# 近傍点探索の条件緩和によるICPマッチングの高速化の検討

谷研究室　花見 桜

2022.2.14 日本大学文理学部情報科学科卒業演習発表会

# 目次

# 1. はじめに

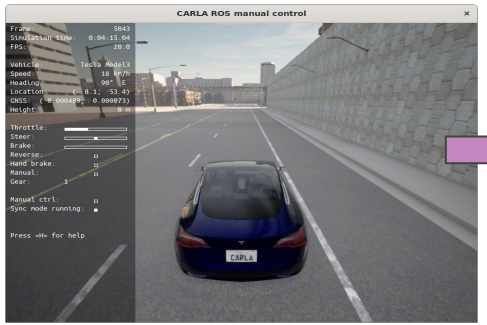
# 1.1 点群とは

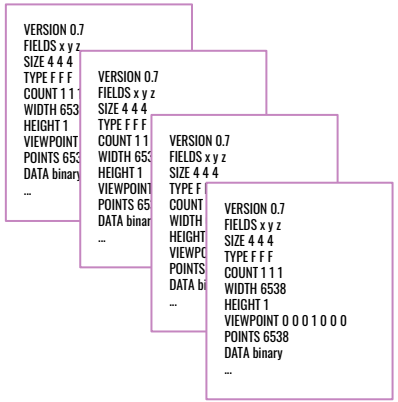
点の集まり

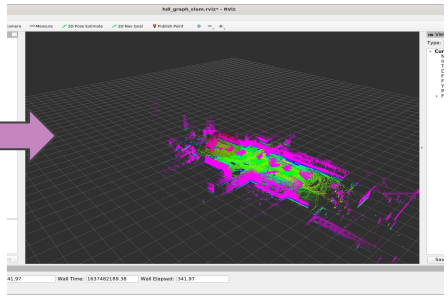
3次元座標値 ( x, y, z)，色情報 ( r, g, b)から構成される.

環境地図の生成・3Dモデル化に利用

# 環境地図の生成の例



カメラで撮る

点群データにする

環境地図を生成

# 1.2 SLAMとは (Simulataneous Localization And Mapping)

同時に自己位置推定と環境地図構成を行うこと

複数の点群データ内にある x, y, z 座標をつなぎ合わせて

環境地図座標を求めていく

複数の点群データから 同じ部分を見つけ出す

マッチング処理

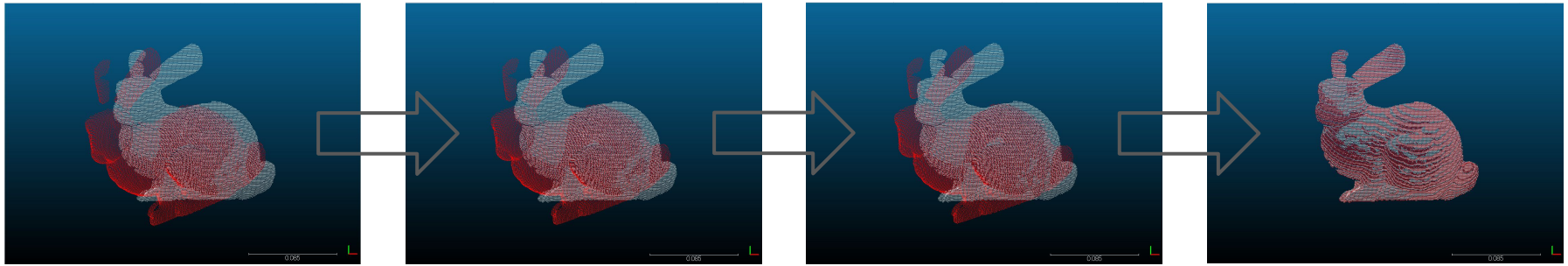
# 1.3 ICPアルゴリズムとは

SLAMで利用される代表的なマッチング処理アルゴリズム

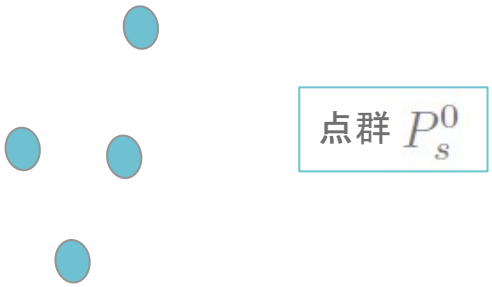
異なる位置から撮影した点群同士の位置関係を推定し点群のマッチングを行い，

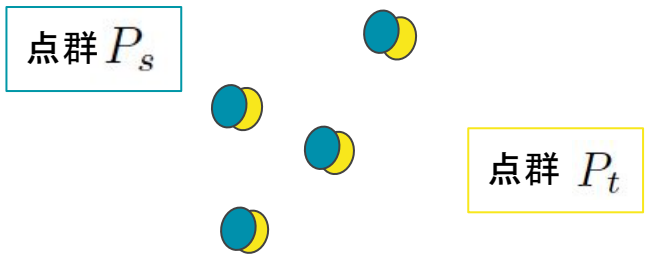
対応点同士がなるべく近くなるように　向きを保ち等距離移動

- 回転 ( 中心は原点 )
- 平行移動



# ICPアルゴリズム




入力

2つの点群データ

- ソース点群　$P_s^0 = \{s_1^0, s_2^0, \cdots, s_N^0\}$
- ターゲット点群　$P_t = \{t_1, t_2, \cdots, t_N\}$

出力

ソース点群を移動した点群データ $P_s$

目的

$P_s$ と $P_t$ がなるべく重なる

「なるべく重なる」の意味は次で説明

# なるべく重なるとは

点群$P_s = \{s_1, s_2, \cdots, s_N\}$

点群 $P_t = \{t_1, t_2, \cdots, t_N\}$

$P_s$と $P_t$の対応 $C : s_i$と $t_{C(i)}$が対応

対応$C$における $P_s$と $P_t$の距離の二乗の総和 $E(Ps, Pt, C) = \sum_{i=1}^{N} |t_{C(i)} - s_i|^2$

$E(Ps, Pt, C)$が小さいほど $P_s$ と $P_t$は重なっていると考える

# ICPアルゴリズム

$E = \infty;$
$P_s = P_s^0;$
do{
  $E_p = E;$
  $C = P_s$ と $P_t$ の対応
  $P_s = P_s$ の移動先
  $E = E(Ps, Pt, C)$
}while$(E - E_p > \gamma);$

入力

2つの点群データ

- ソース点群　$P_s^0 = \{s_1^0, s_2^0, \cdots, s_N^0\}$
- ターゲット点群　$P_t = \{t_1, t_2, \cdots, t_N\}$

出力

ソース点群を移動した点群データ $P_s$

目的

$P_s$ と $P_t$ がなるべく重なる

# ICPアルゴリズムにおける$E$を小さくする$P_s$の移動方法

対応の決め方: $P_s$の各点 $s_i$ に対して $P_t$ の最近傍点を対応させる

移動の決め方

$R$ : 回転を表す 3 × 3 行列

$T$ : 平行移動を表す 3 × 1 行列

再急降下法などで次の $E$ が最小値の近似値となるように $R$ , $T$ を決める

$$E = \sum_{i=1}^{N} |t_{C(i)} - (Rs_i + T)|^2$$

# 演習の目的:ICPマッチング処理を高速化したい

最近傍点探索が

ICPマッチングの時間計算量における

深刻なボトルネック

2つの点群の対応づけの目的は,

$E$ が小さくなる移動先を決めること

最近傍点であることは必須ではない

探索する近傍点の条件を最近傍から緩和する

探索空間が小さくなり近傍点探索の処理速度が上がる

ICPマッチング処理を高速化できるのでは？

# 演習方法

探索する近傍点の条件を最近傍から緩和する

ICPマッチング処理を高速化できるのでは？

- 高速化されるのか？
- 出力の質は？

本演習では計算機実験で検証

# 2. 本演習における点群処理環境

# 2.1 PCL PointCloudLibray


https://pointclouds.org/

PCL は点群処理を行うオープンソースフレームワーク

マッチング処理，法線推定，点群データのサーフェス化などの

点群に関するアルゴリズムを扱える

本演習ではPCLのICPマッチング用クラスを用いる

- PCLは様々な外部ライブラリを利用
- 最近傍点探索はライブラリFLANNを利用

# 2.2 FLANN Fast Library for Approximate Nearest Neighbors

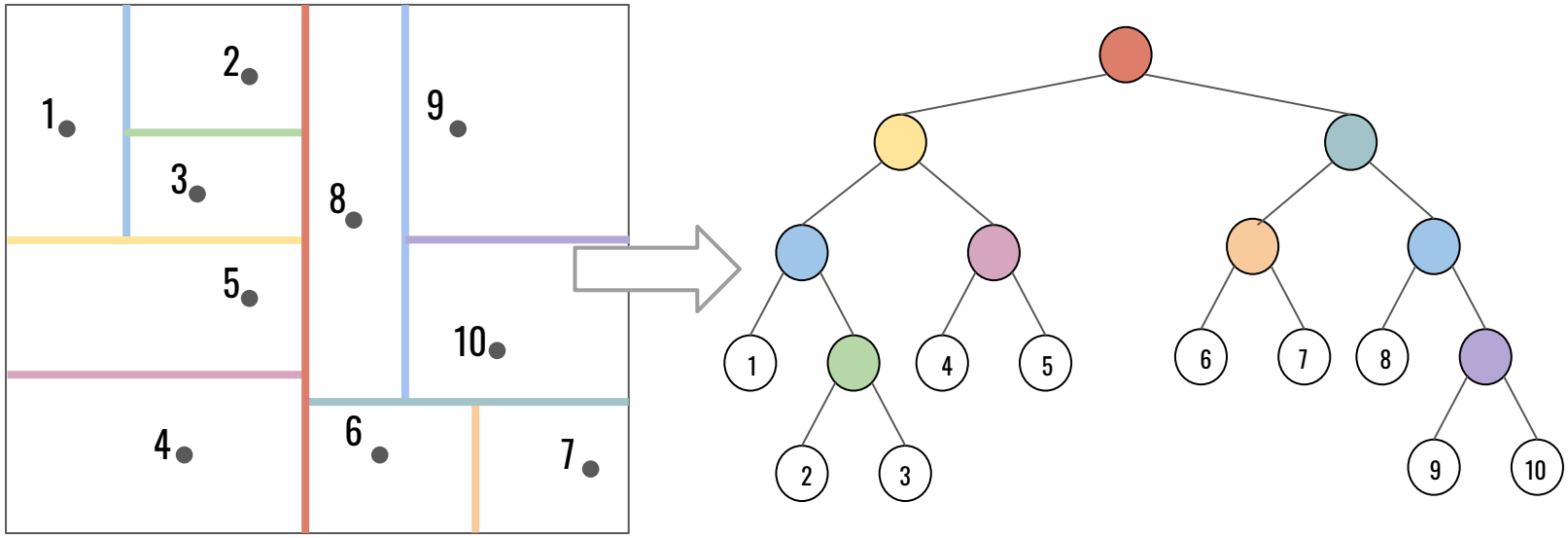
FLANN は C++ で書かれた高速近傍点探索のライブラリ

- FLANN は点群を保持するデータ構造を複数用意
- 本演習ではデータ構造として k-d tree を採用

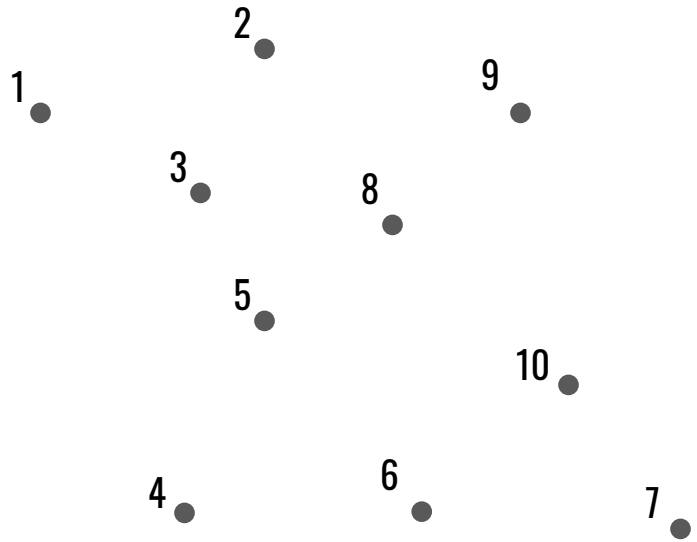
# 2.3 k-d tree

点群データを2分割することを再帰的に繰返し，

バランスの良い2分木を構成→探索の高速化



# k-d treeの構築



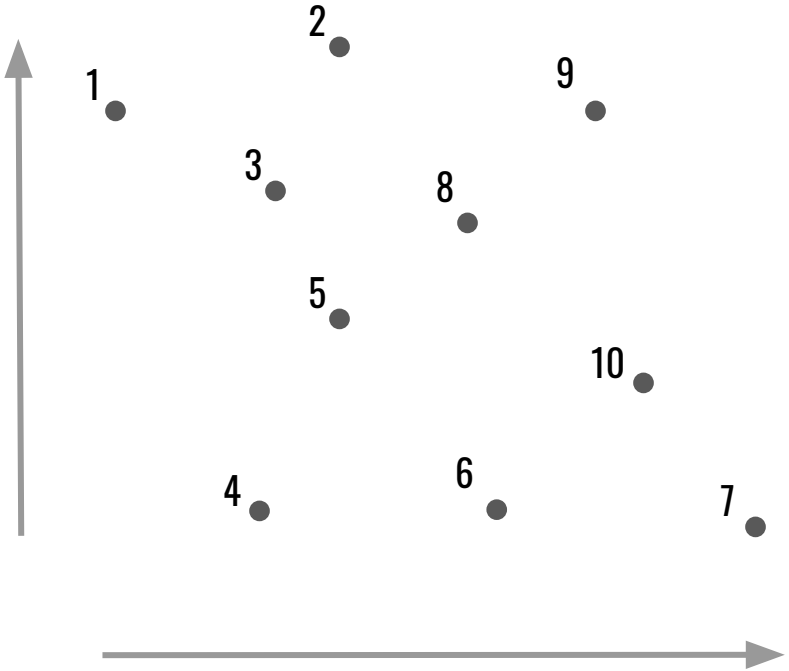

与えられる点群 $P$ はN個の点を持つ

$$P = \{p_1, p_2, \cdots, p_N\}$$

i番目の点 $p_i$ は座標情報と色情報もつが
これ以降では座標情報のみを考える

$$p_i = \{p_{i_x}, p_{i_y}, p_{i_z}\}$$

# k-d treeの構築



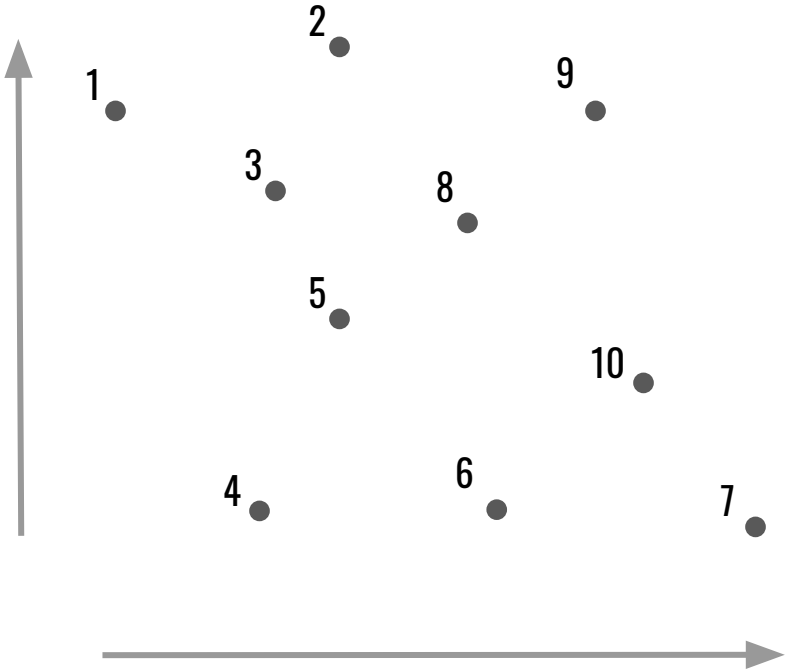

1. 基準となる座標軸を x 軸, y 軸 z 軸から選ぶ

x座標の平均: $\mu_x = \frac{\sum_{i=1}^{N} p_{i_x}}{N}$

y座標の平均: $\mu_y = \frac{\sum_{i=1}^{N} p_{i_y}}{N}$

z座標の平均: $\mu_z = \frac{\sum_{i=1}^{N} p_{i_z}}{N}$

# k-d treeの構築



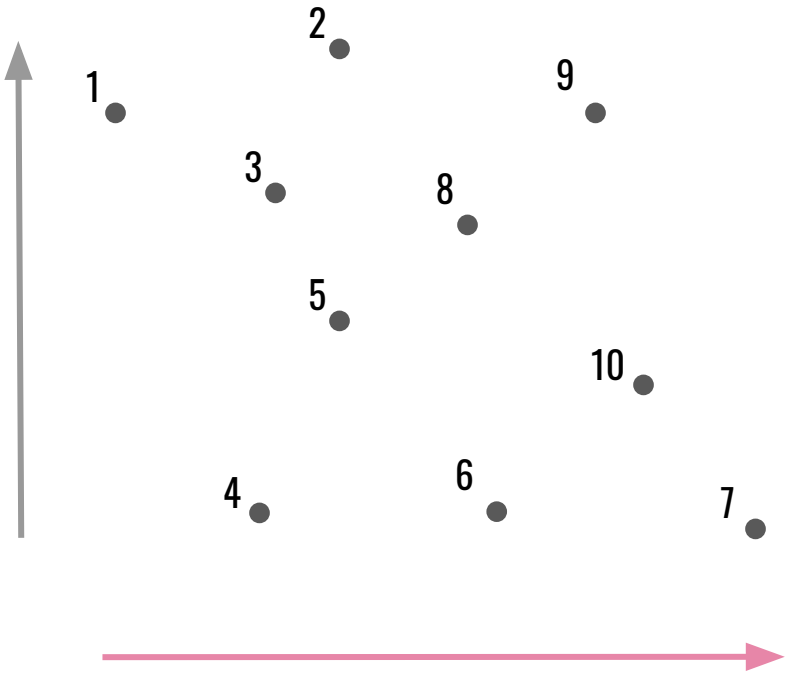

1. 基準となる座標軸を x 軸, y 軸 z 軸から選ぶ

x座標の分散: $\frac{1}{n}\sum_{i=1}^{N}(p_{i_x}-\mu_x)^2$

y座標の分散: $\frac{1}{n}\sum_{i=1}^{N}(p_{i_y}-\mu_y)^2$

z座標の分散: $\frac{1}{n}\sum_{i=1}^{N}(p_{i_z}-\mu_z)^2$

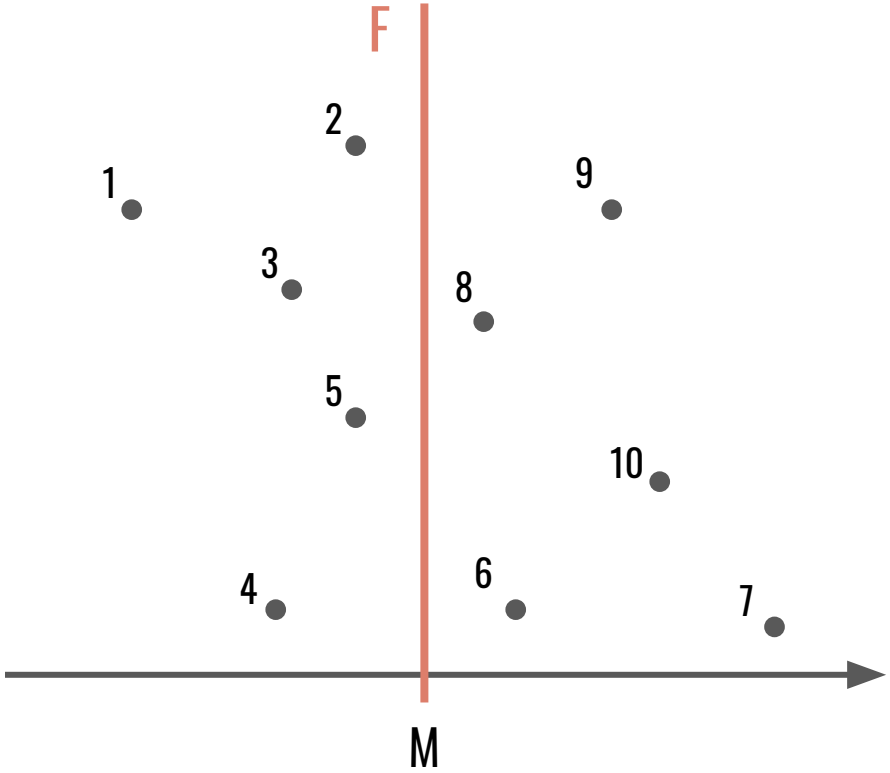
# k-d treeの構築




1. 基準となる座標軸を x 軸, y 軸 z 軸から選ぶ

分散が最大となる軸を選択する

# k-d treeの構築




2. 1.で選択した座標軸に直交する平面 F で空間を分割

1.で選択した座標軸を$a \in \{x, y, z\}$とする

点群 P に含まれる N 個の点 $p_i \ (i \leq N)$ を $a$ 座標の値でソート

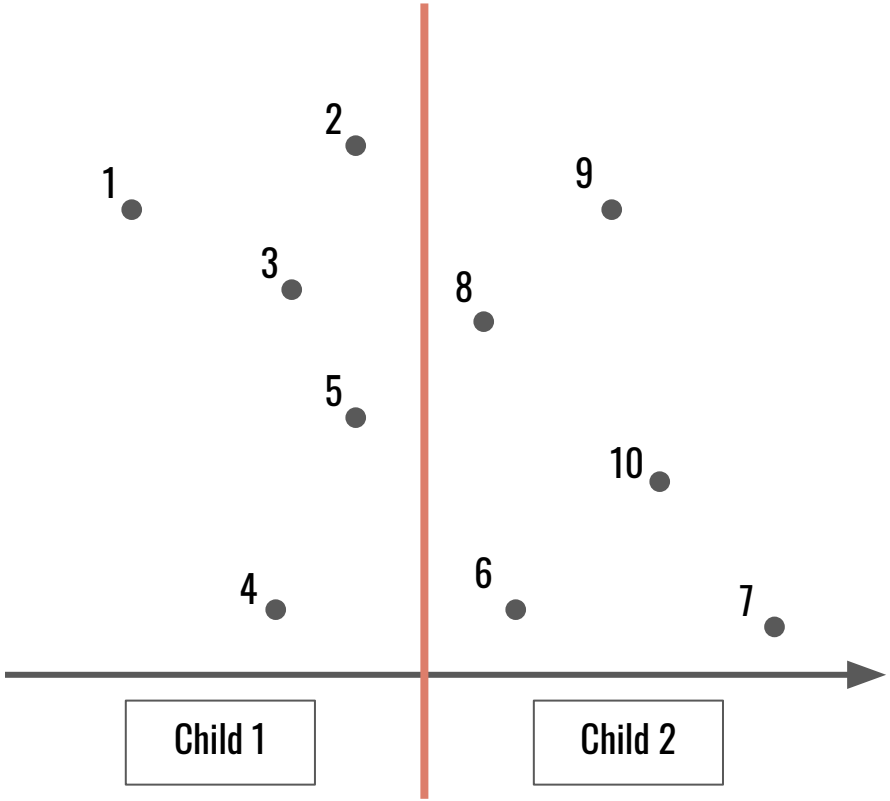
ソート後, $\frac{N}{2}$番目と $\frac{N}{2}+1$ 番目の中間 M を求める

(Nが奇数の場合は $\frac{N-1}{2}$ 番目と $\frac{N+1}{2}$ 番目)

$$M = \frac{p_{\frac{N}{2}} + p_{\frac{N}{2}+1}}{2}$$

M を通り, $a$ 座標軸と直交する平面を F とする.

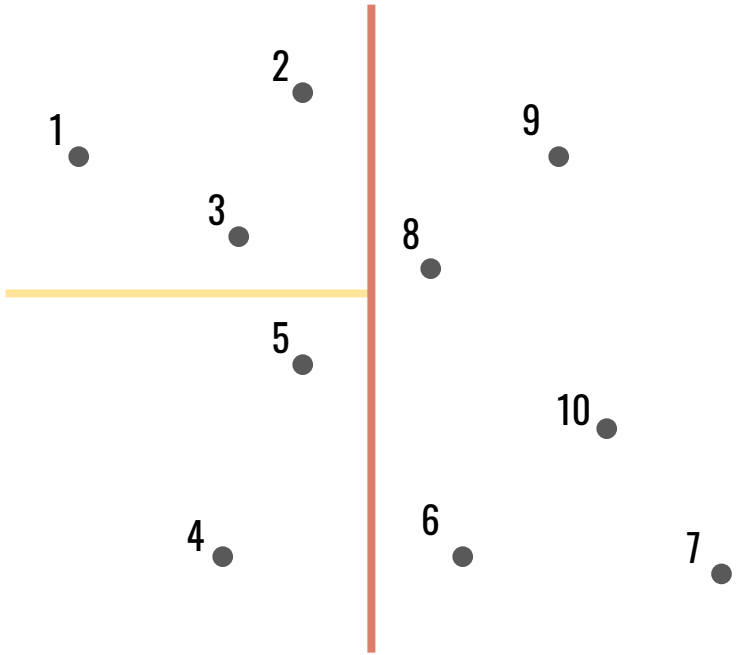
# k-d treeの構築




3. 平面 F をノードとして部分木を構築

1.で選択した座標軸aに対して
平面より負の方向にある点群をChild1,
正の方向にある点群をChild2とする.
平面 F をノードとして，Child1, Child2を
子ノードに設定

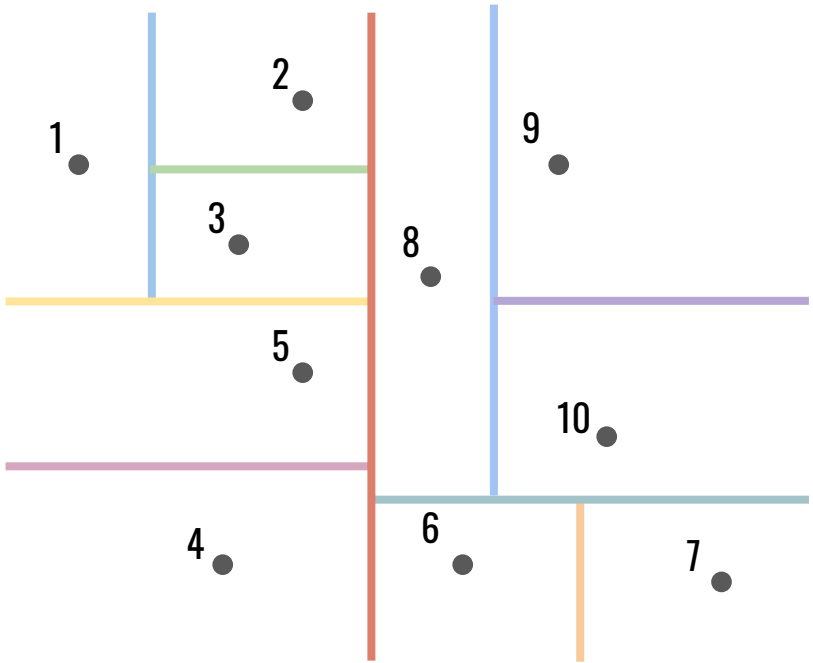
# k-d treeの構築




4.

子ノード Child1, Child2 それぞれに対して，空でなければ再帰的に k-d tree の構築

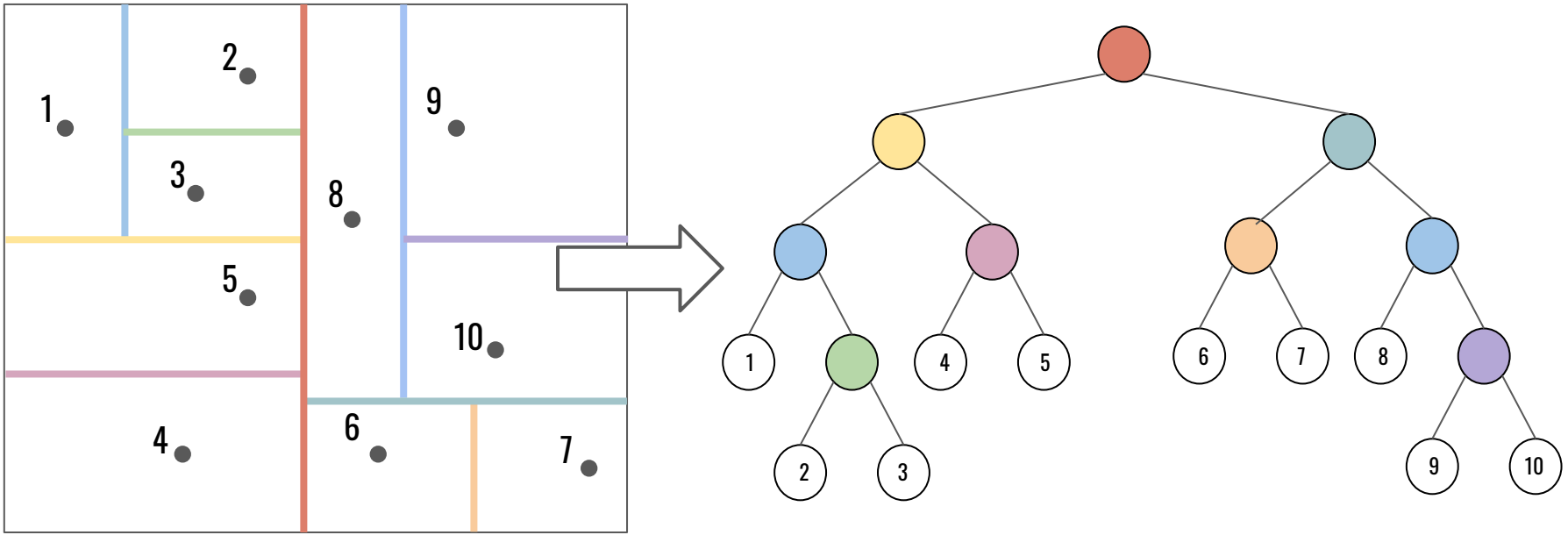
# k-d treeの構築




4.

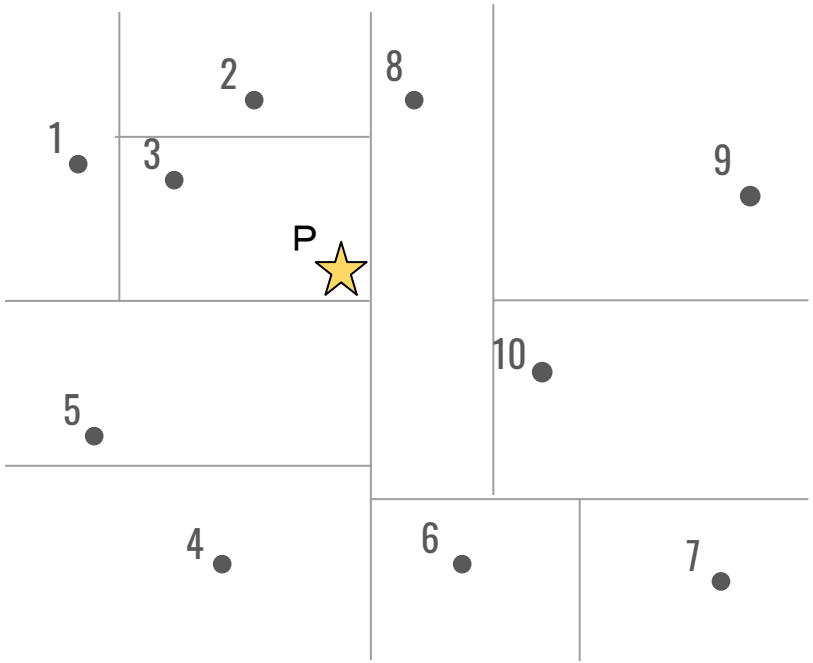
子ノード Child1, Child2 それぞれに対して，空でなければ再帰的に k-d tree の構築

# k-d treeの構築



# 2.4 FLANN における k-d tree 上の近傍点探索

実際には最近傍とは限らない




入力

- 点群 P を表す k-d tree
- 対象点 t

出力

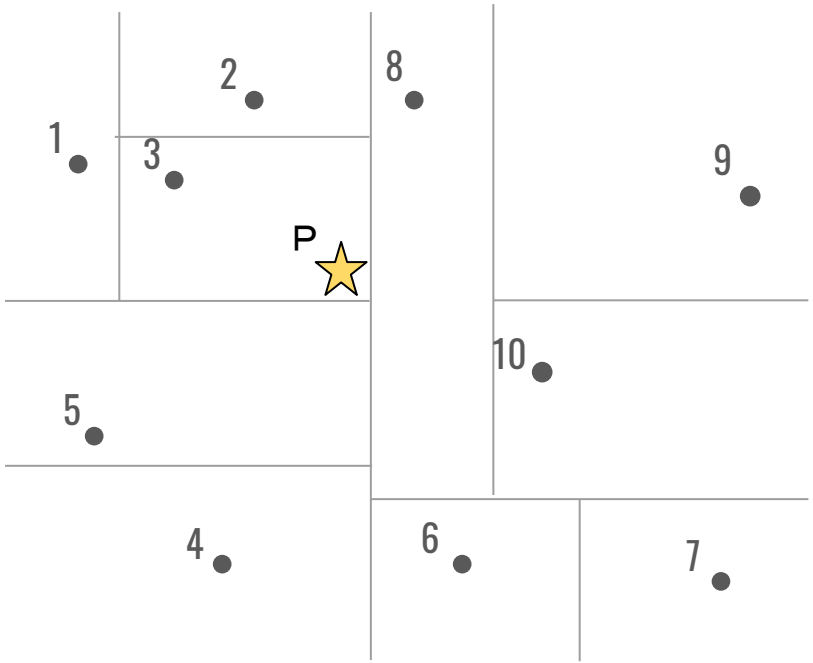
P 中の t に対する近傍点 s

制約

候補となった点の中での最近傍点

# 2.4 FLANN における k-d tree 上の近傍点探索

実際には最近傍とは限らない




入力

- 点群 P を表す k-d tree
- 対象点 t

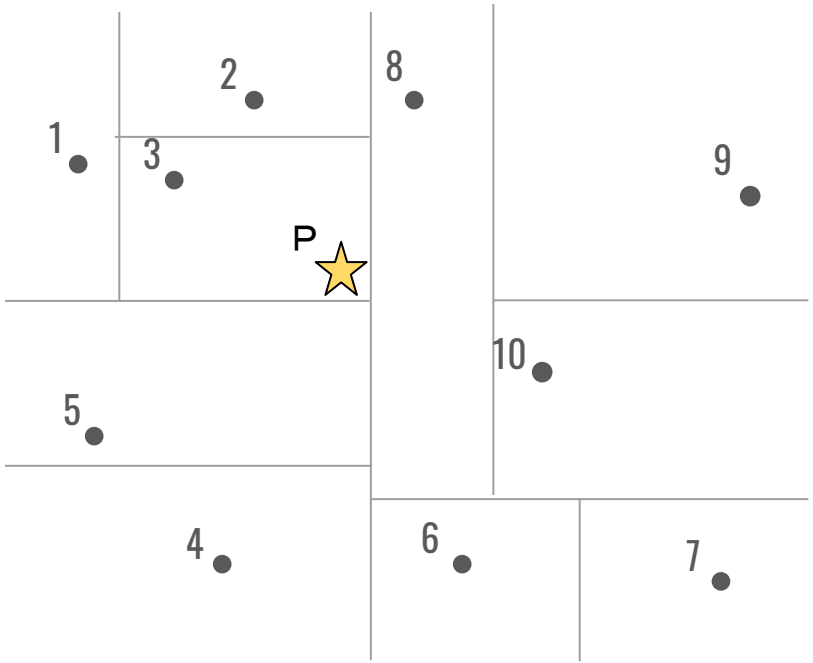
出力

P 中の t に対する近傍点 s

制約

候補となった点の中での最近傍点

# FLANN における k-d tree 上の近傍点候補探索




入力

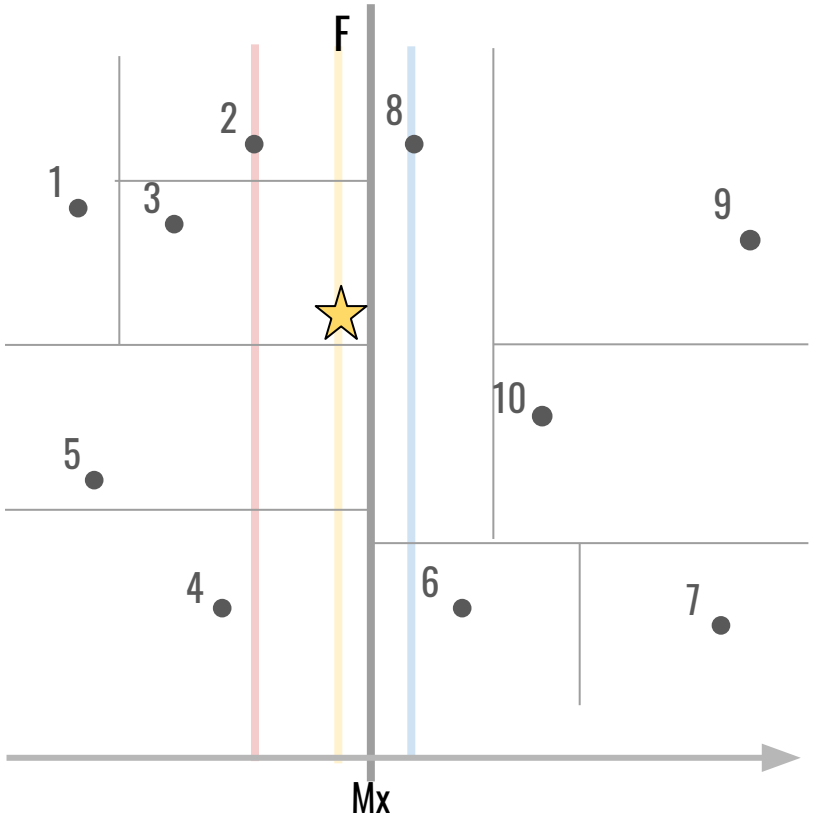
- 点群 P を表す k-d tree
- 対象点 t

出力

P 中の t に対する近傍点候補

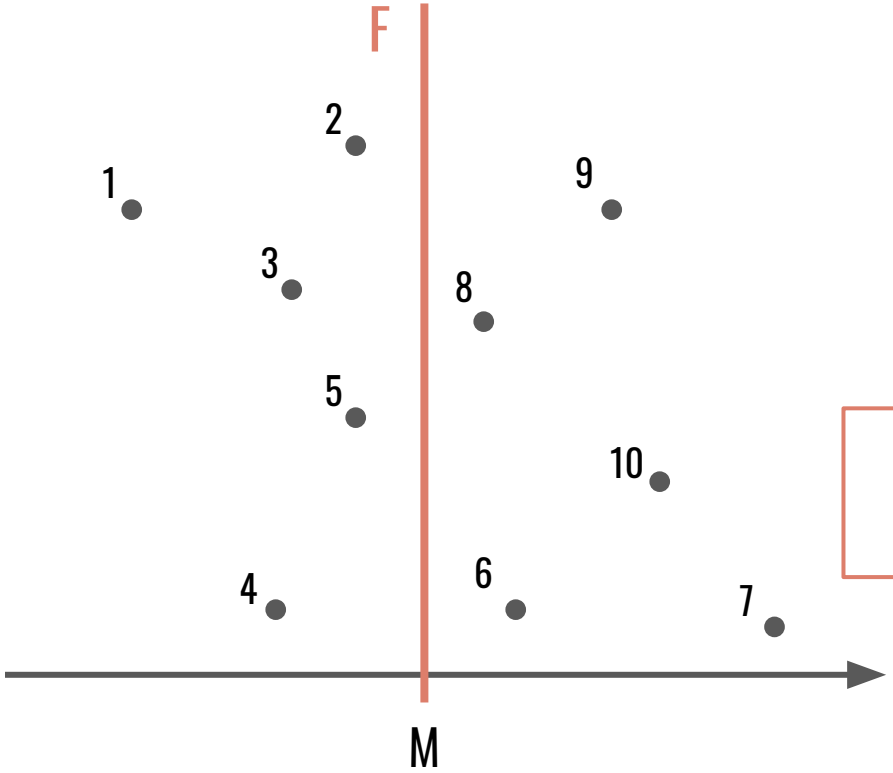
近傍点候補探索の説明では
簡単のため，2次元で説明

# FLANN における k-d tree 上の近傍点候補探索




1. 二分木の根にあたる平面Fを決定するときに求めた2点をP1, P2とする.

# k-d treeの構築




2. 1.で選択した座標軸に直交する平面 F で空間を分割

1.で選択した座標軸を$a \in \{x, y, z\}$とする

点群 P に含まれる N 個の点 $p_i \ (i \leq N)$ を $a$ 座標の値でソート

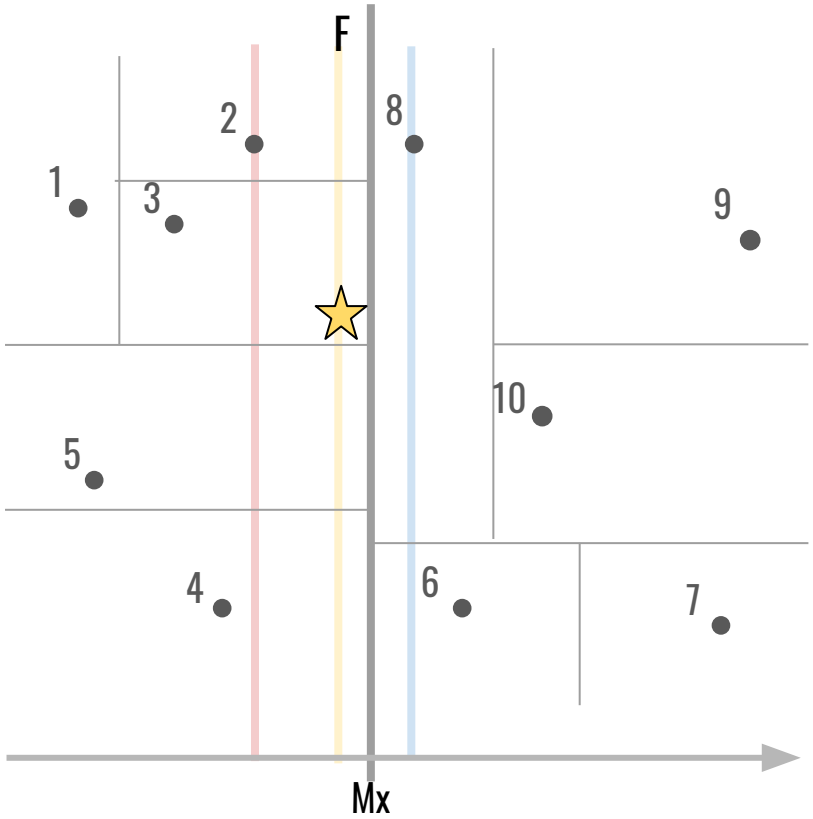
ソート後, $\frac{N}{2}$番目と $\frac{N}{2}+1$ 番目の中間 M を求める

(Nが奇数の場合は $\frac{N-1}{2}$ 番目と $\frac{N+1}{2}$ 番目)

$$M = \frac{p_{\frac{N}{2}} + p_{\frac{N}{2}+1}}{2}$$

M を通り, $a$ 座標軸と直交する平面を F とする.

# FLANN における k-d tree 上の近傍点候補探索

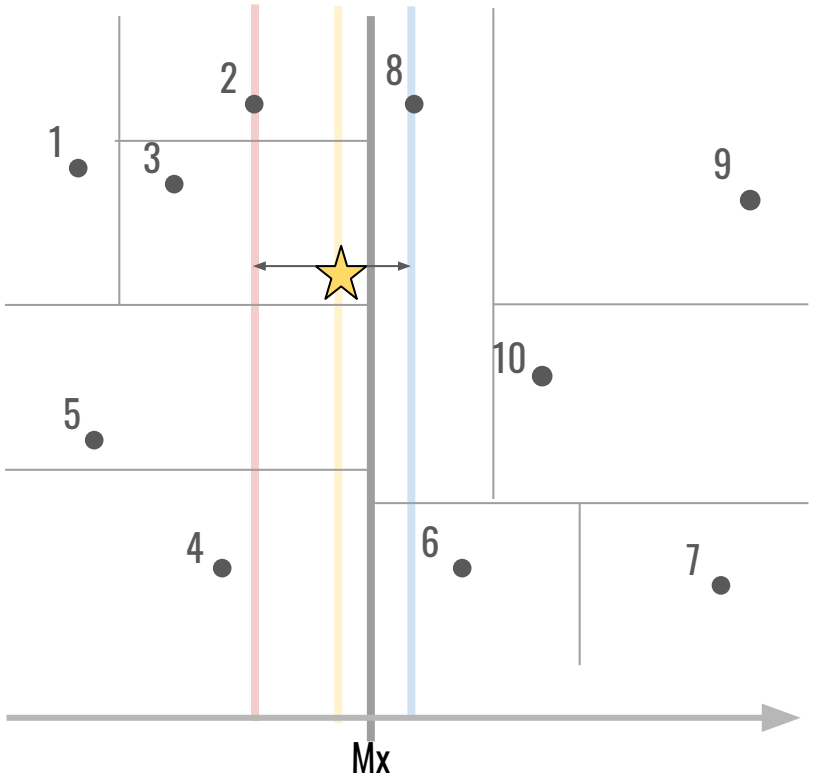



1. 二分木の根にあたる平面Fを決定するときに求めた2点をP1, P2とする.

$$P1 := p_{\frac{N}{2}}$$

$$P2 := p_{\frac{N}{2}+1}$$

# FLANNによるKDTreeでの最近傍点探索




## 2. P1, P2と対象点の距離を調べる

Fがx軸と直交するならば

$$\mathrm{diff1} = p_x - P1_x$$

$$\mathrm{diff2} = p_x - P2_x$$

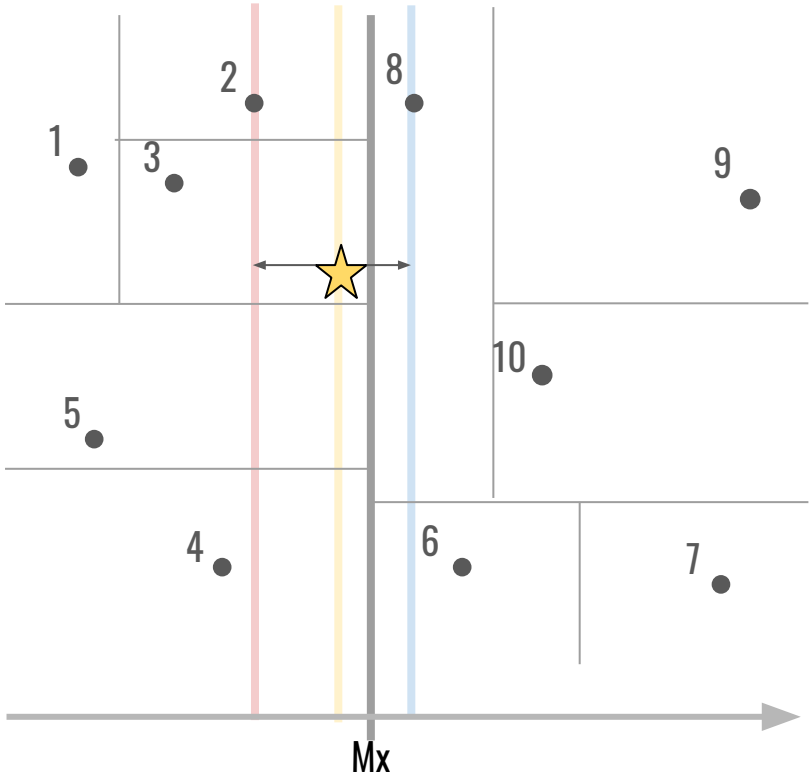
Fがy軸と直交するならば

$$\mathrm{diff1} = p_y - P1_y$$

$$\mathrm{diff2} = p_y - P2_y$$

とする.

# FLANN における k-d tree 上の近傍点候補探索



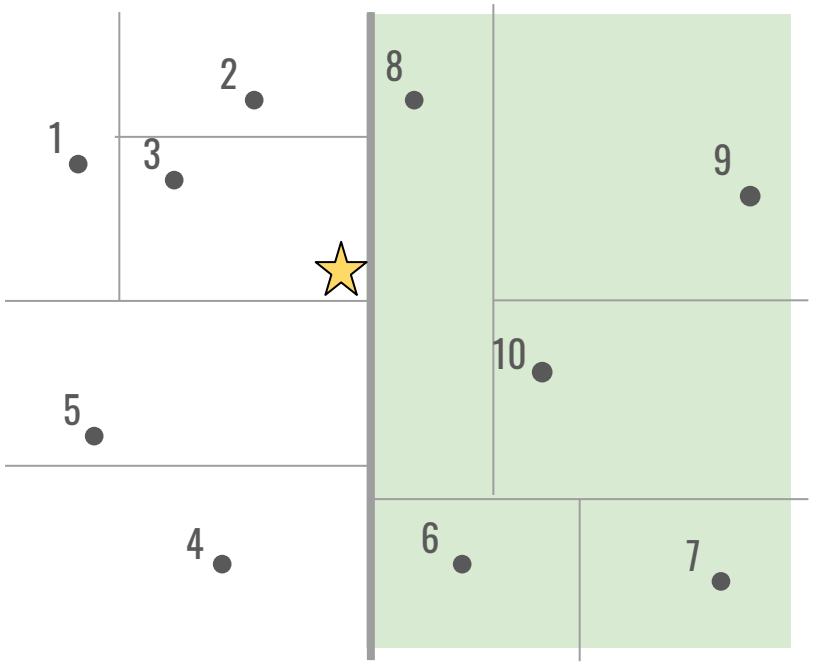

3. 次に探索するノードを決定する

$\mathrm{diff1} + \mathrm{diff2} > 0$ ならば,
Child2を根とする部分木を

$\mathrm{diff1} + \mathrm{diff2} \leq 0$ ならば,
Child1を根とする部分木を

次に探索

# FLANN における k-d tree 上の近傍点候補探索



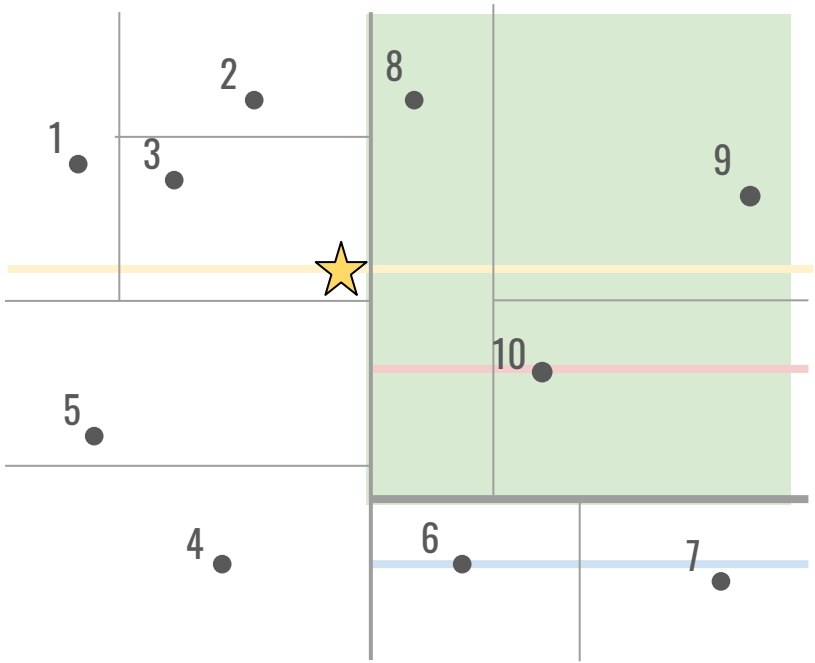

3. 次に探索するノードを決定する

$\mathrm{diff1} + \mathrm{diff2} > 0$ ならば,
Child2を根とする部分木を

$\mathrm{diff1} + \mathrm{diff2} \leq 0$ ならば,
Child1を根とする部分木を

次に探索

# FLANN における k-d tree 上の近傍点候補探索




4.

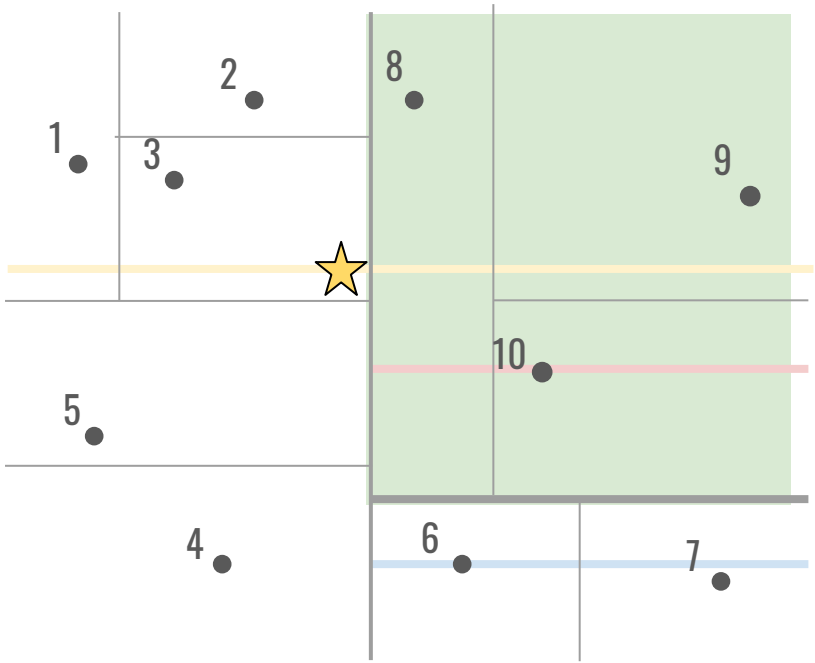
次に探索するノードを根と

する部分木に対して，その部分木が

空でなければ，再帰的に近傍点候

補探索を行う

# FLANN における k-d tree 上の近傍点候補探索




4.

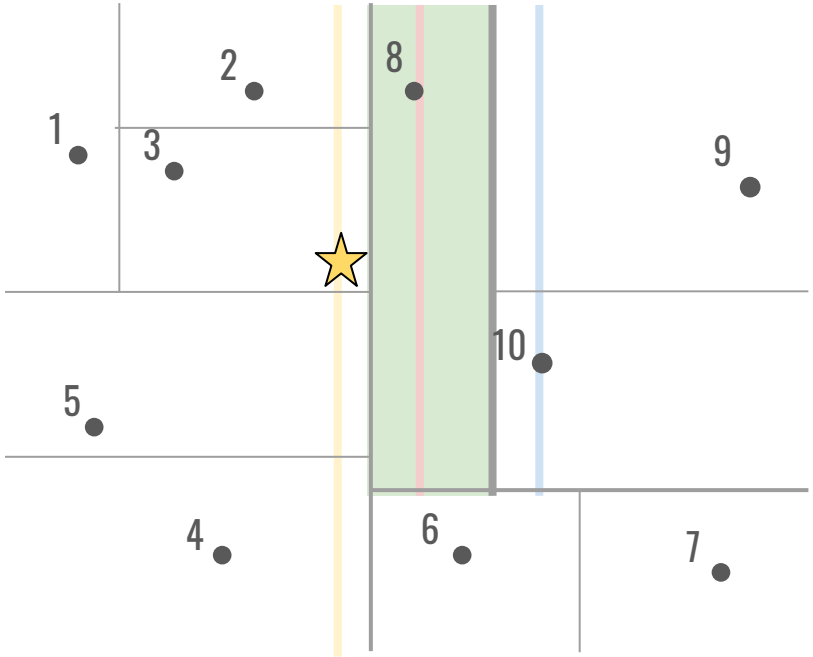
次に探索するノードを根と

する部分木に対して，その部分木が

空でなければ，再帰的に近傍点候

補探索を行う

# FLANN における k-d tree 上の近傍点候補探索




4.

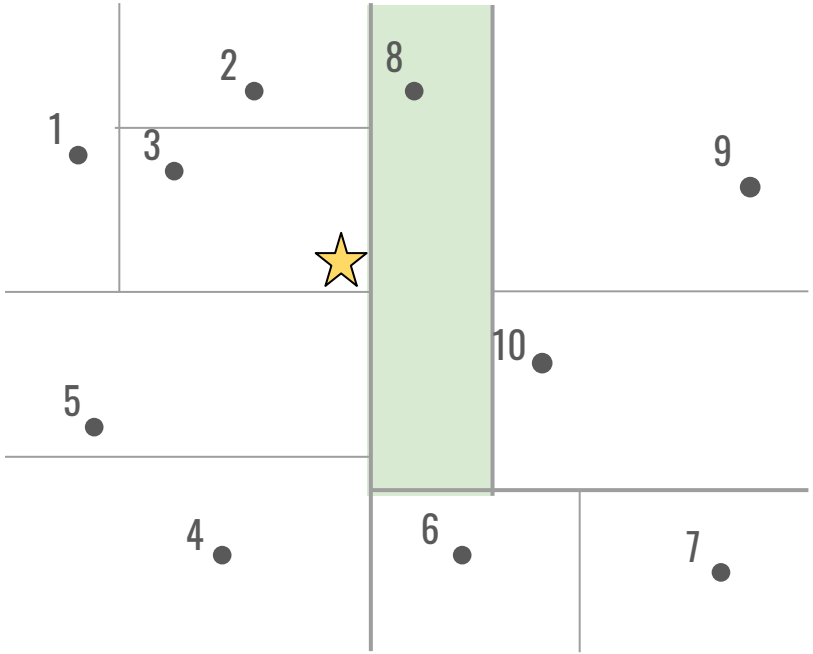
次に探索するノードを根と

する部分木に対して，その部分木が

空でなければ，再帰的に近傍点候

補探索を行う

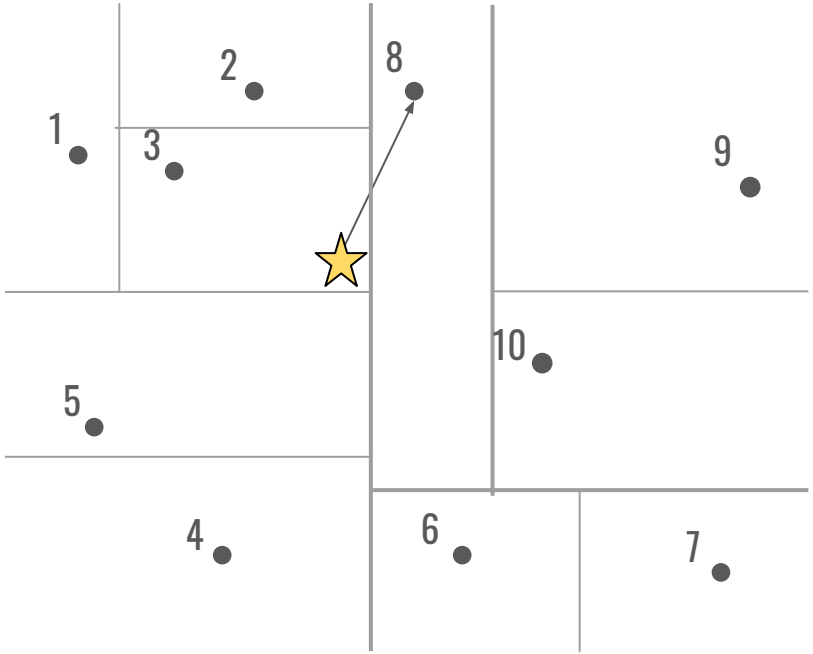
# FLANN における k-d tree 上の近傍点候補探索




4.

次に探索するノードを根とする部分木に対して，その部分木が空でなければ，再帰的に近傍点候補探索を行う

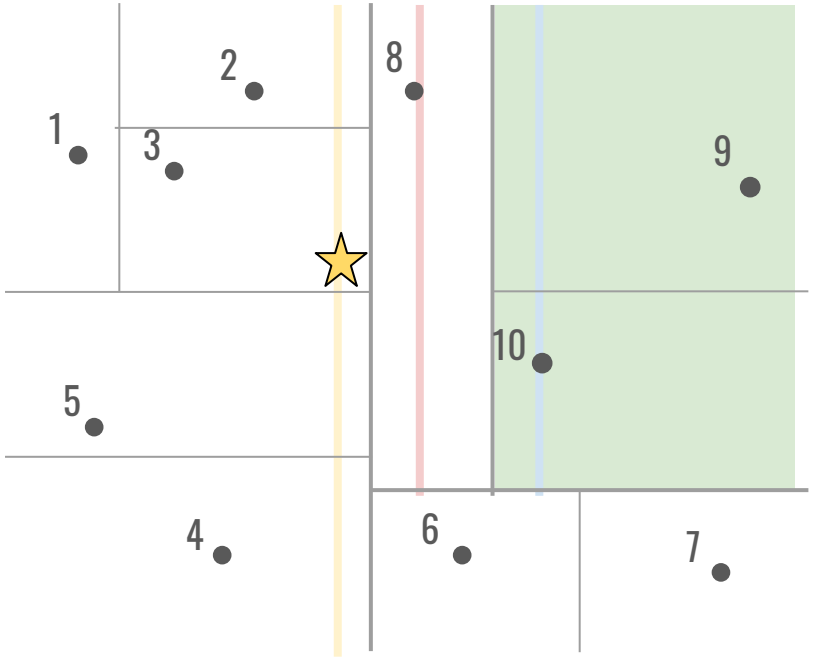
# FLANN における k-d tree 上の近傍点候補探索




5.

探索していない子ノードを根とする部分木中にその時点の候補より対象点に近い点がありそうなら，探索していない子ノードを根とする部分木に対して再帰的に近傍点候補探索

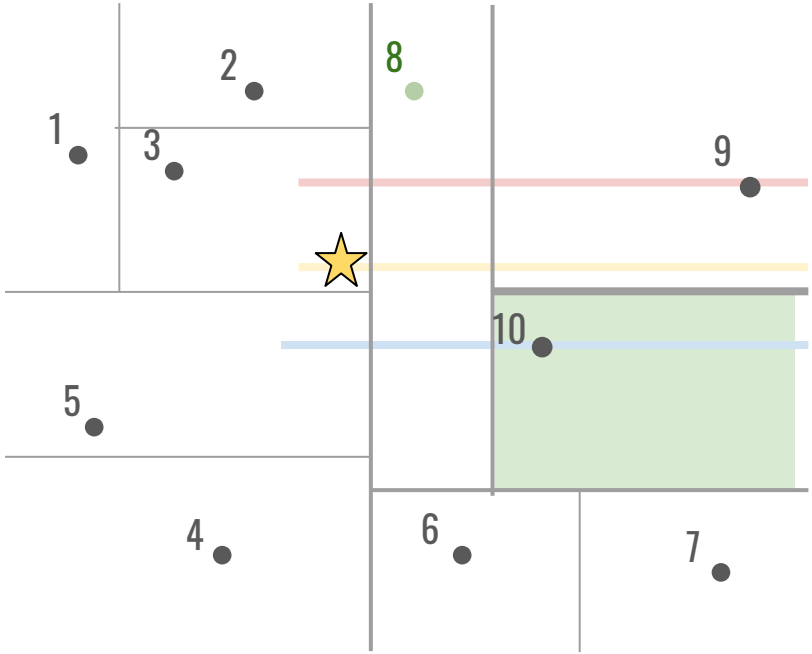
# FLANN における k-d tree 上の近傍点候補探索




5.

探索していない子ノードを根とする部分木中にその時点の候補より対象点に近い点がありそうなら，探索していない子ノードを根とする部分木に対して再帰的に近傍点候補探索

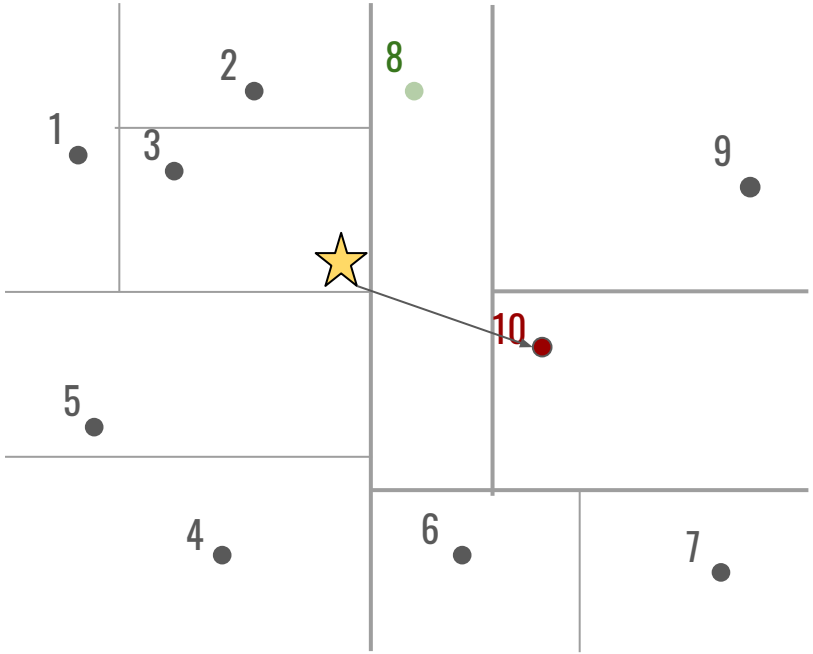
# FLANN における k-d tree 上の近傍点候補探索




5.

探索していない子ノードを根とする部分木中にその時点の候補より対象点に近い点がありそうなら，探索していない子ノードを根とする部分木に対して再帰的に近傍点候補探索

# FLANN における k-d tree 上の近傍点候補探索




5.

探索していない子ノードを根とする部分木中にその時点の候補より対象点に近い点がありそうなら，探索していない子ノードを根とする部分木に対して再帰的に近傍点候補探索

# FLANN における k-d tree 上の近傍点候補探索（実装 A）

```
void search( result, node, target ){
    if ( (node-> child1 == NULL) && (node->childe2 == NULL) {
        result.add(node);
        return;
    }
    diff1 = dist ( target, node -> p1 );
    diff2 = dist ( target, node -> p2 );
    if ( diff1 + diff2 < 0 ){
        bestChild = node -> child1;
        otherChild = node -> child2;
    }else{
        bestChild = node -> child2;
        otherChild = node -> child1;
    }
    search (result, bestChild, target);
    if ( result.worstDist() > euc_dist (target, otherChild) ){
        search(result, otherChild, target);
    }
}
```

nodeが葉ならば探索終了

ノードとの距離を測る

どちらの空間を探索するか決定する.

対象点に近い空間を探索する

探索してない部分木中に候補よりも対象点近い点が存在しそうであれば探索する

# FLANN における k-d tree 上の近傍点候補探索

```
void search( result, node, target ){
    if ( (node-> child1 == NULL) && (node->childe2 == NULL) {
        result.add(node);
        return;
    }
    diff1 = dist ( target, node -> p1 );
    diff2 = dist ( target, node -> p2 );
    if ( diff1 + diff2 < 0 ){
        bestChild = node -> child1;
        otherChild = node -> child2;
    }else{
        bestChild = node -> child2;
        otherChild = node -> child1;
    }
    search (result, bestChild, target);
    if ( result.worstDist() > err){
        search(result, otherChild, target);
    }
}
```

nodeが葉ならば探索終了

ノードとの距離を測る

どちらの空間を探索するか決定する.

対象点に近い空間を探索する

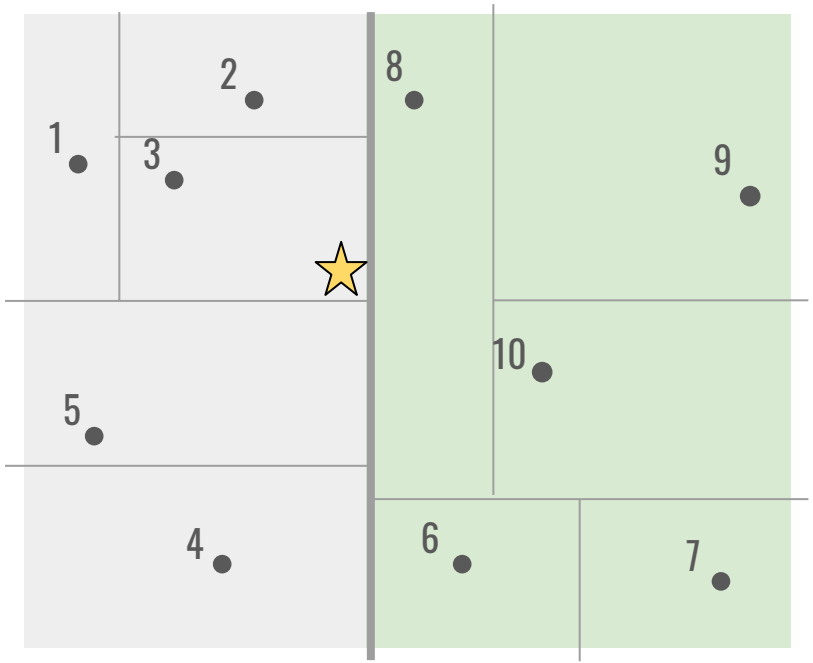
探索してない部分木中に候補よりも対象点近い点が存在しそうであれば探索する

# 2.4 k-d tree上での近傍点探索の条件緩和(実装 B )

```
void search( result, node, target ){
    if ( (node-> child1 == NULL) && (node->childe2 == NULL) {
        result.add(node);
        return;
    }
    diff1 = dist ( target, node -> p1 );
    diff2 = dist ( target, node -> p2 );
    if ( diff1 + diff2 < 0 ){
        bestChild = node -> child1;
        otherChild = node -> child2;
    }else{
        bestChild = node -> child2;
        otherChild = node -> child1;
    }
    search (result, bestChild, target);
    if ( result.worstDist() > err){
        search(result, otherChild, target);
    }
}
```

この探索を行わない

# 2.4 k-d tree上での近傍点探索の条件緩和(実装 B )




$\mathrm{diff1} + \mathrm{diff2} > 0$ ならば, Child2

$\mathrm{diff1} + \mathrm{diff2} \leq 0$ ならば, Child1

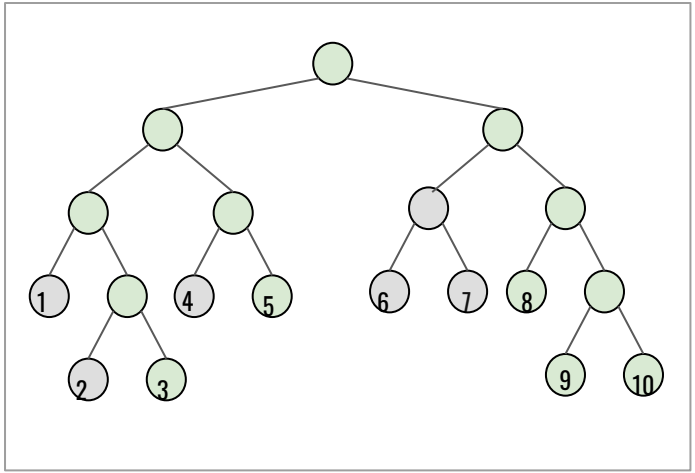
のみを探索する.

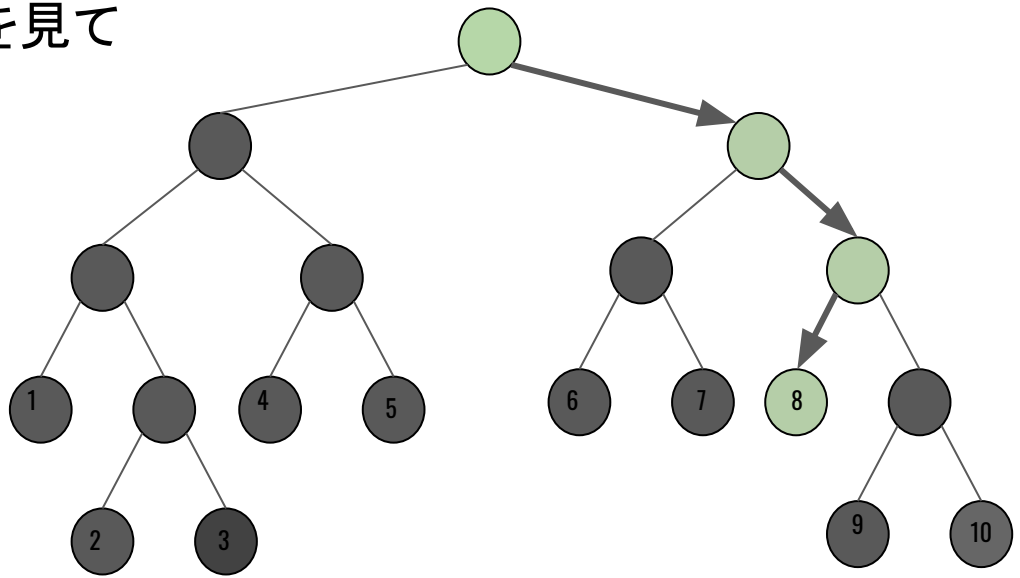
もう一方のノードは探索しない.

# 2.4 k-d tree上での近傍点探索の条件緩和(実装 B )

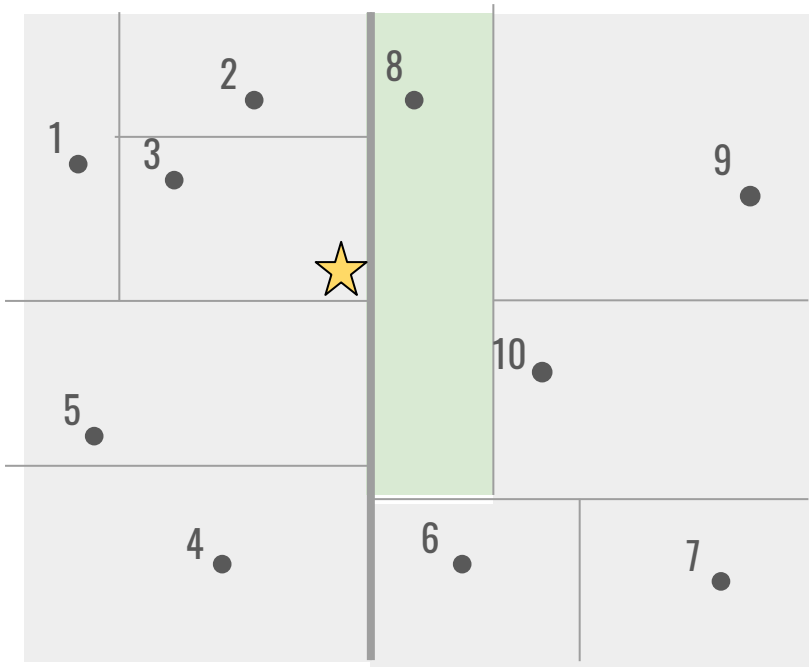
1つノードに対して, 1つの子だけを見て
近傍点を決定していくことになる.

FLANNでの最近傍探索



# 2.4 k-d tree上での近傍点探索の条件緩和(実装 B )




近傍点候補の集合が小さくなる

探索範囲が減るので処理速度は向上

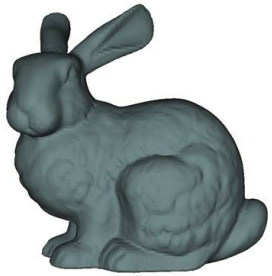
# 3. 演習

# 3.1 演習方法

- PCL ( PointCloudLibrary ) の ICP マッチング用クラスを使用
- ソースデータ：元データ（ターゲットデータ）からxy平面に45°回転，
  z軸に0.4m平行移動したもの
- ICP マッチングの収束条件：移動前後の二乗誤差総和の差 0.1 / 0.05 / 0.01 以下
- 実装 A と実装 B で計算機実験
  - ICPマッチングの処理時間を計測
  - ターゲット点群と処理後の点群の対応点間の距離の 2乗の平均を計測

# データセットについて

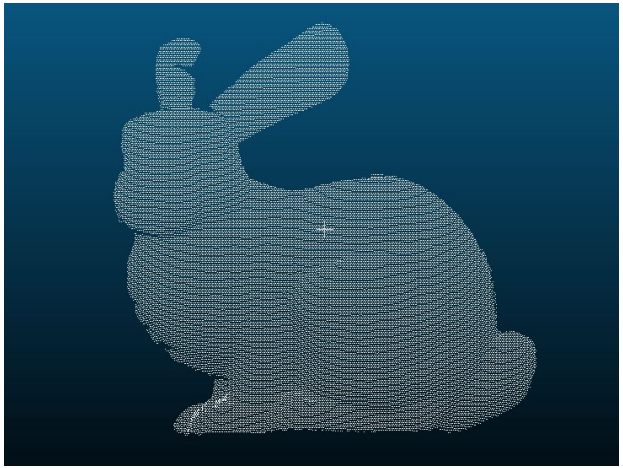
The stanford 3D Scanning Repository



1994年にスタンフォード大学の
グレッグ・タ－ク（英語: Greg Turk），
マーク・リーボイ（英語: Marc Levoy）
によって開発された、コンピューター・グラフィックス用の試験用データセット

# 3.3 演習結果

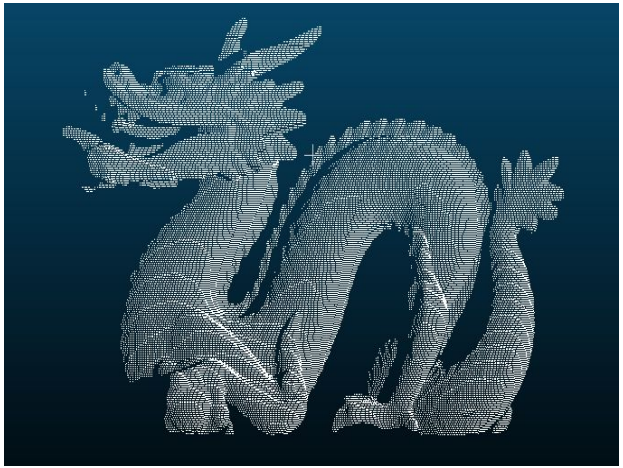
**Stanford Bunny**
(40256 points)



| | 0.1 | 0.05 | 0.01 | 誤差 |
|---|---|---|---|---|
| 実装A | 26466.2 | 28508.6 | 28257.0 | 2.24390e-7 |
| 実装B | 16153.4 | 16392.6 | 19079.5 | 3.56022e-4 |
| | | | ( msec ) | ( m ) |

# 3.2 演習結果

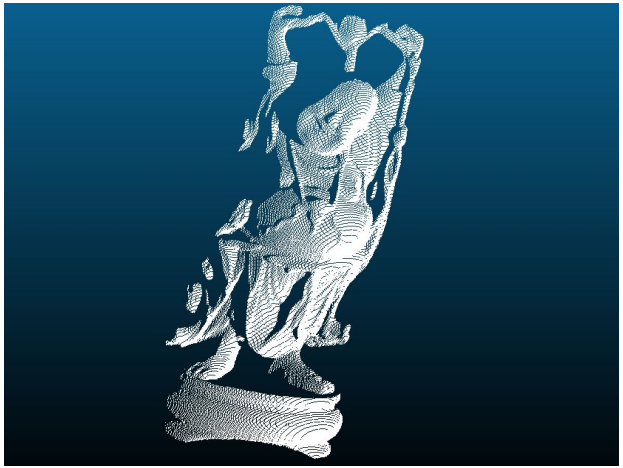
**Dragon**
(41841 points)



| | 0.1 | 0.05 | 0.01 | 誤差 |
|---|---|---|---|---|
| 実装A | 35640.4 | 36149.0 | 42402.3 | 2.55489e-7 |
| 実装B | 21322.1 | 22366.2 | 29030.4 | 2.93749e-7 |
| | | | ( msec ) | ( m ) |

# 3.2 演習結果

**Happy Buddha**
(78056 points)



| | 0.1 | 0.05 | 0.01 | 誤差 |
|---|---|---|---|---|
| 実装A | 96749.3 | 106420.0 | 181725.0 | 1.53319e-4 |
| 実装B | 8502.55 | 49265.3 | 48547.1 | 2.94517e-7 |
| | | | ( msec ) | ( m ) |

# 4. おわりに

# 4.1 演習結果のまとめ

- どのデータに対しても ( 実装 B の処理速度 ) < ( 実装 A の処理速度 )
  - “Happy Butta” では，点群データのポイント数が多いほど実行時間比が大きくなった
- 誤差は優劣はデータに依存
  - “Dragon” , “Happy Butta” : ( 実装 A の誤差 ) < ( 実装 B の誤差 )
  - “Stanford bunny” : ( 実装 A の誤差 ) > ( 実装 B の誤差 )

# 4.2 今後の課題

- 処理速度向上の理論的根拠について
- テストデータの特徴と誤差の関係
- 他のデータ構造の利用
- 位置情報と法線情報を用いたマッチング処理GICPでの利用