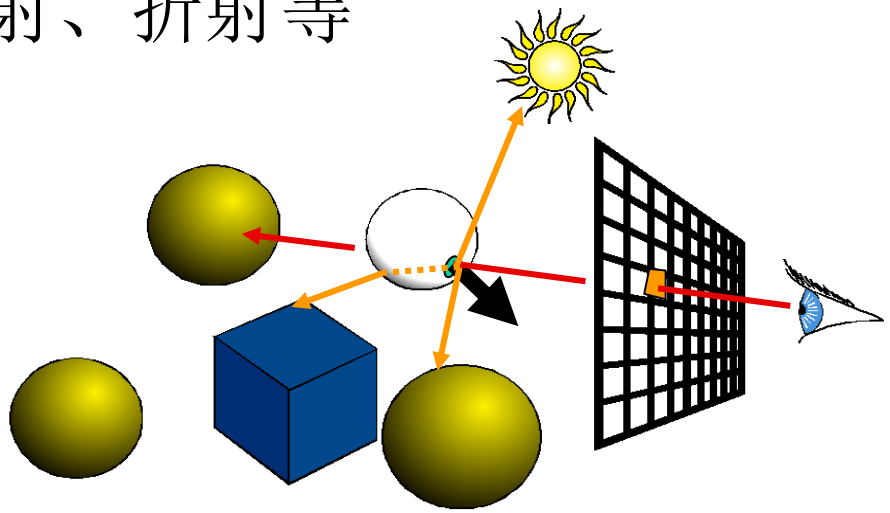
计算机图形学基础

胡事民

清华大学计算机科学与技术系

光线跟踪加速 (Ray Tracing Acceleration)

- 光线跟踪
  - 光线跟踪的发明对图形学具有里程碑式的意义；使用光线跟踪技术，我们可以绘制含有许多特殊视觉效果的真实感图形，如阴影、透明、半透明、反射、折射等

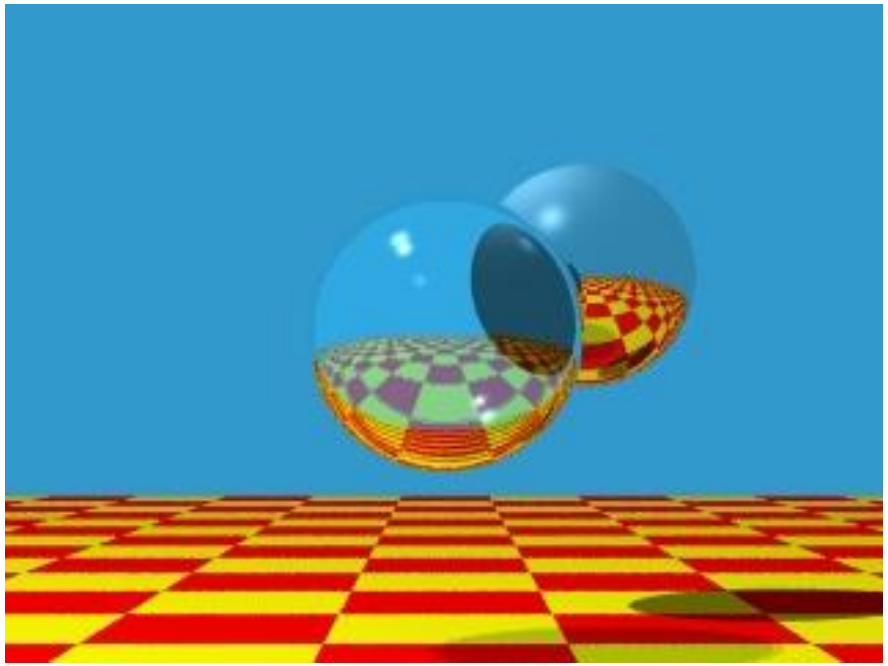


递归的光线跟踪算法

```
IntersectColor( vBeginPoint, vDirection)
{
   for each light
        Color = ambient color;
        Determine IntersectPoint;
        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;
}
```

- 光线跟踪的效果示例：



光线跟踪加速

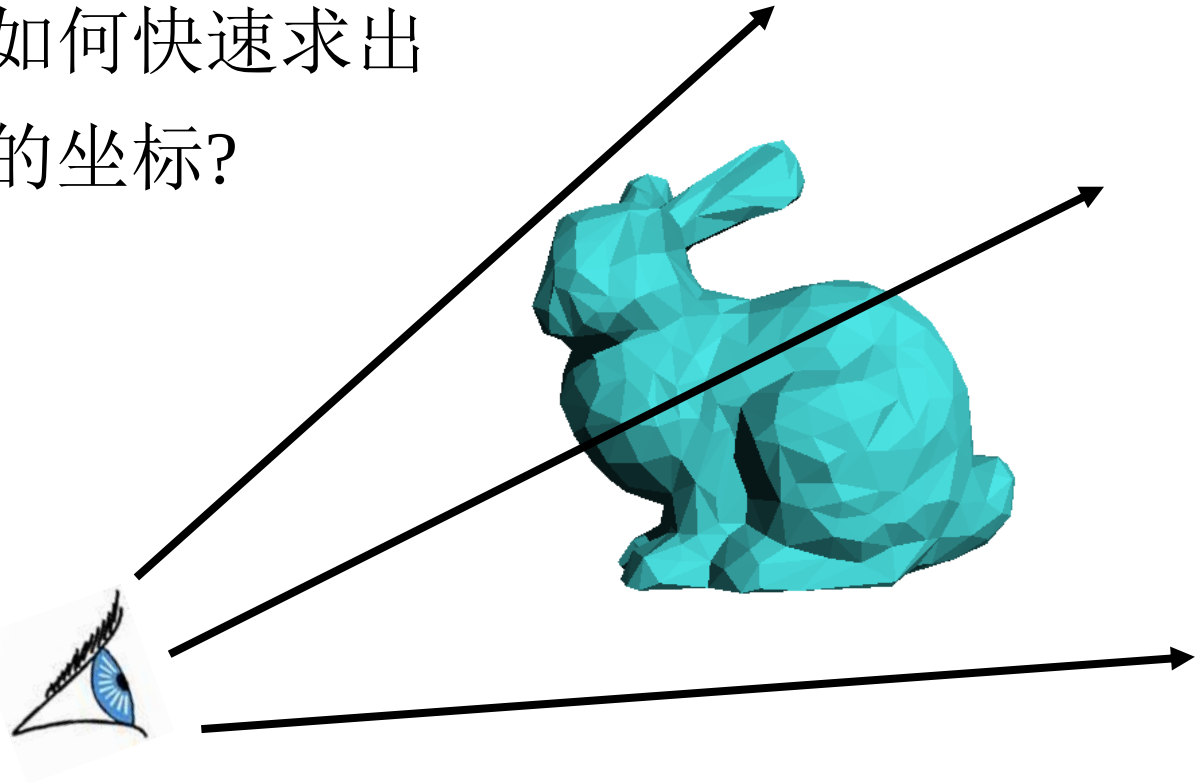
- 研究加速的动机：光线跟踪算法效率不足：
  - 光线跟踪算法的时间和空间复杂度都很高
  - 大量的时间被消耗在可见性判断和求交测试这些几何运算上

光线跟踪加速

- 如何加速?
  - 考虑采用空间数据结构，如：
    - 层次包围体 (Bounding Volume Hierarchys)
    - 均匀格点 (Uniform Grids)
    - 四叉树/八叉树 (Quad tree/Octree)
    - 空间二分树 (K-d tree/BSP tree)
  - 良好的空间数据结构可以使光线跟踪算法加速 10-100 倍

光线求交

- 给定一个模型和一个光线方向，如何快速判定光线是否与模型相交？
- 如果相交，如何快速求出第一个交点的坐标?

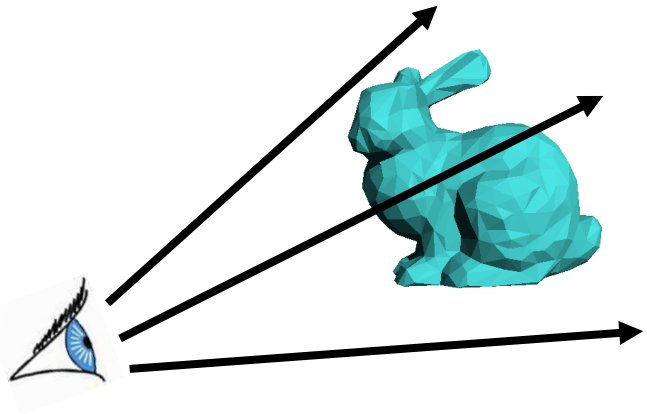


光线求交

- 简单的“笨”方法：

```
boolean IsIntersect(Ray r, Model m)
  For each triangle t in m
    if  IsIntersect(t,r)
        return true
  End For
  return false
```

  - 显然这样的求交判定十分耗时
  - 时间复杂度为 O(*n*)：其中 *n* 是模型包含的三角面片数

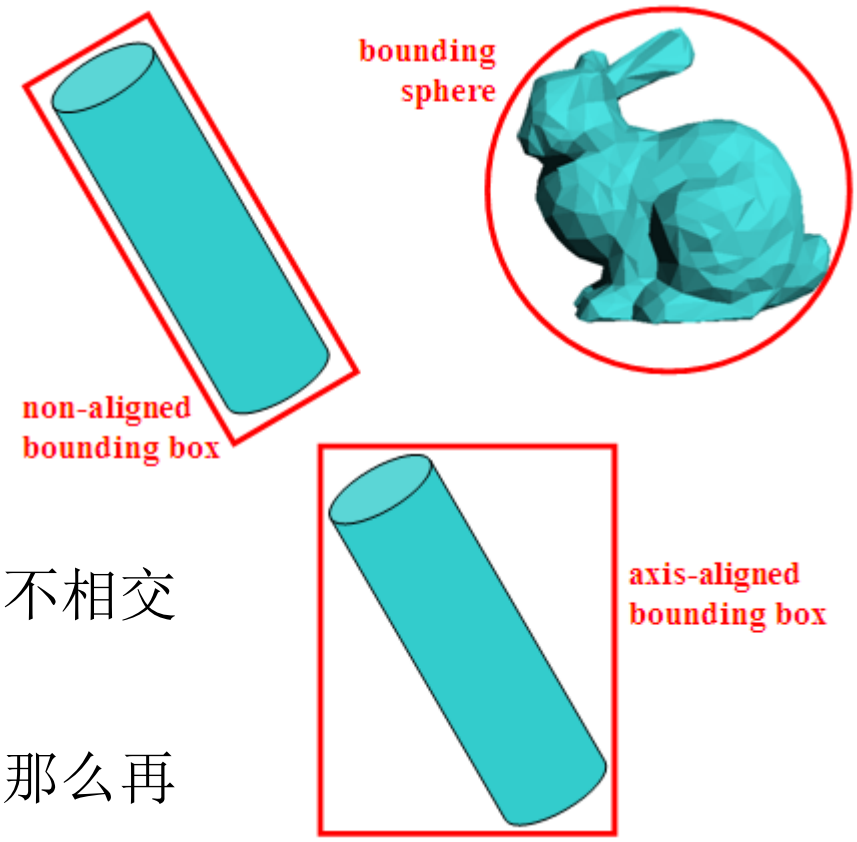


包围体 (Bounding Volumes)

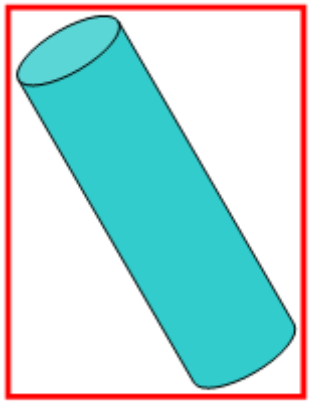
- 把难以进行求交判定的物体，用容易进行求交判定的物体给包围 (Wrap) 起来：
  - 例如，将一个复杂的三角网格模型包围在一个长方体的内部，如果光线不与长方体相交，则一定不会与模型相交
- 最常使用的包围体类型：
  - 长方体包围盒 (bounding box)
  - 包围球 (bounding sphere)
  - 包围盒可以是平行于坐标轴放置的，也可以不是

包围体 (Bounding Volumes)

- 包围体示例：
  - 在对光线和物体进行相交检测之前，先对光线和物体的包围体做相交检测：
    - 如果光线和包围体不相交，那么和物体一定也不相交 (Easy reject)
    - 如果光线和包围体相交，那么再进行光线和物体的相交检测



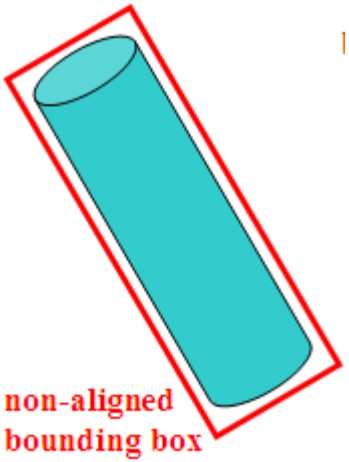
平行于坐标轴的包围盒生成



- Axis-Aligned Bounding Box (AABB)：
  - 包围盒 (Bounding Box) 的三组棱边
    分别平行于世界坐标的 x y z 轴
- 构造方法
  - 对 x 轴，计算物体的最小和最大 x 坐标：**x_min, x_max**
  - 类似地计算：**y_min, y_max; z_min, z_max**
  - 长方体：**[x_min, x_max] * [y_min, y_max] *[z_min, z_max]**
    就是物体的平行于坐标轴的包围盒 (AABB)

只需遍历顶点，可求得！

非平行于坐标轴的包围盒



- Non-Axis-Aligned Bounding Box：
  - 也被称为有向包围盒 (Oriented Bounding Box, OBB)
  - OBB 是比 AABB 更优的解，其匹配程度往往比 AABB 更高
  - 通常会使用一些迭代算法来近似计算OBB

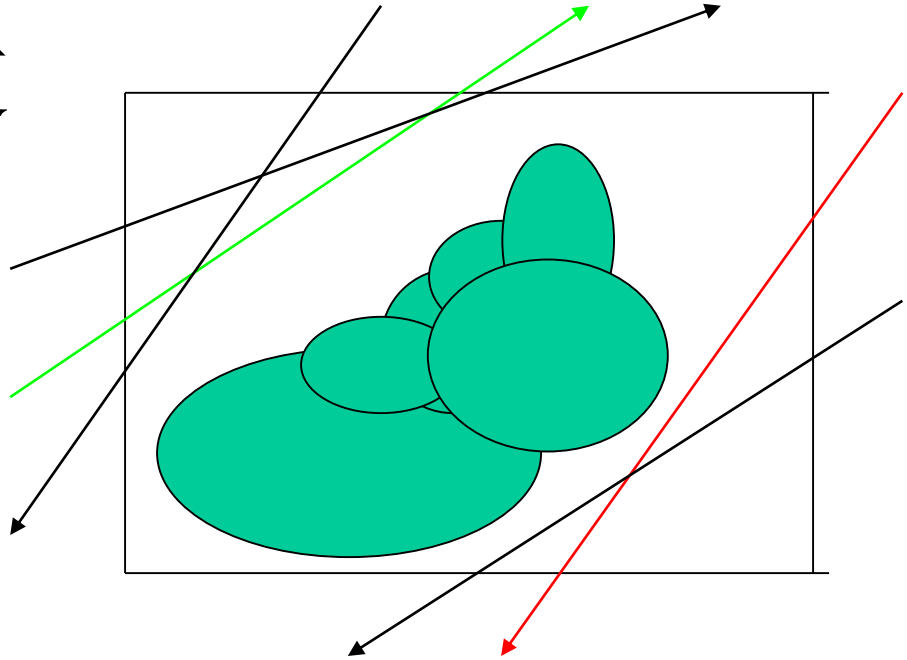
非平行于坐标轴的包围盒

- 1986年，Kay和Kajiya提出：长方体包围盒具有包裹景物不紧的特点
- 根据景物的实际形状选取n组不同方向的平行平面包裹一个景物或一组景物来作为层次包围盒

ACM COONS AWARD, 2011

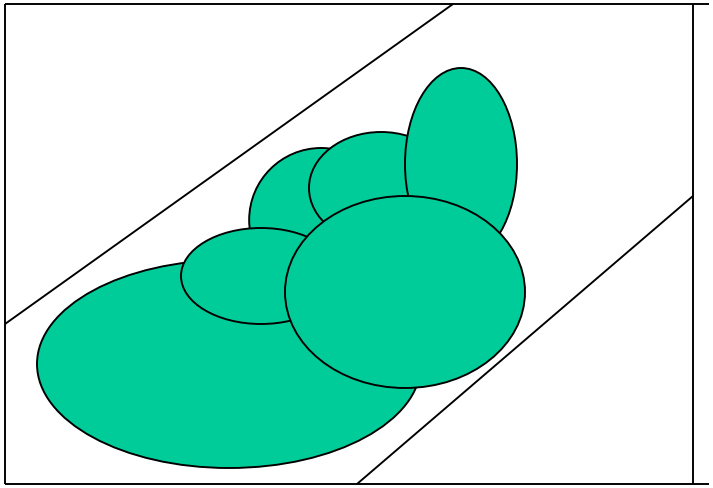
非平行于坐标轴的包围盒

- 包围盒不够紧
- 示意图: 光线与物体并无相交，却与包围盒相交



非平行于坐标轴的包围盒

- 示意图2：如果选取合适的非平行与坐标轴的包围盒，可以“紧”很多



非平行于坐标轴的包围盒

- 令3D空间中的任一平面方程为

$$Ax + By + Cz - d = 0$$

- 不失一般性，设$N_i = (A, B, C)$为单位向量
- 上式定义了一个以N_i为法向量，并且与坐标原点相距d的平面。若法向量保持不变，d为自由变量，那么我们就定义了一组平面。

非平行于坐标轴的包围盒

- 对一个给定的景物，必定存在两个平面将景物夹在中间,不妨记d的值为d_i^{near}, d_i^{far}
- 用几组平面就可以构成一个较为紧致的包围盒
- Kay和Kajiya选取n组方向为平面法向构造包围盒，并取$n < 5$
- 思考：对多面体模型和隐式曲面，如何确定

$$d_i^{near}, d_i^{far}$$

非平行于坐标轴的包围盒

- 对多面体模型，在场景坐标系中可将多面体的所有顶点投影到N_i方向，并计算与原点距离的最小值和最大值，令为d_i^{near}, d_i^{far}

非平行于坐标轴的包围盒

- 对隐函数曲面体$F(x,y,z)=0$，假设在场景坐标系中隐函数曲面体上的点(x,y,z)在方向N_i上的投影为

$$f(x,y,z)=Ax+By+Cz$$

- 根据$f(x,y,z)$的定义，我们必须求d_i^{near}, d_i^{far} 在约束条件

$$F(x,y,z)=0$$

下的极大值和极小值，可以用Lagrange乘子法计算。

包围球

- 包围球 (bounding sphere)：
  - 一个包围球仅包含两组参数，即球心和半径，当旋转物体时，包围球不旋转
  - 然而，计算理想的最优包围球是困难的
  - 下面介绍一个计算 n 个点的最优包围球的近似方法，它具有 O(n) 的时间复杂度，通常来说，该近似算法生成的包围球比理想包围球大 5% 左右

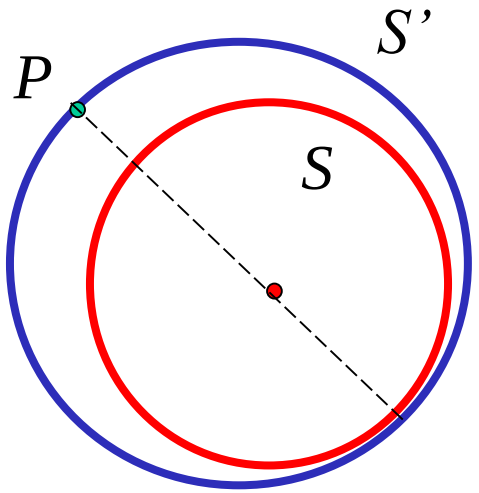
包围球

- 算法包含两个步骤：
  - 首先：遍历所有的 n 个点：
    找到以下三组点：
    - 最小 x 坐标的点，最大 x 坐标的点
      最小 y 坐标的点，最大 y 坐标的点
      最小 z 坐标的点，最大 z 坐标的点

    计算这三组点的距离，选择两点间距离最大的一组点，然后将连接这两点的线段作为包围球的直径，将该包围球作为算法的初始值

包围球

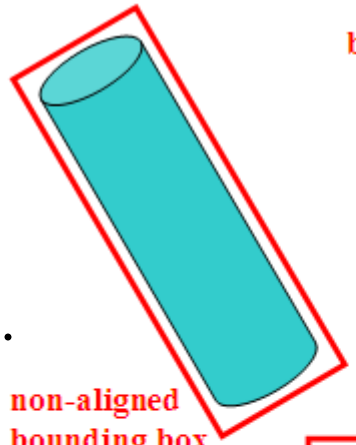
- 对所有 n 个点进行遍历：每次发现有点 P 位于当前包围球 S 的外部，那么将 S 修改为球 S’：S’ 经过点 P，且 S’ 与 S 内切
- 如图所示，原先的球 S (红色) 被修改为新的球 S’ (蓝色)：



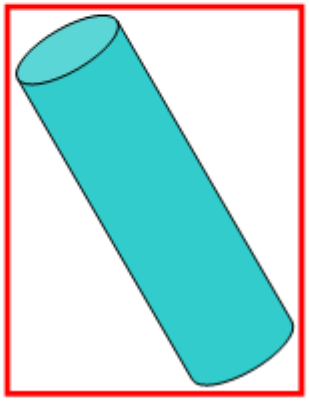
- 迭代若干次
- 求包围球，计算几何问题。

包围体 (Bounding Volumes)

- 包围体 (Bounding Volume)：
  - 除了用于光线跟踪加速，
    包围体在图形学中还有
    许多其他应用，如：
    隐藏表面消除、碰撞检测等…



non-aligned
bounding box



axis-aligned
bounding box

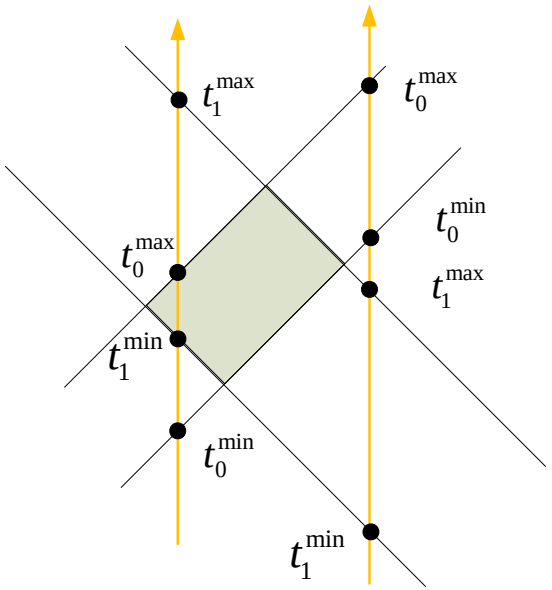
Ming Lin

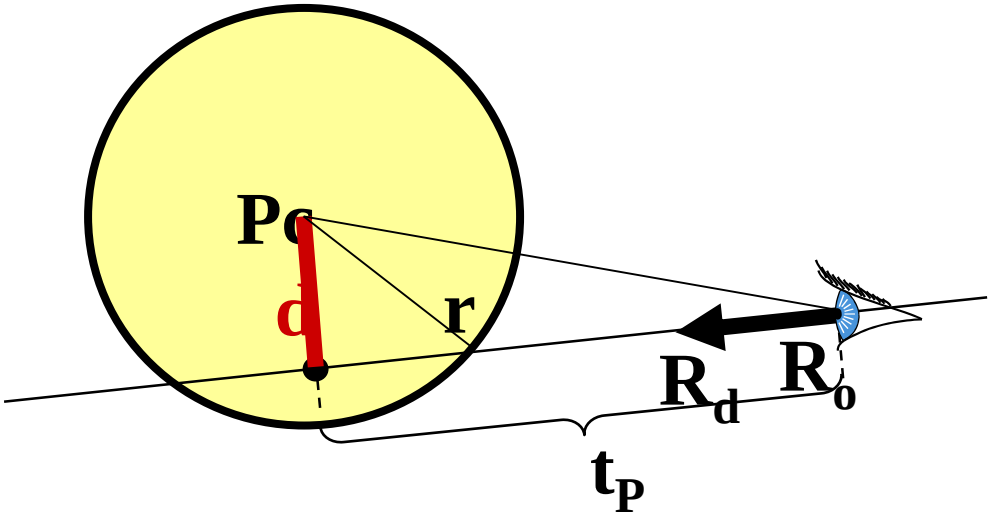
Elizabeth Stevinson Iribe Chair Professor, University of Maryland at College Park
Verified email at cs.umd.edu - Homepage
Computer Animation Computer Graphics Geometric Modeling Robotics Virtual Reality

TITLE	CITED BY
OBBTree: A hierarchical structure for rapid interference detection S Gottschalk, MC Lin, D Manocha Proceedings of the 23rd annual conference on Computer graphics and ...	2859
Collision and Proximity Queries MC Lin, D Manocha, CD edited by Toth, J O'Rourke, JE Goodman Handbook of discrete and computational geometry, 787-808	1387 *
I-collide: An interactive and exact collision detection system for large-scale environments JD Cohen, MC Lin, D Manocha, M Ponamgi Proceedings of the 1995 symposium on Interactive 3D graphics, 189-ff.	982

包围体 (Bounding Volumes)

- 包围体的光线求交：
  - 主要为光线与长方体包围盒以及包围球的求交，这些在之前的课程中都已经有所讨论





多选题 1分

设置

图形学中最常见的包围盒是

A 球，求交测试容易

B 长方体AABB，求交测试容易

C 椭球，包围物体较紧致

D 平行2N面体，包围物体较紧致

提交

层次包围体 (Bounding Volume Hierarchy)

- 包围体的局限性：
  - 假设有 n 个物体，每个物体使用一个单独的包围体，那么光线与所有物体的相交检测仍然需要 O(n) 的时间复杂度
- 包围体一个自然扩展就是使用层次结构来进行分层细化，也就是层次包围体：Bounding Volume Hierarchy (BVH)

层次包围体 (Bounding Volume Hierarchy)

- 给定场景中所有物体的包围体，可以将所有这些包围体作为叶子节点，构建一个基于包围体的树状结构，也就是 BVH
- 对于 BVH 的每个内部节点 v，v 对应的包围体是v 的所有子孙节点所对应的包围体的并集
- 采用自底向上的顺序，我们可以从场景中的每个物体开始逐渐构建整个场景的 BVH

层次包围体 (Bounding Volume Hierarchy)

- 使用 BVH，可以快速计算光线与场景中物体的求交问题：
  - 首先检测光线是否与 BVH 根节点对应的包围体相交；如果不相交，则光线与所有物体均不相交(Easy Reject)
  - 否则，对 BVH 的所有节点递归地判断光线与哪些节点对应的包围体相交

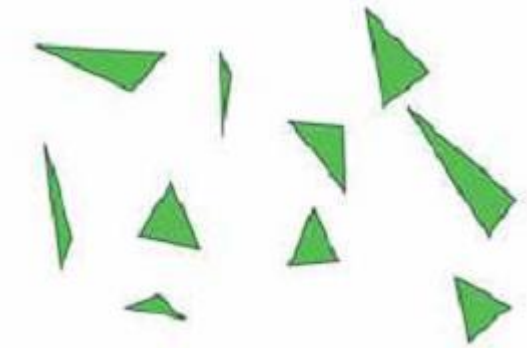
层次包围体 (Bounding Volume Hierarchy)

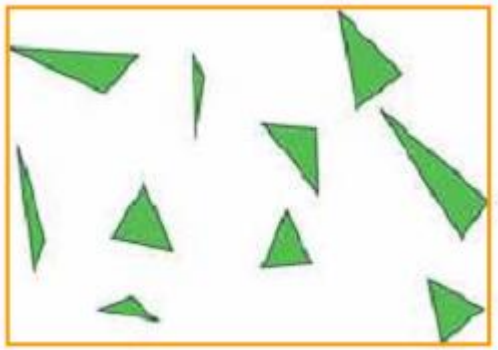
- BVH 的一个有趣的性质是：

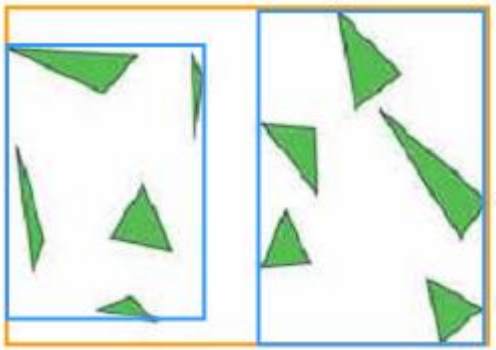
  虽然每个节点的包围体完全包含其所有子节点对应的的包围体；然而其不同子节点对应的包围体却有可能相互相交

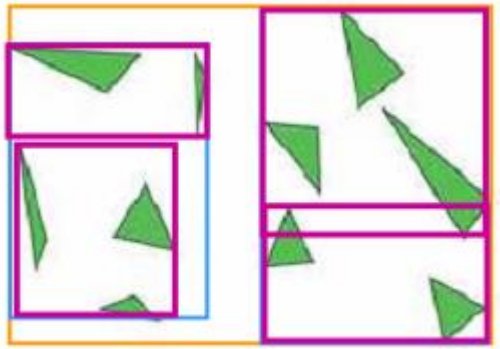
层次包围体 (Bounding Volume Hierarchy)

- 一个例子 (2D 情况)：
  - 右图中有若干三角形 (物体)：
  - 下图给出了它们的层次包围体 (BVH)：



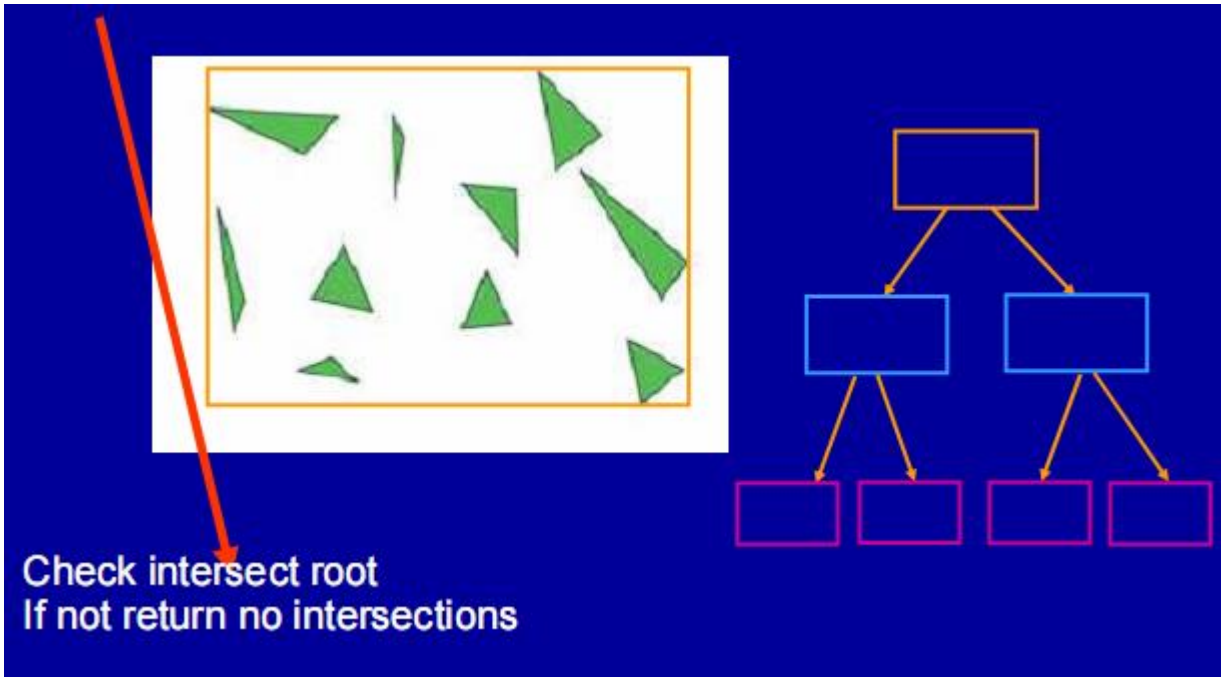






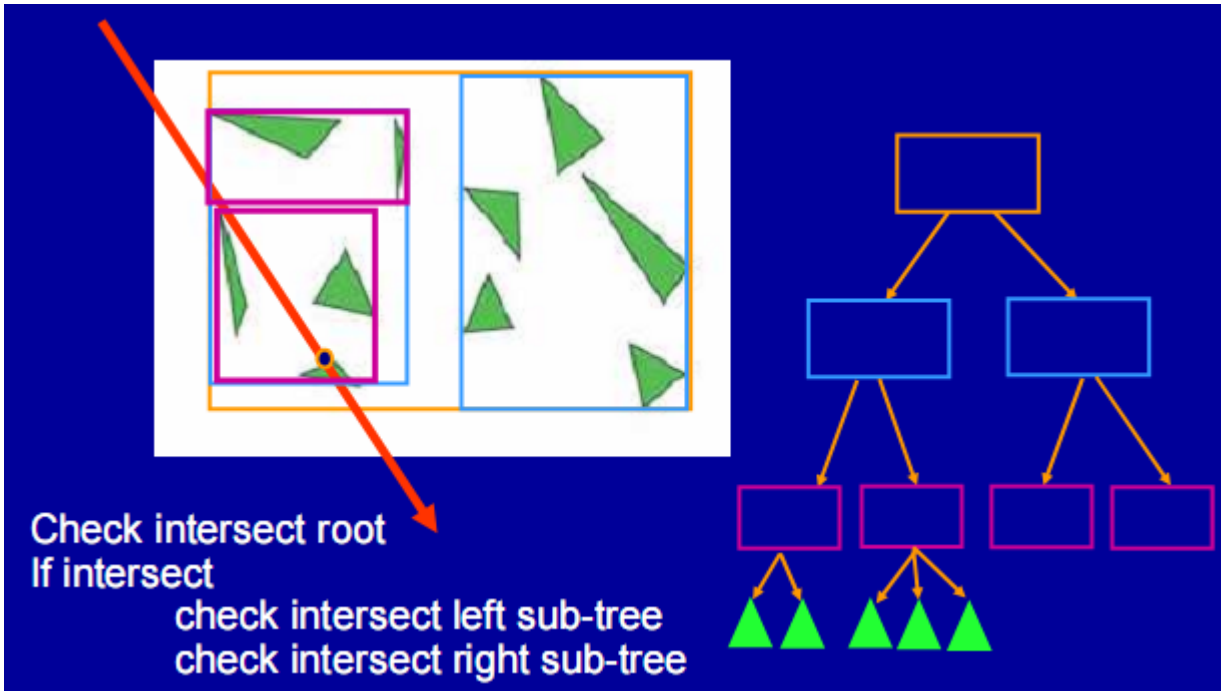
层次包围体 (Bounding Volume Hierarchy)

- 举例：利用 BVH 判断光线与场景中物体的交



层次包围体 (Bounding Volume Hierarchy)

- 举例：利用 BVH 判断光线与场景中物体的交。

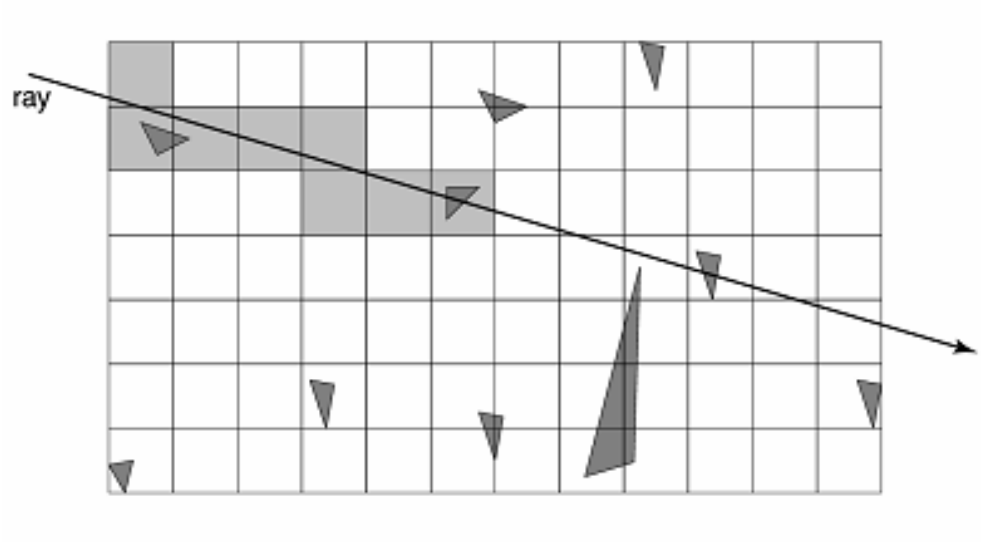


层次包围体 (Bounding Volume Hierarchy)

- 层次包围体的好处：
  - 如果使用恰当的包围体及层次结构，那么 BVH 可以获得非常好的效果
  - 对包含 n 个物体的场景，时间复杂度可以由 O(n) 降为 O(log n)
  - 根据情况，我们还可以对每个节点灵活选择不同的包围体类型 (长方体包围盒、包围球等)

空间数据结构

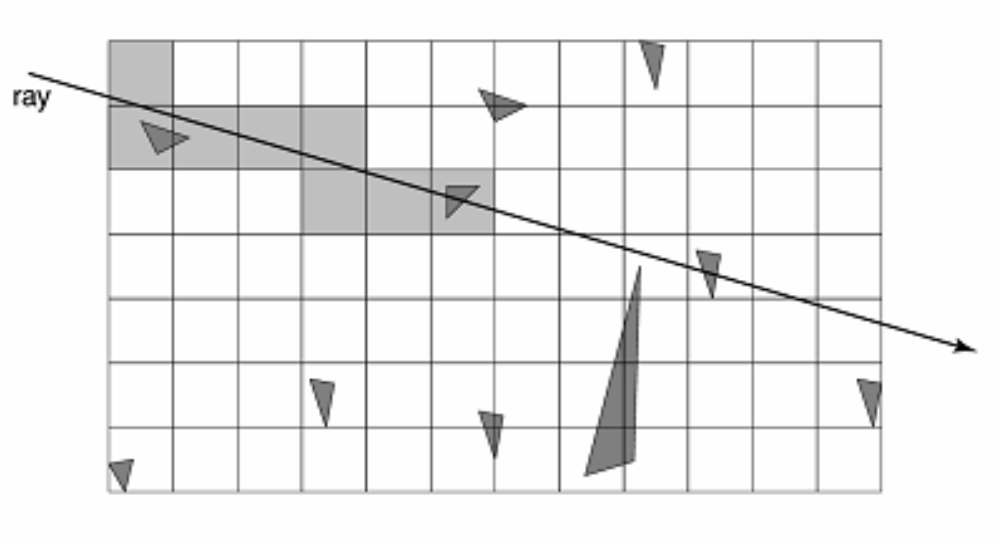
- 为何采用空间数据结构？



  - 关键：让光线变聪明

均匀格点 (Uniform Grids)

- 铺满整个空间的 3 维格点 (立方体) 阵列：
  - 对每一个格点，将所有与其相交的面片记录下来，存储为一个列表

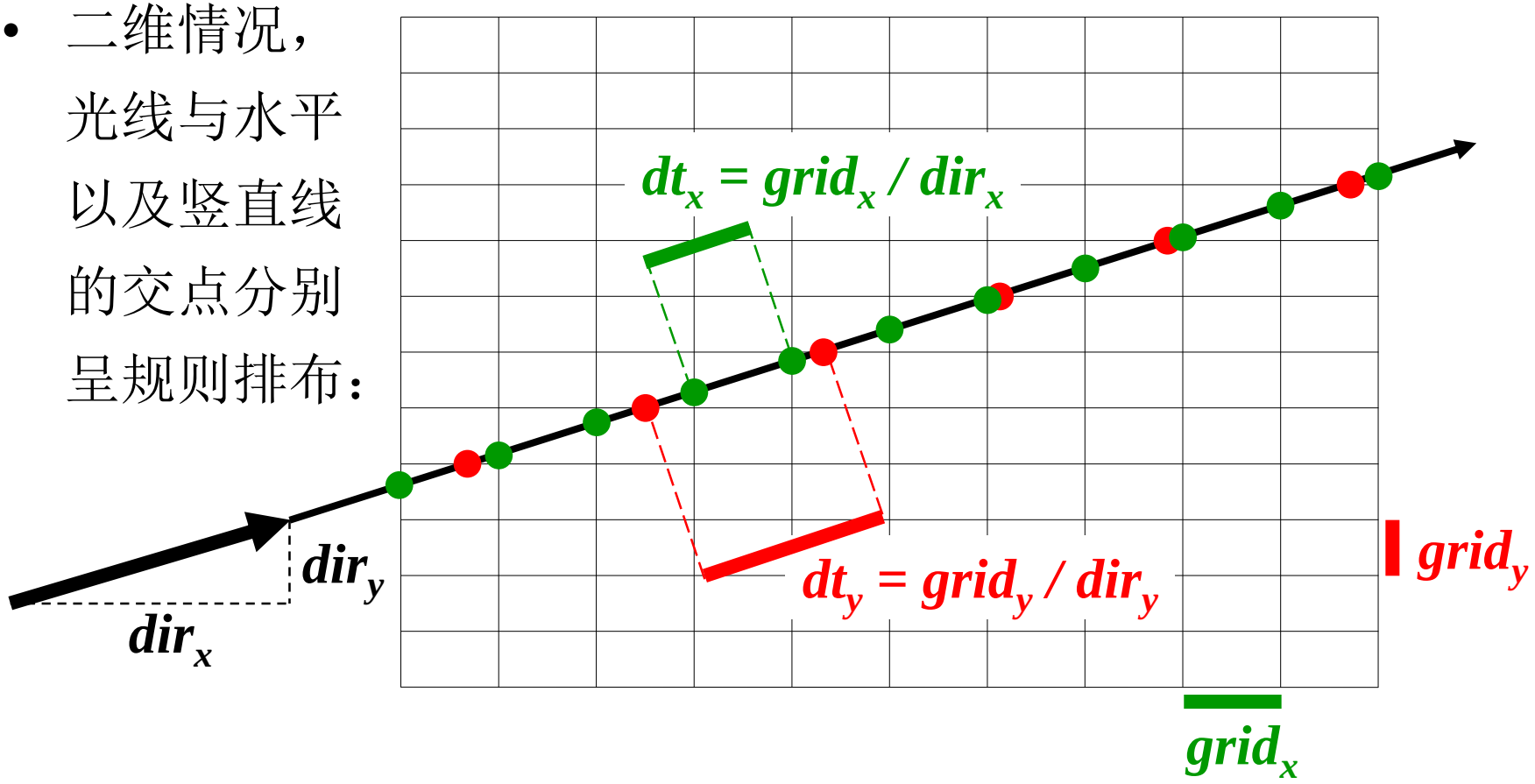


均匀格点 (Uniform Grids)

- 相交检测（如果我们知道光线从哪些格子经过）：
  - 从光线出发的格点开始，对光线经过的格点逐一进行搜索，找到第一个交点
  - 对经过的每个格点，检测光线与其关联的列表中的面片是否相交
  - 如果存在相交，则将最近的交点作为所求的交点
  - 如果一直没有交点，那么光线与场景中的物体不相交

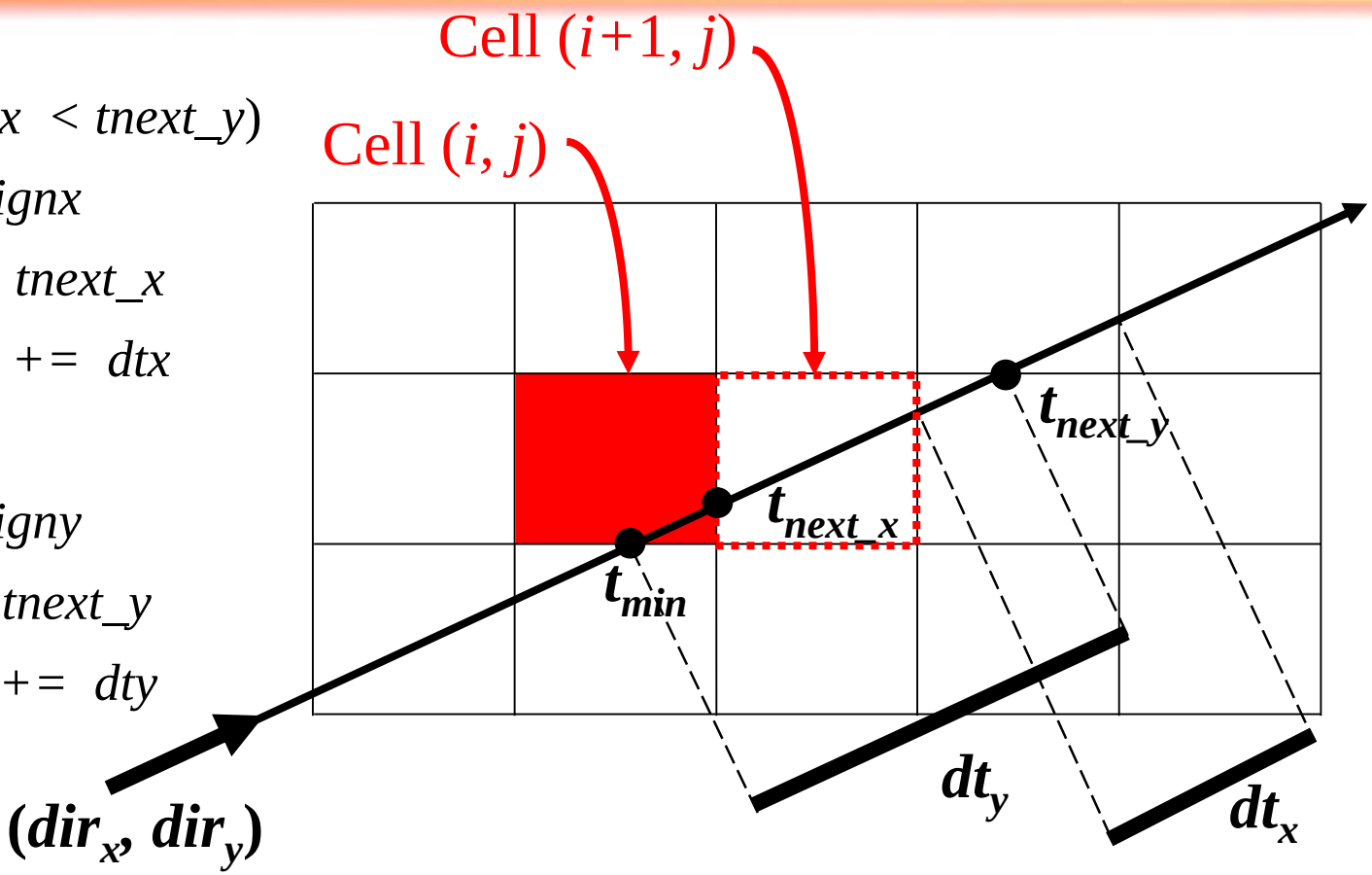
均匀格点 (Uniform Grids)

- 格点如何遍历?
- 二维情况，光线与水平以及竖直线的交点分别呈规则排布：



如何计算下一个经过的格点?

```
if  ( tnext_x  < tnext_y)
    i  +=  signx
    tmin  =  tnext_x
    tnext_x  +=  dtx
else
    j  +=  signy
    tmin  =  tnext_y
    tnext_y  +=  dty
```



if $(dir_x > 0)$ $sign_x = 1$ *else* $sign_x = -1$

if $(dir_y > 0)$ $sign_y = 1$ *else* $sign_y = -1$

如何计算下一个经过的格点?

- 三维情况类似:
  - 假设当前格点是 (i, j, k)

if $(dir_x > 0)$ $sign_x = 1$ ***else*** $sign_x = -1$

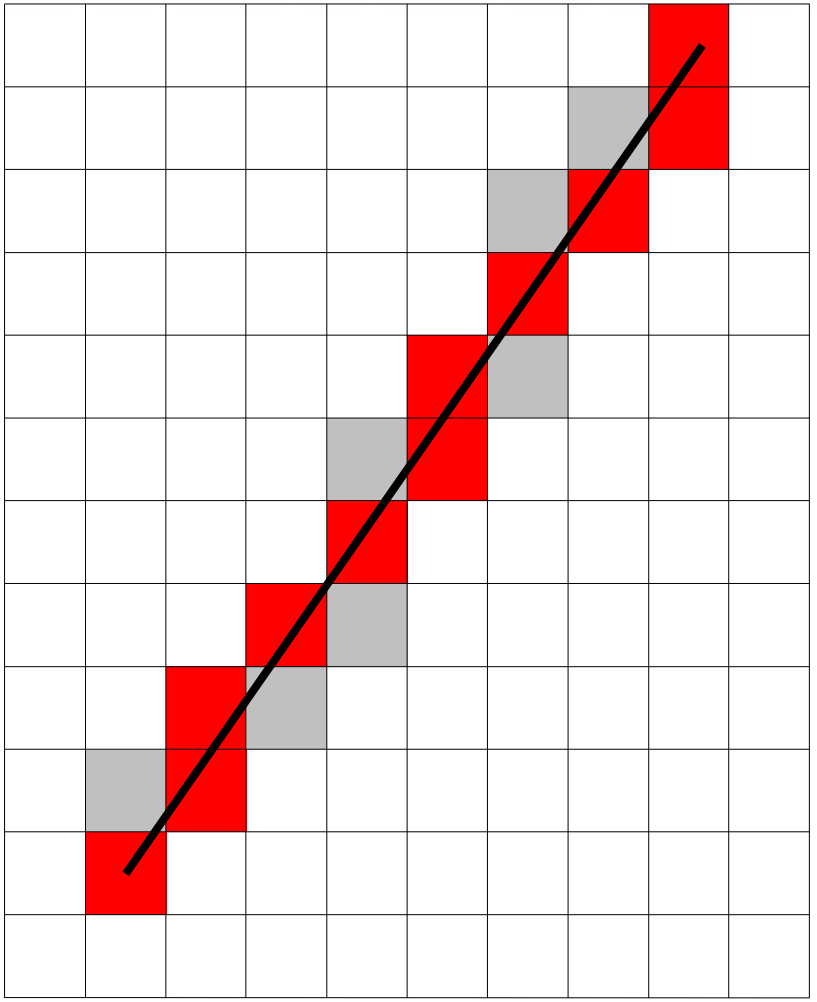
if $(dir_y > 0)$ $sign_y = 1$ ***else*** $sign_y = -1$

if $(dir_z > 0)$ $sign_z = 1$ ***else*** $sign_z = -1$

```
if  ( tnext_x < tnext_y and tnext_x < tnext_z )
    i += signx; tmin = tnext_x; tnext_x += dtx;
else if (tnext_y < tnext_x and tnext_y < tnext_z )
    j += signy; tmin = tnext_y; tnext_y += dty;
else // tnext_z is minimum
    k += signz; tmin = tnext_z; tnext_z += dtz;
```

如何计算下一个经过的格点?

- 3D DDA – Three Dimensional Digital Difference Analyzer
- 与直线的光栅化 (Line Rasterization) 类似

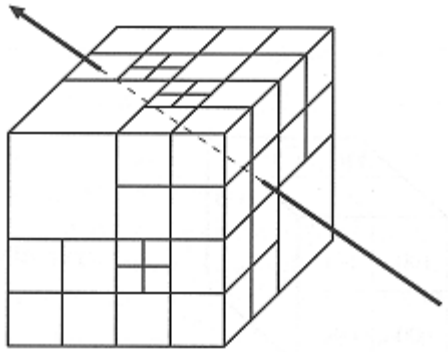


均匀格点 (Uniform Grids)

- 优点：
  - 容易构建、容易遍历
- 缺点：
  - 当场景不均匀时，均匀格点效果不好，因为大部分的多边形实际上位于较小的一部分空间内
  - 使用多少格点? (均匀格点尺度的问题)
    - 太少 → 每个格点存储的面片太多 → 运算变慢
    - 太多 → 需要遍历许多空的格点 → 慢而且需要大量内存
  - 非均匀的空间划分是更好的选择！

八叉树 (Octree)

- 四叉树 (Quad tree)
  - 二叉树 (Binary tree) 的二维推广
  - 每个节点是一个正方形，递归地将每个正方形分成四个等大小正方形
  - 当叶节点中的内容 (面片数) 足够简单时，停止递归



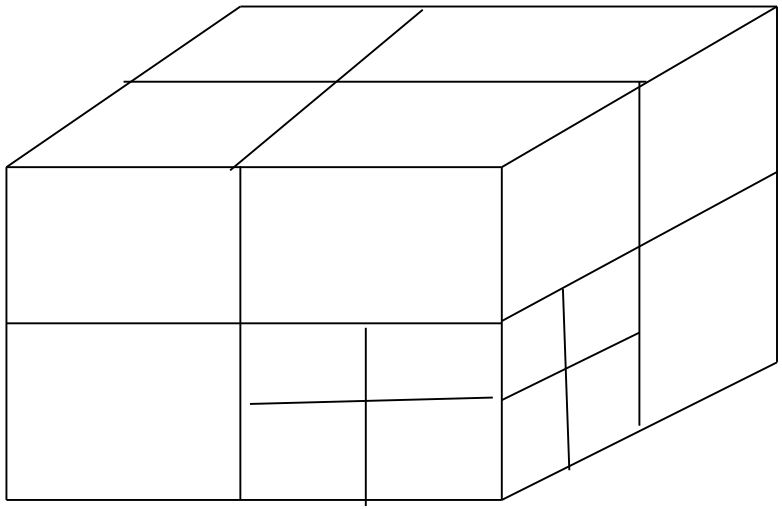
- 八叉树 (Octree)
  - 四叉树的三维推广
  - 每个节点是一个立方体，递归地分成八个小立方体
  - 八叉树的遍历比均匀格点要复杂，但是八叉树能够更好地适应空间分布不均匀的场景

八叉树 (Octree)

- 八叉树的构建：
  - 首先，将整个场景用一个最小的平行于坐标轴放置的立方体包围起来
  - 使用自顶向下的方式不断地将立方体划分为更小的立方体，直到满足一定的终止条件
  - 终止条件可以包括：
    - 达到了某个预先设定的最大递归深度
    - 节点中所包含的基本面片数目已经很少
    - …

八叉树 (Octree)

- 八叉树的构建：
  - 如果某个节点满足终止条件，那么创建该节点对应的面片列表，并终止对该节点的递归
  - 否则，沿坐标轴方向将节点对应的立方体切分成八个 (2×2×2) 大小相等的小立方体，然后对每个小立方体进行递归



八叉树 (Octree)

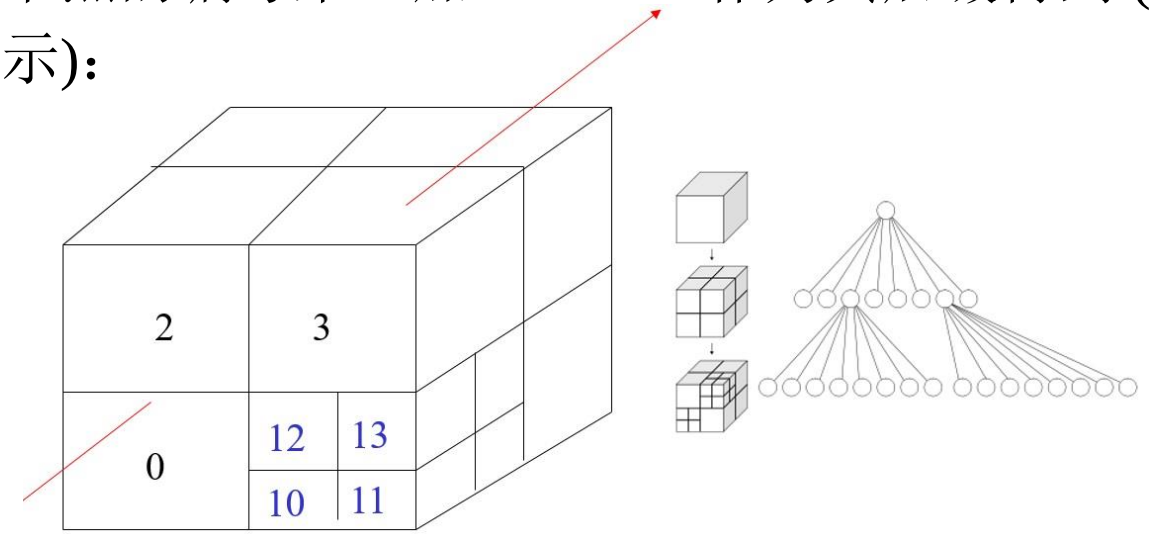
- 八叉树的构建：
  - 面片总是被存储在八叉树的叶节点上；同一个面片有可能被不同的叶节点同时存储 (例如：在某个非叶立方体中心附近的面片)
  - 面片被多个立方体同时存储会在一定程度上影响求交效率
    - 一种办法是将面片从几何上剖分成多个，但这样会使得面片的总数变多
    - 另一个解决方案是使用八叉树 (Octree) 的一个变种：Octree-R

八叉树 (Octree)

- Octree-R
  - Octree-R 与 Octree 的区别在于：Octree 只能从立方体的中心用三个平面将其划分成 8 个，而 Octree-R 的划分平面更加自由
  - 可以采用启发式算法为三个方向选择划分平面
  - Octree-R 通常能使 Octree 提速 4% - 47%，具体提速效果取决于场景中物体的分布情况

八叉树 (Octree)

- 八叉树的子节点寻址
  - 通常有两种办法对八叉树内部的子节点进行寻址：
    - 直接使用指针：八叉树的每个内部节点存有八个指针指向其子节点
    - 编码：对八叉树的每个节点，其八个子节点的编号使用该节点的编号乘 8 加上 0 至 7 作为其后缀得到 (如下图所示)：



八叉树 (Octree)

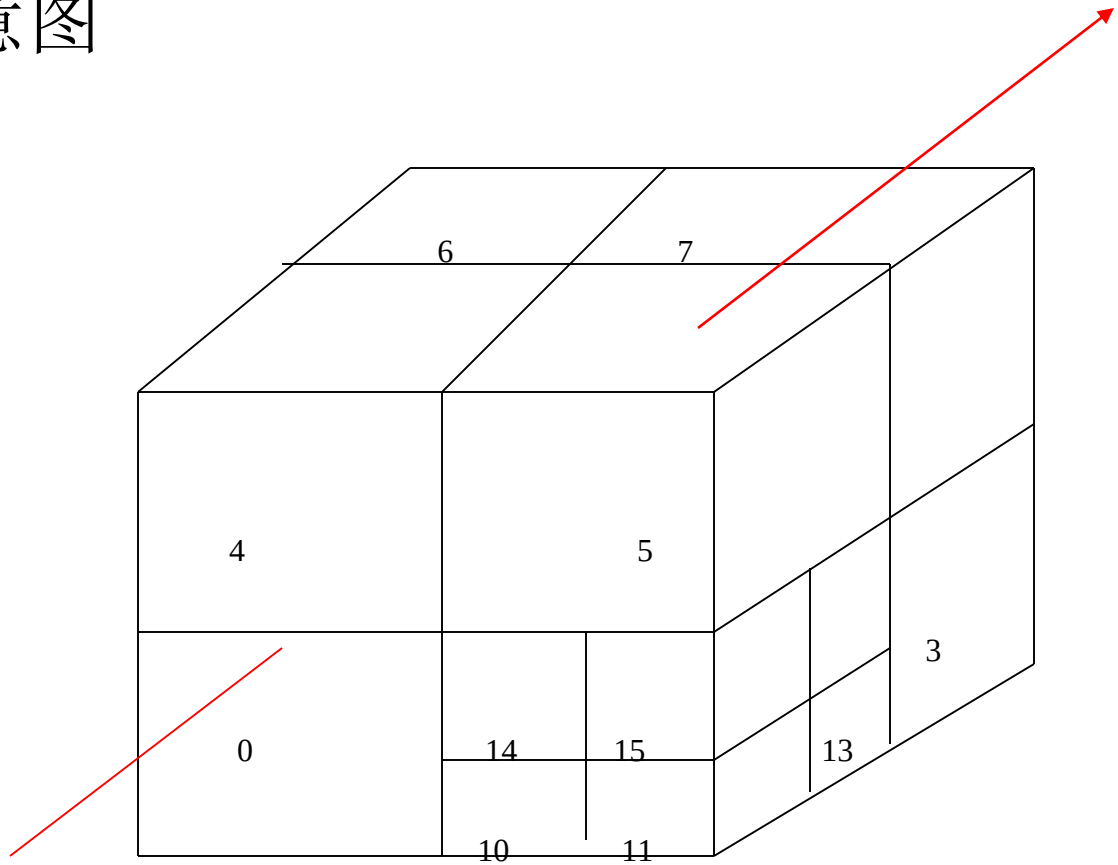
- 光线在八叉树中的遍历 (Traversal in Octree)
  - 光线在八叉树中的遍历算法要比在均匀网格以及 BSP 树 (BSP 树我们下面将要讲到) 中的遍历都更为复杂
  - 只要光线与八叉树的一个内部节点相交，就有可能与该节点的至多四个子节点相交，计算与这些子节点的相交顺序较为复杂

空间八叉树剖分技术

- 空间非均匀网格剖分算法
- 将含有整个场景的空间立方体按三个方向的中剖面分割成八个子立方体网格，组织成一棵八叉树
  - 只要某一子立方体网格中所含景物的面片数大于给定的阈值，就对该子立方体进行递归剖分
- 利用空间连贯性加速光线跟踪

空间八叉树剖分技术

- 示意图



空间八叉树剖分技术

- 八叉树的最大深度表示空间划分的最大层次，称为空间分辨率。假设八叉树的深度为N，则八叉树中节点的编码为

$$q_1 q_2 \cdots q_i \underbrace{FF \cdots FF}_{N-i}$$

其中$i \in [0, N]$, $q_1, q_2, \cdots, q_i \in \{0,1,2, \cdots, 7\}$

- 根据八叉树结点的编码方式，我们可以快速定位空间任一点所在的空间网格单元

八叉树剖分性质

- $P(x,y,z)$为一空间点，坐标为整数，其二进制表示为:
  - $x = i_1 i_2 \cdots i_N$
  - $y = j_1 j_2 \cdots j_N$ $\quad i_l, j_l, k_l \in \{0,1\}, \quad l = 1,2,\cdots,N$
  - $z = k_1 k_2 \cdots k_N$

- 则P会具有如下性质：

八叉树剖分性质

- 性质1：P所在单位立方体网格编码为$q_1 q_2 \cdots q_N$

$$q_l = i_l + 2j_l + 4k_l, \qquad l = 1,2,\cdots,N$$

- 性质2：若已知P位于一编码为$q_1 q_2 \cdots q_i \underbrace{FF\cdots FF}_{N-i}$的空间网格内（包括位于网格边界面上），则该空间网格的前左下角坐标为
  - $x' = i_1 i_2 \cdots i_i \underbrace{00\cdots00}_{N-i}$
  - $y' = j_1 j_2 \cdots j_i \underbrace{00\cdots00}_{N-i}$
  - $z' = k_1 k_2 \cdots k_i \underbrace{00\cdots00}_{N-i}$

光线跟踪过程

- 先利用性质1求光线起点P_0所在空间网格的八叉树编码Q
- 位于立方体边界上的起点要根据光线前进方向R判别光线是否已射出场景；若光线已射出场景，则算法结束
- 对Q在八叉树中计算下面两个值：
  - 代表查找是否成功的布尔量T
  - 匹配位数i

光线跟踪过程

- 设Q为$q_1 q_2 \cdots q_N$，那么在八叉树中含叶结点 $q_1 q_2 \cdots q_i \underbrace{FF \cdots FF}_{N-i}$ 时，T取真值
- 匹配位数i定义为八叉树叶子结点表中与Q获得最大程度匹配的结点与Q编码头部匹配的位数
- T决定当前立方体是否包含景物面片，而Q和i确定当前立方体的空间位置和大小

光线跟踪过程

- 若T取真值，则用光线和该立方体中所含的所有三角形面片求交，若有交，则返回最近交点；否则取T为假，继续向前搜索
- T为假的两种情况
  - 当前节点非叶节点使T取假，则更新当前网格为

$$q_1 q_2 \cdots q_i q_{i+1} \underbrace{FF \cdots FF}_{N-i-1}$$

  - 求交失败使T为假，则跨过当前立方体网格

光线跟踪过程

- 新网格的前左下角坐标由i确定
- 跨越一空间网格后，先求出当前空间网格上的出口点坐标，重置光线起点
  - 光线和六个面求交
  - 预先计算光线在各坐标平面上投影线的截距和斜率，快速求解
- 以新出发点重复跟踪过程，直至光线射出场景或与物体相交为止

- 光线跟踪的例子

空间二分树 (BSP Tree)

- 空间二分树 (BSP: Binary Space Partition Tree)
  - BSP 树是一种空间划分结构，可以用来解决大量几何问题，其最初是被用于解决图形学中隐藏面的问题 (the Hidden Surface Problem)
  - BSP 树是二分查找树的高维推广
  - BSP 树有两种主要类型：*axis-aligned* 类型和 *polygon-aligned* 类型

空间二分树 (BSP Tree)

- Polygon-aligned BSP 树
  - 每次选取一个多边形面片所在的平面，作为空间的划分平面
  - 通常来说，当场景仅由多边形组成时，才会使用 Polygon-aligned BSP 树，因此有一定的局限性
  - 我们在课程中不对 Polygon-aligned BSP 树进行更多讨论

空间二分树 (BSP Tree)

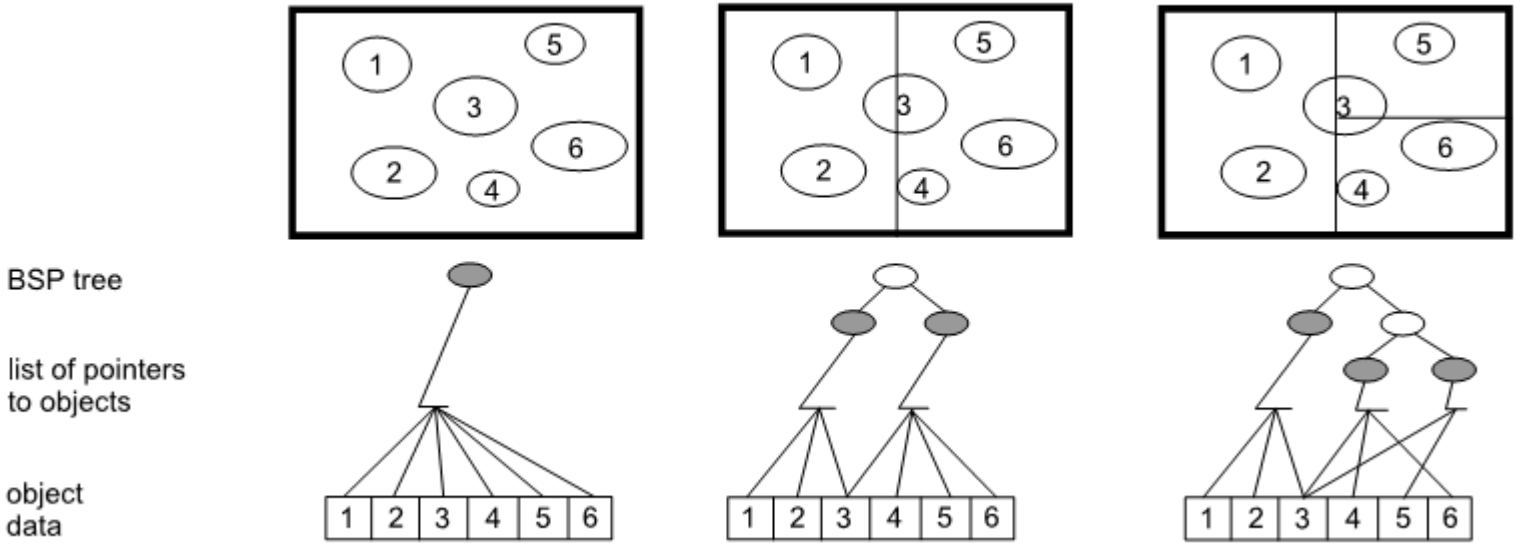
- Axis-aligned BSP 树
  - 划分平面总是和某个坐标轴呈垂直方向，并且划分平面总是把节点按照其空间大小划分为两个相等的子节点
  - 在某些文献对 BSP 的描述中，Axis-aligned BSP 的划分平面也可以自由选择

空间二分树 (BSP Tree)

- Axis-aligned BSP 树
  - 划分平面的正交性极大地简化了光线与划分平面的求交运算：计算光线与一个 Axis-aligned 划分平面的交点所需要的计算量大约是计算光线与一个任意位置划分平面的交点的三分之一

空间二分树 (BSP Tree)

- Axis-aligned BSP 树的一个例子：

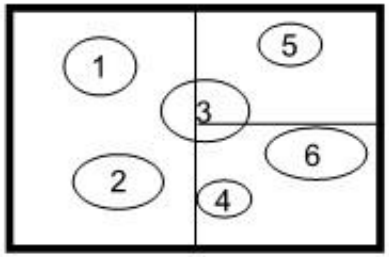


空间二分树 (BSP Tree)

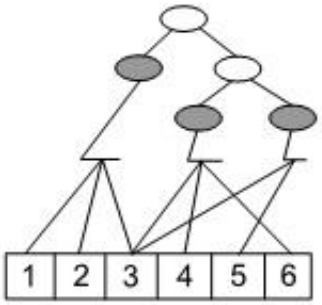
- Axis-aligned BSP 树的构建
  - 与八叉树类似，BSP 树使用由顶至下的顺序递归构建：
    - 每次使用一个垂直于坐标轴的划分平面 (splitting plane) 将一个当前的叶子节点划分为两个相等大小的子节点；然后这个叶子节点变成内部节点，两个子节点变成新的两个叶子节点
    - 递归进行以上过程，直到满足终止条件
    - 与八叉树类似，通常使用的终止条件有：最大叶子节点的深度超过阈值，或者叶子节点所对应的面片数足够少

空间二分树 (BSP Tree)

- Axis-aligned BSP 树的构建
  - 通常划分平面的位置可以放于所选轴的中心位置；轴 x y z 可以按照节点的深度进行规则轮换 (例如，首先是 x，然后是 y，再是 z，再是 x…)
  - 这样的方式可以使得层次结构的划分更为规则 (regular)

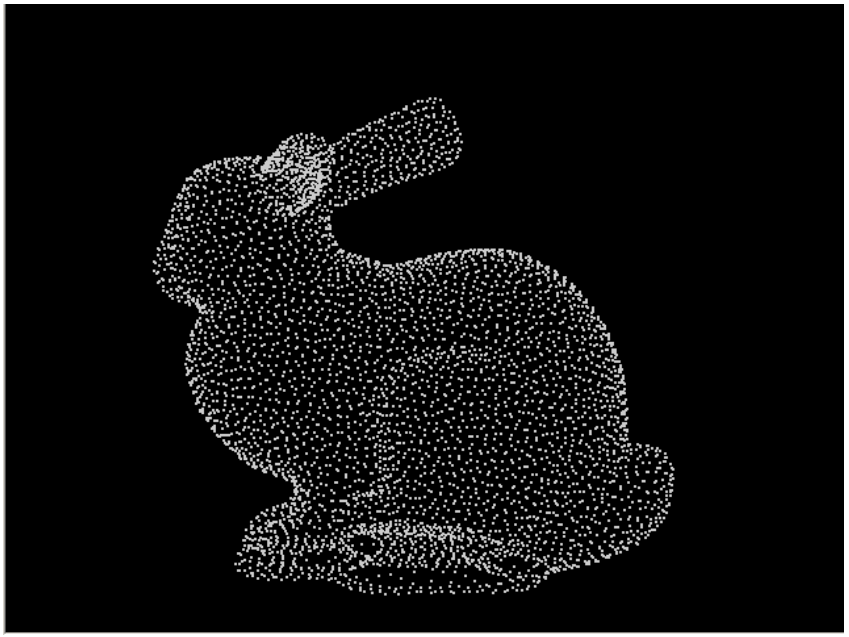


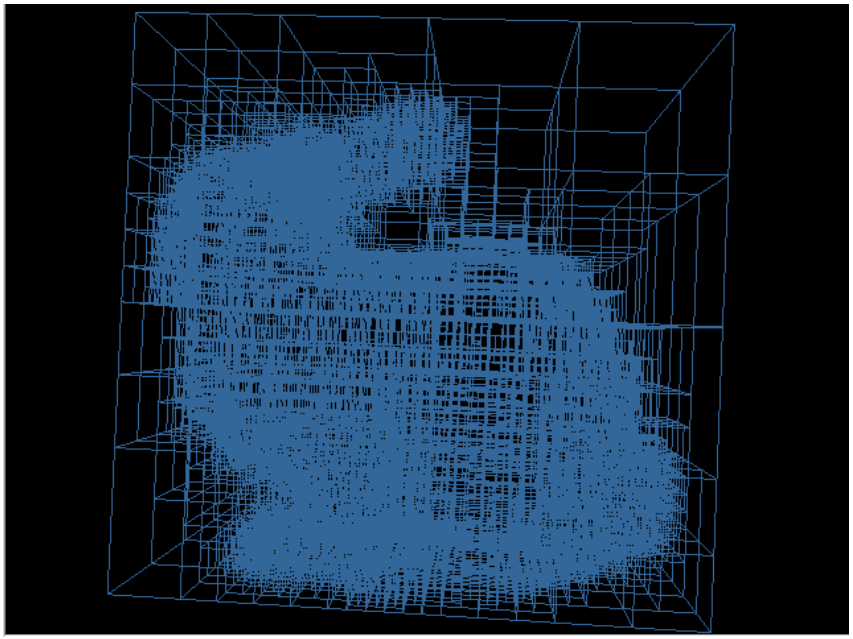
- 如何遍历的原理
  - 给定一个点（比如光源），利用节点保存的方程，判断在哪一侧, …



空间二分树 (BSP Tree)

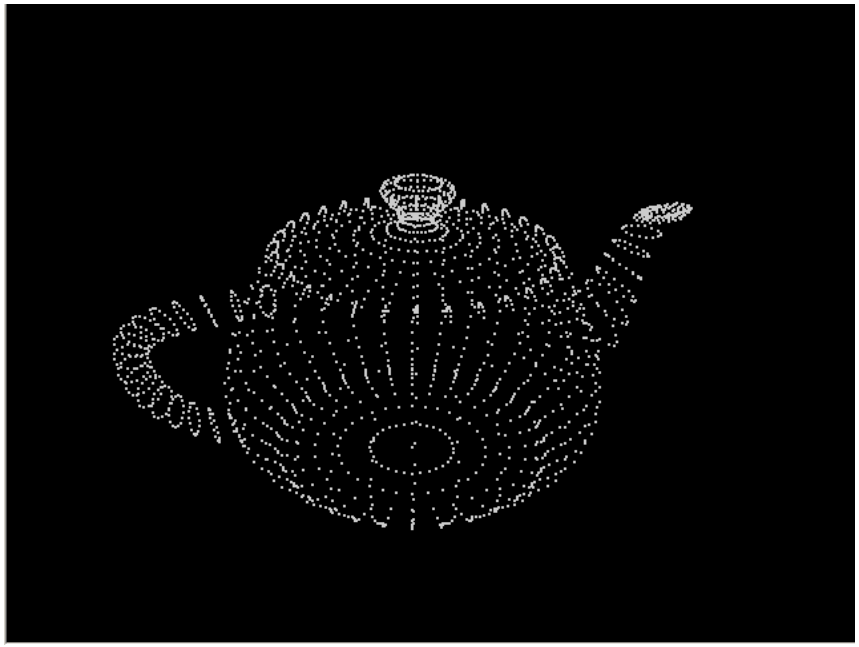
- 一些例子：

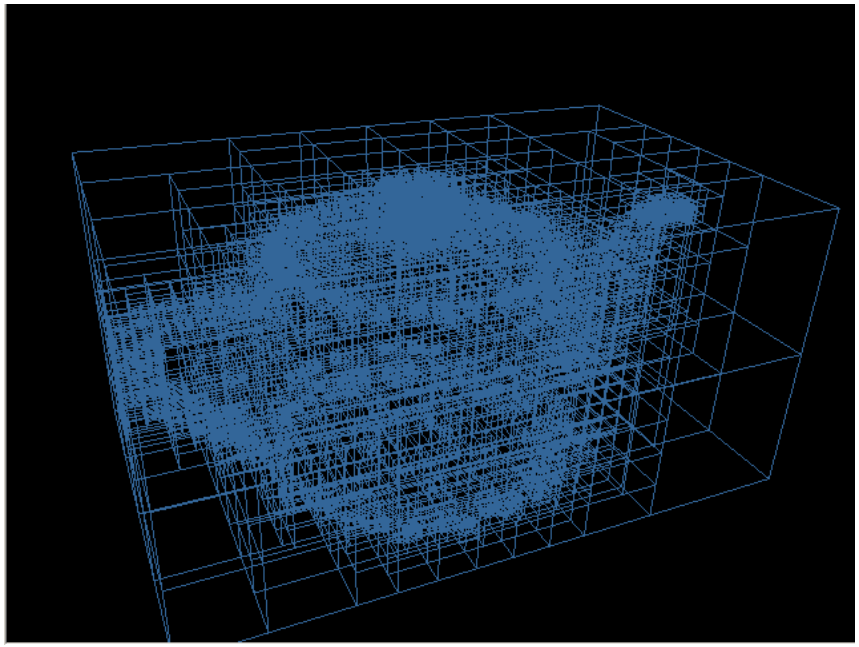




空间二分树 (BSP Tree)

- 一些例子：





空间二分树 (BSP Tree)

- Kd 树 与 BSP 树 的区别：
  - Kd 树与 BSP 树的区别主要体现在划分平面的位置选取上
  - 从概念上来说 Kd 树和 BSP 树是基本一致的，唯一的不同是：BSP 树要求划分平面将节点划分为两个相等大小的子节点；而 Kd 树的划分平面位置可以任意放置
  - 所以，任何一颗 BSP 树 都是一颗 Kd 树，但反过来则不成立

空间二分树 (BSP Tree)

- 光线在 BSP 树中的遍历 (Traversal in BSP Tree)

```
RayTreeIntersect (Ray, Node, min, max)
    if (Node is NIL) then return ["no intersect"]
    if (Node is a leaf) then begin
        intersect Ray with each primitive in the object link list
            discarding those farther away than "max"
        return ["object with closest intersection point"]
    end
    dist ← signed distance along Ray to the cutting plane of the Node
    near ← child of Node for half-space containing the origin of Ray
    far ← the "other" child of Node—i.e. not equal to near
    if ((dist > max) or (dist < 0)) then      /*Whole interval is on near side*/
        return [RayTreeIntersect (Ray, near, min, max)]
    else if (dist < min) then                 /*Whole interval is on far side*/
        return [RayTreeIntersect (Ray, far, min, max)]
    else begin                                /*the interval intersects the plane*/
        hit_data ← RayTreeIntersect (Ray, near, min, dist)    /*test near side*/
        if hit_data indicates that there was a hit then return [hit_data]
        return [RayTreeIntersect (Ray, near, dist, max)]      /*test far side*/
    end
```

空间二分树 (BSP Tree)

- 光线在 BSP 树中的遍历 (Traversal in BSP Tree)
  - 正如前面伪代码中展示的，光线在 BSP 树中的遍历需要递归，当然，可以通过手动维护一个堆栈来减少递归的函数调用开销
  - 当第一次调用函数 *RayTreeIntersect()* 时，变量 *min* 和 *max* 被初始化为光线与 BSP 根节点所对应的立方体的两个交点到光线出发点 (origin)较小和较大两个的距离
  - 如果光线是从 BSP 树根节点所对应立方体的内部发出的，那么变量 *min* 的初始值是负数

空间二分树 (BSP Tree)

- 光线在 BSP 树中的遍历 (Traversal in BSP Tree)
  - 通常，光线在 BSP 树中的遍历要比在八叉树中的遍历快大约 10%
  - 由于 BSP 树总是从中点处进行划分，因此一个物体 (面片) 有可能会跨越 BSP 的多个节点，从而使得相交检测的效率变慢
  - 类似于使用 Octree-R 代替 Octree 的思想，Kd 树可以较有效地解决这一问题

其他技术

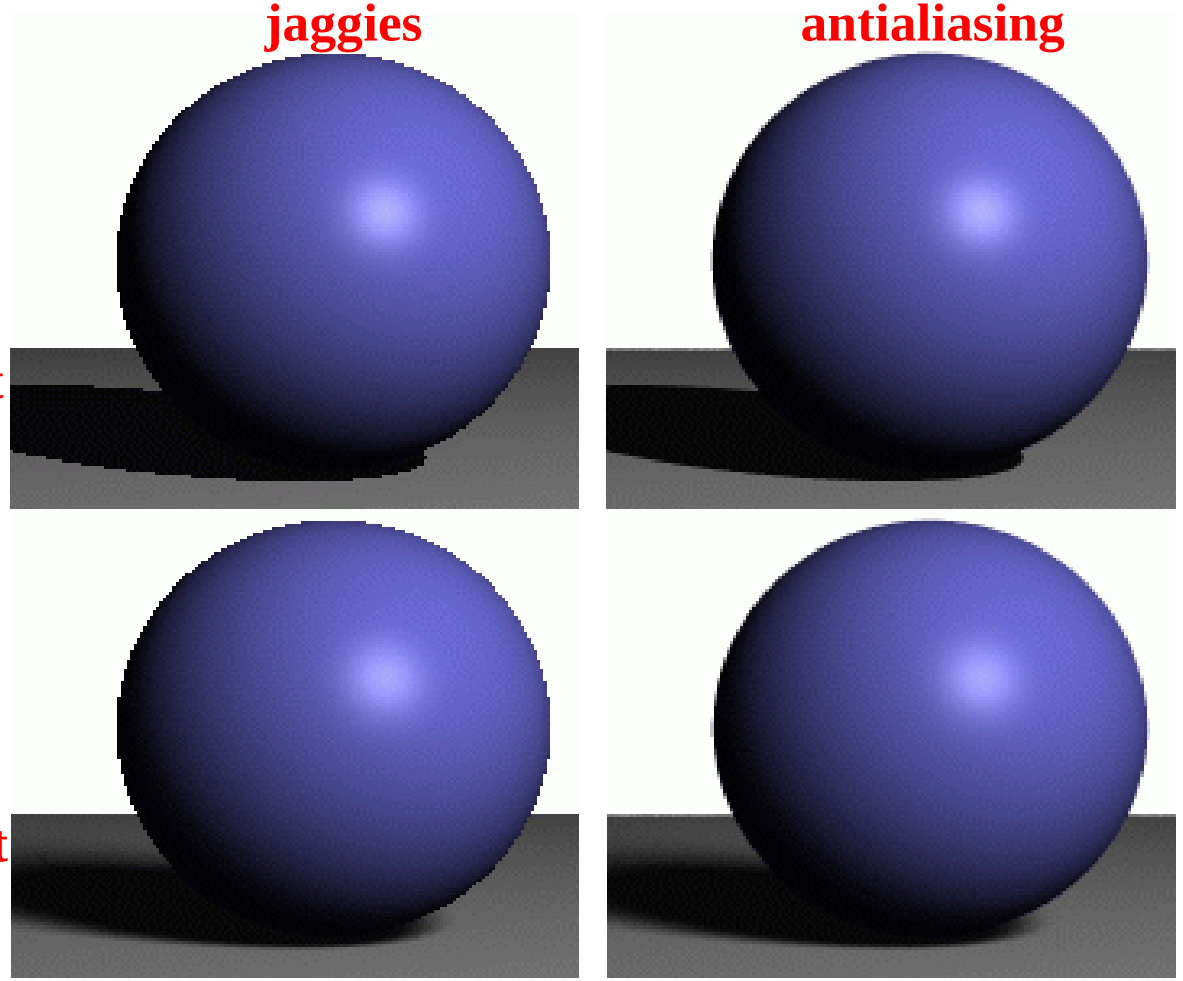
- 分布式光线跟踪 (Distributed Ray Tracing)：
  - 也叫随机光线跟踪 (stochastic ray tracing)，它是光线跟踪的一种改良，可以实现绘制的软效果 (“soft” phenomena)
  - 传统的光线跟踪在需要计算一个物体的颜色时，只对每个光源投射出一道光线，这会导致过于尖锐的阴影效果 (Sharp Shadow)
  - 同样地，传统的光线跟踪同样只在每个交点处产生一道反射光线和一道透射光线，这也会导致过于尖锐的反射与折射效果

分布式光线跟踪

- 分布式光线跟踪(Distributed Ray Tracing)
  - 可以产生一系列视觉效果：
    - 光照明：全局光照，软阴影
    - 像素：反锯齿 效果(Anti-aliasing)
    - 焦距：景深效果 (Depth-of-field)
    - BRDF：模糊镜面反射效果 (Glossy-reflection)
    - 时间轴：运动模糊 (Motion-blur)

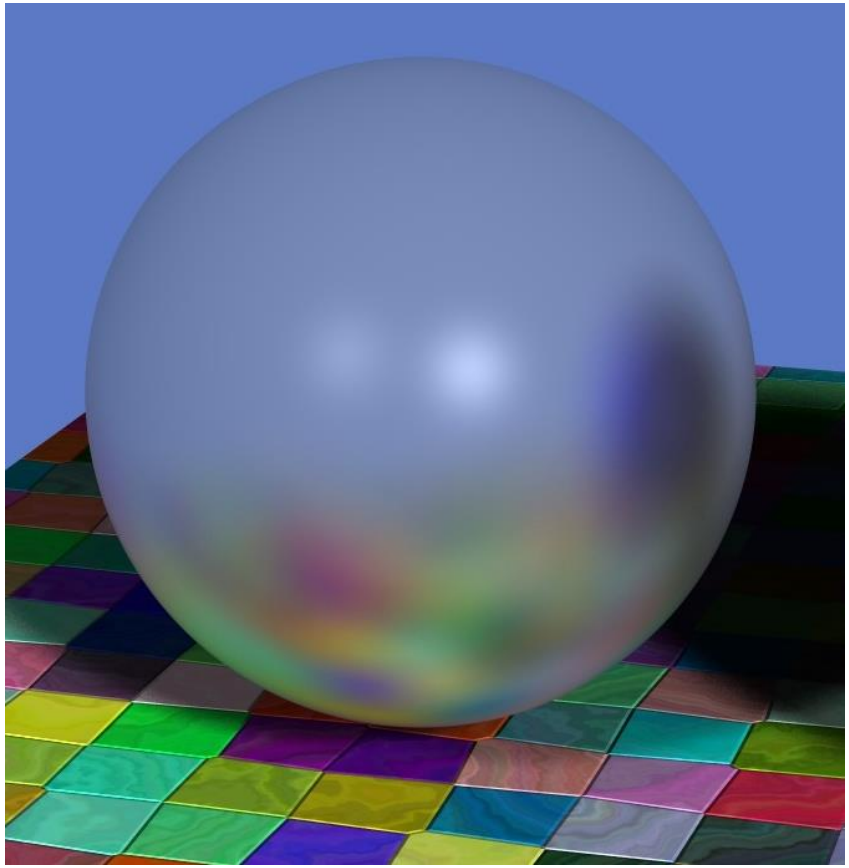
分布式光线跟踪

- 软阴影和反锯齿效果：



分布式光线跟踪

- 模糊镜面反射效果：

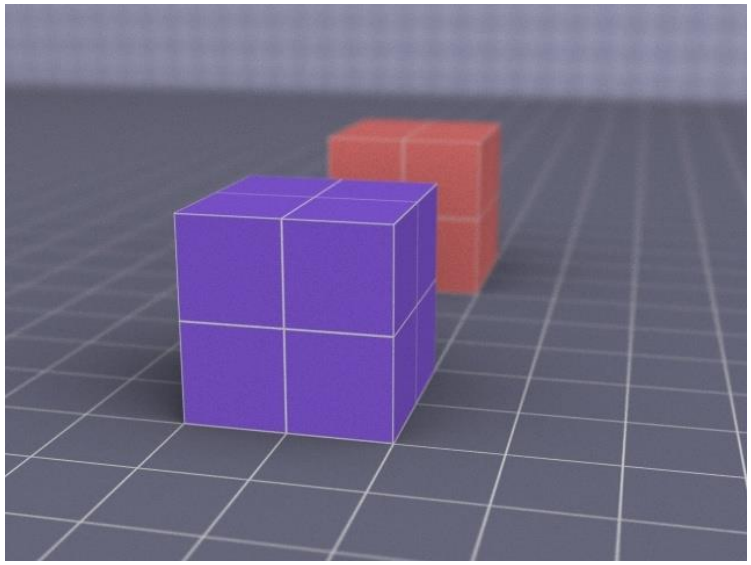


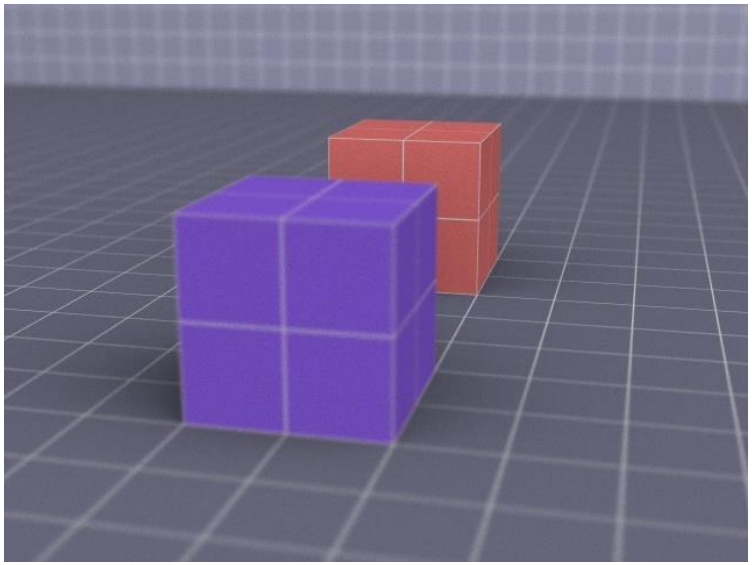
分布式光线跟踪

- 运动模糊效果：

分布式光线跟踪

- 景深 (Depth-of- field) 效果：





光束跟踪 (Beam Tracing)

- 光束跟踪 (Beam Tracing)
  - 光束跟踪是光线跟踪的变种，它将光线跟踪中没有“厚度”的射线变成了锥状的光束
  - 在光束跟踪中，首先将初始的锥状光束投向可视空间，由近及远检测光束与场景中物体的交，在交点处产生新的反射和折射光束，产生方式类似于光线跟踪算法

光束跟踪 (Beam Tracing)

- Heckbert和Hanrahan通过光束跟踪来利用光线之间的空间连惯性，提高光线跟踪的效率(Siggraph’84)
- 只考虑多边形场景
- 光束跟踪也可用一棵光束树描述，光束树中的边表示被跟踪的光束，结点表示光束与场景中多边形的交

光束跟踪 (Beam Tracing)

- 光束跟踪的关键是如何定义初始光束与反射、折射光束
- 从视点出发投向屏幕的初始光束，形成视域四棱锥，但四棱锥表示的光束与景物求交不方便
- 因此，必须作线性变换(包括透视变换)，将景物从景物坐标系变换到屏幕坐标系
- 此时，视域棱锥变为柱体，视点变换到Z轴的无穷远处

光束跟踪 (Beam Tracing)

- 反射光束与入射光束之间遵循反射定律，这等价于在镜面的背后存在一个与入射光束光源对称的虚光源，从虚光源发出的光穿过镜面多边形后形成光束，并与场景中景物求交
- 考虑到如此定义的反射光束与景物表面求交时处理不便，我们**取反射光束的虚像，并将场景中的景物变换到其虚像位置，在虚世界中求反射光束与景物表面的交**
- 由于反射光束的虚像即入射光束的延伸部分，可以延续初始光束的定义，这样做便于统一处理

场景坐标系到其局部坐标系的变换

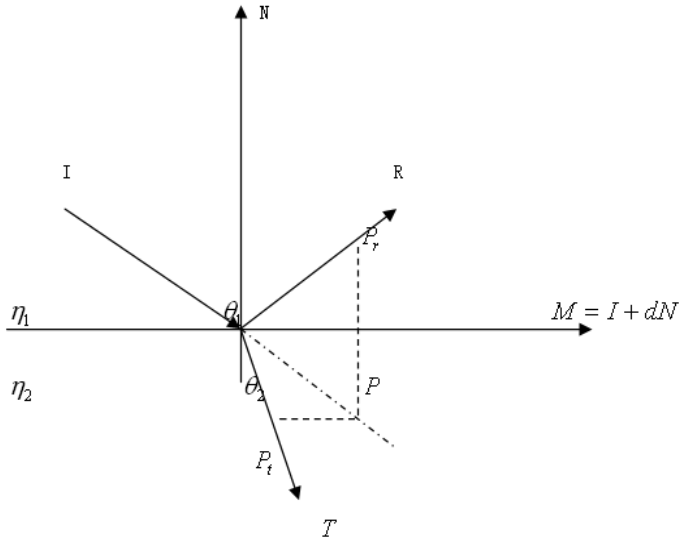
- 假设一条光线入射到一镜面上，入射方向为I，所产生的反射光线方向为R，折射光线方向为T，记镜面所在的平面方程为：

$$L \cdot P = Ax + By + Cz + D = 0$$

其中$L = (A, B, C, D), P = (x, y, z, 1)^{\mathrm{T}}$

且$A^2 + B^2 + C^2 = 1$，

故其单位法向量为$N = (A, B, C, 0)$



反射光线的变换

- 记P_r为反射光线上的任一点，P是其虚像，则$L \cdot P_r$为P_r距平面的垂直距离，容易计算得到：

$$P = P_r - 2(L \cdot P_r)N = M_r P_r$$

- 其中M_r为齐次4×4的反射变换

$$M_r = \begin{bmatrix} 1-2A^2 & -2AB & -2AC & -2AD \\ -2AB & 1-2B^2 & -2BC & -2BD \\ -2AC & -2BC & 1-2C^2 & -2CD \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

反射光束跟踪过程（1）

- 为了跟踪初始光束入射到镜面多边形上所生成的反射光束，现将场景中的景物作上述坐标变换，此时，式

$$P = P_r - 2(L \cdot P_r)N = M_r P_r$$

可理解为将景物变换到反射光束的局部坐标系中，反射光束局部坐标系z轴与反射光束前进方向一致

- 将所有景物按其变换后的z值排序

反射光束跟踪过程（2）

- 取镜面多边形在反射光束局部坐标系xy坐标平面上的投影为反射光束多边形，并将它与z值排序表上第一个景物多边形在xy坐标平面上作二维布尔运算：若存在交集，则将求得的交，映射到多边形上，作为反射光束将照射到的一个多边形区域记入反射光束的数据结构中，并在反射光束多边形中减去交区域，继续与场景多边形表中第二个多边形在xy坐标平面上的投影求交
- 重复上述过程，直到反射光束多边形被完全覆盖或所有景物多边形均处理完毕为止

折射光线的变换

- 折射光束的跟踪，此时的折射变换为非线性变换，一条直线经折射后会变成一条曲线
- 为将折射变换表示为类似于反射变换的线性变换，Heckbert等假设光线接近于垂直地入射到表面上，并假设景物表面的折射率对不同角度的入射光线均取常数

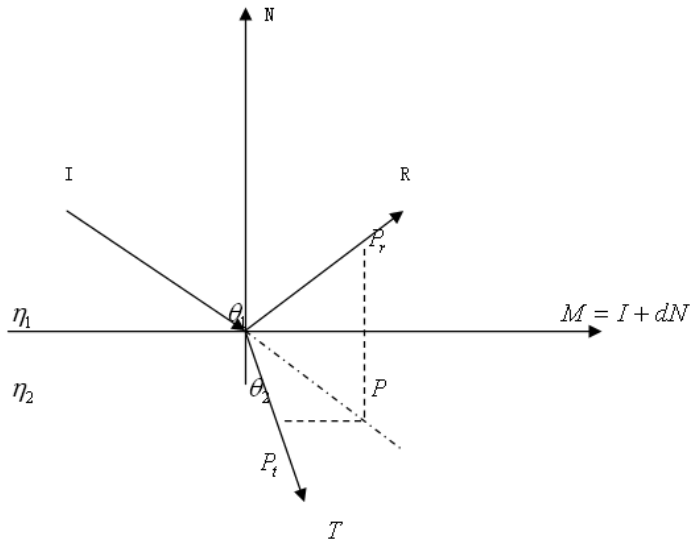
- 这样，与P对应的点P_t可表达为

$$P = P_t + \alpha(L \cdot P_t)M$$

- 其中$M = N \times (I \times N) = I - (N \cdot I)N,$

$|M| = \sin\theta_1, \alpha = \dfrac{(tg\theta_1 - tg\theta_2)}{\sin\theta_1}, \theta_1, \theta_2$ 分别是入射角和折射角，于是折射变换可表达为

$$P = M_t P_t = [E + \alpha(M \cdot L)]P_t$$



表面光亮度的计算

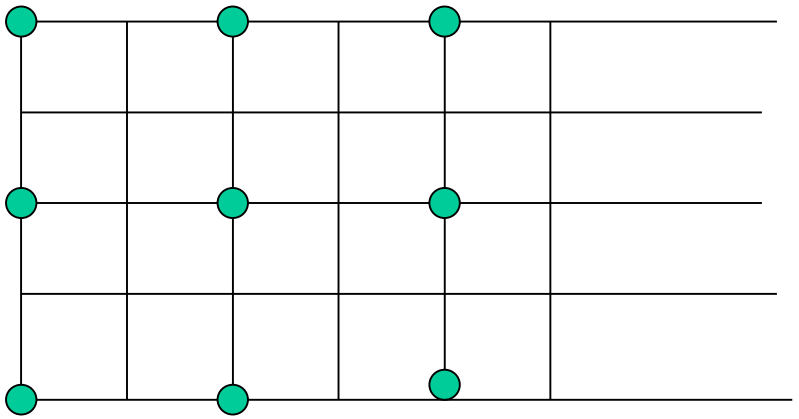
- 对场景的光束跟踪生成一棵光束树。由于光束树实际上是通过景物间的反射和折射对屏幕上可见表面的光亮度所作的贡献的所有表面裁片(fragments)的组合，因而可见景物表面的光亮度可由下式计算：

$$I = C_d I_d + C_s I_s + \Sigma C_r I_r + \Sigma C_t I_t$$

其中，I_d, I_s分别为漫射、镜面反射光亮度，可采用Phong模型计算; I_r, I_t分别为整体镜面反射和整体折射光亮度，可通过遍历光束树来递归计算。C_d, C_s, C_r, C_t为相应的反射系数

其他技术

- 使用选择性的光线跟踪 (Selected Ray Tracing) 结合插值方法 (Interpolation)。
  - 选择需要进行光线跟踪的像素 (pixel for ray racing)，例如，选取下图的角点进行光线跟踪：
  - 然后对其他像素的颜色进行插值



- 基于小波 (Wavelet) 的像素选取
  - 使用小波来进行“重要性采样” (importance sampling)
  - 重要性采样是指：如何选取重要的点和像素

- 光线跟踪的 Demo
  - RPU: 一个用于实时光线跟踪的可编程光线处理单元(Ray Processing Unit)

    rpu_video_2xfpga
  - RPU 是一个完全可编程的光线跟踪硬件，包含对材质、几何、以及光照的编程 (SIGGRAPH 2005 paper)
  - See Video

今日人物： Thomas W. Sederberg

- Thomas W. Sederberg： BYU教授
  - **发表论文140余篇，引用17219次，**
  - H-Index: 52
  - 1975，1977年BYU土木工程专业
  - 1983年Pudure机械工程系博士毕业
  - 1983年起BYU任教
  - 发表14篇SIGGRAPH论文，FFD被引4095次；

    两篇shape blending, 共被引968次；开创T样条和非均匀有理细分方法，被引1228次

2013年获得美国Solid Modeling Association的Bezier奖

谢谢！