PCL 库学习二（查找、点云压缩、octree）

转载 磊磊哈哈 于 2021-03-19 21:08:32 发布 272 收藏 5 版权

激光slam 学习 专栏收录该内容 27 订阅 28 篇文章 订阅专栏

来自 点云 PCL库到精通到入门

1. 查找

- K临近

```
kdtree.nearestKSearch (searchPoint, K, pointIndex, pointSquareDistance)
        //searchPoint            查询点
        //K                      最临近点的个数
        //pointIndex             查询到的临近值
        //pointSquareDistance    存储的临近距离
        //return    搜索到的点的个数
```

- R半径搜索

```
kdtree.radiusSearch(searchPoint,R,RpointIndex,RpointSquareDistance,20)){
        //searchPoint            查询点
        //R                      最大半径
        //RpointIndex             查询到的临近值
        //RpointSquareDistance    存储的临近距离
        //20                     最多搜搜索到的点
        //return    搜索到的点的个数
```

实例：

```
#include <pcl/point_types.h>
#include <pcl/point_cloud.h>
#include <pcl/kdtree/kdtree_flann.h>
#include <vector>
#include <ctime>
#include <iostream>

using namespace std;
using namespace pcl;
int main(int argc, const char** argv) {
    srand(time(NULL));
    //创建数据
    pcl::PointCloud<PointXYZ>::Ptr cloud(new PointCloud<PointXYZ>);
    cloud->width=100;
    cloud->height=20;
    cloud->points.resize(cloud->width*cloud->height);
    for(int 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);
    }

    //创建kd树对象
    KdTreeFLANN<PointXYZ> kdtree;
    //创建搜索空间
    kdtree.setInputCloud(cloud);

    //查询点，渎职随机
    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);

    //k临近搜索
    int K=10;
    vector<int> pointIndex(K);              //存储查询点的k临近索引
    vector<float> pointSquareDistance(K);   //存储临近点的平方距离

    //输出查询点的信息
    cout<<"X :"<<searchPoint.x<<";Y:"<<searchPoint.y<<";Z:"<<searchPoint.z<<endl;


    //K临近搜索
    cout<<" K临近搜索" <<std::endl;
    cout<<"K"<<K<<endl;
    if (kdtree.nearestKSearch (searchPoint, K, pointIndex, pointSquareDistance) >0 )
    {
        //searchPoint            查询点
        //K                      最临近点的个数
        //pointIndex             查询到的临近值
        //pointSquareDistance    存储的临近距离
        //return    搜索到的点的个数
        for (size_t i=0; i<pointIndex.size (); ++i){
            std::cout<<"第i个"<<i;
            std::cout<<"点的索引值: "<<pointIndex[i]<<endl;
            cout<<"X:"<<  cloud->points[ pointIndex[i] ].x <<endl;
            cout<<"Y:"<< cloud->points[pointIndex[i] ].y  <<endl;
            cout<<"Z:"<<cloud->points[pointIndex[i] ].z <<endl;
            cout<<" 平方距离: "<<pointSquareDistance[i] <<std::endl;;
        }
    }
    //R半径搜索
    float R=256.0f* rand () / (RAND_MAX +1.0f);
    vector<int> RpointIndex;               //存储查询点的R半径索引
    vector<float> RpointSquareDistance;   //存储临近点的平方距离
    cout<<" R半径搜索" <<std::endl;
     cout<<"R"<<R<<endl;
    if(kdtree.radiusSearch(searchPoint,R,RpointIndex,RpointSquareDistance,20)){
        //searchPoint            查询点
        //R                      最大半径
        //RpointIndex             查询到的临近值
        //RpointSquareDistance    存储的临近距离
        //20                     最多搜搜索到的点
        //return    搜索到的点的个数
        for(int i=0;i<RpointIndex.size();i++){
            std::cout<<"第i个"<<i;
            std::cout<<"索引值:"<<RpointIndex[i]<<endl;
            std::cout<<"X:"<<cloud->points[RpointIndex[i]].x<<endl;
            std::cout<<"Y:"<<cloud->points[RpointIndex[i]].y<<endl;
            std::cout<<"Z:"<<cloud->points[RpointIndex[i]].z<<endl;
            std::cout<<"距离:"<<RpointSquareDistance[RpointIndex[i]]<<endl;
        }
    }
```

磊磊哈哈
码龄6年 暂无认证
94 原创 15万+ 周排名 118万+ 总排名 10万+ 访问 等级
1352 积分 911 粉丝 105 获赞 34 评论 701 收藏
私信 关注
搜博主文章

热门文章
PCL点云学习六（边缘提取） 4164
gtsam 学习十（ISAM2 理论） 3685
PNP问题学习 2813
gtsam 学习九（imu 因子） 2598
获取激光雷达数据 2556

最新评论
PCL点云库 学习六(提取、条件剔除、半...
十一.先生: 大佬，我用条件剔除的方法对深度相机返回的点云进行高度上的剔除，但...
捷联惯导系统学习6.4(平方根滤波）
weixin_57937717: 想请问一下用待定系数法求出rk有两个值，代到量测更新中求Δ...
gtsam库 学习一 （因子图理论基础）
一大碗面加俩蛋: 贝叶斯定理写错了bro
捷联惯导系统学习2.1（反对称矩阵）
qq_41364482: 唉，网上这么多博客，没一个把行列式和矩阵分清的，，，
PCL 点云库学习四（双边滤波器）
小灰啾: 有序点云只是说点云存储结构像图片像素一样有width和height，并不是点云...

您愿意向朋友推荐“博客详情页”吗？
强烈不推荐 不推荐 一般般 推荐 强烈推荐

最新文章
FreeRTOS 笔记二 （中断配置和临界段）
freertos 笔记一（HAL 库移植）
pcl 学习论文阅读（Towards 3D Object Maps for Autonomous Household Robots）
2021年 39篇 2020年 105篇

分类专栏
Freertos
arm_linux
c++/c 2篇
学习 17篇
makefile 学习 2篇
ROS
BP
opencv 5篇
任务 1篇

```
    return 0;
}

```

2. 压缩

```
#include <pcl/point_types.h>
#include <pcl/point_cloud.h>
#include <pcl/io/ply_io.h>
#include <pcl/compression/octree_pointcloud_compression.h>
#include <iostream>
#include <pcl/octree/octree_pointcloud.h>
#include <pcl/visualization/cloud_viewer.h>
#include <fstream>
using namespace std;
using namespace pcl;
int main(int argc, const char** argv) {

    pcl::PointCloud<PointXYZRGB>::Ptr cloud(new pcl::PointCloud<PointXYZRGB>());
    //读取ply数据
    io::loadPLYFile("/home/n1/notes/pcl/compress/init.ply",*cloud);
    io::compression_Profiles_e compressionProfile=pcl::io::MANUAL_CONFIGURATION;
    //压缩选项
    // 通过设置:
    //      pointResolution:        点采样率，即精度
    //      octreeResolution:       八叉树分辨率，八叉树最小块
    //      doVoxelGridDownDownSampling: 是否开启体素滤波的下采样率
    //      iFrameRate:             编码率，每隔30次进行一次I编码，中间帧使用P编码
    //      colorBitResolution:     颜色所占bit;
    //      doColorEncoding:        是否开启颜色编码
    //   LOW_RES_ONLINE_COMPRESSION_WITHOUT_COLOR,
    //   LOW_RES_ONLINE_COMPRESSION_WITH_COLOR,

    //   MED_RES_ONLINE_COMPRESSION_WITHOUT_COLOR,
    //   MED_RES_ONLINE_COMPRESSION_WITH_COLOR,

    //   HIGH_RES_ONLINE_COMPRESSION_WITHOUT_COLOR,
    //   HIGH_RES_ONLINE_COMPRESSION_WITH_COLOR,

    //   LOW_RES_OFFLINE_COMPRESSION_WITHOUT_COLOR,
    //   LOW_RES_OFFLINE_COMPRESSION_WITH_COLOR,

    //   MED_RES_OFFLINE_COMPRESSION_WITHOUT_COLOR,
    //   MED_RES_OFFLINE_COMPRESSION_WITH_COLOR,

    //   HIGH_RES_OFFLINE_COMPRESSION_WITHOUT_COLOR,
    //   HIGH_RES_OFFLINE_COMPRESSION_WITH_COLOR,

    //   COMPRESSION_PROFILE_COUNT,
    //   MANUAL_CONFIGURATION
    bool showStatistics=true;
    io::OctreePointCloudCompression<PointXYZRGB>* PointCloudEncoder;//通过八叉树进行点云压缩

    //压缩
    PointCloudEncoder=new pcl::io::OctreePointCloudCompression<pcl::PointXYZRGB> (compressionProfile, showStati
    //io::OctreePointCloudCompression<> 参数
    //          compressionProfile_arg 参数设置
    //          自定义compressionProfile_arg参数
    //          octreeResolution_arg   分辨率
    //          doVoxelGridDownDownSampling_arg 是否开启体素滤波的下采样率
    //          iFrameRate_arg:         编码率，每隔30次进行一次I编码，中间帧使用P编码
    //          doColorEncoding_arg     启动颜色编码
    //          colorBitResolution_arg  RGB每一位的所占位数
    //          showStatistics_arg  是否将压缩相关的统计信息打印到标准输出上
    std::stringstream compressedData;//输入输出流
    pcl::PointCloud<pcl::PointXYZRGB>::Ptr compresscloud(new pcl::PointCloud<pcl::PointXYZRGB>());
    //压缩点云
    PointCloudEncoder->encodePointCloud(cloud,compressedData);
    //cloud
    //compressedData 二进制流文件

    //解压缩点云
    PointCloudEncoder->decodePointCloud(compressedData, compresscloud);
    cout<<"compressedData:size"<<sizeof(compressedData)<<endl;
    fstream out("/home/n1/notes/pcl/compress/compress.txt");
    string data;
    compressedData>>data;
    out.write(data.c_str(),sizeof(data));
    pcl::PLYWriter write;
    write.write("/home/n1/notes/pcl/compress/compress.ply",*compresscloud,false,false);
    delete(PointCloudEncoder);
    return 0;
}
```

3. octree

```
#include <pcl/pcl_macros.h>
#include <pcl/point_cloud.h>
#include <pcl/octree/octree.h>
#include <iostream>
using namespace std;;
using namespace pcl;
int main(int argc, const char** argv) {
    srand( (unsigned int)time(NULL));
    PointCloud<PointXYZ>::Ptr cloud(new PointCloud<PointXYZ>());
    cloud->width=200;
    cloud->height=20;
    cloud->resize(cloud->width*cloud->height);
    for(int i=0;i<cloud->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;
    //resolution octree最低的分辨率
    octree::OctreePointCloudSearch<PointXYZ> octree(resolution);
    //设置点云输出
    octree.setInputCloud(cloud);
    //构建八叉树
    octree.addPointsFromInputCloud();
    PointXYZ searchPoint;//查找点
    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);

    vector<int>pointIdexVec;//查询序号值
    //体素临近搜索
    if(octree.voxelSearch(searchPoint,pointIdexVec)){
        //查询点          searchPoint
        //查询点的序号      pointIdexVec
        cout<<"x:"<<searchPoint.x<<"y:"<<searchPoint.y<<"z:"<<searchPoint.z<<endl;
          cout<<"pointIdexVec.size():"<<pointIdexVec.size()<<endl;
        for(size_t i=0;i<pointIdexVec.size();i++){
            cout<<"x:"<<cloud->points[pointIdexVec[i]].x<<"y:"<<cloud->points[pointIdexVec[i]].y<<"z:"<<cloud->

        }
    }
    //K搜索
    vector<int> pointKsearchIdl;
    vector<float> pointKsquare;
    int K=10;
    if(octree.nearestKSearch(searchPoint,K,pointKsearchIdl,pointKsquare)>0){
        cout<<"x:"<<searchPoint.x<<"y:"<<searchPoint.y<<"z:"<<searchPoint.z<<endl;
        cout<<"pointKsearchIdl.size():"<<pointKsearchIdl.size()<<endl;
        for(size_t i=0;i<pointKsearchIdl.size();i++){
            cout<<"x:"<<cloud->points[pointKsearchIdl[i]].x
            <<"y:"<<cloud->points[pointKsearchIdl[i]].y
            <<"z:"<<cloud->points[pointKsearchIdl[i]].z
            <<"距离:"<<pointKsquare[i]<<endl;
        }
    }
    //R搜索
    vector<int> pointRsearchIdl;
    vector<float> pointRsquare;
    int R=256.0f* rand () / (RAND_MAX +1.0f);
    if(octree.radiusSearch(searchPoint,R,pointRsearchIdl,pointRsquare,20)>0){
        cout<<"x:"<<searchPoint.x<<"y:"<<searchPoint.y<<"z:"<<searchPoint.z<<endl;
        cout<<"pointRsearchIdl.size():"<<pointRsearchIdl.size()<<endl;
        for(size_t i=0;i<pointRsearchIdl.size();i++){
            cout<<"x:"<<cloud->points[pointRsearchIdl[i]].x
            <<"y:"<<cloud->points[pointRsearchIdl[i]].y
            <<"z:"<<cloud->points[pointRsearchIdl[i]].z
            <<"距离:"<<pointRsquare[i]<<endl;
        }
    }
    octree.approxNearestSearch
    return 0;
}

```