Acceleration Structures

> 参考文章：
>
> - https://www.jianshu.com/p/6d8710d205f3

应用场景：当场景非常复杂，我们会将场景分成包围盒，如果光线/包围盒/其他形状与包围盒相交【包围盒往往是:球体/AABB/OBB】，才会具体与包围盒中的物体进行求交计算，用于加速空间查询。

加速结构的相关应用：

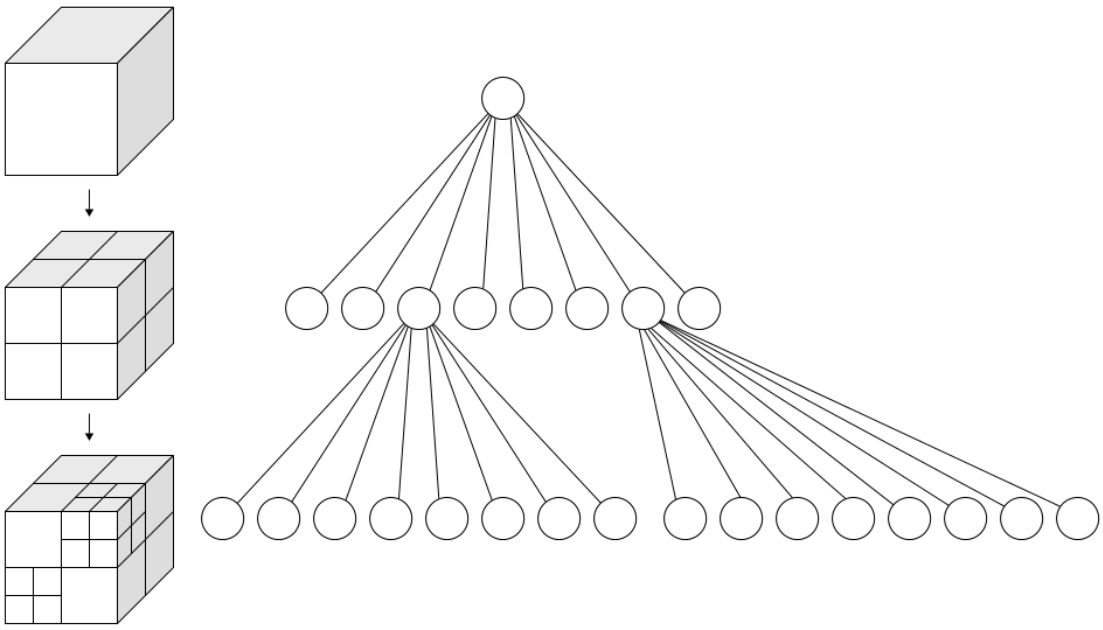
- **碰撞检测**的策略阶段，找到潜在可能碰撞的物体对
- 加速**视锥体裁剪**
- 加速**射线投射**(Ray Casting)、**光线追踪**(Ray Tracing)
- **邻近查询**，查询玩家角色某半径内的敌方NPC

常见的加速结构：

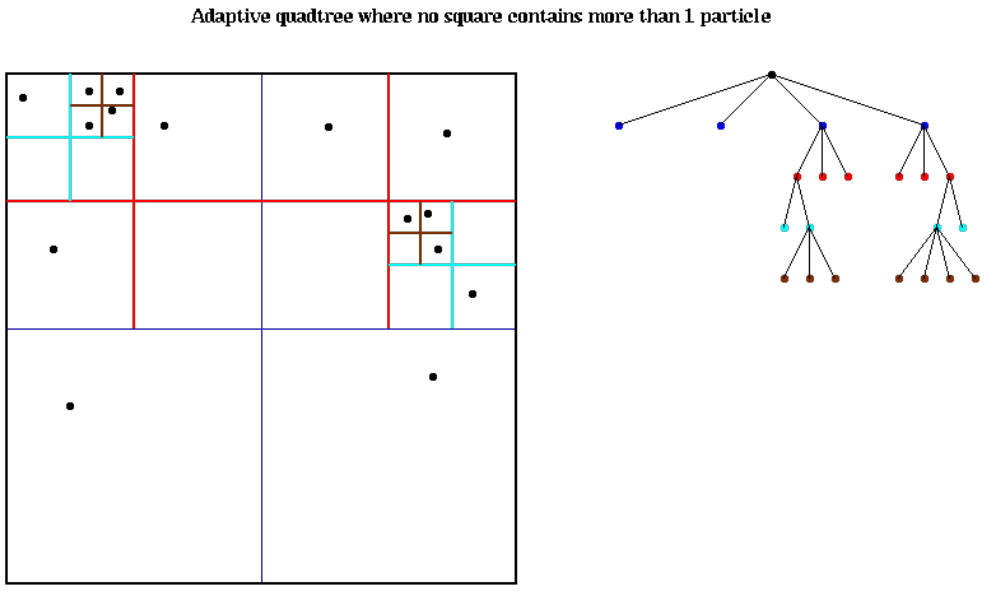
- **空间划分**
  - Oct-Tree(八叉树)：将三维空间切三刀，切成八份 => 对于n维空间，就是切n刀，划分成2^n份，维度越高越复杂(人们不喜欢这样)
  - KD-Tree(K-Dimensional)：KD-Tree的思路和Oct-Tree几乎完全相同，KD-Tree每次只沿某一轴切一刀，因此空间被划分为二叉树的结构(划分与维度无关) => 为了均匀起见，我们会按照x/y/z轴的顺序依次砍
  - BSP-Tree：与KD-Tree几乎一致，只是每次砍一刀，并不一定是横屏竖直的 => 这里的一刀，在二维空间中是线，三维空间是平面，高维空间是超平面
- **物体划分**
  - BVH(Bounding Volume Hierarchy)：二叉树的形式，每次将物体分成两堆，重新计算包围盒 => 问题：不同物体之间包围盒可能相交，所以也没有把物体完全划分开

（1）八叉树

> 参考链接：游戏场景管理的八叉树算法是怎样的？ - Milo Yip的回答 - 知乎
> https://www.zhihu.com/question/25111128/answer/30129131



每次划分，若正方体含有超过1个点，就将正方体分割，直到每个小正方体(叶节点)仅含有一个点。我们以二维空间的四叉树为例，作图说明 👇：



具体细节：

- 设置条件来进行划分：比如最多划分四层
- 分割时检测点在每个轴的某一侧，就能知道点属于哪个节点
- 八叉树的增、删、查问题

问题：

- 物体与划分边界相交，**物体可能属于多个子空间中** => 这会导致该物体要与光线多次相交判断
- 解决方法：(1)允许一个物体存在多个子空间中，每个子空间都有该物体的引用；(2)**让非叶子结点也能存放物体**<较常用> => 方法1很精确，但不便于空间管理；方法2更为普遍，但对于一些特殊case，也会造成麻烦（比如某个很小的物体刚好在场景正中心，所以只能放在根节点中）
- 改良思路：松散八叉树 - 适当略微扩大包围盒，以容纳边缘物体

缺陷：

- 八叉树可能不便于内存存储，应该使用更为扁平的结构并利用SIMD以提高性能，或者将八叉树改良为只有两、三层
    - 不利于内存存储：每个节点存储8个指向其他节点的指针，这种结构导致大量的随机访问，不利于内存的缓存机制(缓存最近使用的数据、连续内存访问等)
    - 扁平的数据结构：扁平的数据结构，比如数组，往往意味着连续的内存，也利于SIMD操作

> 拓展：OpenVDB技术 - https://www.openvdb.org/

（2）KD-Tree

> 参考链接：数据结构专题(一) | kd-tree 原理深入理解【看这一篇就够了】 - 无疆WGH的文章 - 知乎 https://zhuanlan.zhihu.com/p/529487972

每次切一刀，这一刀是平行于坐标轴。比如以x轴划分，只需要比较各点的x值与划分值，分为左右两个部分，因此得到二叉树的结构。kd-tree的构建是一个不断“寻找划分轴和划分值，并划分左子树和右子树”的递归过程。

划分选择：

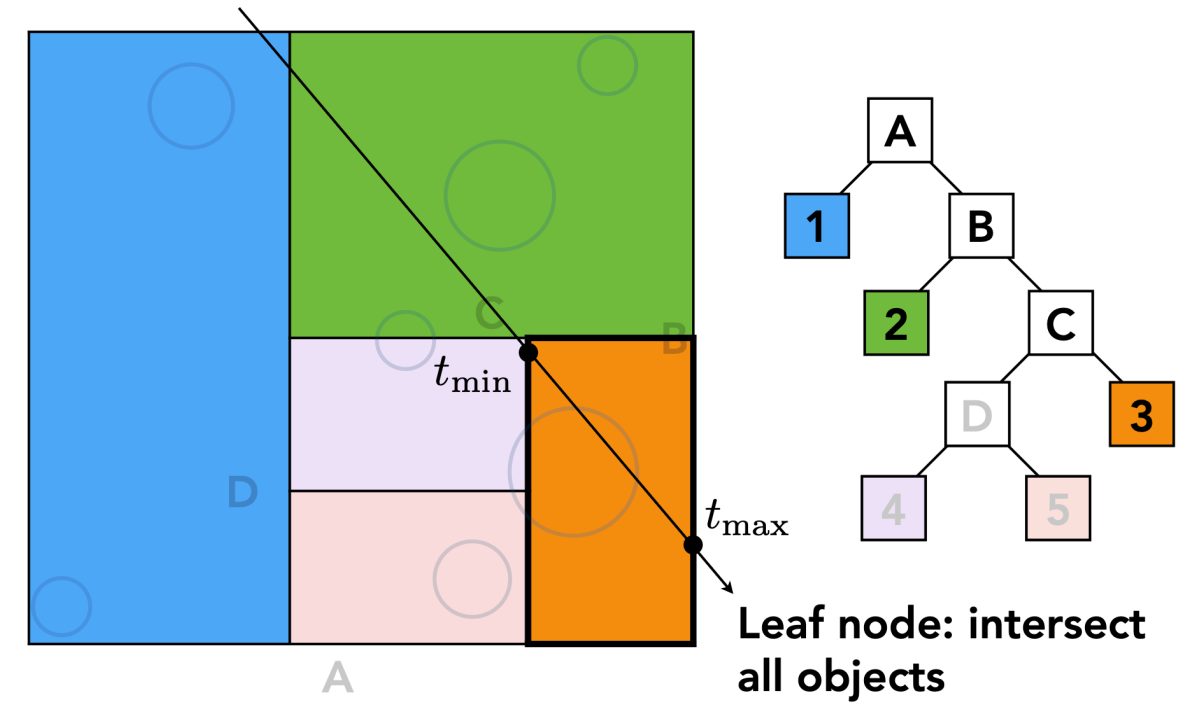
- 划分值的选择：通常是中值点法，选择某轴的中位数
- 划分轴的选择：
    - 策略1：各维坐标轴轮流作为划分轴：循环选择x/y/z
    - 策略2：始终选择分布最分散的轴作为划分轴

kd-tree的搜索问题：

- 应用：kNN(k近邻搜索)、ranged-kNN(有范围限制的k近邻搜索)
- 要找到最近n个节点，最重要的是子树的剪枝问题，因为最近的n个点 => 给定范围会好计算很多，因为能直接剪枝；但未给范围就需要动态维护**目标解空间**
- 某个子树可以被剪枝掉的必要条件是：该子树所代表的空间与目标解空间无交叉！
- 并且通过排序队列维护最近的n个节点

其他问题：

- 插入：从根节点出发，决定往左或者往右走，直到叶子节点
- 删除：删除节点的下方子树所有节点打散并重新构建子树



搜索过程：从根节点，依次判断是否与左右节点相交，如果相交，则向下继续递归

> 拓展：kd-tree家族：ANN、FLANN、libnano、3DTK、CGAL、STANN、Ball-Tree等等

（3）BVH

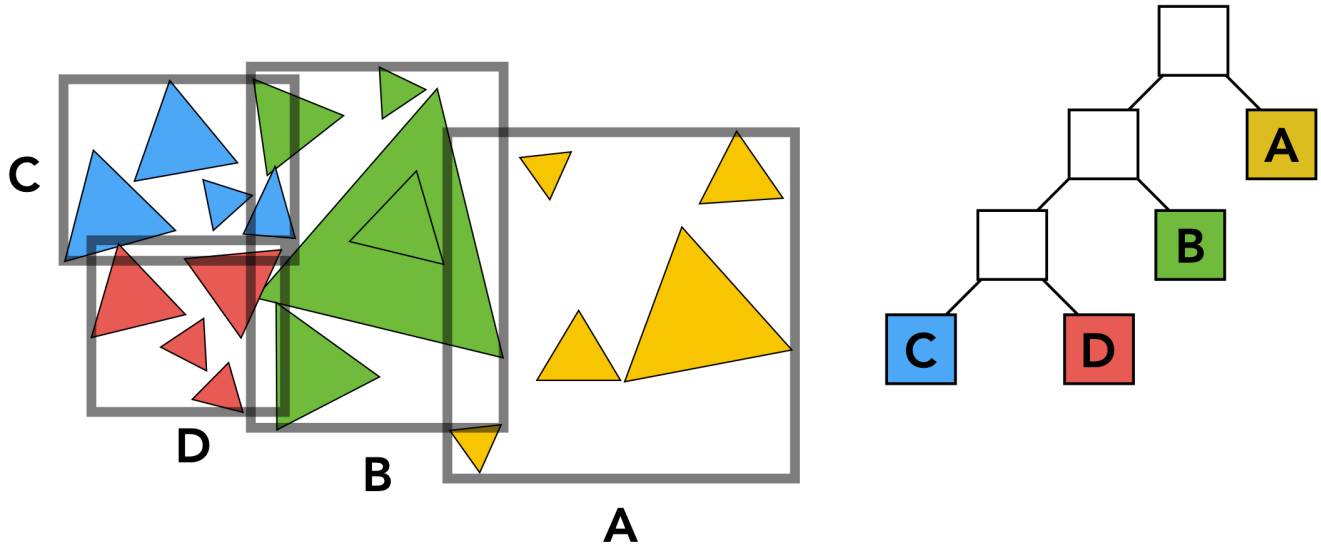
KD-Tree是将盒子分成两部分，考虑物体与盒子的相交情况；BVH是将物体划分为两部分，然后重新计算包围盒，当叶子节点里的三角形数量足够少时停止

BVH的特性：

- 一个物体只会出现在一个盒子里，避免同一个物体多次求交
- 但对空间的划分可能产生重叠，应当找一个重叠少的划分方式，提高效率

划分依据：

- 每次沿着最长的轴进行划分
- 划分时选取处在包围盒中心的物体(一般会排序,取中间序号的三角形)，划分成两部分，构造二叉平衡树



（4）总结

加速结构遇到的挑战：

- 当物体移动，加速结构涉及增、删节点操作，需要重新计算、维护树结构