PA3-REPORT

Score Sheet

Name 1	
Email	
Other contact information (optional)	
Name 2	
Email	
Other contact information (optional)	
Signature (required)	I (we) have followed the rules in completing this assignment Jiaming Zhang Chongzhi Yang.

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	

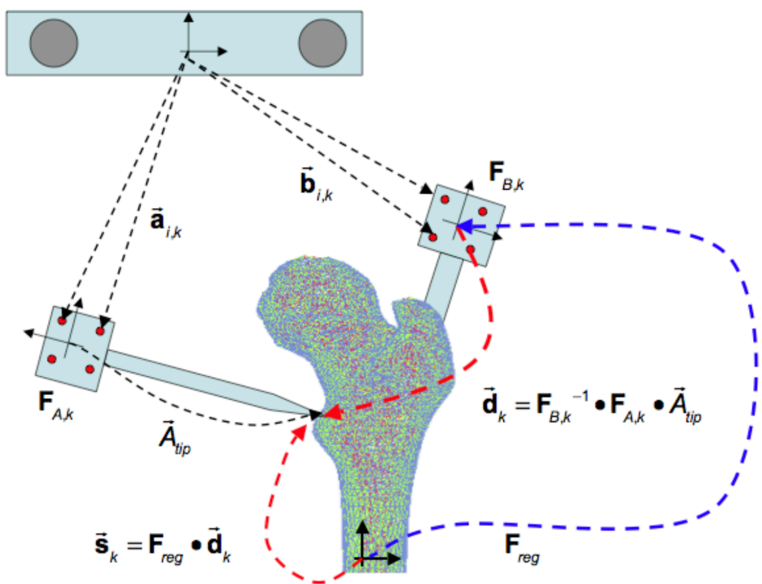
I. Mathematics & Algorithms Implementation

This section introduces the mathematical principles and implemented algorithm for locating the closest point to a given mesh file described in Programming Assignment 3.

0. Scienario

In the assignment of programming, a simplified version of Iterative Closest Point (ICP) has been developped. Our purpose is to find the closest points in the surface of the model to each point in the the point cloud. A naive search approach has been implemented, and the performance is tested based on the running time consumption. To speed up the searching process, we introduced Octree structure. Both methods have been tested and validated based on given data.

In the programming assignment, we use LED trackers to detect 2 rigid bodies and obtain the point cloud from intraoperative reality. The CT system could obtain the information of the 3D surface model.



The mathematical steps followed are described in this section.

1. Find A Tip with respect to B Frame

1) Mathematical Method

Given two fiducials with known rigid markers sticked on them, we want to find one fiducial described in the local frame of another fiducial rigid body. First, get the body definition files, recorded as $\boldsymbol{a}$ and $\boldsymbol{b}$; Second, for each data frame $\boldsymbol{k}$, we get body definition files, recorded as $\vec{A}_{i,k}$ and $\vec{B}_{i,k}$; Then, we read rigid body description file and import them as A and B. We put the rigid body A as a pointer and keep its tip contact with the points on the bone's surface. We put the tip of rigid body B screwing into the bone. We calculate the point cloud $\{\vec{d}_k\}$ according to the formula as follows.

$$\vec{d}_k = F_{B,k}^{-1} F_{A,k} \vec{A}_{tip} \tag{1}$$

where $A_k = F_{A,k}\boldsymbol{a}$, $B_k = F_{B,k}\boldsymbol{b}$ are derived from the point cloud to point cloud registration method introduce in prevoius programming assignments Ref 1.

2) Code Implementation

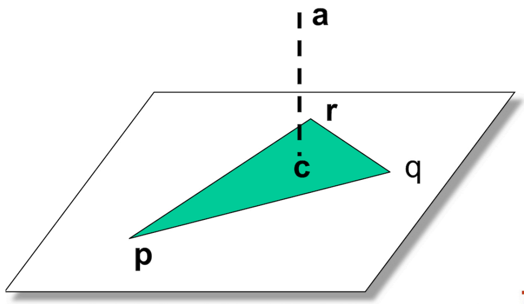
The code implemented in **"/PA3/pa3_compute_dk_test.py"**. The basic steps are:

Algorithm 1	**Compute dk**
Input:	Point Clouds: LED readings A_k and B_k , rigid body description file A and B
Output:	A_{tip} position w.r.t. B body frame, i.e. d_k
	Load A_k , B_k , A and B
	FOR each frame in $\{ A_k, B_k\}$
	CALL $F_{A,k}$ = Registration(A_k , A) $F_{B,k}$ = Registration(B_k , B)
	COMPUTE $\vec{d}_k = F_{B,k}^{-1} F_{A,k} \vec{A}_{tip}$
	ENDFOR

2. Find the Closest Point in Triangle

1) Mathematical Method

According to the notes in class, the closest point $\vec{C}_{k,i}$ would be calculated as follows



For the system as shown in the above figure, $\vec{ac}$ can be derived from

$$\vec{ac} = (\vec{a} - \vec{p}) - (\lambda(\vec{q} - \vec{p}) + \mu(\vec{r} - \vec{p})) \tag{2}$$

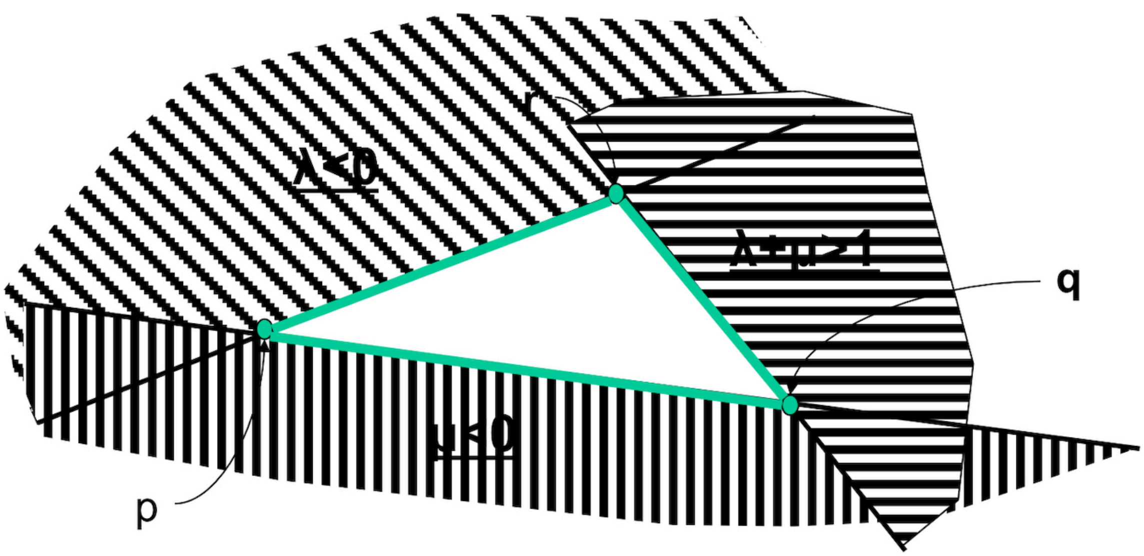
Since the closest point c must ensure $\vec{ac}$ to be the shortest vector among all possible choices of point c. The above functions could be rewritten as a Least Squares form as follows:

$$\begin{bmatrix} \vec{q} - \vec{p} & \vec{r} - \vec{p} \end{bmatrix} \begin{bmatrix} \lambda \\ \mu \end{bmatrix} \approx \vec{a} - \vec{p} \tag{3}$$

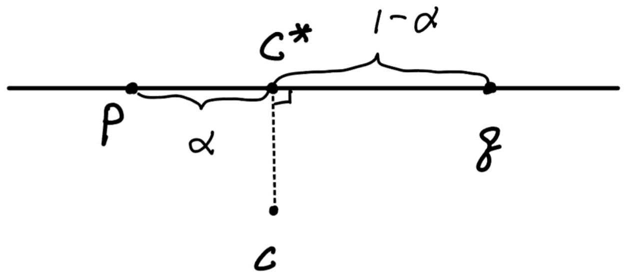
Hence, by solving the Least Squares question, λ and μ could be obtained. According to the geometric relationship in Fig.1, the function could be written as follows.

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

If $\lambda \geq 0, \mu \geq 0, \lambda + \mu \leq 1$, then $\vec{c}$ locates within the triangle and become the closest point.



Otherwise, we have to find a point on the edge of the triangle. There are 3 different regions divided by the λ and μ. For every case, we can project the computed c on a certain edge to find the actual closest point. The projection $c^*$is shown in the following figure:



where α is given by:

$$\alpha = Max(0, Min(\frac{(c-p)\cdot(q-p)}{(q-p)\cdot(q-p)}, 1)) \tag{5}$$

Therefore, c^* can be established by the linear combination of p and q:

$$c^* = p + \alpha \times (q-p) \tag{6}$$

Now, replace c with c^*, we will be able to find the closest point to a specific triangle all the time.

2) Code Implementation

The code implemented in **"/cispa/FindClosestPoint2Triangle.py"** and the test script is developped in **"/PA3/pa3_find_closest_on_triangle.py"**. The steps are as following:

Algorithm 2	**Find Closest Point to Triangle**
Input:	Point a and Triangle $T = \{p, q, r\}$
Output:	Closest Point h
	CALL $[\lambda \quad \mu]^T$ = LeastSquare($[\vec{q}-\vec{p} \quad \vec{r}-\vec{p}], \vec{a}-\vec{p}$)
	COMPUTE: $h = p + \lambda(q-p) + \mu(r-p)$
	IF $\lambda < 0$:
	h = ProjectOnSegment(h, r, p)
	ELIF $\mu < 0$:
	h = ProjectOnSegment(h, p, q)
	ELIF $\lambda + \mu > 1$:
	h = ProjectOnSegment(h, q, r)
	ENDIF
RETURN:	h is the closest point on the triangle

3. Find the Closest Point on Mesh - Linear Search

1) Mathematical Method

For each point in $\{\vec{d}_k\}$, finding the closest triangle mesh equals to find the nearest point $\vec{C}_{k,i}$. The closest triangle could be find in many methods. The methods such as Brute-Force Search, and Octree-based Search have been used in our assignment.

Build linear list of triangles and search for closest triangle to a given point. By calling FindClosestPoint2Triangle function introduced in subsection 2, we can get a set of closest points for all triangles in the given mesh. Compare the minimum distance for all triangles and find the closest one. This method is called Brute-Force Search.

2) Code Implementation

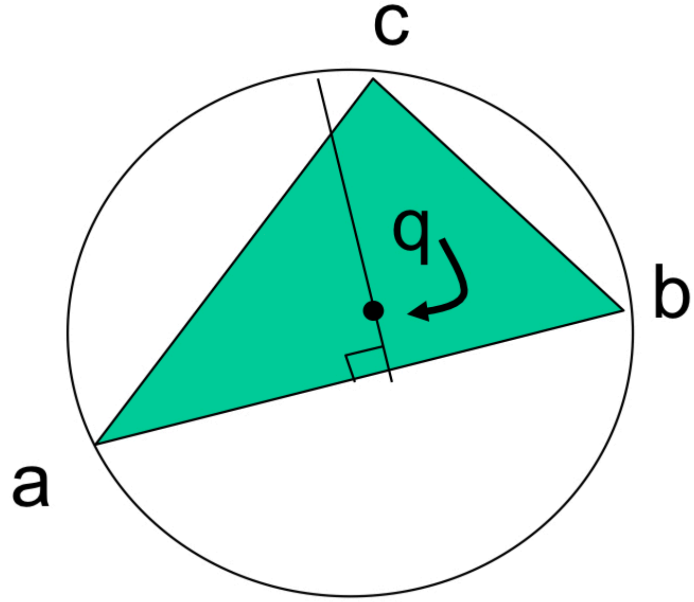
The code implemented in**"cispa/FindClosestPoint2Mesh.py"** and the test script is **"/PA3/pa3_linear_search_test.py"**.

Algorithm3	**Linear Search by Brute Force**
INPUT	Vertices Coordinates and Triangle Indices
OUTPUT	A tuple that contains closest point and its corresponding distance
	minimum distance = ∞
	FOR i = index of the triangles
	CALL: $\vec{C}_{k,i}^{nearest}$ = FindClosestPoint2Triangle($\vec{d}_k$, $triangle_i$)
	$\vec{d}_k = \|\|\vec{d}_k - \vec{C}_{k,i}^{nearest}\|\|$
	IF $\vec{d}_k$ $\leq$minimum distance **then**
	minimum distance =$\vec{d}_k$
	closest point = $\vec{C}_{k,i}^{nearest}$
	ENDIF
	ENDFOR
RETURN	[minimum distance, closest point]

4. Generate Bounding Spheres

1) Mathematical Method

Build bounding spheres around each triangle and with the help of these spheres, we could reduce the number of careful checks required. The working flow is as follows. The scenario to find bounding sphere for each mesh triangle is shown as follows



Here we assume edge $\vec{a}-\vec{b}$ is the longest. Then the center $\vec{q}$ of the sphere will work as follows.

$$(\vec{b}-\vec{q})(\vec{b}-\vec{q}) = (\vec{a}-\vec{q})(\vec{a}-\vec{q}) \tag{7}$$

$$(\vec{c}-\vec{q})(\vec{c}-\vec{q}) \leq (\vec{a}-\vec{q})(\vec{a}-\vec{q}) \tag{8}$$

$$(\vec{b}-\vec{a}) \times (\vec{c}-\vec{a}) \cdot (\vec{q}-\vec{a}) = 0 \tag{9}$$

We could caculate the radius and center as follows.

(1) Compute

$$\vec{f} = \frac{(\vec{a}+\vec{b})}{2} \tag{10}$$

(2) Define

$$\vec{u} = \vec{a}-\vec{f}; \vec{v} = \vec{c}-\vec{f}; \vec{d} = (\vec{u} \times \vec{v}) \times \vec{u} \tag{11}$$

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

$$\vec{q} = \vec{f} + \lambda\vec{d} \tag{12}$$

with $(\lambda\vec{d}-\vec{v})^2 \leq (\lambda\vec{d}-\vec{u})^2$. We could simplified as follows:

$$\lambda \geq \frac{\vec{v}^2 - \vec{u}^2}{2d(\vec{v} - \vec{u})} = \gamma \tag{13}$$

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

2) Code Implementation

The code implemented in **"/cispa/FindBoundingSphere.py"**. We use spheres to represent the triangles and did the same brute force search process.

Algorithm4	**Linear Search by Bounding Spheres**
Step. 1	bound $\leftarrow\infty$
Step. 2	**for** $i = 1$ to number of triangles do
Step. 3	**if** $\|\|q_i - a\|\| - r_i \leq bound$ **then**
Step. 4	$c_i = \|\|\vec{d}_k - \vec{C}_{k,i}^{nearest}\|\|$
Step. 5	**if** $d_i \leq bound$ **then**
Step. 6	bound$\leftarrow d_i$
Step. 7	closest point$\leftarrow c_i$
Step. 8	**end if**
Step. 9	**end if**
Step 10	**end for**

####

5 . Find Closest Point on Mesh - Octree Search

1) Mathematical Method

The algorithm for Octree Search has two main parts: the tree construction and the closest point detection. Since the octree is a pure data structure, hence we demonstrate the method directly through pseudocode. We have implemented a Python according to the slides in the class.

Algorithm5	**Octree Search**
INPUT	Mesh, tip point d_k
OUTPUT	Closest point
	CALL Octree(mesh) to generate an octree for given mesh
	CALL FindClosestPoint to find the closest subtree
	CALL UpdateClosest

And the octree follows the given process:

Algorithm 6 **Octree**

Split the Spheres into 8 quadrants
According to the center position, keep dividing the Sphere into 8 parts;
Keep generating subtrees until the subtree only contains 1 sphere, i.e.
if Subtree.length < 2
endif

---

2) Code Implementation

The octree generation part implemented in **"/cispa/octree.py"**. To call the octree object, we use an octree object to perform the FindClosestPoint2Mesh process, which is **"/cispa/FindClosestPoint2Mesh.py"**.

II. Overall Structure

The overall structure for the ../PROGRAMS folder is described as follows:

```
└──── PROGRAMS
   ├──── PA3                         # Test scripts are contained in this directory
   │    ├──── Data                   # This DIR contains all the provided data
   │    ├──── output                 # This DIR contains all the result of our program
   │    ├──── pa3_main.py            # This is the main process that output the result
   │    ├──── pa3_compute_dk_test.py # compute the tip point
   │    ├──── pa3_find_closest_on_triangle.py
   │    ├──── pa3_octree_search_test.py  # Octree search to find closest point on mesh
   │    └──── pa3_linear_search_test.py  # Linear search to find closest point on mesh
   └──── cispa                       # Utility Functions are in this directory
      ├──── CarteFrame.py
      ├──── ComputeExpectValue.py
      ├──── CorrectDistortion.py
      ├──── DataProcess.py           # Import and Export Data
      ├──── FindBoundingSphere.py
      ├──── FindClosestPoint2Mesh.py     # Find the closest point on the mesh to point
      ├──── FindClosestPoint2Triangle.py # Find the closest point on the triangle
      ├──── HomoRepresentation.py
      ├──── Octree.py                # Generate the octree based on given mesh
```

```
├── PivotCalibration.py
└── Registration.py
```

III. Unit Test and Debug

NOTE: Please change your directory to the ${PROGRAMS} directory before you start the unit test. In our case, the command is:

```
$ cd ~/*PARENT DIR*/PROGRAMS
```

And list all files to check whether you are in the correct directory.

```
$ ls
PA3   cispa
```

Dear Graders, Before starting to run our code, please remember to install **matplotlib** in your virtual environment.

1. Compute dk Unit Test

This script is implement to compute the tip positions w.r.t. B body frame for all LED data frames. In the ../PROGRAMS directory, run test script "/PA3/pa3_compute_dk_test.py". In the terminal, run the following command:

```
$ python3 /PA3/pa3_compute_dk_test.py
INFO     dk for PA3-A-Debug data:
                    [[  5.75770718  19.75264402  -3.01776797]
                     [ 17.90665894  25.32838102 -18.84286624]
                     [-39.30008812 -22.32569893 -30.75256506]
                     [ -9.82756727 -13.44235012  -1.22125489]
                     [-24.99293076 -28.7577944  -38.54036667]
                     [-41.37003445 -14.70559823 -29.90157133]
                     [ 33.83443216  16.2533508    7.01085956]
                     [ -2.48501569  -7.41132429  29.48201166]
                     [ 26.65360204  23.29655741 -10.08941296]
                     [  3.48440717   8.81466607 -14.49922852]
                     [-22.84699817 -12.85719931 -48.78797397]
                     [ 23.77236338  -6.19173927  13.76863119]
                     [ -6.24773552  10.3658506    2.06087393]
                     [-39.00797053 -17.63316542 -14.57114666]
                     [-24.84466034 -11.01359415  -6.08798191]]
```

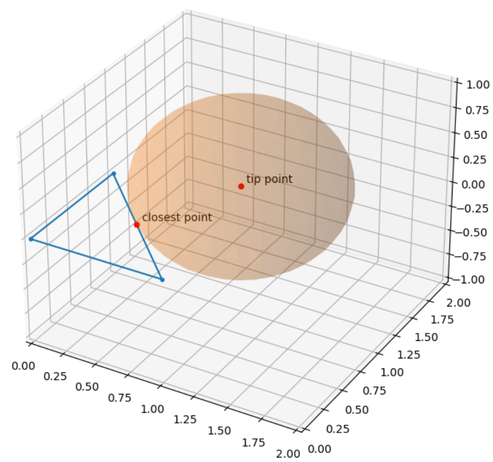
Compare the result to given answer, we can see our code is correct.

2. Closest Point on Triangle Unit Test

This script is to find closest point for a given point and a given triangle. In the ../PROGRAMS directory, run test script "/PA3/pa3_find_closest_on_triangle.py". In the terminal, run the following command:

```
$ python3 PA3/pa3_find_closest_on_triangle.py
```

And we use matplotlib to demonstrate our result:



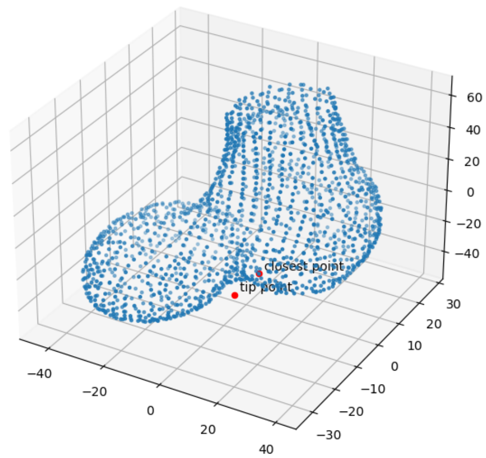
The sphere is a little bit distorted due to the unequal axis range. But as we can see, the result closest point is correct.

3. Linear Search Unit Test

This script is used to find closest point on mesh routine that works by linear search through all the triangles. In the ../PROGRAMS directory, run test script "/PA3/pa3_linear_search_test.py". In the terminal, run the following command:

```
$ python3 /PA3/pa3_linear_search_test.py
```

To demonstrate the result, we plotted the triangle and the point together with the corresponding closest point.



4. Compute Bounding Sphere Unit Test

This script is designed to check whether the Bounding Sphere for one given triangle is correctly generated.

```
$ python3 PA3/pa3_single_sphere_test
[0.5 0.5 0. ]
0.7071067811865476
```

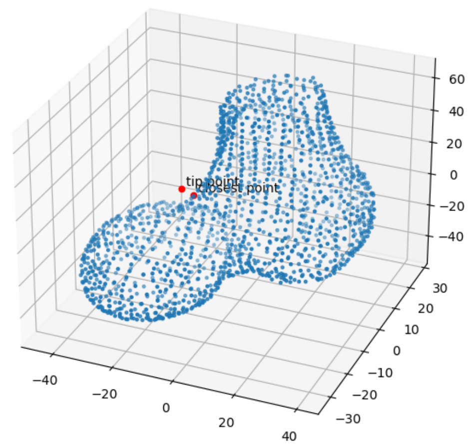
Here we use the triangle with three vertices, which are [[0, 0, 0], [1, 0, 0], [0, 1, 0]]. Through simple inspection, we can see that our result is correct.

5 Octree Search Unit Test

We generate a octree based on the given triangular mesh. And then we perform octree search. In the ../PROGRAMS directory, run test script "/PA3/pa3_octree_search_test.py". In the terminal, run the following command:

```
$ python3 PA3/pa3_octree_search_test.py
```

To demonstrate the result, we plotted the triangle and the point together with the corresponding closest point.



By simple inspection, we can see that our result is correct.

3. Discussion for the results of finding point pairs in different cases

1) Debug case

Table 1 shows us a quantitative evaluation of various proposed methods to find the closest points. For both the Debug and Unknown cases, we compared the running time of the four methods to demonstrate the efficiency improvement.

In Table 2 , compared with the Brutal Search, the Octree Search has largely reduced running time. Beside the octree generation time comsuption, the search process only takes 10% of time for the brute force search. This could be caused by the performance in the implementation of python. During the process of the tree construction, there would be a series of calling and assignment operations for List type limiting the efficiency of Octree.

Table 1: Quantitative evaluation of the proposed methods' accuracy. All the methods come to same accuracy. The error has been computed in L2 Norm

Case	***d* error**	***c* error**
A	0.0053	0.0061
B	0.0054	0.0056
C	0.0061	0.0062
D	0.0087	0.0067
E	0.0081	0.0078
F	0.0082	0.0073

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

Method	**Time(s)**					
	A	B	C	D	E	F
Brutal Force Solver	1.4505	1.4548	1.4521	1.4238	1.4501	1.4422
Octree Solver	0.135787	0.162758	0.161032	0.129104	0.1443400	0.099773

2) Unknown case

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

In the section, we also give time comparison. For the 3 cases, the Octree based Search method performs better than the Bounding Sphere Search method.

Table 3: Comparison of various methods in running time for Debug case G-K

Method	**Time(s)**			
	G	H	I	J
Brute Force Solver	1.4533	1.4526	1.4563	1.4601
Octree Solver	0.145598	0.139905	0.125763	0.173482

IV. Result and Discussions

1.A-Debug

The result is compared with the "PA3-A-Debug-own-output.txt" and shown in the table below.

TABLE 1 PA3-A-Debug-own-output

OUR RESULTS	GIVEN RESULTS
(17.90666 25.32838 -18.84287 17.90731 25.32410 -18.84304 0.00434)	(17.91 25.32 -18.84 17.91 25.32 -18.84 0.000)
(-39.30009 -22.32570 -30.75257 -39.30331 -22.32773 -30.75337 0.00389)	(-39.30 -22.33 -30.75 -39.30 -22.33 -30.75 0.000)
(-9.82757 -13.44235 -1.22125 -9.82755 -13.44231 -1.22128 0.00005)	(-9.83 -13.44 -1.22 -9.83 -13.44 -1.22 0.000)

2.B-Debug

The result is compared with the "PA3-B-Debug-own-output.txt" and shown in the table below.

TABLE 2 PA3-B-Debug-own-output

OUR RESULTS	GIVEN RESULTS
(1.15508 11.01661 -8.29899 1.18421 10.94645 -8.26507 0.08320)	(1.15 11.01 -8.30 1.18 10.95 -8.27 0.082)
(-31.61757 -5.26328 -12.85190 -32.46366 -3.96838 -9.86385 3.36468)	(-31.62 -5.26 -12.85 -32.46 -3.97 -9.86 3.366)
(-24.43960 -0.75184 -14.42152 -25.33479 0.98430 -12.44331 2.78008)	(-24.44 -0.76 -14.42 -25.34 0.98 -12.44 2.784)

3.C-Debug

The result is compared with the "PA3-C-Debug-own-output.txt" and shown in the table below.

TABLE 3 PA3-C-Debug-own-output

OUR RESULTS	GIVEN RESULTS
(27.51716 -8.17727 6.19635 27.51501 -8.14967 6.18664 0.02933)	(27.52 -8.18 6.20 27.51 -8.15 6.19 0.030)
(-8.77465 6.25957 46.45507 -6.95442 6.25081 46.40286 1.82100)	(-8.78 6.25 46.45 -6.95 6.25 46.40 1.823)
(27.63058 -11.79476 -12.06240 27.82258 -12.01527 -11.99464 0.30014)	(27.63 -11.80 -12.06 27.82 -12.02 -12.00 0.298)

4.D-Debug

The result is compared with the "PA3-D-Debug-own-output.txt" and shown in the table below.

TABLE 4 PA3-D-Debug-own-output

OUR RESULTS	GIVEN RESULTS
(15.42274 19.93007 35.51362 15.44614 21.76373 36.08316 1.92021)	(15.42 19.93 35.51 15.45 21.76 36.08 1.922)
(-5.15204 -18.60613 -8.49794 -5.19629 -18.17795 -8.68783 0.47048)	(-5.15 -18.60 -8.50 -5.19 -18.18 -8.69 0.470)
(3.33449 22.29966 27.79139 3.45898 23.17946 27.91469 0.89708)	(3.34 22.29 27.79 3.46 23.18 27.92 0.902)

5.E-Debug

The result is compared with the "PA3-E-Debug-own-output.txt" and shown in the table below.

TABLE 5 PA3-E-Debug-own-output

OUR RESULTS	GIVEN RESULTS
(-2.28196 -5.06556 -27.61830 -0.78391 -4.32855 -28.66912 1.97271)	(-2.28 -5.07 -27.62 -0.78 -4.34 -28.67 1.976)
(-41.51150 -13.61948 -42.49447 -38.68369 -13.27770 -40.83609 3.29599)	(-41.51 -13.61 -42.49 -38.68 -13.28 -40.84 3.296)
(30.09727 15.19825 -6.44416 33.02990 17.64569 -6.29841 3.82251)	(30.09 15.21 -6.45 33.02 17.65 -6.30 3.816)

6.F-Debug

The result is compared with the "PA3-F-Debug-own-output.txt" and shown in the table below.

TABLE 6 PA3-F-Debug-own-output

OUR RESULTS	GIVEN RESULTS
(-9.27443 -29.37243 -38.63571 -10.81595 -27.29781 -37.41410 2.85878)	(-9.27 -29.37 -38.63 -10.81 -27.29 -37.41 2.856)
(-38.37045 0.91475 -25.07241 -39.95337 1.99688 -25.06008 1.91750)	(-38.36 0.91 -25.07 -39.95 2.00 -25.06 1.924)
(-5.78249 -29.91310 -19.62803 -7.35776 -27.20945 -20.30206 3.20085)	(-5.79 -29.91 -19.63 -7.36 -27.21 -20.30 3.195)

7.G-Debug

The result is compared with the "PA3-G-Unknown-own-output.txt" and shown in the table below.

TABLE 7 PA3-G-Unknown-own-output.txt

OUR RESULTS
(-13.66093 12.38037 30.02801 -13.96138 11.96082 30.24026 0.55799)
(16.00902 24.41337 8.53591 16.55037 26.03651 9.01081 1.77572)
(9.75001 15.57415 -10.16029 9.92203 15.53096 -9.94482 0.27908)

8.H-Debug

The result is compared with the "PA3-H-Unknown-own-output.txt" and shown in the table below.

TABLE 8 PA3-H-Unknown-own-output.txt

OUR RESULTS
(2.51675 -13.64304 8.64420 2.83203 -11.96537 8.47300 1.71560)
(-4.31288 -12.33908 -37.45976 -2.14691 -12.27012 -38.16811 2.27991)
(-35.97575 -7.62909 -42.13914 -36.66262 -7.35385 -42.91870 1.07483)

9.J-Debug

The result is compared with the "PA3-J-Unknown-own-output.txt" and shown in the table below.

TABLE 9 PA3-J-Unknown-own-output.txt

OUR RESULTS
(21.48054 -0.92945 49.72730 20.59832 -0.49203 49.58690 0.99467)
(25.36021 6.03934 25.74393 27.53223 5.82745 26.43426 2.28892)
(8.03087 10.08615 -11.70728 7.97719 10.55353 -11.99916 0.55365)

10. Discussion

The result for Debug-A to F is pretty accurate, the maximum error is limited to 0.001 level. And there exists slightly difference observed between Answer data and Output data. According to the PA instruction, there might be some noise in the data.

Contributions

Jiaming Zhang developed method ; Chongjun Yang developed other methods of this PA. Both team member complete a part of this report.

References

[1] Jiaming Zhang, Chongjun Yang. PA1-REPORT

[2] Jiaming Zhang, Chongjun Yang. PA2-REPORT

[3] https://en.wikipedia.org/wiki/Octree