why Ray Tracing?

- 光栅化很难处理全局光照（有trick，但是不能保证正确性）
  - Soft Shadows
  - Glossy reflection
  - Indirect Illumination
- 光栅化：快速近似、质量低
- 光线追踪：准确、非常慢
  - offline渲染
  - 实际生产中，渲染电影的一帧= ~ 10000CPU小时

Whitted-Style Ray Tracing

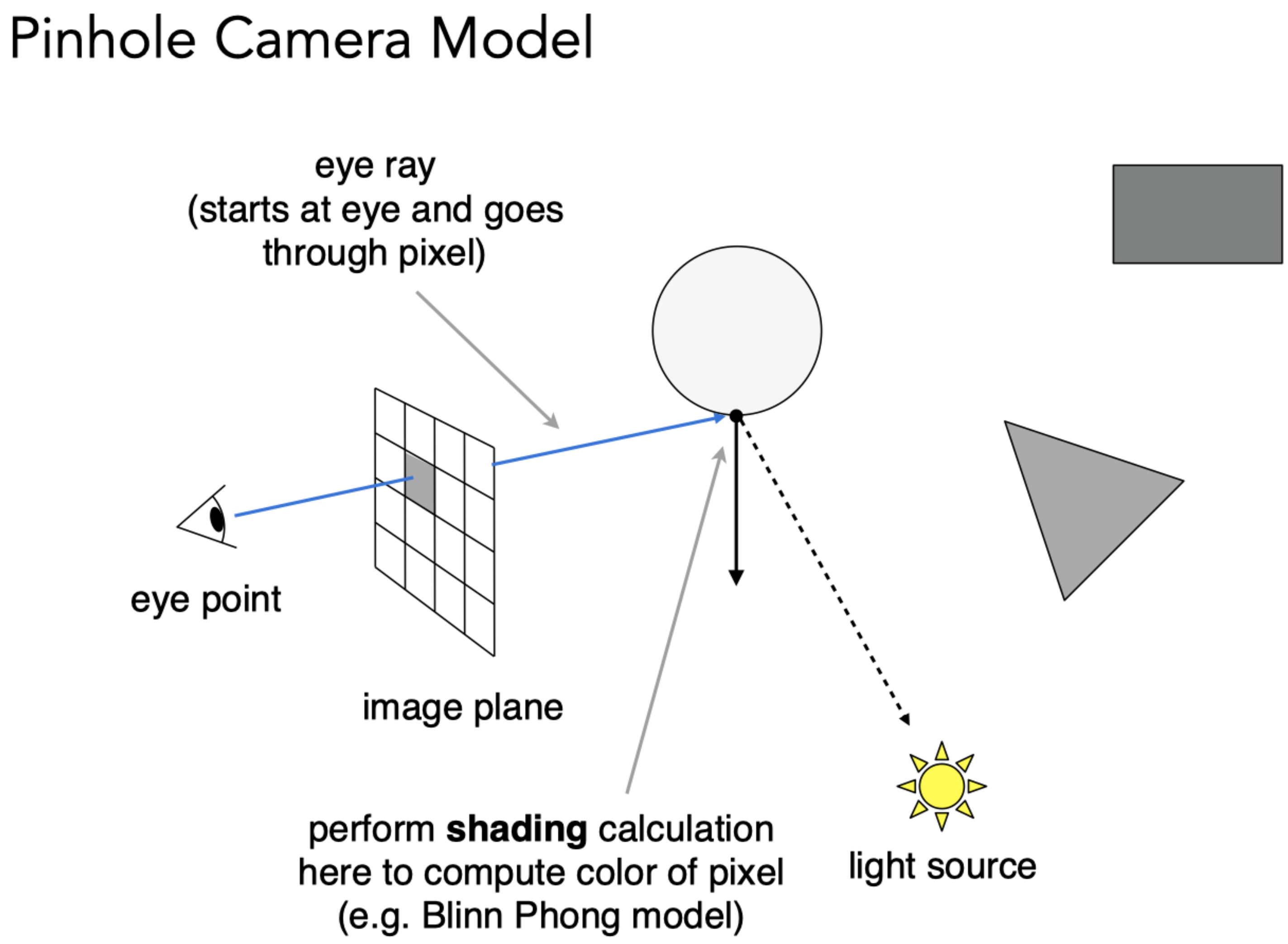
Ray Casting:

1. Generate an image by casting one ray per pixel

   从眼睛出发，经过像素点，射向场景

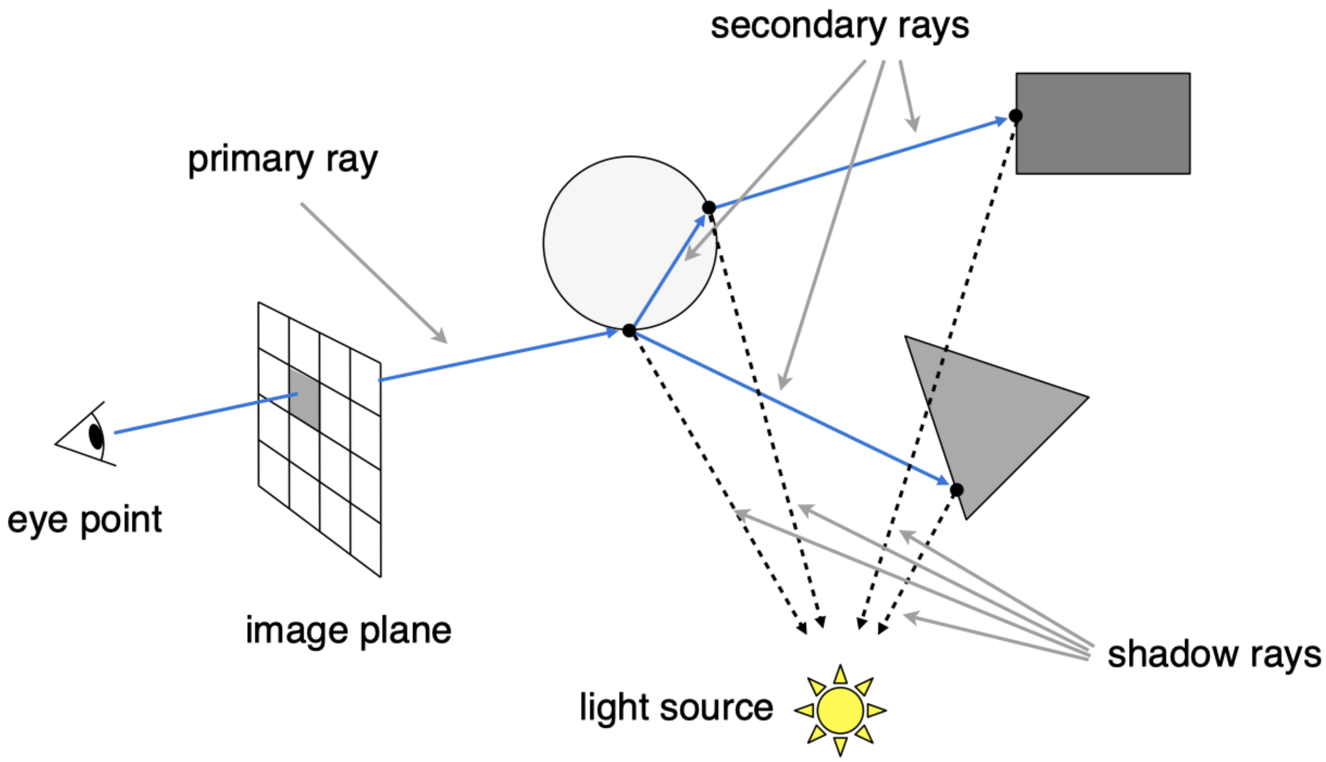
   判断最近碰撞物体，Shading计算颜色

2. Check for shadows by sending a ray to the light



光线弹射多次：Recursive (Whitted-Style) Ray Tracing

- Shading：每次折射点都计算一次颜色值，最后累加



Ray equation:

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

Ray Intersection(交点):

- with Sphere （c: center, R: 半径）

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

- 推广：隐式表面Geometry

$$\mathbf{p} : f(\mathbf{p}) = 0$$
$$f(\mathbf{o} + t\mathbf{d}) = 0$$

- 显式表面 - Triangle Mesh
  - 很重要，Rendering: visibility, shadows, lighting；Geometry: inside/outside test(内部射线与几何的交点数奇/偶)
  - intersect ray with each triangle
    - 太慢了
    - 加速：k-dtree, Bounding Box
  - 计算：光线与平面求交+交点是否在三角形内
    - 平面方程与光线方程组合，求t，确定为正实数
  - Möller Trumbore Algorithm: 计算交点
  - 三个未知数xyz对应三个方程--克莱默法则

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

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

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)

A faster approach, giving barycentric coordinate directly

判断合理：t为正

判断在三角形内：重心坐标三个系数均为正

Accelerating Ray-Surface Intersection

原始：每根光线和每个三角形求交（太慢！）

加速：包围盒kdtree

Bounding Volumes 包围盒

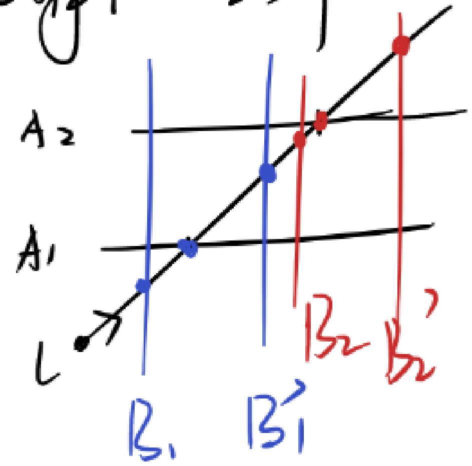
- bound complex object with a simple volume
- 碰不到包围盒，就肯定碰不到里面的物体

We often use an **Axis-Aligned Bounding Box (AABB)**

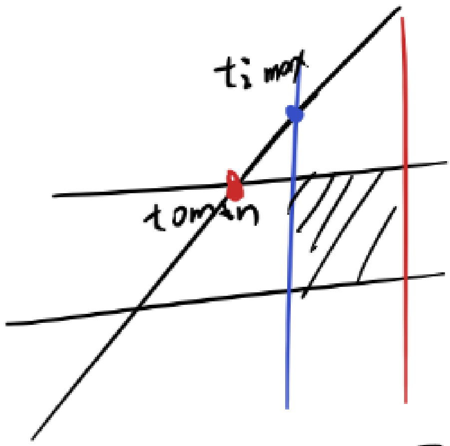
- 三对面形成的交集 xmin, xmax, ymin, ymax, zmin, zmax
- 判断求交：光线与三对面的交点：三组(tmin,tmax) 如果有交集（即有同时在三对面内的时间），则与盒子有交
- 原理：The ray enters the box only when it enters all pairs of slabs The ray exits the box as long as it exits any pair of slabs
- If t enter < t exit , we know the ray stays a while in the box (so they must intersect!)
- 需要检查各个t是否为正
- tenter = max{tmin}, texit = min{tmax}
- ray and AABB intersect iff (t enter< t exit && t exit >= 0)
- 垂直时计算更简单：每个t只需要一个`-`和一个`/`

- 光源在L上任意位置t相对大小不变

证明：2D中



∵ L的对称性，
∴ 任取一边起始
2条竖线B_1,B_2确定包围盒
比较L与A_1、B_1、B_1'的交点
确定进入时间t
发现 t_i max 为进入时间
t_o min 为出去时间



对于未穿过

$t_i - t_o < 0$

二维同理

How to use AABB to accelerate ray tracing?

- Uniform grids 均一网格
- Spatial partitions 空间划分

Uniform Spatial Partitions(Grids)

Preprocess - Build Acceleration Grid

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

Ray-Scene Intersection

- Step through grid in ray traversal order(Bresenham 线扫描)

- For each grid cell

  Test intersection with all objects stored at that cell

注意：

- 格子不能太密集/稀疏

- #cells = C * #objs
    - C ~ 27 in 3D
- Work in 当物体分布均匀时，还挺好用的
- Fail in "Teapot in a stadium" problem

Spatial Partitions - KD Tree

Oct-tree 八叉树（三维均匀切分）（与维数有关）

KD-Tree 每次只沿某一个轴划分 二叉树like

BSP-Tree 每次取一个方向（非横平竖直）将空间分为两部分 （会很麻烦）

KD-Tree Pre-Processing

- 划分空间，存入二叉树
- 内部节点存储以下信息

  split axis: x-, y-, or Z-axis split position: coordinate of split plane along axis children: pointers to child nodes

- 叶节点存储

  list of objects

Traversing a KD-Tree

- 光线从根结点开始向下遍历，若有交点则深入（可能与其子节点都有交点，继续判断），若无交点则离开（不可能与其子节点有交点），碰到叶子节点则与其中所有物体求交

问题1：三角形与Bounding Box的包含情况求解困难（可能出现三角形三个顶点都不在Box内，三角形却有一部分在Box内的情况）

最近渐渐不用KD-Tree了，上述原因为其中一个原因

问题2：一个物体可能存在于多个叶子节点中

Object Partitions & Bounding Volume Hierarchy (BVH)

以object为单位划分空间

二叉树，两个子节点分别存两部分物体的AABB

一个物体只可能出现在一个包围盒中

如何划分很有讲究，不好的划分会使包围盒重合，降低效率

1. Find bounding box
2. Recursively split set of objects in two subsets
    - Choose a dimension to split

- Heuristic #1: Always choose the longest axis in node
- Heuristic #2: Split node at location of median object(中位数)
  - 中位数

3. Recompute the bounding box of the subsets

4. Stop when necessary

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

5. Store objects in each leaf node

Internal nodes store

- Bounding box
- Children: pointers to child nodes

Leaf nodes store

- Bounding box
- List of objects

Nodes represent subset of primitives in scene

- All objects in subtree

动态场景不可

Traversal:

- 如果miss，直接返回
- 如果已经到了叶子节点，和其中的object求交找最近
- 如果hit，和下面两个子节点求交，返回最近

Spatial vs Object Partitions

- 后者目前更多应用

Whitted Style Ray Tracing到此结束