UAS-Based LiDAR Mapping

Video F-I

PURDUE UNIVERSITY

LiDAR Data Structuring

kd-tree Structure

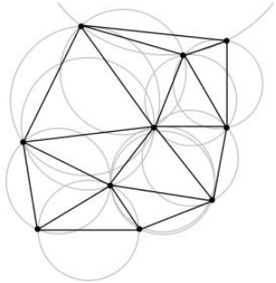
PURDUE UNIVERSITY

Overview

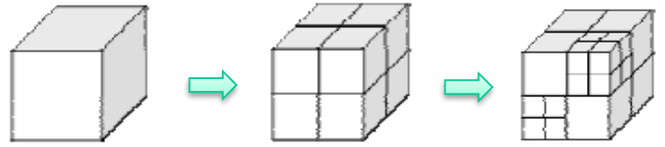
- **LiDAR Data Structuring**
  - Triangular Irregular Network (TIN) data structure
  - Octree data structure
  - Kd-tree data structure

Structuring the LiDAR Point Cloud

- **Objectives:**
  - Efficient sorting and organization of LiDAR point clouds
  - Speed up the process of searching for the nearest neighbor(s) of a point
- **Data structures:**
  - **Triangular Irregular Network (TIN)**: A triangulation of the LiDAR point cloud divides its convex hull into a set of triangles. A circle passing through the vertices of any triangle doesn't contain any other point of the point set (Okabe et al., 1992).
    - × This structure is defined in the XY-plane and does not consider the points' heights.



  - **Octree data structure**: Octrees are used to partition a three-dimensional space by recursively subdividing it into eight subspaces.
    - × It cannot guarantee a fully balanced hierarchical data structure.



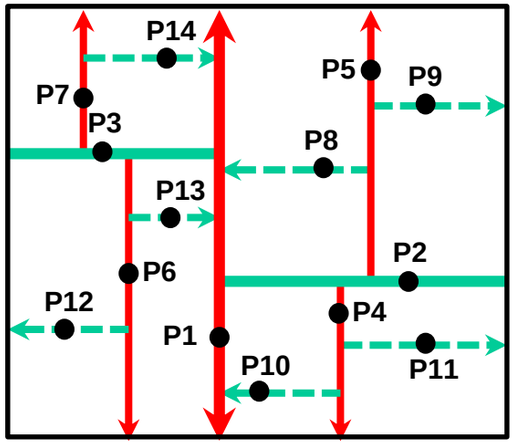
Structuring the LiDAR Point Cloud: kd-tree Structure

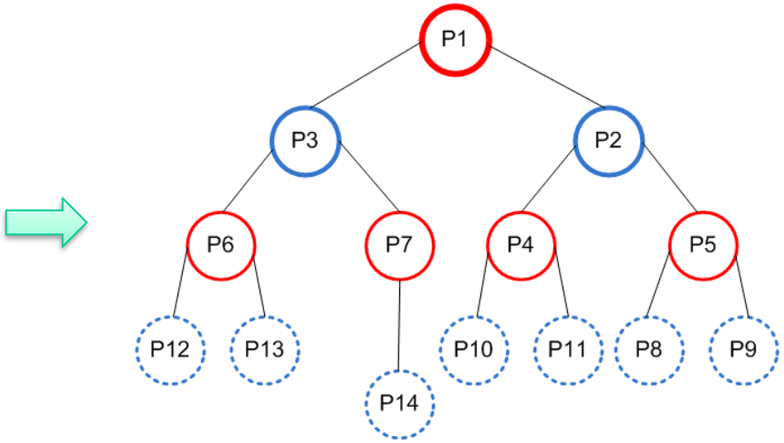
- **kd-tree data structure construction:**
  - Recursive subdivision of the three-dimensional space along the longest extent of the data in the X, Y, or Z direction
  - The splitting plane is perpendicular to the chosen extent direction and passes through the point with the median coordinate along the selected extent (Sadgewick, 1992).
- **Advantages**

✓ **Efficient structuring with minimal number of subdivisions**

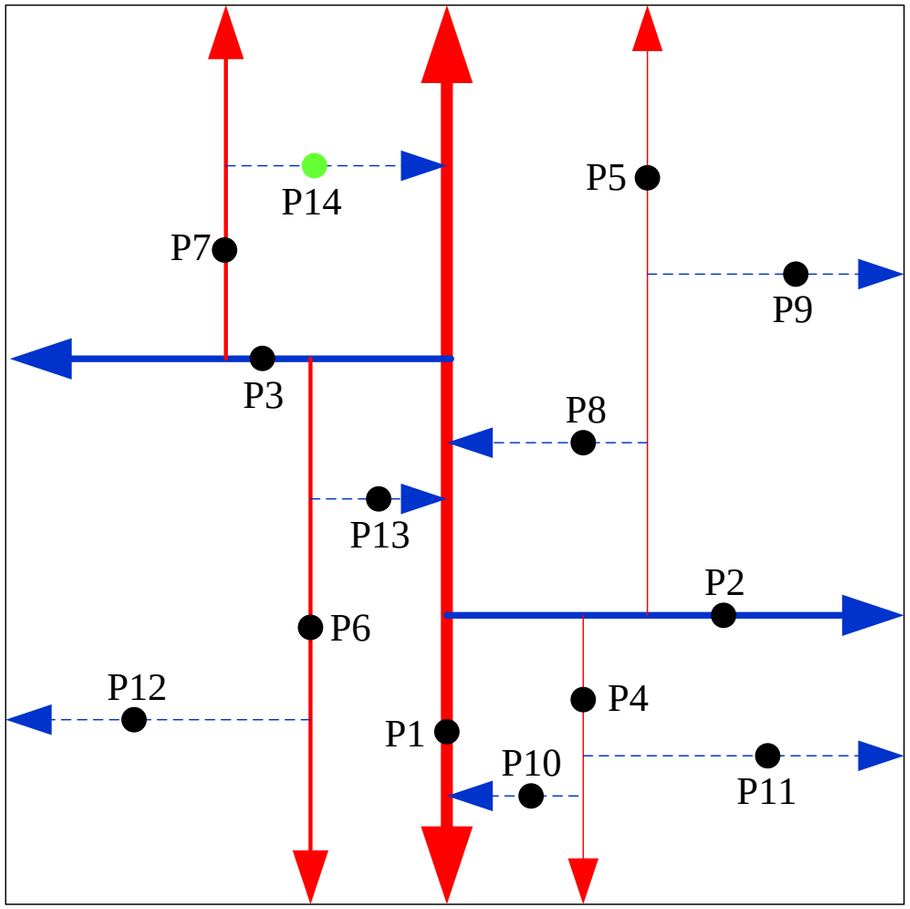
✓ **More efficient nearest neighbor search algorithms**

✓ **Balanced data structure**





Structuring the LiDAR Point Cloud: kd-tree Structure

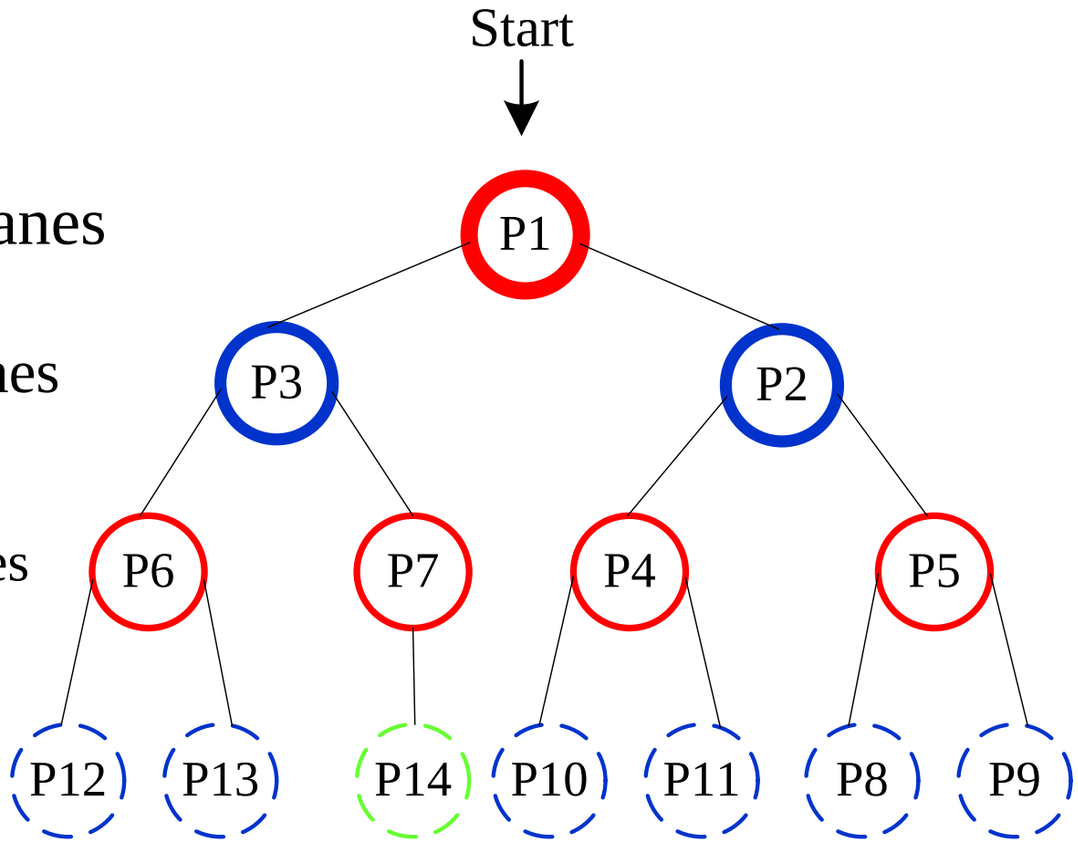


X-splitting planes

Y-splitting planes

X-splitting planes

Y-splitting planes



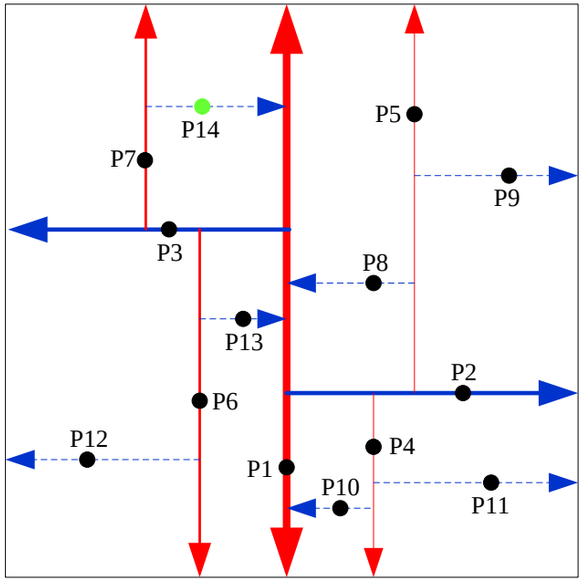
PURDUE UNIVERSITY

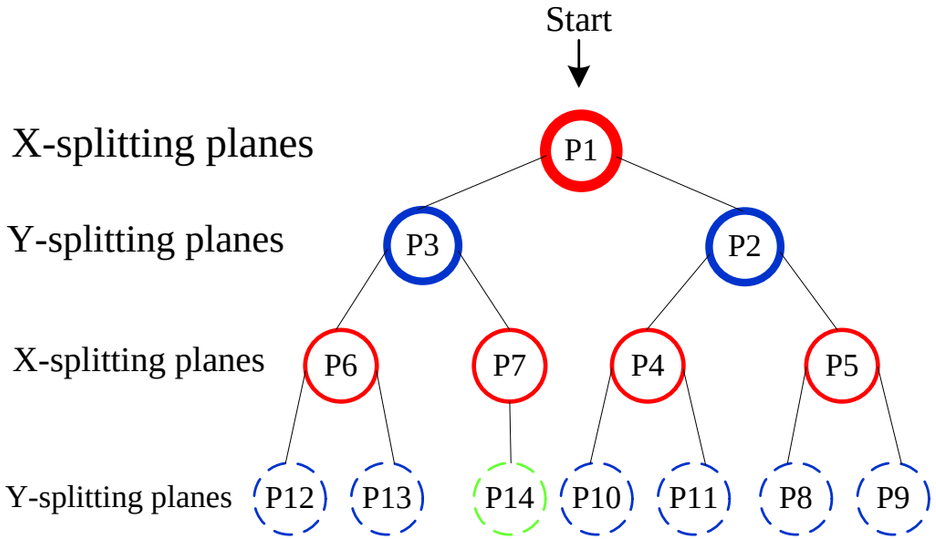
Searching for the Nearest Neighbor of a Point

- **Nearest neighbor of a given point**
  I. Start with the root node, the algorithm moves down the tree recursively in the same way it would if the point in question were being inserted.
  II. Initial distance (infinity) is reduced as closer points are discovered.
  III. Steps I and II are repeated until the algorithm reaches the leaf node.
  IV. Search the other side of the splitting plane for points which may be closer to the point in question by checking the intersection of the splitting hyper plane with a sphere centered at the point in question with a radius equivalent to the distance to the closest discovered point. In case of intersection between them, the other branch of the tree should also be searched for a closest neighbor.
  V. The node with the smallest distance is returned as the nearest neighbor.

Searching for the Nearest Neighbor of a Point

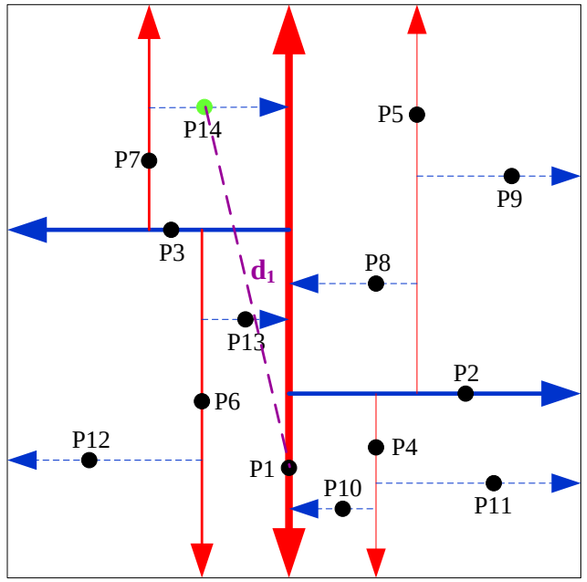
- **Nearest neighbor of a given point (P14): Schematic view**



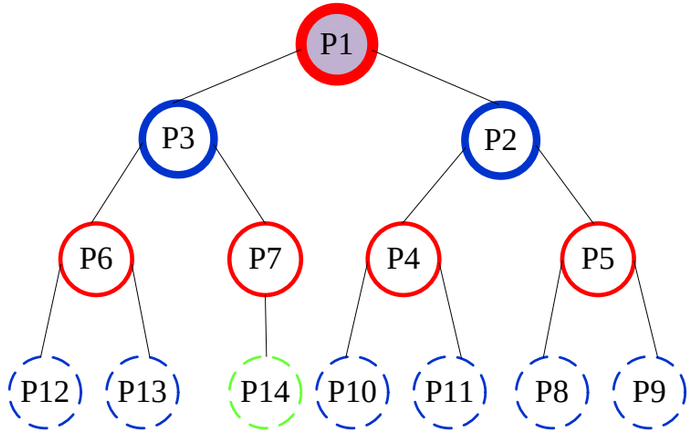


Searching for the Nearest Neighbor of a Point

- **Nearest neighbor of a given point (P14): Schematic view**



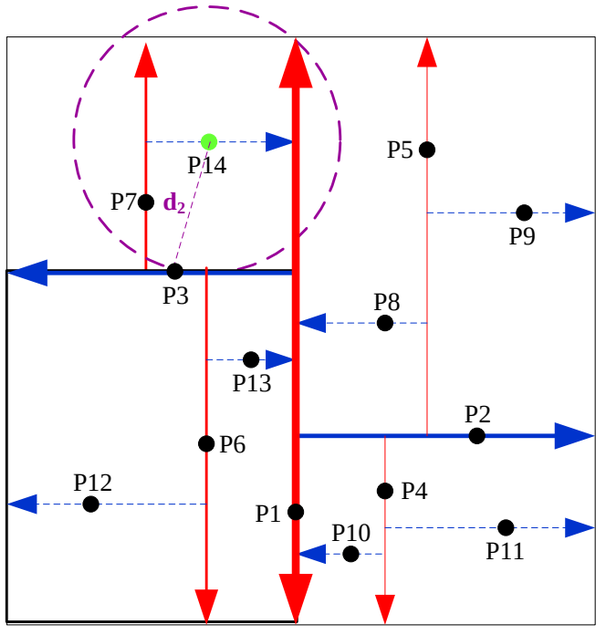
Candidate Point = P1
Minimum Distance = d_1



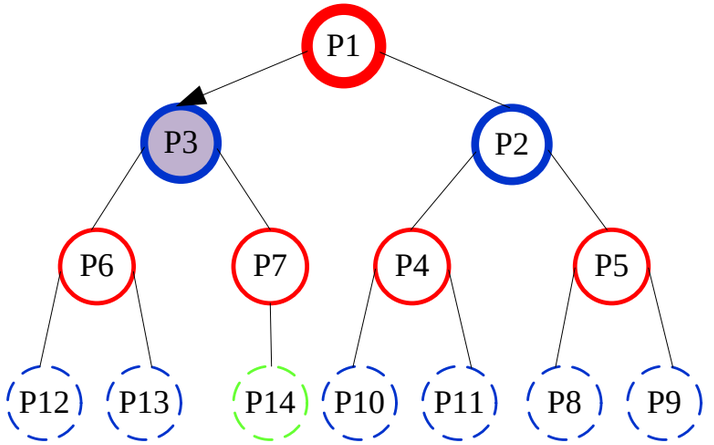
Since P14 is on the left hand side of P1, the left hand side of P1 is traced for nearest neighbour first.

Searching for the Nearest Neighbor of a Point

- **Nearest neighbor of a given point (P14): Schematic view**



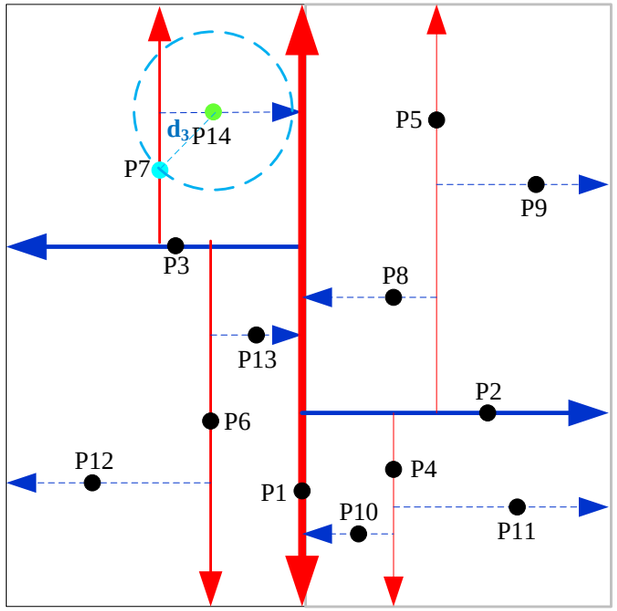
Candidate Point = P3
Minimum Distance = d_2



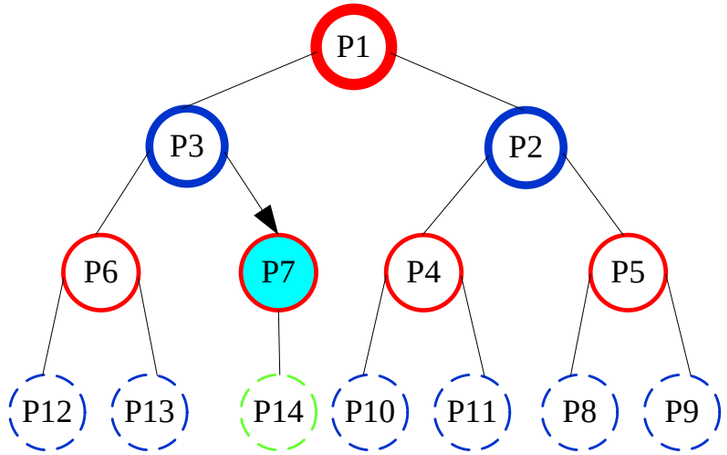
Since P14 is on the right hand side of P3, the right hand side of P3 is traced for nearest neighbour first.

Searching for the Nearest Neighbor of a Point

- **Nearest neighbor of a given point (P14): Schematic view**



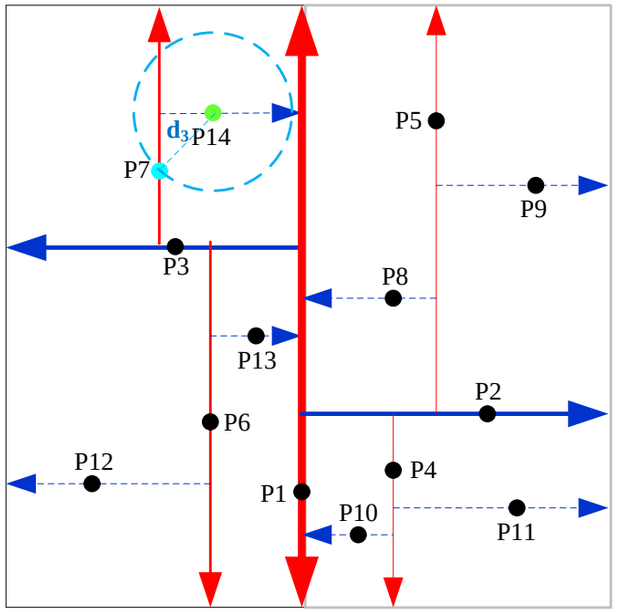
Candidate Point = P7
Minimum Distance = d_3



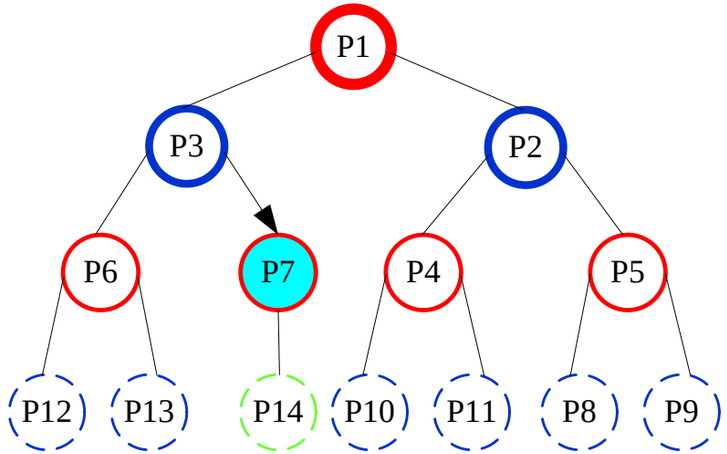
Since P14 is on the right hand side of P7, the right hand side of P7 is traced for nearest neighbour first.

Searching for the Nearest Neighbor of a Point

- **Nearest neighbor of a given point (P14): Schematic view**



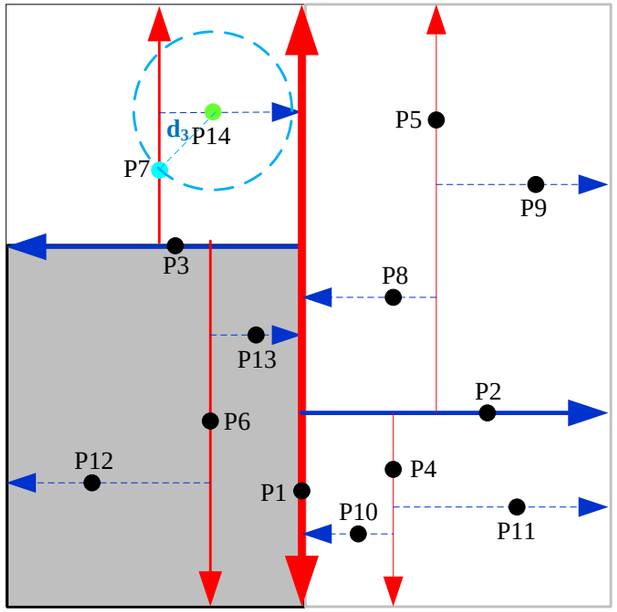
Candidate Point = P7
Minimum Distance = d_3

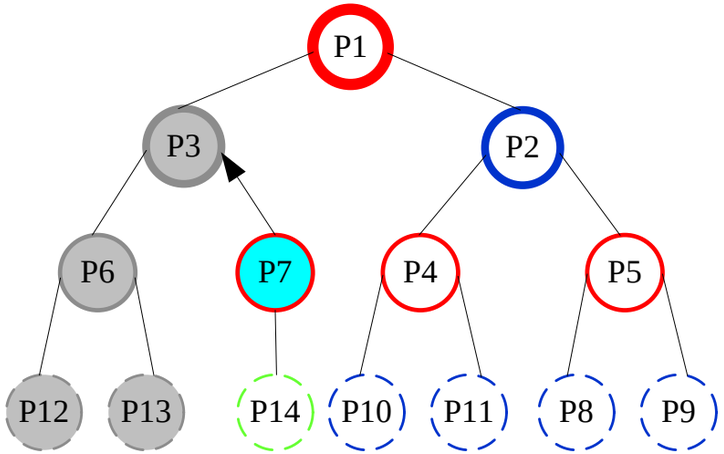


The hyperplane passing through P7 intersects the sphere, with radius equivalent to the minimum distance, centered at the point of interest, We should also check the left hand side of P7 for the nearest neighbour (there is not any node on the left hand side of P7).

Searching for the Nearest Neighbor of a Point

- **Nearest neighbor of a given point (P14): Schematic view**



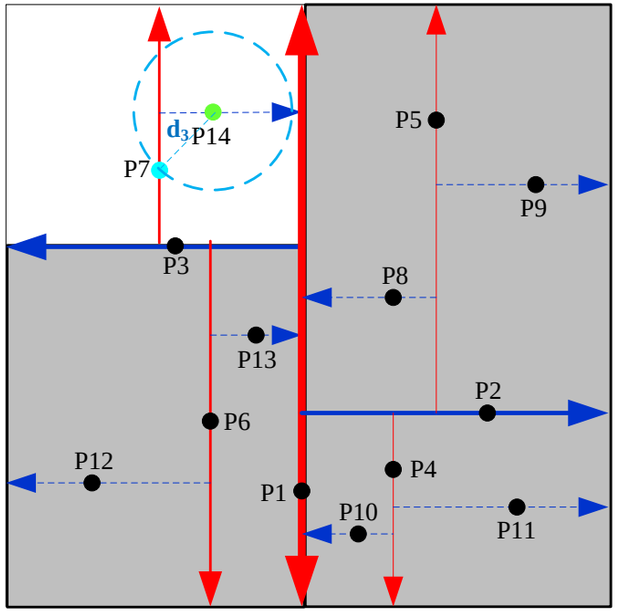


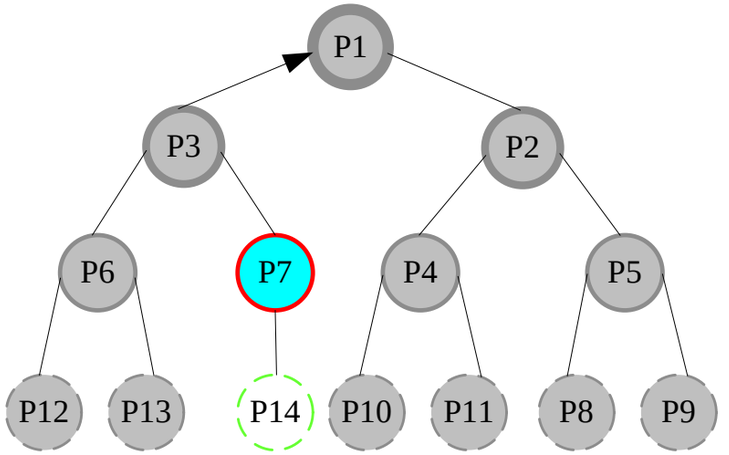
Candidate Point = P7
Minimum Distance = d_3

Since the hyperplane passing through P3 does not intersect the sphere, with radius equivalent to the minimum distance, centered at the point of interest, there is no need to check the left hand side of P3 for the nearest neighbour.

Searching for the Nearest Neighbor of a Point

- **Nearest neighbor of a given point (P14): Schematic view**





Nearest Neighbour Point = P7
Minimum Distance = d_3

Since the hyperplane passing through P1 does not intersect the sphere, with radius equivalent to the minimum distance, centered at the point of interest, there is no need to check the right hand side of P1 for the nearest neighbour.

Searching for N. N. of a Point in a Given Range

- **This is implemented by a modified nearest neighbor search. The modifications are:**

  I. Start with the root node, the algorithm moves down the tree recursively in the same way it would if the point in question were being inserted (Initial distance is not reduced as closer points are discovered)

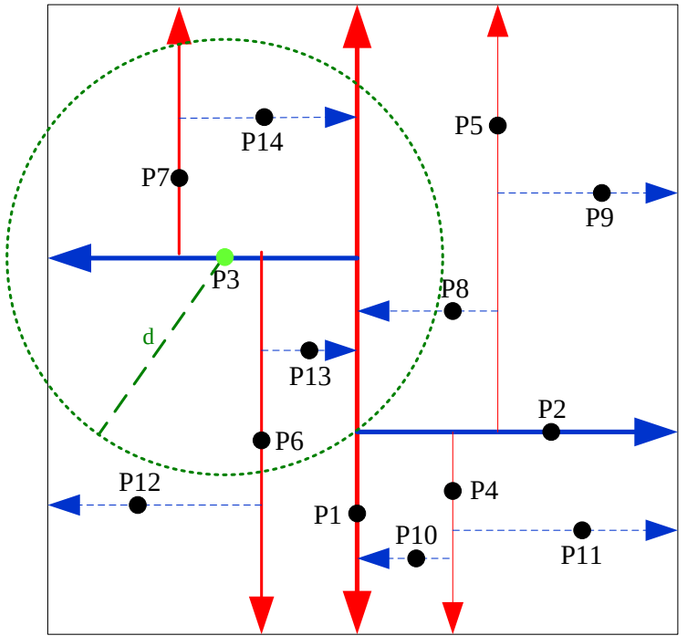
  II. Steps I is repeated until the algorithm reaches the leaf node.

  III. Search the other side of the splitting plane for points with distances less than the defined range by checking the intersection of the splitting hyper plane with a sphere centered at the point in question with a radius equivalent to the defined range. In case of intersection between them, the other branch of the tree should also be searched.

  IV. All discovered points within the defined range "**d**" are returned.

Searching for N. N. of a Point in a Given Range

- **This is implemented by a modified nearest neighbor search. The modifications are:**
  - Initial distance is not reduced as closer points are discovered.
  - All discovered points within the distance **d** are returned.



Searching for *k* Nearest Neighbors of a Point

I. Find the nearest neighbor of the point in question

II. Compute the distance between the point in question and its nearest neighbor

III. Calculate the radius for a new search by assuming a square whose dimensions are $\sqrt{k}d \times \sqrt{k}d$, where d is the distance to the nearest neighbor

$$r = \frac{\sqrt{2}\sqrt{k}d}{2} = \frac{\sqrt{2k}}{2}d$$

IV. Find the neighboring points in a spherical neighborhood with radius *r* centered at the point in question

V. If less than *k* points are found in the spherical neighborhood, the search radius is increased until at least *k* points are found in the defined neighborhood.

VI. If more than *k* points are found in the spherical neighborhood, only the *k* nearest neighbors are returned.