Octree 算法

Wenping Guo

1. 算法原理

Octree 是基于空间切分算法(space partitioning)中的一种, 另外一种著名的算法是 kdtree[1]. Octree 算法根据数据点的坐标特征, 将数据点划分到大小不同的多个箱体中(Octant). 如果某个箱体中包含多个数据点,再以递归的方式进行分解处理, 直至箱体的尺寸达到某个阈值. 这些箱体形成像树一样的层级结构. 一旦 Octree 构造完成, 搜索某个数据点的邻近点, 就可以循着树根搜索相关的子箱体, 然后再计算与这些箱体中数据点的距离, 从而最终确定邻近与否. Octree 算法通过牺牲一定的内存(Octree 数据结构), 极大地减小了后续需要几何距离计算的次数, 降低了计算复杂度, 因而可以处理更大体系.

2. 问题定义

P 是三维数据点集合:

$$P = \{p_1, \cdots, p_N\}, p_i \in \mathbb{R}^3$$

求某点 q 在距离限制范围 r 内的最近邻数据点:

$$N(q, r) = \{p \in P \mid ||p - q|| < r\},$$

3. 数据结构

3.1. Octant

Octant 是三维几何空间中, 以中心点分界的八个象限(或箱体)之一 [2].

1 https://en.wikipedia.org/wiki/K-d_tree
2 http://mathworld.wolfram.com/Octant.html

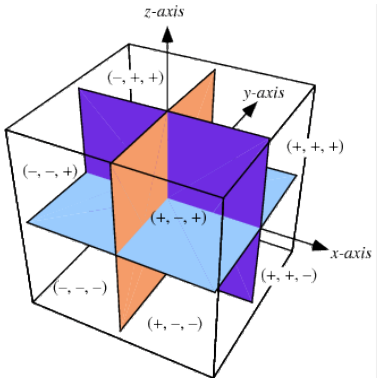

Figure 1: Octant

八个 Octant 可以唯一指认, 比如(+, +, +) 对应第一象限, (-, +, +) 对应第二象限等.[3]

3.2. Octree

Octree 是 Octant 所构成的树状结构. 对于三维数据点所在几何空间里:

- 将数据点以递归的方式分割成不重叠的区域. 一个或几个数据点属于唯一的一个 Octant.
- Octant 之间存在树状的派生关系. parents, children, siblings 等.

4. 构建 Octree

4.1. Root Octant

root octant 包含所有数据点. 构建方式:

- 遍历数据点, 找到数据点在 X 轴, Y 轴和 Z 轴这三个方向的边界值.
- 以最长的一端来设置 Octant 的大小.
- 根据边界值确定 Octant 的中心位置坐标.

4.2. 如何划分 child octant

- 计算新 octant 的 center
- 计算新 octant 的 extent
- 遍历 octant 中的点, 计算与 center 的相对坐标
- 对相对坐标的 xyz 进行分类, 8 个象限中的点, 放入 8 个不同的 octant.

4.3. 递归中止条件

- Octant 的大小.
- bucket size: Octant 所含数据点的多少.

3 https://en.wikipedia.org/wiki/Octant_(solid_geometry)

5. 搜索球(Query ball)

5.1. 实现思路概述

对搜索球与 Octant 可能关系进行完全分类:

1. Octant 与搜索球有重叠区域:
    1. 搜索球完全包含 Octant
        - 将 Octant 中的所有点纳入需要进行距离计算的点集.
    2. 搜索球不完全包含 Octant:
        - 将 Octant 切分为更小的 8 个 Octant, 递归处理下去.
2. Octant 与搜索球无重叠区域: 忽略该 Octant 中的所有点.

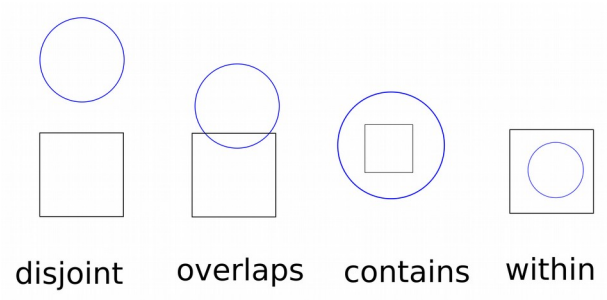

Figure 2: Octant 与 Query ball 关系的完全分类

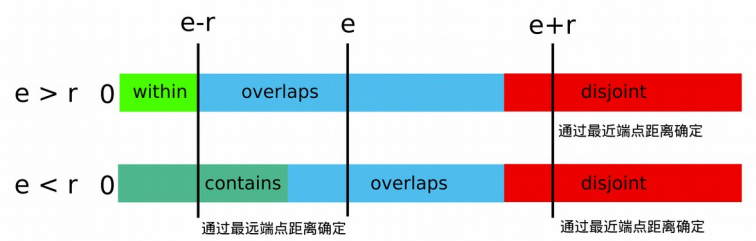

Figure 3: 逻辑分类关系

Query ball 与 Octant 关系的计算方式:

- 如果 query ball 大于 extent
    - disjoint
    - overlaps
    - contains
- 如果 query ball 小于 extent
    - disjoint
    - overlaps
    - within

关于搜索策略:

- 对于自根向叶的搜索, e 越来越小, 用 contains 判断, 可提前返回, 减少向下递归深度.
- 对于自叶向根的搜索, e 越来越大, 用 within 判断, 可提前返回, 可减少向上递归深度.

5.2. 判断重叠与否

1. 计算 q 在立方体坐标系下的相对坐标 q', 将其镜像到第一象限 (+, +, +).
2. 如果 q' 的坐标 xyz 中任一分量大于 $e + r$: 不重叠.

```
if (x > maxdist || y > maxdist || z > maxdist) return false;
```

3. 如果 q' 的坐标 xyz 中任一分量小于 e: 重叠

```
int32_t num_less_extent = (x < o->extent) + (y < o->extent) + (z < o-
>extent);
// a. inside the surface region of the octant.
if (num_less_extent > 1) return true;
```

4. 如果 q' 的坐标 xyz 中任一分量都在 e 和$ e+r$ 之间找到立方体中距离 q' 最近的端点 m(e, e, e), 计算 m 至 q' 的距离, 判断是否小于 r

```
// b. checking the corner region && edge region.
x = std::max(x - o->extent, 0.0f);
y = std::max(y - o->extent, 0.0f);
z = std::max(z - o->extent, 0.0f);
return (Distance::norm(x, y, z) < sqRadius);
```

问题:

- 对于 corner case, 是否全部当做 overlap 来处理, 带来的损耗不一定很大? 这样可以避免计算一次 distance. `[2018-03-06 Tue]` 经测试, 有微小改进, 几乎不可见.
- 原文中为了节省存储空间, octant 只保留 start_index 和 end_index. 这增加了代码的复杂度, 但对检索效率没有改进.

5.3. 判断包含与否

找到 Octang 中距离搜索球中心最远的边界点(对角), 计算其距离. 或将 q' 镜像到第一象限 (+, +, +), 求其与 (-, -, -) 对角点的距离.

```
float x = get<0>(query) - o->x;
float y = get<1>(query) - o->y;
float z = get<2>(query) - o->z;

x = std::abs(x);
y = std::abs(y);
z = std::abs(z);
// reminder: (x, y, z) - (-e, -e, -e) = (x, y, z) + (e, e, e)
x += o->extent;
y += o->extent;
z += o->extent;

return (Distance::norm(x, y, z) < sqRadius);
```

5.4. 特殊情况: 如果 query point 是 octree 中的一个数据点

这种情况下, 检索效率可以进一步提高.

- 所有的数据点分别属于大小不同的 Octant (leaf nodes).
- 根据 query point 的位置, 可以确定所归属的 Octant.
- 根据所归属的 octant, 可以快速判断邻近的 Octants.
- 逆向搜索: 从枝节点向根节点搜索.

6. 效率对比

SciPy 中的 KDTree [4] 是最主流的最近邻搜索算法. 对于 SciPy, 我们选择用 C 语言实现的最高性能版本 cKDTree.

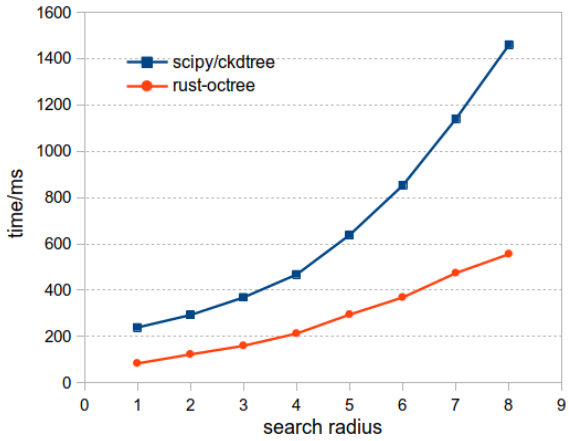

Figure 4: SciPy/ckdtree vs Rust-Octree

我们构建了一个包含约 50000 个坐标点的测试体系(examples/data/3wu2.xyz). 在单核运行模式下, 分别测试 Kdtree 和 Octree 在搜索半径由 1 至 9 时, 所用的总时间. 由 Figure 4 所示, 我们实现的 Octree 算法明显优于 Kdtree 算法.

7. 参考文献

- 原始文献: Behley20152IICRAI
- 原作者 Cpp 实现代码: [jbehley/octree: Fast radius neighbor search with an Octree (ICRA 2015)]

4 https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.cKDTree.html