1. Overview
2. Octree Interface
3. Octree Framework
4. Neighbor Search
5. Application Scenarios
6. Summary - Compile & Try
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
Provided algorithms in PCL using octrees for spatial decomposition:

- Search operations (neighbor, radius, 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 merging/accumulation
- ...

Julius Kammerl / PCL :: Search
1. Overview

2. Octree Interface

3. Octree Framework

4. Neighbor Search

5. Application Scenarios

6. Summary - Compile & Try
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:

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)

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

Query points within a voxel:

std::vector<int> pointIdxVec;
octree.voxelSearch (searchPoint, pointIdxVec);

Delete voxel:

pcl::PointXYZ point_arg( 1.0, 2.0, 3.0 );
octree.deleteVoxelAtPoint ( point );
Instantiate octree:

```cpp
OctreePointCloud<PointXYZ> octreeA (1.0f);
// add point data to octree
octreeA.setInputCloud (cloudIn);
octreeA.addPointsFromInputCloud ();
```

Instantiate iterators:

```cpp
// iterates over all octree nodes
OctreePointCloud<PointXYZ>::Iterator it (octreeA);
// iterates over all leaf nodes of the octree
OctreePointCloud<PointXYZ>::LeafNodeIterator itL (octreeA);
```

Iterate over leaf nodes:

```cpp
std::vector<int> indexVector;
unsigned int leafCount = 0;
while ( *++itL )
{
    *itL->getData (indexVector);
    ++leafCount;
}
```
How to access all leaf nodes in the octree:

- Full tree traversal during serialization
- Virtual callback methods are executed for every leaf node

Example:

```cpp
class MyOctreePointCloud : public OctreePointCloud {
  ...
  virtual void serializeLeafCallback
    (OctreeLeaf& leaf_arg, const OctreeKey& key_arg) {
    // reference to point indices vector stored within octree leaf
    const std::vector<int>& leafIdx = leaf_arg.getIdxVector();
    ...
  }
  ...
}
```
Octree Framework

Template configuration:

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

Optimized performance & memory usage:

- Select octree base implementation
- Select/define leaf node class
- Serialization callbacks (serializeLeafCallback, deserializeLeafCallback, serializeNewLeafCallback)
OctreeTypes

OctreePointCloud classes:

```cpp
default
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
default
OctreePointCloud<PointXYZ>::SingleBuffer octreeSB (resolution);
OctreePointCloud<PointXYZ>::DoubleBuffer octreeDB (resolution);
OctreePointCloud<PointXYZ>::LowMem octreeLM (resolution);
```
1. Overview
2. Octree Interface
3. Octree Framework
4. Neighbor Search
5. Application Scenarios
6. Summary - Compile & Try
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 only occupied child voxels (closest voxel first)
- Radius search with radius=distance to Kth point candidate
General PCL search interface:

```cpp
Search<PointXYZ>* octree = new Octree<PointXYZ> (resolution);
Search<PointXYZ>* kdtree = new KdTree<PointXYZ>();
Search<PointXYZ>* organizedSearch = new OrganizedNeighbor<PointXYZ>();
```

Define search precision / error bound:

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

K nearest neighbor search:

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

if ( octree.nearestKSearch (searchPoint, K, pointIdxNKNSearch, pointNKNSquaredDistance) > 0 )
{
    for (size_t i = 0; i < pointIdxVec.size (); ++i)
    {
        ...
    }
}
```
Neighbors within radius search:

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

// scan all voxel candidates
if (octree.radiusSearch(searchPoint, radius,
                        pointIdxRadiusSearch, pointRadiusSquaredDistance) > 0 )
    { ... }

// scan only points within "search point voxel"
if (octree.approxNearestSearch(searchPoint, radius,
                                pointIdxRadiusSearch, pointRadiusSquaredDistance) > 0 )
    { ... }
```

Ray tracing:

```cpp
PointCloud<PointXYZ>::Ptr voxelCenters (new PointCloud<PointXYZ> ());
Eigen::Vector3f origin ( 0.0f, 0.0f, 0.0f );
Eigen::Vector3f direction ( 0.1f, 0.2f, 0.3f );

if (octree.getIntersectedVoxelCenters(origin, direction,
                                      voxelCenters->points) > 0 )
    { ... }
```
1. Overview

2. Octree Interface

3. Octree Framework

4. Neighbor Search

5. Application Scenarios

6. Summary - Compile & Try
Example: Point density estimation
Extending the octree - design your own leaf node class:

```cpp
#include <pcl::octree.h>

// Define a new leaf node class for point density estimation

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:

```cpp
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;
    }
};
```
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);

        }
    }
};
Double buffered octree nodes:

- Ensure structural consistency to the preceding point cloud by adaptively matching spatially related octree nodes.

- XOR operator extract changes in children configuration.
Real-time spatial change detection based on XOR comparison of octree structure

DEMO: See /visualization/tool/openni_change_viewer
General compression approach for unstructured point clouds (varying size, resolution, density, point ordering)

Exploit spatial&temporal redundancies in point clouds

Keep introduced coding distortion below sensor noise
Point Cloud Compression

Encoding Pipeline:

- Octree Structure
- Point Component Encoding
- Position Detail Coefficients
- Entropy Encoding
- Compressed PC
- Binary Serialization
- Component Voxel Avg. + Detail Coefficients
- Point Detail Encoding

Example:

```cpp
/* 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);
```
Outline

1. Overview
2. Octree Interface
3. Octree Framework
4. Neighbor Search
5. Application Scenarios
6. Summary - Compile & Try
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