```
#include <pcl/point_cloud.h>
#include <pcl/octree/octree.h>

#include <iostream>
#include <vector>
#include <ctime>

int
main (int argc, char** argv)
{
  srand ((unsigned int) time (NULL));

  pcl::PointCloud<pcl::PointXYZ>::Ptr cloud (new
pcl::PointCloud<pcl::PointXYZ>);

  // Generate pointcloud data
  cloud->width = 1000;
  cloud->height = 1;
  cloud->points.resize (cloud->width * cloud->height);

  for (size_t i = 0; i < cloud->points.size (); ++i)
  {
    cloud->points[i].x = 1024.0f * rand () / (RAND_MAX + 1.0f);
    cloud->points[i].y = 1024.0f * rand () / (RAND_MAX + 1.0f);
    cloud->points[i].z = 1024.0f * rand () / (RAND_MAX + 1.0f);
  }

  float resolution = 128.0f;

  pcl::octree::OctreePointCloudSearch<pcl::PointXYZ> octree
(resolution);

  octree.setInputCloud (cloud);
  octree.addPointsFromInputCloud ();

  pcl::PointXYZ searchPoint;

  searchPoint.x = 1024.0f * rand () / (RAND_MAX + 1.0f);
  searchPoint.y = 1024.0f * rand () / (RAND_MAX + 1.0f);
  searchPoint.z = 1024.0f * rand () / (RAND_MAX + 1.0f);

  // Neighbors within voxel search

  std::vector<int> pointIdxVec;

  if (octree.voxelSearch (searchPoint, pointIdxVec))
  {
    std::cout << "Neighbors within voxel search at (" << searchPoint.x
     << " " << searchPoint.y
     << " " << searchPoint.z << ")"
     << std::endl;

    for (size_t i = 0; i < pointIdxVec.size (); ++i)
   std::cout << "    " << cloud->points[pointIdxVec[i]].x
```

```
        << " " << cloud->points[pointIdxVec[i]].y
        << " " << cloud->points[pointIdxVec[i]].z << std::endl;
  }

  // K nearest neighbor search

  int K = 10;

  std::vector<int> pointIdxNKNSearch;
  std::vector<float> pointNKNSquaredDistance;

  std::cout << "K nearest neighbor search at (" << searchPoint.x
            << " " << searchPoint.y
            << " " << searchPoint.z
            << ") with K=" << K << std::endl;

  if (octree.nearestKSearch (searchPoint, K, pointIdxNKNSearch,
pointNKNSquaredDistance) > 0)
  {
    for (size_t i = 0; i < pointIdxNKNSearch.size (); ++i)
      std::cout << "    "  <<   cloud->points[ pointIdxNKNSearch[i] ].x
                << " " << cloud->points[ pointIdxNKNSearch[i] ].y
                << " " << cloud->points[ pointIdxNKNSearch[i] ].z
                << " (squared distance: " << pointNKNSquaredDistance[i]
<< ")" << std::endl;
  }

  // Neighbors within radius search

  std::vector<int> pointIdxRadiusSearch;
  std::vector<float> pointRadiusSquaredDistance;

  float radius = 256.0f * rand () / (RAND_MAX + 1.0f);

  std::cout << "Neighbors within radius search at (" << searchPoint.x
      << " " << searchPoint.y
      << " " << searchPoint.z
      << ") with radius=" << radius << std::endl;


  if (octree.radiusSearch (searchPoint, radius, pointIdxRadiusSearch,
pointRadiusSquaredDistance) > 0)
  {
    for (size_t i = 0; i < pointIdxRadiusSearch.size (); ++i)
      std::cout << "    "  <<   cloud->points[ pointIdxRadiusSearch[i]
].x
                << " " << cloud->points[ pointIdxRadiusSearch[i] ].y
                << " " << cloud->points[ pointIdxRadiusSearch[i] ].z
                << " (squared distance: " <<
pointRadiusSquaredDistance[i] << ")" << std::endl;
  }

}
```