# K-D Tree: Query Nearest Neighbor

Weikang Chen

May 9, 2016

## 1 Main Idea

Traverse the kd-tree by recursive function: *findNearestUtil* (...)

1. On each node, determine whether to go to left or to right by checking the query point on the left or right of split axis.
2. Recursively call *findNearestUtil* on the left child or right chile, and get the its most likely nearest point, there is still chance on another side.
3. Backtracing: calculate the distance between the found nearest above and the query point. If that distance is larger than the distace between the query point and split axis, we have more nearest points candidates waiting on anther child of current node. We recursively call *findNearestUtil* on another side, as well

## 2 Class Design

1. Class Template Point:
   (a) The template type could be float, double or even long double, depend on how much precision we want.
   (b) I need it to work like a vector, so I overloade the index operator []
   (c) Also need to store its original index, so we could trace the identity of point

```
struct Point {
    vector<double> coordinates;
    int original_index;
    Point(vector<double> xs, int id=-1): coordinates(xs), original_index(id) {}
    double& operator[] (const int id);
    const double& operator[] (const int id) const;
    const size_t dimension() const;

};
```

2. Class KDTree
   (a) Construtor and destructor recursively
   (b) For constructor or build, I use quick select to find the middle of the current list of points
   (c) Save() into and load() from custom file, serialization and deserialization the whol kdtree recursively. Respect RAII.

(d) Store refernce on leaf. And store split point and split axis on path

(e) **Bonus Point: I made a spearted member function: splitPoint_method1(...), which allow to change it independently. One easy way to set split axis as the mod: depth % dimension, see splitPoint_method2(...) in the source file KDTree.cpp for details.**

(f) Put the kd-tree into static library, expose the public member function for outside use. Final application is build_kdtree and query_kdtree.

```
class KDTree {
private:
    struct KDNode {
        Point &point;
        int split_dim;
        KDNode *left;
        KDNode *right;
    };
    KDNode *root;
    vector<Point> m_points;
    int m_points_size;
    int m_point_dim;
    int splitPoint_method1(vector<Point*> &points, int l, int r);
    int splitPoint_method2(vector<Point*> &points, int depth);
    Point* KDTree::findNearestUtil(KDNode *cur, Point query);

public:
    KDTree();
    KDTree(vector<Point> &points);
    ~KDTree();
    Point &findNearest(Point query);
    int save(const char* filename);
    int load(const char* filename);
};
```

# 3 Space and Time Complexity

| | Average | Worst |
|---|---|---|
| Space | $O(n)$ | $O(n)$ |
| Construct Time | $O(n \log n)$ | $O(n \log n)$ |
| Search Time | $O(\log n)$ | $O(n)$ |

# 4 Further Improvements

If give more time, I might

1. Design a more clear interface between point, kd-node and kd-tree;
2. Add more functionalites of kd-tree, like insert and erase of KDTree's Node.
3. Further study on the impact of other choice of splitting axis.
4. Reduce the size of saving file for kd-tree, by using binary format file.