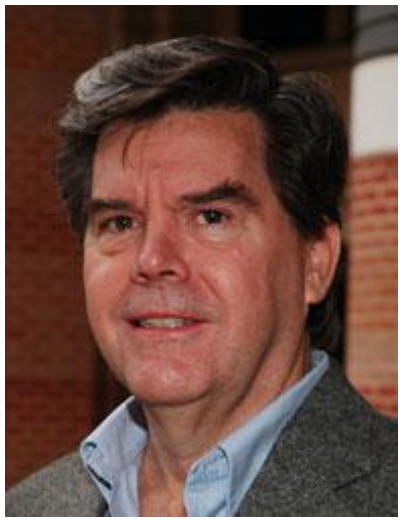
计算机图形学基础

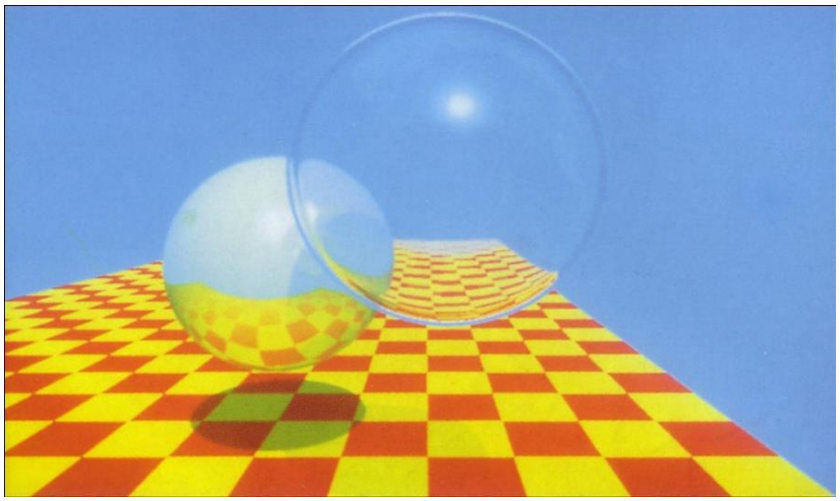
胡事民

清华大学计算机科学与技术系

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





光线跟踪算法的历史

- Turner Whitted , An improved illumination model for shaded display, Communications of the ACM, Vol. 23, No. 6, 343-349, June 1980. （SIGGRAPH 1979）
- http://en.wikipedia.org/wiki/Ray_tracing_(graphics)

Turner Whitted

- Turner Whitted 于 1978 年在北卡罗来纳州立大学 (NCSU) 获得 Ph.D学位，之后加入贝尔实验室，提出了著名的光线跟踪算法
- Turner Whitted 一共只发表过36篇论文（11篇SIG，2篇Communication of the ACM）
- Turner Whitted 于 2003 年当选美国工程院院士


NATIONAL ACADEMY OF ENGINEERING
MEMBERSHIP WEBSITE
member directories

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

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

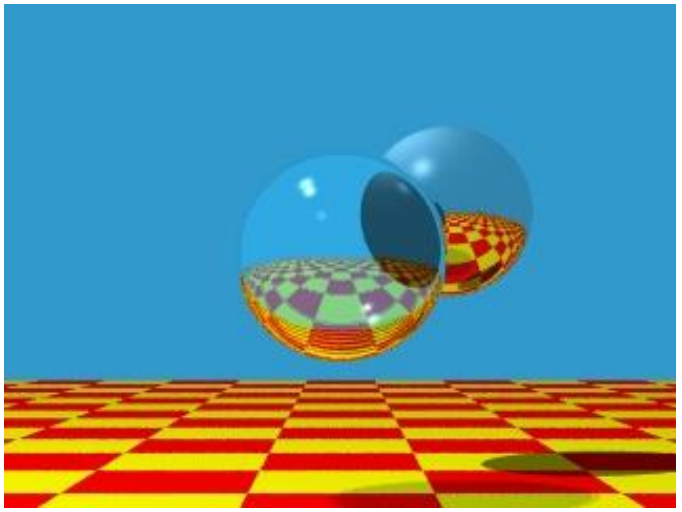
Country: United States
State: WA

Member Type: Member

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

光线跟踪 (Ray Tracing)

- 光线跟踪概述
- 光线求交
- 阴影
- 透明和镜面反射
- 纹理
- 光线跟踪的一些思考
- 更复杂的光线追踪框架

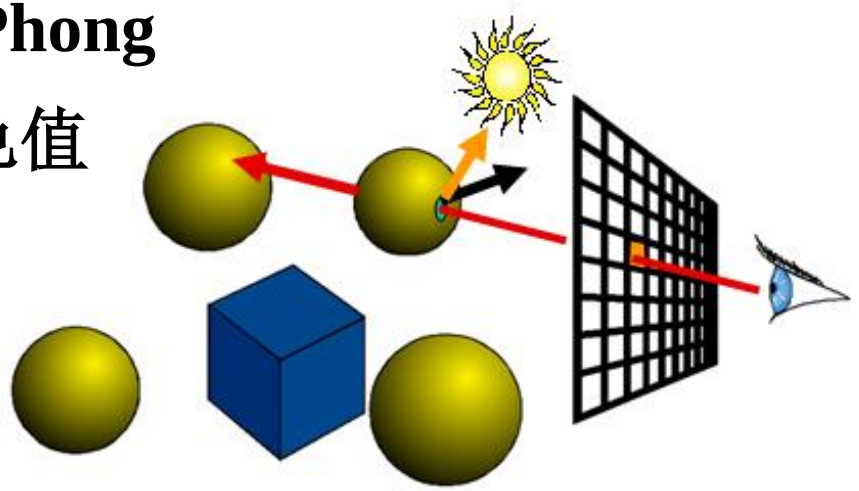


光线跟踪概述

- 光线跟踪算法
  - 一个有效、并被广泛使用的真实感绘制算法，它可以实现之前算法很难达到的效果
  - 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、经物体表面反复弹射的光线 (Rays)
    - 一部分光线 (Rays)最终进入视点、射入人们的眼中，使我们看到物体

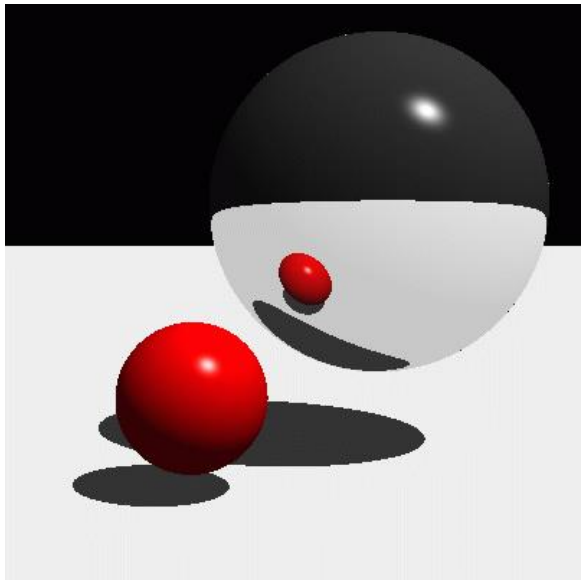
光线跟踪的思路和框架(Whitted idea)

- 将视窗区域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（逆向跟踪）：
  - 计算从视点出发、经过像素 P的中心位置的光线 (Ray) 与场景中物体的第一个交点
  - 使用局部光照模型 (如 Phong 模型) 计算交点处的颜色值
  - 沿交点处的反射和折射方向对光线进行跟踪



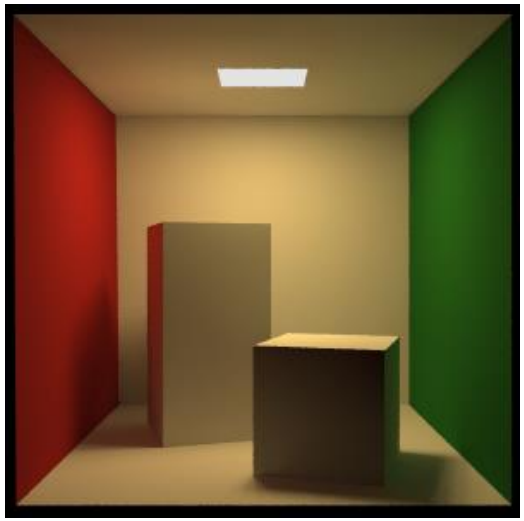
光线跟踪概述

- 光线跟踪的特征：
  - 通过光线跟踪，可以很容易地表现出阴影、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、立方体等)，光线跟踪还适用于更复杂的物体表示方法(例如多边形网格、复合形体等)

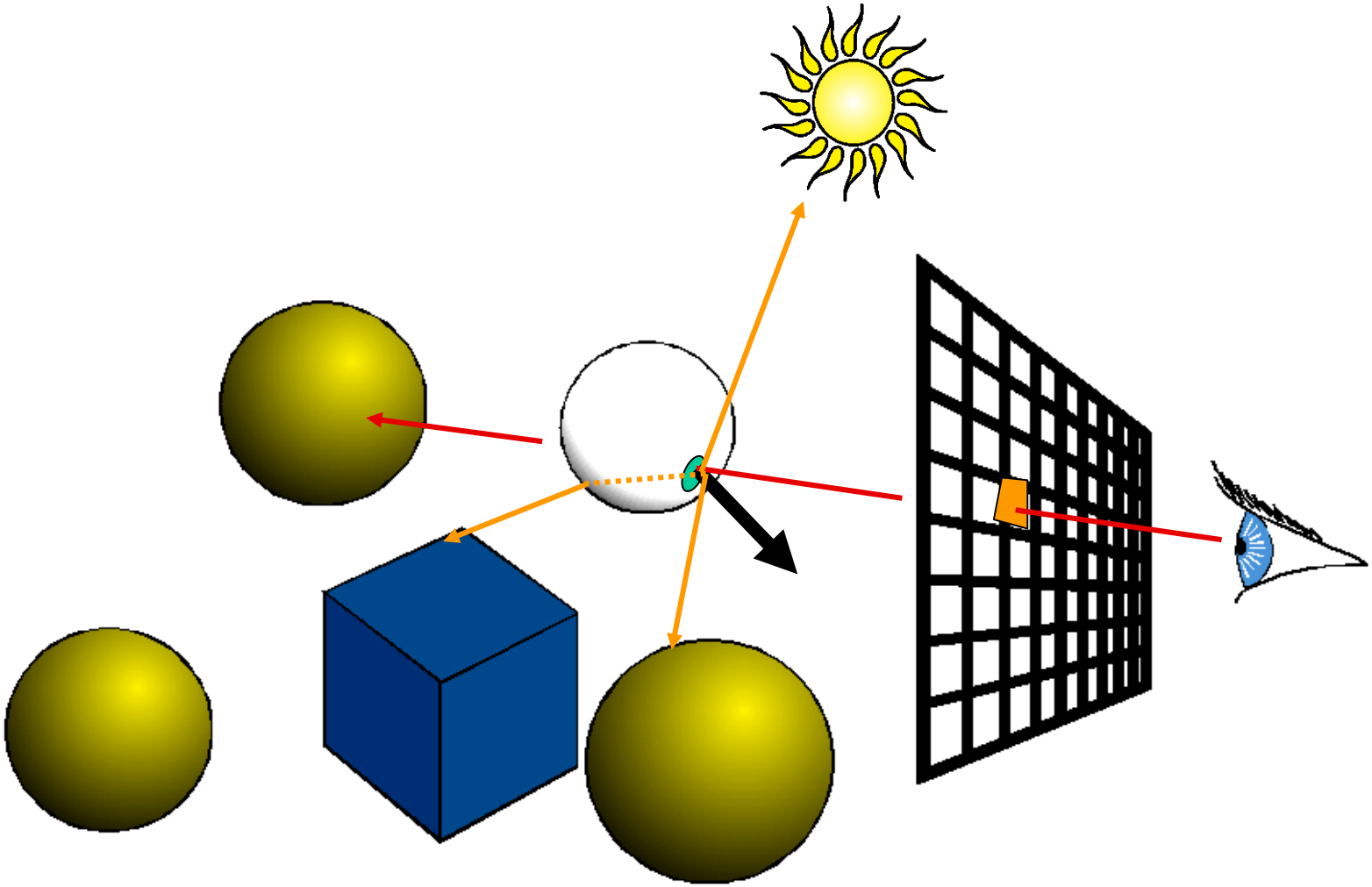


光线跟踪：最基本的绘制方法

- 早期的光线跟踪算法主要面向室内规则场景，如著名的康奈尔盒 **Cornell Box**
- 作为一种最经典、最基本的绘制方法，光线跟踪一般会作为**ground truth**绘制结果，用于验证比较新的绘制方法的正确性。



使用递归实现的光线跟踪算法



使用递归实现的光线跟踪算法

```
IntersectColor( vBeginPoint, vDirection)
{
   Determine IntersectPoint;
   Color = ambient color;
   for each light
        Color += local shading term;
   if(surface is reflective)
      color += reflect Coefficient *
           IntersectColor(IntersecPoint, Reflect Ray);
   else if ( surface is refractive)
     color +=  refract Coefficient *
           IntersectColor(IntersecPoint, Refract Ray);

   return color;
}
```

单选题 1分

设置

光线跟踪算法的主要计算量在于

A 计算可见点处的漫反射光强

B 计算可见点处的反射方向

C 计算可见点处的折射方向

D 确定光线的可见点

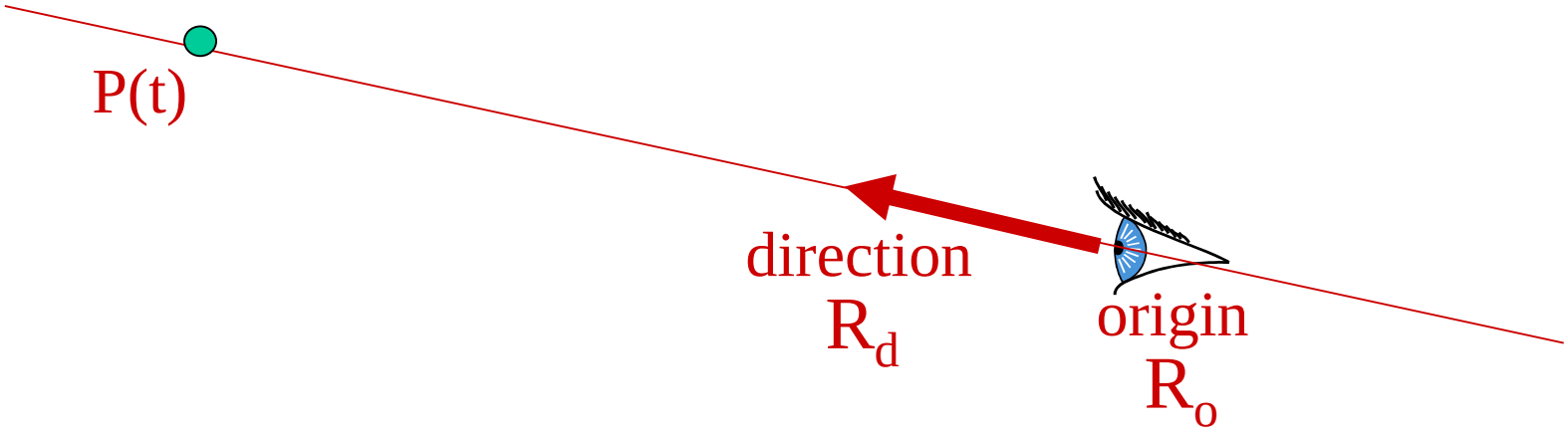
提交

光线求交 (Ray Intersection)

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

光线的表示

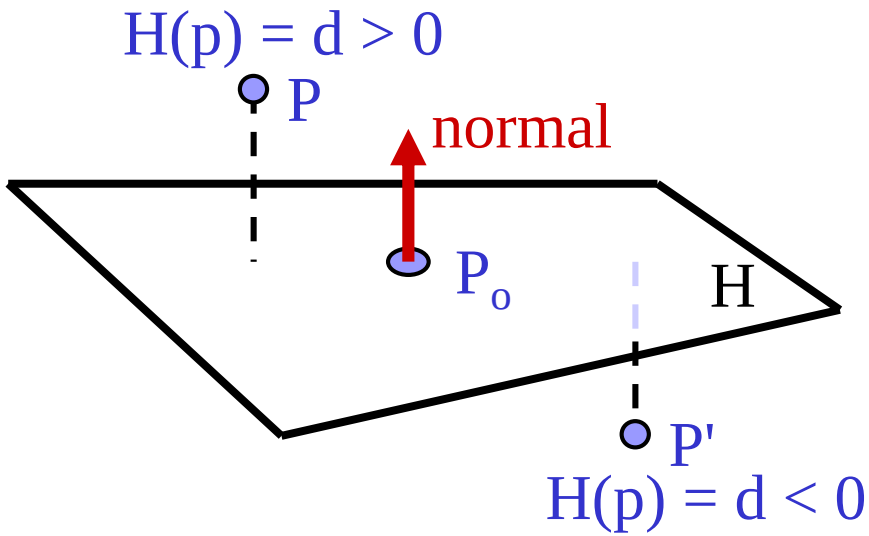
- 光线 (射线) 的参数表示
  - $\boldsymbol{P(t) = R_o + t\,R_d}$
  - $R_o = (x_o,y_o,z_o)$ 是光线的源点；向量 $R_d = (x_d,y_d,z_d)$ 代表光线的朝向，通常来说 R_d 是单位向量
  - 参数 t 表示光线到达的位置：在光线的正方向上，参数 t 都是正数：$t > 0$



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：$H(P) = Ax+By+Cz+D = 0$

    $= n\,P + D = 0$
- 点到平面的距离：
  - 当 n 是单位法向量时，P 到平面 H 的距离就是 $H(P)$



光线与平面求交

- 给定满足如下的平面方程：

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

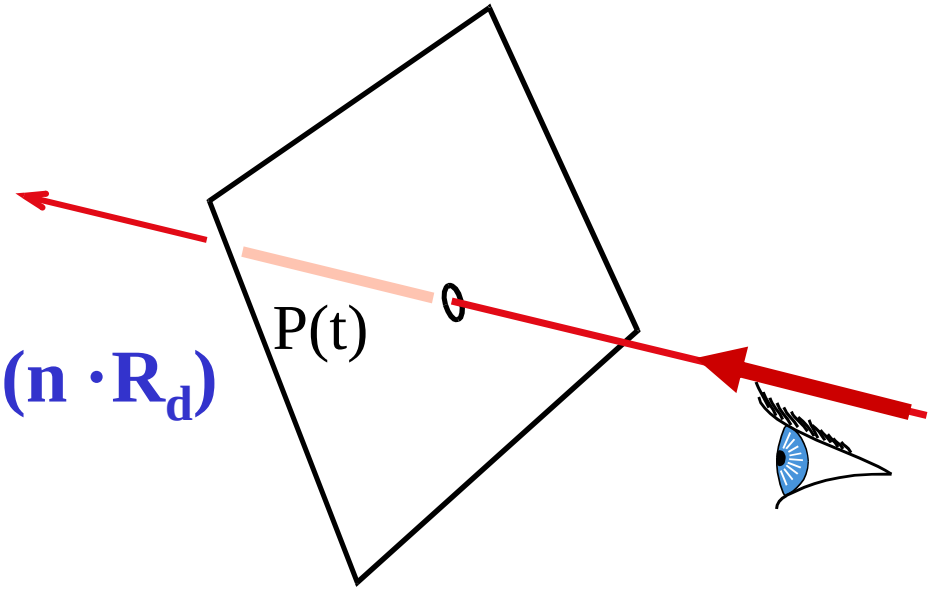
- 如何计算一条光线与该平面的交点？
- 连列方程如下：

$$P(t) = R_o + t\,R_d$$

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

解得：$\mathbf{t = -(D+n \cdot R_o) / (n \cdot R_d)}$

最后验算 $t > 0$



光线与三角形求交

- 在实时绘制中，基于三角形网格的几何表示(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
- 计算光线和三角形的交点可以有多种方法，但都包含如下两个部分：
  - 首先计算光线和三角形所在平面的交点
  - 在三角形所在平面内考虑，判断光线与平面的交点是否位于三角形的内部

光线与三角形求交

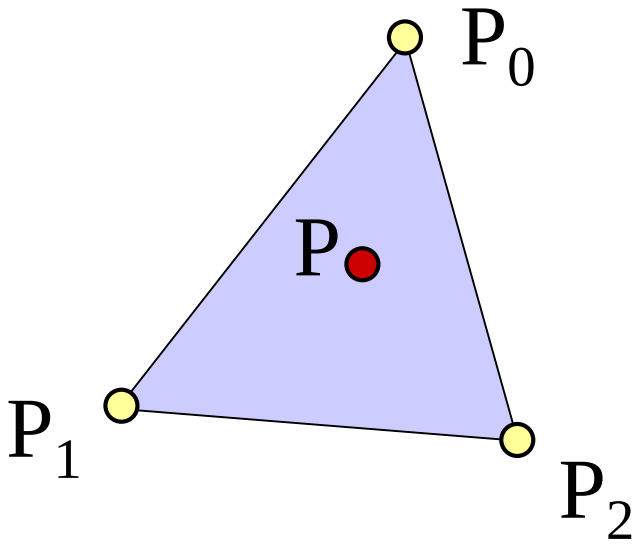
- 重心坐标：
  - 三角形 $P_0P_1P_2$ 内的一点 P 可以表示成：

$$P = \alpha P_0 + \beta P_1 + \gamma P_2$$

其中 (α, β, γ) 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



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：

$$R_o+tR_d=(1-\beta-\gamma)P_0+\beta P_1+\gamma P_2$$

- 即：

$$(R_d \quad P_0-P_1 \quad P_0-P_2)\begin{pmatrix} t \\ \beta \\ \gamma \end{pmatrix}=P_0-R_o$$

- 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：$E_1 = P_0 - P_1, \mathrm{E}_2 = P_0 - P_2, S = P_0 - R_o$
  根据 Cramer 法则，我们可以写出：

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

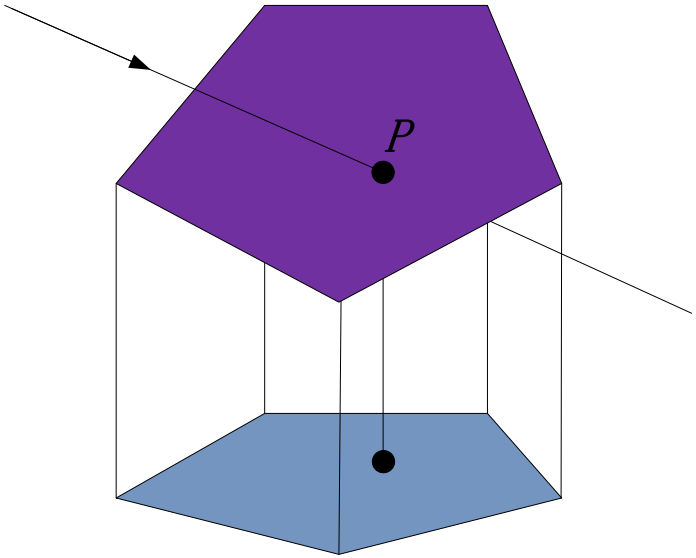
- 最后需要检查 $t > 0$ 且 $0 \le \beta, \gamma \le 1, \beta + \gamma \le 1$ 以保证交点位于三角形的内部

光线与多边形求交

- 尽管三角面片是最为常用的基本渲染单元，但是在很多场合也需要对光线与多边形求交
- 一个 n–多边形可以表示成 n 个顶点的有序列表：$\{v_0, v_1, \ldots, v_{n-1}\}$，且该多边形的 n 条边分别为：$v_0v_1, v_1v_2, \ldots, v_{n-2}v_{n-1}, v_{n-1}v_0$。其中 $v_0, v_1, \ldots, v_{n-1}$位于同一平面，该平面可以表示为：

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

光线与多边形求交



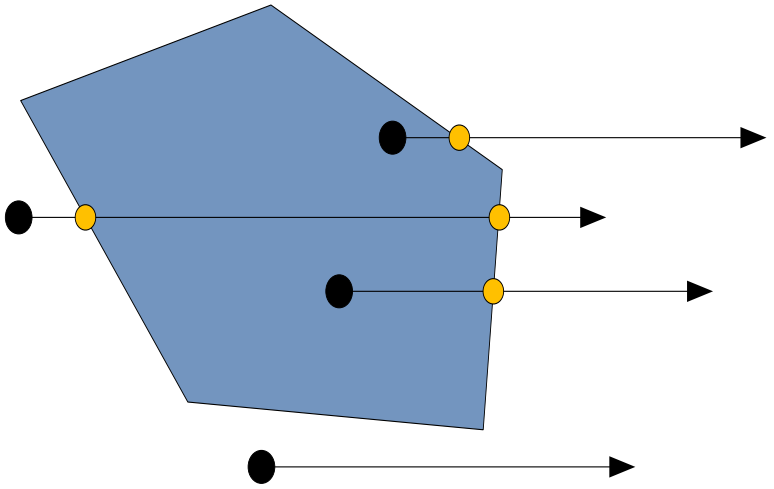
- 我们首先计算光线与多边形所在平面的交点
- 如果交点存在，我们需要判断这个交点是否位于多边形的内部
- 为了进行这一判断，我们将交点以及多边形的所有顶点投影到 X-Y, Y-Z, Z-X平面中一个 (如图所示)，以方便后续处理

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，因此基于该定理的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
- 奇偶检测算法是所有无需预处理的方法中最快的

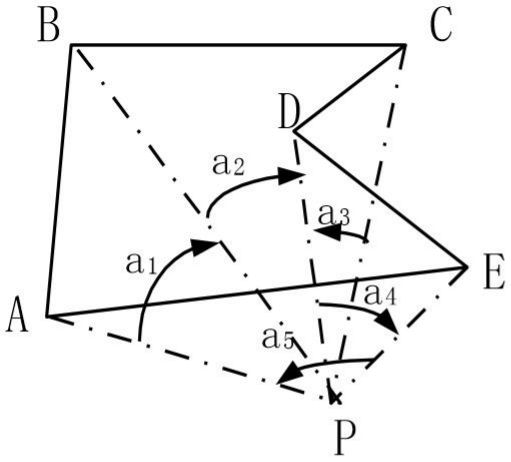


光线与多边形求交

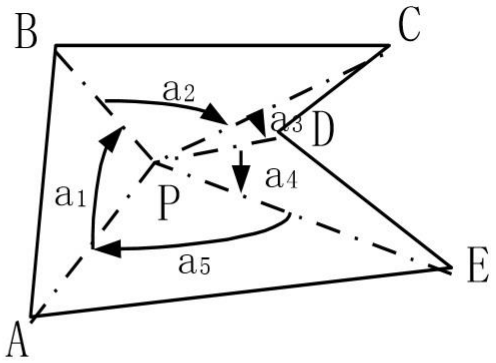
- 可以将需要检测的点作为坐标原点，使用 X 轴正半轴作为检测交点的射线，然后逐一对多边形的每条边进行判断：
  - 如果多边形某条边的两端点的 Y 坐标符号相同，则这条边与 X 轴正半轴无交点
  - 否则，计算这条边与 X 轴的交点：
    - 如果这个交点位于 X 轴正半轴上，那么将统计总交点个数的变量 +1
    - 否则，这条边与X 轴正半轴无交点

最快的点包含判别算法

- 弧长法



(a) 被测点 P 在多边形外



(b) 被测点 P 在多边形内

弧长法

- 以被测点为圆心，作单位圆，计算其在单位圆上弧长的代数和
  - 代数和为 0:　点在多边形外部
  - 代数和为 2π: 点在多边形内部
  - 代数和为 π:　点在多边形边上

  优点：算法稳定

  但计算角度需要算反三角函数，计算量太大！

  此外还会受浮点数精度影响

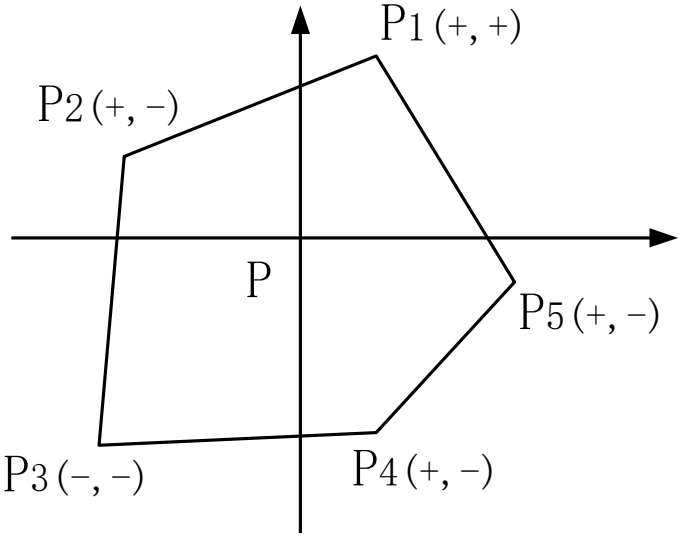
弧长法

- **以顶点符号为基础的弧长累加方法**
  - 将坐标原点移到被测点 P。各象限内点的符号对分别为 (＋, ＋)，(－, ＋)，(－, －)，(＋，－)
  - 算法规定：若顶点 P_i 的某个坐标为0，则其符号为＋；若顶点 P_i 的 x, y 坐标都为0，则说明这个顶点为被测点，我们在这之前予以排除。于是弧长变化如下表：

弧长法

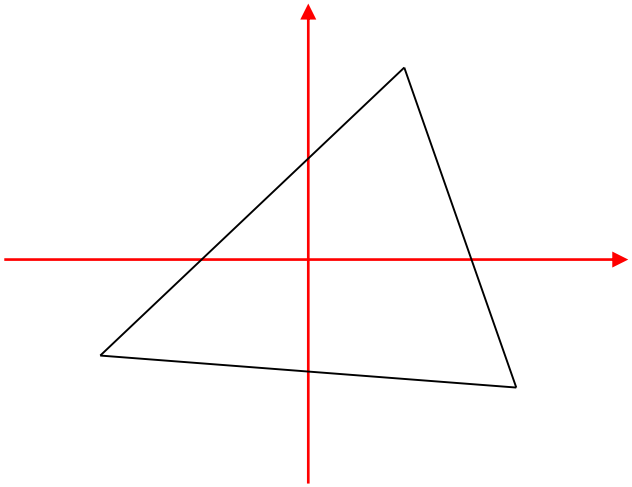
- 表：符号对变化与弧长变化的关系

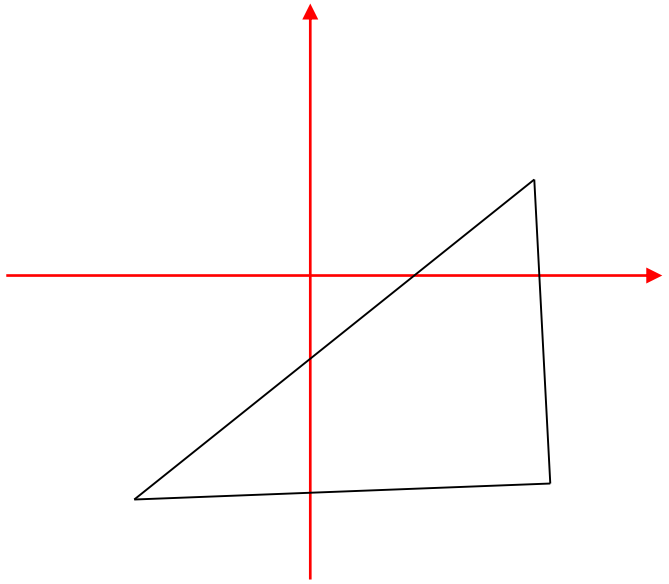
(sx_i, sy_i)	(sx_{i+1}, sy_{i+1})	弧长变化	象限变化
(+ +)	(+ +)	0	I → I
(+ +)	(– +)	π/2	I → II
(+ +)	(– –)	±π	I → III
(+ +)	(+ –)	-π/2	I → IV
…	…	…	…



弧长法

- 值得注意的是，当边的终点 P_{i+1} 在起点 P_i 的相对象限时，弧长变化可能增加或减少 π：





弧长法

- 设 (x_i, y_i) 和 (x_{i+1}, y_{i+1}) 分别为边的起点和终点坐标。计算：

$$f = y_{i+1}x_i - x_{i+1}y_i$$

- 若 $f=0$，则边穿过坐标原点。若 $f > 0$，则弧长代数和增加 π，若 $f < 0$，则弧长代数和减少 π。

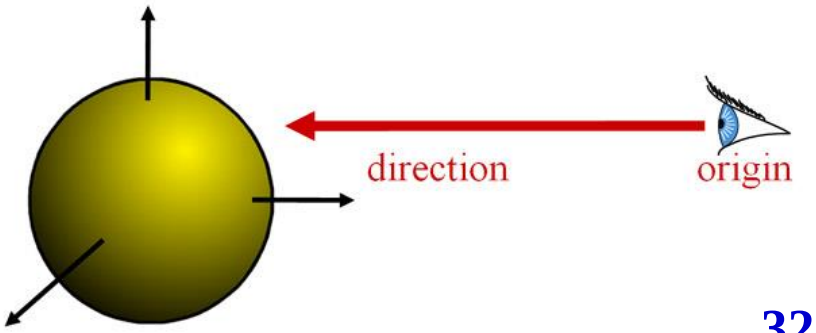
光线与球面求交

- 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：

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

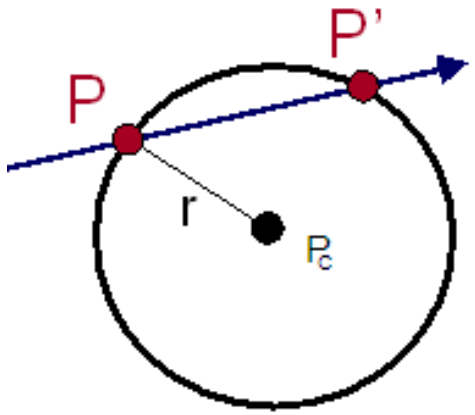
  - 求光线与球面的交点，相当于要求解如下的方程：

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



球面求交的代数方法

- 将之前的方程进行化简：



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

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

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

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

由于 R_d 是单位向量：

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

记为：　　$t^2+2tb+c=0$

解得：　　$t=-b\pm\sqrt{b^2-c}$

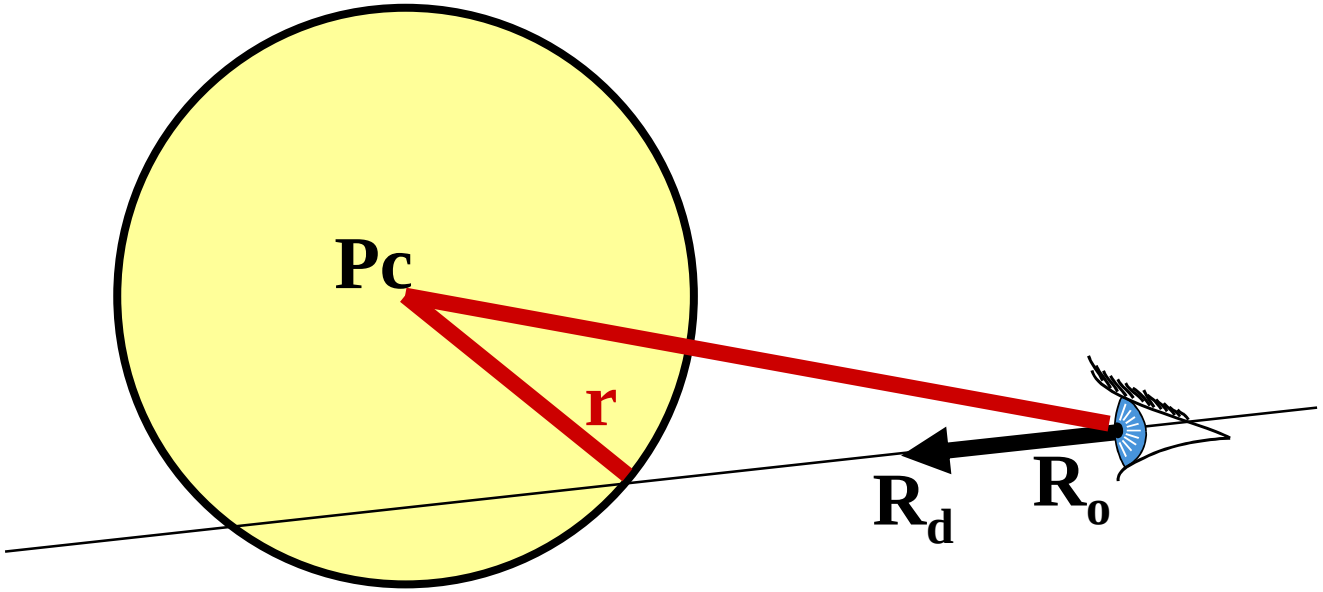
光线与球面求交

- 之前的代数方法存在很多改进和加速的空间：例如，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

球面求交的几何方法

- 首先，计算由光源指向球心的向量 l：

$$\vec{l} = P_c - R_0$$



球面求交的几何方法

- 计算由光源指向球心的向量 l：

$$\vec{l} = P_c - R_0$$

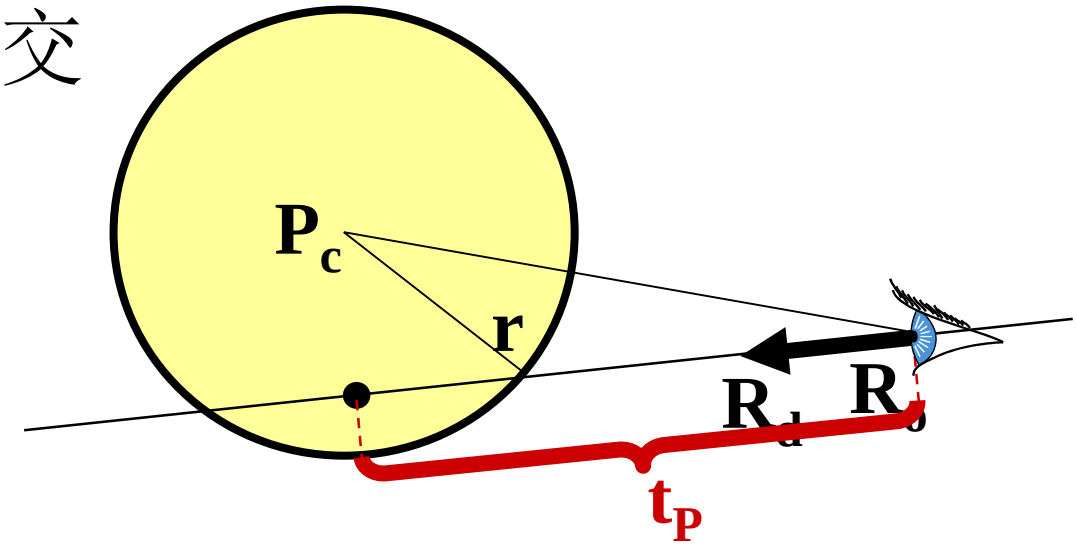
- 判断光源是否位于球体内部：
  - 位于球体内部：$\vec{l}^2 < r^2$
  - 位于球体外部：$\vec{l}^2 > r^2$
  - 位于球面上：$\vec{l}^2 = r^2$
  - 当光源位于球面上时，需要小心考虑退化情况。

球面求交的几何方法

- 然后，计算球心到光线所在直线的投影点 (垂足)：

$$t_p = \vec{l} \cdot R_d$$

- 如果光源在球体外部并且 $t_p < 0$，那么光线与球面不相交

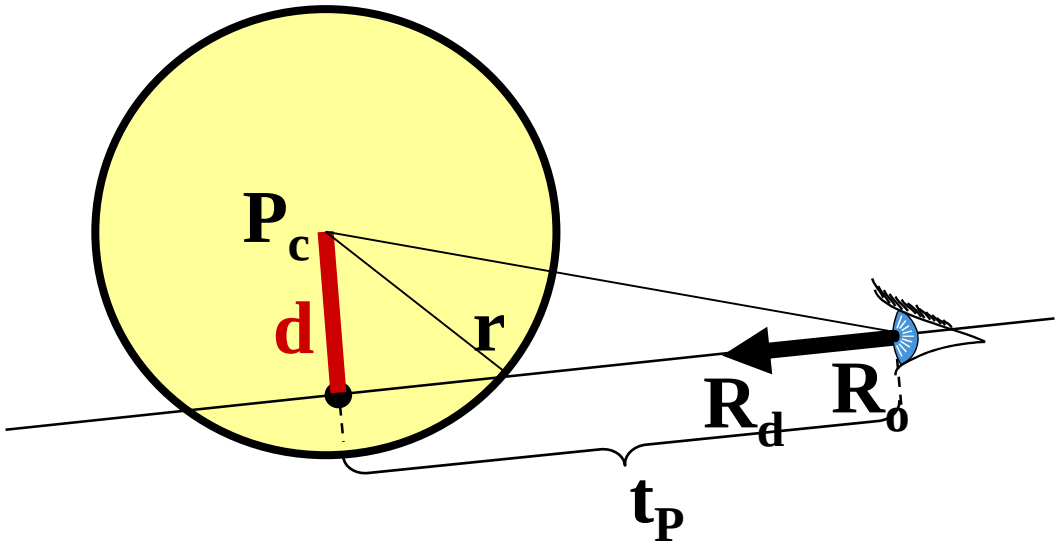


球面求交的几何方法

- 然后，计算球心到光线所在直线的距离：

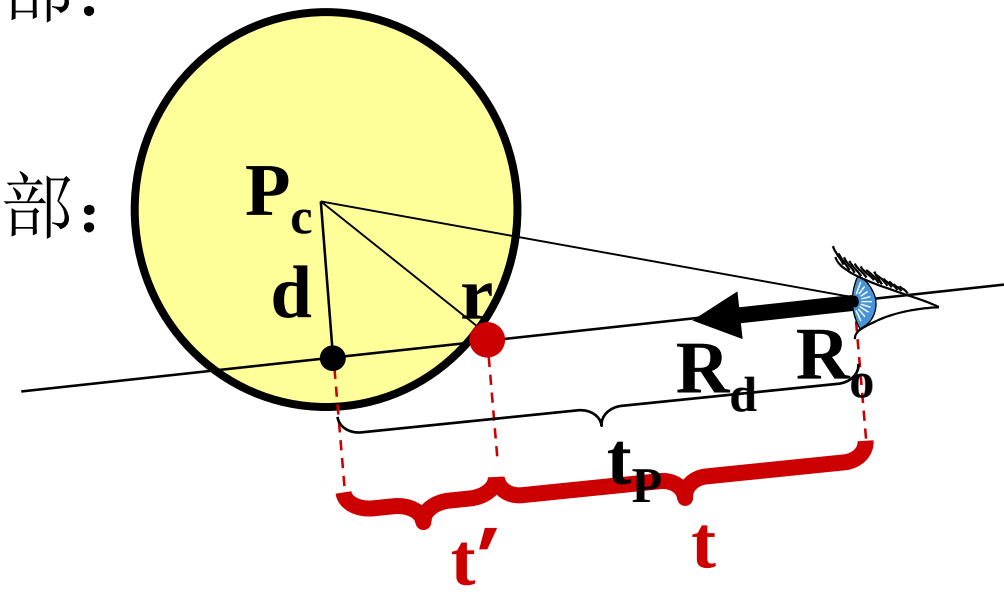
$$d^2 = \vec{l}^2 - t_p^2$$

- 如果 $d > r$，那么光线与球面不相交



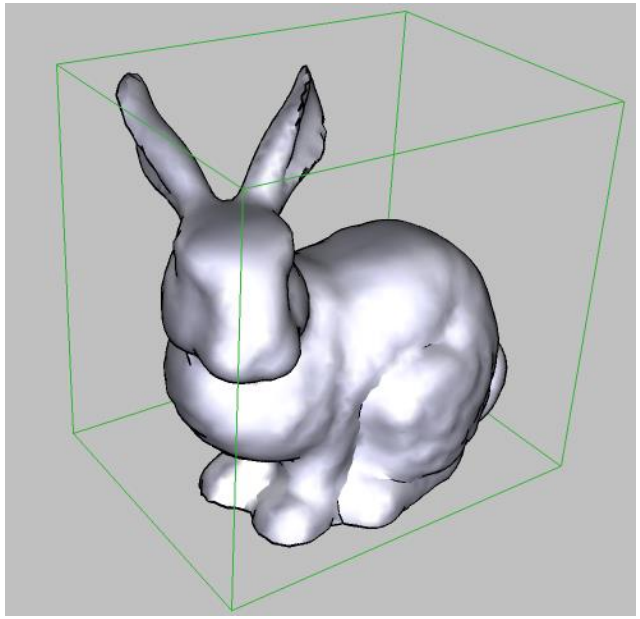
球面求交的几何方法

- 然后，计算距离 t'，t' 是投影点 到光线与球面的交点的距离：$t'^2 = r^2 - d^2$
- 最后，求解光线与球面的交点：
  - 如果光源在球体外部：
    - $t = t_p - t'$
  - 如果光源在球体内部：
    - $t = t_p + t'$



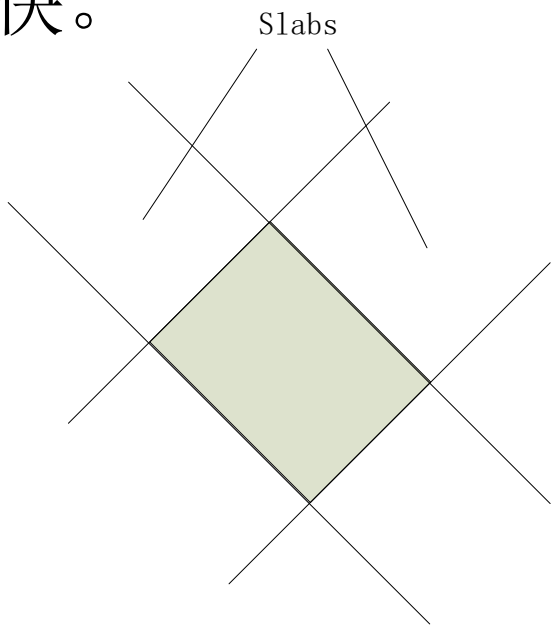
光线与长方体求交

- 光线与长方体 (Boxes) 的求交在图形学中有重要的意义，因为我们经常会使用长方体来包围复杂的几何形体，这样的长方体称为包围盒 (Bounding Box)
- 通过包围盒，在需要判断光线和物体是否相交时，可以先使用包围盒进行一次粗糙判断：如果光线和包围盒不相交，则和物体一定不相交



光线与长方体求交

- 我们介绍一个基于 Slab 的快速长方体求交算法，由 Haines 提出：
  - 一个 Slab 是指两个平行的平面，将两个平行平面放在一组是为了计算更快。
  - 每个长方体是由三个 Slab (三组相对的面) 构成的



光线与长方体求交

- 对每个 Slab，计算其两个平面与光线的交点，并将这两个交点按照其参数 t 的大小排序，记为：$t_i^{\min}$ 和 $t_i^{\max}$ $(i = 0,1,2)$。
- 然后，我们计算：

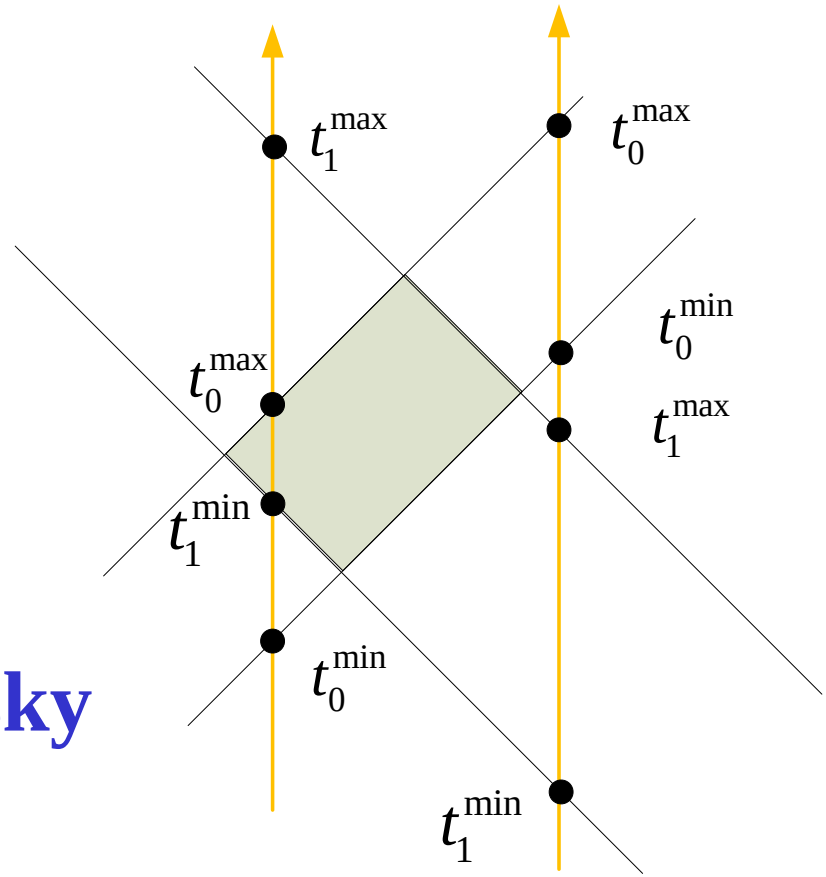
$$t^{\min} = \max(t_0^{\min}, t_1^{\min}, t_2^{\min});$$

$$t^{\max} = \min(t_0^{\max}, t_1^{\max}, t_2^{\max});$$

- 并测试：$t^{\min} < t^{\max}$
  - 如果 $t^{\min} < t^{\max}$，那么光线与长方体相交，且 $t^{\min}$ 是射入长方体时的交点；$t^{\max}$ 是射出长方体时的交点
  - 否则，光线与长方体不相交

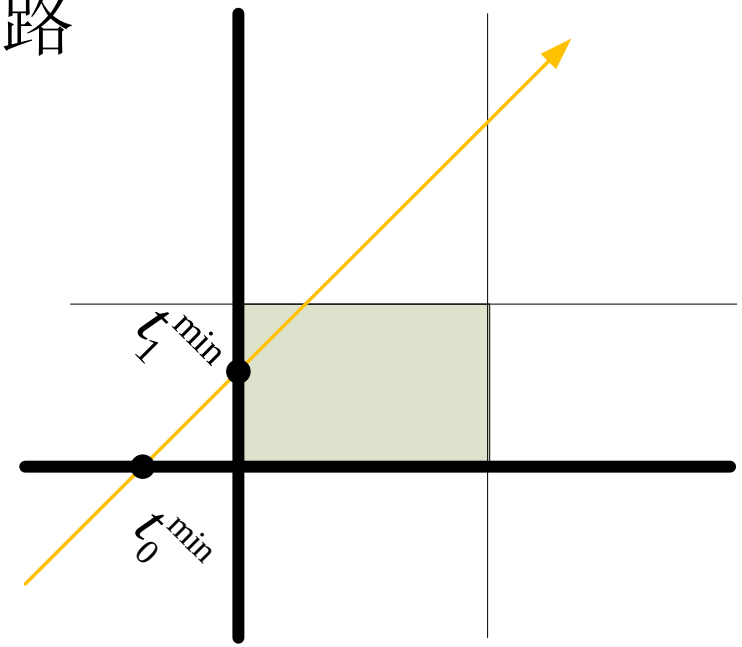
光线与长方体求交

- 如右图中的两条射线 (二维情况):
  - 左边的射线与矩形相交，满足:
    $t^{\min} < t^{\max}$
  - 右边的射线与矩形不相交，不满足:
    $t^{\min} < t^{\max}$
- 思想上与 **Liang-Barsky** 裁剪算法类似



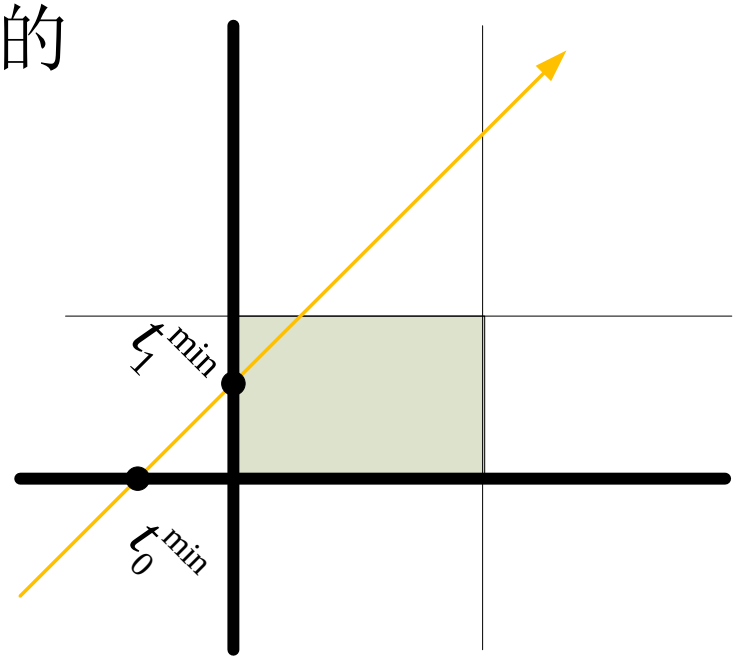
光线与长方体求交

- 针对沿坐标轴放置的长方体，可以使用更为高效的 Woo 算法
  - Woo 针对沿坐标轴放置的长方体，给出了一些特别的优化和加速思路
  - [Andrew Woo, Fast ray–box intersection, 1990]



光线与长方体求交

- Woo 算法
  - 首先，确定三个比较近的平面 (将远的平面过滤掉，如图中较细的黑线)
  - 计算光线与这三个平面的交点对应的 t 值
  - 将这三个 t 值中的最大一个对应的点作为可能的交点



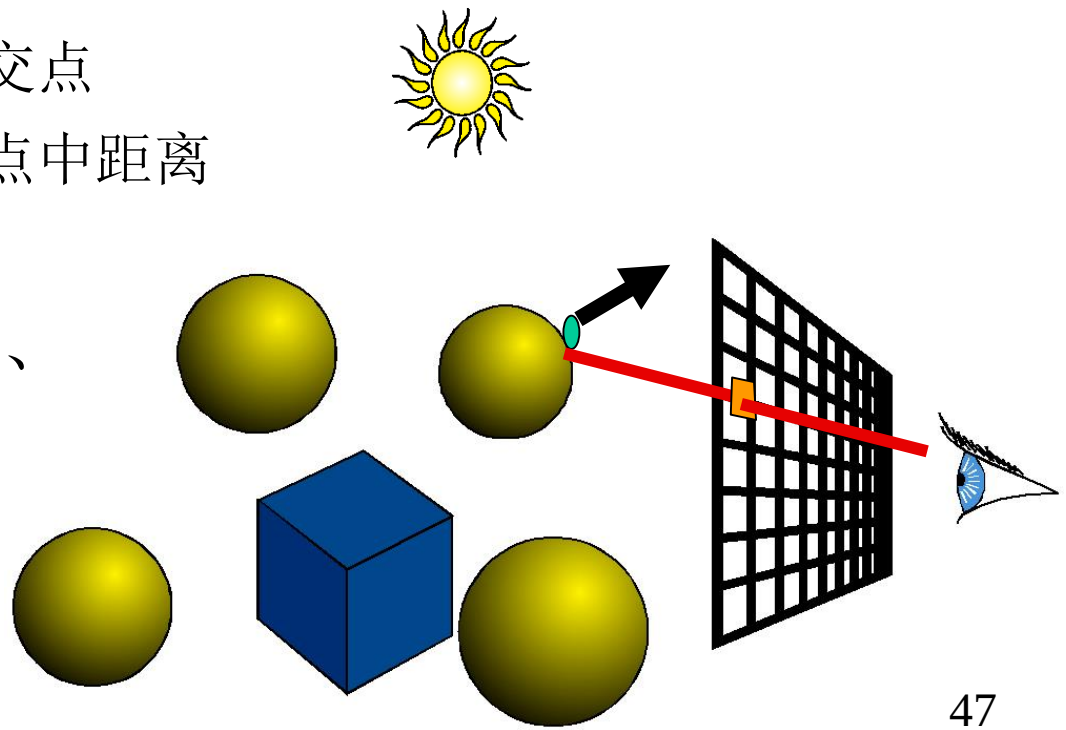
光线与长方体求交

- Woo 算法
  - 检查这个可能的交点是否位于长方体的表面上，如果交点不位于长方体的表面，则光线与长方体不相交

讨论完光线求交，我们接下来讲光线跟踪

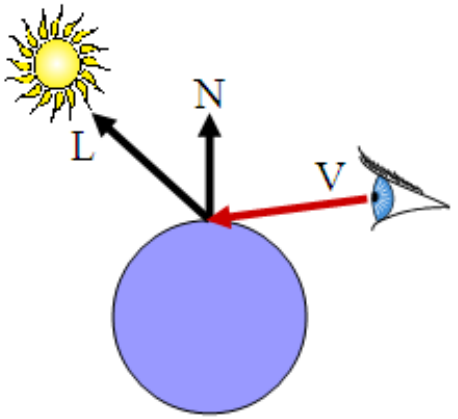
光线投射 (Ray Casting)

- 对于每个像素：
  - 由视点向该像素中心投射一条光线，射向场景中
  - 对场景中的各个物体：
    - 计算与投射光线的交点
    - 存储所有物体的交点中距离视点最近的一个
  - 根据光照、物体材质、以及法向方向，使用局部光照模型计算像素颜色值

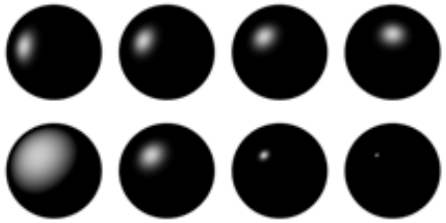


光线投射 (Ray Casting)

- 明暗效果由光线第一次相交的物体的表面法向、材质、视点、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
- **光线投射如何加速？**



Diffuse sphere

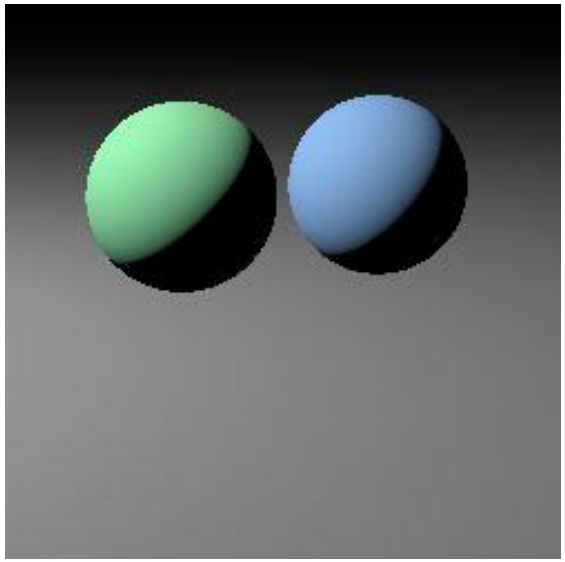


Specular spheres

光线投射的伪代码

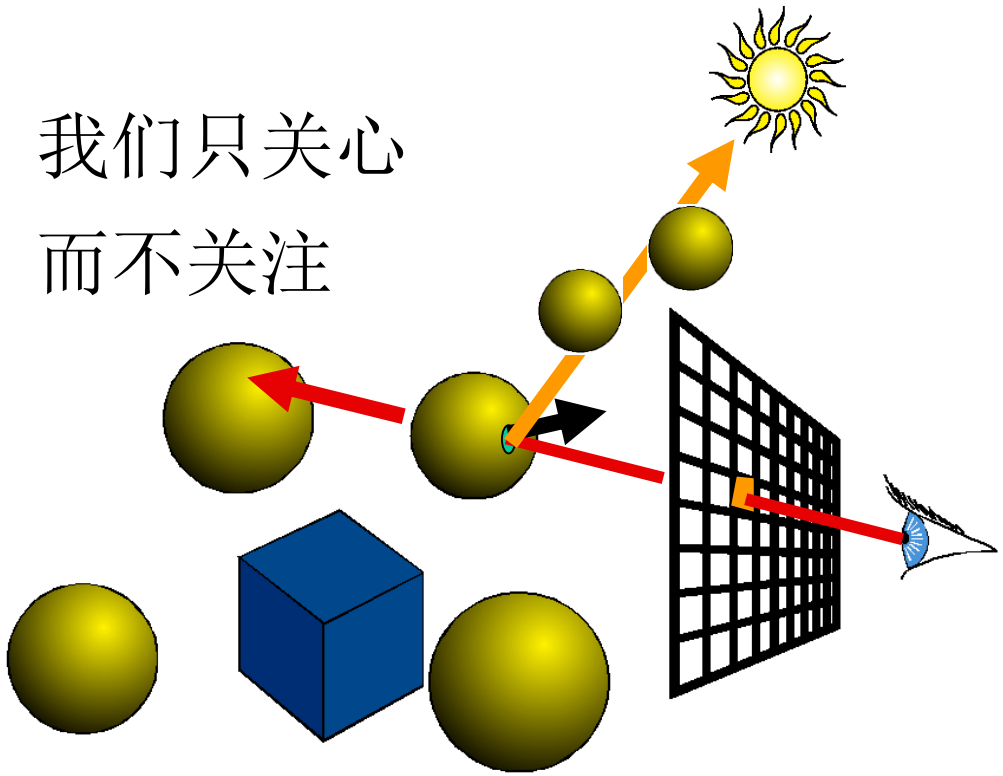
- FOR each pixel
  - Cast a ray and find the intersection point 最近邻交点
  - IF there is an intersection
    - color = ambient
    - FOR each light

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



添加阴影 (Shadows)

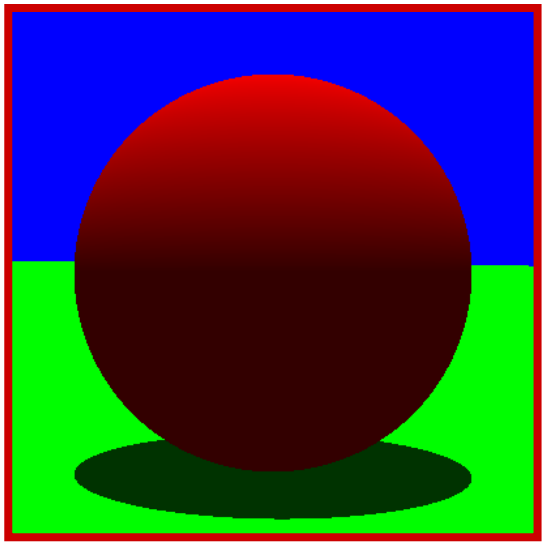
- 如图所示，由交点向光源方向投射一道光线，如果这道光线与场景中物体有交点，则该交点属于阴影区域
- 计算阴影区域时，我们只关心是否与物体相交，而不关注哪个是最近交点



添加阴影 (Shadows)

- FOR each pixel
  - Cast a ray and find the intersection point
  - IF there is an intersection
    - color = ambient
    - FOR each light
      - IF intersection point is not in shadow area of the light (evaluated by shadow rays)

        color += shading from this light
    - return color
  - ELSE return background color



添加反射和折射效果

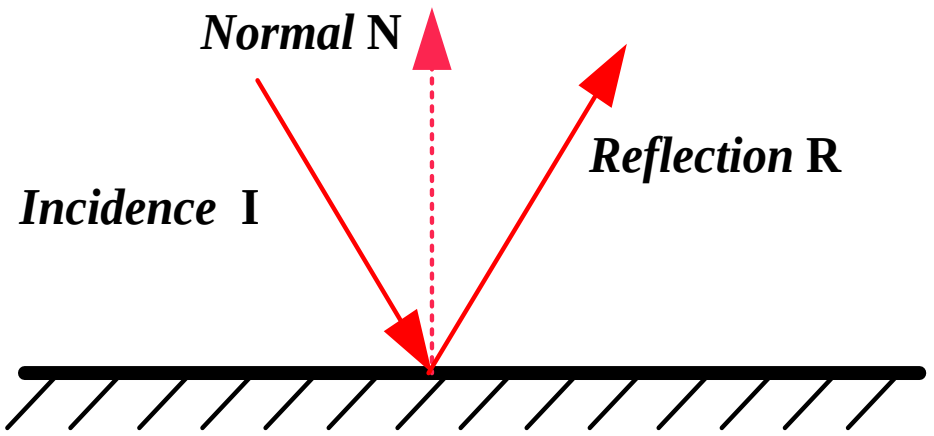
- 光线跟踪具有模拟物体表面镜面反射、折射效果的能力：
  - 首先，计算光线与场景中物体的最近的交点
  - 然后，计算光线在交点处因物体反射和折射所产生的新的光线的方向，新的方向由入射光方向、物体表面法向、及介质共同决定
  - 对新产生的光线 (反射光线和折射光线) 分别继续进行跟踪

光线的反射

- 反射定律：
  - 入射角 = 反射角
  - 入射光线、反射光线、以及物体表面法向量位于同一个平面内
- 反射光线的方向向量 R 可以按如下公式计算：

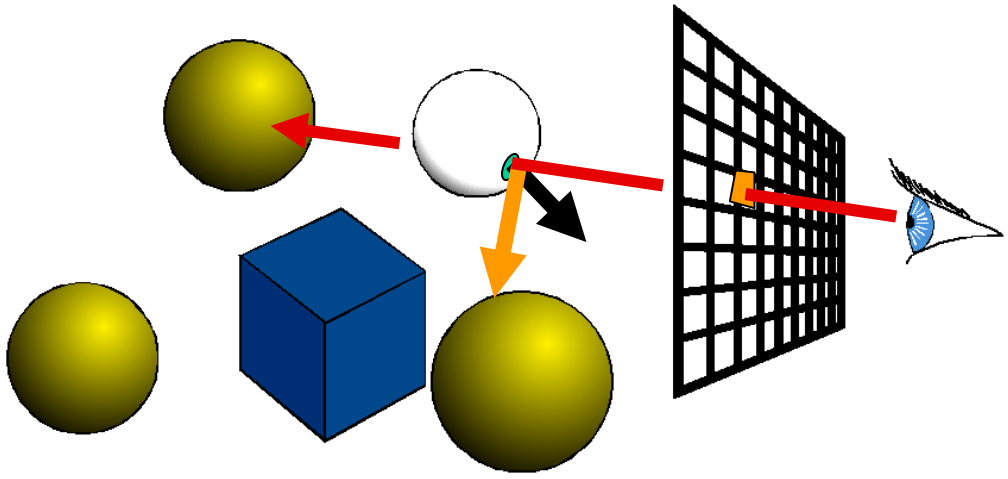
$$R = I - 2(I \cdot N)N$$

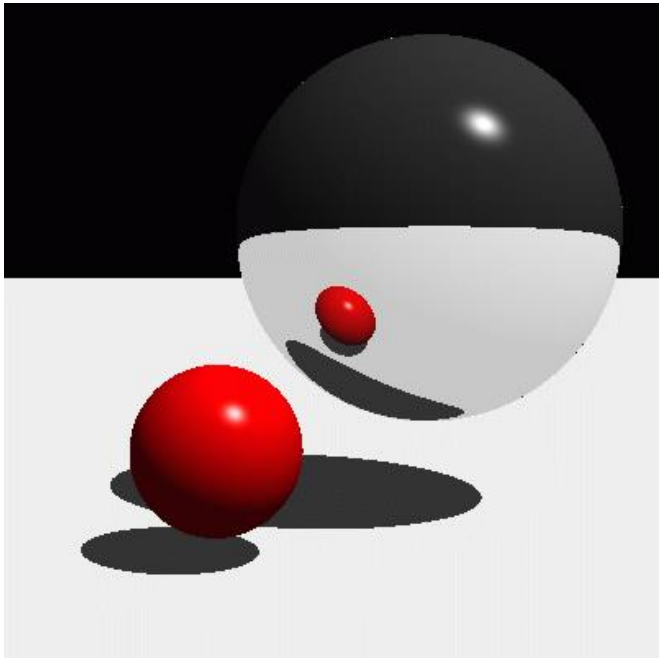
  - 其中 I, N, R 均为单位向量。



光线的反射

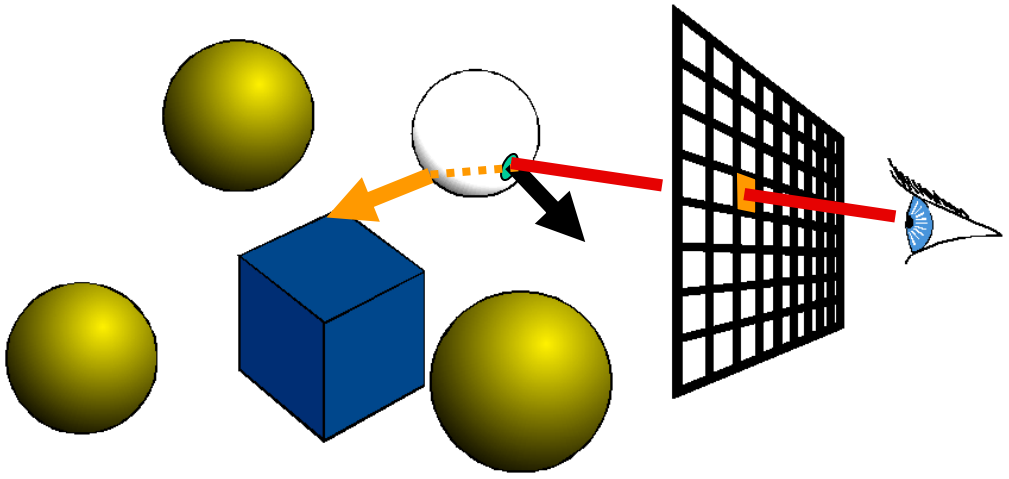
- 如图所示，反射光线的方向和视点方向相对于法向方向呈对称关系：





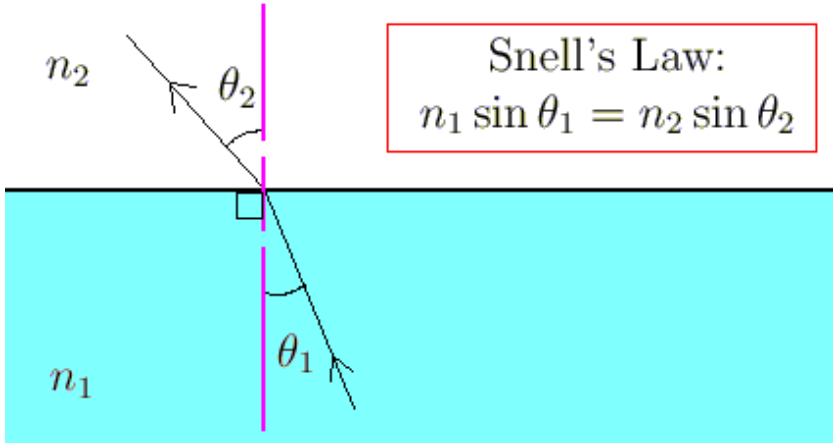
光线的折射

- 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

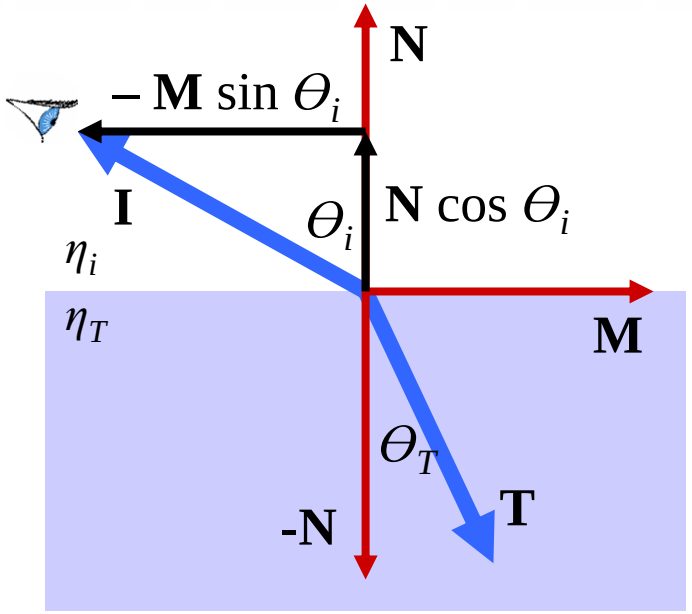
- 由 Snell 定律：

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

$$\eta_i^2 \sin^2\theta_i = \eta_T^2 \sin^2\theta_T$$

$$\eta_i^2(1-\cos^2\theta_i) = \eta_T^2(1-\cos^2\theta_T)$$

$$\cos\theta_T = \sqrt{1-\frac{\eta_i^2(1-\cos^2\theta_i)}{\eta_T^2}}$$

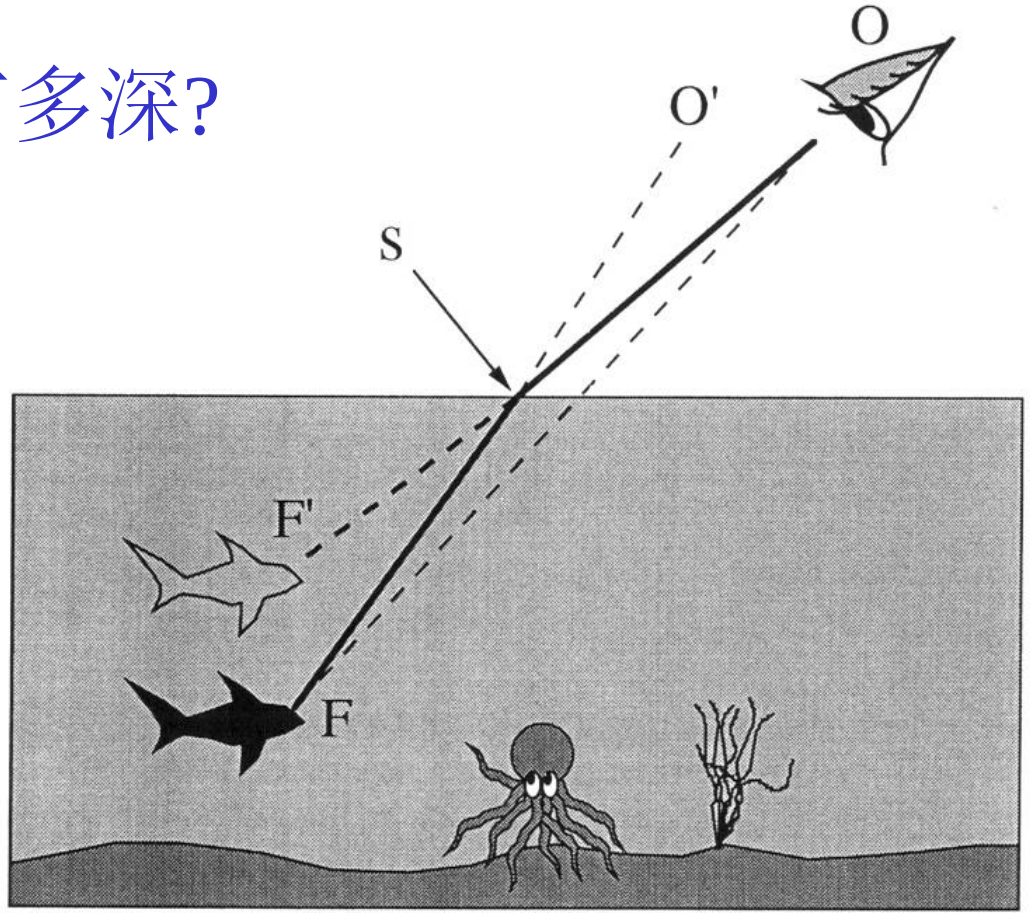


利用 $\cos\theta_T$，可以将折射光的方向写成：

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

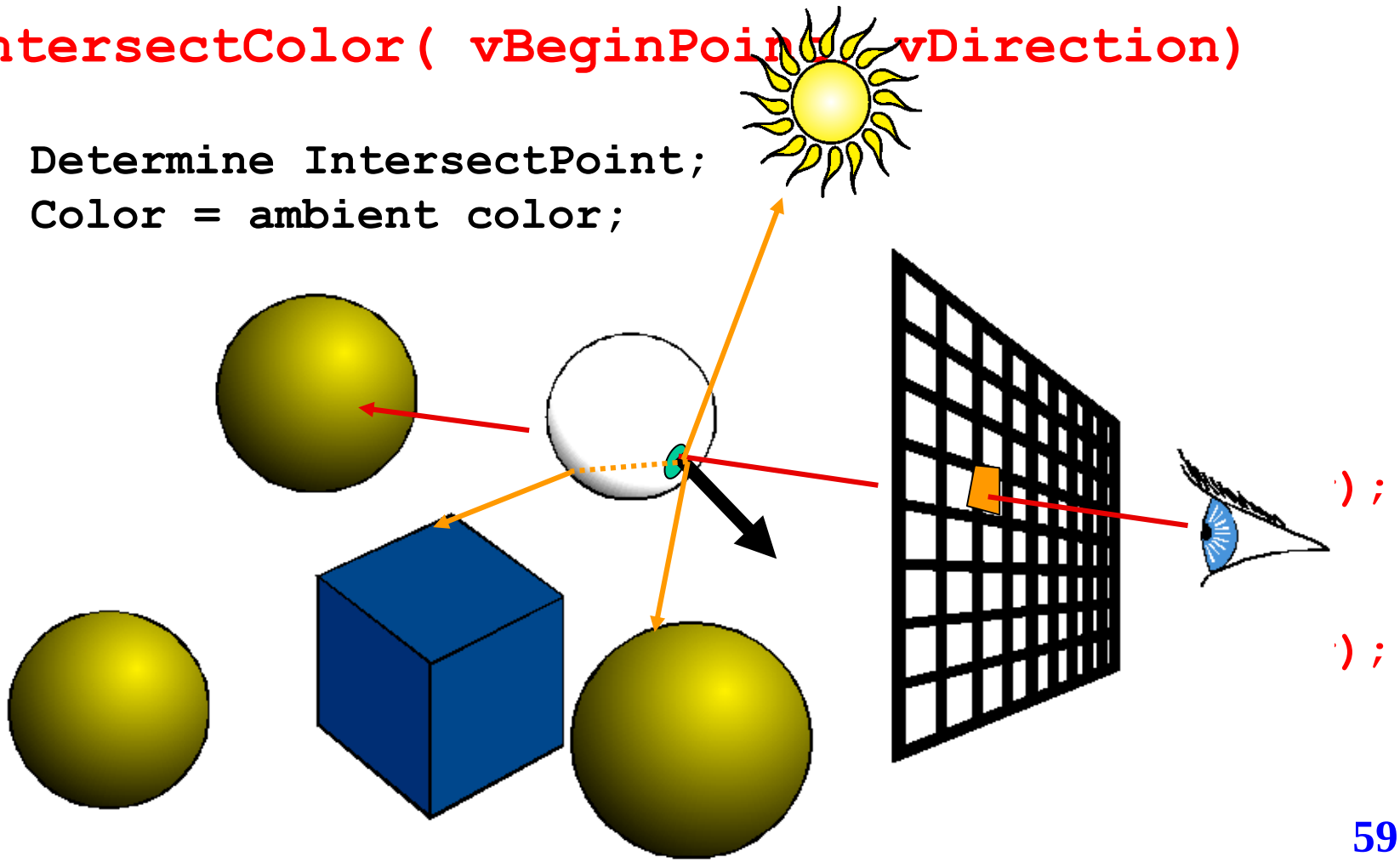
光线的折射

鱼有多深?



递归的光线跟踪算法

```
IntersectColor( vBeginPoint, vDirection)
{
    Determine IntersectPoint;
    Color = ambient color;
```



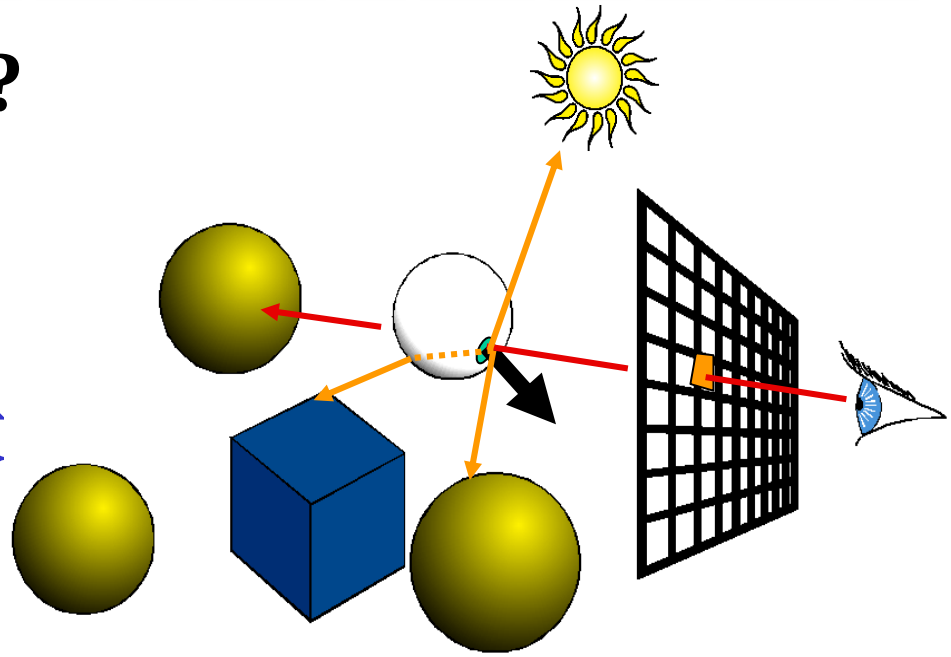
```
}
```

递归的光线跟踪算法

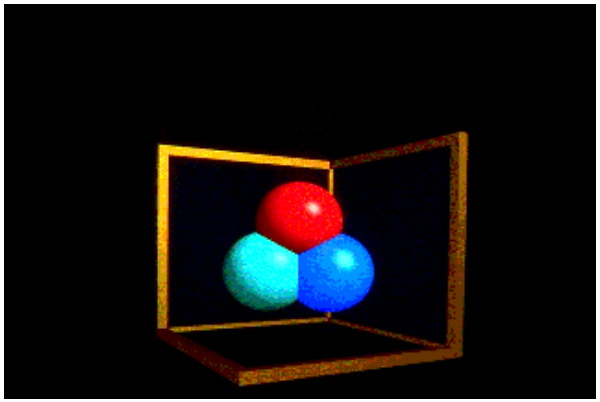
- 什么时候递归结束?

  结束条件可以是:

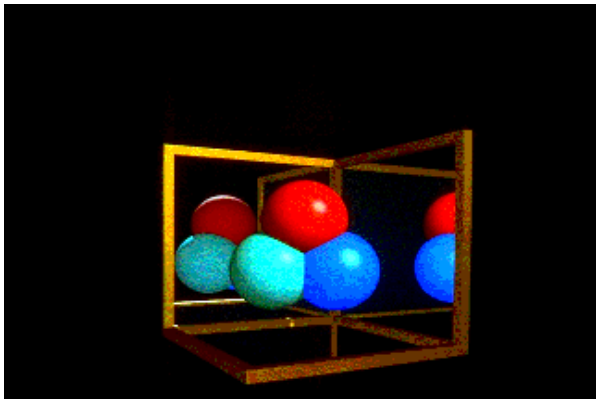
- 递归深度
  - 在光线弹射一定次数后停止
- 光线的贡献
  - 在光线的贡献衰减到足够小时停止



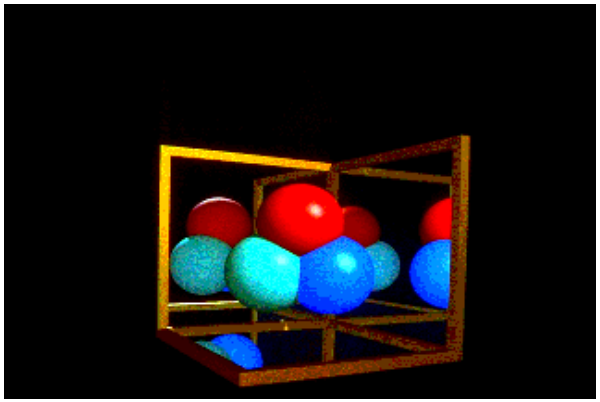
光线跟踪效果示例



0 层递归



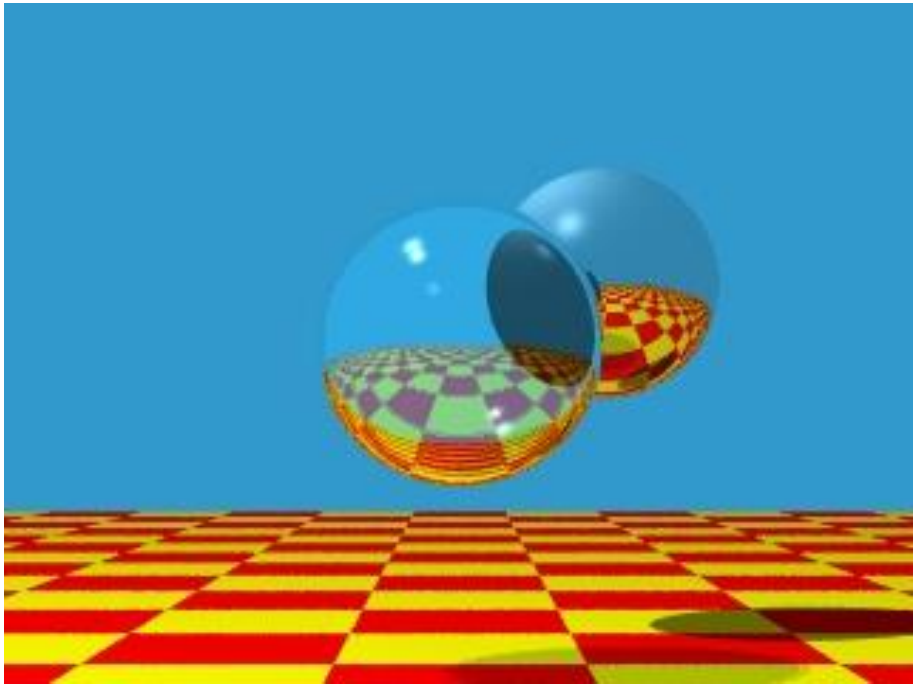
1 层递归



2 层递归

添加纹理 (Texture)

- 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
  - 二维纹理
  - 三维纹理
  - 右图的地面上添加了棋盘格形式的纹理



添加纹理 (Texture)

- 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
  - 关于纹理的更多内容我们会在后面的课程中专门讲述

多选题 1分

设置

光线跟踪算法的递归可以有哪些终止办法?

A 达到指定的最大递归深度

B 当一根光线的贡献太小（因为折射、反射的衰减）

C 光线射到场景之外

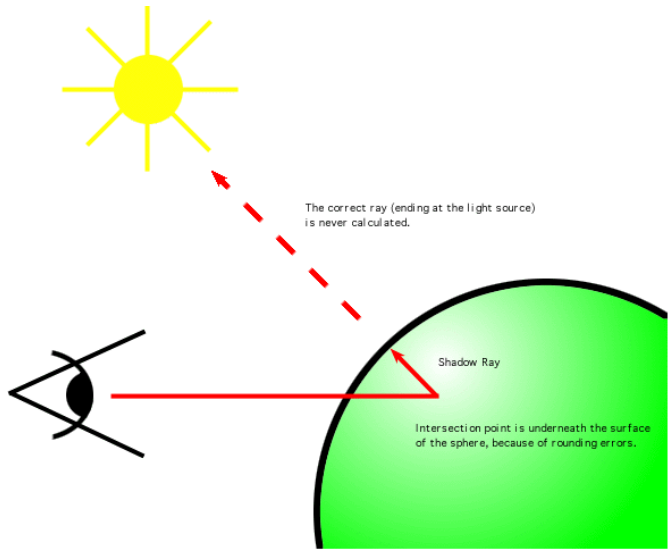
D 当光线方向和视点垂直

提交

光线跟踪的一些思考

- 自遮挡问题
  - 由于浮点数的计算精度问题，计算出的光线与物体表面的交点可能实际上位于物体内部
  - 在这种情况下进行光线跟踪的时候，光线延反射方向会立即碰撞到自身内表面
  - 从而导致绘制结果中出现很多

  原本不应该出现的黑色斑点



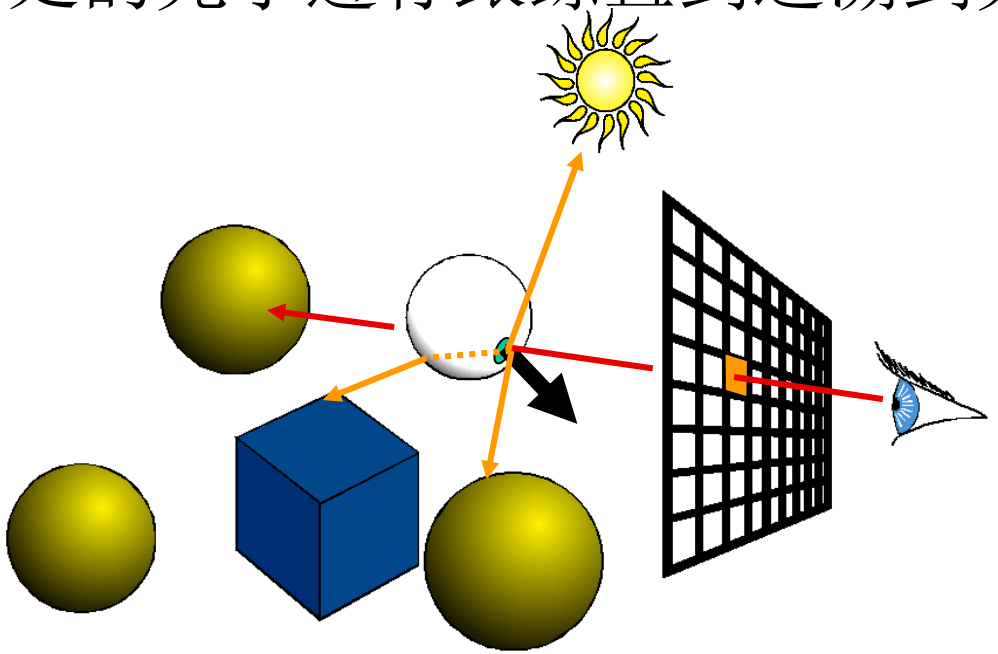
如何避免该问题？

光线跟踪的一些思考

- 衰退情况/特殊情况
  - 当光线与平面、球面相切，或者与多边形相交于其某个顶点时，需要仔细考虑
- 加速算法
  - 使用包围盒 (Bounding Box) 加速
  - 使用层次结构 (Hierarchical Structure) 加速

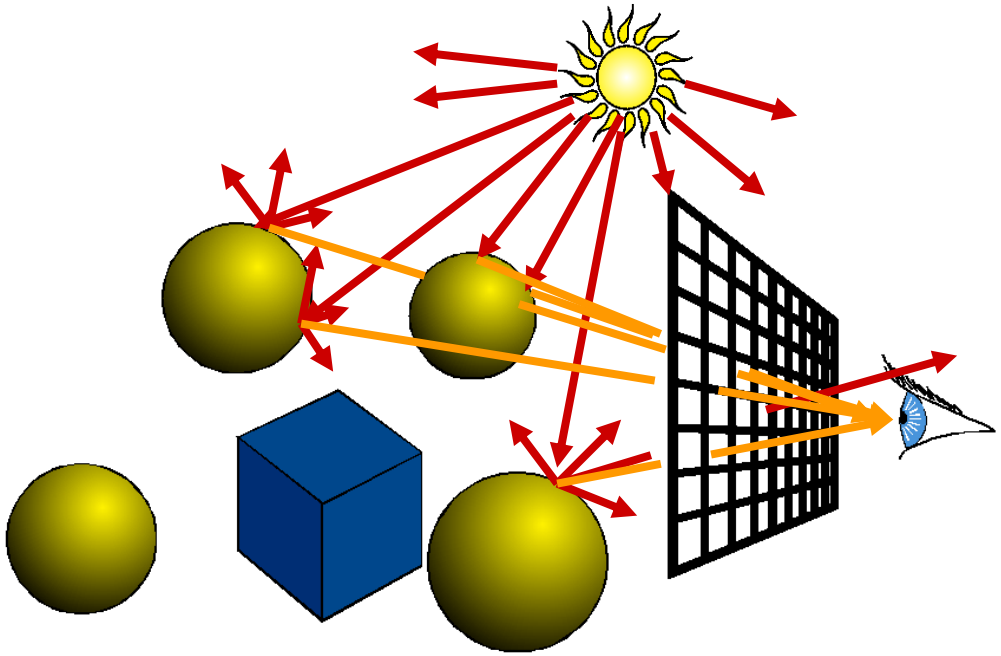
光线跟踪符合物理原理吗？

- 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
- 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



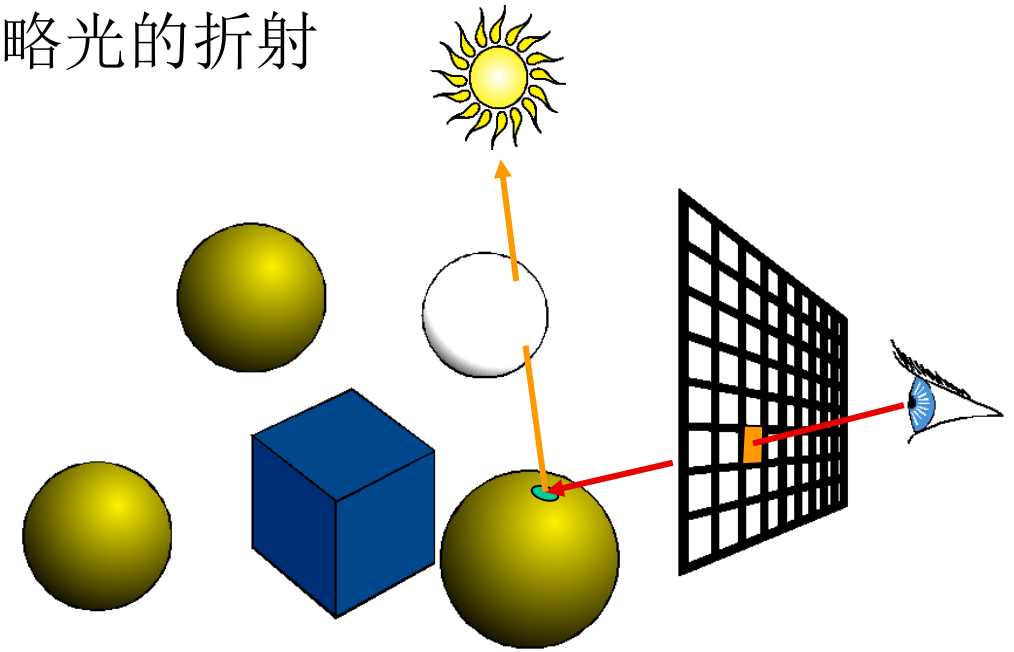
前向 (Forward) 光线跟踪

- 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

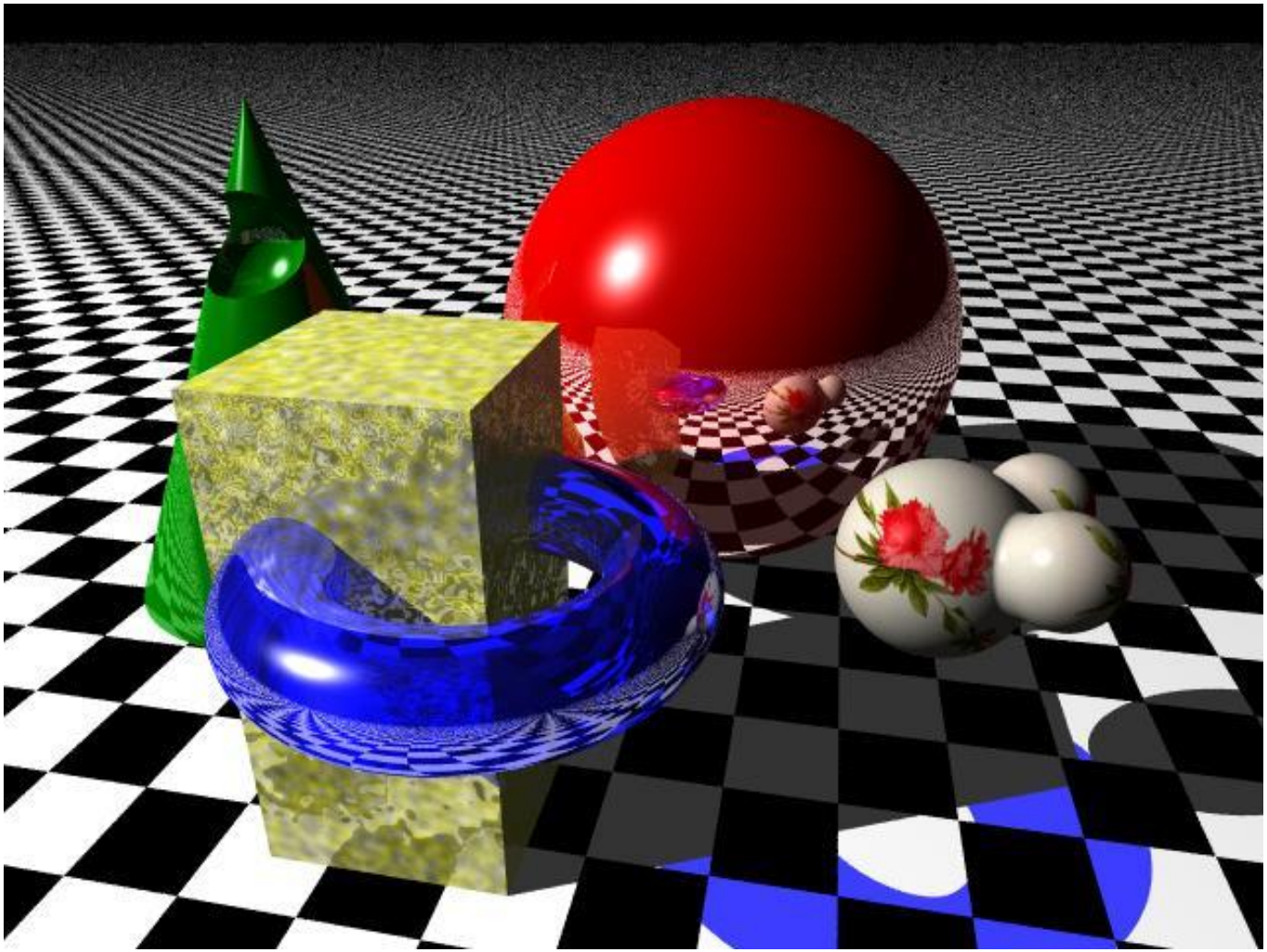


光线跟踪符合物理原理吗？

- 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



光线跟踪效果示例

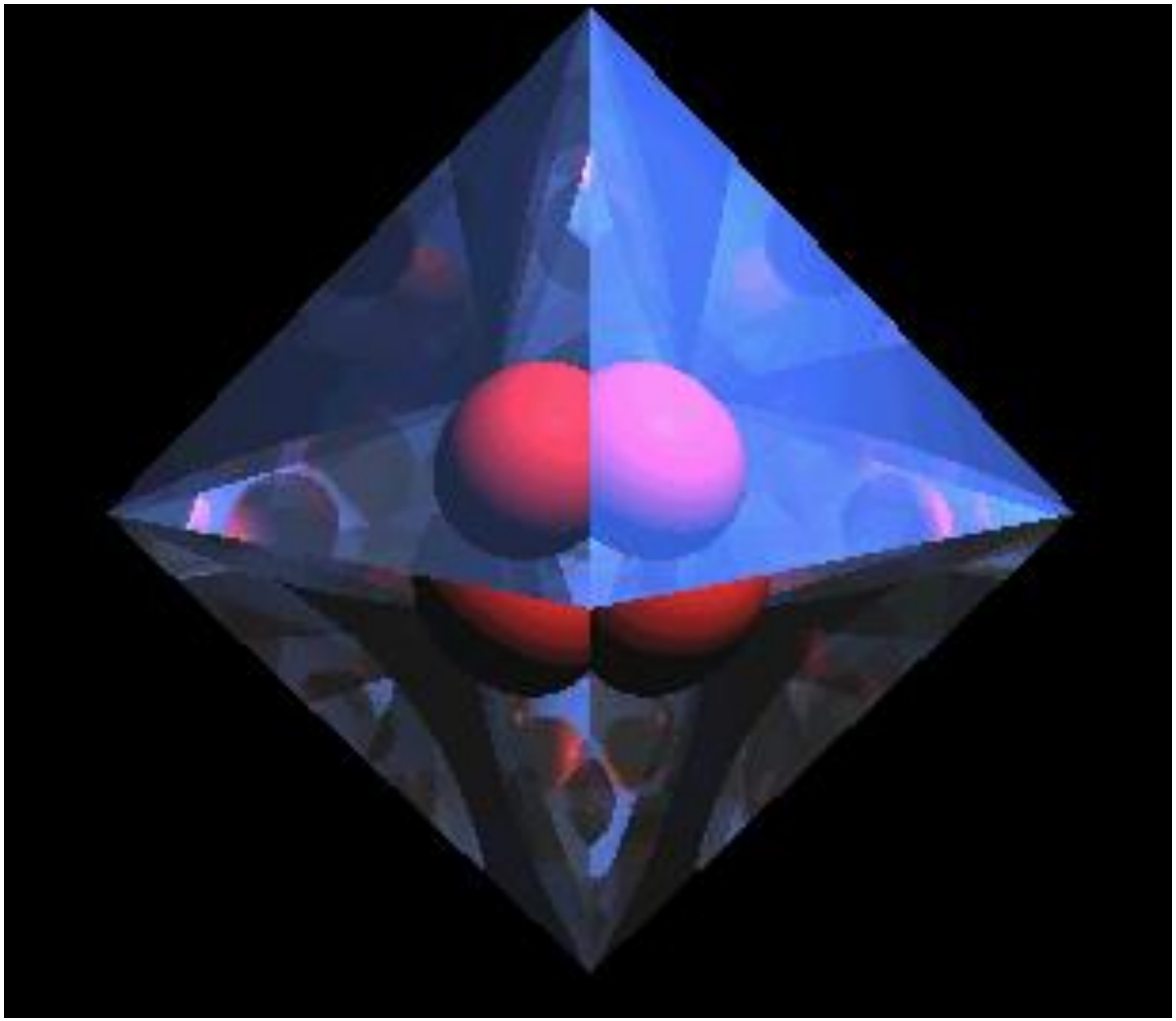


光线跟踪效果示例

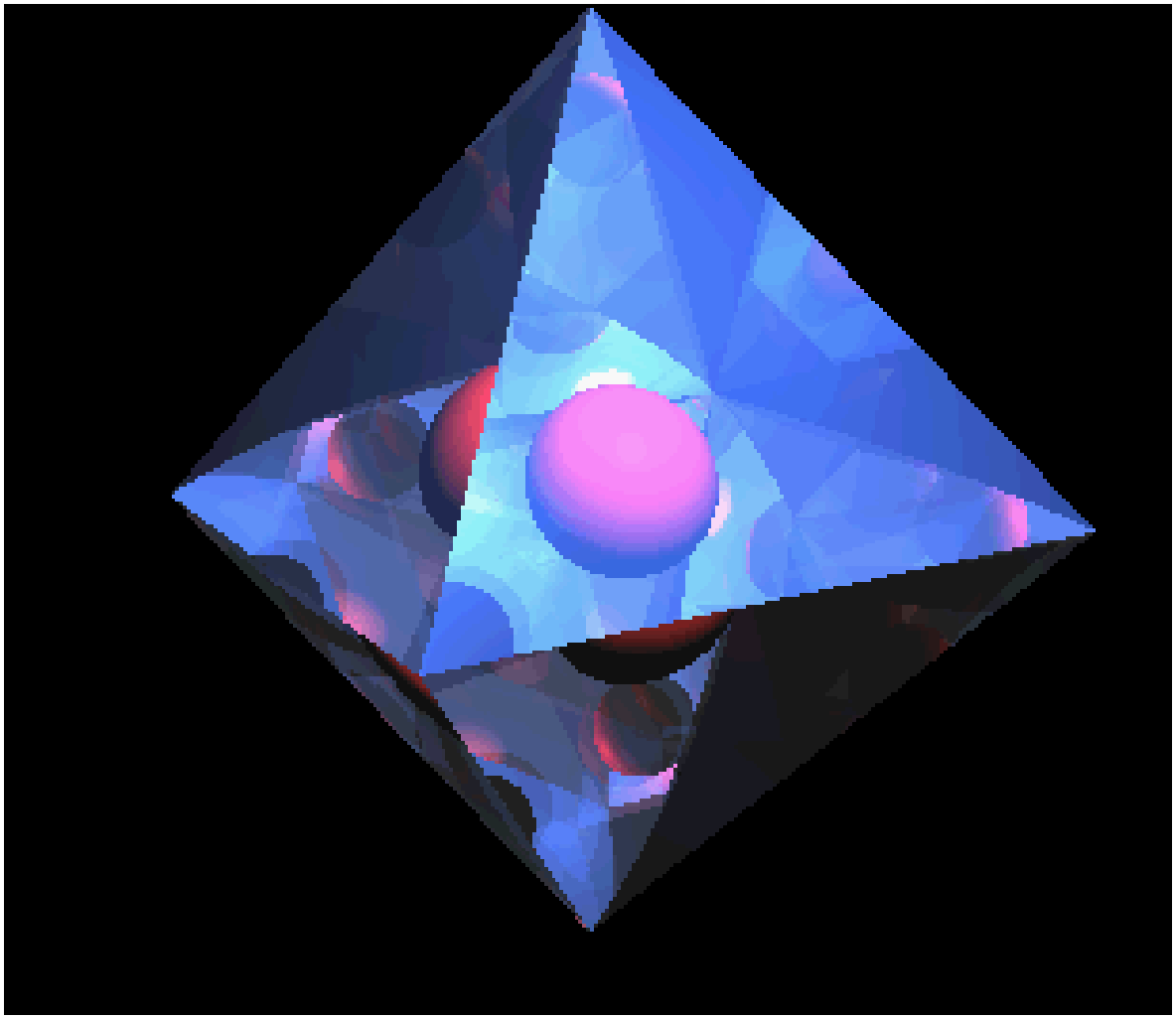
光线跟踪效果示例

光线跟踪效果示例

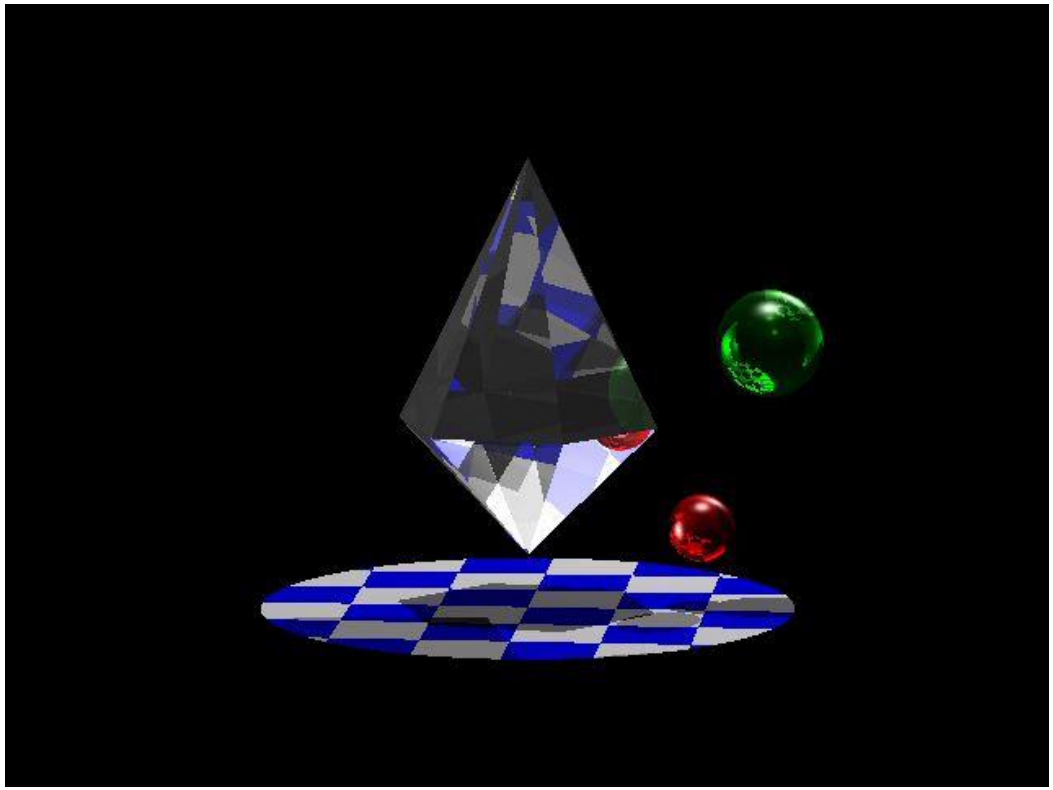
光线跟踪效果示例



更多示例



更多示例



- http://www.siggraph.org/education/materials/HyperGraph/raytrace/rtrace0.htm

光线跟踪效果示例

- 计02年级高岳的大作业：

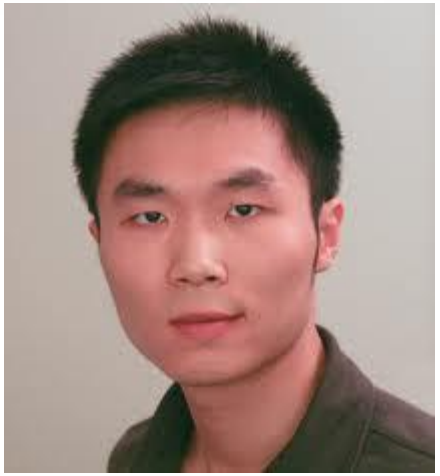
  Launch Application

  高岳同学2011年获博士学位

- 计2年级邓嘉同学的作品：

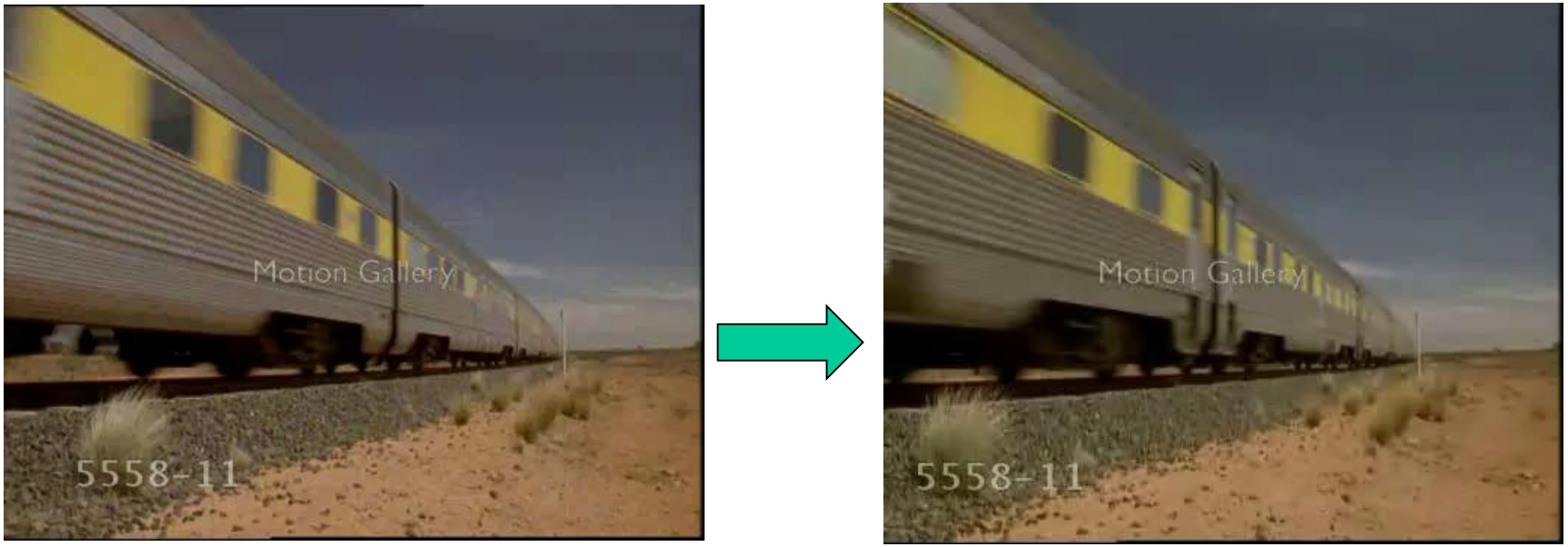
  Play Video

昔日学长：邓嘉

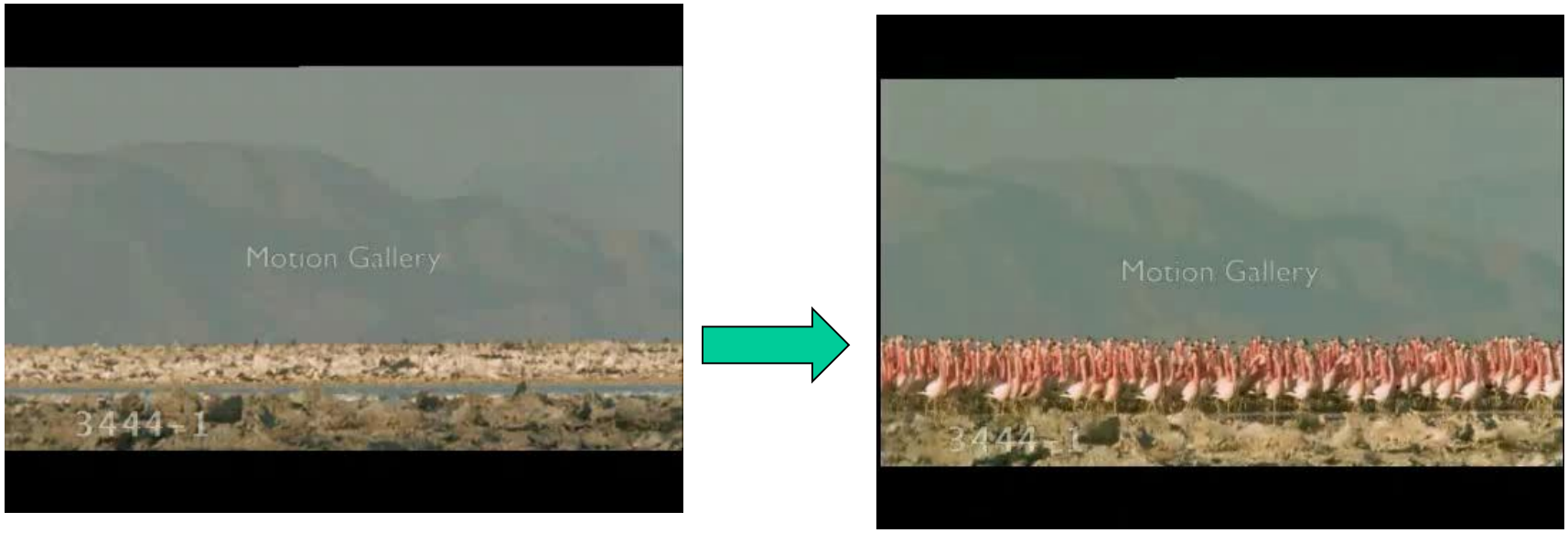


- 邓嘉于 2002 年至 2006 在清华完成本科学业，GPA 位居全系第一
- 2007年起，于Princeton, Stanford 攻读 Ph.D.，2007 年发表了一篇关于Bas-Relief的SIGGRAPH
- 2009年发表 “Imagenet”, CVPR 2009, 已引用51120, http://image-net.org/；论文他引109511次，获马尔奖。
- 2014年，受聘密西根大学计算机系助理教授
- 2018年，受聘普林斯顿大学计算机系助理教授
- 邓嘉在媒体计算课程的故事

视频的重复化 (Video Repetition)



- “火烈鸟来了”



- IEEE Transactions on Multimedia 撤稿。 - 科学精神

蒙特卡洛光线跟踪

- 前面介绍的逆向光线追踪框架很难绘制出非常具有真实感的图像，主要缺陷为：
  - 表面属性单一，不是反射就是折射
  - 没有漫反射效果
- 针对这些问题，我们可以对框架进行改进：
  - 表面的属性可以是混合的，比如30%的成分是镜面反射，20%的成分是折射，50%的成分是漫反射
  - 一条光线在该表面有：30%的概率发生镜面反射，20%的概率发生折射，50%的概率发生漫反射

蒙特卡洛光线跟踪

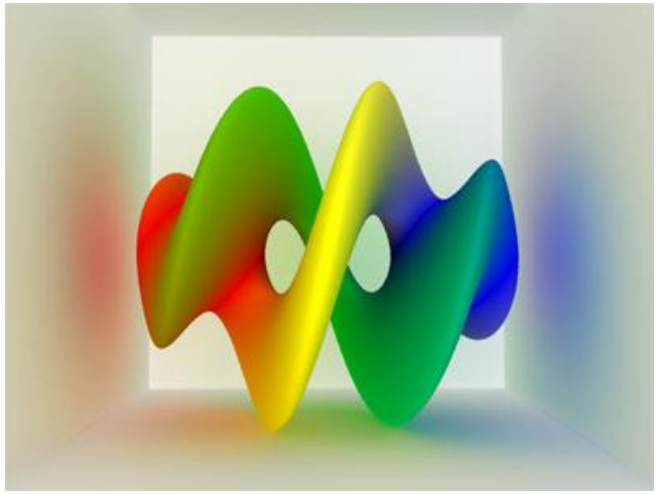
- 基于这些改进，光线跟踪的过程修改为：
  - 从视点出发，经过投影屏幕上的每一个像素向场景发射一根虚拟的光线
  - 当光线与景物相交时按轮盘赌规则决定其反射属性
  - 根据不同的反射属性继续跟踪计算，直到正常结束或者异常结束；如果反射的属性为漫反射，则随机选择一个反射方向进行跟踪
  - 重复前面的过程，把每次渲染出来的结果逐像素叠加混合，直到结果达到预期

蒙特卡洛光线跟踪

- 这种方法的实质是通过大量的随机采样来模拟半球积分，在光照细节上可以产生真实度很高的图像，但是图像整体质量会有严重的走样，而且效率极其低下
- **蒙特卡罗光线追踪**
  - 通过概率理论，把半球积分方程进行近似简化
  - 通过少量相对重要的采样来模拟积分
  - 蒙特卡罗光线追踪已经发展成为了一套比较成熟的真实感绘制框架（也称路径追踪，Path Tracing），广泛应用于动画、游戏等产业中

蒙特卡洛光线跟踪

- 感兴趣的同学可以参考03年的Siggraph course：
  - Monte Carlo Ray Tracing
  - http://www.cs.odu.edu/~yaohang/cs714814/Assg/raytracing.pdf
- 作为一种全局光照模型，Path Tracing天然支持很多效果，最突出的就是color bleeding效果：



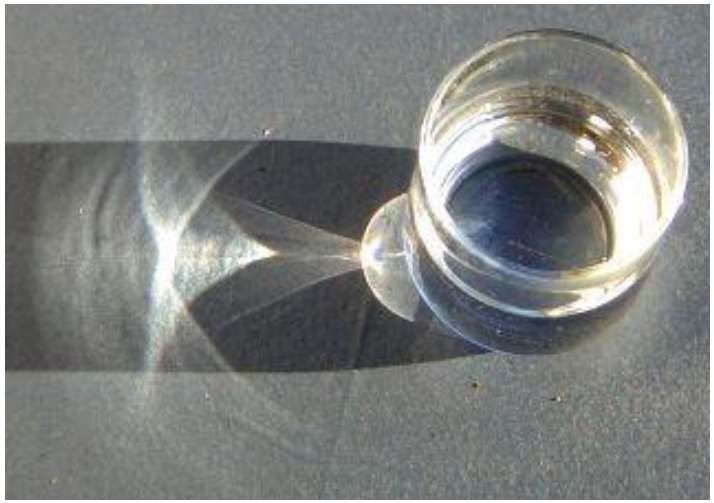
墙壁和地面上的有色光晕

蒙特卡洛光线跟踪

- Path Tracing的缺点：
  - 其性能完全取决于所选取的采样模型，不同的采样模型适用于不同的场景和材质，不合适的采样模型可能导致渲染结果无法收敛
  - 本质上还是对物理模型的近似模拟，渲染结果整体上还是会有一定程度的走样
  - 不太容易渲染出由镜面反射或者规则透射引起的漫反射---- **Caustics**(焦散)效果

Caustics效果

- Caustics是体现场景真实感不可或缺的效果
- Caustics效果一般使用光子映射技术绘制



地面上的光斑

光子映射

- 光子映射（photon mapping），其核心思想是从光源开始以光子为载体追踪光能的传递，把每一个光子的传递中间过程都记录下来，最后按照投影或者逆向光线追踪来收集这些信息，以达到渲染的目的。绘制过程分为下面两个步骤：
  - 通过正向光线跟踪来构建光子图，
  - 通过光子图中的信息来渲染整个场景

- 感兴趣的同学可以参考：
  http://graphics.ucsd.edu/~henrik/papers/photon_map/global_illumination_using_photon_maps_egwr96.pdf

今日人物：Greenberg

- Donald P. Greenberg , Cornell教授
  - 论文198篇，H因子72，60篇100+
  - 1934年生，毕业于Cornell大学
  - 1968年，受聘Cornell 工学院和建筑学院
  - 辐射度方法的发明人，是关于Cornell Box的系列论文的主要作者
  - 1971年完成图形学电影“*Cornell in Perspective*”
  - 1987年获Coons奖 （第三届）
  - 培养的5个学生获ACM SIGGRAPH成就奖
  - 美国工程院士，ACM Fellow 1995， **NSF 首任主任**

谢 谢!