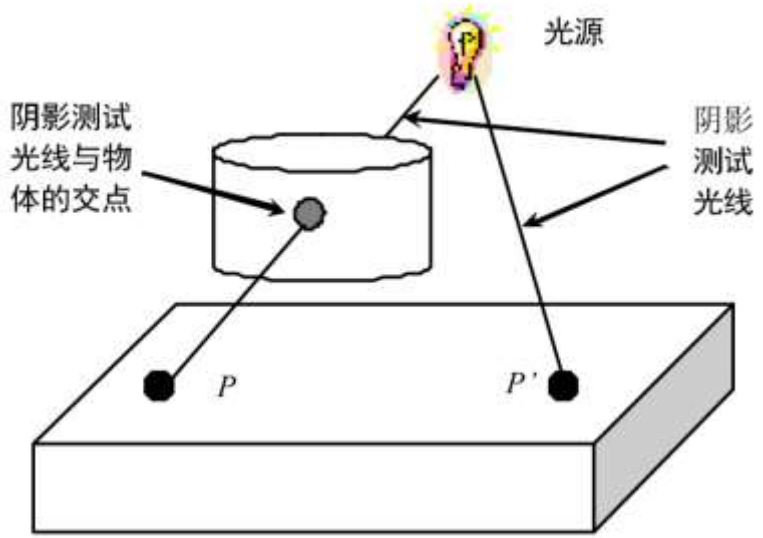
计算阴影区域时，我们只关心是否与物体相交，而不关注哪个是最近交点。



# 阴影计算 (Shadows)

从P出发向光源L发射一条阴影测试光线R:

- 若R在到达L的途中与场景中的物体不相交，则点P受光源L直接照射
- 反之，点P被位于它与光源L之间某一物体所遮挡
- 若遮挡物为不透明体，则点P位 于光源阴影之中



# 阴影计算 (Shadows)

包含阴影计算的Phong模型:

$$I(P) = K_a I_a + \sum_{i=1}^{m} (f_i(P) I_i (k_{d\,i}(\boldsymbol{N}, \boldsymbol{L}) + k_s(\boldsymbol{N}, \boldsymbol{H})^n))$$

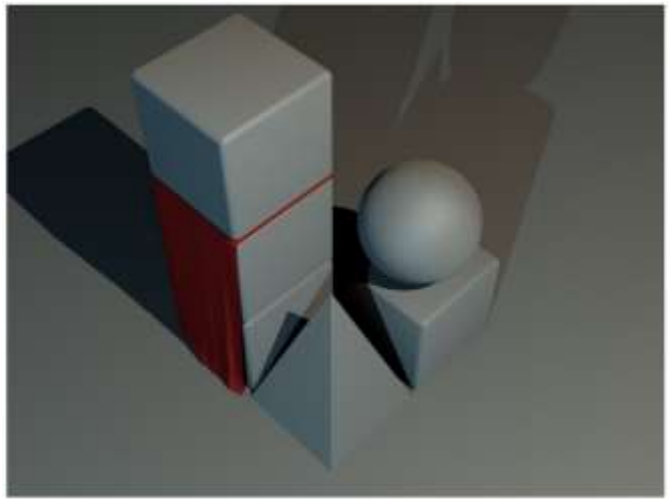
$$f_i(P) = \begin{cases} 0 & \text{若}P\text{未受光源}i\text{直接照射} \\ 1 & \text{若}P\text{受光源}i\text{直接照射} \end{cases}$$

# 阴影计算 (Shadows)

包含阴影计算的Phong模型：

$$I(P) = K_a I_a + \sum_{i=1}^{m} (f_i(P) I_i (k_{di}(\boldsymbol{N}, \boldsymbol{L}) + k_s(\boldsymbol{N}, \boldsymbol{H})^n))$$

$$f_i(P) = \begin{cases} 0 & \text{若}P\text{未受光源}i\text{直接照射} \\ 1 & \text{若}P\text{受光源}i\text{直接照射} \end{cases}$$



光线跟踪阴影效果

# 走样

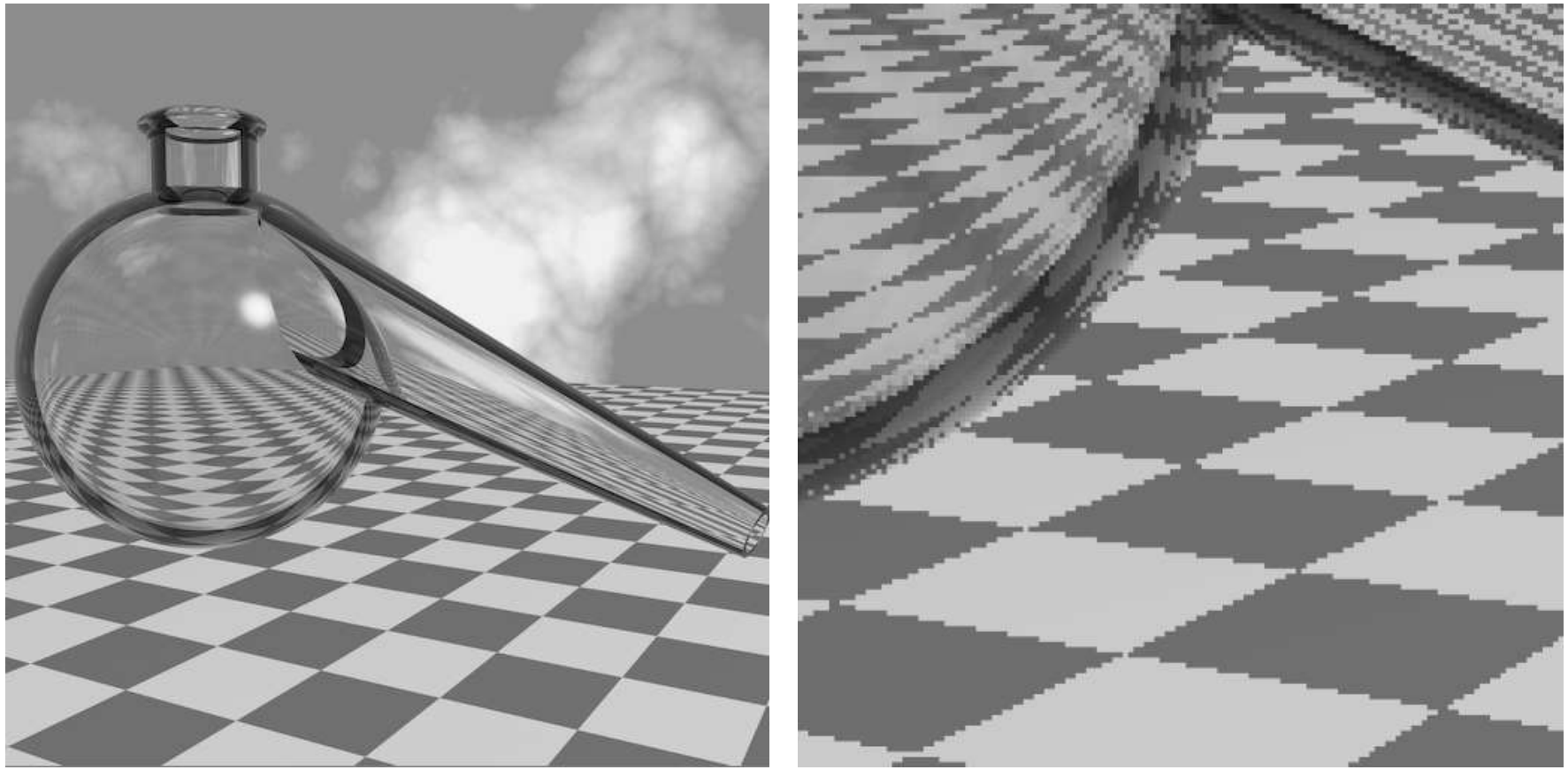
走样：用离散的量（像素）表示连续的量（图形）而引起的失真

- 数学上的点、直线是无宽度、面积的
- 像素是有面积的

光栅图形的走样现象：

- 阶梯状边界
- 图形细节失真
- 狭小图形丢失
- 动画序列中时隐时现，产生闪烁

# 走样



走样现象

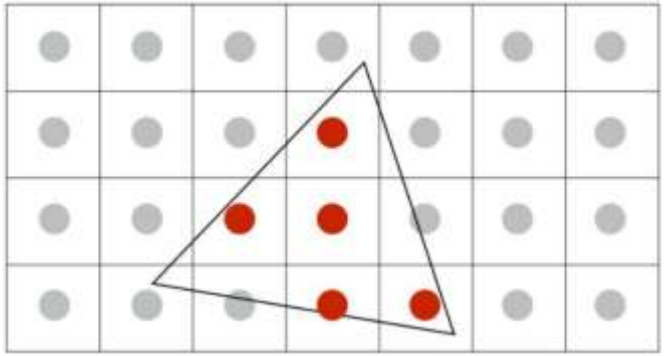
# 反走样

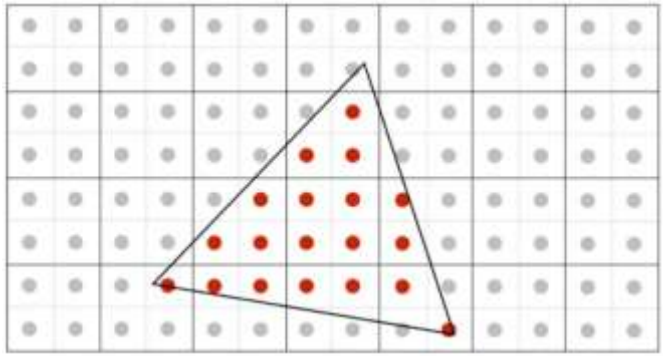
超采样（超级采样抗锯齿）

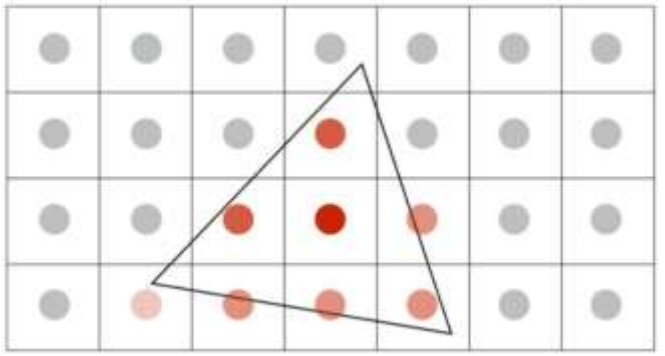
- 把当前分辨率成倍提高，然后再把画缩放到当前的显示器上
- 有限离散像素点逼近结果不好，用更多的采样点去逼近得到更好的效果

自适应超采样：

- 只在当像素在物体边沿时进行超级采样
- 减少缓存、储存带宽的消耗



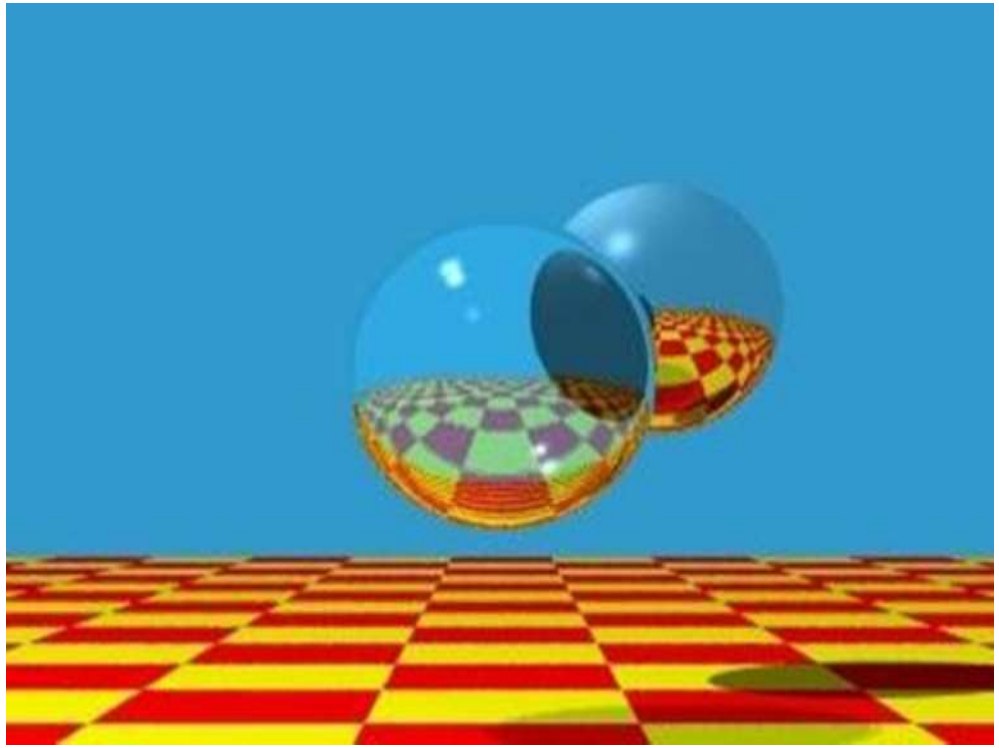




# 添加纹理

在不同的表面添加多样化的纹理，可以使得计算机绘制的结果更加逼真。

右图的地面上添加了棋盘格形式的纹理。



# 添加纹理

二维纹理

– 以矩形表面为例：

为矩形的四个顶点指定二维纹理坐标。

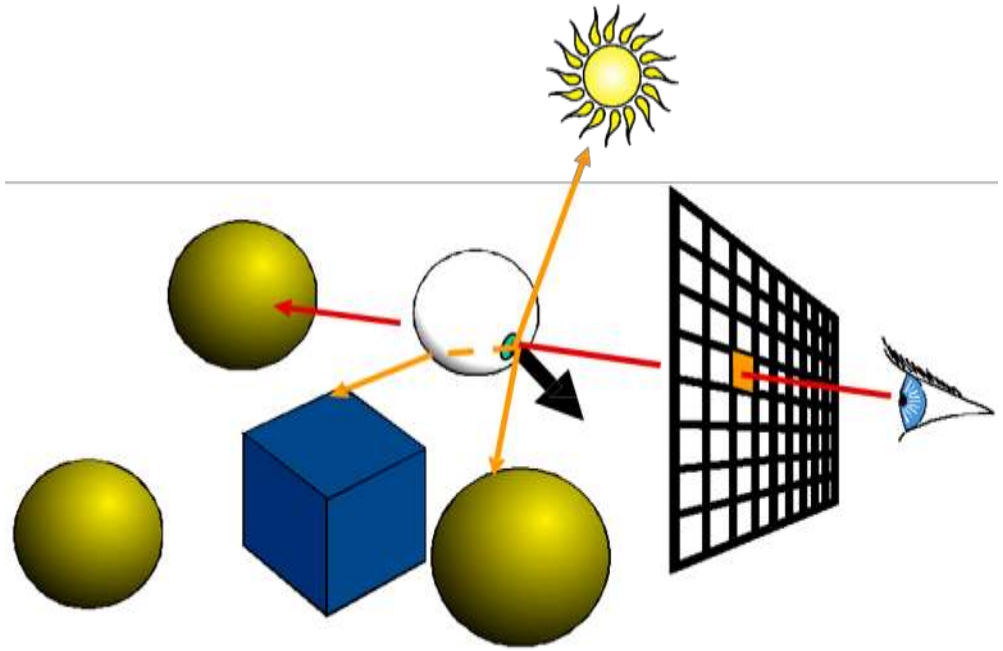
计算矩形内部与光线的交点的二维纹理坐标(双线性插值)。

使用该纹理坐标在纹理图上进行查找，根据查找结果赋予交点相应的颜色值。

# 光线跟踪符合物理原理吗？

真实的光子由光源出发，经场景物体弹射最终进入观察者的眼睛，而不是向光线跟踪一样。

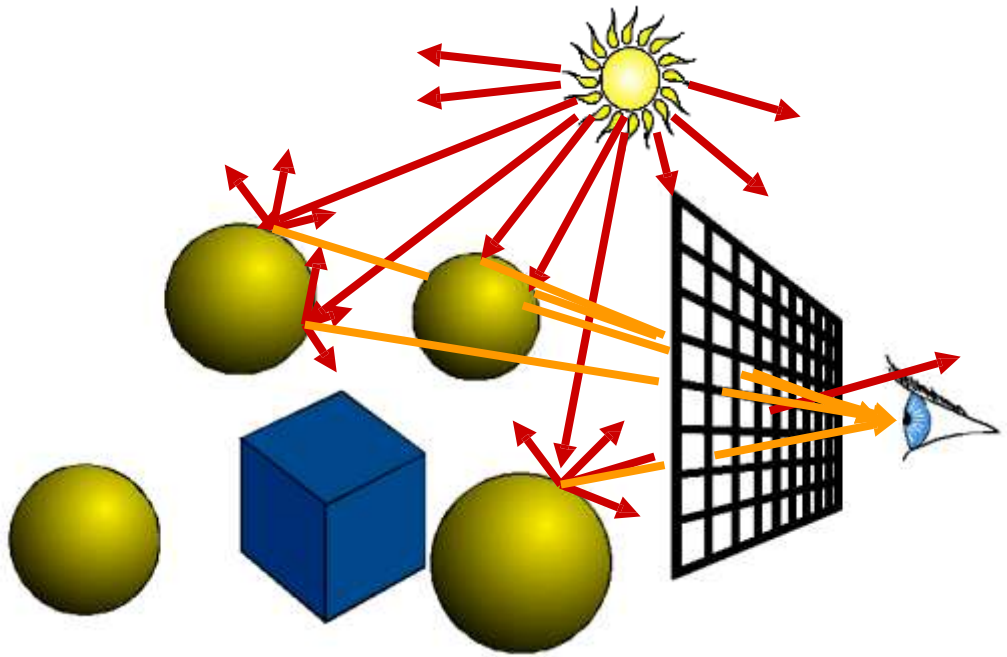
光线跟踪算法作的是反向 (Backward) 的跟踪， 对视点处的光子进行跟踪直到追溯到光源。



# 前向 (Forward) 光线跟踪

由光源出发对光线 (光子轨迹) 进行跟踪，经过弹射最终能够抵达视点的光线是极其少的。

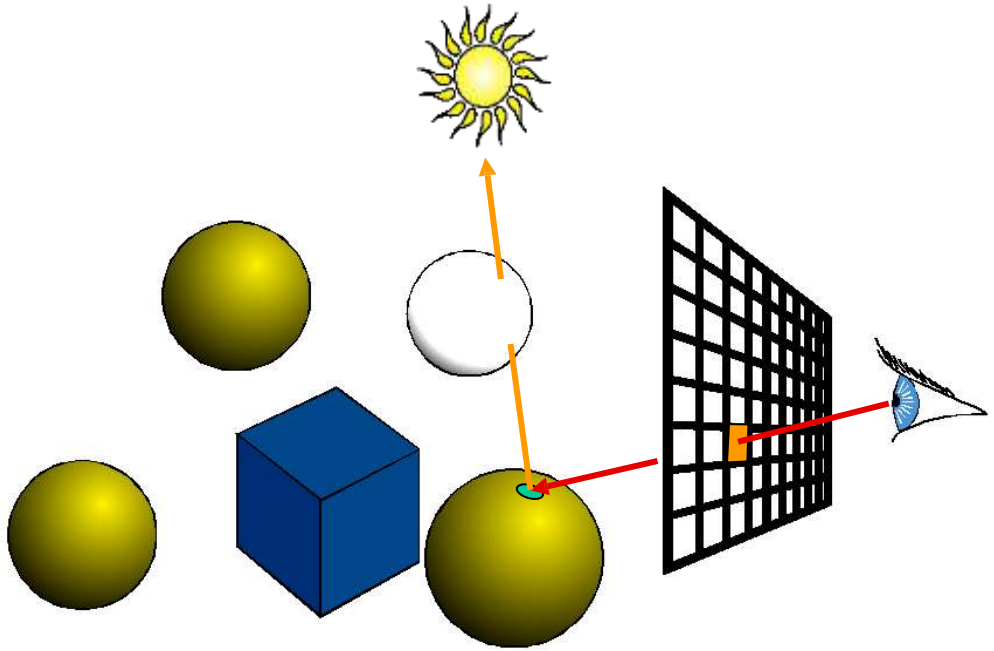
如何改进？ 每次弹射都发出一条指向视点的光线，但仍然低效。



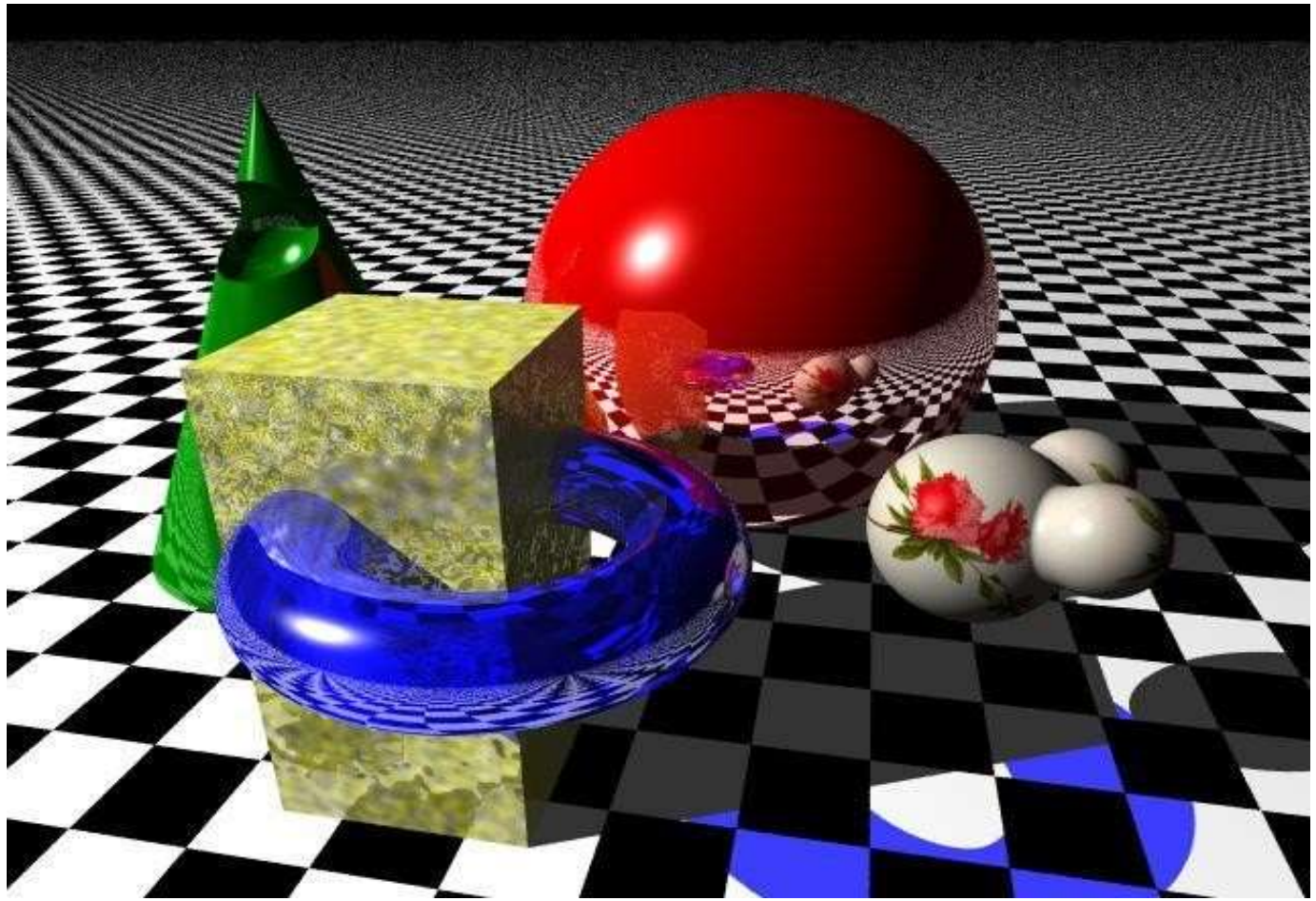
# 光线跟踪符合物理原理吗？

光线跟踪其实是使用许多巧妙的 Trick 来达到类似真实物理场景的效果。

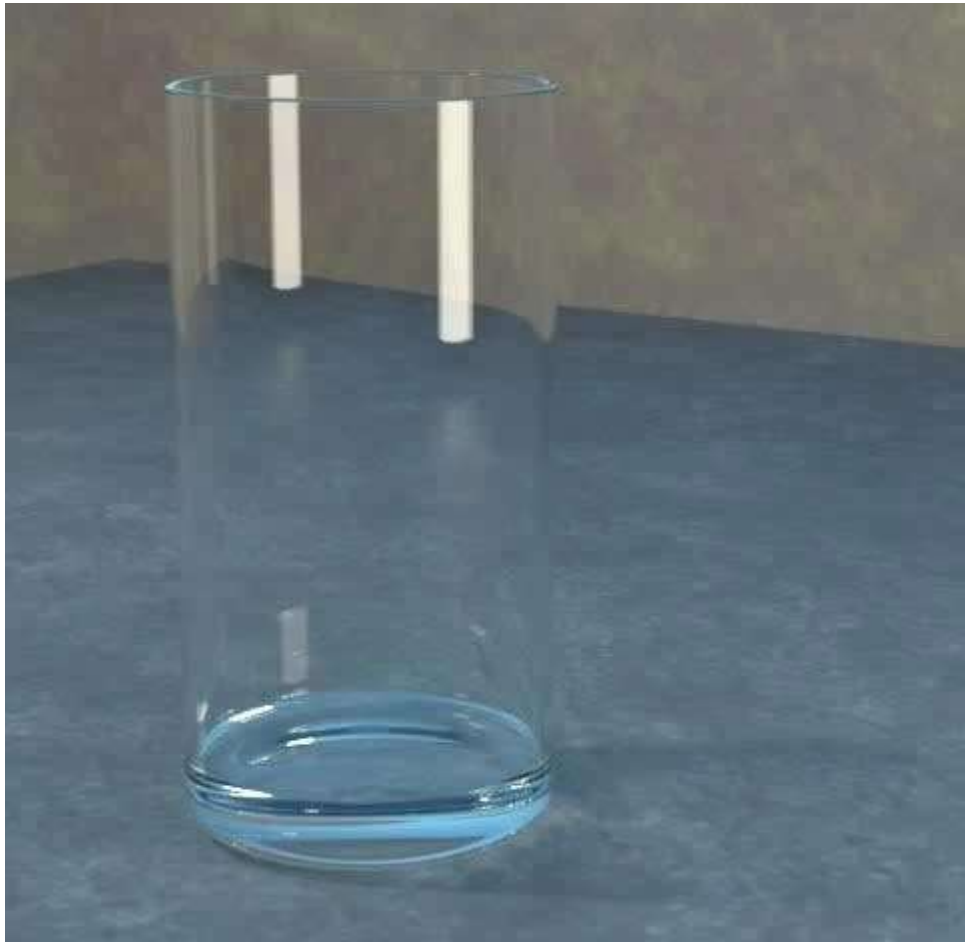
例如，在计算半透明物体的阴影时：直接将光的颜色乘以遮挡物体的透明因子以达到阴影效果，而忽略光的折射。



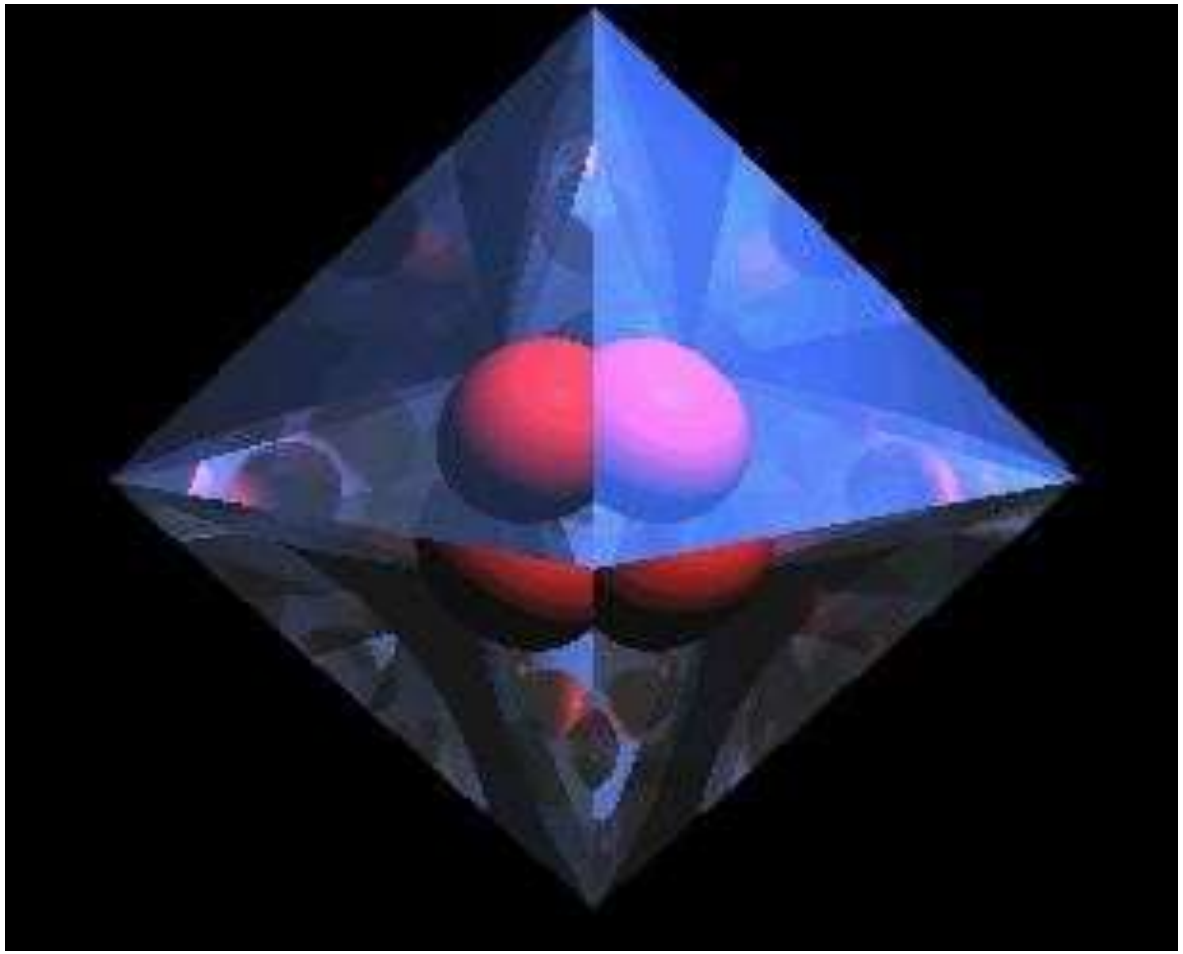
# 光线跟踪-效果演示



# 光线跟踪-效果演示



# 光线跟踪-效果演示



# 光线跟踪-效果演示