深蓝学院
shenlanxueyuan.com

三维点云处理第二次作业讲评

主讲人 李恺

题目

深蓝学院
shenlanxueyuan.com

Homework

- We provide one $N \times 3$ point cloud
- 8-NN search for each point to the point cloud
- Implement 3 NN algorithms
    1. Numpy brute-force search
    2. scipy.spatial.KDTree
    3. Your own kd-tree/octree in python or C++
- Report timing using method 1 as baseline
- This is a competition!
    - Timing of method 3 determine your grade

评分标准

- **优秀**　用C++重新实现，效率有提升
- **良好**　Python实现，对课上代码有思考，结果展示较为用心
- **合格**　按课上代码实现，得出合理结果
- **不合格**　完成度低，内容不完整

题目一

深蓝学院 shenlanxueyuan.com

- 用暴力搜索作为benchmark
  - 沿用作业代码给出的暴力搜索

```
begin_t = time.time()
diff = np.linalg.norm(np.expand_dims(query, 0) - db_np, axis=1)
nn_idx = np.argsort(diff)
nn_dist = diff[nn_idx]
brute_time_sum = time.time() - begin_t
```

排序不是必要的

排序：nlogn

不排序：nk

题目二

深蓝学院 shenlanxueyuan.com

- 测试Scipy的实现：较为简单，很多同学漏了这一题

```
# 功能: scipy_kdtree 匹配 k个最近邻
# 输入:
#     k: 匹配个数
#     dp_np :原始数据
#     query: 搜索信息
# 输出:
#     result_set: 搜索结果
def scipy_kdtree(k:int, dp_np:np.ndarray, query:np.ndarray):
    # 构建一棵树
    begin_t = time.time()
    sci_kdtree = scipy.spatial.KDTree(dp_np)
    dur_construct = time.time() - begin_t
    print("for scipy the construction duration is %.3f ms" %(dur_construct*1000))
    # 寻找一个点
    begin_t = time.time()
    d,index = sci_kdtree.query(query,k=k)
    dur = time.time()-begin_t
    print("for scipy kdtree the searching duration is %.3f ms" %(dur*1000))
```

构建

查找

题目三

深蓝学院 shenlanxueyuan.com

- 作业1：kd树

问题：

有的同学修改了切分位置的代码，切分点没有刚好落在某个方向的中点上，向左或右偏离了一点。

切分点没有刚好落在中点，会导致树的深度增加，构建、搜索时间延长。

```
# determine whether to split into left and right
if len(point_indices) > leaf_size:
    # --- get the split position ---
    point_indices_sorted, _ = sort_key_by_vale(point_indices, db[point_indices, axis])  # M
    middle_left_idx = math.ceil(point_indices_sorted.shape[0] / 2) - 1
    middle_left_point_idx = point_indices_sorted[middle_left_idx]
    middle_left_point_value = db[middle_left_point_idx, axis]

    middle_right_idx = middle_left_idx + 1
    middle_right_point_idx = point_indices_sorted[middle_right_idx]
    middle_right_point_value = db[middle_right_point_idx, axis]

    root.value = (middle_left_point_value + middle_right_point_value) * 0.5
    # === get the split position ===
    root.left = kdtree_recursive_build(root.left,
                                       db,
                                       point_indices_sorted[0:middle_right_idx],
                                       axis_round_robin(axis, dim=db.shape[1]),
                                       leaf_size)
    root.right = kdtree_recursive_build(root.right,
                                        db,
                                        point_indices_sorted[middle_right_idx:],
                                        axis_round_robin(axis, dim=db.shape[1]),
                                        leaf_size)
return root
```

题目三

深蓝学院 shenlanxueyuan.com

- 作业1：kd树

增加leaf_size
可以规避切分点不在正中央的影响，但会增加底层搜索计算量，降低查询效率

```python
def kdtree_knn_search(root: Node, db: np.ndarray, result_set: KNNResultSet, query: np.ndarray):
    if root is None:
        return False

    if root.is_leaf():
        # compare the contents of a leaf
        leaf_points = db[root.point_indices, :]
        diff = np.linalg.norm(np.expand_dims(query, 0) - leaf_points, axis=1)
        for i in range(diff.shape[0]):
            result_set.add_point(diff[i], root.point_indices[i])
        return False

    if query[root.axis] <= root.value:
        kdtree_knn_search(root.left, db, result_set, query)
        if math.fabs(query[root.axis] - root.value) < result_set.worstDist():
            kdtree_knn_search(root.right, db, result_set, query)
    else:
        kdtree_knn_search(root.right, db, result_set, query)
        if math.fabs(query[root.axis] - root.value) < result_set.worstDist():
            kdtree_knn_search(root.left, db, result_set, query)

    return False
```

题目三

深蓝学院 shenlanxueyuan.com

- 作业3：octree构建

如果中心取均值，
点分布不均，后续可能越界。

Github的代码fix了这个问题。
有的同学注意到了，在作业中注明出来；有的同学忽略了这里。

```python
# 功能：构建octree，即通过调用octree_recursive_build函数实现对外接口
# 输入：
#     dp_np：原始数据
#     leaf_size：scale
#     min_extent：最小划分区间
def octree_construction(db_np, leaf_size, min_extent):
    N, dim = db_np.shape[0], db_np.shape[1]
    db_np_min = np.amin(db_np, axis=0)
    db_np_max = np.amax(db_np, axis=0)
    db_extent = np.max(db_np_max - db_np_min) * 0.5
    db_center = np.mean(db_np, axis=0)

    root = None
    root = octree_recursive_build(root, db_np, db_center, db_extent, list(
                                  leaf_size, min_extent)

    return root
```

```python
    db_center = db_np_min + db_extent
```

题目三

深蓝学院 shenlanxueyuan.com

- 作业3：octree搜索 && 构建

```
if len(point_indices) <= leaf_size or extent <= min_extent:
    root.is_leaf = True
else:
    root.is_leaf = False
    children_point_indices = [[] for i in range(8)]
    for point_idx in point_indices:
        point_db = db[point_idx]
        morton_code = 0
        if point_db[0] > center[0]:
            morton_code = morton_code | 1
        if point_db[1] > center[1]:
            morton_code = morton_code | 2
        if point_db[2] > center[2]:
            morton_code = morton_code | 4
        children_point_indices[morton_code].append(point_idx)
```

优化方式：

```
# 作业4
# 屏蔽开始
quadrant = 0
root.is_leaf = False
cpvector = db[point_indices] - center
# print (cpvector.shape)
quadrant_mask = np.asarray([4, 2, 1])
quadrant_index = (cpvector > 0 ).astype('int')*quadrant_mask
quadsum = np.sum(quadrant_index, axis = 1)

octpart_point_indices = [point_indices[quadsum == i] for i in range(8)]
# print (octpart_point_indices[0].shape)
```

Mask = [2,1]

[0,0]
* [1,1]
[0,1]
[1,0]

求和

[0]
[3]
[1]
[2]

构建耗时会缩短

C++实现

- 部分同学用C++实现了kdtree
- C++效率与优化等级有关，有的同学没有开启相关选项

```
set(CMAKE_BUILD_TYPE "Release")
set(CMAKE_CXX_FLAGS "-std=c++11 -o3 -Wall")
```

	Kd tree构建	Kd tree搜索
A	6.5ms	0.2ms
B	13ms	0.0045ms
C	10ms	0.004ms
scipy	160ms	0.7ms
Cpp PCL 1.7	2.2ms	0.003ms

开启相关选项，选取10000个点建树，leaf_size为1,对比实验。

下面请实现效果最好的A同学分享下经验。

在线问答

深蓝学院
shenlanxueyuan.com

Q&A

深蓝学院
shenlanxueyuan.com

感谢各位聆听

Thanks for Listening