Nearest neighbor search is an inner loop of many parts of PCL (filters, surface, features, registration)

- Needs to be as fast as possible

**FLANN** - Fast Library for Approximate Nearest Neighbors

- [http://www.cs.ubc.ca/~mariusm/flann](http://www.cs.ubc.ca/~mariusm/flann)
- C, C++, Matlab and Python bindings
- Exact nearest neighbor search in low dimensional spaces (3D) using kd-trees
- Approximate nearest neighbor search in high dimensional spaces

**Octree** - 3D search
Nearest Neighbor Search

- Nearest neighbor search problem
  - Given a set of points $P = p_1, p_2, ..., p_n$ in a metric space $X$, preprocess them in such a way that given a new point $q \in X$ finding the closest $p_i$ to $q$ can be done easily
- K-Nearest neighbor search
  - find the closest $K$ neighbors
- Radius nearest neighbor search
  - find all the neighbors within a certain radius
1. KdTree

2. 3D Nearest Neighbor Search

3. High Dimensional Nearest Neighbor Search

4. Octree
The KD-Tree

- recursively divide the data points based on a single dimension
  - how to choose the dimension in which to divide the data?
  - where to divide?
- binary tree
- when searching entire branches can be ignored due to being too far away from the query point
- very efficient for low dimensionality data
KD-Tree Example
KD-Tree Example
KD-Tree Example

Point Cloud Library (PCL)
KD-Tree Example
KD-Tree Example

Point Cloud Library (PCL)
KD-Tree Example
KD-Tree Example

Point Cloud Library (PCL)
during 2011 GSOC, 2 students are working on a GPU based kd-tree implementation

encouraging preliminary results, speedups of 8-10x compared to CPU implementation
Header: #include `<pcl/kdtree/kdtree_flann.h>`

Class: `template<typename PointT> class pcl::KdTreeFLANN`

K-nearest neighbor search

```cpp
int nearestKSearch (const PointT &point, int k,
                    vector<int> &k_indices, vector<float> &k_distances);
```

Radius search

```cpp
int radiusSearch (const PointT &point, double radius,
                  vector<int> &k_indices,
                  vector<float> &k_distances, int max_nn = -1);
```
PointCloud<PointXYZ>::Ptr cloud (new PointCloud<PointXYZ>);
PointXYZ searchPoint;

// ... populate the cloud and the search point

// create a kd-tree instance
KdTreeFLANN<PointXYZ> kdtree;

// assign a point cloud - this builds the tree
kdtree.setInputCloud (cloud);

// pre-allocate the neighbor index and
// distance vectors
int K = 10;
std::vector<int> pointsIdx(K);
std::vector<float> pointsSquaredDist(K);

// K nearest neighbor search
kdtree.nearestKSearch (searchPoint, K, pointsIdx, pointsSquaredDist);
PointCloud<PointXYZ>::Ptr cloud (new PointCloud<PointXYZ>);
PointXYZ searchPoint;

// ... populate the cloud and the search point

// create a kd-tree instance
KdTreeFLANN<PointXYZ> kdtree;

// assign a point cloud - this builds the tree
kdtree.setInputCloud (cloud);

std::vector<int> pointIdxRadius;
std::vector<float> pointsSquaredDistRadius;
float radius = ...;

// radius search
int count = kdtree.radiusSearch (searchPoint, radius,
        pointIdxRadiusSearch, pointsSquaredDistRadius);
$ cd $PCL_ROOT/doc/tutorials/content/sources/kdtree_search
$ mkdir build
$ cd build
$ cmake ..
$ make
$ ./kdtree_search

K nearest neighbor search at (701.248 662.202 554.841) with K=10
  702.91 601.583 521.043 (squared distance: 4819.7)
  676.792 699.92 482.203 (squared distance: 7297.07)
  731.215 717.665 491.714 (squared distance: 7959.15)
  670.142 707.355 476.051 (squared distance: 9214.31)
  681.636 728.872 479.31 (squared distance: 10534.5)
  683.843 581.742 492.494 (squared distance: 10663.9)
  696.085 705.888 457.71 (squared distance: 11369.7)
  683.603 667.109 430.477 (squared distance: 15801.9)
  721.228 684.503 430.334 (squared distance: 16398.5)
  829.566 676.396 560.64 (squared distance: 16700.6)

Neighbors within radius search at (701.248 662.202 554.841) with radius=114.069
  702.91 601.583 521.043 (squared distance: 4819.7)
  676.792 699.92 482.203 (squared distance: 7297.07)
  731.215 717.665 491.714 (squared distance: 7959.15)
  670.142 707.355 476.051 (squared distance: 9214.31)
  681.636 728.872 479.31 (squared distance: 10534.5)
  683.843 581.742 492.494 (squared distance: 10663.9)
  696.085 705.888 457.71 (squared distance: 11369.7)
1. KdTree

2. 3D Nearest Neighbor Search

3. High Dimensional Nearest Neighbor Search

4. Octree
Motivation

- Object recognition (TOD)

TOD ©Willow Garage
Motivation

- Object recognition (TOD)
- Image stitching (AutoStitch, Hugin)
Motivation

- Object recognition (TOD)
- Image stitching (AutoStitch, Hugin)
- 3D Reconstruction (Photosynth)
Motivation

- Object recognition (TOD)
- Image stitching (AutoStitch, Hugin)
- 3D Reconstruction (Photosynth)
- 3D object classification (VFH)
Motivation

- Object recognition (TOD)
- Image stitching (AutoStitch, Hugin)
- 3D Reconstruction (Photosynth)
- 3D object classification (VFH)
- Content based image retrieval
Motivation

- Object recognition (TOD)
- Image stitching (AutoStitch, Hugin)
- 3D Reconstruction (Photosynth)
- 3D object classification (VFH)
- Content based image retrieval
- Visual SLAM
For high dimensionality data, no exact algorithm faster than linear search is known

Approximate nearest neighbor search is used to obtain large speedups

FLANN contains several algorithms for high dimensional approximate nearest neighbor search

- KDTreeIndex (randomized kd-tree forest)
- KMeansIndex (hierarchical k-means tree)
- HierarchicalClusteringIndex (clustering tree in a generic metric space)*
- LshIndex (locality sensitive hashing)*
Randomized KD-Trees

- multiple trees are built in parallel
- the split dimension is chosen randomly from the first $D$ dimensions with greatest variance ($D=5$)
- at search time a single priority queue is used across all trees
- search is terminated after a predefined number of tree leaves are checked
Hierarchical K-Means Tree

- **Building the tree**
  - built by splitting the data at each level of the tree using k-means clustering
  - apply the same procedure recursively on each cluster
  - just a few iterations of the k-means clustering give good results

- **Exploring the tree**
  - unexplored branches are added to a priority queue while traversing the tree
  - restart search from best branch in the priority queue

(Nistér & Stewénius, 2006)
FLANN Usage

- **Header:**
  ```cpp
  #include <flann/flann.h>
  ```

- **Class:**
  ```cpp
  template<typename Distance>
  class flann::Index
  ```

- **K-nearest neighbor search**
  ```cpp
  void knnSearch(const Matrix<ElementType>& queries,
                 Matrix<int>& indices,
                 Matrix<DistanceType>& dists,
                 int knn, const SearchParams& params)
  ```

- **Radius search**
  ```cpp
  int radiusSearch(const Matrix<ElementType>& query,
                   Matrix<int>& indices,
                   Matrix<DistanceType>& dists,
                   float radius, const SearchParams& params)
  ```
```cpp
flann::Matrix<float> data;
flann::Matrix<float> queries;
// populate the matrix with features
// one feature/row

// build randomized kd-tree index (4 trees)
flann::Index< L2<float> > index(data, flann::KDTreeIndexParams(4));
index.buildIndex();

// allocate memory for results
int k = 10;
int n = queries.rows;
flann::Matrix<int> k_indices(new int[n*k], n, k);
flann::Matrix<float> k_distances(new float[n*k], n, k);

// KNN search
index.knnSearch (queries, k_indices, k_distances, k,
                 flann::SearchParams(256));
```
Speedup with precision for different dataset sizes
KdTree 3D Nearest Neighbor Search
High Dimensional Nearest Neighbor Search
Octree

VFH Recognition

Point Cloud Library (PCL)
VFH Recognition

Point Cloud Library (PCL)
Compile & Try


```
$ cd $PCL_ROOT/doc/tutorials/content/sources/vfh_recognition
$ wget http://dev.pointclouds.org/attachments/download/216/vfh_recognition_tutorial_data.tbz
$ tar -xzf vfh_recognition_tutorial_data.tbz
$ mkdir build
$ cd build
$ cmake ..
$ make
$ cd ..
$ ./build/build_tree data
...
$ ./build/nearest_neighbors -k 16 -thresh 50 data/000.580.67/1258730231333_cluster_0_nxyz_vfh.pcd
```
Octree Overview

Octree - 3D hierarchical spatial tree data structure
- Recursive divide & conquer algorithm
- Binary subdivision of occupied cells into 8 octants (voxels)

2D Example (Quadtree):
Octree Overview

- Root node describes a cubic bounding box which encapsulates all points
- Child nodes recursively subdivide point space
- Nodes have up to eight children ⇒ Byte encoding
Octree Usage

Instantiate octree:

```cpp
float voxelSize = 0.01f; // voxel resolution
OctreePointCloud<PointXYZ> octree (voxelSize);
```

Set input point cloud (via Boost shared pointers):

```cpp
octree.setInputCloud (cloud);
```

Define octree bounding box (optional):

```cpp
// calculate bounding box of input cloud
octree.defineBoundingBox ();
// manually define bounding box
octree.defineBoundingBox (minX, minY, minZ, maxX, maxY, maxZ);
```

Add points from input cloud to octree:

```cpp
octree.addPointsFromInputCloud ();
```

Delete octree data structure:

```cpp
(pushes allocated nodes to memory pool!)
octree.deleteTree ();
```
Check if voxel at given point coordinates exist:

```cpp
double X, Y, Z;
bool occupied;
X = 1.0; Y = 2.0; Z = 3.0;
occupied = octree.isVoxelOccupiedAtPoint (X, Y, Z);
```

Get center points of all occupied voxels:
((voxel grid filter/downsampling))

```cpp```
std::vector<PointXYZ> pointGrid;
octree.getOccupiedVoxelCenters (pointGrid);
```

Delete voxel:

```cpp```
pcl::PointXYZ point_arg( 1.0, 2.0, 3.0 );
octree.deleteVoxelAtPoint ( point );
```
Provided algorithms in PCL using octrees for spatial decomposition:

- Search operations (neighbor search, radius search, voxel search)
- Downsampling (Voxel-grid / Voxel-centroid filter)
- Point cloud compression
- Spatial change detection
- Spatial point density analysis
- Occupancy checks/maps
- Collision detection
- ...

Point Cloud Library (PCL)
Points within radius search

- Depth first tree exploration
- At every node investigate occupied child voxels that overlap with search sphere

K nearest neighbor search:

- Priority queue (binary heap) of nodes and point candidates
- Investigate occupied child voxels (closest voxel first)
- Radius search with radius=distance to Kth point candidate
- Update radius with every new point candidate

Point Cloud Library (PCL)
Define search precision / error bound:

```cpp
octree.setEpsilon (double eps); // default: 0.0
```

Neighbors within voxel search:

```cpp
std::vector<int> pointIdxVec;

if (octree.voxelSearch (searchPoint, pointIdxVec))
{
    for (size_t i = 0; i < pointIdxVec.size (); ++i)
        std::cerr << cloud->points[pointIdxVec[i]].x
                    << cloud->points[pointIdxVec[i]].y
                    << cloud->points[pointIdxVec[i]].z << std::endl;
}
```

K nearest neighbor search:

```cpp
int K = 10;
std::vector<int> pointIdxNKNSearch;
std::vector<float> pointNKNSquaredDistance;

if ( octree.nearestKSearch (searchPoint, K,
                             pointIdxNKNSearch, pointNKNSquaredDistance) > 0 )
{
    ...
}
```
Neighbors within radius search:

```cpp
std::vector<int> pointIdxRadiusSearch;
std::vector<float> pointRadiusSquaredDistance;
float radius = 0.1;

if ( octree.radiusSearch (searchPoint, radius,
    pointIdxRadiusSearch, pointRadiusSquaredDistance) > 0 )
{
    ...
}
```

Approx. neighbors within radius search:
(only scans points within “search point voxel”)

```cpp
std::vector<int> pointIdxRadiusSearch;
std::vector<float> pointRadiusSquaredDistance;
float radius = 0.1;

if ( octree.approxNearestSearch (searchPoint, radius,
    pointIdxRadiusSearch, pointRadiusSquaredDistance) > 0 )
{
    ...
}
```
Template configuration:

```
OctreeBase Octree2BufBase OctreeLowMemBase
OctreePointCloud< PointT, LeafT, OctreeT >
OctreeLeafEmpty OctreeLeafDataT OctreeLeafDataTVector
```

Optimized performance&memory usage:
- Select octree base implementation
- Select/define leaf node class
- Serialization callbacks (serializeLeafCallback, deserializeLeafCallback, serializeNewLeafCallback)
OctreePointCloud classes:

```cpp
float resolution = 0.01f;

// equal to OctreePointCloudPointVector<PointXYZ>
OctreePointCloud<PointXYZ> octreeA (resolution);

// manages indices vectors in leaf nodes
OctreePointCloudPointVector<PointXYZ> octreeB (resolution);
// keeps a single point indices in leaf nodes
OctreePointCloudSinglePoint<PointXYZ> octreeC (resolution);
// does not store any point information in leaf node
OctreePointCloudOccupancy<PointXYZ> octreeD (resolution);
```

Octree-Base selection via typedefs:

```cpp
OctreePointCloud<PointXYZ>::SingleBuffer octreeSB (resolution);
OctreePointCloud<PointXYZ>::DoubleBuffer octreeDB (resolution);
OctreePointCloud<PointXYZ>::LowMem octreeLM (resolution);
```
Octree2BufBase implementation:

- Create octrees at high rate
- Advanced memory management:
  - Previous tree structure is kept in memory
  - Maximum reusage of already allocate branch&leaf nodes
  - Unused node instances are pushed to a memory pool for later reusage
- Enables comparison of octree structure (change detection)

Switching between octree buffers:

```java
octree.switchBuffers();
```
class SimpleSpatialChangeDetection
{
  public:
    OctreePointCloudChangeDetector<PointXYZRGB>* octree;
    ...
  
  void cloud_cb_ (const pcl::PointCloud<pcl::PointXYZRGB>::ConstPtr &cloud)
  {
    if (!viewer.wasStopped ())
    {
      // Switch octree buffers
      octree.switchBuffers ();

      // Add points from cloud to octree
      octree.setInputCloud (cloud);
      octree.addPointsFromInputCloud ();

      std::vector<int> newPointIdxVector;

      /* Get vector of point indices from octree voxels
         which did not exist in previous buffer */
      octree.getPointIndicesFromNewVoxels (newPointIdxVector);
    }
  }
};
Real-time spatial change detection based on XOR comparison of octree structure

DEMO: See /visualization/tool/openni_change_viewer
Example: Point density estimation
Design your own leaf node class:

```cpp
template<typename DataT>
class OctreePointCloudDensityLeaf : public OctreeLeafAbstract<DataT> {
public:

  virtual void setData (const DataT& point_arg) {
    pointCounter_++;
  }

  unsigned int getPointCounter () {
    return pointCounter_;  
  }

private:
  unsigned int pointCounter_;  
};
```
.. and your own OctreePointCloud class:

class OctreePointCloudDensity : public OctreePointCloud
  <PointT, OctreePointCloudDensityLeaf<int> , OctreeT>
{
  public:
    ...

    unsigned int
getVoxelDensityAtPoint (const PointT& point_arg) const
    {
      unsigned int pointCount = 0;

      OctreePointCloudDensityLeaf<int>* leaf =
        this->findLeafAtPoint (point_arg);

      if (leaf) pointCount = leaf->getPointCounter ();

      return pointCount;
    }
};
/* for a full list of profiles see: /io/include/pcl/compression/compression_profiles.h */

```
compression_Profiles_e compressionProfile = pcl::octree::MED_RES_ONLINE_COMPRESSION_WITH_COLOR;
```

```
// instantiate point cloud compression for encoding and decoding
PointCloudCompression<PointXYZ> PointCloudEncoder (compressionProfile);
PointCloudCompression<PointXYZ> PointCloudDecoder ();
```

```
// iostream to read/write compressed point cloud data
std::stringstream compressedData;
```

```
// compress & decompress point cloud
PointCloudEncoder->encodePointCloud (cloud, compressedData);
PointCloudDecoder->decodePointCloud (compressedData, cloudOut);
```
See octree search tutorial at:
http://pointclouds.org/documentation/tutorials/octree.php

See point cloud compression tutorial at:
http://pointclouds.org/documentation/tutorials/compression.php

See change detection tutorial at:
http://pointclouds.org/documentation/tutorials/octree_change.php

Point cloud compression and streaming app:
PCL_ROOT/apps/openni_stream_compression

Change detection app:
PCL_ROOT/visualization/tools/openni_change_viewer