Final Report

Final Report

Department of Computer Science and Engineering

BY

JiaLin Li

HaoTian Gao

January 2021

Abstract

After a semester for octree implementation, we have realized three algorithms: KNN algorithm, radius search algorithm, point cloud compression algorithm. First, we will have a review on all of work we have done. The second part is the achievement after mid-term, including: point cloud compression realization and comparison of radius search between octree and kdtree.

1 Review of work in this semester

1.1 Our original goal

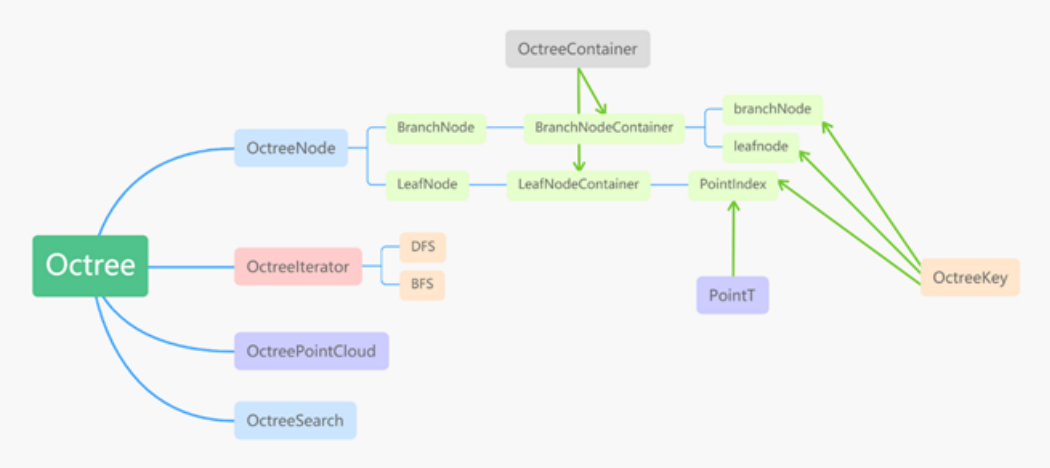
1. Realize the base class of Octree
2. Implement three main algorithms of octree:
   Pointcloud compression algorithm

   KNN search, Radius search algorithm
   Spatial change detection algorithm
3. Optimize the base class and algorithms to merge them into OpenCV library.

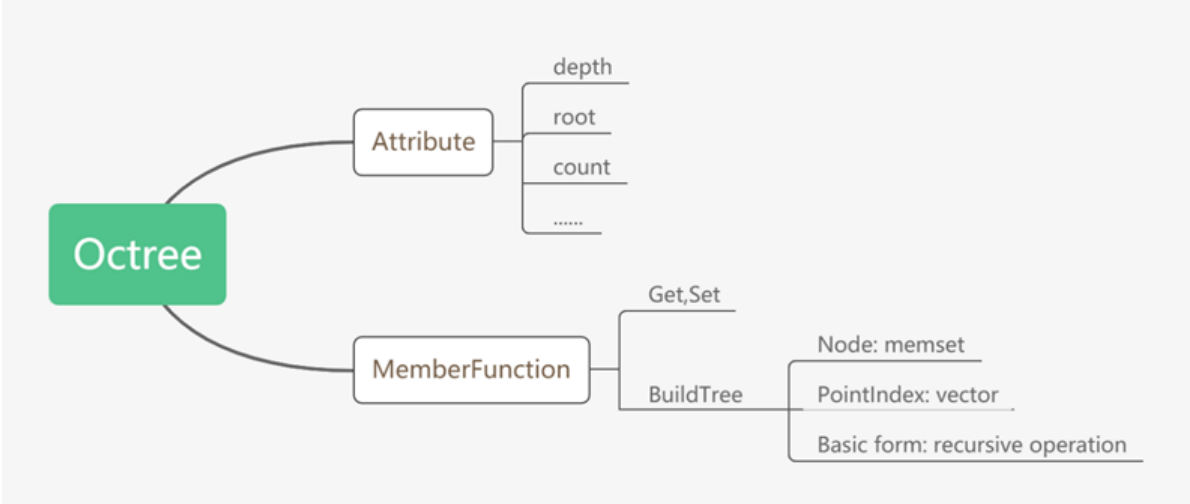
1.2 Progress

1.2.1 Analyze on the whole PCL octree class

It is too hard for us to realize the task from zero. So we study the structure of PCL octree first.



And we implement a simple octree class like structure below:

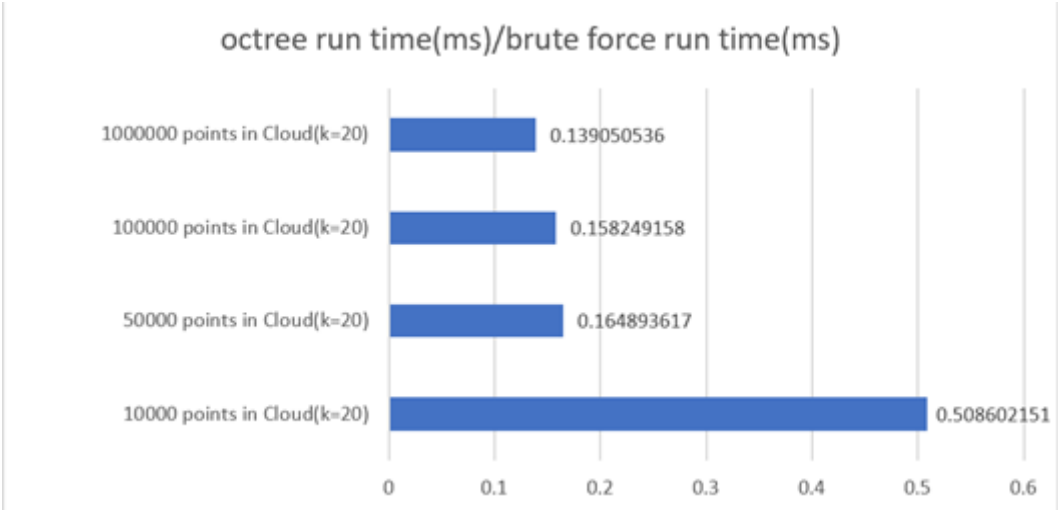


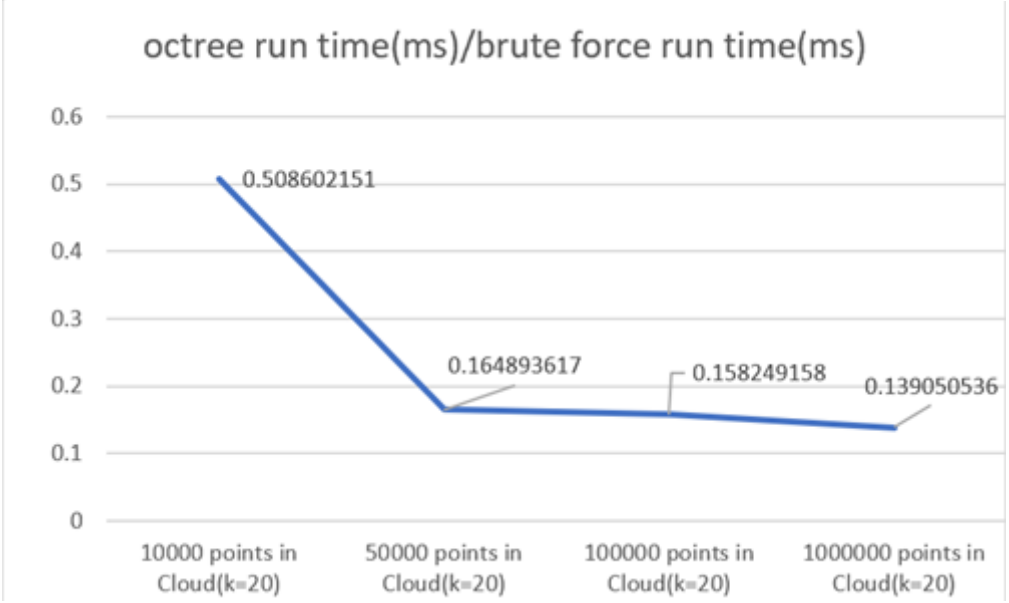
1.2.2 Realize KNN search algorithm

Below is our pseudo code for KNN search algorithm:

```
FindKNN(priority_ queue Q, Point P,  OcTreeNode *root,  Result Res, int K)
    first add root to Q
    Min_ distance <-- +Inf
    while(Q is not empty ):
        OCTreeNode T=Q. pop
        if Min_ distance<T. distance
            return res
        if T is leaf:
            if Min. distance<distance(P, T) and res.size > K:
                Min_ distance=distance(P,T)
                put T into Res
        else if T is Branch:
            for child in T:
                calculate distance from child to P
                put child into Q
    return ans
```

We realize this algorithm and get a result as below:





1.2.3 Realize Radius search algorithm

Below is our pseudo code for Radius search algorithm:

```
Improved radiusNeighbors(Octant O, query point q, radius r, result set R)
   Input: Octant O, query point q, radius r, result set R
   Result: R contains all radius neighbors N(q, r)
   if O is inside S(q,r) then
       Add all points inside O to R
       return
   end
   if O is leaf octant then
      foreach point p inside O do
           AddptoRif|p-q||<rholds
      end
      return
  end
  foreach child octant C of O do
     if S(q, r) overlaps C:
        radiusNeighbors(C, q, r, R)
  end
```

Reference from [1]

2 New Progress

Since last presentation, we do a lot of work on implementation of Pointcloud Compression and experiments of Radius Search.

For PointCloud compression, we basically achieve a result of 50% compression rate. For Raidus Search, we verify the high efficiency of octree compared to kdtree.

2.1 Implementation of Pointcloud Compression

2.1.1 Point Cloud Compression Synoptic

For limited channel transferring, the efficacy and the accuracy are the most important part for transferring the point cloud information. Assuring the high accuracy, the efficacy will be the worthiest part for improving the transferring. Point cloud compression will compress the structure of the octree to decrease the size of point cloud information and then improve the transferring efficacy.

For compression encoding and decoding, their aims are serialize octree and deserialize octree. Each node has one branch bit pattern, the branch bit pattern has 8 digits corresponding to 8 child nodes. If the child node is none, then the corresponding digit is 0. Otherwise, the digit is 1. Encoding is using DFS to traverse the octree and get the branch bit pattern into the binary tree argument which is an array to store the bit patterns and decoding is using DFS to with binary tree argument to reconstruct the octree. For the leaf node, the points will be stored into the point data vector argument and encoding and decoding will use it to reload the information of the points.

2.1.2 Implementation

After knowing the algorithm of point cloud compression, we try to realize it. We use python 3.7 to realize the compression. Initially, we use array to store the binary tree argument and the point information(point data vector argument array). However, the compression rate is not ideal. Then we convert the node index information(binary tree argument) into bitstream. In this time, we convert the array storing to bit storing. Then the compression rate decrease about 10%. Finally, we encode the point data vector argument array to string. This operation decreases about 40% space in contrast with the previous version.

2.1.3 Pseudo-code

compression algorithm:

```
Algorithm: Point Cloud Compression
Input: octree T
Output: compression file F
Static array: bitPatternArray, pointInformation
Function getBitPattern(D):
    bitArray ← new Array
    for each child in D.children:
        if child is not none:
            bitArray.add(1)
        else:
            bitArray.add(0)
        End if
    End for
    bitPattern ← convert bitArray to bits
    return bitPattern
End Function

Function serializeTree(node):
    for each D in node.children:
        bitPatternArray.add(getBitPattern(D))
        if D is branchNode:
            serializeTree(D)
        else:
            Store the point information into the pointInformation
        End if
    End for
End Function

Function PCLCompression(T):
    Root R ← T.root
    serializeTree(R)
    Encode bitPatternArray as bit_pattern and pointInformation as points into
the file F
    return F
End Function
```

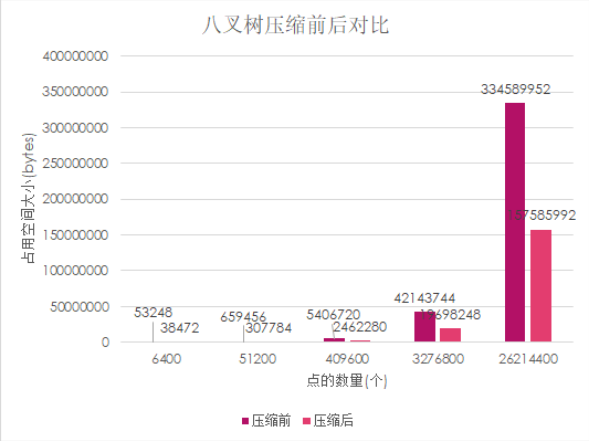
decompression algorithm:

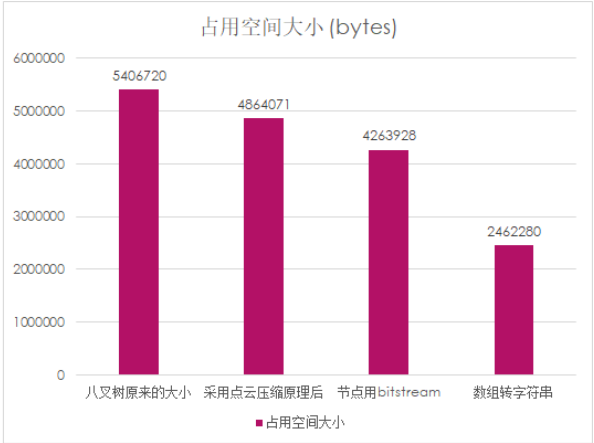
```
Algorithm: Point Cloud Decompression
Input: file F
Output: root node R
Function PCLDecompression(F):
```

```
    bitPattern ← readFlie(F).bit_pattern
    pointInformation ← readFlie(F).points
    Root ← new Node
    deserializeTree(Root)
    return Root
End Function

Function deserializeTree(node):
    Bits ← bitPattern.firstElement
    bitPattern.removeFirstElement()
    IsLeaf ← True
    for each bit in Bits:
        if bit != 0:
            IsLeaf ← False
            newNode ← new Node
            node.children[bit.index] ← newNode
            deserializeTree(newNode)
        else:
            node.children[bit.index] ← none
        End if
    End for
    if IsLeaf:
        points ← pointInformation.firstElement
        pointInformation.removeFirstElement()
        Store the points into the node
    End if
End Function
```

2.1.4 Results





For the first chart, it is overt that the compressed space is about a half of the octree structure. The compression rate is about 50%. For the second chart, the results are the effect for adding the functions for point cloud compression. The initial octree size is 5406720 bytes. With the point cloud compression algorithm utility, the compressed size is down to the 4864071 bytes and the compression rate is about 90%. Then we convert the node information into the bitstream and then the compressed result is down to 4263928 bytes. The compression rate is about 80%. Finally, we encode the point data vector argument into string and the compressed result is 2462280 bytes.

2.1.5 Analysis

Considering the implemented compression, binary tree argument could be the simplest compression method, because one byte corresponds to one node children information (one bit corresponds to one child). For point data vector argument array, we just encode the array to string but not bytes (binary tree argument is the array of 0 and 1, but point data vector argument is the array of three dimension vector). Moreover, it is obvious that the conversion from point data vector argument to the string decrease the space about 40%. Decreasing the space for storing the point data vector could be the most promising direction. Therefore, we can attempt to design an algorithm to convert the 3d vector to bytes for better compression rate in the further study.

2.2 Verify of high efficiency of octree

2.2.1 Experiment Process

for radius search

compare with k-d tree(another data structure used in three dimension) and brute force

radius search with radius R and point number N

Q for the number of points in result set

M for the side length of the root node

for brute force

$$O（N）$$

for k-d tree

$$O(N^{\frac{k-1}{k}}+Q)$$

recall for octree

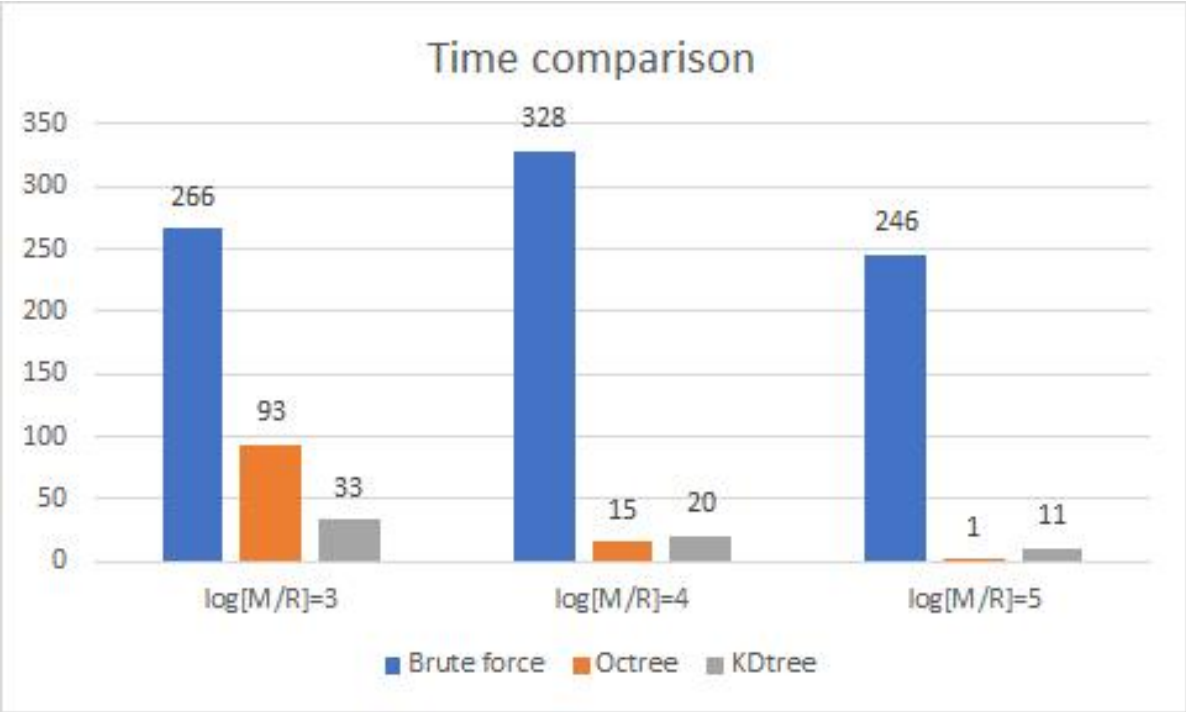
$$O\left(\frac{N}{8^{log\lfloor\frac{M}{R}\rfloor}}\right)$$

using -O3 in c++ to optimize

some advantages for O3:

1. An attempt was made to merge the same constants
2. Optimizing the handling of conditional and unconditional branches in assembly code
3. Optimization of loop generation in assembly language
4. Reduce or delete conditional branches

2.2.2 Results



Since the ratio of radius r to initial radius m is more than 10, that is **log[M/R] > 3**, the experiment starts with **log[M/R] = 3**

As shown in the result graph, when **log [M / R] > = 4**, the search effect of octree is better than kdtree, which is also an important reason for the existence of octree.

3 Future Plan

There are still a lot of improvement for our octree construction. First, for pcl compression part, we will attempt to improve the compression for points information. A better algorithm for three dimension point vector compression will be a promising direction for the more advanced compression. Second, as we have implemented the three algorithms in the tutorial, the next goal for algorithm is to realize the down sampling, which is not demonstrated in the tutorial. Down sampling is the one of the most important part for extracting feature points. Therefore, it is important to realize the down sampling for future construction. Third, we plan to modularize our octree code. As we have just implemented these algorithm now, the design pattern hasn't been hierarchical. The improvement for our code includes extracting methods, extracting class, adding annotations, organize the code structure and better the hierarchy.

In conclusion, there is still much space for improvement of our octree in three aspects: perfect algorithm, realize more algorithm and improve the design pattern.

4 Reference

[1] Castro, Rener, et al. "Statistical Optimization of Octree Searches." ***Wiley Online Library*****, John Wiley & Sons, Ltd, 4 Jan. 2008, from https://onlinelibrary.wiley.com/doi/full/10.1111/j.1467-8659.2007.01104.x**

[2] Ruwen Schnabel Institut für Informatik II, Schnabel, R., II, I., Reinhard Klein Institut für Informatik II, Klein, R., Zürich, E., . . . Authors: Ruwen Schnabel Institut für Informatik II. (2006, July 01). Octree-based point-cloud compression. Retrieved January 07, 2021, from https://dl.acm.org/doi/10.5555/2386388.2386404