


Programming Assignments 3 and 4 – 601.455/655 Fall 2021

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

(one in each section is OK)

Name 1	Shuojue Yang
Email	syang131@jh. edu
Other contact information (optional)	
Name 2	Zijian Wu
Email	zwu52@jhu. edu
Other contact information (optional)	
Signature (required)	<p>I (we) have followed the rules in completing this assignment</p> <p style="text-align: center;">  Yang  </p>

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	

CIS I - Programming Assignment 3

Zijian Wu and Shoujue Yang

1 Method Overview

In this programming assignment, our task is to find the closest points in the model surface corresponding to each point in the point cloud. We respectively implemented Brutal Search, Bounding Sphere Search, KDtree Search and Octree Search. We also compared their performance in boosting the efficiency.

For this assignment's scenario, we use 2 rigid bodies, which can be detected by the EM tracker, to obtain the point cloud from intraoperative reality. The 3D surface model is obtained from CT system. The workflow is as following steps:

1. Obtaining the point cloud in intraoperative reality

We have 2 rigid bodies, A and B. The A rigid body is used as a pointer. I.e., its tip is placed into contact with a number of points on the surface of the bone. The “tip” of the B rigid body is rigidly screwed into the bone in some unknown orientation. The point cloud $\{\vec{d}_k\}$ can be calculate according to following formula:

$$\vec{d}_k = F_{B,k}^{-1} \cdot F_{A,k} \cdot \vec{A}_{tip}. \quad (1)$$

2. Find the closest point \vec{c}_k in a triangle mesh

We find a point in the triangle so that the distance from this point to a point in the point cloud is the closest. This distance is defined as the distance from a point in the point cloud to a triangle mesh. For each \vec{d}_k , we can obtain a point set $\{\vec{c}_{k,i}\}$, where i is the index of triangle mesh.

3. Find the closest point pairs $\{\vec{d}_k, \vec{c}_{k,i}^{nearest}\}$

For each point in $\{\vec{d}_k\}$, finding the closest triangle mesh is equivalent to find the nearest point $\vec{c}_{k,i}$. By utilizing Brute-Force search or other smarter approaches, we search the $\{\vec{c}_{k,i}\}$ space to find the $\vec{c}_{k,i}^{nearest}$ (the closest triangle mesh) corresponding to \vec{d}_k . After traversing the whole point cloud, we get the point pairs $\{\vec{d}_k, \vec{c}_{k,i}^{nearest}\}$.

1.1 Find the closest point in triangle

According to the class note, we calculate the closest point $\vec{c}_{k,i}$ as following steps:

1. For the system shown in Fig.1, the following equation holds:

$$\vec{a} - \vec{p} \approx \lambda(\vec{q} - \vec{p}) + \mu(\vec{r} - \vec{p}) \quad (2)$$

This equation can be rewritten as a Least Squares form:

$$\begin{bmatrix} \vec{q} - \vec{p} & \vec{r} - \vec{p} \end{bmatrix} \cdot \begin{bmatrix} \lambda \\ \mu \end{bmatrix} \approx \vec{a} - \vec{p} \quad (3)$$

Thus we can obtain λ and μ through solving Least Squares question.

2. According to the geometric relationship shown in Figure 1, we get the following formula:

$$\vec{c} = \vec{p} + \lambda(\vec{q} - \vec{p}) + \mu(\vec{r} - \vec{p}) \quad (4)$$

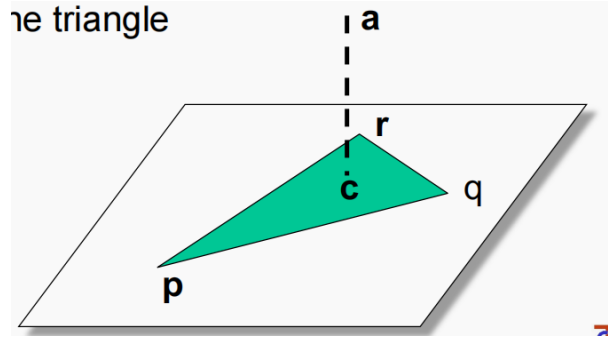


Figure 1: The calculation method of closest point \vec{c}^* when \vec{c} is the exterior of the triangle

If $\lambda \geq 0$, $\mu \geq 0$, $\lambda + \mu \leq 1$, then \vec{c} lies within the triangle and is the closest point. Otherwise, we need to find a point on the border of the triangle. In this case, we define \vec{c}^* , \vec{c} 's project on the border of the triangle, as the closest point. The calculation method of \vec{c}^* is shown in Figure 2 and Figure 3.

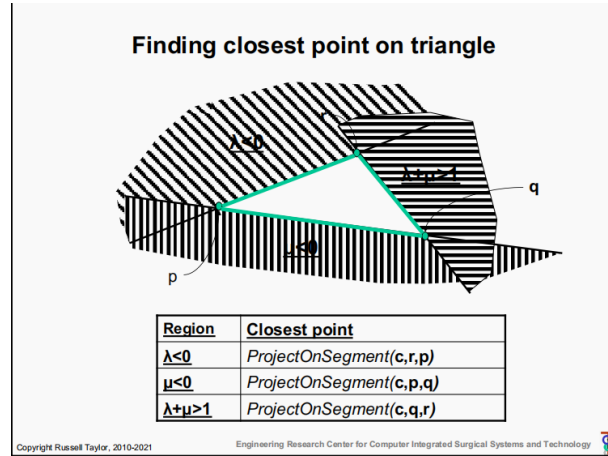


Figure 2: The calculation method of closest point \vec{c}^* when \vec{c} is the exterior of the triangle

3. For each \vec{d}_k , we find closest point in each triangle mesh according to above procedure and get point set $\{\vec{c}_{k,i}\}$.

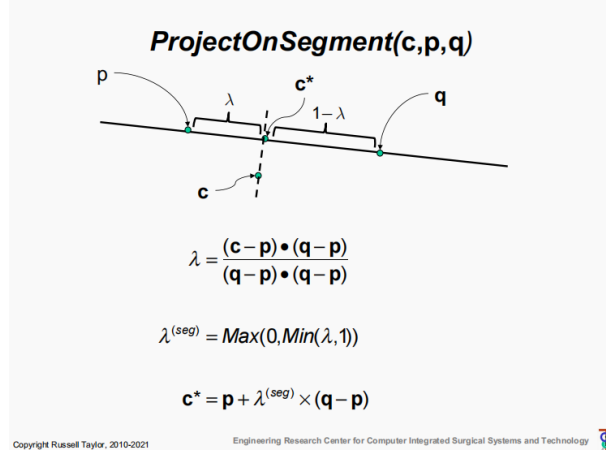


Figure 3: The function $\{ProjectOnSegment(c, p, q)\}$ used in calculation method of closest point \vec{c}^*

1.2 Find the closest triangle

For each point in $\{\vec{d}_k\}$, finding the closest triangle mesh is equivalent to find the nearest point $\vec{c}_{k,i}$. There are many ways to find the closest triangle. In this assignment, we respectively adopted Brute-Force Search, Simple Search with Bounding Spheres, Search Based on KDtree, and Search Based on Octree to find the closest $\vec{c}_{k,i}$, namely $\vec{c}_{k,i}^{nearest}$.

1.2.1 Brute-Force Search

Construct linear list of triangles and search sequentially for closest triangle to each point \vec{d}_k . For each point \vec{d}_k , Brute-Force Search will compute the distance from \vec{d}_k to every corresponding $\vec{c}_{k,i}$ and find the closest $\vec{c}_{k,i}^{nearest}$.

1.2.2 Simple Search with Bounding Spheres

Construct bounding spheres around each triangle and use these spheres to reduce the number of careful checks required. The specific procedure is shown as following:

1. Find bounding sphere for each mesh triangle The Schematic diagram of a triangle's bounding sphere is shown in the Figure 4.

Assume edge $\vec{a} - \vec{b}$ is the longest. Then the center \vec{q} of the sphere will obey

$$(\vec{b} - \vec{q})(\vec{b} - \vec{q}) = (\vec{a} - \vec{q})(\vec{a} - \vec{q}) \quad (5)$$

$$(\vec{c} - \vec{q})(\vec{c} - \vec{q}) \leq (\vec{a} - \vec{q})(\vec{a} - \vec{q}) \quad (6)$$

$$(\vec{b} - \vec{a}) \times (\vec{c} - \vec{a}) \cdot (\vec{q} - \vec{a}) = 0 \quad (7)$$

The calculation method of the radius and center is as following steps:

- (1) Compute

$$\vec{f} = \frac{(\vec{a} + \vec{b})}{2} \quad (8)$$

- (2) Define

$$\vec{u} = \vec{a} - \vec{f}; \vec{v} = \vec{c} - \vec{f}; \vec{d} = (\vec{u} \times \vec{v}) \times \vec{u} \quad (9)$$

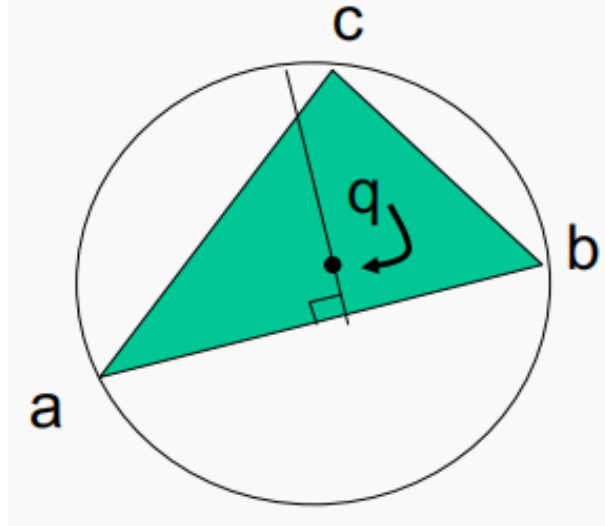


Figure 4: Schematic diagram of a triangle's bounding sphere

(3) Then the sphere center \vec{q} lies somewhere along the line

$$\vec{q} = \vec{f} + \lambda \vec{d} \quad (10)$$

with $(\lambda \vec{d} - \vec{v})^2 \leq (\lambda \vec{d} - \vec{u})^2$. Simplifying gives us

$$\lambda \geq \frac{\frac{\vec{v}^2}{v} - \frac{\vec{u}^2}{u}}{2d(\frac{\vec{v}}{v} - \frac{\vec{u}}{u})} = \gamma \quad (11)$$

If $\gamma \leq 0$, then just pick $\lambda \leq 0$. Else pick $\lambda = \gamma$.

2. Simple Search with Bounding Bpheres The pseudo code of the Simple Search with Bounding Spheres algorithm is shown in Figure 5.

Simple Search with Bounding Spheres

```
// Triangle i has corners [ $\vec{p}_i, \vec{q}_i, \vec{r}_i$ ]
// Surrounding sphere i has radius  $\rho_i$  center  $\vec{q}_i$ 
bound =  $\infty$ ;
for i=1 to N do
{ if  $\|\vec{q}_i - \vec{a}\| - \rho_i \leq bound$  then
  {  $\vec{h} = \text{FindClosestPoint}(\vec{a}, [\vec{p}_i, \vec{q}_i, \vec{r}_i])$ ;
    if  $\|\vec{h} - \vec{a}\| < bound$  then
      {  $\vec{c} = \vec{h}$ ; bound =  $\|\vec{h} - \vec{a}\|$ ; };
  };
};
```

Copyright Russell Taylor, 2010-2021 Engineering Research Center for Computer Integrated Surgical Systems and Technology

Figure 5: Simple Search with Bounding Spheres

1.2.3 Search Based on KDtree

The Search Based on KDtree algorithm is an improvement on Brute-Force Search. Same as Brute-Force Search, we first compute every $\vec{c}_{k,i}$ corresponding to \vec{d}_k . Then we built a KDtree on the $\{\vec{c}_{k,i}\}$, and search for the point $\vec{c}_{k,i}$ in KDtree closest to corresponding \vec{d}_k .

1.2.4 Search Based on Octree

The algorithm for Octree Search can be split into two main components: The tree construction and the closest point detection. In accordance with the given slides, we have implemented a Python based class. *BoundingBaseNode(Spheres, nSpheres)* recursively constructs the tree and member method *FindClosestPoint(v)* recursively find the closest point on the surface.

Since the component test of Octree is not intuitive to represent, one can verify its correctness by comparing with the reference ground-truth.

2 Result and Discussion

2.1 Verification for important subroutine

We wrote a test program, *unit_test.py*, for the two most important subroutines *ComputeBoundingSphere(vertex)* and *ComputeClosestPoint(point, vertex)*.

2.1.1 Unit test for *ComputeBoundingSphere(vertex)*

As shown in Figure 6, the input vertices form a triangle in the x-y plane. The bounding sphere calculated by our unit test program is obviously consistent with the geometric relationship in the Figure 6.

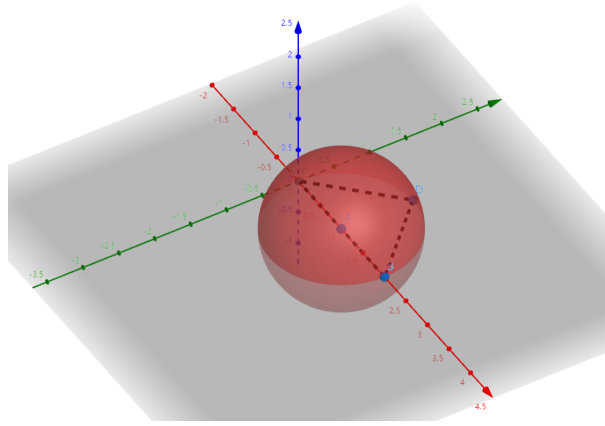


Figure 6: Simple Search with Bounding Spheres

2.1.2 Unit test for *ComputeClosestPoint(point, vertex)*

To rigorously test this function, we conduct test for 4 situation, which are \vec{c} in the inside of the triangle, \vec{c} in the exterior of the triangle, \vec{c} in the vertices of the triangle, and \vec{c} in the side of the triangle, respectively. Visualizing the results calculated by the test program can get Figure 7. From the geometric relationship, we can valid this function.

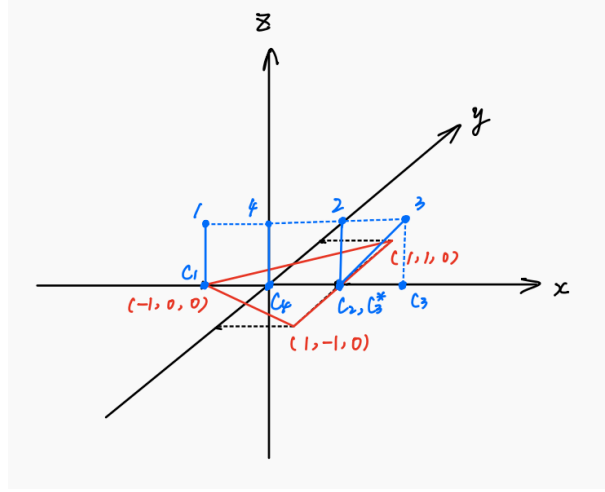


Figure 7: Simple Search with Bounding Spheres

2.2 Result of finding point pairs in different case

In this section, we first give a quantitative evaluation of various implemented methods in finding the closest points shown in Table 1. In addition, for both the Debug and Unknown cases, we compared the running time of the four made methods to demonstrate the efficiency improvement.

2.2.1 Debug case

Table 1: Quantitative evaluation of the implemented methods' accuracy. Due to the same closest point on mesh w.r.t. point, all the four methods achieve the same accuracy. The error is computed in terms of L2 Norm.

Case	d error	c error	mag error
A	0.0064	0.0053	0.0023
B	0.0064	0.0056	0.0024
C	0.0061	0.0062	0.0021
D	0.0078	0.0061	0.0029
E	0.0087	0.0075	0.0032
F	0.0082	0.0073	0.0037

All the errors in this table are computed based on L2 Normal.

According to Table 1, the error less than 0.01 means that the predicted number is exactly the same as the given reference ground-truth, which verifies the correctness of our implemented module in finding the closest point on mesh.

As shown in Table 2, 'BoundSp' denoting the Bounding Sphere Search demonstrates the highest speed in computing the paired points. Compared to naive Brutal Search, all the three advanced methods significantly improved the computation efficiency apart from the KDtree. Considering that a KDtree pipeline requires all the closest points on single triangle mesh to be found beforehand, the algorithm would repeat this step (generally requires about 0.7s in this dataset) when dealing with each point of the point cloud, which degrades the efficiency to a large degree. Furthermore, despite that Octree Search has largely reduced the running time compared with the Brutal Search, it is outperformed by Bounding Sphere Search, a simpler version to improve the efficiency. We ascribe such a inferiority to the implementation of python. During the tree construction, there would be a series of calling and assignment operations for List type, which limits the efficiency of Octree.

Table 2: Comparison of various methods in terms of running time for Debug case A-F.

Method	Running Time (s)					
	A	B	C	D	E	F
Brutal	3.08	3.00	3.04	2.95	3.11	3.14
BoundSp*	0.92	0.99	0.97	0.97	0.99	1.04
Octree	1.02	1.12	1.03	1.02	1.06	0.97
KDtree	3.04	3.08	3.00	2.96	3.10	3.07

* ‘BoundSp’ represents the Bounding Sphere Search method.

2.2.2 Unknown case

For Unknown cases, we save the results in `./output`.

We simply give the time comparison in this section. For these three cases, the Octree based Search method slightly outperforms the Bounding Sphere Search method.

Table 3: Comparison of various methods in terms of running time for Unknown case G,H,J,K.

Method	Running Time (s)		
	G	H	J
Brutal	4.03	4.02	3.93
BoundSp*	1.14	1.15	1.19
Octree	1.13	1.07	1.19
KDtree	4.06	4.13	4.12

* ‘BoundSp’ represents the Bounding Sphere Search method.

3 Contribution

Shoujue Yang is responsible for Octree Based Search algorithm and Zijian Wu is responsible for other search algorithm.