Ray Tracing Outlines

Mainly Reference:“清华胡事民课件”, “虎书”《Fundamentals of Computer Graphics》, Cornell CS4620 ppt
GAMES101 https://www.bilibili.com/video/BV1X7411F744?p=13&vd_source=c93f9d5c2c6ee8043fe0db22203390ee

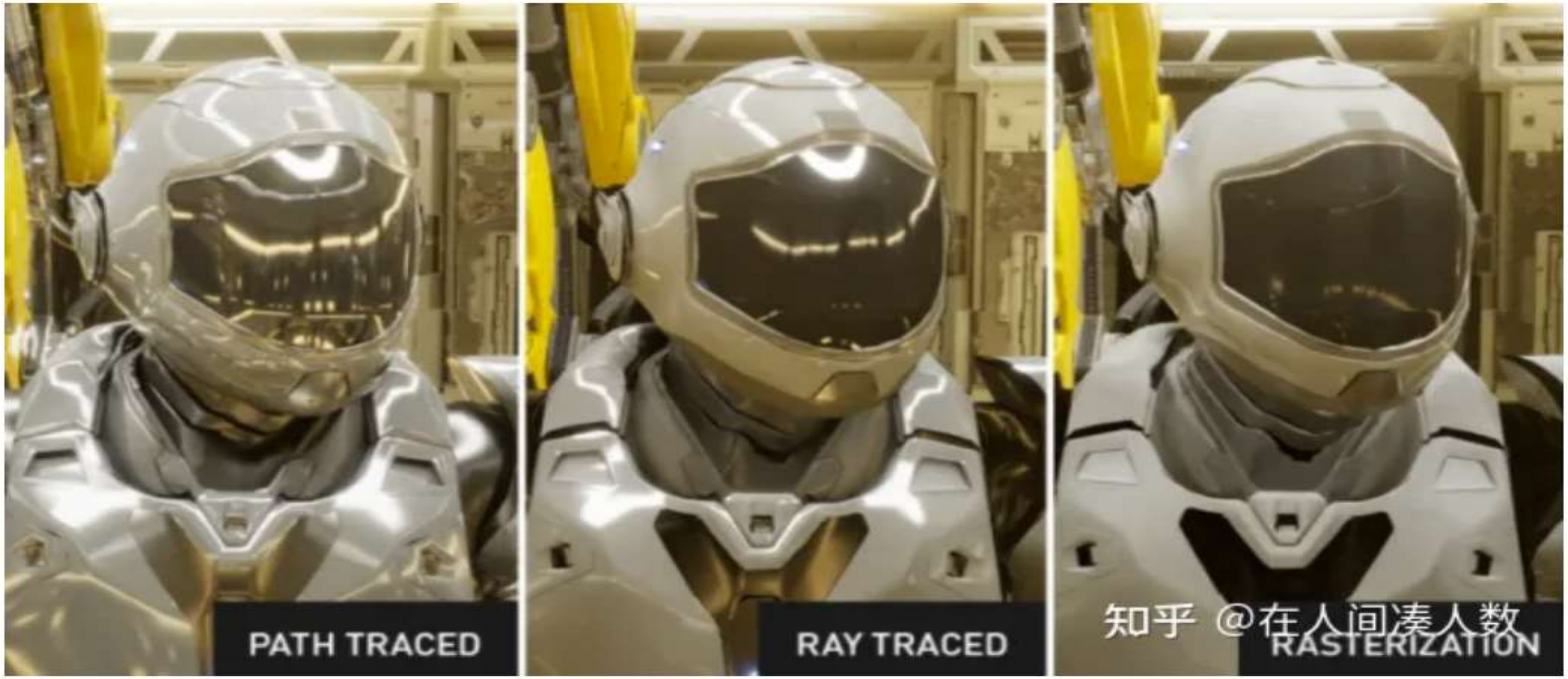
- **Ray Tracing vs. Rasterization**

- **Ray Casting**
- **Whitted-Style（Recursive）**

- **Ray-Surface Intersection**
- **Accelerating Ray-Surface Intersection**

Ray Tracing vs. Rasterization

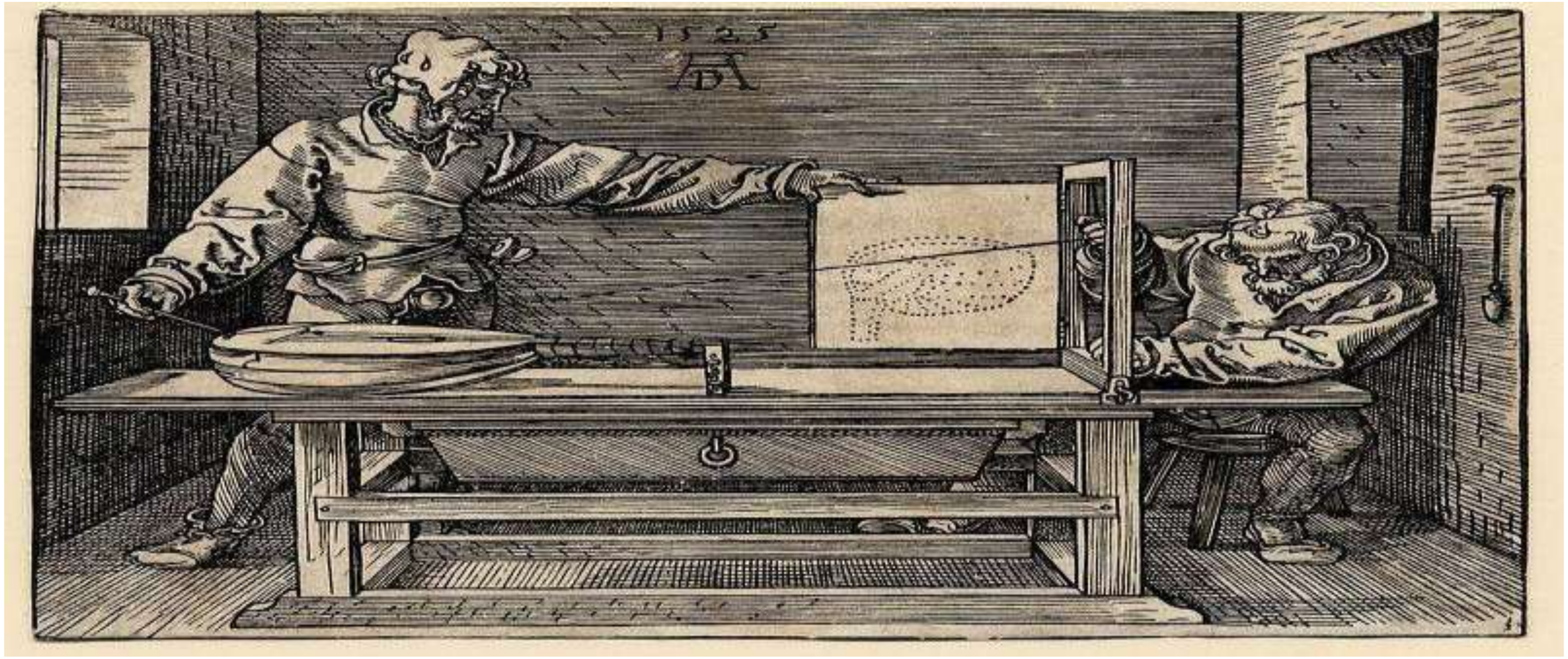
光栅化和光线追踪有什么区别?



Ray Tracing vs. Rasterization(cont.)

渲染方法1：正向采样➔*光栅化渲染算法*

➢从物体表面采样点出发，细线(投影线)连接该采样点和墙上点（投影中心点）的直线交于合页（投影面）处画投影点。

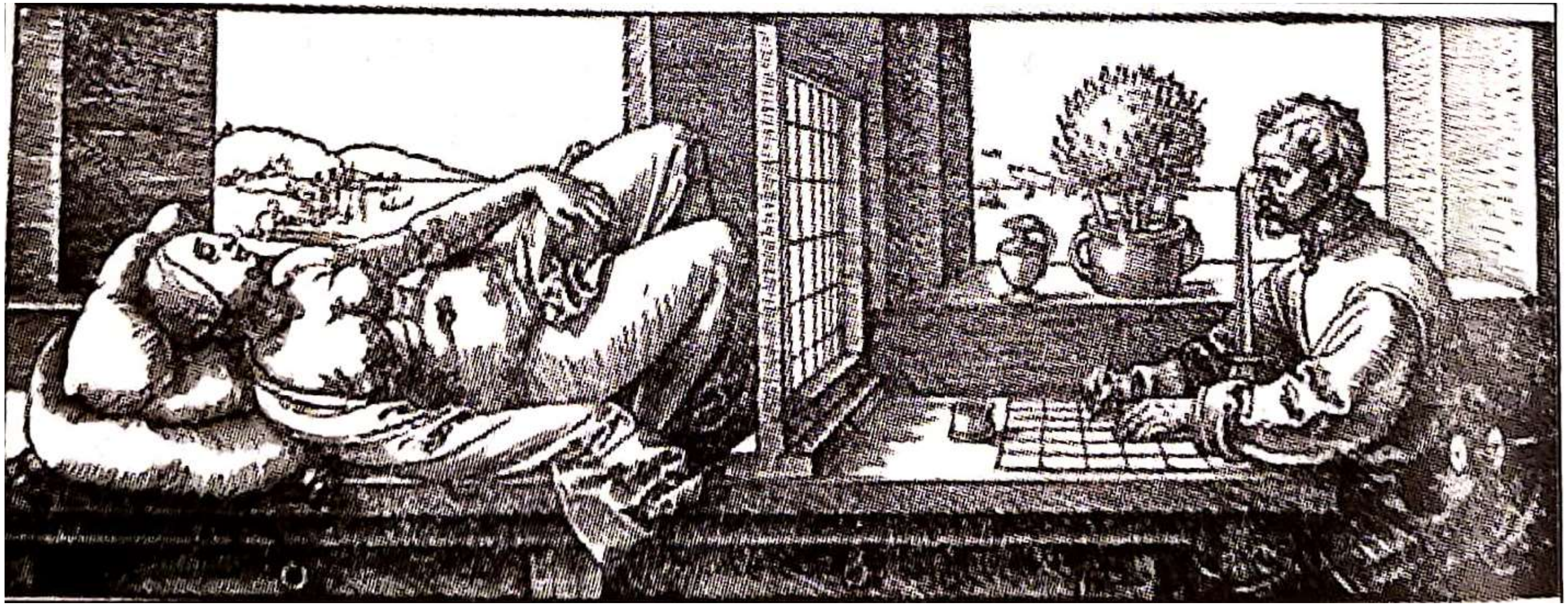


- *木刻画"鲁特琴图" Ref:《计算机图形学原理及实践》约翰.F.休斯等著， 彭群生等译 机械工业出版社*

Ray Tracing vs. Rasterization(cont.)

渲染方法2：逆向采样➔*光线跟踪渲染算法*

从投影中心点(眼睛)出发，视线穿过投影面方格(像素)，交于物体表面，再将“看到”的内容绘制到对应图纸方格(像素)中。



Ref:《计算机图形学原理及实践》约翰.F.休斯等著， 彭群生等译 机械工业出版社

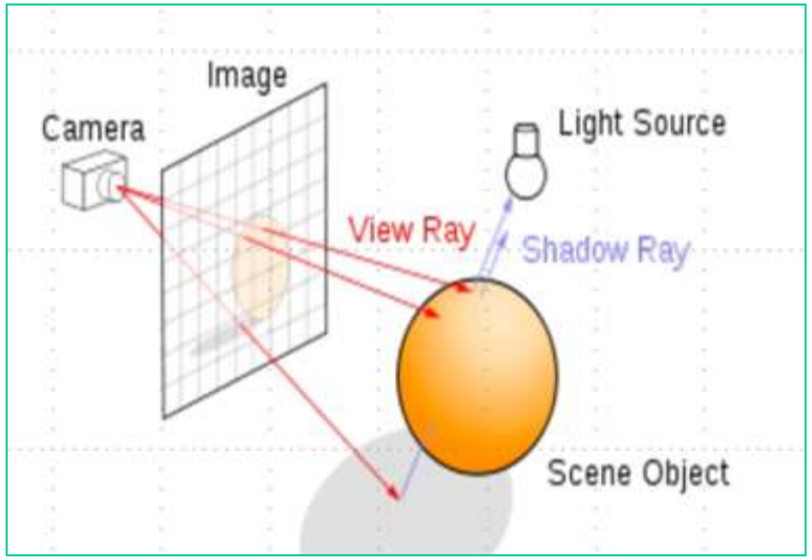
Ray Tracing vs. Rasterization(cont.)

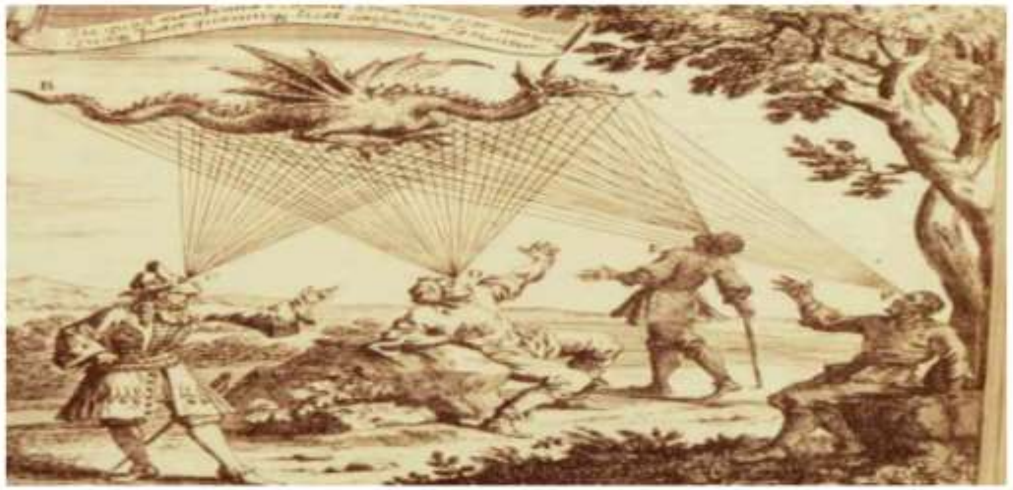
- **Three ideas about light rays（关于光线的三个理想化）**

1. Light travels in straight lines (though this is wrong) 光线沿着直线传播
2. Light rays do not “collide” with each other if they cross (though this is still wrong) 光线交叉时互不碰撞
3. Light rays travel from the light sources to the eye (but the physics is invariant under path reversal - reciprocity). （光线从光源传播到眼睛（但在路径反转-互易作用下，物理学是不变的））

“光线的可逆性”：如果眼睛看见的东西，那东西也能看见你。从光源到眼睛存在一条可逆的光路。

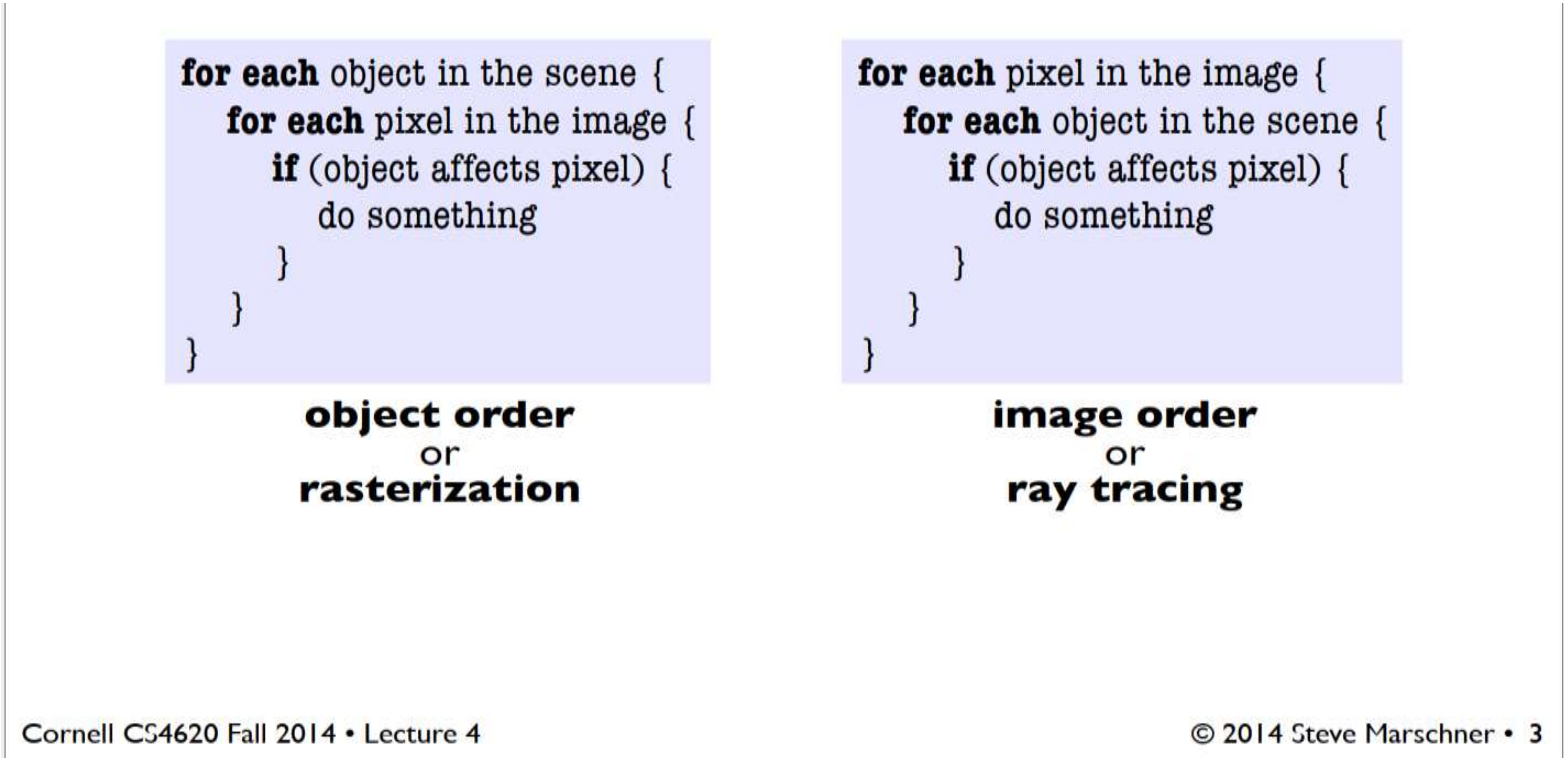
- 光线跟踪算法正是基于“光线可逆性”这一假设，从相机出发跟踪光线，找到可见面并计算光照色~




Eyes send out “feeling rays” into the world

Ray Tracing vs. Rasterization(cont.)

- Two approaches to rendering：



```
for each object in the scene {
    for each pixel in the image {
        if (object affects pixel) {
            do something
        }
    }
}
```

object order
or
rasterization

```
for each pixel in the image {
    for each object in the scene {
        if (object affects pixel) {
            do something
        }
    }
}
```

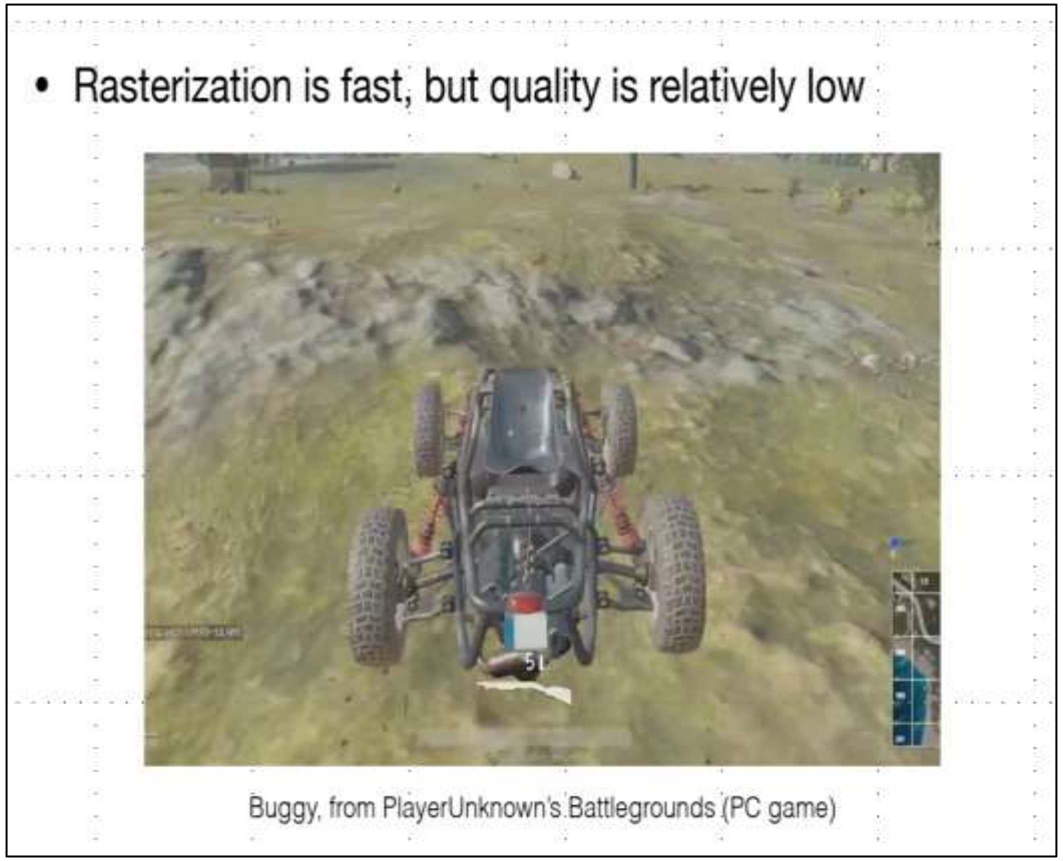
image order
or
ray tracing

Cornell CS4620 Fall 2014 • Lecture 4

© 2014 Steve Marschner • 3

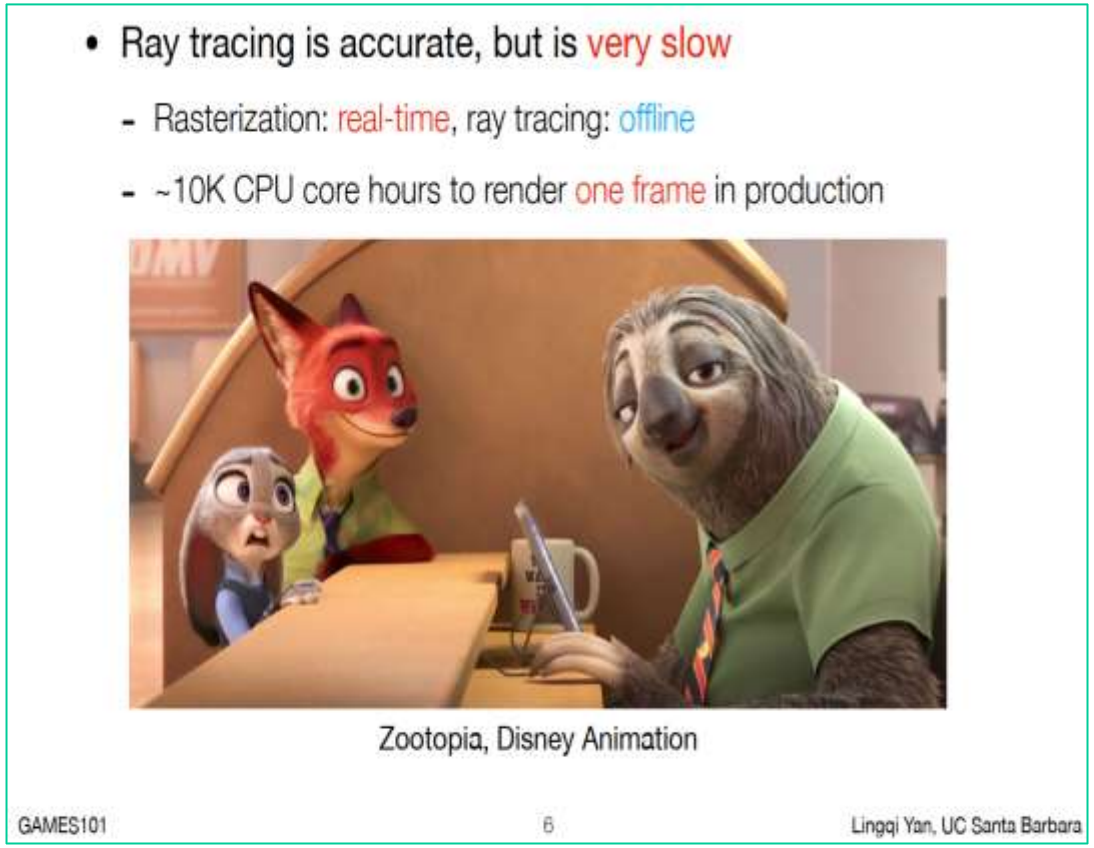
Ray Tracing vs. Rasterization(cont.)

Rasterization versus **Ray Tracing**



- Rasterization is fast, but quality is relatively low

Buggy, from PlayerUnknown's Battlegrounds (PC game)



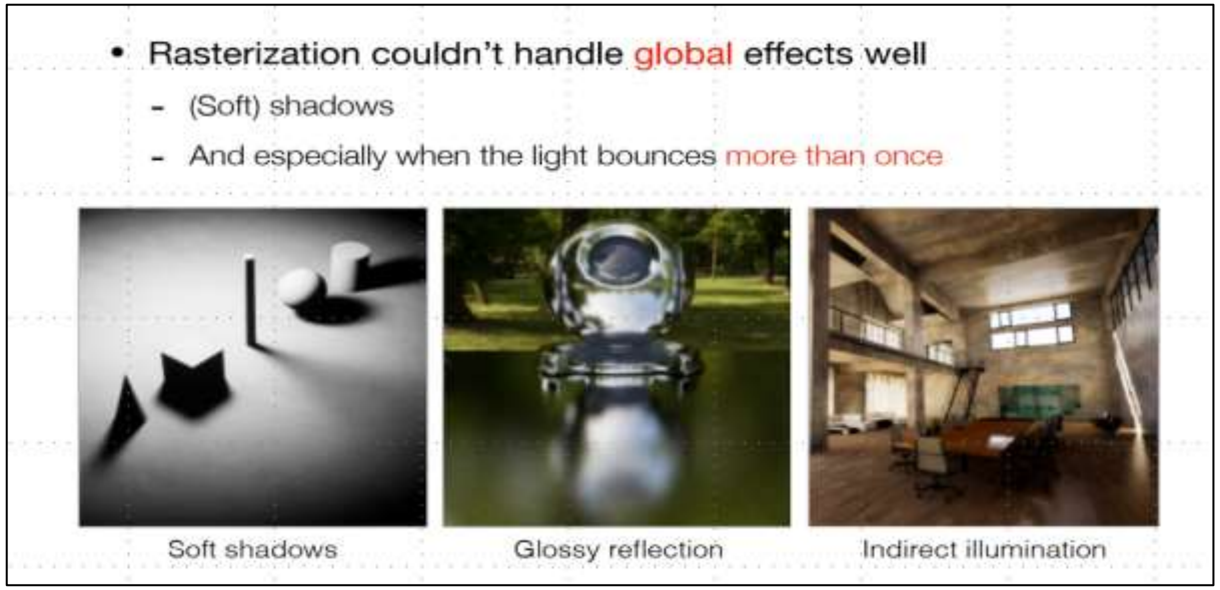
- Ray tracing is accurate, but is very slow
  - Rasterization: real-time, ray tracing: offline
  - ~10K CPU core hours to render one frame in production

Zootopia, Disney Animation

GAMES101 6 Lingqi Yan, UC Santa Barbara

Ray Tracing vs. Rasterization(cont.)

- **Ray Tracing characteristics:**
  - **Gold standard in Animations/Movies(Offline Applications)**
  - Shoot rays from the camera through each pixel (逆向跟踪)
  - Ray Tracing is accurate , but is verry slow（准确但是慢）
  - Calculate intersection and shading：**Continue** to bounce the rays till they hit light sources(计算交点和着色：连续弹射直到击中光源）
  - **可较好实现全局效果global effects（如 软阴影，光面反射glossy reflection，间接光照效果）**



- Rasterization couldn't handle global effects well
  - (Soft) shadows
  - And especially when the light bounces more than once

Soft shadows　　Glossy reflection　　Indirect illumination

Ray Tracing Outlines

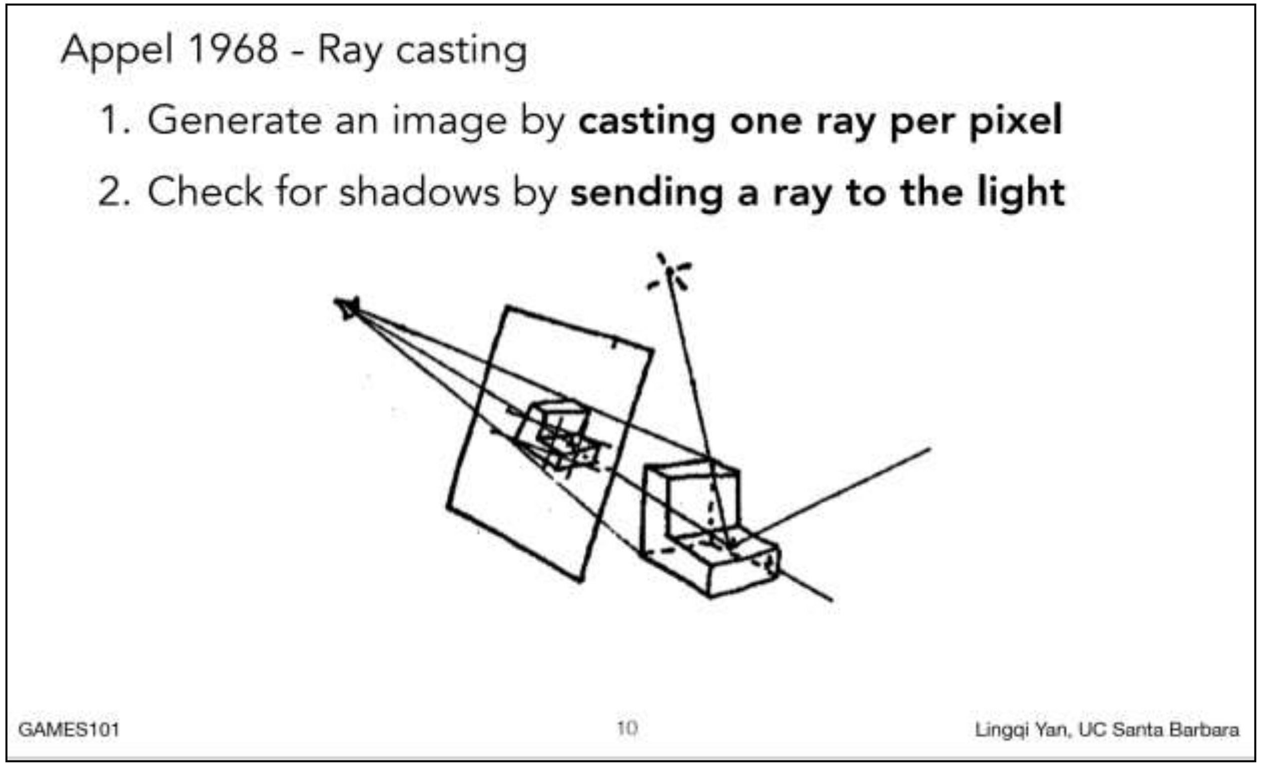
- Ray Tracing vs. Rasterization

- Ray Casting
- Whitted-Style（Recursive）

- Ray-Surface Intersection
- Accelerating Ray-Surface Intersection

Ray Casting（光线投射）

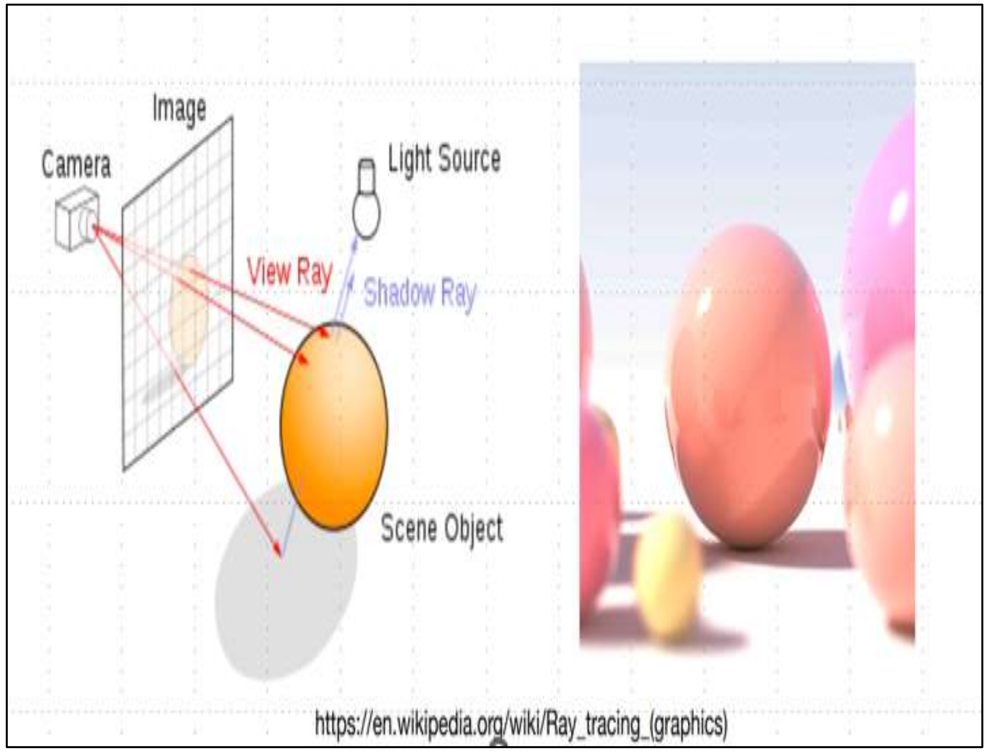
➢Arthur Appel 于 1968 年首次提出用于渲染的光线投射算法



Appel 1968 - Ray casting

1. Generate an image by **casting one ray per pixel**
2. Check for shadows by **sending a ray to the light**

GAMES101 10 Lingqi Yan, UC Santa Barbara



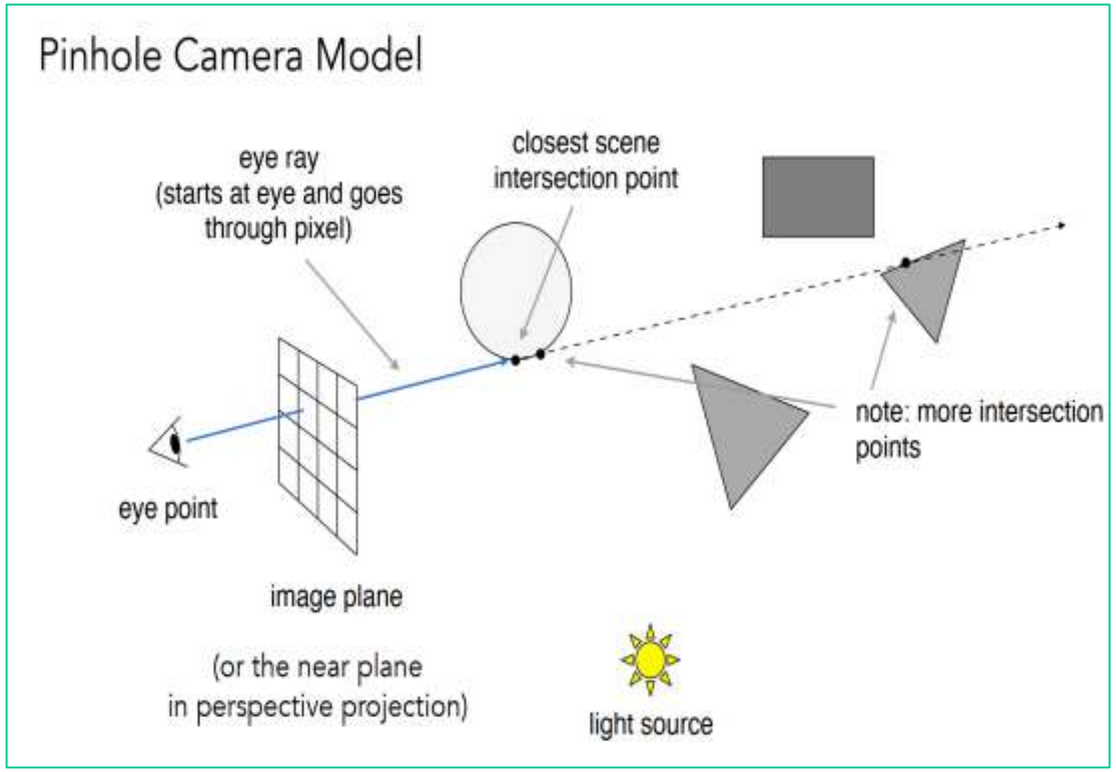
Ray Casting光线投射(cont.)

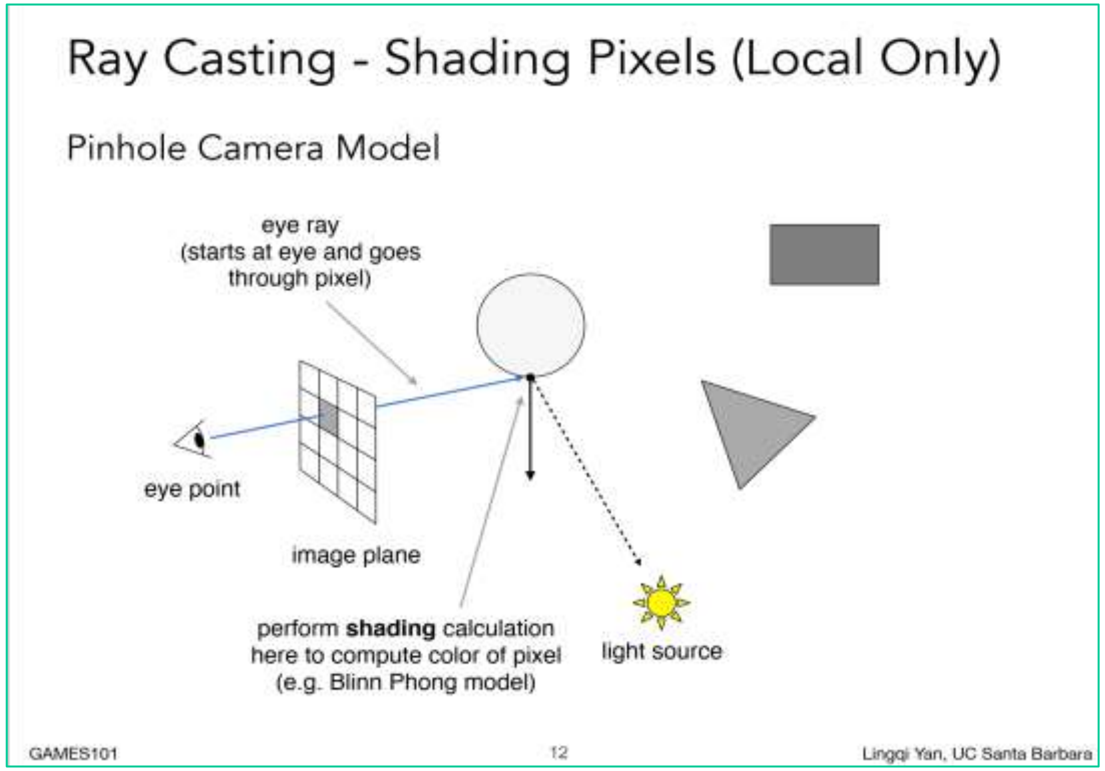
➢**Ray Casting算法基本步骤**:

Step1: Generating Eye Rays Per Pixel，从相机出发为每像素发射一条光线（View Ray/ Eye Ray）

Step2: Find Intersections of Scene Objects 光线与物体表面求交(Ray-Surface Intersection)，找到离视点最近交点

Step3: Perform Shading Calculation at the First intersection 从最近交点弹射shadow ray到光源，按光照模型计算得交点颜色，座位该像素颜色


Pinhole Camera Model
eye ray
(starts at eye and goes through pixel)
closest scene intersection point
note: more intersection points
eye point
image plane
(or the near plane in perspective projection)
light source


Ray Casting - Shading Pixels (Local Only)
Pinhole Camera Model
eye ray
(starts at eye and goes through pixel)
eye point
image plane
perform shading calculation here to compute color of pixel (e.g. Blinn Phong model)
light source
GAMES101
12
Lingqi Yan, UC Santa Barbara

Ray Casting光线投射(cont.)

➢Ray Casting算法基本步骤:(cont.)

Step1: Generating Eye Rays Per Pixel，从相机视点出发，为每像素发射光线（View Ray/ Eye Ray）

➢**Ray Equation: r(t)=o + t*d （o是发出光线的点位置，d是单位方向向量）:**

➢**Ray Equation 是一条参数化表达的射线方程**

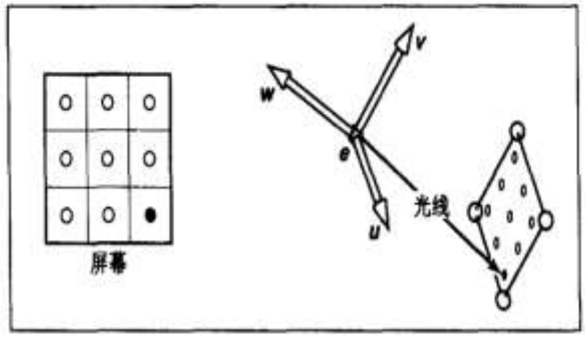
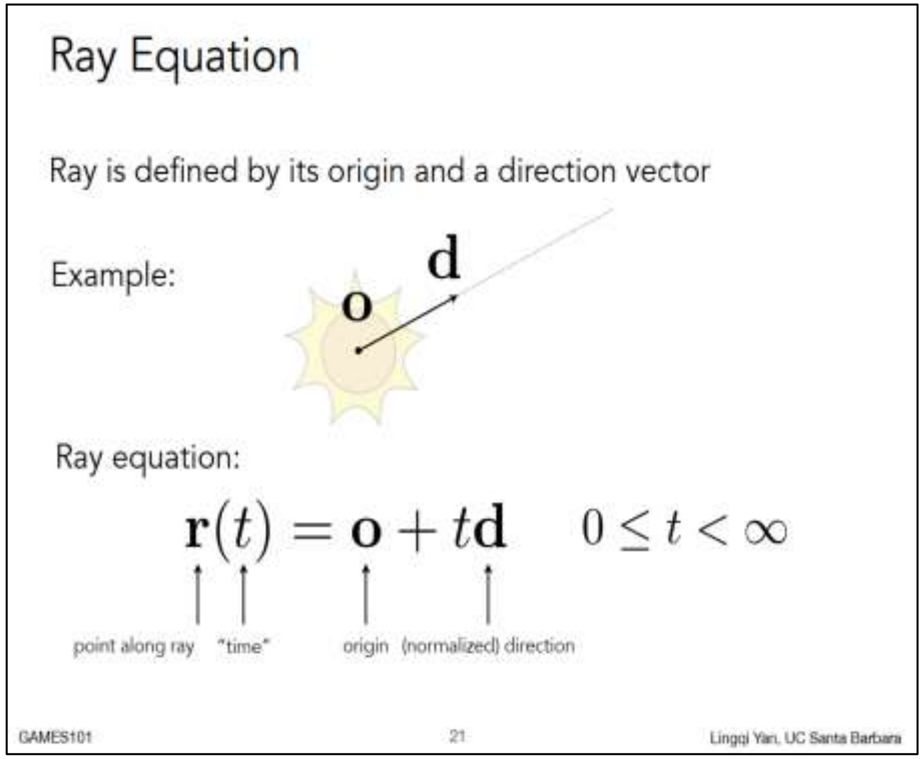


图10-2 屏幕上的采样点映射到三维窗口中一个类似的阵列中。观察光线发射到每一个这样的位置



Ray Equation

Ray is defined by its origin and a direction vector

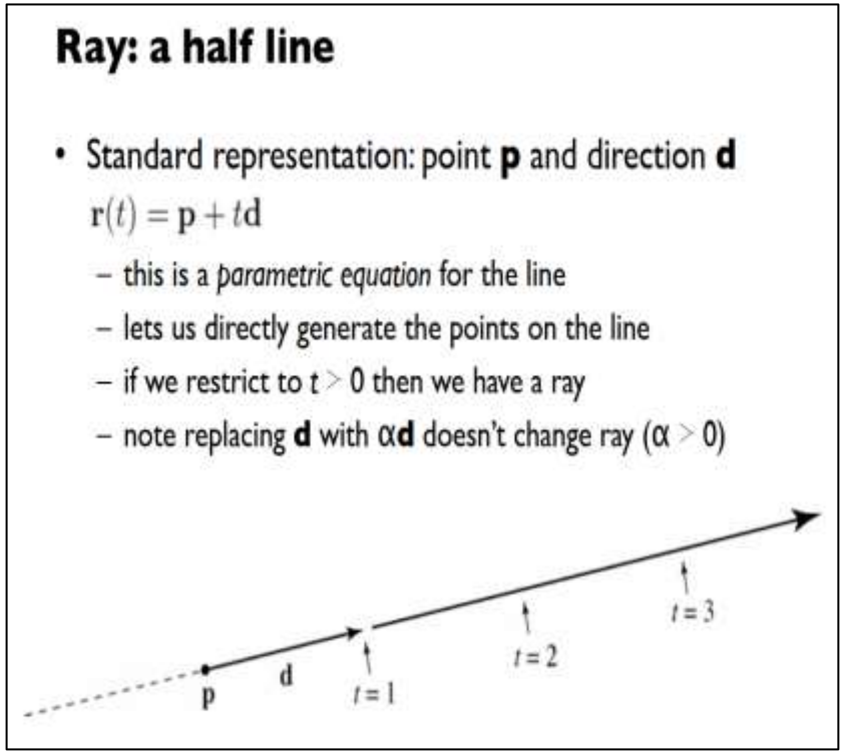
Example:

Ray equation:

$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d} \quad 0 \le t < \infty$$

point along ray "time" origin (normalized) direction

GAMES101 21 Lingqi Yan, UC Santa Barbara



Ray: a half line

- Standard representation: point **p** and direction **d**

  $\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$

  - this is a *parametric equation* for the line
  - lets us directly generate the points on the line
  - if we restrict to $t > 0$ then we have a ray
  - note replacing **d** with α**d** doesn't change ray ($\alpha > 0$)

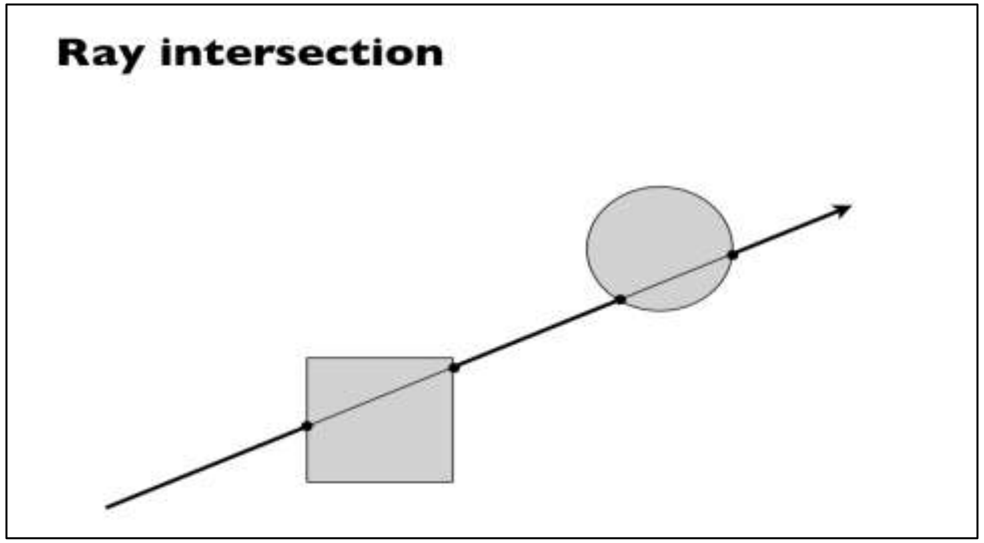
Ray Casting光线投射(cont.)

- **Ray Casting算法基本步骤:(cont.)**

  Step1: Generating Eye Rays Per Pixel, 为每像素发射光线（View Ray/ Eye Ray）

  Step2: Find Intersections of Scene Objects 光线与物体表面求交(Ray-Surface Intersection)，找到离视点最近交点

  - **描述:将光线r(t)=O + t*d (t in [0, ∞]) 带入场景中的物体表示方程f(x,y,z)=0或f(r(t))=0, 看是否有根, 若有则找到参数t值最小但是大于0的参数t1,再带入光线r(t1)得到最近交点的坐标值.**
  - 如何求交?(后面再介绍)
    - 光线与"隐式表示的表面" 求交
    - 光线与"显示表示的表面"求交



Ray Casting光线投射(cont.)

- **Ray Casting算法设计步骤:**

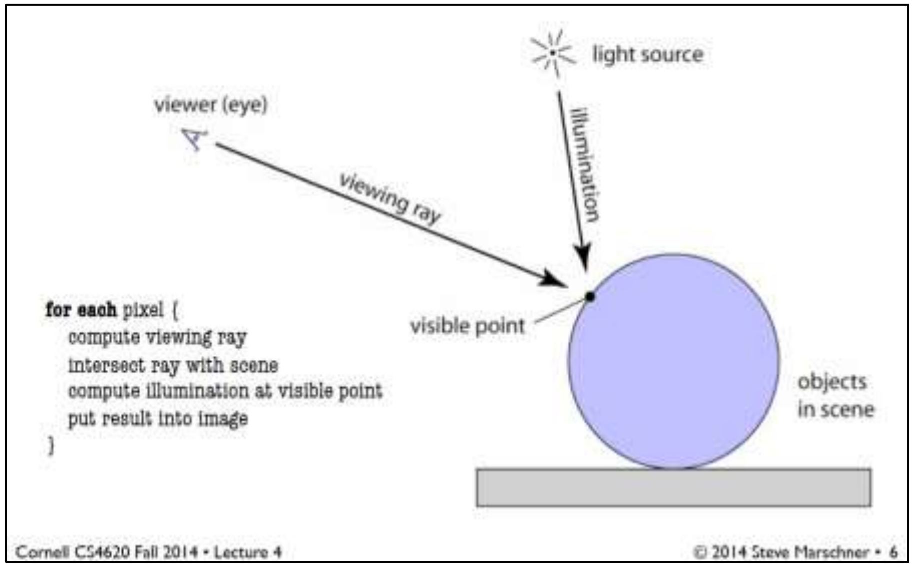
  Step1: Generating Eye Rays Per Pixel, 为每像素发射光线（View Ray/ Eye Ray）

  Step2: Find Intersections of Scene Objects 光线与物体表面求交(Ray-Surface Intersection)

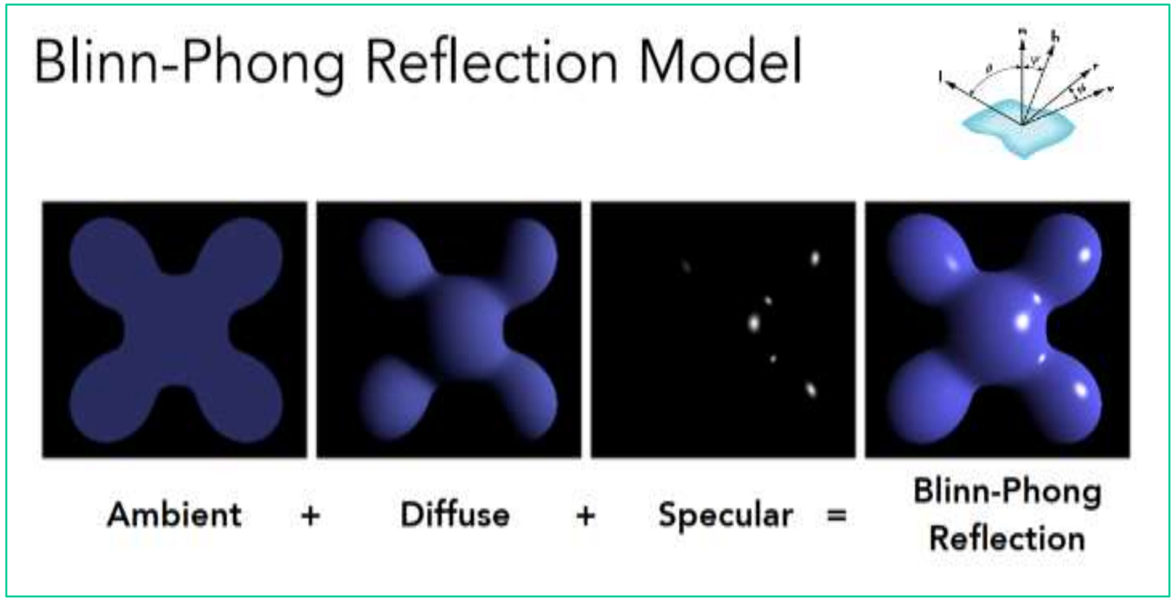
  Step3: Perform Shading Calculation at the First intersection 用Blinn-Phong光照模型得“最近”交点

  - 用简单光照模型(如Blinn-Phong)计算交点颜色

$$I = f(d)\left[k_d L_d \max(\vec{L} \bullet \vec{N}, 0) + k_s L_s \max((\vec{N} \bullet \vec{H})^e, 0)\right] + k_a L_a$$

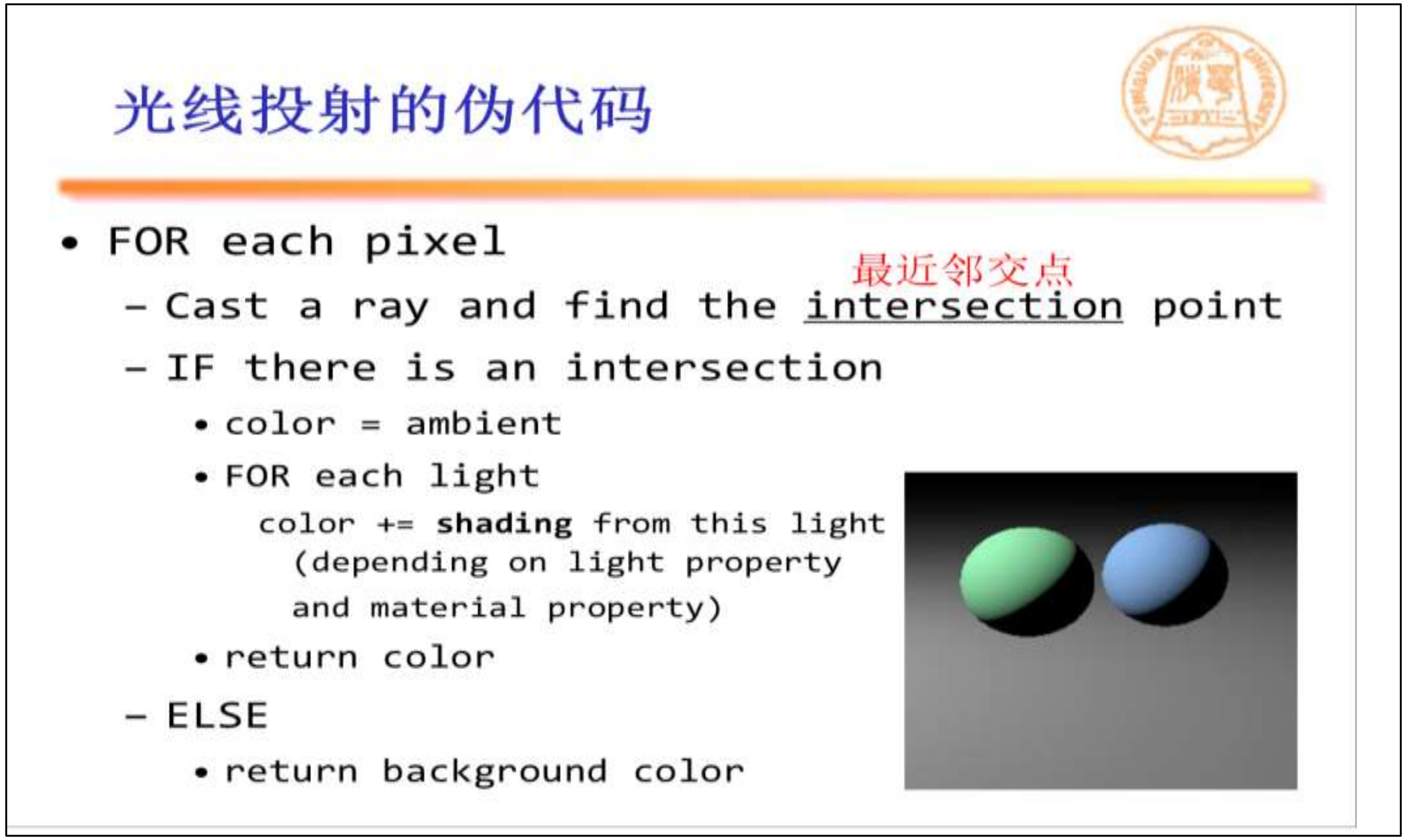


```
for each pixel {
    compute viewing ray
    intersect ray with scene
    compute illumination at visible point
    put result into image
}
```



Ray Casting光线投射(cont.)

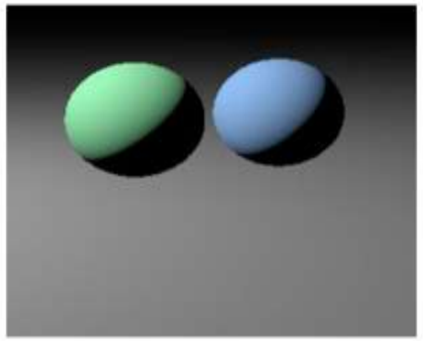
➢**Ray Casting算法(无阴影)**



光线投射的伪代码

最近邻交点

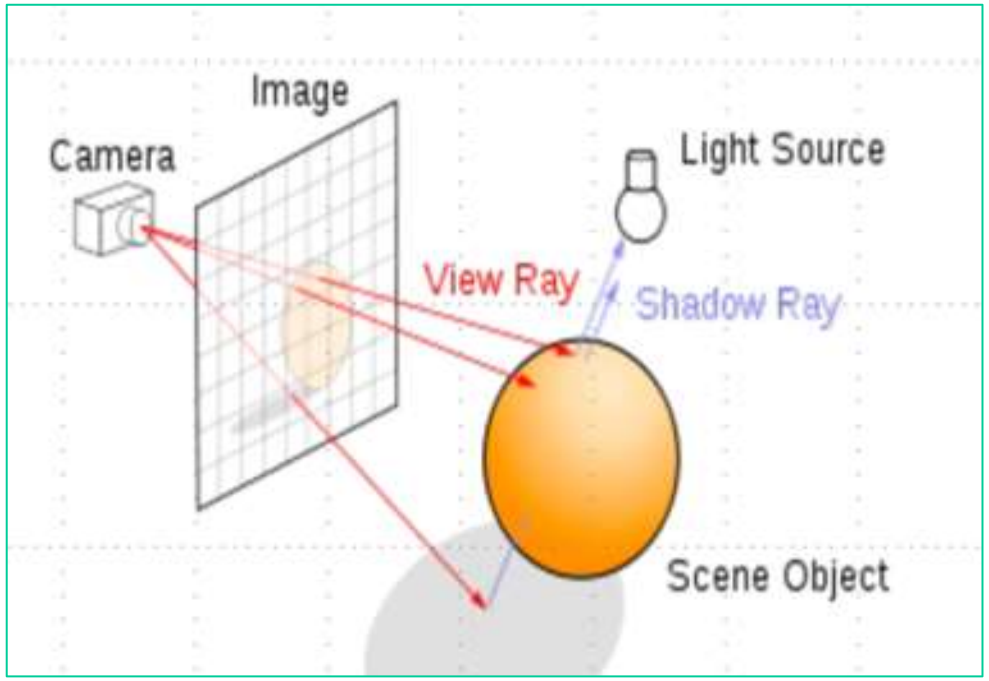
```
• FOR each pixel
    – Cast a ray and find the intersection point
    – IF there is an intersection
        • color = ambient
        • FOR each light
            color += shading from this light
              (depending on light property
              and material property)
        • return color
    – ELSE
        • return background color
```

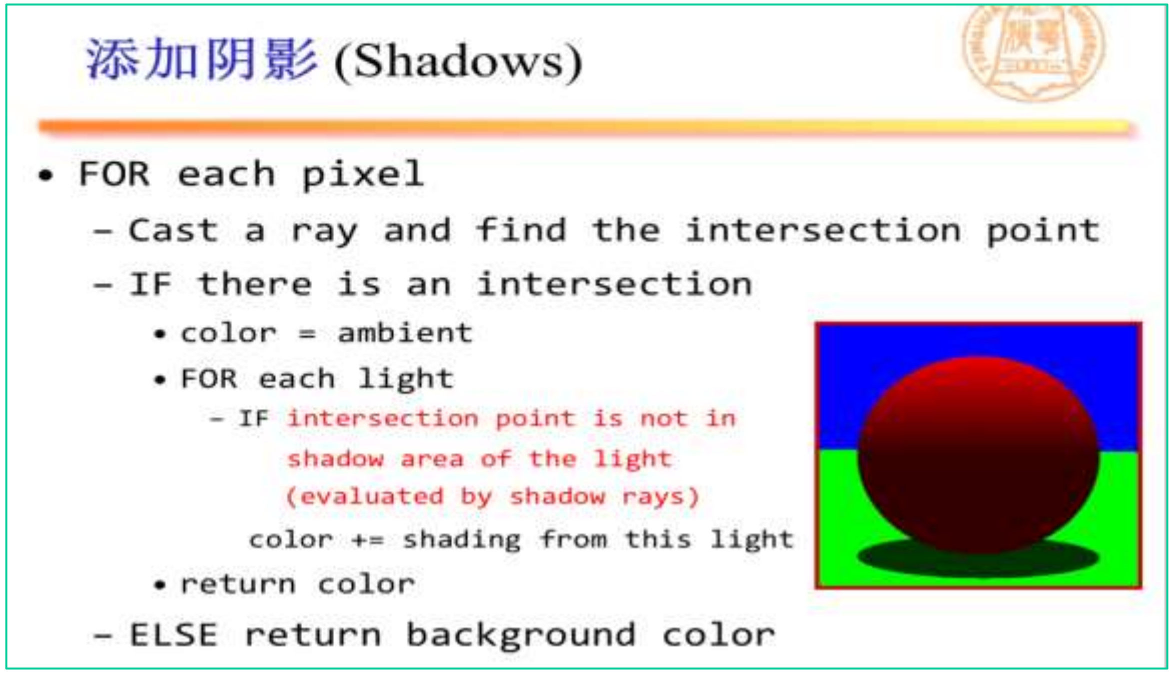


Ray Casting光线投射(cont.)

➢Ray Casting算法(有阴影)

- 由“相机”出发向每“像素”发出“view ray”，若和物体表面相交,得到“最近可见交点”,
- 从“最近可见交点”向“光源”投射光线“Shadow Ray”，如果它和光源之间有遮挡物，则有交，则该点赋予阴影色；否则，按blinn phong模型计算该交点的颜色值。




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
```

Ray Casting光线投射(cont.)

➢Ray Casting的优缺点

- 优点：
    - 算法思路简单，不需要投影，只发射光线，求交，计算颜色即可。
    - 比较容易计算得到“阴影”。

- 缺点：**Shading Pixels(Local Only)!**
    - 只考虑了物体到光源的1次光线弹射(bounce)，未考虑物体之间的光线弹射！

（即：计算物体表面点的光照颜色时，只考虑了“直接光照”（来自光源的光），没有考虑“间接光照”（如来自其它物体的弹射光）

（即：是局部“local”，而非“global”的光照计算）

Ray Tracing Outlines

- Ray Tracing vs. Rasterization

- Ray Casting
- Whitted-Style（Recursive）

- Ray-Surface Intersection
- Accelerating Ray-Surface Intersection

Whitted-Style RT

➤**Whitted-Style Ray Tracing**

➤考虑了光线在物体表面弹射多次（折射，镜面反射）

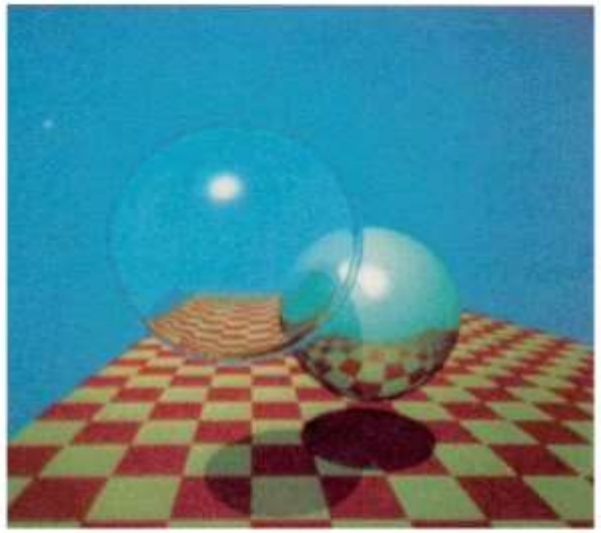
- 硬件加速后，渲染每帧花费时间越来越短：74minute，6second， 1/30 second

Recursive (Whitted-Style) Ray Tracing

“An improved Illumination model for shaded display”
T. Whitted, CACM 1980

Time:

- VAX 11/780 (1979) 74m
- PC (2006) 6s
- GPU (2012) 1/30s



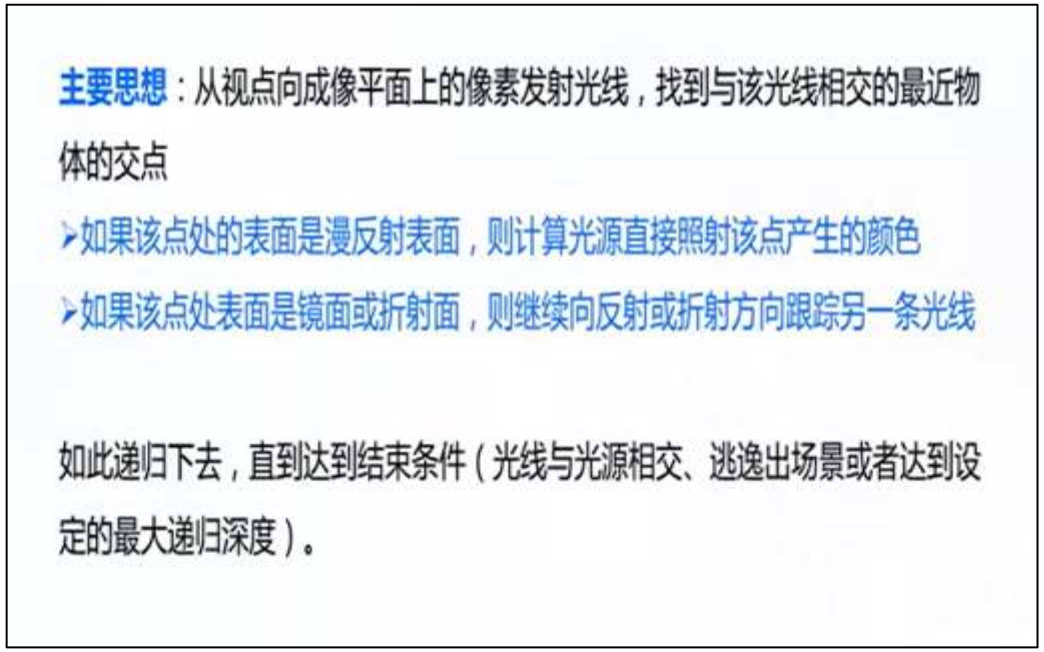
Spheres and Checkerboard, T. Whitted, 1979

Whitted-Style RT (cont.)

➢Algorithm Idea

- **Bouncing always bouncing perform “Specular Reflections” and “Refractions”（弹射总是发生在有“镜面反射”和“折射”时，即遇到镜面或者半透明表面时，会继续跟踪光线）**
- **Stop bouncing at Diffuse Surface！（在遇到漫反射表面时，会停止弹射）**

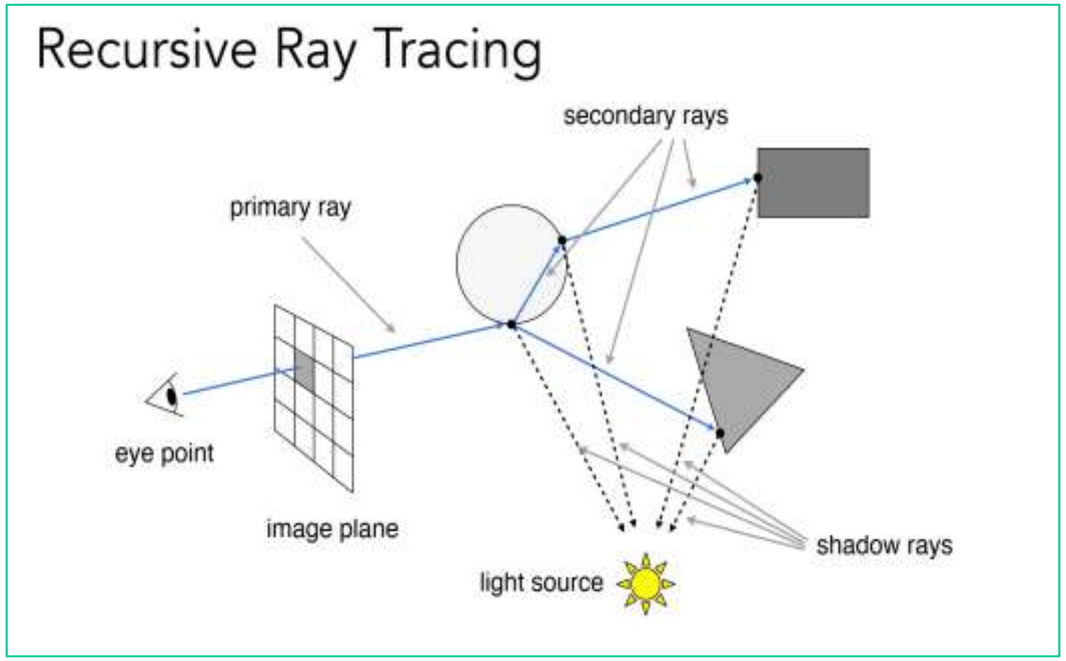
思考:为什么光线交到镜面和半透明材质的物体表面，才会继续弹射新光线继续跟踪呢？


主要思想：从视点向成像平面上的像素发射光线，找到与该光线相交的最近物体的交点

➢如果该点处的表面是漫反射表面，则计算光源直接照射该点产生的颜色

➢如果该点处表面是镜面或折射面，则继续向反射或折射方向跟踪另一条光线

如此递归下去，直到达到结束条件（光线与光源相交、逃逸出场景或者达到设定的最大递归深度）。



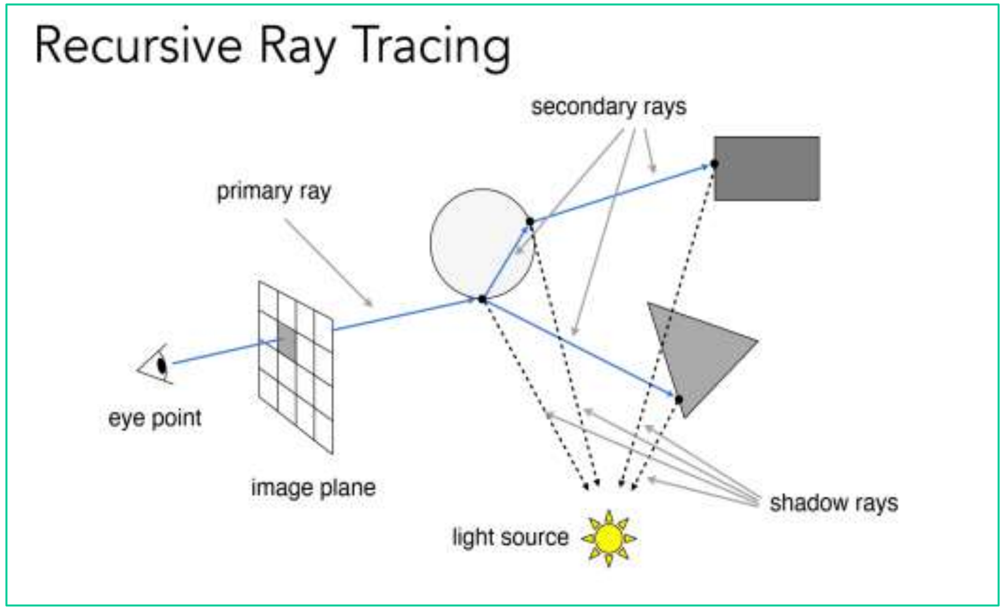
Whitted-Style RT (cont.)

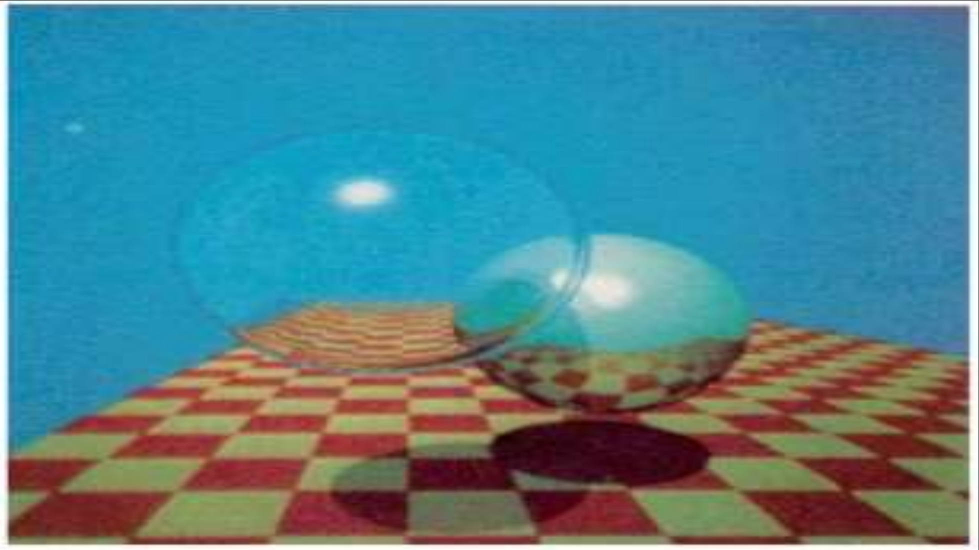
➢交点的颜色组成：

Phong模型 + 透射光强 + 反射光强

$$I=I_aK_a+I_pK_d(L\cdot N)+I_pK_s(H\cdot N)^n+I_tK_t+I_rK_r$$

- ➢直接光照(blinn phong模型)+ 间接光照(透射光模型+ 镜面反射光模型)




Spheres and Checkerboard, T. Whitted, 1979

- ➢格子平面(漫反射表面)：漫反射了光源来的光，只有”直接光照” 色（以及“阴影色”）
- ➢后面的球(镜面表面)：镜面反射了格子平面和蓝色背景来的间接光，有“间接光”(镜面反射光)
- ➢前面的球(半透明)：折射了后面球，格子平面和蓝色背景反射来的光，有“间接光”(透射光)

Whitted-Style RT (cont.)

- 算法过程描述：

1. 初始化：为图像中的每个像素发射一条光线。
2. 光线追踪：对于每条光线，执行以下步骤：

   a.光线与场景的求交点：

   从摄像机位置沿着光线方向发射一条射线。计算射线与场景中物体的交点。如果没有交点，返回背景颜色结束；否则选择最近交点进行下一步：

   b. 计算直接光照

   对于场景中的每个光源：计算从最近交点到光源的阴影射线，如果阴影射线上没有物体（即不在阴影中），计算漫反射和镜面反射并将漫反射和镜面反射的颜色加到当前像素的颜色上，结束。（如果阴影射线上有物体（即在阴影中），则设置阴影色，并加到当前像素的颜色上，结束。）

   c. 计算反射（递归）

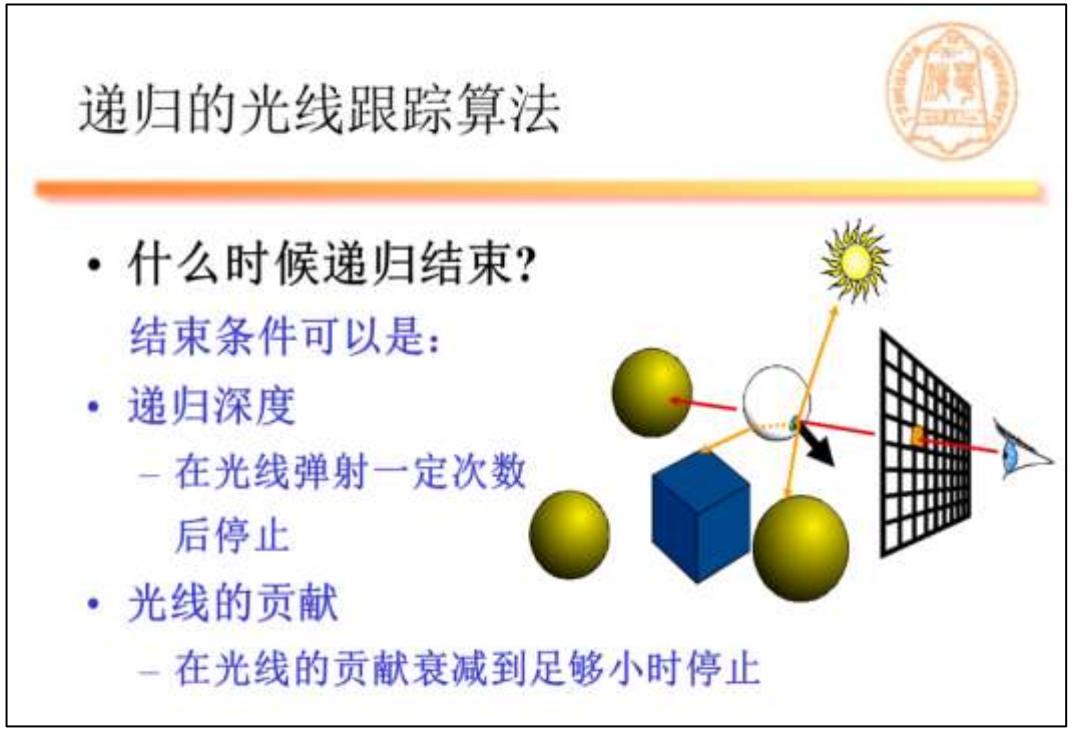
   如果交点处的物体表面是镜面反射性的：计算反射射线的方向。递归地追踪反射射线，并乘以物体的反射率。将反射颜色加到当前像素的颜色上。

   d. 计算折射（递归）

   如果交点处的物体是折射性的：计算折射射线的方向。递归地追踪折射射线，并乘以物体的透明度。将折射颜色加到当前像素的颜色上。
3. 递归终止：如果达到**结束条件**（如：最大递归深度），则停止递归。
4. 颜色合成：将所有计算出的颜色（直接光照、反射、折射、阴影色）合成，得到最终的像素颜色。
5. 输出图像：将计算出的颜色赋给相应的像素。

Whitted-Style RT (cont.)

- ➢递归结束条件:
  - ➢达到设定的最大递归次数
  - ➢光线贡献衰减到足够小时
  - ➢光线与光源相交
  - ➢光线逃逸出场景



递归的光线跟踪算法

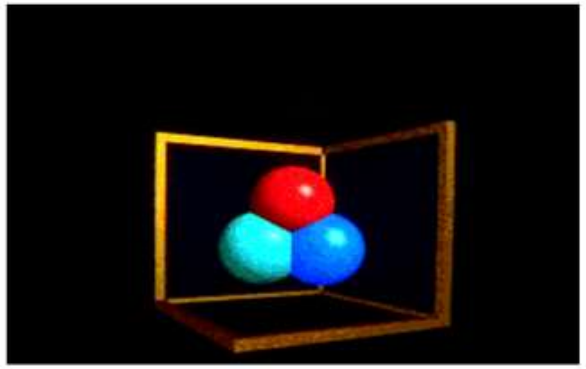
- 什么时候递归结束?

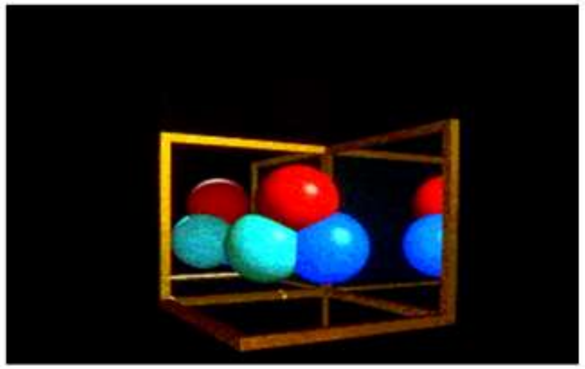
  结束条件可以是:
- 递归深度
  - 在光线弹射一定次数后停止
- 光线的贡献
  - 在光线的贡献衰减到足够小时停止

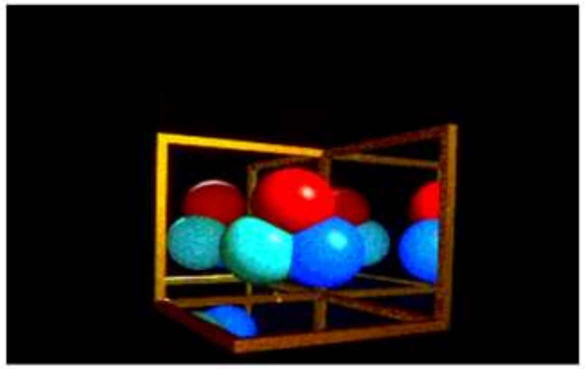
Whitted-Style RT (cont.)

➢The number of recursion递归次数对渲染结果的影响

光线跟踪效果示例

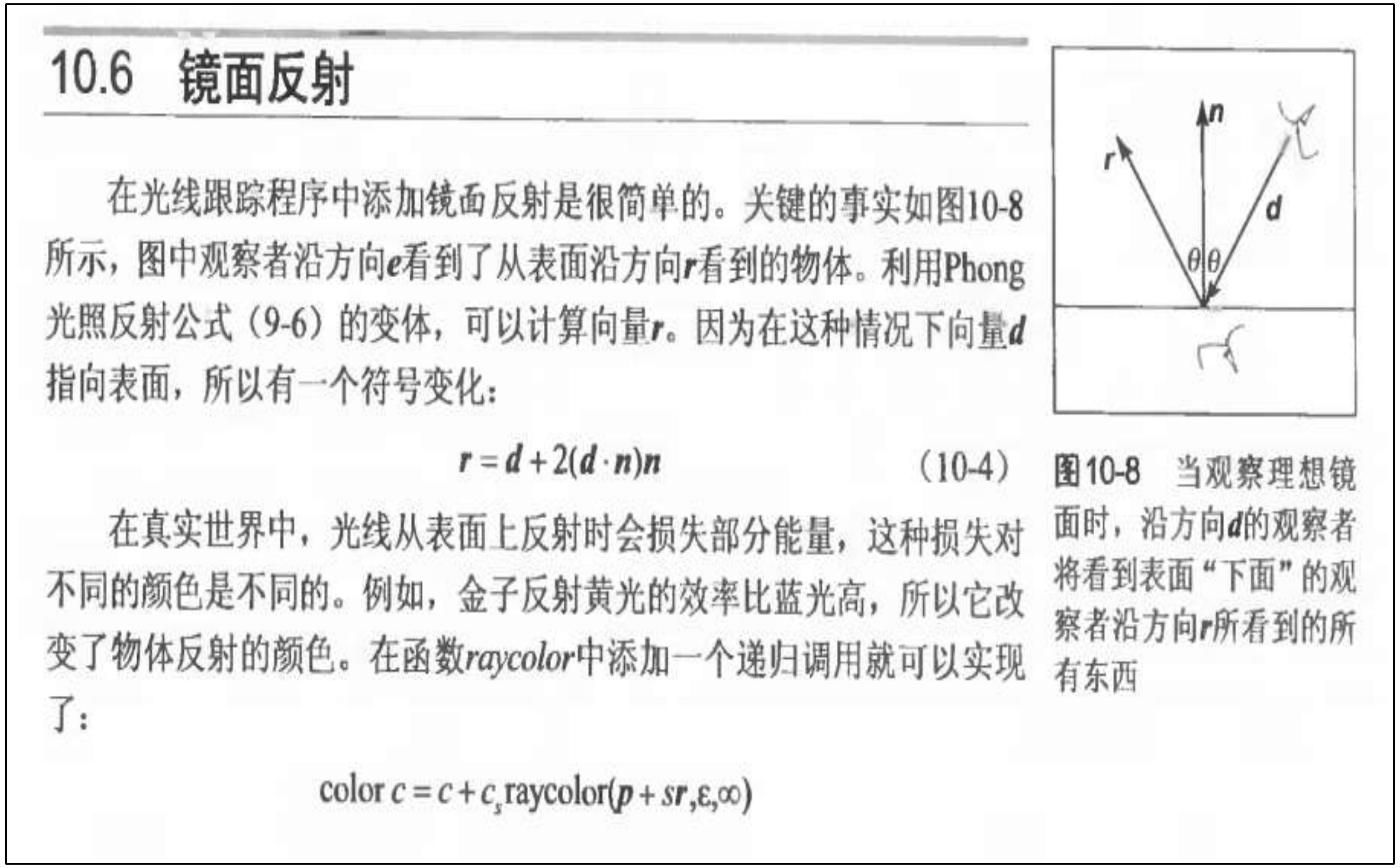

0 层递归


1 层递归


2 层递归

Whitted-Style RT (cont.)

- Specular Reflections 镜面反射光的计算 *(参“虎书”)*



10.6 镜面反射

在光线跟踪程序中添加镜面反射是很简单的。关键的事实如图10-8所示，图中观察者沿方向*e*看到了从表面沿方向***r***看到的物体。利用Phong光照反射公式（9-6）的变体，可以计算向量***r***。因为在这种情况下向量***d***指向表面，所以有一个符号变化：

$$\boldsymbol{r} = \boldsymbol{d} + 2(\boldsymbol{d}\cdot\boldsymbol{n})\boldsymbol{n} \qquad (10\text{-}4)$$

在真实世界中，光线从表面上反射时会损失部分能量，这种损失对不同的颜色是不同的。例如，金子反射黄光的效率比蓝光高，所以它改变了物体反射的颜色。在函数*raycolor*中添加一个递归调用就可以实现了：

$$\text{color } c = c + c_s \text{raycolor}(\boldsymbol{p} + s\boldsymbol{r}, \varepsilon, \infty)$$

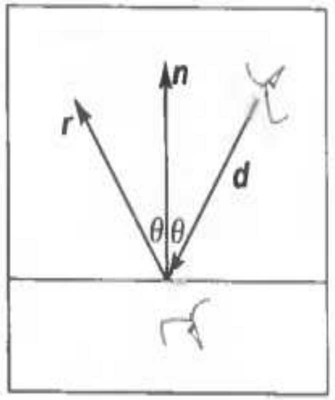
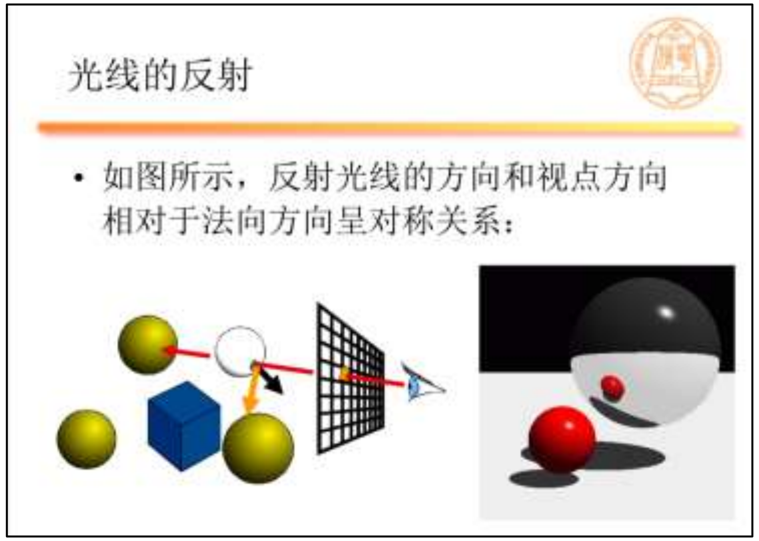
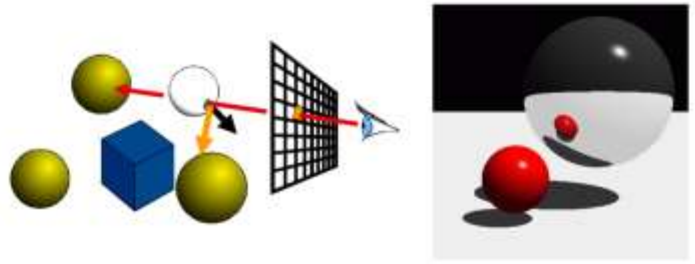


图10-8 当观察理想镜面时，沿方向***d***的观察者将看到表面“下面”的观察者沿方向***r***所看到的所有东西



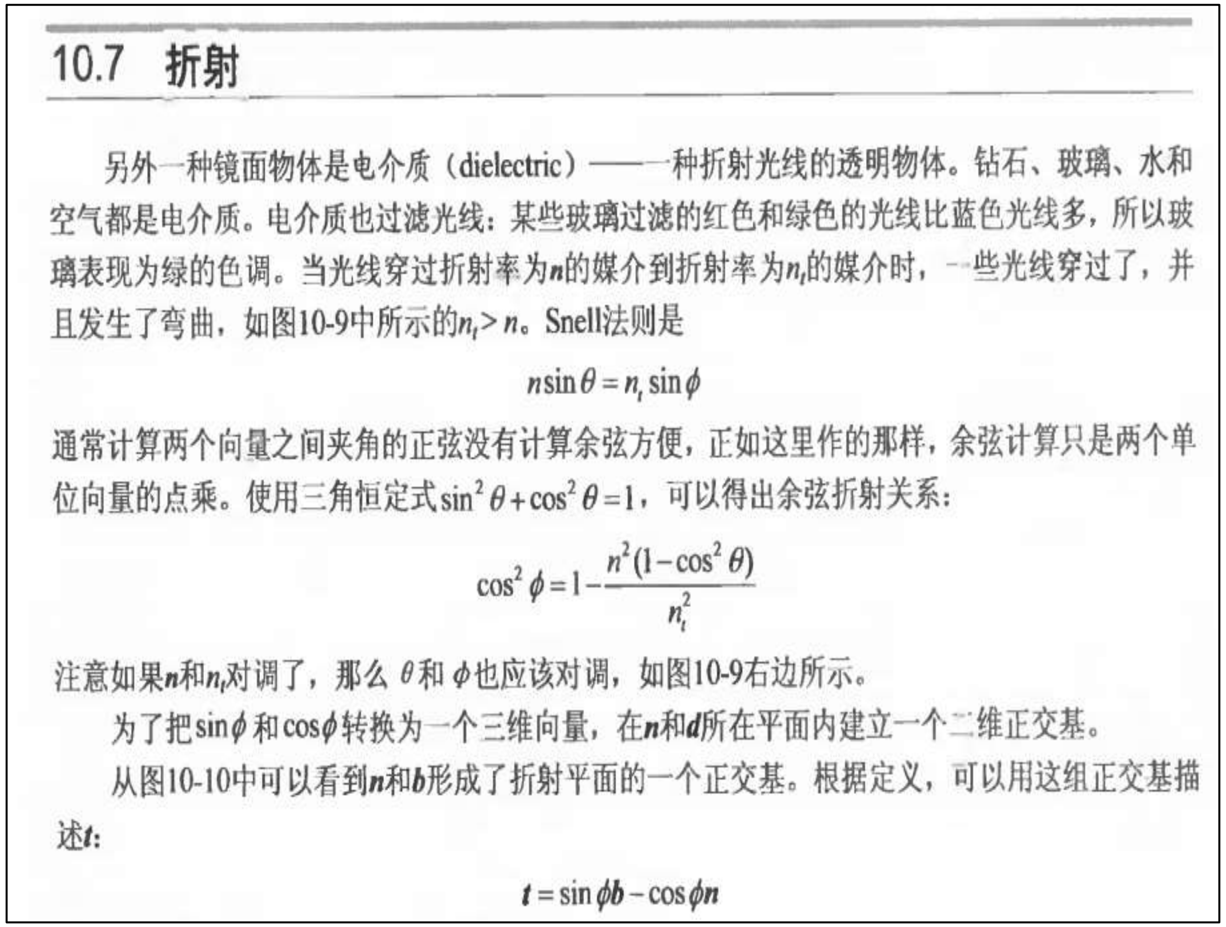
光线的反射

- 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：



Whitted-Style RT (cont.)

➢**Ray Refractions折射光的计算** *(参"虎书")*



10.7 折射

另外一种镜面物体是电介质（dielectric）——一种折射光线的透明物体。钻石、玻璃、水和空气都是电介质。电介质也过滤光线：某些玻璃过滤的红色和绿色的光线比蓝色光线多，所以玻璃表现为绿的色调。当光线穿过折射率为n的媒介到折射率为n_t的媒介时，一些光线穿过了，并且发生了弯曲，如图10-9中所示的$n_t > n$。Snell法则是

$$n\sin\theta = n_t \sin\phi$$

通常计算两个向量之间夹角的正弦没有计算余弦方便，正如这里作的那样，余弦计算只是两个单位向量的点乘。使用三角恒定式$\sin^2\theta + \cos^2\theta = 1$，可以得出余弦折射关系：

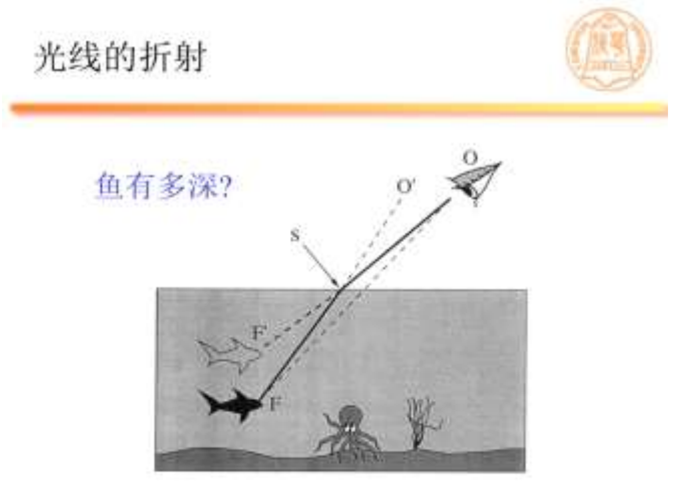
$$\cos^2\phi = 1 - \frac{n^2(1-\cos^2\theta)}{n_t^2}$$

注意如果n和n_t对调了，那么 θ和 ϕ也应该对调，如图10-9右边所示。

为了把$\sin\phi$和$\cos\phi$转换为一个三维向量，在$\mathbf{n}$和$\mathbf{d}$所在平面内建立一个二维正交基。

从图10-10中可以看到$\mathbf{n}$和$\mathbf{b}$形成了折射平面的一个正交基。根据定义，可以用这组正交基描述$\mathbf{t}$:

$$\mathbf{t} = \sin\phi\mathbf{b} - \cos\phi\mathbf{n}$$



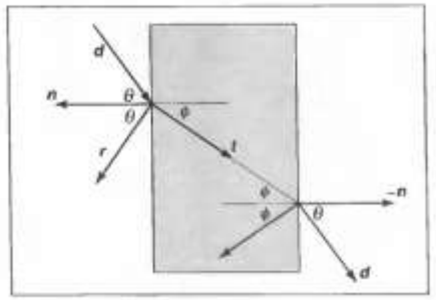


图10-9 Snell法则描述了角度ϕ如何依赖角度θ、物体和周围介质的反射率

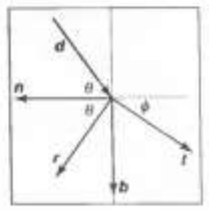


图10-10 向量$\mathbf{n}$和$\mathbf{b}$形成了一个正交基，它与传播向量$\mathbf{t}$平行

既然可以用同样的正交基描述$\mathbf{d}$，而$\mathbf{d}$是已知的，所以可以求解$\mathbf{b}$：

$$\mathbf{d} = \sin\theta\mathbf{b} - \cos\theta\mathbf{n}$$

$$\mathbf{b} = \frac{\mathbf{d} + \mathbf{n}\cos\theta}{\sin\theta}$$

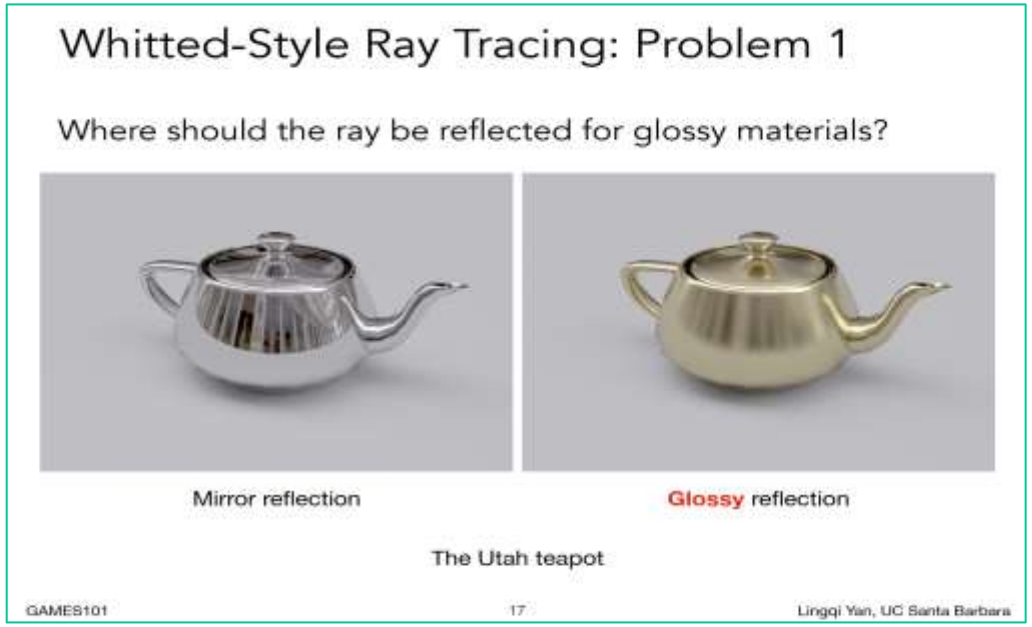
这意味着可以利用已知的变量求解$\mathbf{t}$:

Whitted-Style RT (cont.)

➢ Merits and Defects of Whitted-Style Ray Tracing

- ➢**merits:考虑了光线在物体之间的弹射，实现了部分间接光照效果（如透明）**
- ➢**Defects:**严格意义讲：whitted-style RT 还算不上全局光照global illumination

1. Where should the ray be reflected for **glossy materials（not mirror/specular)?** (光亮表面不是理想镜面）
2. No reflections between **diffuse materials**？（没有考虑漫反射材质物体之间的漫反射光）

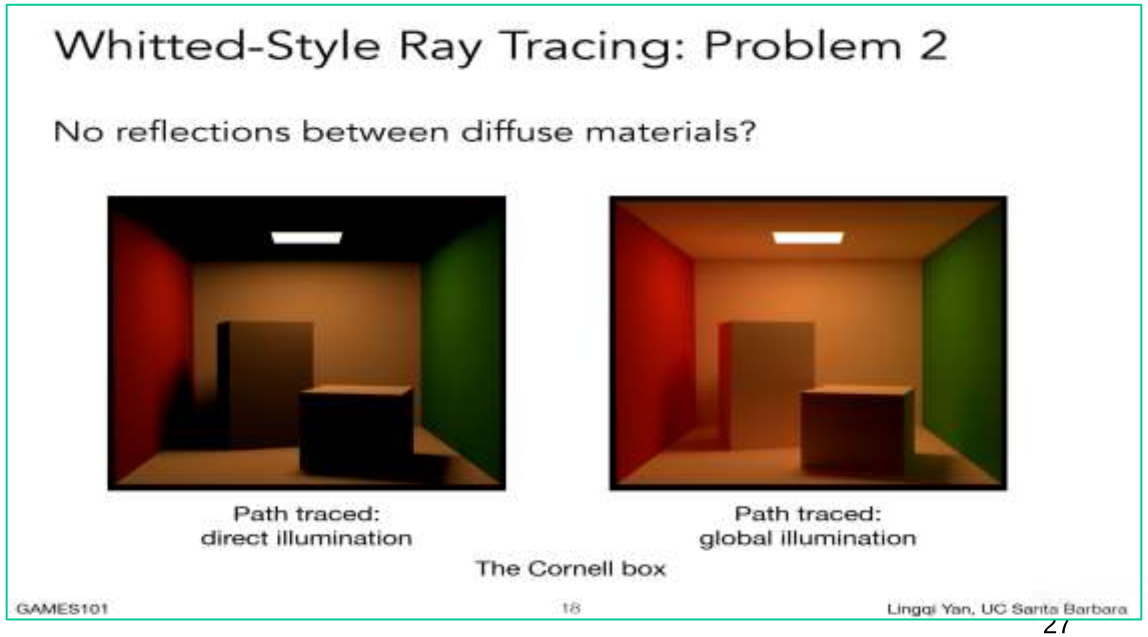


Whitted-Style Ray Tracing: Problem 1

Where should the ray be reflected for glossy materials?

The Utah teapot

GAMES101 17 Lingqi Yan, UC Santa Barbara



Whitted-Style Ray Tracing: Problem 2

No reflections between diffuse materials?

The Cornell box

GAMES101 18 Lingqi Yan, UC Santa Barbara

Ray Tracing Outlines

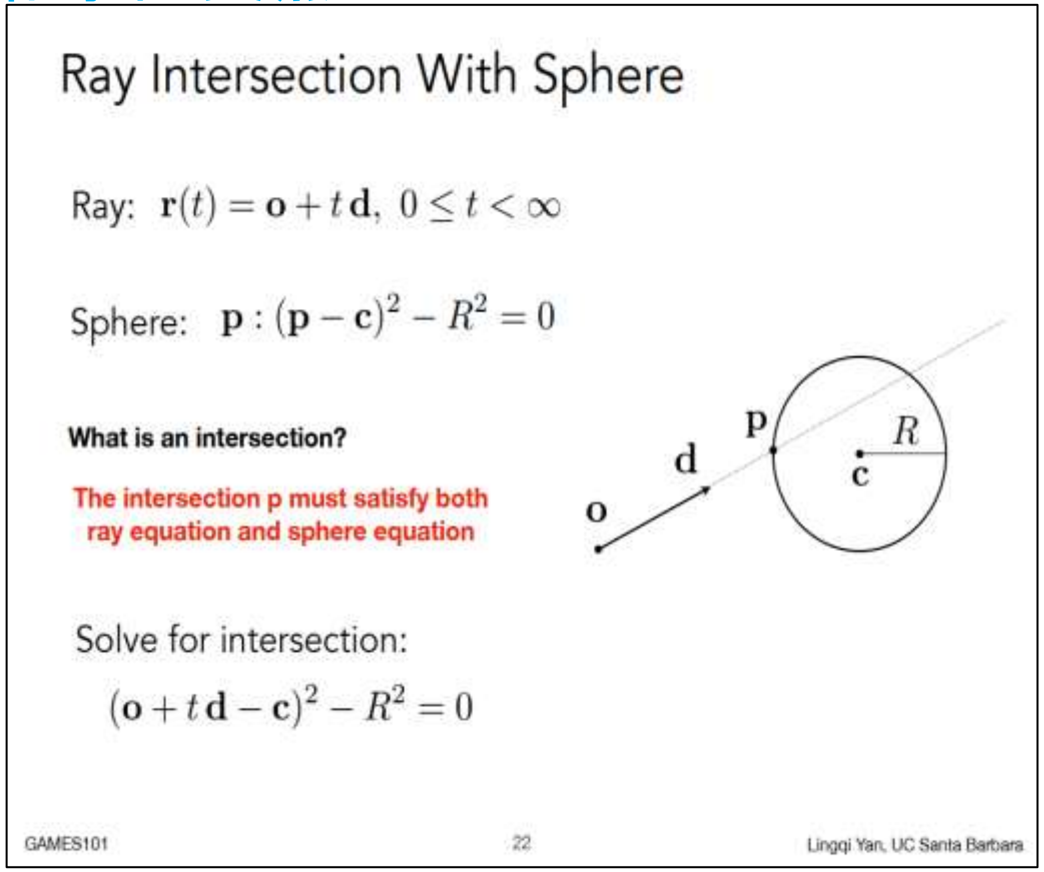
- Ray Tracing vs. Rasterization

- Ray Casting
- Whitted-Style（Recursive）

- Ray-Surface Intersection
- Accelerating Ray-Surface Intersection

Accelerating Ray-Surface Intersection（cont.）

- 光线与“隐函数表示的表面”求交
  - 光线与球面求交：需将光线参数方程带入球面函数求解参数t值后，进行判断和求出交点



Ray Intersection With Sphere

Ray: $\mathbf{r}(t) = \mathbf{o} + t\,\mathbf{d},\ 0 \le t < \infty$

Sphere: $\mathbf{p} : (\mathbf{p} - \mathbf{c})^2 - R^2 = 0$

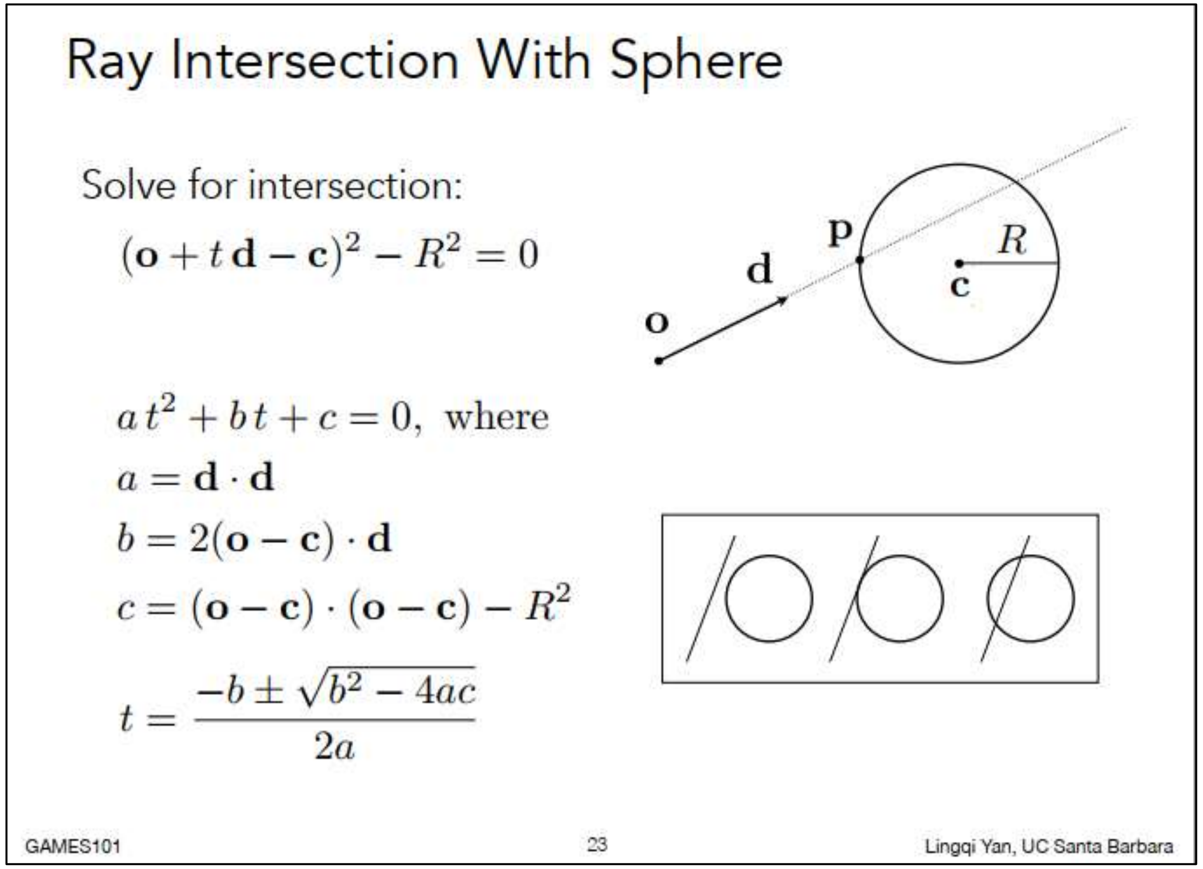
What is an intersection?

The intersection p must satisfy both ray equation and sphere equation

Solve for intersection:

$$(\mathbf{o} + t\,\mathbf{d} - \mathbf{c})^2 - R^2 = 0$$

GAMES101 22 Lingqi Yan, UC Santa Barbara



Ray Intersection With Sphere

Solve for intersection:

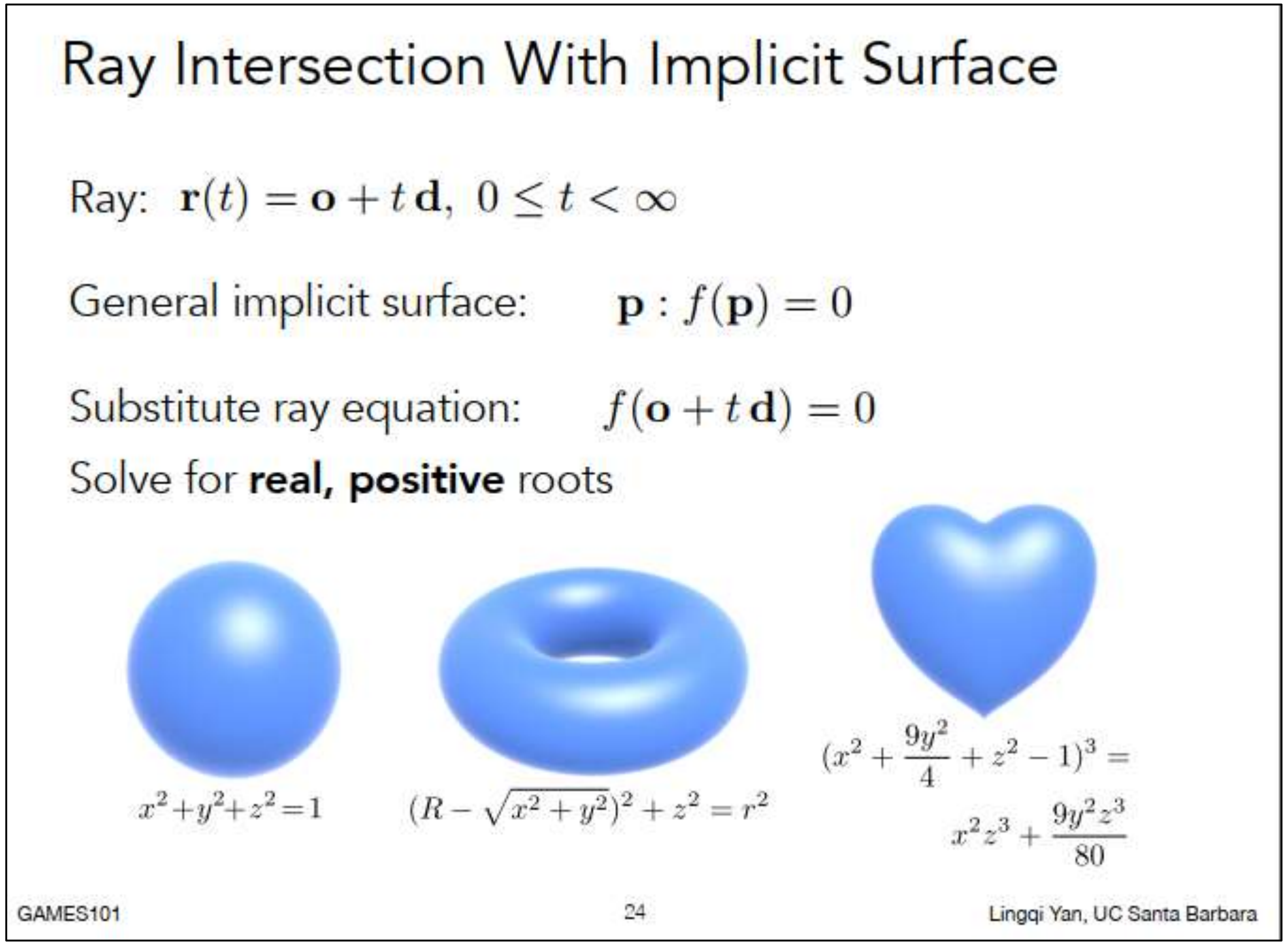
$$(\mathbf{o} + t\,\mathbf{d} - \mathbf{c})^2 - R^2 = 0$$

$$a\,t^2 + b\,t + c = 0, \text{ where}$$
$$a = \mathbf{d} \cdot \mathbf{d}$$
$$b = 2(\mathbf{o} - \mathbf{c}) \cdot \mathbf{d}$$
$$c = (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - R^2$$
$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

GAMES101 23 Lingqi Yan, UC Santa Barbara

Ray-Surface Intersection（cont.）

- 光线与“隐函数表示的表面”求交（cont.)



Ray Intersection With Implicit Surface

Ray: $\mathbf{r}(t) = \mathbf{o} + t\,\mathbf{d},\ 0 \leq t < \infty$

General implicit surface: $\mathbf{p} : f(\mathbf{p}) = 0$

Substitute ray equation: $f(\mathbf{o} + t\,\mathbf{d}) = 0$

Solve for **real, positive** roots

$x^2 + y^2 + z^2 = 1$

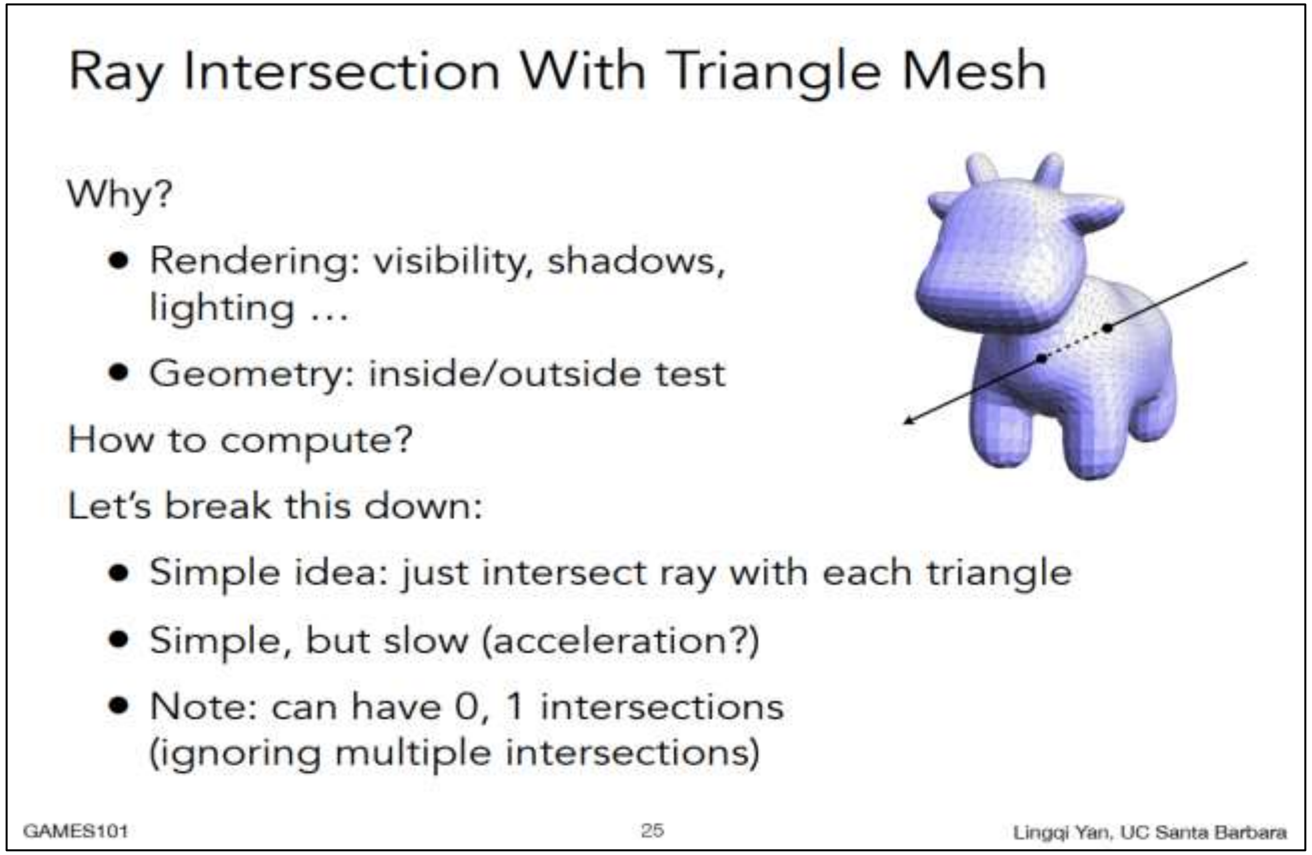
$(R - \sqrt{x^2 + y^2})^2 + z^2 = r^2$

$$(x^2 + \frac{9y^2}{4} + z^2 - 1)^3 = x^2 z^3 + \frac{9y^2 z^3}{80}$$

GAMES101 24 Lingqi Yan, UC Santa Barbara

Ray-Surface Intersection（cont.）

- .光线与”显示表示”的表面求交
  - 光线与（网格）求交: 需要和每个小三角形求交，复杂度高，慢需要加速！



Ray Intersection With Triangle Mesh

Why?

- Rendering: visibility, shadows, lighting …
- Geometry: inside/outside test

How to compute?

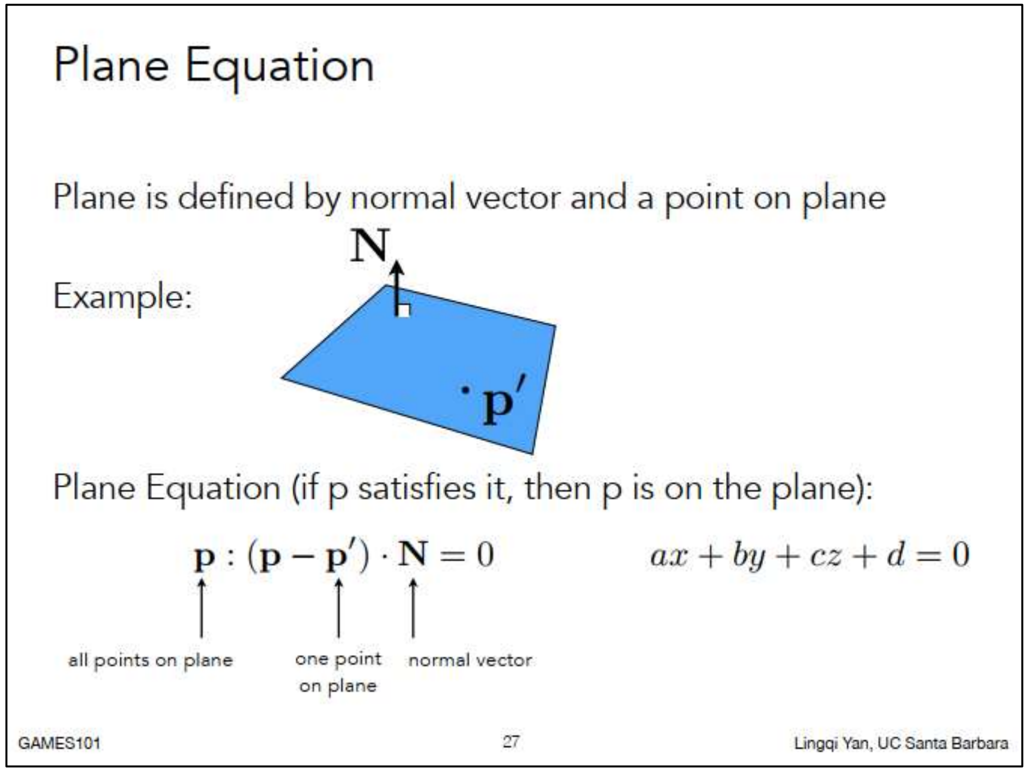
Let's break this down:

- Simple idea: just intersect ray with each triangle
- Simple, but slow (acceleration?)
- Note: can have 0, 1 intersections (ignoring multiple intersections)

GAMES101 25 Lingqi Yan, UC Santa Barbara

Ray-Surface Intersection（cont.）

- .光线与”显示表示”的表面求交（cont.)
  - 光线与平面求交：需要知道“面外法向量”和“面上的一点p”



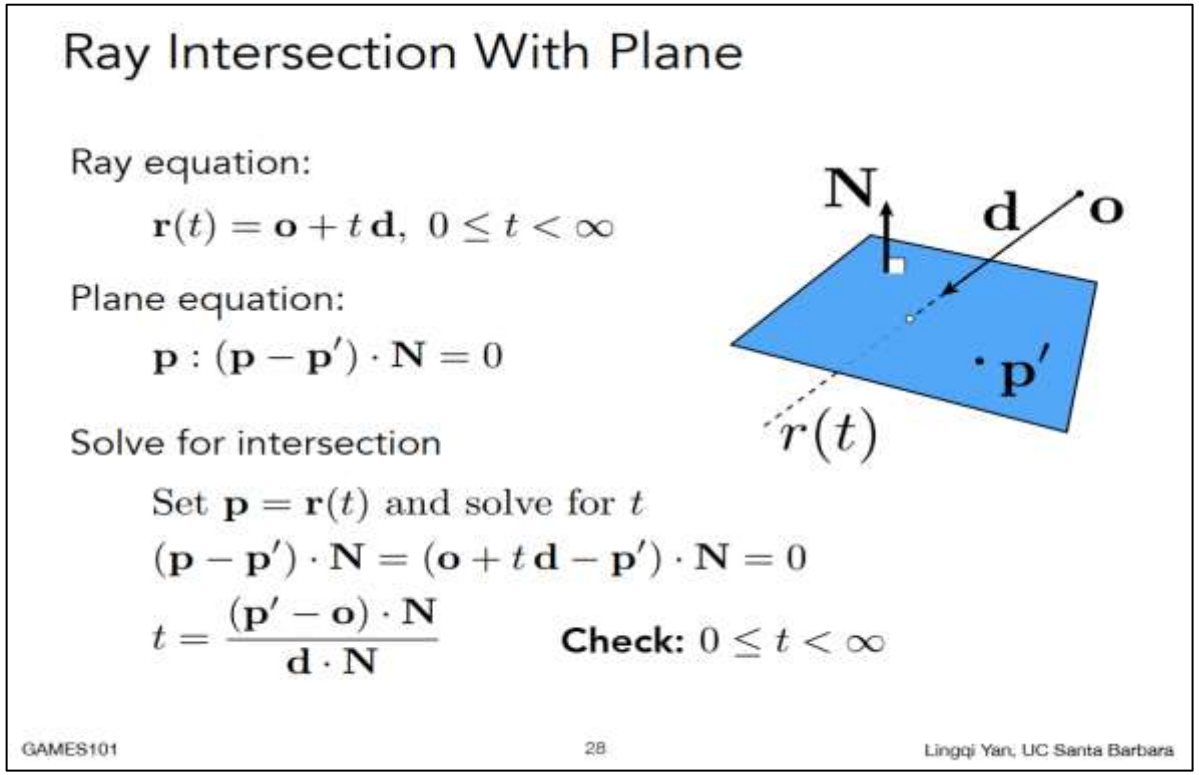
Plane Equation

Plane is defined by normal vector and a point on plane

Example:

Plane Equation (if p satisfies it, then p is on the plane):

$$\underset{\text{all points on plane}}{\mathbf{p}} : (\mathbf{p} - \underset{\text{one point on plane}}{\mathbf{p}'}) \cdot \underset{\text{normal vector}}{\mathbf{N}} = 0 \qquad ax + by + cz + d = 0$$



Ray Intersection With Plane

Ray equation:

$$\mathbf{r}(t) = \mathbf{o} + t\,\mathbf{d},\ 0 \leq t < \infty$$

Plane equation:

$$\mathbf{p} : (\mathbf{p} - \mathbf{p}') \cdot \mathbf{N} = 0$$

Solve for intersection

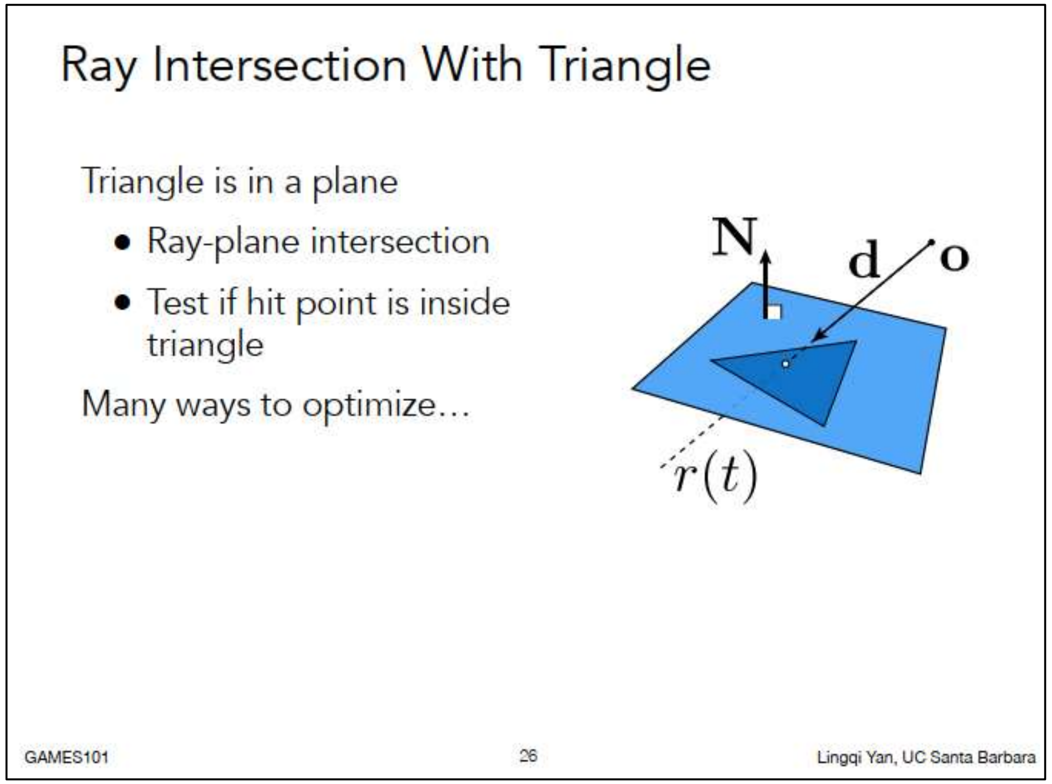
Set $\mathbf{p} = \mathbf{r}(t)$ and solve for t

$$(\mathbf{p} - \mathbf{p}') \cdot \mathbf{N} = (\mathbf{o} + t\,\mathbf{d} - \mathbf{p}') \cdot \mathbf{N} = 0$$

$$t = \frac{(\mathbf{p}' - \mathbf{o}) \cdot \mathbf{N}}{\mathbf{d} \cdot \mathbf{N}} \qquad \textbf{Check: } 0 \leq t < \infty$$

Ray-Surface Intersection（cont.）

- .光线与”显示表示”的表面求交(cont.)
  - 光线与三角形求交:
    - ➢分两步：先光线和“面”求交，然后判定交点在三角形内。
    - ➢也可合并求解：“光线方程=重心坐标表示的平面方程”，可求出交点参数t和重心坐标b1,b2
      - **– 若t>0,则有交，再判若0<b1<1,0<b2<1，0<1-b1-b2<1,则交点在三角形内。**

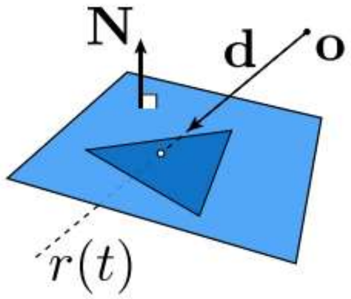


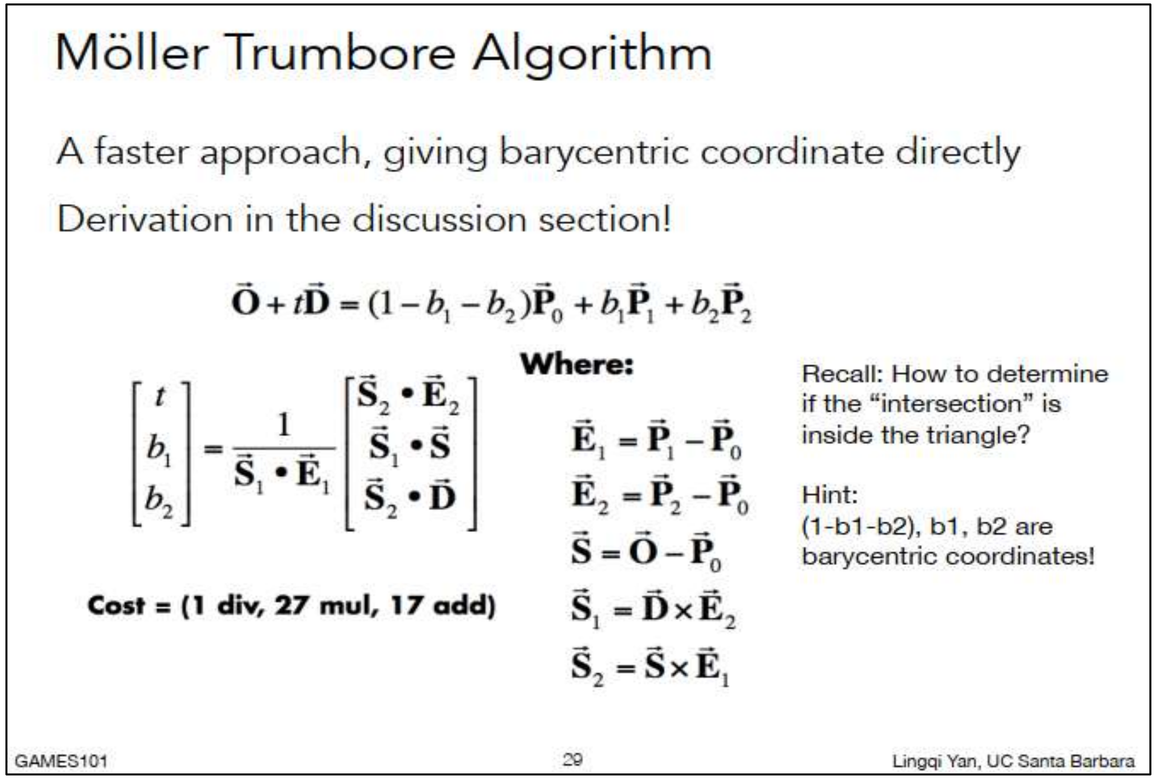
Ray Intersection With Triangle

Triangle is in a plane

- Ray-plane intersection
- Test if hit point is inside triangle

Many ways to optimize…





Möller Trumbore Algorithm

A faster approach, giving barycentric coordinate directly

Derivation in the discussion section!

$$\vec{\mathbf{O}} + t\vec{\mathbf{D}} = (1 - b_1 - b_2)\vec{\mathbf{P}}_0 + b_1\vec{\mathbf{P}}_1 + b_2\vec{\mathbf{P}}_2$$

$$\begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{\vec{\mathbf{S}}_1 \bullet \vec{\mathbf{E}}_1} \begin{bmatrix} \vec{\mathbf{S}}_2 \bullet \vec{\mathbf{E}}_2 \\ \vec{\mathbf{S}}_1 \bullet \vec{\mathbf{S}} \\ \vec{\mathbf{S}}_2 \bullet \vec{\mathbf{D}} \end{bmatrix}$$

Where:

$$\vec{\mathbf{E}}_1 = \vec{\mathbf{P}}_1 - \vec{\mathbf{P}}_0$$
$$\vec{\mathbf{E}}_2 = \vec{\mathbf{P}}_2 - \vec{\mathbf{P}}_0$$
$$\vec{\mathbf{S}} = \vec{\mathbf{O}} - \vec{\mathbf{P}}_0$$
$$\vec{\mathbf{S}}_1 = \vec{\mathbf{D}} \times \vec{\mathbf{E}}_2$$
$$\vec{\mathbf{S}}_2 = \vec{\mathbf{S}} \times \vec{\mathbf{E}}_1$$

Cost = (1 div, 27 mul, 17 add)

Recall: How to determine if the “intersection” is inside the triangle?

Hint:
(1-b1-b2), b1, b2 are barycentric coordinates!

Ray Tracing Outlines

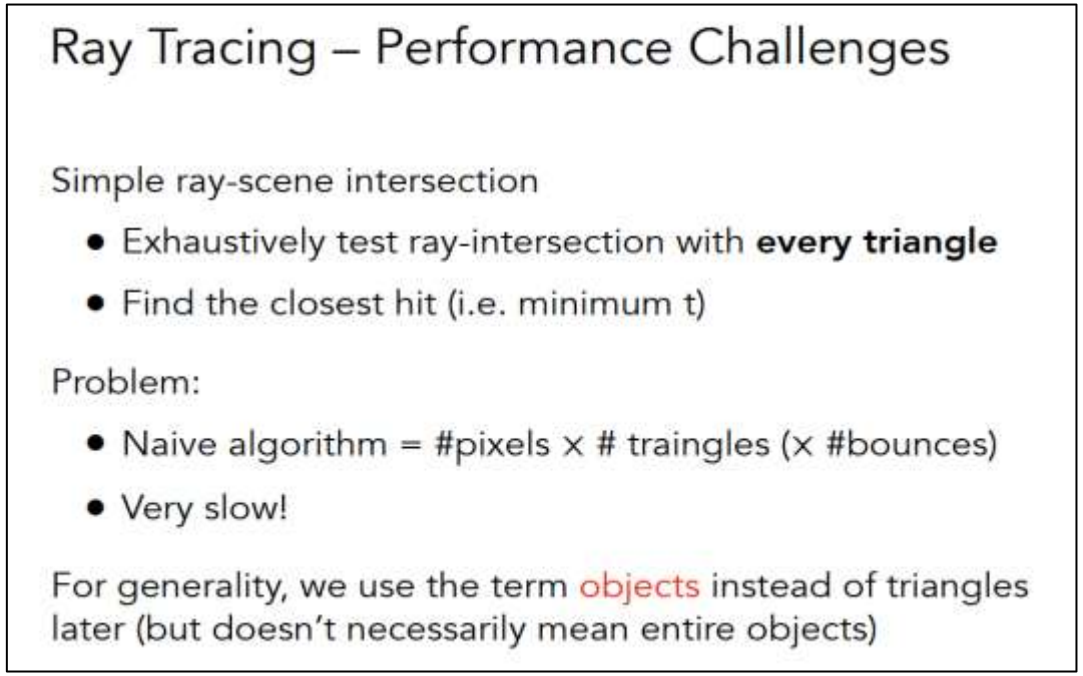
- Ray Tracing vs. Rasterization

- Ray Casting
- Whitted-Style（Recursive）

- Ray-Surface Intersection
- Accelerating Ray-Surface Intersection

Accelerating Ray-Surface Intersection

- Performance Challenges: **intersection**
  - 简单的光线-场景（物体）求交：需要和网格表示的物体的“每个三角形”进行测试，找到最近的交点，
  - 原始的求交算法非常慢，因为计算复杂度很高：像素个数* 三角形个数* 弹射次数！
    - 如右图：场景中具有10.7M（一千多万个三角形）



Ray Tracing – Performance Challenges

Simple ray-scene intersection

- Exhaustively test ray-intersection with **every triangle**
- Find the closest hit (i.e. minimum t)

Problem:

- Naive algorithm = #pixels × # traingles (× #bounces)
- Very slow!

For generality, we use the term objects instead of triangles later (but doesn't necessarily mean entire objects)

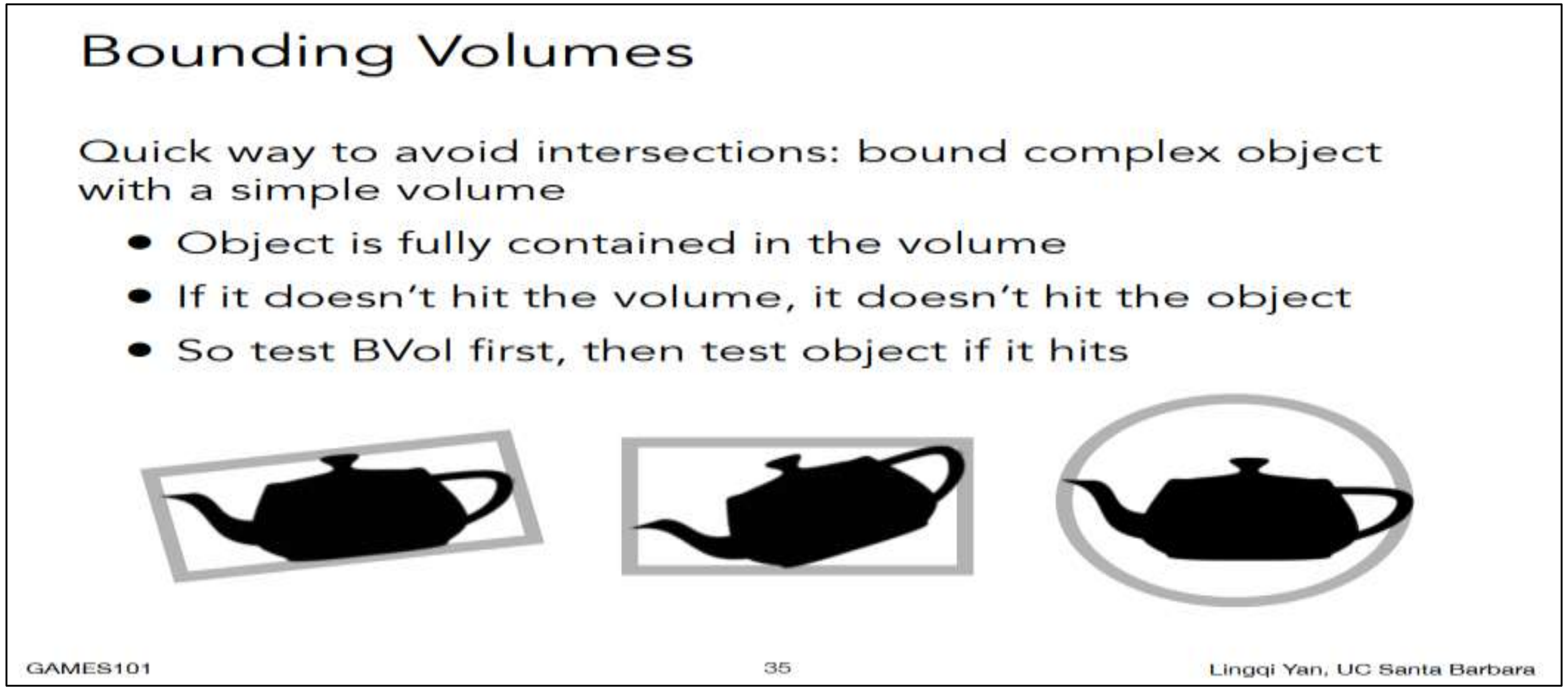
Ray Tracing – Performance Challenges

San Miguel Scene, 10.7M triangles

Accelerating Ray-Surface Intersection(cont.)

➢**Bounding Volumes(包围体)：**

- ➢思想：将复杂表示的几何体（如mesh）包在一个体（volume）中，如果光线没有击中（hit)这个体，那么它就不会击中这个几何体。所以，先测试是否hit包围体，然后再测试是否hit几何体对象。
- ➢包围体类型：包围盒Bounding Box，球盒Bounding Sphere



Bounding Volumes

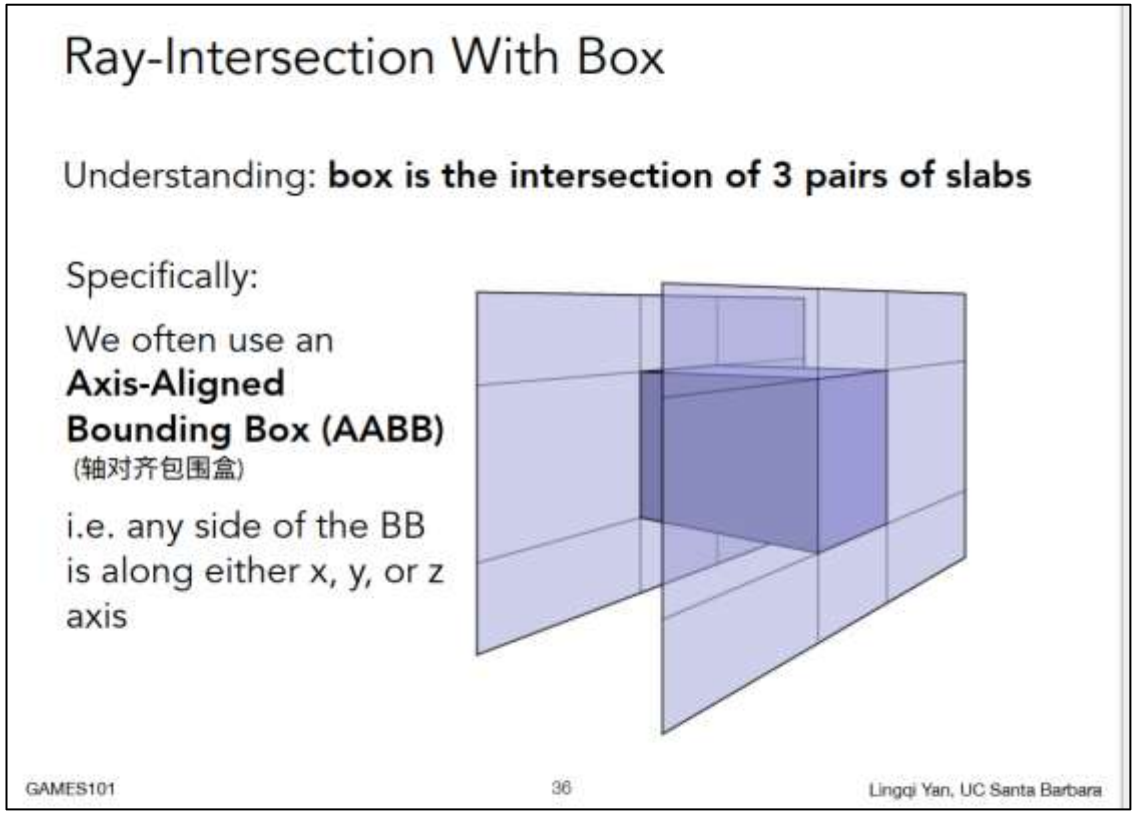
Quick way to avoid intersections: bound complex object with a simple volume

- Object is fully contained in the volume
- If it doesn't hit the volume, it doesn't hit the object
- So test BVol first, then test object if it hits

GAMES101 35 Lingqi Yan, UC Santa Barbara

Accelerating Ray-Surface Intersection(cont.)

- **Ray-Intersection With Bounding Box(包围盒)**
  - **Bounding Box: 由三对相互垂直的对面平板(slabs)界定的盒子**
    - **Axis-Alighed Bounding Box(AABB): 轴对齐包围盒(slabs平行于主面)**


Ray-Intersection With Box

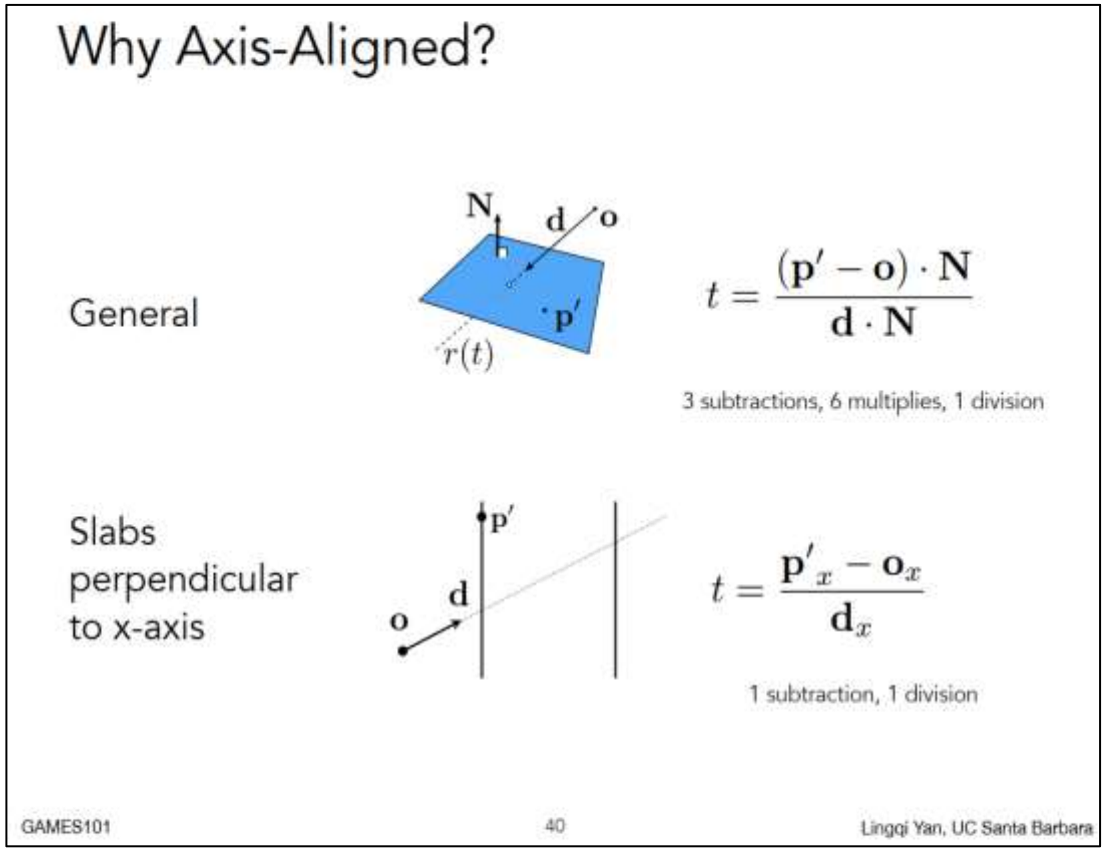
Understanding: **box is the intersection of 3 pairs of slabs**

Specifically:

We often use an **Axis-Aligned Bounding Box (AABB)** (轴对齐包围盒)

i.e. any side of the BB is along either x, y, or z axis

GAMES101 36 Lingqi Yan, UC Santa Barbara


Why Axis-Aligned?

General

$$t = \frac{(\mathbf{p}' - \mathbf{o}) \cdot \mathbf{N}}{\mathbf{d} \cdot \mathbf{N}}$$

3 subtractions, 6 multiplies, 1 division

Slabs perpendicular to x-axis

$$t = \frac{\mathbf{p}'_x - \mathbf{o}_x}{\mathbf{d}_x}$$

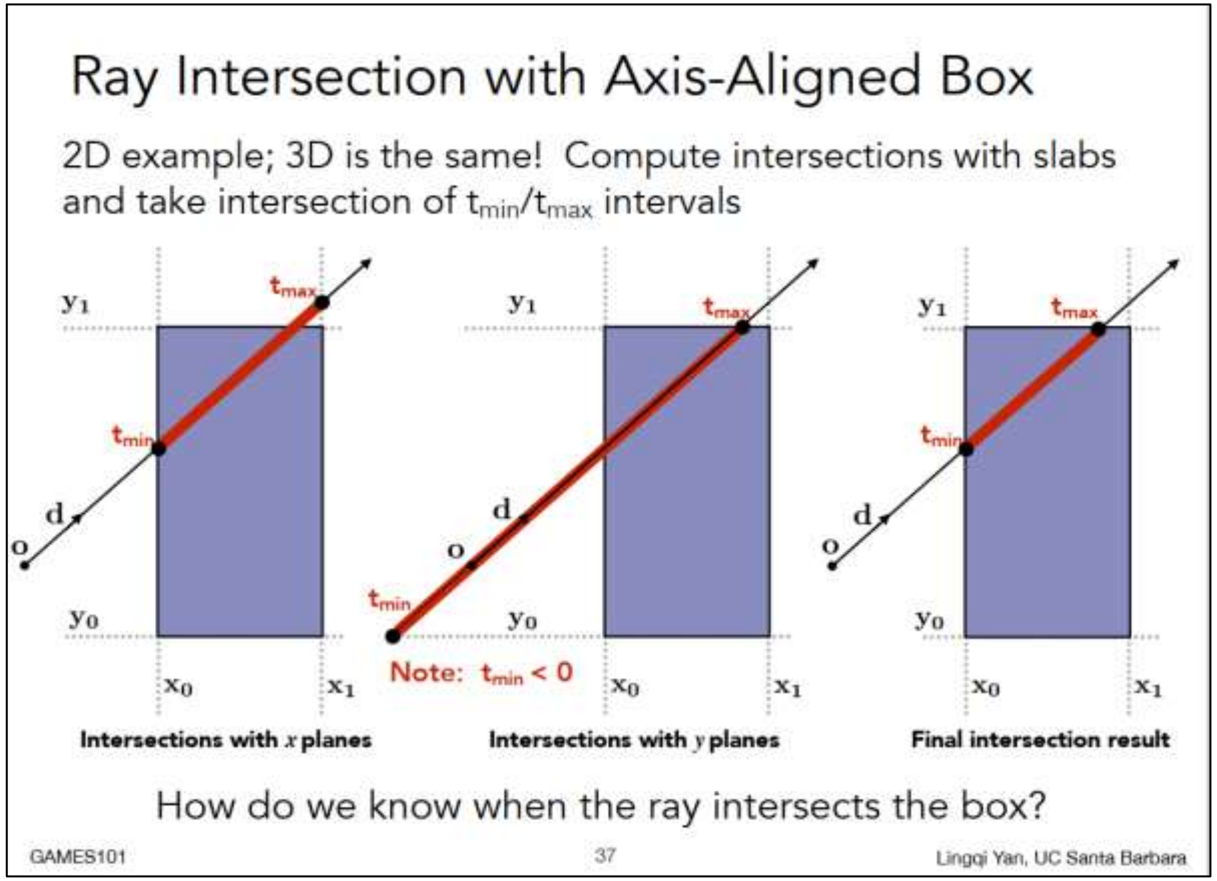
1 subtraction, 1 division

GAMES101 40 Lingqi Yan, UC Santa Barbara

Accelerating Ray-Surface Intersection(cont.)

➢**Ray-Intersection With Axis-Aligned Bounding Box(AABB)**

- 光线和AABB的求交计算简单！*参梁永栋Baskey裁剪算法求交点方法*



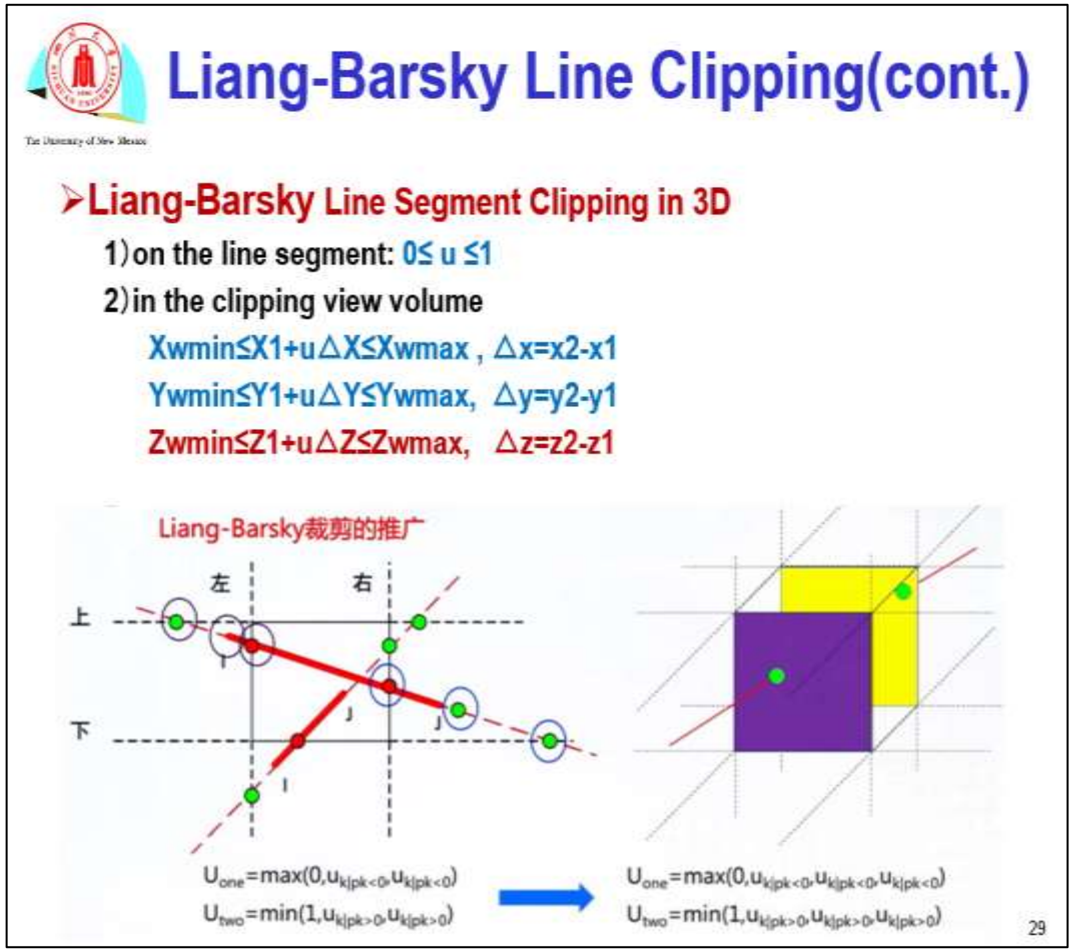
Ray Intersection with Axis-Aligned Box

2D example; 3D is the same! Compute intersections with slabs and take intersection of t_{min}/t_{max} intervals

Intersections with x planes **Intersections with y planes** **Final intersection result**

How do we know when the ray intersects the box?

GAMES101 37 Lingqi Yan, UC Santa Barbara



Liang-Barsky Line Clipping(cont.)

➢Liang-Barsky Line Segment Clipping in 3D

1)on the line segment: 0≤ u ≤1

2)in the clipping view volume

Xwmin≤X1+u△X≤Xwmax , △x=x2-x1

Ywmin≤Y1+u△Y≤Ywmax, △y=y2-y1

Zwmin≤Z1+u△Z≤Zwmax, △z=z2-z1

Liang-Barsky裁剪的推广

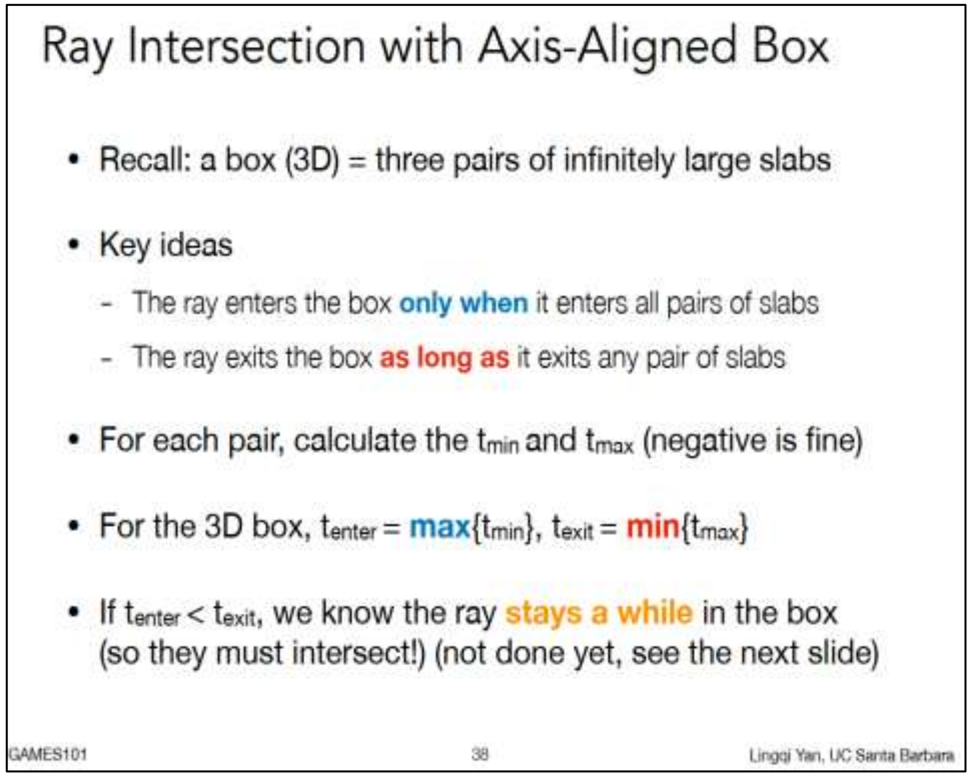
$U_{one}=\max(0,u_{k|pk<0},u_{k|pk<0})$
$U_{two}=\min(1,u_{k|pk>0},u_{k|pk>0})$

$U_{one}=\max(0,u_{k|pk<0},u_{k|pk<0},u_{k|pk<0})$
$U_{two}=\min(1,u_{k|pk>0},u_{k|pk>0},u_{k|pk>0})$

29

Accelerating Ray-Surface Intersection(cont.)

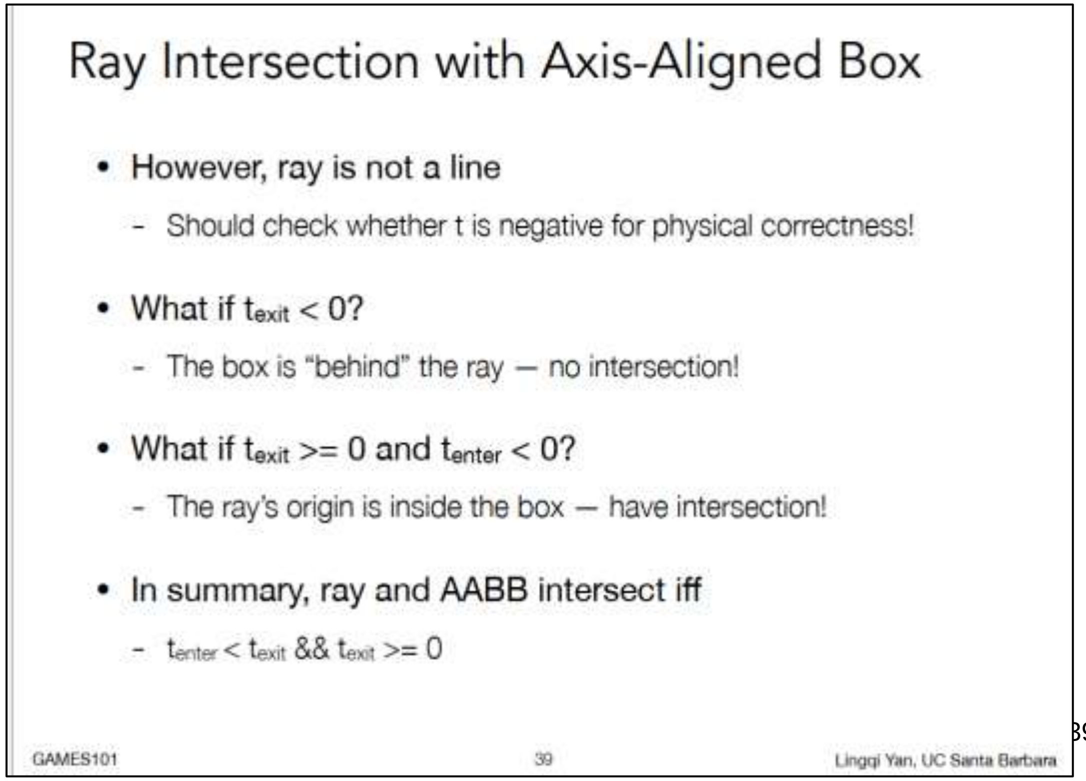
- **Ray-Intersection With Axis-Aligned Bounding Box(AABB)** ***(cont.)***
  - **与AABB包围盒有交的判定 iff ($t_{enter}<t_{exit}$ && $t_{exit}>=0$)**
    - 有交条件1：t_{enter} =max(t_{xmin} t_{ymin}, t_{zmin}), texit=min(t_{xmax},t_{ymax}, t_{zmax})，在包围盒内，需**tenter<texit**
    - 有交条件2：不是t_{enter}>=0，而是**t_{exit}>=0**



Ray Intersection with Axis-Aligned Box

- Recall: a box (3D) = three pairs of infinitely large slabs
- Key ideas
  - The ray enters the box **only when** it enters all pairs of slabs
  - The ray exits the box **as long as** it exits any pair of slabs
- For each pair, calculate the t_{min} and t_{max} (negative is fine)
- For the 3D box, t_{enter} = **max**{t_{min}}, t_{exit} = **min**{t_{max}}
- If $t_{enter} < t_{exit}$, we know the ray **stays a while** in the box (so they must intersect!) (not done yet, see the next slide)

GAMES101 38 Lingqi Yan, UC Santa Barbara



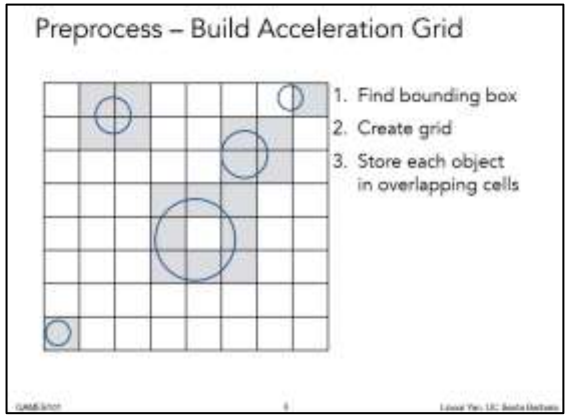
Ray Intersection with Axis-Aligned Box

- However, ray is not a line
  - Should check whether t is negative for physical correctness!
- What if $t_{exit} < 0$?
  - The box is "behind" the ray — no intersection!
- What if $t_{exit} >= 0$ and $t_{enter} < 0$?
  - The ray's origin is inside the box — have intersection!
- In summary, ray and AABB intersect iff
  - $t_{enter} < t_{exit}$ && $t_{exit} >= 0$

GAMES101 39 Lingqi Yan, UC Santa Barbara

Accelerating Ray-Surface Intersection(cont.)

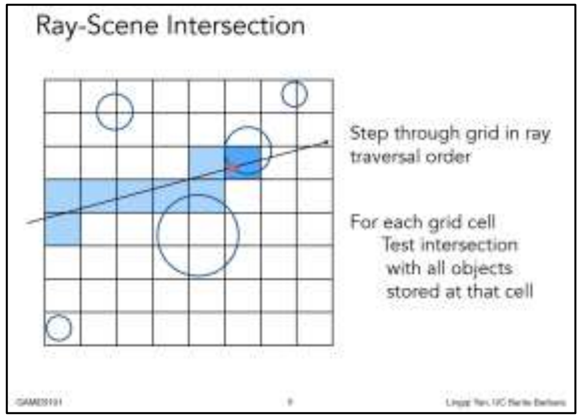
- ➢Using AABBs to accelerate ray tracing
  - ➢Uniform grids（均匀格子）
    - ➢适用画面中物体分布均匀，但是不适用于物体稀疏场景



Preprocess – Build Acceleration Grid

1. Find bounding box
2. Create grid
3. Store each object in overlapping cells

GAMES101 [illegible] Lingqi Yan, UC Santa Barbara



Ray-Scene Intersection

Step through grid in ray traversal order

For each grid cell
  Test intersection
  with all objects
  stored at that cell

GAMES101 9 Lingqi Yan, UC Santa Barbara

Uniform Grids – When They Work Well

Deussen et al.; Pharr & Humphreys, PBRT

Grids work well on large collections of objects that are distributed evenly in size and space

GAMES101 13 Lingqi Yan, UC Santa Barbara

Uniform Grids – When They Fail

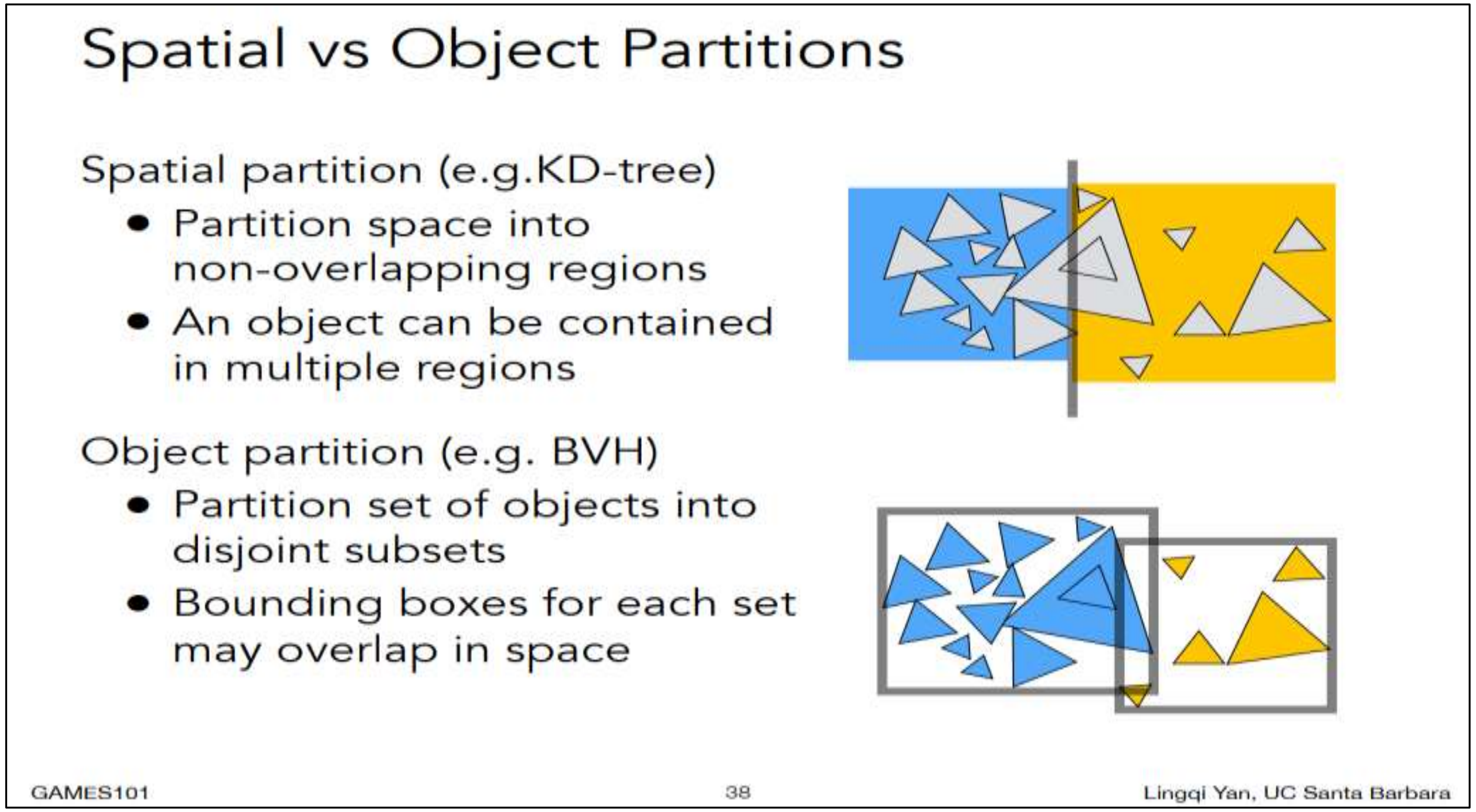
Jun Yan, Tracy Renderer

"Teapot in a stadium" problem

GAMES101 14 Lingqi Yan, UC Santa Barbara

Accelerating Ray-Surface Intersection(cont.)

- ➢Using AABBs to accelerate ray tracing(cont.)
  - ➢ Spatial Partitions(空间划分)和 Object partitions（对象划分）:



Spatial vs Object Partitions

Spatial partition (e.g.KD-tree)

- Partition space into non-overlapping regions
- An object can be contained in multiple regions

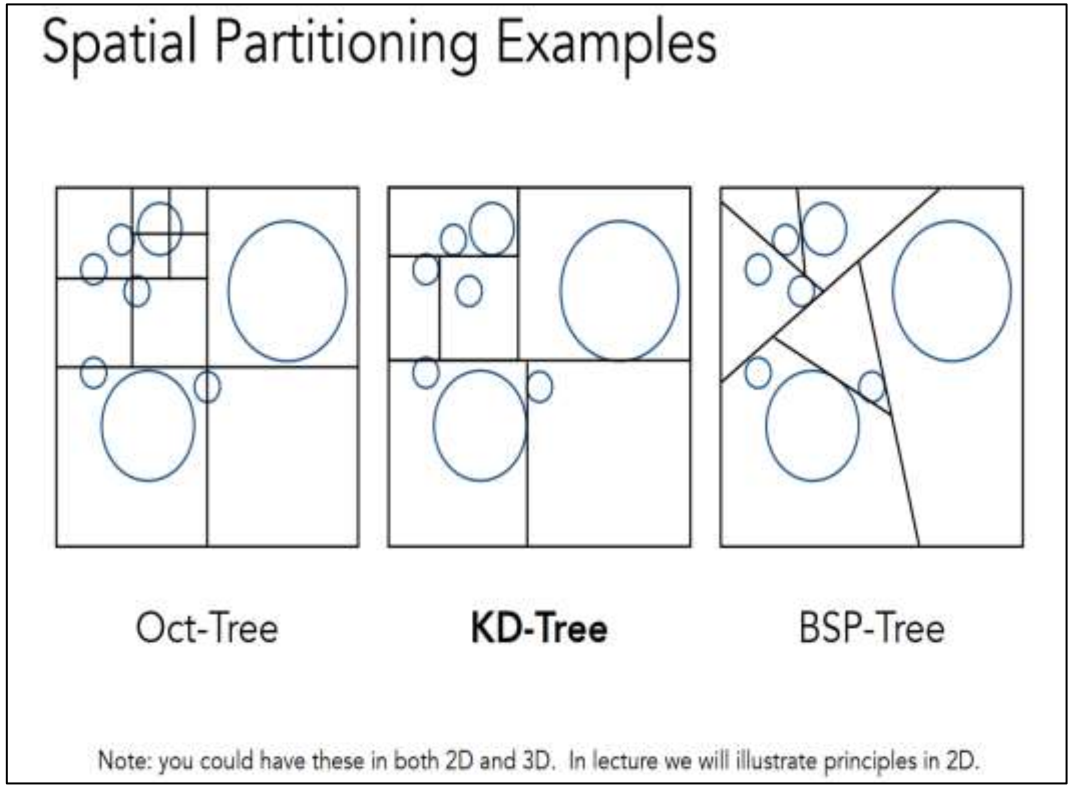
Object partition (e.g. BVH)

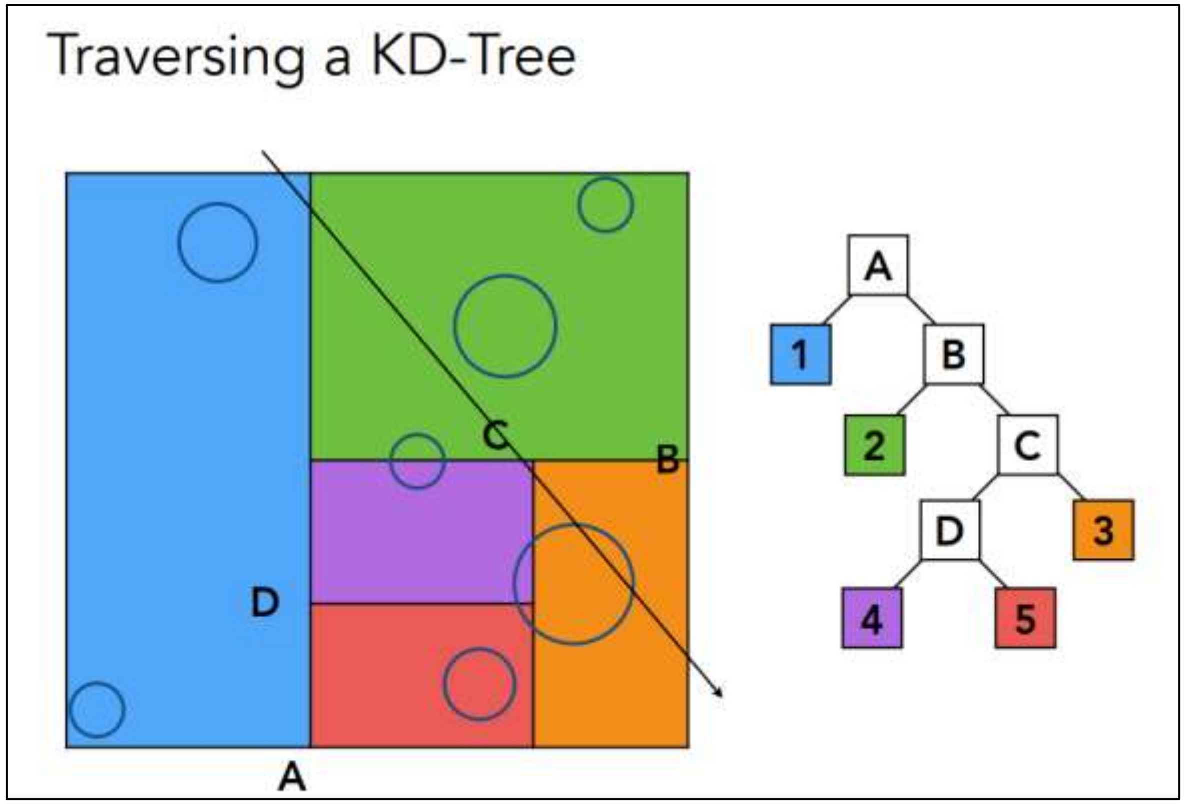
- Partition set of objects into disjoint subsets
- Bounding boxes for each set may overlap in space

GAMES101 38 Lingqi Yan, UC Santa Barbara

Accelerating Ray-Surface Intersection(cont.)

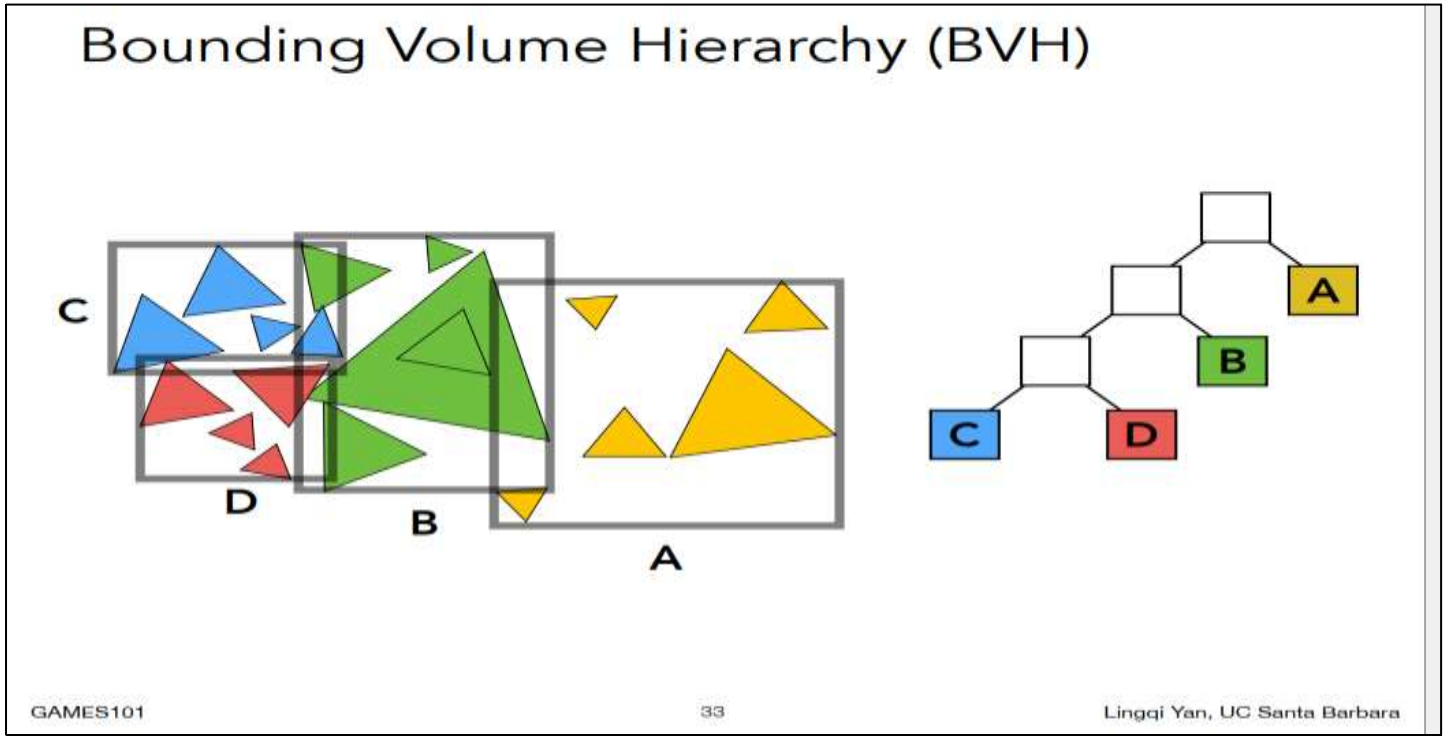
- **Using AABBs to accelerate ray tracing(cont.)**
  - **Spatial partitions（空间划分）：**八叉树，**KD树**，BSP树
    - 适合物体稀疏分布的空间，但物体可能出现在多个叶子节点里!


Spatial Partitioning Examples
Oct-Tree
KD-Tree
BSP-Tree
Note: you could have these in both 2D and 3D. In lecture we will illustrate principles in 2D.



Accelerating Ray-Surface Intersection(cont.)

- **Using AABBs to accelerate ray tracing(cont.)**
  - **Object Partitions(对象划分）：BVH（Bounding Volume Hierarchy层次包围盒）**
    - BVH通过递归地将场景中的几何对象包裹在包围盒中，每个包围盒外面再包裹一个更大的包围盒，最终形成一个根节点包裹整个场景的结构。
    - 这种结构适用于物体数量较多的场景，能够显著提高计算效率


Bounding Volume Hierarchy (BVH)

GAMES101 33 Lingqi Yan, UC Santa Barbara

Accelerating Ray-Surface Intersection(cont.)

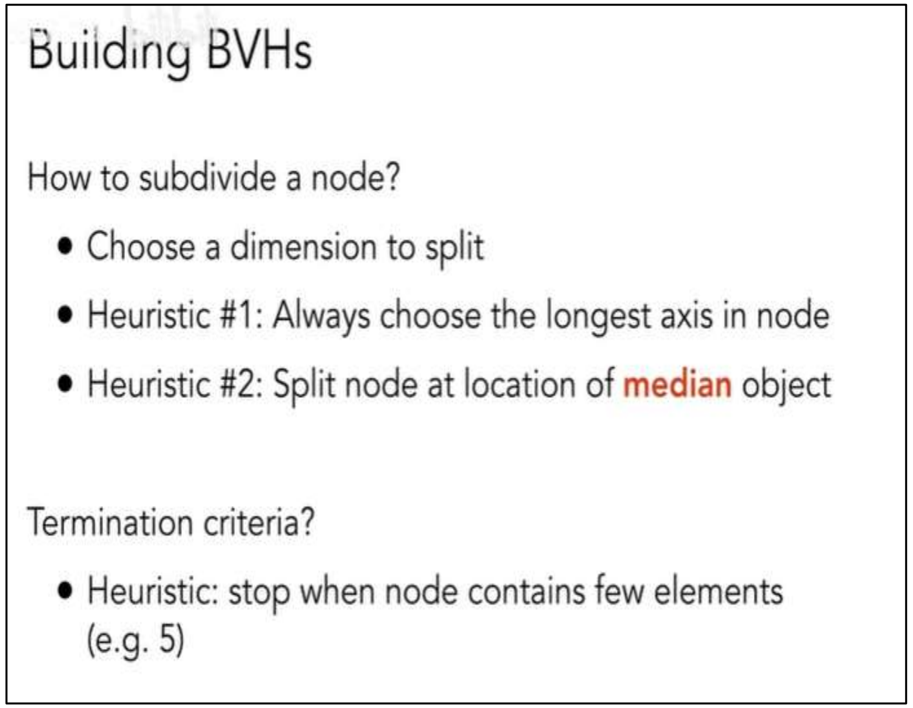
➢**Using AABBs to accelerate ray tracing(cont.)**

➢Object Partitions（对象划分）：BVH（层次包围盒）（cont.)

➢**Building BVHs：**

➢**递归拆分将几何体划分成两子集中，直到每个子集中只有少数几个几何体。**

➢**是二叉树，中间节点保存“包围盒”和”指针”，叶子节点存”实际几何体”**

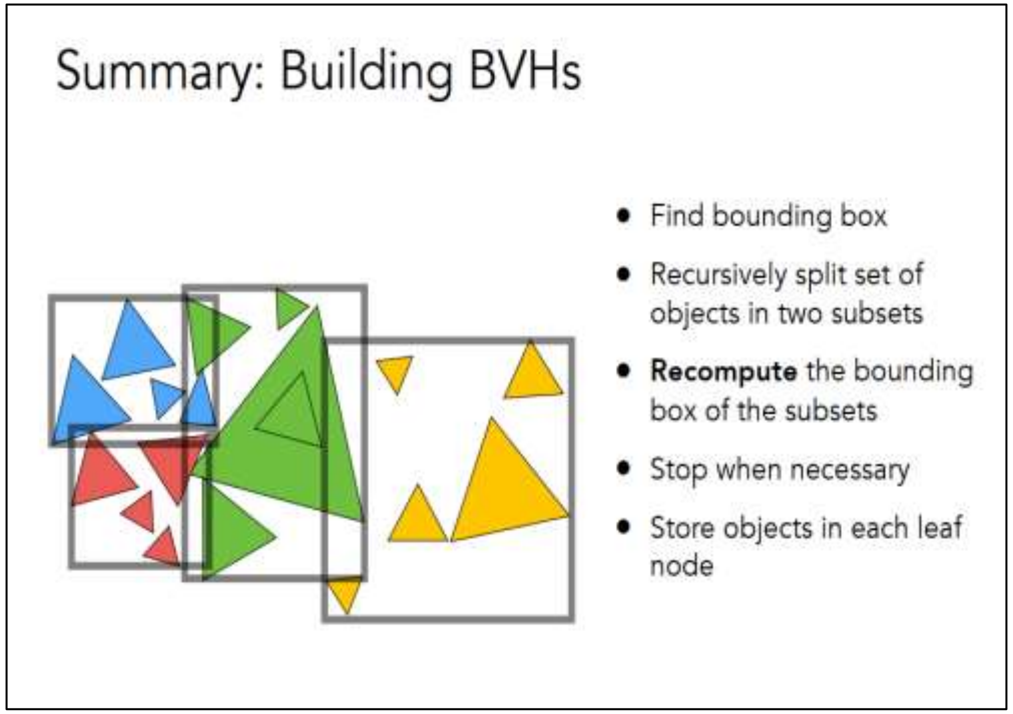

Building BVHs

How to subdivide a node?

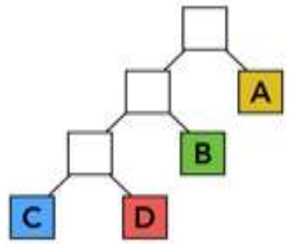
- Choose a dimension to split
- Heuristic #1: Always choose the longest axis in node
- Heuristic #2: Split node at location of **median** object

Termination criteria?

- Heuristic: stop when node contains few elements (e.g. 5)

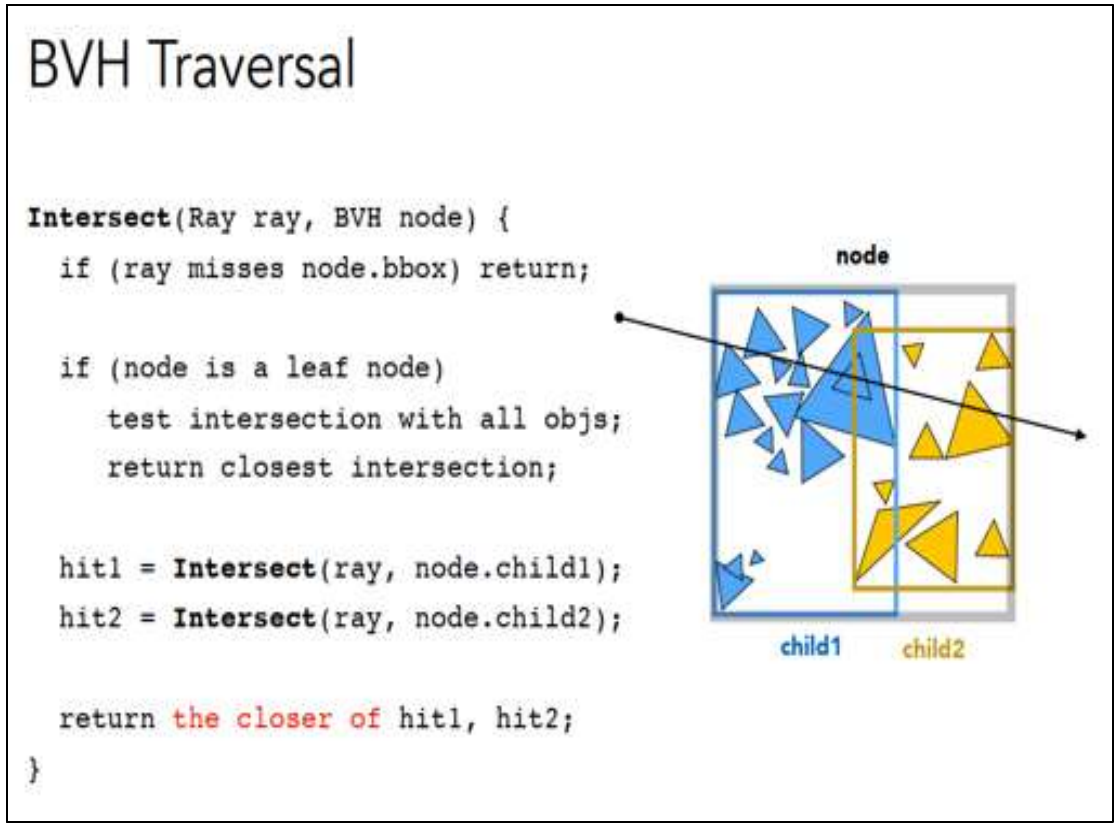

Summary: Building BVHs

- Find bounding box
- Recursively split set of objects in two subsets
- **Recompute** the bounding box of the subsets
- Stop when necessary
- Store objects in each leaf node



Accelerating Ray-Surface Intersection(cont.)

- **Using AABBs to accelerate ray tracing(cont.)**
  - Object Partitions（对象划分）：BVH（层次包围盒）（cont.)
    - **BVH Traversal**



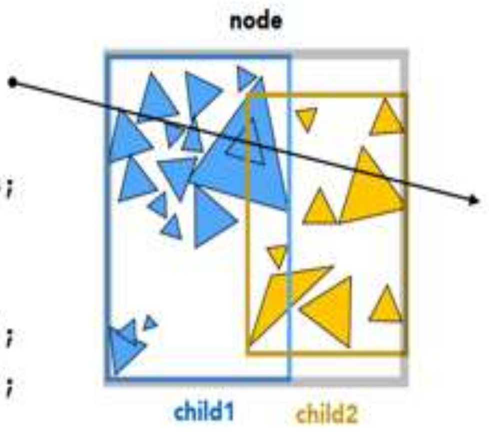
BVH Traversal

```
Intersect(Ray ray, BVH node) {
  if (ray misses node.bbox) return;

  if (node is a leaf node)
     test intersection with all objs;
     return closest intersection;

  hit1 = Intersect(ray, node.child1);
  hit2 = Intersect(ray, node.child2);

  return the closer of hit1, hit2;
}
```



Summary

- **Ray Tracing vs. Rasterization**
  - **光栅化渲染计算相对快，但是真实感不强，**
  - **光追渲染计算相对慢，但是真实感强**
- **Ray Casting**
  - **基本步骤：视线生成，与场景物体求交，最近交点处计算着色**
  - **带阴影步骤：视线生成，与场景物体求交，最近交点处判是否阴影而着色**
- **Whitted-Style（Recursive）**
  - **对半透明物体折射方向和镜面物体反射方向继续进行递归跟踪，直到结束条件。**
  - **可自动生成阴影，具有“部分的全局光照”的效果。**
- **Ray-Surface Intersection**
  - Implicit surfaces（隐式数学表示表面）
  - Mesh, Triangles, Plane（显示表示表面）
- **Accelerating Ray-Surface Intersection**
  - BB, AABB, Ray-AABB intersection, BVH Building and Traversal