



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 | hfau15@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   |

| 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 Iterative Closest Point
- 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 to continue to write the rest ICP algorithm after matching part we did in PA3. We wrote the report together.

HongYi Fan:

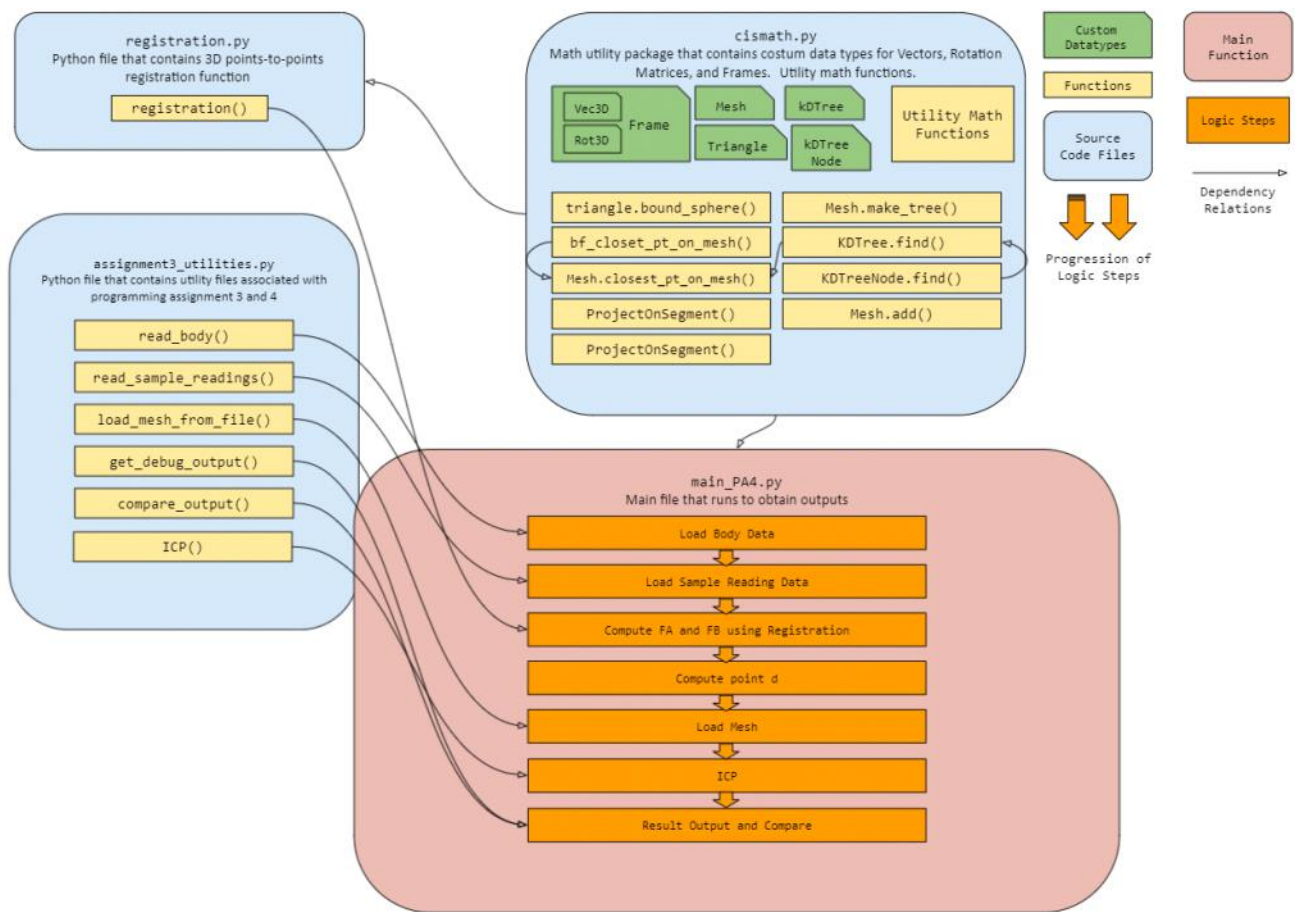
- Contributed in majority of CIS math package
- Contributed in closest point on mesh with brutal force and KD-Tree
- Contributed in ICP optimization
- Contributed in report writing and validating

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 Iterative Closest Point (ICP)

Saturday, November 13, 2021 7:17 PM

Algorithm of computing \vec{d}_k has been explained in PA3 report.

Iterative Closest Point :

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 \vec{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 $F_{reg} * \vec{d}_k$, lie on surface model M. In PA3, we assume the F_0 to be $[I \mid 0]$.
- 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:

- Given a set of triangles (mesh) of the patient's bone model, and a set of points obtained on the surface of the mesh, obtain the rigid transformation that describes the pose of the patient's bone relative to the tracker.

Steps:

Step 0: Initialization

$n \leftarrow 0$

$\eta_0 \leftarrow$ large number

$\vec{d}_{k_{sample}} \leftarrow$ an uniform random sample of \vec{d}_k , this group of vector used to shorten ICP time cost in early iterations. Here choose $\vec{d}_{k_{sample}}$ to be 1/3 or 1/4 of the size of \vec{d}_k

$C \leftarrow$ closest points on M

$$e_k = \left\| F_n \cdot \vec{d}_k - \vec{c}_k \right\|$$

Step 1: matching

$A \leftarrow \emptyset, B \leftarrow \emptyset$

For $k \leftarrow 1$ step 1 to N do:

$$bne_k = \left\| F_n \cdot \vec{d}_{k_{sample}} - \vec{c}_k \right\|$$

$[\vec{c}_k, i, e_k] \leftarrow \text{FindClosestPoint}(F_n \cdot \vec{d}_{k_{sample}}, \vec{c}_k, i_k, bne_k, T)$

- 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 \vec{d}_k into A; put \vec{c}_k in to B};

Step 2: transformation update

$n = n + 1$

$F_n \leftarrow \text{FindBestRigidTransformation}(A, B)$

$$\sigma_n \leftarrow \frac{\sqrt{\sum_k \vec{e}_k \cdot \vec{e}_k}}{\text{NumElts}(E)}$$

$$(\varepsilon_{max})_n \leftarrow \max_k \sqrt{\vec{e}_k \cdot \vec{e}_k}$$

$$\bar{\varepsilon}_n \leftarrow \frac{\sum_k \sqrt{\vec{e}_k \cdot \vec{e}_k}}{\text{NumElts}(E)}$$

Algorithm of Iterative Closest Point (ICP)

Monday, November 22, 2021 9:02 PM

Step 3: adjustment

Compute η_n from $\{\eta_0, \dots, \eta_{n-1}\}$:

η is the threshold parameter that determines whether a pair of closest points are considered as matched. If a pair of closest points has euclidean distance less than η , then they are considered as a matched pair. η is set to be a large number at the beginning of ICP to accommodate large euclidean distance between points, then η is adaptively decreased as the ICP iterates.

The estimation of η in Programming Assignment 4 is calculated as follow:

$$\eta = 3 \times \bar{\epsilon}_n$$

Step 4: Termination Check With Sample $\vec{d}_{k_{sample}}$ Group

The termination condition is determined using parameters $\{\sigma_0, \dots, \sigma_n\}$, $\{(\epsilon_{max})_0, \dots, (\epsilon_{max})_n\}$, $\{\bar{\epsilon}_0, \dots, \bar{\epsilon}_n\}$.

The program checks if the ICP procedure using $\vec{d}_{k_{sample}}$ returns promising results by checking following condition:

$$1 \geq \frac{\bar{\epsilon}_n}{\bar{\epsilon}_{n-1}} \geq 0.98 \quad \text{for 7 continuous } n$$

OR

$$\sigma_n < 0.0008 \text{ and } \epsilon_{max} < 0.0008 \text{ and } \bar{\epsilon}_0 < 0.0008$$

- If so, the program proceeds to Step 5 and continues ICP using the full group of \vec{d}_k .
- If not so, the program continues ICP from step 1.

Step 5: Termination Check With \vec{d}_k Group

If a desired condition is reached by ICP algorithm when using \vec{d}_k group, then exit iterations. Otherwise, continue ICP.

The termination condition is determined using parameters $\{\sigma_0, \dots, \sigma_n\}$, $\{(\epsilon_{max})_0, \dots, (\epsilon_{max})_n\}$, $\{\bar{\epsilon}_0, \dots, \bar{\epsilon}_n\}$.

Termination Condition:

$$1 \geq \frac{\bar{\epsilon}_n}{\bar{\epsilon}_{n-1}} \geq 0.98 \quad \text{for 7 continuous } n$$

OR

$$\sigma_n < 0.0008 \text{ and } \epsilon_{max} < 0.0008 \text{ and } \bar{\epsilon}_0 < 0.0008$$

Otherwise, continues ICP back from step 1(still using \vec{d}_k)

Algorithm of find closest point

Monday, November 22, 2021 9:14 PM

Simple brutal-force method:

1) Find the closest triangles finding

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

- We assume the point that we are looking at is point a.
- Three vertices of the closest triangle we found are r, p and q.
- To find the closest point c on this closest triangle. We use the following equation:

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

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

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

- Then we compute

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

- 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.

- If $\mu < 0$,

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

- If $\lambda < 0$,

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

- If $\lambda + \mu > 1$,

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

Algorithm of find closest point

Wednesday, November 17, 2021 20:47

Optimized Search Using Bounding Spheres and 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 bounding sphere inequalities:
 - a. $(\vec{b} - \vec{q}) \cdot (\vec{b} - \vec{q}) = (\vec{a} - \vec{q}) \cdot (\vec{a} - \vec{q})$
 - b. $(\vec{c} - \vec{q}) \cdot (\vec{c} - \vec{q}) \leq (\vec{a} - \vec{q}) \cdot (\vec{a} - \vec{q})$
 - c. $(\vec{b} - \vec{a}) \times (\vec{c} - \vec{a}) \cdot (\vec{q} - \vec{a}) = 0$
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 $\lambda \geq \frac{\vec{v}^2 - \vec{u}^2}{2\vec{d} \cdot (\vec{v} - \vec{u})} = \gamma$, if $\gamma \leq 0$ then $\lambda = 0$. Otherwise $\lambda = \gamma$
7. With center \vec{q} , calculate the radius by calculate the 2-norm of $(\vec{q} - \vec{a})$

With a known bounding sphere of a triangle, the program can skip some of closest-point-on-triangle calculations based on previously calculated values. The program only proceed with closest-point-on-triangle calculation only when the following condition is satisfied:

$$||\vec{p} - \vec{q}|| - r < ||\vec{p} - \vec{u}||$$

Where \vec{p} is a point in space, \vec{q} is the center of the bounding sphere, r is the radius of the bounding sphere, and \vec{u} is the closest point found on other triangles.

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)

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.

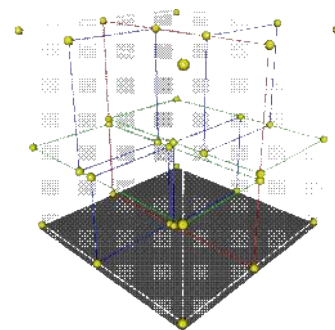


Figure 1

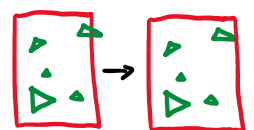


Figure 2

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{s} with the output data
- 2) Verify the \vec{c} with the output data
- 3) Verify the $\|\vec{s}_k - \vec{c}_k\|$ with the output data
- 4) Verify the termination condition

Advanced method with kD-Tree:

- 1) Verify the \vec{s} with the output data
- 2) Verify the \vec{c} with the output data
- 3) Verify the $\|\vec{s}_k - \vec{c}_k\|$ with the output data
- 4) Verify the termination condition

Compare the clever method against the simple brute force search

- 1) Record time consumed by the program to run each file
- 2) Closest point result

Simple Brutal-Force method:

- 1) Verify the \vec{s} with the output data
- 2) Verify the \vec{c} with the output data
- 3) Verify the $\|\vec{s}_k - \vec{c}_k\|$ with the output data

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

| Dataset | Average error in $\vec{s} (s_x, s_y, s_z)$ | Average error in $\vec{c} (c_x, c_y, c_z)$ | Average error in $\ \vec{s}_k - \vec{c}_k\ $ |
|---------|--|--|--|
| A | [0.003, 0.005, 0.002] | [0.003, 0.004, 0.002] | 0.002 |
| B | [0.003, 0.004, 0.002] | [0.003, 0.004, 0.002] | 0.002 |
| C | [0.003, 0.004, 0.003] | [0.003, 0.003, 0.003] | 0.002 |
| D | [0.004, 0.005, 0.003] | [0.004, 0.004, 0.003] | 0.004 |
| E | [0.011, 0.015, 0.011] | [0.012, 0.013, 0.012] | 0.011 |
| F | [0.012, 0.017, 0.009] | [0.011, 0.015, 0.010] | 0.009 |

When terminate:

| Unknown Dataset | σ_n | $(\varepsilon_{max})_n$ | $\bar{\varepsilon}_n$ | Iteration number |
|-----------------|------------|-------------------------|-----------------------|------------------|
| A | 0.00024 | 0.0047 | 0.0016 | 28 |
| B | 0.00406 | 0.0001 | 0.0016 | 85 |
| C | 0.00017 | 0.0058 | 0.00199 | 75 |
| D | 0.00030 | 0.0097 | 0.0035 | 100 |
| E | 0.00487 | 0.1532 | 0.0526 | 37 |
| F | 0.00492 | 0.1596 | 0.0556 | 39 |

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 at most off by low two decimals. Since the output result provided from the debug files are only two decimal result, these error are negligible. The table could prove that our simple brute force method works properly and give the correct result.

Validation and Result

Wednesday, November 17, 2021 19:51

Advanced method with kD-Tree:

- 1) Verify the \vec{s} with the output data
- 2) Verify the \vec{c} with the output data
- 3) Verify the $\|\vec{s}_k - \vec{c}_k\|$ with the output data

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

| Dataset | Average error in $\vec{s} (s_x, s_y, s_z)$ | Average error in $\vec{c} (c_x, c_y, c_z)$ | Average error in $\ \vec{s}_k - \vec{c}_k\ $ |
|---------|--|--|--|
| A | [0.003, 0.005, 0.002] | [0.003, 0.004, 0.002] | 0.002 |
| B | [0.003, 0.004, 0.002] | [0.003, 0.004, 0.002] | 0.002 |
| C | [0.003, 0.004, 0.003] | [0.003, 0.003, 0.003] | 0.002 |
| D | [0.004, 0.005, 0.003] | [0.004, 0.004, 0.003] | 0.004 |
| E | [0.012, 0.015, 0.008] | [0.012, 0.013, 0.006] | 0.008 |
| F | [0.018, 0.022, 0.013] | [0.017, 0.021, 0.013] | 0.010 |

When terminate:

| Unknown Dataset | σ_n | $(\varepsilon_{max})_n$ | $\bar{\varepsilon}_n$ | Iteration number |
|-----------------|------------|-------------------------|-----------------------|------------------|
| A | 0.00467 | 0.0088 | 0.00164 | 26 |
| B | 0.00015 | 0.0042 | 0.00159 | 83 |
| C | 0.00017 | 0.0058 | 0.00199 | 77 |
| D | 0.00030 | 0.0099 | 0.00345 | 51 |
| E | 0.00476 | 0.1525 | 0.05206 | 50 |
| F | 0.00473 | 0.1589 | 0.05338 | 35 |

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 at most off by low 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 consumed by the program to run each file with different method

| Dataset | Time Elapsed Using Simple Brutal Force Method (s) | Time Elapsed Using kDTree (s) | Time Reduced (s) |
|---------|---|-------------------------------|------------------|
| A | 39.247 | 5.011 | 34.236 |
| B | 144.287 | 31.383 | 112.9 |
| C | 146.584 | 24.564 | 122.02 |
| D | 232.166 | 20.816 | 211.35 |
| E | 44.47 | 16.003 | 28.46 |
| F | 85.6 | 16.216 | 69.384 |

Validation and Result

Thursday, December 2, 2021 12:21 AM

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

| Dataset | Difference in $\vec{s} (s_x, s_y, s_z)$ | Difference in $\vec{c} (c_x, c_y, c_z)$ | Difference in $\ \vec{s}_k - \vec{c}_k\ $ |
|---------|---|---|---|
| A | [0.000, 0.000, 0.000] | [0.000, 0.000, 0.000] | 0.000 |
| B | [0.000, 0.000, 0.000] | [0.000, 0.000, 0.000] | 0.000 |
| C | [0.000, 0.000, 0.000] | [0.000, 0.000, 0.000] | 0.000 |
| D | [0.000, 0.000, 0.000] | [0.000, 0.000, 0.000] | 0.000 |
| E | [0.000, 0.000, 0.003] | [0.000, 0.000, 0.006] | 0.003 |
| F | [0.006, 0.005, 0.004] | [0.006, 0.006, 0.003] | 0.001 |

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 around 30 seconds up to several minutes. 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 only negligible 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.

Compare between ICP Using Sampled Points $\vec{d}_{k_{sample}}$ and Full Set of Points

\vec{d}_k :

To **further** reduce the time cost of ICP, the program first sample a portion of points $\vec{d}_{k_{sample}}$, from \vec{d}_k , to excute the early iterations of ICP. When the error of ICP is small enough using $\vec{d}_{k_{sample}}$, the program then switch to \vec{d}_k to ensure a optimal result of ICP. Since $\vec{d}_{k_{sample}}$ contains less points, the program does less computation in early stage of convergence therefore reduces the time cost.

Here we prove that the time cost is reduced using $\vec{d}_{k_{sample}}$ without loosing accuracy

Record time consumed by the program to run each file with differene methods

| Dataset | Time Elapsed Using kD-Tree method W/OUT $\vec{d}_{k_{sample}}$ | Time Elapsed Using kD_Tree method (s) W/ $\vec{d}_{k_{sample}}$ | Time Reduced (s) | Difference in $\ \vec{s}_k - \vec{c}_k\ $ |
|---------|--|---|------------------|---|
| A | 3.913 | 5.011 | -1.504 | $< 10^{-3}$ |
| B | 44.606 | 31.383 | 3.314 | $< 10^{-3}$ |
| C | 50.686 | 24.564 | 3.355 | $< 10^{-3}$ |
| D | 39.727 | 20.816 | 3.298 | $< 10^{-3}$ |
| E | 23.904 | 16.003 | 3.296 | 0.002 |
| F | 24.958 | 16.216 | 3.248 | 0.002 |

Discussion:

$\vec{d}_{k_{sample}}$ is uniformly randomly sampled from set \vec{d}_k . The formmer's size is about 1/3 or 1/4 of the latter's. As shown in the table above using smaller sampled point set $\vec{d}_{k_{sample}}$ at early ICP iterations significantly reduces the time cost of the program without loosing accuracy especially in complex cases that needs longer to converge. There is a decrease of performance in data set A because d_k is very close to the mesh in data set A before the ICP algorithm, causing the use of $\vec{d}_{k_{sample}}$ to have a negative impact on the performance.

A tabular summary of the results obtained for unknown data

Saturday, November 13, 2021 7:17 PM

Results obtained for unknown data

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

| Unknown Dataset | Linear Search Time (s) | Linear Search iteration | kD-tree method Time (s) | kD-tree iteration | Average $\ \vec{s}_k - \vec{c}_k\ $ |
|-----------------|------------------------|-------------------------|-------------------------|-------------------|-------------------------------------|
| G | 122.947 | 68 | 19.034 | 79 | 0.0032 |
| H | 140.397 | 79 | 33.642 | 98 | 0.0023 |
| J | 93.164 | 40 | 11.990 | 35 | 0.0630 |

When terminate:

| Unknown Dataset | σ_n | $(\epsilon_{max})_n$ | $\bar{\epsilon}_n$ |
|-----------------|------------|----------------------|--------------------|
| G | 0.00027 | 0.0088 | 0.0031 |
| H | 0.00019 | 0.0058 | 0.0021 |
| J | 0.00557 | 0.1796 | 0.0630 |

Termination Reason:

| Unknown Dataset | Terminate Reason |
|-----------------|---|
| G | $1 \geq \frac{\bar{\epsilon}_n}{\bar{\epsilon}_{n-1}} \geq 0.98$ for 8 continuous n |
| H | $1 \geq \frac{\bar{\epsilon}_n}{\bar{\epsilon}_{n-1}} \geq 0.98$ for 8 continuous n |
| J | $1 \geq \frac{\bar{\epsilon}_n}{\bar{\epsilon}_{n-1}} \geq 0.98$ for 8 continuous n |

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 should give us the same result. As shown in previous sections, the average $\|\vec{s}_k - \vec{c}_k\|$, σ_n , $(\epsilon_{max})_n$, $\bar{\epsilon}_n$ are all very small, which indicates that our ICP works properly. Thus, we believe that our result for unknown dataset should be correct. And the time by using advanced method(kd-Tree) has largely reduced time cost as shown in the tables above. Thus we think our result for unknown dataset should be correct.