

# 自动驾驶与机器人中的 SLAM技术

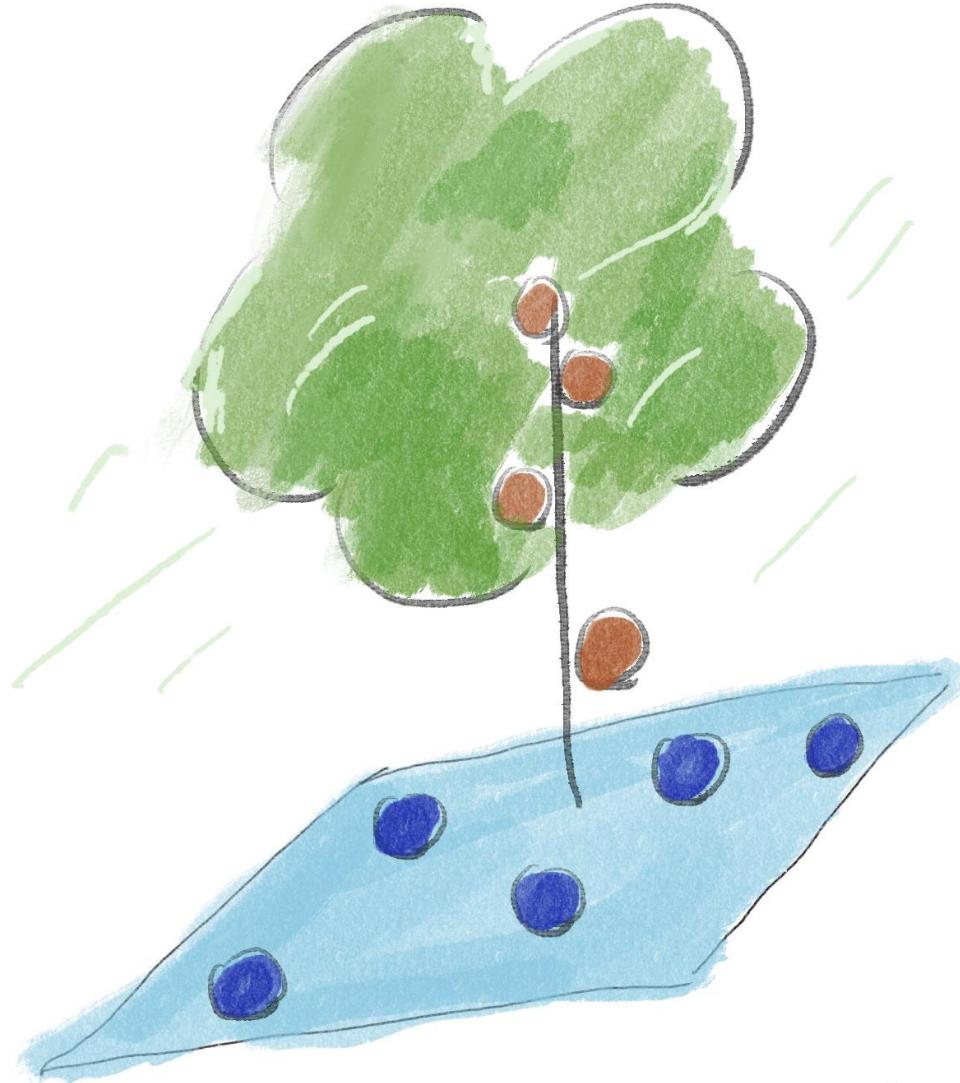
点云基础数据结构与最近邻





# Contents

- ❖ 激光雷达测量模型与点云
- ❖ 点云的最近邻问题
- ❖ 点云的拟合问题
- ❖ 点云其他表达方式的讨论



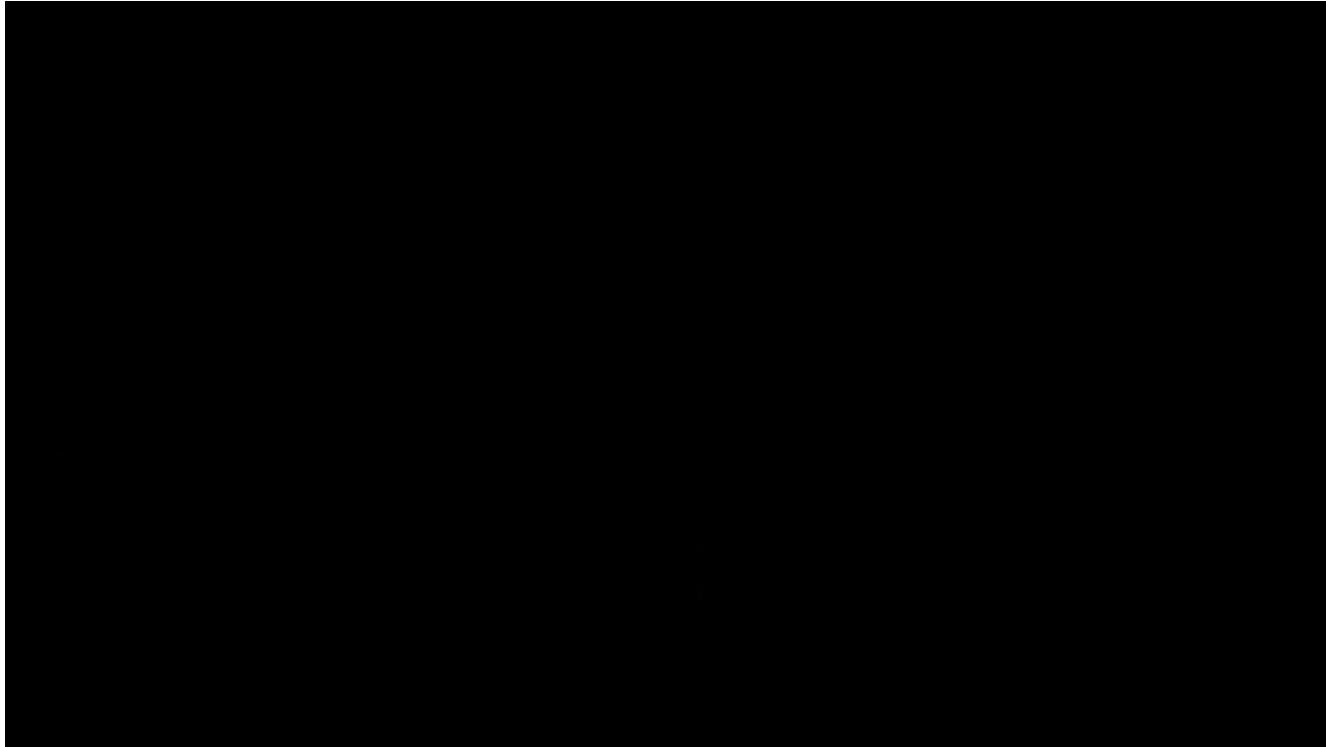
点云的相邻关系是许多算法的基础。

# 激光雷达测量模型与点云



## 激光雷达测量模型与点云

- 本节开始讨论自动驾驶与机器人中的主要传感器：激光雷达，本节与下两节课将介绍相对完整的2D/3D激光SLAM系统。
- 激光雷达是一种重要的测距传感器，通过测量来回脉冲的时间间隔来计算物体距离（ToF, Time-of-Flight），这些距离读数可以进一步转换为点云，作为物体空间结构信息的表达方式。





# 激光雷达测量模型与点云

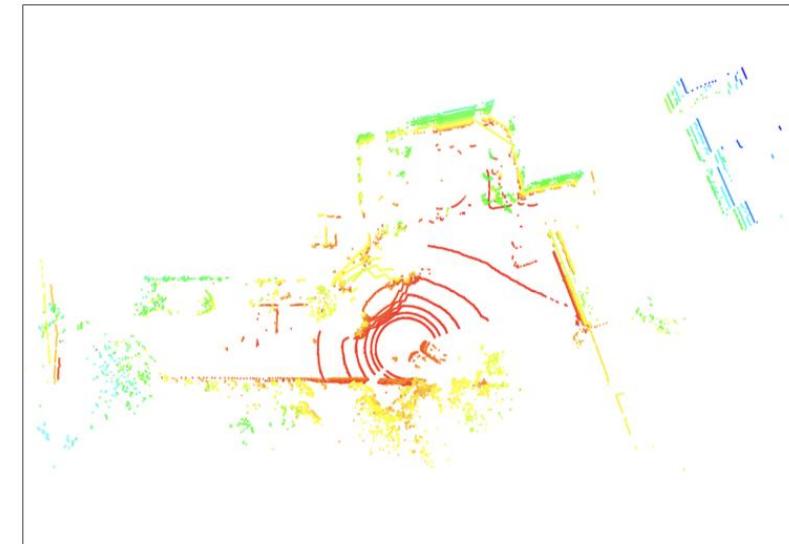
激光雷达可以分为机械旋转式雷达与固态雷达两种。

## □ 机械旋转式雷达

- 获取360度视野；
- 需要转动探头，通常在10hz下运行；
- 由于机电结构复杂，通常价格较昂贵，也无法通过车辆标准。



机械旋转式雷达 (spinning lidar)



一个scan数据



# 激光雷达测量模型与点云

## □ 固态雷达

- 不需要转动探头，可以转动镜片（转镜式）或者不转动（相控或Flash）；
- 进一步分为半固态与全固态两种。



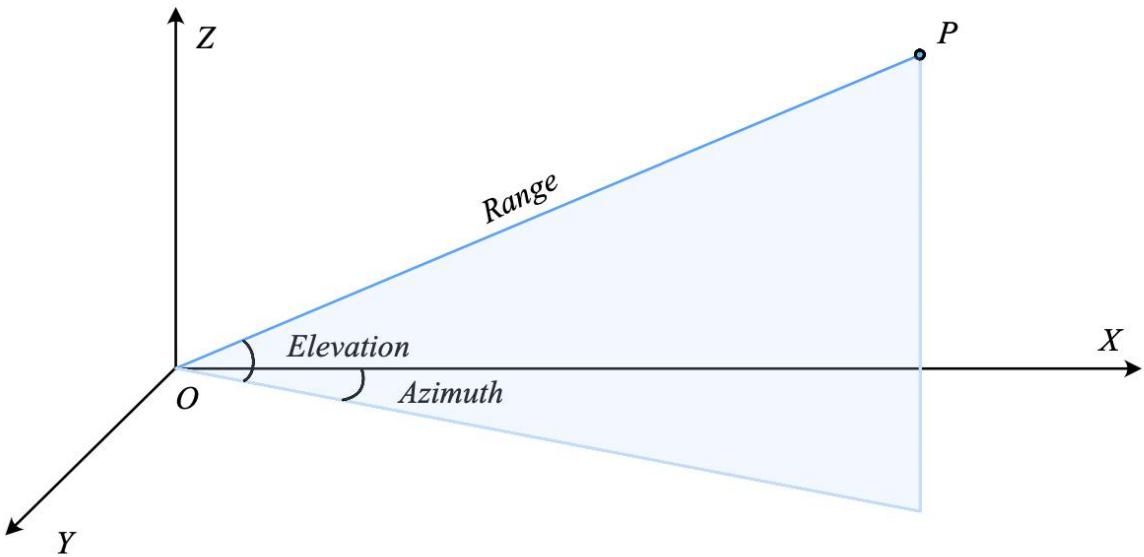
### 固态雷达特点：

- 相似的精度指标
- 有限的视野（120度左右）
- 更稠密的点云（等效线数更高）
- 更便宜，符合车规



# 激光雷达测量模型与点云

## □ 激光雷达的Range-Azimuth-Elevation (RAE) 测量模型



- 单个点的XYZ既可以用笛卡尔坐标描述，也可以用RAE坐标描述；
- RAE，即点的极坐标，也可以理解为俯仰和偏航角；
- 在多线雷达中，可以认为每条线对应的elevation角是固定的，azimuth随时间匀速变化，只有range是实际测量的数据。

$$\mathbf{P} = [r \cos E \cos A, r \cos E \sin A, r \sin E]^\top.$$

$$r = \sqrt{x^2 + y^2 + z^2},$$

$$A = \arctan(y/x),$$

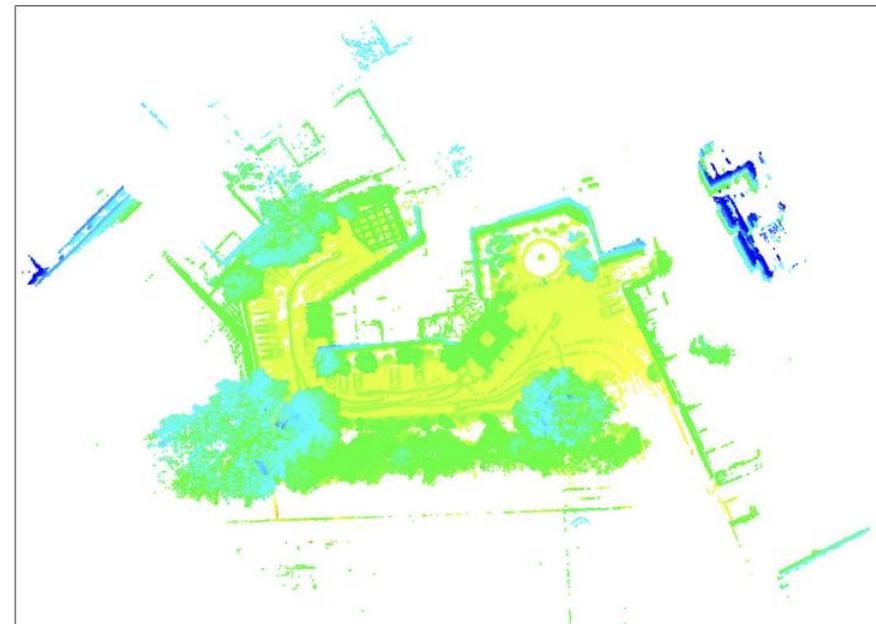
$$E = \arcsin(z/r).$$



# 激光雷达测量模型与点云

## □ 点云的表达

- 最基本的即是用一组xyz坐标表达整个点云，PCL中的点云即使用这种格式，点云字段可以自己定义；
- 常见的点云由xyz加上反射率组成；
- 大部分3D可视化软件都可以很好地支持点云显示。



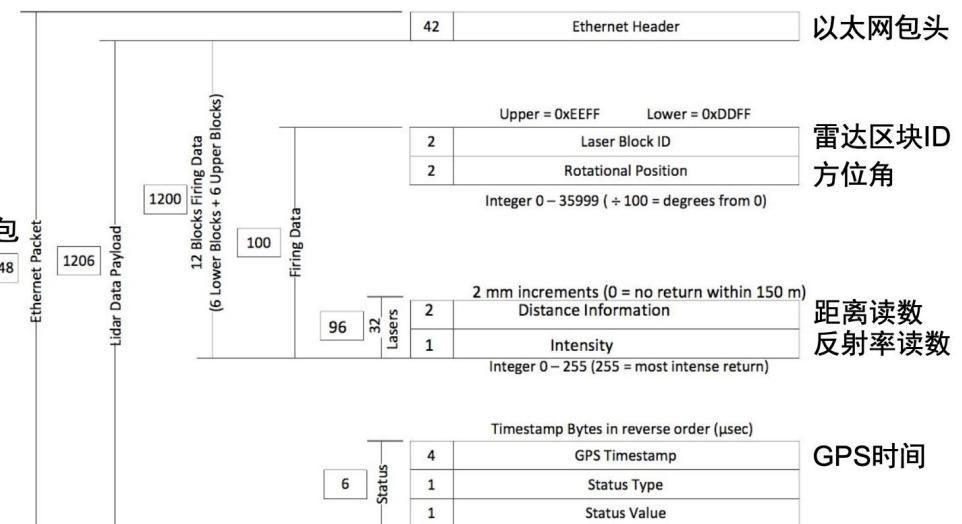
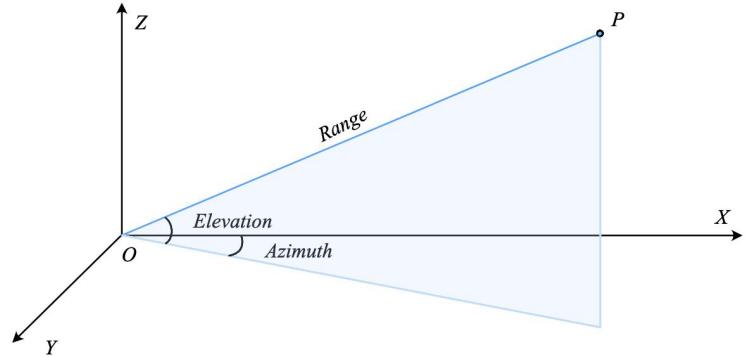
俯视视角下的一个点云地图



# 激光雷达测量模型与点云

## □ 点云的Packets表达

- 在RAE模型中，只有range是实测数据，其他两个维度都可以通过雷达参数得到；
- 可以利用该特点对点云数据进行压缩，即packets数据来表达点云：
  - 同一条线的测量数据还可以共享同一个azimuth，或者同一时刻的测量数据可以共享一个elevation；
  - 该例的32线数据只需要100个字节，而32个xyzi则需要416个字节。



Packets有更好的压缩比，通常用于通讯或存储

Packets是很多雷达硬件厂商提供的测量方式



# 激光雷达测量模型与点云

## □ 其他视角的激光雷达数据

- 大部分自动驾驶场景都是水平的，所以可以将点云投影到俯视图中处理；
- 将点云按照图像坐标进行转换，即可得到俯视图坐标。

$$\begin{cases} u = (x - c_x)/r + I_x, \\ v = (y - c_y)/r + I_y. \end{cases}$$

虚拟相机的焦距、图像中心和点云中心

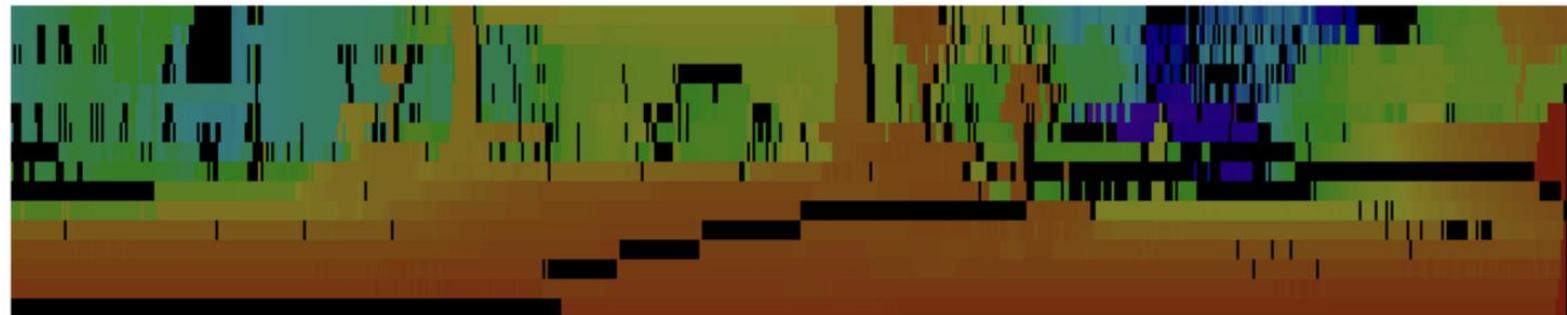




# 激光雷达测量模型与点云

## □ Range image

- 横坐标为点云azimuth角，纵坐标为点云线数，读数为距离测量值；
- 在range image中进行纵向物体聚类可以快速分割垂直形状的物体。
  - 实际上是用图像中像素的近邻关系代替了空间近邻关系

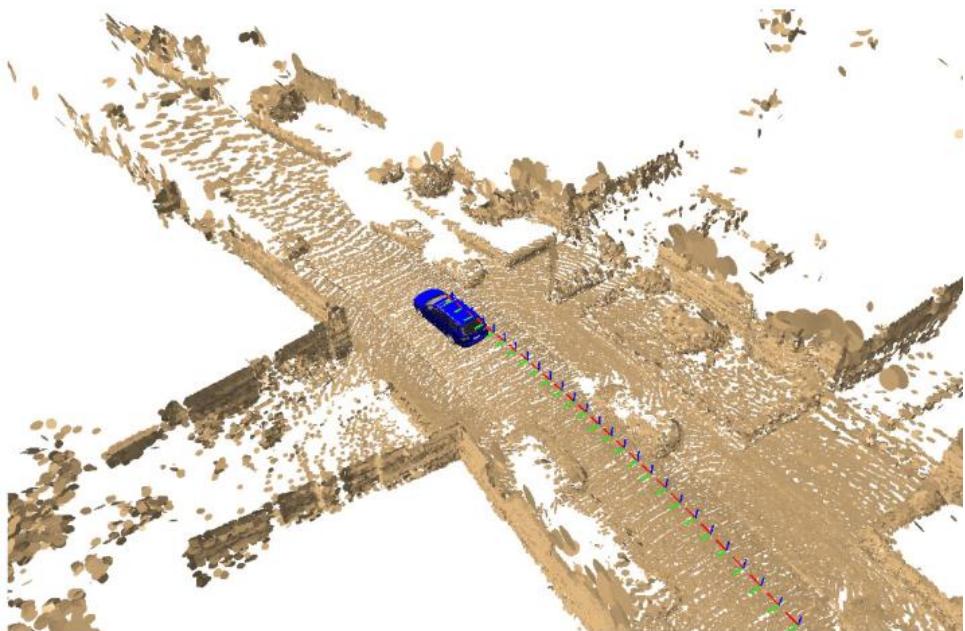




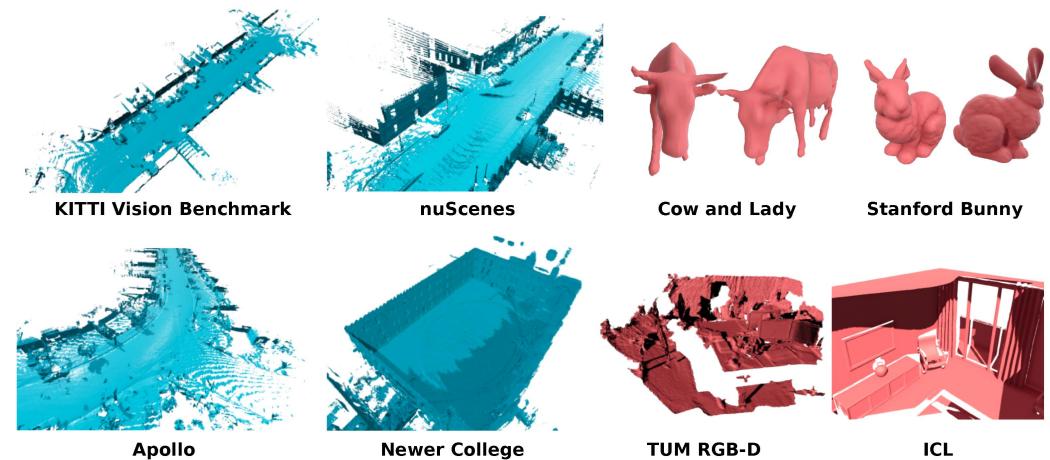
# 激光雷达测量模型与点云

## □ 其他的表达方式

- 许多VSLAM中的表达方式都可以扩展到点云，例如surfels, TSDF



Surfel: 由面片组成的地图



SDF and TSDF



# 激光雷达测量模型与点云

程序实验

点云的近邻问题 (kNN)



## 点云的近邻问题 (kNN)

近邻是最基本的点云关系，但近邻问题并不像看上去那么简单。

- 最近邻问题：查找某个点在一堆点中的最近邻。
- K近邻问题 (kNN) : 查找某个点在一堆点中k个最近邻。
  - ANN (approximate NN) : 查找近似的kNN



# 点云的近邻问题 (kNN)

## □ Brute-force kNN

- 最简单的遍历搜索
- 很容易并发，可以作为真值校验，也可以实现GPU版本

**暴力最近邻搜索** 给定点云  $\mathcal{X}$  和待查找点  $x_m$ ，计算  $x_m$  与  $\mathcal{X}$  中每个点的距离，并给出最小距离。

### 暴力 $k$ 近邻 (BF kNN)

1. 对给定点云  $\mathcal{X}$  和查找点  $x_m$ ，计算  $x_m$  对所有  $\mathcal{X}$  点的距离；
2. 对第 1 步的结果排序；
3. 选择  $k$  个最近的点；
4. 对所有  $x_m$  重复 1-3 步骤。



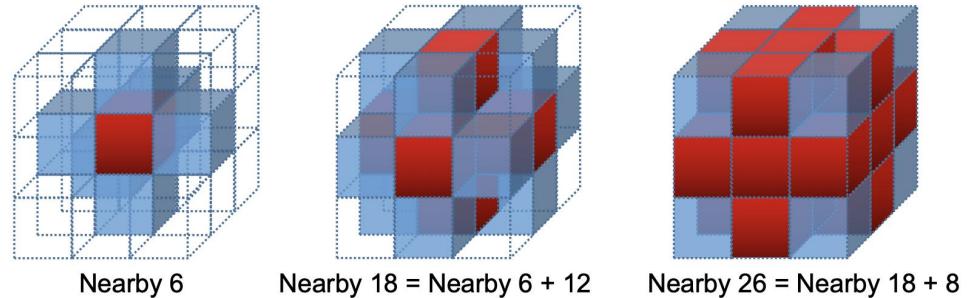
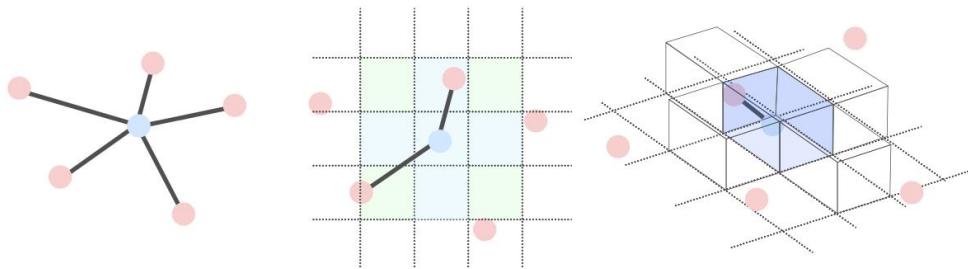
## 点云的近邻问题 (kNN)

- 暴力最近邻相当于将点云视为未排序的数据进行查找操作。
- 如果数组已经事先处理（例如排序、分割），那么查找操作还可以加速。
- 现在尝试先对空间进行分割，然后用空间索引来加速对每个点进行查找的思路。
  - 根据维度不同，可以分为2D栅格方法和3D体素方法



# 点云的近邻问题 (kNN)

## □ 棚格方法



- 将空间均匀分成网格或体素；
- 近邻查找时，先查找所在的栅格或体素ID，再查找内部最近邻ID；
- 栅格本身的邻近关系提供了点的有限搜索范围，加快了点的最近邻搜索。



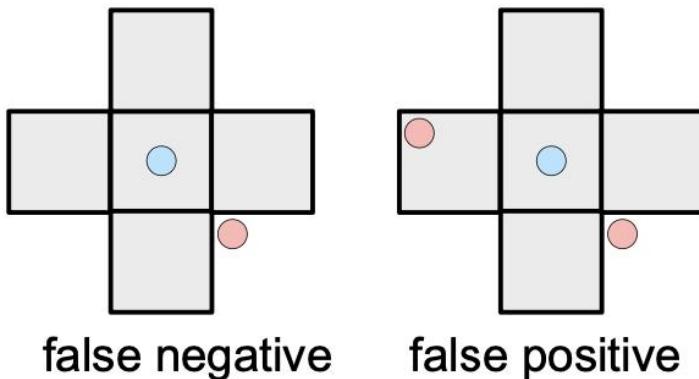
# 点云的近邻问题 (kNN)

## □ 最近邻算法的效果评估

- 效率：计算每个点的平均查询时间 (Query per second, QPS)；
- 精度：每个最近邻是否为真实最近邻的评价指标；

准召率指标： $\text{precision} = \frac{1 - FP}{m}$ ,  $\text{recall} = \frac{1 - FN}{n}$

- Brute-force kNN (BFNN) 可以看成是准召都为100%，但效率很低的最近邻方法；
- 大部分算法可以在较好的准召率下（不一定100%），达到较高的效率指标。



- 栅格法可能存在的错误最近邻示例
- 落在栅格边界处的点很容易出现错误的最近邻
- 近邻栅格越多，出现错误的概率越低，但效率也会降低

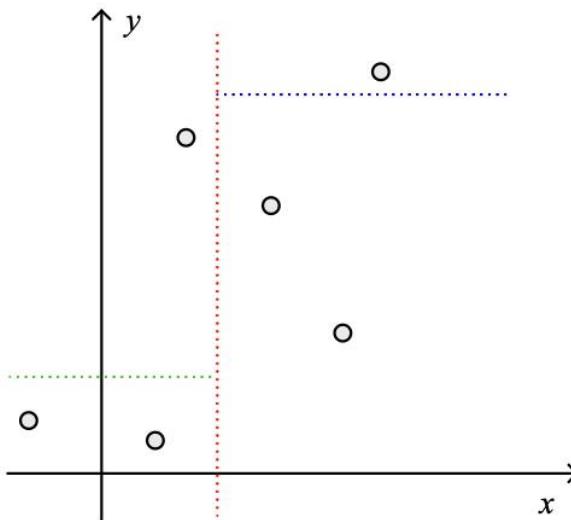
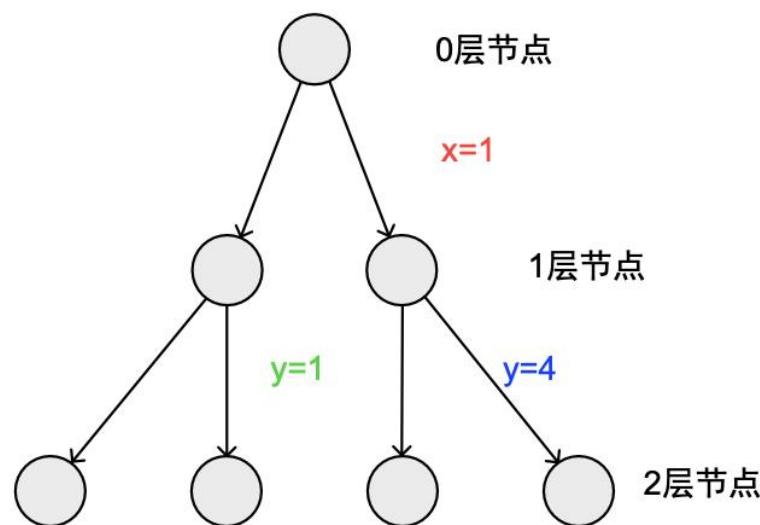


# 点云的近邻问题 (kNN)

□ K-d树：类似于对数组进行排序后再查找的过程，对数复杂度

思路：每次都将3D空间分为两半，一半点云落在左侧，另一半落在右侧

问题：1. 如何定义两半？2. 如何建树？3. 如何查找？

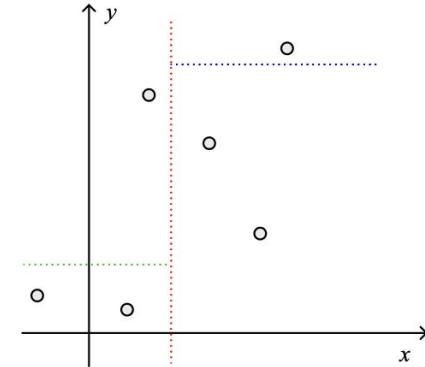
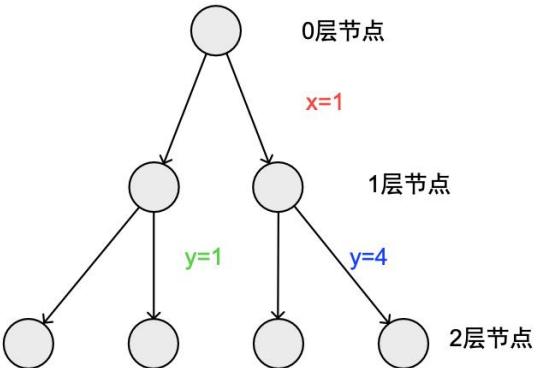




# 点云的近邻问题 (kNN)

## □ K-d树定义：

1. 每个节点有左右两个分枝；
2. 叶子节点表示实际点云（或索引）；
3. 非叶子节点保存一个分割轴和分割阈值，表示沿该轴的某个数值进行分割。



K-d 树的构建：

1. 输入：点云数据  $\mathbf{X} = \mathbf{x}_1, \dots, \mathbf{x}_n$ , 其中  $\mathbf{x}_i \in \mathbb{R}^k$ .
2. 考虑将子集  $\mathbf{X}_n \subset \mathbf{X}$  插入节点  $n$ :
3. 如果  $\mathbf{X}_n$  为空，退出；
4. 如果  $\mathbf{X}_n$  只有一个点，记为叶子节点，退出；
5. 计算  $\mathbf{X}_n$  在各轴方差，挑选分布最大的一个轴，记为  $j$ ；取平均数  $m_j = \mathbf{X}_n[j]$  作为分割阈值。
6. 遍历  $\mathbf{x} \in \mathbf{X}_n$ , 对于  $\mathbf{x}[j] < m_j$  的，插入左节点；否则插入右节点。
7. 递归上述步骤直到所有点都被插入至树中。

在计算机中，树形数据结构都可以由递归来实现，且代码非常简洁，这种称为axis-aligned k-d tree。



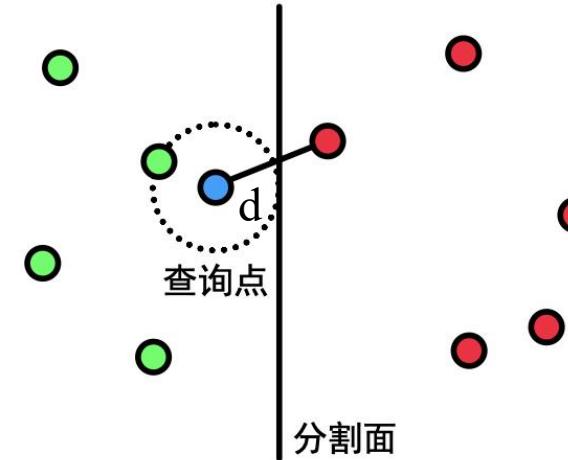
# 点云的近邻问题 (kNN)

## □ K-d树的查找

- K-d树查找比BFNN快的本质原因是剪枝；
- 由于分割面的存在，在查询点另一侧的分枝存在一个最小距离 $d$ ；
- 若在同侧找到一个比 $d$ 更小的最近邻，则另一侧分枝可以直接略过，相当于剪枝。

K-d 树的最近邻查找：

1. 输入：K-d 树  $T$ ，查找点  $x$ ；
2. 输出： $x$  的最近邻；
3. 记当前节点为  $n_c$ ，最初取  $n_c$  为根节点。记  $d$  为当前搜索到的最小距离。
  - (a) 如果  $n_c$  是叶子，计算  $n_c$  与  $x$  的距离。看它是否小于  $d$ ；若是，记  $n_c$  为最近邻，回退到它的父节点。
  - (b) 如果  $n_c$  不是叶子，计算  $x$  落在  $n_c$  的哪一侧。若  $x$  所在的一侧未被展开，优先展开  $x$  所在的一侧。
  - (c) 计算是否需要展开  $n_c$  的另一侧。记  $x$  与  $n_c$  分割面的距离为  $d'$ 。若  $d' < d$  时，必须展开另一侧；否则跳过另一侧分枝；
  - (d) 如果两侧都已经展开，或者不必展开，则返回上一节点，直到  $n_c$  变为根节点。



查询点距离与 $d$ 谁更近是一个运气问题；  
所以K-d树在最差情况下可能要比BFNN更慢，  
不过通常意义下还是要快很多。



# 点云的近邻问题 (kNN)

## □ K-d树的kNN可以从单点NN拓展而来

- 此时要比较kNN中最远点与 $d$ 的距离

K-d树的ANN:

$$d_{split} > \alpha d_{max}$$

分割面      最远的NN

K-d 树的 K 近邻查找:

- 输入:  $K - d$  树  $T$ , 查找点  $x$ , 最近邻数  $k$ ;
- 输出:  $k$  近邻集合  $N$ ;
- 记当前节点为  $n_c$ , 最初取  $n_c$  为根节点。记函数  $S(n_c)$  表示在  $n_c$  下进行  $k$  近邻搜索:
  - 如果  $n_c$  是叶子, 计算  $n_c$  与  $x$  的距离是否小于  $N$  中最大距离; 若是, 将  $n_c$  放入  $N$ 。若此时  $|N| > k$ , 删除  $N$  中距离最大的匹配点;
  - 计算  $x$  落在  $n_c$  的哪一侧。递归调用  $S(n_c.left)$  或  $S(n_c.right)$ 。
  - 计算是否需要展开  $n_c$  的另一侧。展开的条件判定:  $|N| < k$  时, 必须展开;  $|N| = k$  且  $x$  与  $n_c$  的分割面距离小于  $N$  中最大匹配距离, 也进行展开;
  - 若  $n_c$  的另一侧不需要展开, 函数返回; 否则继续调用另一侧的近邻搜索算法。

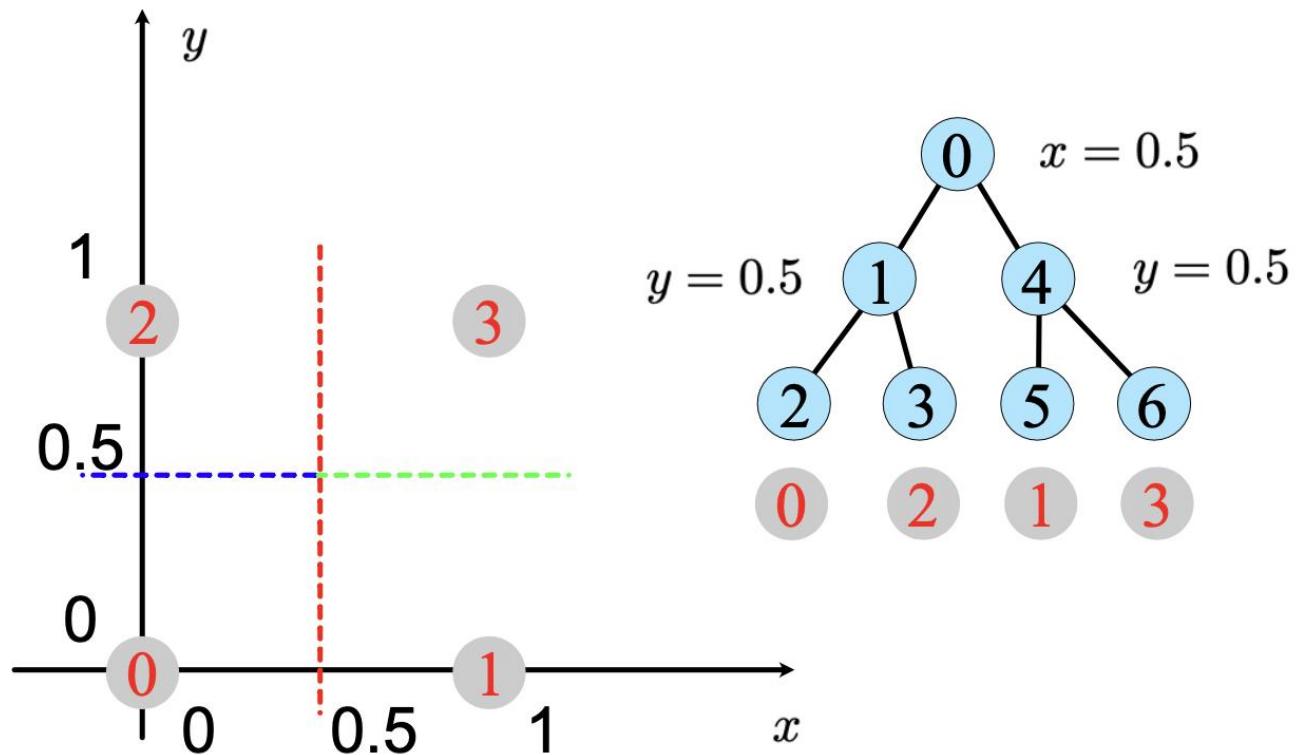
Alpha<1时, 可以加快剪枝, 但会损失准召率



# 点云的近邻问题 (kNN)

## □ 2d栅格、3d栅格、K-d树的代码实现

- 注意我们会优化使用并发算法

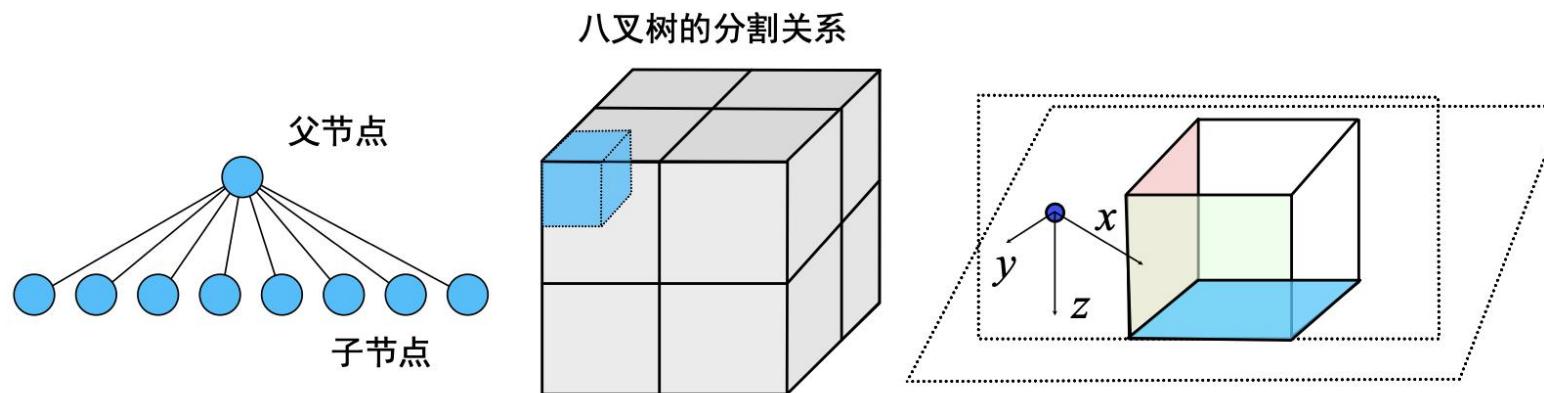




# 点云的近邻问题 (kNN)

## □ 四叉树与八叉树

- 与K-d树类似，仍然是对空间进行划分；
- K-d树通过平面划分，四叉树与八叉树是均匀划分，分别对应2D和3D的情形；
- 以八叉树为例，建树和查找的过程与K-d树十分相似。



此时，查询点离八叉树的包围盒距离下界可以取x轴距离



# 点云的近邻问题 (kNN)

## □ 八叉树的建树与kNN查找

八叉树的构建:

1. 输入: 点云数据  $\mathbf{X} = \mathbf{x}_1, \dots, \mathbf{x}_n$ , 其中  $\mathbf{x}_i \in \mathbb{R}^k$ .
2. 考虑将子集  $\mathbf{X}_n \in \mathbf{X}$  插入节点  $n$ :
3. 如果  $\mathbf{X}_n$  为空, 退出;
4. 如果  $\mathbf{X}_n$  只有一个点, 记为叶子节点, 退出;
5. 按照一分为八的准则对  $n$  进行展开。
6. 遍历  $\mathbf{x} \in \mathbf{X}_n$ , 记录  $\mathbf{x}$  落在哪一个子节点。然后对子节点和对应点云递归调用构建方法。
7. 递归上述步骤直到所有点都被插入至树中。



# 点云的近邻问题 (kNN)

## □ 八叉树的建树与kNN查找

八叉树的 k 近邻查找：

1. 输入：八叉树  $T$ , 查找点  $x$ , 最近邻数  $k$ ;
2. 输出： $k$  近邻集合  $N$ ;
3. 记当前节点为  $n_c$ , 最初取  $n_c$  为根节点。记函数  $S(n_c)$  表示在  $n_c$  下进行  $k$  近邻搜索：
  - (a)如果  $n_c$  是叶子, 计算  $n_c$  与  $x$  的距离是否小于  $N$  中最大距离; 若是, 将  $n_c$  放入  $N$ 。若此时  $|N| > k$ , 删除  $N$  中距离最大的匹配点;
  - (b)计算  $x$  落在  $n_c$  的哪一个子节点。如果  $x$  在  $n_c$  的边界盒外面, 则展开每一个子节点; 若落在内部, 则优先展开  $x$  所在的子节点。
  - (c)计算是否需要展开  $n_c$  的其他子节点。展开的条件判定:  $|N| < k$  时, 必须展开;  $|N| = k$  且  $x$  与  $n_c$  的前面所述距离计算结果小于  $N$  中最大匹配距离, 也进行展开;
  - (d)若  $n_c$  的子节点不需要展开, 函数返回; 否则继续调用其他节点的近邻搜索算法。

同样, 八叉树也可以实现类似的ANN查找方法



# 点云的近邻问题 (kNN)

## □ 其他树类方法以及K-d树的改进

- K-d树本身可以有更紧凑的表示；
- 空间划分方式可以有很多种 (B树, R树, AABB树)；
- SFC (space filling curve) 曲线：伪Hilbert曲线、Z曲线、Dragon曲线、Gosper曲线；
- LSH (locality sensitive hashing) 或空间哈希；
- 大部分数据结构更适用于静态空间数据的查询，通常用来制作数据库，而SLAM或LO更需要能够适应快速动态变化的数据结构。

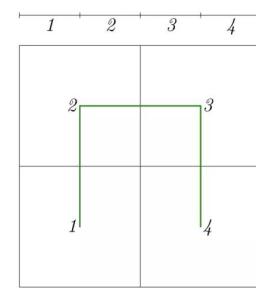


Fig. 1.

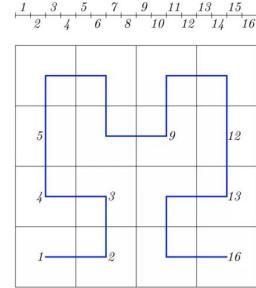


Fig. 2.

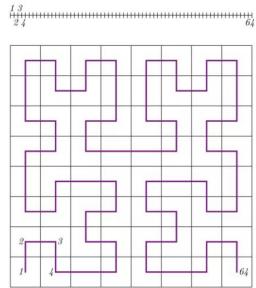
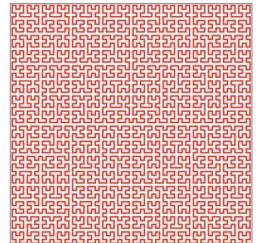
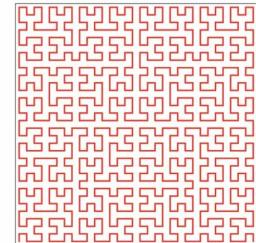
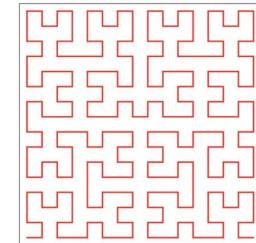


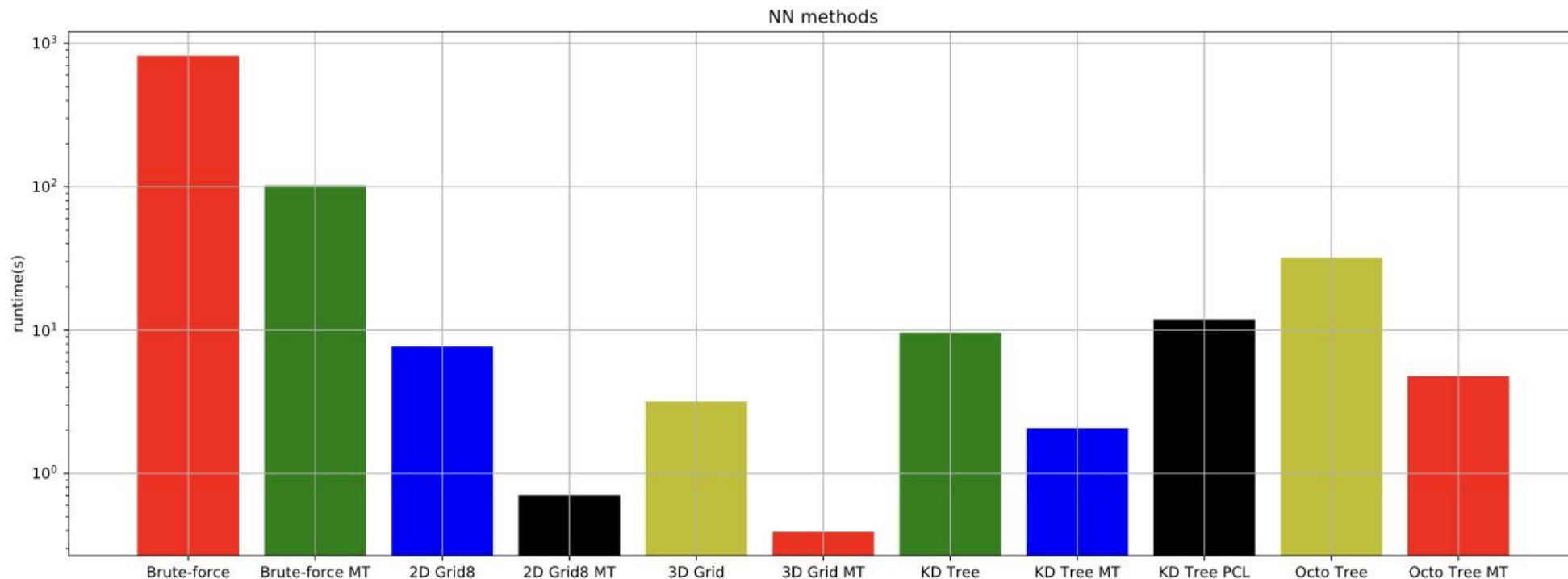
Fig. 3.





# 点云的近邻问题 (kNN)

## □ 程序实现以及结果对比



# 点云的拟合问题



# 点云的拟合问题

□ 我们通常认为一个点和它附近的点组成了某种形状：

- 该形状可以是基本的几何元素：直线、平面、立方体；
- 也可以是更复杂的几何体：二次曲面、样条线、SDF等；
- 也可以是某种统计分布：高斯分布等；
- 也可以是非参数化表示的某种形式。

其中最简单的就是将一组点进行线性拟合；

□ 下面来推导平面拟合和直线拟合过程，并讨论它们的联系。



# 点云的拟合问题

## □ 平面拟合

给定一组点:  $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ ,

寻找平面参数  $\mathbf{n}, d$ , 使得:  $\forall k \in [1, n], \mathbf{n}^\top \mathbf{x}_k + d = 0$ ,

显然有4个未知量, 一个点提供1个方程;

还有  $\|\mathbf{n}\| = 1$  的约束, 故未知量有3个自由度, 至少需要3个点才能确定一个平面;

通常点的数量要更大, 组成一个超定方程组, 通过最小二乘解来求解:

$$\min_{\mathbf{n}, d} \sum_{k=1}^n \|\mathbf{n}^\top \mathbf{x}_k + d\|_2^2.$$



## 点云的拟合问题

□ 取齐次坐标:  $\tilde{\mathbf{x}} = [\mathbf{x}^\top, 1]^\top \in \mathbb{R}^4$ .  $\tilde{\mathbf{n}} = [\mathbf{n}^\top, d]^\top \in \mathbb{R}^4$

□ 那么可以写成:  $\min_{\tilde{\mathbf{n}}} \sum_{k=1}^n \|\tilde{\mathbf{x}}_k^\top \tilde{\mathbf{n}}\|_2^2$ . 且  $\mathbf{n}$  不为零

□ 还可以写成矩阵形式:  $\tilde{\mathbf{X}} = [\tilde{\mathbf{x}}_1, \dots, \tilde{\mathbf{x}}_n]$ ,  $\min_{\tilde{\mathbf{n}}} \|\tilde{\mathbf{X}}^\top \tilde{\mathbf{n}}\|_2^2$ .

□ 转换为通用的线性代数问题: 对任意矩阵  $A$ , 寻找  $x$ , 使得  $Ax$  最小化

- 通常限制  $\|x\| = 1$
- 此时  $A$  为  $n \times 4$  矩阵,  $x$  为 4 维矢量



# 点云的拟合问题

□ 一般的代数解法:  $x^* = \arg \min_x \|Ax\|_2^2 = \arg \min_x x^\top A^\top Ax, \quad s.t., \|x\| = 1.$

□ 该问题有两种视角: 特征值分解和奇异值分解

□ 因为  $A^\top A$  是实对称的, 它必然存在特征值分解:

$$A^\top A = V \Lambda V^{-1} \quad V \text{的列向量为: } v_1, \dots, v_n \text{ 是一组单位正交基}$$

□  $x$  在这组基下的表示为:  $x = \alpha_1 v_1 + \dots + \alpha_n v_n,$

□ 于是  $\|Ax\|_2^2 = \sum_{k=1}^n \lambda_k \alpha_k^2$ , 且:  $\alpha_1^2 + \dots + \alpha_n^2 = 1$

□ 设特征值是降序排列的, 于是取:  $\alpha_n = 1$ , 其它为0, 目标最小化

□ 即得  $x^* = v_n$ , 即最小特征值向量



# 点云的拟合问题

□ 另一方面:  $A = U\Sigma V^\top$   $U, V$  正交, 中间为对角的奇异值阵

□ 此时:  $x^\top A^\top Ax = x^\top V \Sigma^2 V^\top x.$

□ 同样,  $x$  可取  $V$  最后一列, 让目标函数最小化

□ 所以,  $x$  也就是奇异值分解后的  $V$  阵最后一列

- 实际当中甚至不需要完全计算SVD, 只需计算最小奇异值对应的  $V$  向量
- 上述推导表明了特征值分解和奇异值分解在本问题中的联系



# 点云的拟合问题

## □ 直线拟合

- 相比平面，直线要更复杂一些
- 我们使用点+方向的方式描述一个直线：

$$\boldsymbol{x} = \boldsymbol{dt} + \boldsymbol{p},$$

- 直线外一点与直线的垂直距离：

$$f_k^2 = \|\boldsymbol{x}_k - \boldsymbol{p}\|^2 - \|(\boldsymbol{x}_k - \boldsymbol{p})^\top \boldsymbol{d}\|^2,$$

- 估计直线参数，优化该平方距离：

$$(\boldsymbol{d}, \boldsymbol{p})^* = \arg \min_{\boldsymbol{d}, \boldsymbol{p}} \sum_{k=1}^n f_k^2, \quad s.t. \|\boldsymbol{d}\| = 1.$$



# 点云的拟合问题

□ 目标函数对直线参数的导数：

$$(\mathbf{d}, \mathbf{p})^* = \arg \min_{\mathbf{d}, \mathbf{p}} \sum_{k=1}^n f_k^2, \quad s.t. \|\mathbf{d}\| = 1.$$

$$\begin{aligned} \frac{\partial f_k^2}{\partial \mathbf{p}} &= -2(\mathbf{x}_k - \mathbf{p}) + 2 \underbrace{(\mathbf{x}_k - \mathbf{p})^\top \mathbf{d}}_{\text{标量, } = \mathbf{d}^\top (\mathbf{x}_k - \mathbf{p})} \mathbf{d}, \\ &= (-2)(\mathbf{I} - \mathbf{d}\mathbf{d}^\top)(\mathbf{x}_k - \mathbf{p}). \end{aligned}$$

□ 求和以后：

$$\begin{aligned} \frac{\partial \sum_{k=1}^n f_k^2}{\partial \mathbf{p}} &= \sum_{k=1}^n (-2)(\mathbf{I} - \mathbf{d}\mathbf{d}^\top)(\mathbf{x}_k - \mathbf{p}), \\ &= (-2)(\mathbf{I} - \mathbf{d}\mathbf{d}^\top) \sum_{k=1}^n (\mathbf{x}_k - \mathbf{p}). \end{aligned}$$

□ 令它为零，则：

$$\mathbf{p} = \frac{1}{n} \sum_{k=1}^n \mathbf{x}_k, \quad \text{说明 } \mathbf{p} \text{ 应该取点云的中心}$$



# 点云的拟合问题

□ 因为  $p$  已知，所以定义：  $\mathbf{y}_k = \mathbf{x}_k - \mathbf{p}$

原先的  $f_k^2 = \|\mathbf{x}_k - \mathbf{p}\|^2 - \|(\mathbf{x}_k - \mathbf{p})^\top \mathbf{d}\|^2$ , 变为：  $f_k^2 = \mathbf{y}_k^\top \mathbf{y}_k - \mathbf{d}^\top \mathbf{y}_k \mathbf{y}_k^\top \mathbf{d}$ .

□ 因此  $\mathbf{d}^* = \arg \max_{\mathbf{d}} \sum_{k=1}^n \mathbf{d}^\top \mathbf{y}_k \mathbf{y}_k^\top \mathbf{d} = \sum_{k=1}^n \|\mathbf{y}_k^\top \mathbf{d}\|_2^2$ .

□ 它的矩阵形式：

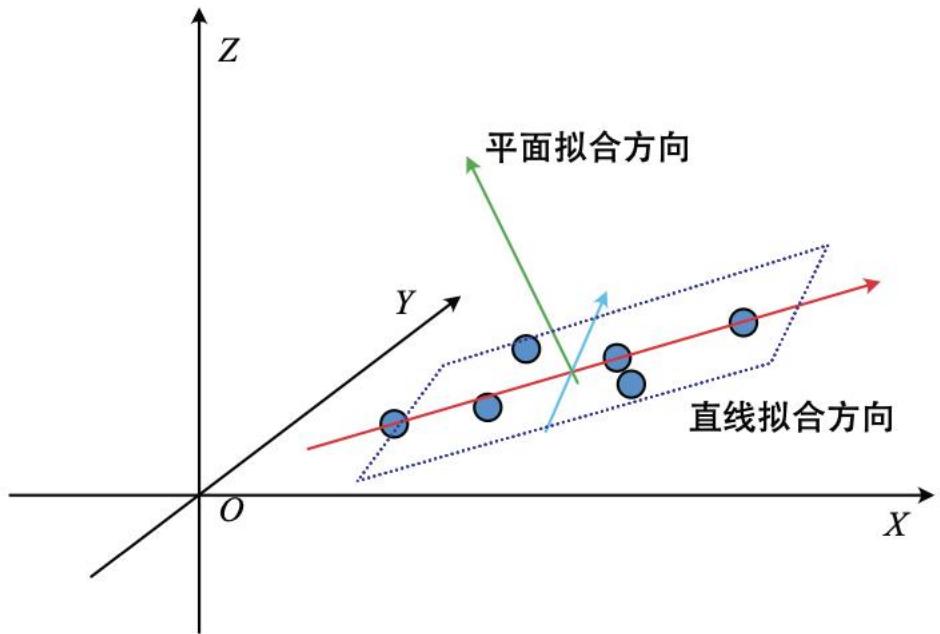
$$\mathbf{A} = \begin{bmatrix} \mathbf{y}_1^\top \\ \dots \\ \mathbf{y}_n^\top \end{bmatrix}, \quad \mathbf{d}^* = \arg \max_{\mathbf{d}} \|\mathbf{A}\mathbf{d}\|_2^2.$$

该问题和平面拟合是一样的，只是变成了最大化！  
于是， $\mathbf{d}$  应该取  $\mathbf{A}^\top \mathbf{A}$  的最大特征向量，或  $\mathbf{A}$  的最大右奇异向量



# 点云的拟合问题

□ 从SVD角度来看，平面法向和直线方向正好对应点云的主成分方向



- 直线是第一个主成分（散布最大）
- 平面是第二个主成分
- 该结论与PCA一致（主成分分析）
- 由于都是线性的问题，不需要引入迭代最小二乘的方法来求解



# 点云的拟合问题

□ 程序实现

## 点云其他实现方式的讨论



# 点云其他实现方式的讨论

□ 传统方法大多以点+附加数据结构来表示一个点云地图，例如点云+K-d树/体素来实现kNN查找，并进一步用参数信息来表达点云组成的几何体。这种方法的缺点：

1. 单纯点云并不含连接信息；
2. 体素类方法通常有体素个数限制（内存限制）和分辨率限制；
3. 几何体只能限定在非常简单的类型；
4. 缺乏对点云更进一步的描述（高层级，连续性）。

□ 事实上，除了显式参数化描述一个点云以外，还可以用隐式的非参数化形式表达一个点云。

□ 类比来说，点云就像人类语言中的字母。单个字母本身没有太多意义，它们排列之后组成的单词才有意义。然而传统方法并没有描述单词结构以及单词关系的方式。



## 点云其他实现方式的讨论

- 另一方面，当我们查询点云地图的最近邻或占据概率时，实际上需要的是一个查询函数：

该处是否有点

点云地图：  $o : \mathbb{R}^3 \rightarrow \{0, 1\}$  或者，近邻查找给出若干个最近邻/平面信息  
查询位置

- 这是一个典型的映射，自然也可以通过神经网络来学习，该主题对应点云的**Neural implicit representation**，和**NeRF**关联性很强；
- 由于点云已经给定，训练数据可以很容易生成出来，不需要标注。



# 点云其他实现方式的讨论

## □ 学习占据栅格的案例：

- Occupancy networks



Mescheder L, Oechsle M, Niemeyer M, et al. Occupancy networks: Learning 3d reconstruction in function space[C]//Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019: 4460-4470.  
[https://github.com/autonomousvision/occupancy\\_networks](https://github.com/autonomousvision/occupancy_networks)

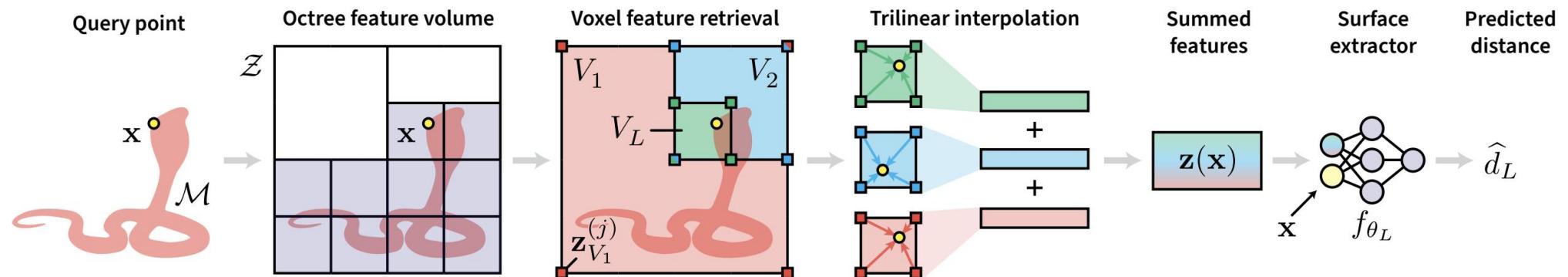


# 点云其他实现方式的讨论

## □ NeRF and Neural SDFs

- NGLOD
- Shine-mapping
- NeRF-LOAM

与传统NeRF不同，激光点云不带有色彩信息，通常学习SDF函数；  
NeRF大多数用来渲染，但很少被用来作配准、定位使用。



- Takikawa T, Litalien J, Yin K, et al. Neural geometric level of detail: Real-time rendering with implicit 3D shapes[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021: 11358-11367.
- Zhong X, Pan Y, Behley J, et al. SHINE-Mapping: Large-Scale 3D Mapping Using Sparse Hierarchical Implicit Neural Representations[J]. arXiv preprint arXiv:2210.02299, 2022.
- Deng J, Chen X, Xia S, et al. NeRF-LOAM: Neural Implicit Representation for Large-Scale Incremental LiDAR Odometry and Mapping[J]. arXiv preprint arXiv:2303.10709, 2023.



## 习题

1. 在三维体素中定义 NEARBY14，实现 14 格最近邻的查找。
2. 推导下式的解为  $A^T A$  的最大特征向量或奇异向量。

$$\mathbf{d}^* = \arg \max_{\mathbf{d}} \|\mathbf{A}\mathbf{d}\|_2^2$$

3. 将本节的最近邻算法与一些常见的近似最近邻算法进行对比，比如nanoflann，给出精度指标和时间效率指标。

J. L. Blanco and P. K. Rai, “nanoflann: a C++ header-only fork of FLANN, a library for nearest neighbor (NN) with kd-trees.”

<https://github.com/jlblancoc/nanoflann>, 2014.

感谢聆听！  
Thanks for Listening

