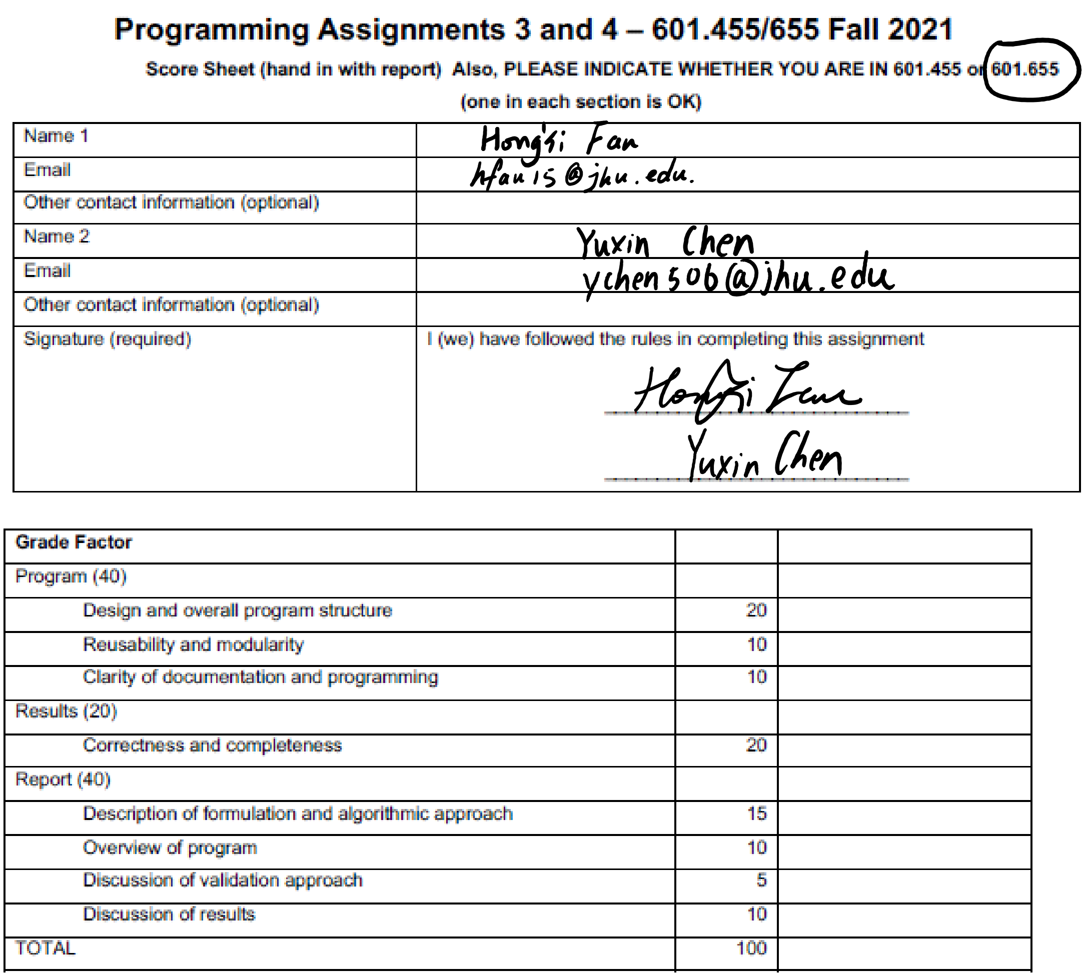
Overview

Saturday, November 13, 2021 7:13 PM



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	Hongyi Fan
Email	hfan15@jhu.edu.
Other contact information (optional)	
Name 2	Yuxin Chen
Email	ychen506@jhu.edu
Other contact information (optional)	
Signature (required)	I (we) have followed the rules in completing this assignment [illegible] Yuxin Chen

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	

Overview

Contents of this document:

- Program Structure
- Algorithm of Algorithm of computing $\overrightarrow{d_k}$
- Algorithm of Iterative Closest Point (matching part)
- Algorithm of find closest point
- Validation and Results
- A tabular summary of the results obtained for unknown data

A short statement of who did what:

Yuxin Chen:

We went through the whole assignment together and discussed the algorithm and mathematical approach we decided to use. Then I contributed simple method. We wrote the report together.

HongYi Fan:

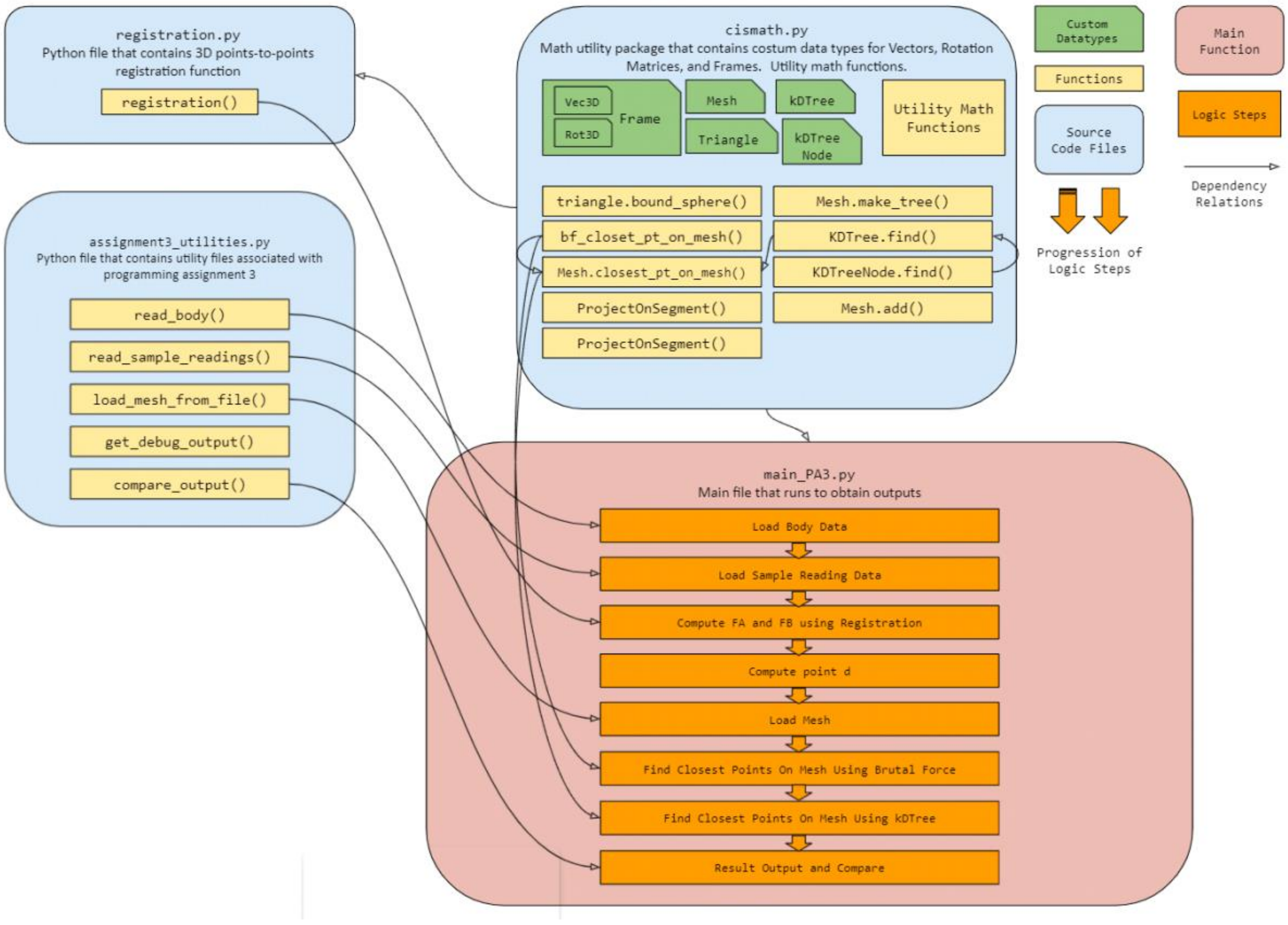
- Contributed in majority of CIS math package
- Contributed in closest point on mesh with brutal force and KD-Tree
- Contributed in report writing and validating

External Resources:

- *Numpy (https://numpy.org/)*
- *Figure 1[GPL, https://commons.wikimedia.org/w/index.php?curid=589909]*

Program Structure

Saturday, November 13, 2021 7:16 PM



Algorithm of computing $\overrightarrow{d_k}$

Wednesday, November 17, 2021 12:46 PM

Algorithm of computing $\overrightarrow{d_k}$

Known:

- $\mathrm{N_A}$: number of LED markers on A rigid body
- N_B: number of LED markers on B rigid body
- N_{samps}: number of sample frams
- $\overrightarrow{a_i}$: xyz coordinates of A body LED markers in body coordinates
- $\overrightarrow{b_i}$: xyz coordinates of B body LED markers in body coordinates
- $\overrightarrow{A_i}$: xyz coordinates of A body LED markers in tracker coordinates
- $\overrightarrow{B_i}$: xyz coordinates of B body LED markers in tracker coordinates
- $\overrightarrow{A_{tip}}$: xyz coordinates of tip in A body coordinates

Goal:

- Calculate the $\overrightarrow{d_k}$ (the position of the pointer tip with respect to rigid body B)

STEPS:

1. For each sample frame k, using $\overrightarrow{a_{i,k}}$ (xyz coordinates of A body LED markers in body coordinates) and $\overrightarrow{A_{i,k}}$ (xyz coordinates of A body LED markers in tracker coordinates) to calculate $F_{A,k}$

   Since $\overrightarrow{A_{i,k}} = F_{A,k} \cdot \overrightarrow{a_{i,k}}$,
   To find $F_{A,k}$, we used the 3D point set to 3D point set registration algorithm we developed in PA1 (details of algorithm also explained in PA1 report).
   We set coordinates of $\overrightarrow{a_{i,k}}$ as the first 3D point set and $\overrightarrow{A_{i,k}}$ as the second 3D point set.
   Then the algorithm will calculate $F_{A,k}$

2. Then for each sample frame k, using $\overrightarrow{b_{i,k}}$ (xyz coordinates of B body LED markers in body coordinates) and $\overrightarrow{B_{i,k}}$ (xyz coordinates of B body LED markers in tracker coordinates) to calculate $F_{B,k}$

   Since $\overrightarrow{B_{i,k}} = F_{B,k} \cdot \overrightarrow{b_{i,k}}$,
   To find $F_{B,k}$, we used the 3D point set to 3D point set registration algorithm we developed in PA1 (details of algorithm also explained in PA1 report).
   We set coordinates of $\overrightarrow{b_{i,k}}$ as the first 3D point set and $\overrightarrow{B_{i,k}}$ as the second 3D point set.
   Then the algorithm will calculate $F_{B,k}$

3. After we get $F_{A,k}$ *and* $F_{B,k}$, we compute the $\overrightarrow{d_k}$ by using the following equation
   $$\overrightarrow{d_k} = {F_{B,k}}^{-1} \cdot F_{A,k} \cdot \overrightarrow{A_{tip}}$$

4. Then we get the output: $\overrightarrow{d_k}$

Algorithm of ICP(matching) and Simple brutal-force method

Saturday, November 13, 2021 7:17 PM

Iterative Closest Point (matching part):

Known:

Input:

- A 3D Surface model M, represented as a mesh of triangles read from the mesh file (the coordinates of the vertices of this mesh in CT coordinates
- Set of points $\overrightarrow{d_k}$ (the position of the pointer tip with respect to rigid body B) that known to be on the 3D surface model M.
- Initial guess F_0 for transformation F_{reg} such that the points $F_{reg} * \overrightarrow{d_k}$, lie on surfece model M. In PA3, we assume the F_0 to be [I | 0].
- Initial threshold η_0 for the match closeness: Initial threshold η_0 will be firstly set as fairly large value based on the data we collected. The threshold will then decrease after a couple of iterations.

Goal:

- For the matching part of ICP in PA3, our goal is to find the closest point $\overrightarrow{c_k}$ on surface model M for $F_n \cdot \overrightarrow{d_k}$. While in PA3, we assume F_0 to be [I | 0] and only run first itiration for only matching part.

Steps:

```
n ← 0
A ← ∅ ,   B ← ∅
For k ← 1 step 1 to N do:
    $bne_k = \left\| F_n \cdot \overrightarrow{d_k} - \overrightarrow{c_k} \right\|$
    [ $\overrightarrow{c_k}$, i, $e_k$] ← FindClosestPoint ($F_n \cdot \overrightarrow{d_k}$, $\overrightarrow{c_k}$, $i_k$, $bne_k$, T)
        ○ Where T is the method we choose to use
        ○ Two method are used in this programming method:
            1. Simple brutal force method
            2. Advanced method with kD-Tree
    ○ If ($e_k < \eta_k$) then {put $\overrightarrow{d_k}$ into A; put $\overrightarrow{c_k}$ in to B};
In PA3, as required in prompt, we output the $\overrightarrow{c_k}$ for $F_0 \cdot \overrightarrow{d_k}$ for first itiration
```

Simple brutal-force method:

1) Find the closest triangles finding

a. Construct the linear list of triangles. List three vertices of each triangle with the index of the triangle.

b. Search sequentially for closest point on each triangle using method introduced below. Among all results, return the point with shortest distance to the given point among.

2) Find the closest point in the triangle

a. We assume the point that we are looking at is point a.

b. Three vertices of the closest triangle we found are r, p and q.

c. To find the closest point c on this closest triangle. We use the following equation:

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

d. To solve this equation, we use the least square error method.

$$[\,\vec{q} - \vec{p} \quad \vec{r} - \vec{p}\,]\begin{bmatrix}\lambda \\ \mu\end{bmatrix} = [\vec{a} - \vec{p}]$$

e. Then we compute

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

f. If $\lambda \geq 0, \mu \geq 0, \lambda + \mu \leq 1$, then $\vec{c}$ lies within the triangle and is the cloest point. Otherwise, we need to find a point on the border of the triangle.

g. If $\mu < 0$,

$$\lambda = \frac{(\vec{c} - \vec{p})(\vec{q} - \vec{p})}{(\vec{q} - \vec{p})(\vec{q} - \vec{p})}$$
$$\lambda^{(seg)} = Max(0, Min(\lambda, 1))$$
$$c^* = \vec{p} + \lambda^{(seg)} \times (\vec{q} - \vec{p})$$

h. If $\lambda < 0$,

$$\lambda = \frac{(\vec{c} - \vec{r})(\vec{p} - \vec{r})}{(\vec{p} - \vec{r})(\vec{p} - \vec{r})}$$
$$\lambda^{(seg)} = Max(0, Min(\lambda, 1))$$
$$c^* = \vec{r} + \lambda^{(seg)} \times (\vec{p} - \vec{r})$$

i. If $\lambda + \mu > 1$,

$$\lambda = \frac{(\vec{c} - \vec{q})(\vec{r} - \vec{q})}{(\vec{r} - \vec{q})(\vec{r} - \vec{q})}$$
$$\lambda^{(seg)} = Max(0, Min(\lambda, 1))$$
$$c^* = \vec{q} + \lambda^{(seg)} \times (\vec{r} - \vec{q})$$

Algorithm of find closest point

Wednesday, November 17, 2021 20:47

Optimized Search Using KD-Tree Structure

Since a high definition mesh structure may have an overwhelming quantity of triangles, calculating closest point to each triangle is not time efficient. Therefore, a KD-Tree Data Structure can be used to optimize the time efficiency by selectively choosing portion of triangles that is closest to the given point in space.

1) Add a bounding sphere attribute to each triangle

Doing so helps the program to keep track of triangles' locations and bounding using one point instead of three

Mathematical Approach:

For each triangle with vertices a, b, c, where (a, b) is the long edge:

1. Calculate possible center $\vec{q} = \frac{\vec{a}+\vec{b}}{2}$
2. Check $\vec{q}$ with counding sphere inequalities:
   a. $\left(\vec{b}-\vec{q}\right)\cdot\left(\vec{b}-\vec{q}\right) = (\vec{a}-\vec{q})\cdot(\vec{a}-\vec{q})$
   b. $(\vec{c}-\vec{q})\cdot(\vec{c}-\vec{q}) \le (\vec{a}-\vec{q})\cdot(\vec{a}-\vec{q})$
   c. $\left(\vec{b}-\vec{a}\right)\times(\vec{c}-\vec{a})\cdot(\vec{q}-\vec{a}) = 0$
3. If above inequalities hold, then $\vec{q}$ is the center of the sphere. Continue otherwise
4. Calculate $\vec{f} = \frac{\vec{a}+\vec{b}}{2}$
5. Define $\vec{u} = \vec{a}-\vec{f}, \vec{v} = \vec{c}-\vec{f}, \vec{d} = (\vec{u}\times\vec{v})\times\vec{u}$, now center $\vec{q}$ lies in $\vec{q} = \vec{f}+\lambda\vec{d}$
6. Solve $\lambda \ge \frac{\vec{v}^2-\vec{u}^2}{2\vec{d}\cdot(\vec{v}-\vec{u})} = \gamma$, if $\gamma \le 0$ $then\ \lambda = 0. Otherwise\ \lambda = \gamma$
7. With center $\vec{q}$, calculate the radius by calculate the 2-norm of $(\vec{q}-\vec{a})$

2) Construct kD-Tree (3D-Tree)

1. Separate triangles in the 3D space into two sub-space by the x value of their bounding spheres'. That is, as shown in the figure on the right, pick the median element m among all triangles, for all triangles with x value less than m, is on the leftside of the red plane. All other triangles, including m, is on the right side of the red plane.
2. For each sub space of triangles, divide it again as described in step 1, but uses y values instead of x value (the green planes). And apply again using the z values (the blue plane). And then x values again. Until the subspace contains only one triangle or desired depth is reached.
3. Since triangle shape may reach out of these boxes, like figure on the right shows. Therefore adjust boxes to enclose triangles.
4. Now we have a tree-like structure in which a node (space)
   Has 2 child nodes(subspaces)

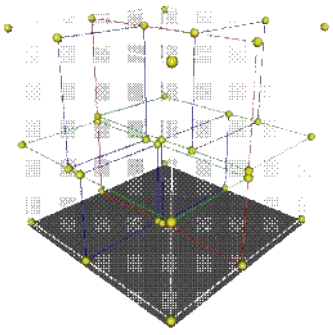


Figure 1

3) Search Using the kD-Tree

1. Given a point $\vec{p}$ in space, locate the closest leaf node
   If the x coordinate of $\vec{p}$ is less than the median value of the current node then go the left node, otherwise the right node, etc.
2. After reaching the leaf node, find closest point on each of the triangles, store the closest point $\vec{c}$ and the distance to the closest point r.
3. Consider a sphere formed on point $\vec{c}$ with radius r. The sphere includes other possible closer triangles in the space. If the sphere touches other node spaces, it means that space may contain closer triangle. Therefore, we also need to look into these nodes.
4. If a closer point on triangle is found during the process, update $\vec{c}$ and r accordingly. Until the sphere on $\vec{c}$ with radius r does not intersect with pther node. We now have the closest point on mesh.

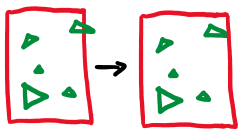


Figure 2

Validation and Result

Saturday, November 13, 2021 7:17 PM

To verify the result, we have the following testing:
Simple Brutal-Force method:

1) Verify the $\vec{c}$ with the output data
2) Verify the $\vec{d}$ with the output data
3) Verify the $\left\|\overrightarrow{d_k} - \overrightarrow{c_k}\right\|$ with the output data

Advanced method with kD-Tree:

1) Verify the $\vec{c}$ with the output data
2) Verify the $\vec{d}$ with the output data
3) Verify the $\left\|\overrightarrow{d_k} - \overrightarrow{c_k}\right\|$ with the output data

Compare the clever method against the simple brute force search

1) Record time comsumed by the program to run each file
2) Cloest point result

Simple Brutal-Force method:

1) Verify the $\vec{c}$ with the output data
2) Verify the $\vec{d}$ with the output data
3) Verify the $\left\|\overrightarrow{d_k} - \overrightarrow{c_k}\right\|$ with the output data

Average error between our output with the output file provided by professor

Dataset	Average error in $\vec{d}\ (d_x, d_y, d_z)$	Average error in $\vec{c}\ (c_x, c_y, c_z)$	Average error in $\left\|\overrightarrow{d_k} - \overrightarrow{c_k}\right\|$
A	[0.003, 0.005, 0.002]	[0.003, 0.004, 0.002]	0.002
B	[0.003, 0.005, 0.002]	[0.003, 0.002, 0.003]	0.002
C	[0.003, 0.004, 0.002]	[0.004, 0.002, 0.003]	0.002
D	[0.004, 0.005, 0.003]	[0.003, 0.003, 0.003]	0.003
E	[0.005, 0.006, 0.003]	[0.005, 0.004, 0.003]	0.003
F	[0.004, 0.005, 0.002]	[0.003, 0.005, 0.003]	0.004

Discussion of the result:
As presented in the table above, the average error between our output with the output file provided by professor are very small, which are all less then two decimals. Since the output result provided from professor are only two decimal result, these error are very small. The table could prove that our simple brute force method works properly and give the correct result.

Advanced method with kD-Tree:

1) Verify the $\vec{c}$ with the output data
2) Verify the $\vec{d}$ with the output data
3) Verify the $\left\|\overrightarrow{d_k} - \overrightarrow{c_k}\right\|$ wotj the output data

Validation and Result

Wednesday, November 17, 2021 19:51

Average error between our output with the output file provided by professor

Dataset	Average error in $\vec{d}\ (d_x, d_y, d_z)$	Average error in $\vec{c}\ (c_x, c_y, c_z)$	Average error in $\left\|\overrightarrow{d_k} - \overrightarrow{c_k}\right\|$
A	[0.003, 0.005, 0.002]	[0.003, 0.004, 0.002]	0.002
B	[0.003, 0.005, 0.002]	[0.003, 0.002, 0.003]	0.002
C	[0.003, 0.004, 0.002]	[0.004, 0.002, 0.003]	0.002
D	[0.004, 0.005, 0.003]	[0.003, 0.003, 0.003]	0.003
E	[0.005, 0.006, 0.003]	[0.005, 0.004, 0.003]	0.003
F	[0.004, 0.005, 0.002]	[0.003, 0.005, 0.003]	0.004

Discussion of the result:

As presented in the table above, the average error between our output with the output file provided by professor are very small, which are all less then two decimals. Since the output result provided from professor are only two decimal result, these error are very small. The table could prove that our advanced method works properly and give the correct result.

Compare between simple method and advanced method with kD-Tree:

Record time comsumed by the program to run each file with differene method

Dataset	Time Elapsed Using Simple Brutal Force Method (s)	Time Elapsed Using kDTree (s)	Time Reduced (s)
A	3.415	0.082	3.333
B	3.430	0.116	3.314
C	3.465	0.110	3.355
D	3.393	0.095	3.298
E	3.448	0.152	3.296
F	3.412	0.164	3.248
Average	3.427	0.120	3.307

Difference between the output from simple Brutal-Force method and advanced method with kD-Tree

Dataset	Difference in $\vec{d}\ (d_x, d_y, d_z)$	Difference in $\vec{c}\ (c_x, c_y, c_z)$	Difference in
A	[0.00, 0.00, 0.00]	[0.00, 0.00, 0.00]	0.00
B	[0.00, 0.00, 0.00]	[0.00, 0.00, 0.00]	0.00
C	[0.00, 0.00, 0.00]	[0.00, 0.00, 0.00]	0.00
D	[0.00, 0.00, 0.00]	[0.00, 0.00, 0.00]	0.00
E	[0.00, 0.00, 0.00]	[0.00, 0.00, 0.00]	0.00
F	[0.00, 0.00, 0.00]	[0.00, 0.00, 0.00]	0.00

Discussion of the result:

As shown in two tables above, we can see that by using the advanced method. We successfully reduce the whole matching time from average 3.427s to average 0.120s among debug data set A-F, which reduced about 96.5% time to match point. The average time consumed with 3135 triangles is 0.120 second. It is much faster than brutal-force method because the average case time complexity using kD-Tree structure is $O(\log n)$ while bf method has a time complexity of $O(n)$.While reducing the time, the accuracy remain the same. From the second table, we can see there is no difference between the result using simple method and advanced method. Thus, these two tables proves that our advanced method works very well to reduce the time to match point by giving the same correct result as simple method.

A tabular summary of the results obtained for unknown data

Saturday, November 13, 2021 7:17 PM

Results obtained for unknown data G

$\vec{d}\ (d_x, d_y, d_z)$	$\vec{c}\ (c_x, c_y, c_z)$	$\left\lVert \overrightarrow{d_k} - \overrightarrow{c_k} \right\rVert$
-11.09, 11.91, 30.48	-12.08, 10.86, 30.9	1.505
18.05, 22.94, 7.77	18.87, 25.41, 8.49	2.708
10.79, 14.21, -10.62	10.65, 14.24, -10.79	0.222
6.72, -15.18, 5.91	6.92, -12.63, 5.29	2.636
-38.36, -18.31, -36.04	-39.16, -18.51, -36.2	0.838
26.74, -18.27, -14.80	24.54, -14.72, -14.63	4.185
-0.66, -8.45, -32.46	-1.46, -8.62, -32.01	0.934
-8.42, 4.53, 8.30	-11.56, 6.18, 8.07	3.549
5.42, -8.81, 43.85	5.26, -6.96, 43.89	1.857
-13.92, -1.82, -45.59	-13.65, -1.42, -46.20	0.775
-13.65, 7.8, -24.14	-11.99, 10.19, -24.67	2.957
41.50, 6.5, -1.60	40.30, 6.11, -1.69	1.267
26.09, -7.70, 18.99	24.64, -4.97, 17.87	3.297
12.37, 9.53, 61.26	12.35, 9.41, 63.09	1.834
16.92, 8.93, -19.68	15.92, 8.73, -19.90	1.049
17.90, 15.30, -33.16	17.99, 15.26, -31.16	2.001
0.63, -14.16, -34.53	-0.33, -14.15, -34.13	1.040
-8.05, -6.03, 11.70	-8.96, -6.79, 12.02	1.224
5.48, -4.59, 61.94	5.46, -4.55, 63.41	1.470
0.68, 6.84, -9.53	0.36, 9.97, -10.20	3.230

Time by using simple brute force method: 4.577s
Time by using advanced method: 0.182s
Time reduced: 4.395s

Difference between the result from simple Brutal-Force method and advanced method with kD-Tree

Difference in $\vec{d}\ (d_x, d_y, d_z)$	Difference in $\vec{c}\ (c_x, c_y, c_z)$	Difference in
[0.00, 0.00, 0.00]	[0.00, 0.00, 0.00]	0.00

Discussion of result:
According to the validation and result proved with debug files, we believe that the result for the unknown dataset should be correct. Both simple method and advanced method gives us the same result. And the time by using advanced method has largely reduced from 4.577s to 0.182s. Thus we think our result for unknow G dataset should be correct.

A tabular summary of the results obtained for unknown data

Wednesday, November 17, 2021 23:25

Results obtained for unknown data H

$\vec{d}\,(d_x, d_y, d_z)$	$\vec{c}\,(c_x, c_y, c_z)$	$\left\Vert\vec{d_k} - \vec{c_k}\right\Vert$
2.23, -14.50, 7.27	2.68, -12.09, 7.02	2.467
-4.47, -8.43, -38.52	-3.05, -8.07, -39.75	1.922
-36.11, -2.65, -42.70	-35.75, -2.76, -42.29	0.556
-12.56, -0.54, -43.45	-11.99, -0.08, -44.16	1.021
-37.99, -21.26, -13.03	-36.80, -20.69, -13.71	1.483
17.23, -13.34, 5.77	16.61, -11.66, 4.93	1.979
-4.81, 7.68, -9.14	-5.20, 8.87, -9.04	1.260
-2.56, -9.47, 42.66	-1.26, -6.07, 42.80	3.646
19.26, -10.74, 14.07	18.70, -7.60, 13.42	3.255
28.99, 18.66, 0.19	31.04, 20.04, 0.17	2.468
12.29, 15.23, 61.82	12.32, 15.23, 63.09	1.274
11.42, 19.05, 29.5	11.44, 22.57, 30.16	3.584
-30.73, -28.13, -14.8	-30.18, -27.55, -15.23	0.909
23.20, 8.21, 40.28	24.06, 8.24, 40.41	0.874
-32.40, -11.01, -6.84	-32.28, -11.06, -7.09	0.285
-20.73, -13.17, -3.95	-20.57, -12.76, -4.71	0.877
-2.13, -17.27, -34.78	-1.18, -17.58, -35.16	1.063
-33.52, 8.38, -31.11	-33.20, 7.86, -30.87	0.653
16.76, -9.15, 42.90	15.51, -5.45, 42.67	3.912
-41.64, -15.61, -34.73	-39.92, -15.47, -34.58	1.737

Time by using simple brute force method: 4.545s
Time by using advanced method: 0.166s
Time reduced: 4.379s

Difference between the result from simple Brutal-Force method and advanced method with kD-Tree

Difference in $\vec{d}\,(d_x, d_y, d_z)$	Difference in $\vec{c}\,(c_x, c_y, c_z)$	Difference in
[0.00, 0.00, 0.00]	[0.00, 0.00, 0.00]	0.00

Discussion of result:
According to the validation and result proved with debug files, we believe that the result for the unknown dataset should be correct. Both simple method and advanced method gives us the same result. And the time by using advanced method has largely reduced from 4.545s to 0.166s. Thus we think our result for unknow H dataset should be correct.

A tabular summary of the results obtained for unknown data

Wednesday, November 17, 2021 12:45 PM

Results obtained for unknown data J

$\vec{d}\,(\mathrm{d}_{x,}\, d_{\mathrm{y}}, d_z)$	$\vec{c}\,(c_{x,}\, c_{\mathrm{y}}, c_z)$	$\left\|\overrightarrow{d_k} - \overrightarrow{c_k}\right\|$
19.70, 3.06, 50.58	21.67, 2.24, 50.91	2.164
24.40, 10.96, 27.18	26.18, 11.51, 27.69	1.933
8.69, 16.56, -10.86	9.65, 16.91, -9.73	1.525
-18.47, -1.62, -0.32	-17.76, -1.85, -0.62	0.807
-40.71, -10.98, -40.88	-39.41, -10.78, -40.17	1.490
-25.18, 15.95, -26.95	-24.48, 12.07, -27.07	3.943
-14.67, 5.46, 9.27	-12.71, 4.33, 9.16	2.270
37.48, 5.33, -12.51	37.82, 5.48, -12.68	0.405
25.04, 12.40, 27.34	25.86, 12.66, 27.57	0.891
20.64, 22.41, 24.94	20.06, 21.48, 24.8	1.097
-2.50, -7.65, 16.92	-3.21, -9.65, 17.22	2.136
10.17, 27.62, -6.92	11.55, 25.40, -6.09	2.741
-28.10, -26.13, -18.53	-29.33, -29.10, -17.45	3.390
32.28, 18.90, 4.38	32.18, 18.82, 4.37	0.126
3.84, -11.02, -22.26	5.16, -11.57, -23.38	1.812
-9.25, 10.69, 35.74	-9.14, 10.75, 35.70	0.129
-10.19, -3.24, 16.72	-9.66, -3.14, 16.6	0.551
4.71, -4.30, 62.62	4.71, -4.28, 63.40	0.781
-2.78, 25.92, 29.73	-3.11, 23.97, 29.28	2.029
-43.05, -13.95, -20.10	-42.63, -13.85, -20.23	0.458

Time by using simple brute force method: 4.576s
Time by using advanced method: 0.219s
Time reduced: 4.357s

Difference between the result from simple Brutal-Force method and advanced method with kD-Tree

Difference in $\vec{d}\,(\mathrm{d}_{x,}\, d_{\mathrm{y}}, d_z)$	Difference in $\vec{c}\,(c_{x,}\, c_{\mathrm{y}}, c_z)$	Difference in
[0.00, 0.00, 0.00]	[0.00, 0.00, 0.00]	0.00

Discussion of result:
According to the validation and result proved with debug files, we believe that the result for the unknown dataset should be correct. Both simple method and advanced method gives us the same result. And the time by using advanced method has largely reduced from 4.576s to 0.219s. Thus we think our result for unknow J dataset should be correct.