Programming Assignments 3 and 4 – 601.455/655 Fall 2018

Score Sheet (hand in with report) Also, PLEASE INDICATE WHETHER YOU ARE IN 601.455 or 601.655

Name 1	*Tianyu Song*
Email	*tsong11@jhu.edu*
Other contact information (optional)	
Name 2	*Huixiang Li*
Email	*lhuixia1@jhu.edu*
Other contact information (optional)	
Signature (required)	I (we) have followed the rules in completing this assignment Tianyu Song Huixiang Li

Grade Factor		
Program (40)		
Design and overall program structure	20	
Reusability and modularity	10	
Clarity of documentation and programming	10	
Results (20)		
Correctness and completeness	20	
Report (40)		
Description of formulation and algorithmic approach	15	
Overview of program	10	
Discussion of validation approach	5	
Discussion of results	10	
TOTAL	100	

a) Overall Algorithm & Iteration Logic

Step 0: Read inputs, point clouds and meshes

Step 1: calculate mesh point cloud M, tip point cloud d

Step 2: initialize the $F_{reg} = I_4$

Step 3: Calculate $s = F_{reg} * d$

Step 4: Find the closest point, $c = FindClosestPoint(d, M)$

Step 5: Find the transformation $F_t = PointCloudRegistration(s, c)$

Step 6: Calculate $F_{reg} = F_t * F_{reg}$

Step 7: While not converge: iterate Step2~Step6; else: finish iteration and go to Step 8

Step 8: Calculate distance between s and c, output result to .txt file

b) Methods and Theory

1. Bounding Sphere

To find the closest triangle to a given point in a three-dimensional space, we construct bounding spheres around each triangle, and use this to reduce the number of careful checks required while searching all the triangles. For each triangle, there are three points a, b and c . To find the bounding sphere of this triangle, we must find its center and radius.

Let edge (a,b) is the longest side of triangle

Step 1: Given three points a, b and c, compute

$$f = (a + b)/2$$

Step 2: define

$$u = a - f, \quad v = c - f, \quad d = (u \times x) \times u$$

Step 3: Calculate the sphere center q lines along the line

$$q = f + \lambda d$$

$$(\lambda d - v)^2 \le (\lambda d - u)^2 \quad \Rightarrow \quad \lambda \ge \frac{v^2 - u^2}{2d \bullet (v - u)} = \gamma$$

If $\gamma \le 0$, then $\lambda = 0$. Else $\lambda = \gamma$

2. FindClosestPoint(a,[p,q,r])

Step 1: solve the least squares problem for λ and μ

$$a - p \approx \lambda(q - p) + \mu(r - q)$$

Step 2: compute c

$$c = p + \lambda(q - p) + \mu(r - p)$$

Step 3: if $\lambda \ge 0$, $\mu \ge 0$, and $\lambda + \mu \le 1$

c lies in the triangle

c $\Rightarrow$ the closest point

Step 4: if not satisfy the condition ($\lambda \ge 0$, $\mu \ge 0$, and $\lambda + \mu \le 1$)

The closest point is on the border of the triangle

3. Find Closest Point on Triangle

Step 1: if $\lambda < 0$

Closest point = ProjectionOnSegment(c,r,p)

Step 2: if $\mu < 0$

Closest point = ProjectionOnSegment(c,p,q)

Step 1: if $\lambda + \mu > 1$

Closest point = ProjectionOnSegment(c,q,r)

4. ProjectionOnSegment(c,p,q)

Step 1: calculate λ

$$\lambda = \frac{(c-p)\cdot(q-p)}{(q-p)\cdot(q-p)}$$

Step 2: find λ^*

$$\lambda^* = max(0, min(\lambda, 1))$$

Step 3: find c^*

$$c^* = p + \lambda^*(q - p)$$

5. Simple Search with Bounding Spheres

Step 1: assume triangle i has corners[p,q,r]

Step 2: surrounding sphere i has radius ρ center q, and let bound equal to infinity

Step 3: compute for-loop

```
for i = 1 to N {
    if |q − a| − ρ ≤ bound {
        h = FindClosestPoint(a, [p, q, r]);
            if |h − a| < bound {
                c = h;
                bound = |h − a|;
            }
        }
    }
```

c) Description of Functions

functions	**input variable**	**output variable**
ConstructBoundingSphere.py	***v*** **xyz coordinates of vertices in CT coordinates(N*3 matrix),** ***tri*** **vertex indices of the three vertices for each triangle(M*3 matrix)**	**a list of sphere class with properties: center, radius and object**
Calculate the bounding sphere of the given triangle		
FindClosestPoint.py	***a*****, tri vertex indices of the three vertices for each triangle**	***c*** **the closest point calculated**
Find the closest point with a given triangle		
ProjectionOnSegment.py	***c,p,q*** **three points of the triangle**	***c_s*** **projection on segment c***
apply the given three inputs to calculate the projection on segment c*		
tryICP.py	***Nsample*****, number of samples;** ***d,*** **calculated point could of the tip;** ***spheres*****, a list of class;** ***octree*****, object;**	***c, s*** **the point cloud calculated**
apply the ICP iterative algorithm to update ***F_reg*****, then calculate** ***s=F_reg*d*****, and** ***c*****.**		
ProgrammingAssignment4.py	**/**	**./OUTPUT/*.txt**
automatically import input files in the data directory and output results for PA4 as *.txt files for each test set		

d) Description of Class

Sphere:

```
class Sphere(object):
    def __init__(self):
        self.Center = np.zeros((3, 1))
        self.Radius = 0
        self.Object = np.zeros((3, 3))
```

The Sphere class initialize properties of the bounding sphere. The introduction of this class can make the program more readable and easier to maintain.

BoundingBoxTreeNode:

```
# Empty Octree Class


class trees:

    def __init__(self):
        pass

# Class of Bounding box octree


class BoundingBoxTreeNode:

    # Initialize properties
    def __init__(self, BS, ns): ...

    # Construct subtrees to the BoundingBoxTreeNode object passed in
    def constructSubTrees(self): ...

    # Sort the spheres in current node according to their spatial position
    def SplitSort(self): ...

    # Find closest point in current node
    def FindCP(self, v, bound, closest): ...

    # Update the closest point in the node that do not have subtrees
    def UpdateClosest(S, v, bound, closest): ...
```

This class defines a bounding box tree node in an octree. All the bounding spheres are divided into subregions by the spatial position of their centers until the bounding box that bounds all the sphere are small enough or the number of spheres bounded reaches certain minimum value.

Each property in this class is described in the above image. Details of the use of each method are explained below:

```
# Initialize properties
def __init__(self, BS, ns): ...
```

Initialize each properties of the object and define the data type of property *Subtree*. Call the method *constructSubTrees* at the end.

```
# Construct subtrees to the BoundingBoxTreeNode object passed in
def constructSubTrees(self): ...
```

This method is called at the end of the constructor. It is to construct sub-trees of the current node unless it is small enough in dimension or the number of spheres it contains reaches certain minimum value in which case the iteration constructing sin-trees stops. (In our program, minimum diagonal = 8 and minimum count of spheres = 8. How these values are determined will be discussed later in this report). If the current bounding box node is large enough and contains more than minimum number of spheres, the space of current node will be divided by the property *Center* into eight pieces, and spheres in current node are sorted according to the spatial position of the center of these spheres while the number of spheres in each subspace is recorded at the same time. After sorting the spheres in current node, *SubTrees* including 1*8 *BoundingBoxTreeNode* is constructed accordingly

```
# Sort the spheres in current node according to their spatial position
def SplitSort(self): ...
```

Divide the space of current node into 8 sub-spaces according to property *Center*. Sort the spheres in current node according to the spatial position of the center of each sphere.

```
# Find closest point in current node
def FindCP(self, v, bound, closest): ...
```

Keep finding the closest point recursively in sub-trees until it reaches the end of the tree, when the method *UpdateClosest* will be called to actually calculating the closest point to *v*on the *Object*(triangle) of the spheres in the node. At the beginning of this method, a checking is done to test whether current node is worth further searching.

```
# Update the closest point in the node that do not have subtrees
def UpdateClosest(S, v, bound, closest): ...
```

This method is called when the node reaches the end of the octree. Simple search approach same as the one used in the previous assignment is implemented where bounding sphere is used to reduce the number of careful checks required in current node. The new closest point and corresponding bound is updated once closer point is found.

e) Results of Functions

For the dataset ABCDEF we show the magnitude of difference between d_k and c_k and compare our output and given output. Full result can be found in the OUTPUT folder.

Data	Mean of distance [given output]	Mean of distance [our output]	Mean of difference between our output and debug output
A	**0.000000**	**0.001693**	**0.001693**
B	**0.000000**	**0.001880**	**0.001880**
C	**0.000000**	**0.001945**	**0.001945**
D	**0.000000**	**0.003325**	**0.003325**
E	**0.060975**	**0.060700**	**-0.000275**
F	**0.073410**	**0.073345**	**-0.000065**

Data	Mean of distance [our output]	Standard deviation of our output
G	**0.003685**	**0.002951**
H	**0.002990**	**0.002458**
J	**0.075985**	**0.061526**
K	**0.072425**	**0.060483**

f) Discussion of the Results and Analysis

Generally speaking, the result is still good enough as the differences are mostly within 0.0035. For group A to D, the distance between point cloud ***c*** and ***s*** in given debug output files are close to zeros, which means that the point cloud ***d*** can match pretty nicely with points on the mesh.

However, although the error term in our ICP method has started to converge, the point s still deviates relatively significantly from the closest point found on the mesh (around 0.0018 on average). This difference may be caused by different method of point cloud to point cloud subroutine applied in each iteration.

Comparing to our original ICP algorithm without speedup method, the octree method we implemented did not significantly increase the calculation speed. But with the octree method, it takes about 500 seconds to compute all groups of data, while 518 seconds for without octree speedup method. This might because our implementation of octree data structure is not efficient.

In conclusion, our program works well, as one can observe from the comparison between the given debug output file and ours. The calculation errors are in reasonable range. Our program can achieve all the goals of this assignment with relatively good performance.

g) Work Distribution

Name	Work
Tianyu Song	● Collaborated with Huixiang on building the octree class, sphere class. ● Finished ConstructBoundingSphere.py and BoundingSphere.py ● Tested and debugged programs
Huixiang Li	● Collaborated with Tianyu on building the overall program structure and icp iteration algorithm ● Finished ProgrammingAssignment4.py and tryICP. ● Tested and debugged programs

References

[1] Find Closest Point from Dense Cloud (P4-P8 in lecture slide ‘Finding point-pairs’)
[2] Octree and ICP iteration (P10-P29 in lecture slide ‘Finding point-pairs’)