# Programming Assignment 5 – 601.455/655 Fall 2022

**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 | Jiaming Zhang. |
| Email | jzhan182@jh.edu. |
| Other contact information (optional) | |
| Name 2 | Jiawei Liu |
| Email | jliu298@jh.edu |
| Other contact information (optional) | |
| Signature (required) | I (we) have followed the rules in completing this assignment<br>Jiaming Zhang<br>Jiawei Liu |

| **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 | |

**NOTE: This is an optional assignment.**

If you hand it in, I will use the grade to replace the lowest other programming assignment or written homework assignment with one exception:

You may not drop **both** HW#4 and HW#5. If these are your two lowest grades, then I will drop the lower of those two under the drop 1 homework scenario and replace the next lowest grade (other than the other of HW#4-5) with this score


# Computed Integrated Surgery I

## Programming Assignment 5

**AUTHOR: JIAMING ZHANG**
**JIAWEI LIU**

# I. Math Approach

### *1.1 Frame Transformation*

The frame transformation method is used to compute the multiple relationships of specific points and frames. It can give the specific point coordinate in different frames, since each frame can be represented as a homogenous matrix which includes rotation and translation.

$$\begin{aligned}\mathbf{F}_1 \bullet \mathbf{F}_2 &= [\mathbf{R}_1, \vec{\mathbf{p}}_1] \bullet [\mathbf{R}_2, \vec{\mathbf{p}}_2] \\ &= [\mathbf{R}_1 \bullet \mathbf{R}_2, \mathbf{R}_1 \vec{\mathbf{p}}_2 + \vec{\mathbf{p}}_1]\end{aligned}$$

**Fig.1 Frame multiplication functions(Taylor,2022)**

As shown in **Fig.1**, the transformation between two frames can be represented as an inner product of two homogeneous matrices. For frame inverse transformation and point transformation, the function shown in **Fig.2** and **Fig.3**.

$$\mathbf{F}^{-1} = [\mathbf{R}^{-1}, -\mathbf{R}^{-1} \bullet \vec{\mathbf{p}}]$$

**Fig.2 Inverse Frame transformation(Taylor,2022)**

$$\begin{aligned}\vec{\mathbf{v}} &= \mathbf{F} \bullet \vec{\mathbf{b}} \\ &= [\mathbf{R}, \vec{\mathbf{p}}] \bullet \vec{\mathbf{b}} \\ &= \mathbf{R} \bullet \vec{\mathbf{b}} + \vec{\mathbf{p}}\end{aligned}$$

**Fig.3 Frame transformation(Taylor,2022)**

### *1.2. Point Cloud Registration*

Point Cloud registration algorithm is a widely used method to registrate two different frame who has several coordinate clear points. The main theorem is finding the centroid of two frames, then compute the orientation and frame transformation between them based on the spatial frame.

Step 1: Compute

$$\bar{\mathbf{a}} = \frac{1}{N}\sum_{i=1}^{N} \vec{\mathbf{a}}_i \qquad \bar{\mathbf{b}} = \frac{1}{N}\sum_{i=1}^{N} \vec{\mathbf{b}}_i$$

$$\tilde{\mathbf{a}}_i = \vec{\mathbf{a}}_i - \bar{\mathbf{a}} \qquad \tilde{\mathbf{b}}_i = \vec{\mathbf{b}}_i - \bar{\mathbf{b}}$$

**Fig.4 Calculation the centroid of local frames (Taylor,2022)**

By using the SVD algorithm, we could compute the rotation matrix

Step 1: Compute

$$\mathbf{H} = \sum_i \begin{bmatrix} \tilde{a}_{i,x}\tilde{b}_{i,x} & \tilde{a}_{i,x}\tilde{b}_{i,y} & \tilde{a}_{i,x}\tilde{b}_{i,z} \\ \tilde{a}_{i,y}\tilde{b}_{i,x} & \tilde{a}_{i,y}\tilde{b}_{i,y} & \tilde{a}_{i,y}\tilde{b}_{i,z} \\ \tilde{a}_{i,z}\tilde{b}_{i,x} & \tilde{a}_{i,z}\tilde{b}_{i,y} & \tilde{a}_{i,z}\tilde{b}_{i,z} \end{bmatrix}$$

NOTE well

Step 2: Compute the SVD of $\mathbf{H} = \mathbf{USV}^t$

Step 3: $\mathbf{R} = \mathbf{VU}^t$

Step 4: Verify $Det(\mathbf{R}) = 1$. If not, then algorithm may fail.

**Fig.5 SVD solving rotation matrix R**

Then translation can be easily get by frame transformation

Step 3: Find $\vec{\mathbf{p}}$

$$\vec{\mathbf{p}} = \bar{\mathbf{b}} - \mathbf{R} \cdot \bar{\mathbf{a}}$$

Step 4: Desired transformation is

$$\mathbf{F} = Frame(\mathbf{R}, \vec{\mathbf{p}})$$

**Fig.6 Point cloud frame transformation**

By then, we got the frame transformation between two local frames based on spatial frames.

### *1.3. Find Closest Point*

Find Closest Point is a kind of approach to closest triangle finding, it searches sequentially in a linear list of triangles for the closest triangle to each point.

As the **Fig.7** shown, **a** is a point out of the triangle, in order to find the closest point **c** within the triangle.

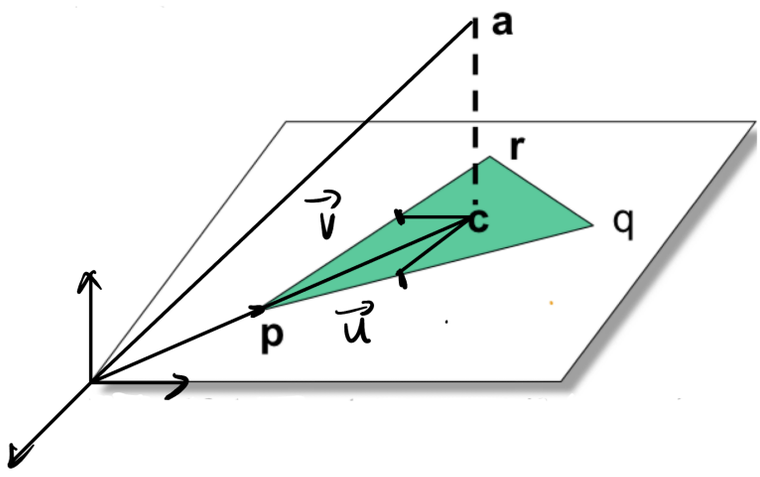



**Fig.7 Find closest point**

let **v**=mu(**r**-**p**), **u**=lam(**q**-**p**)

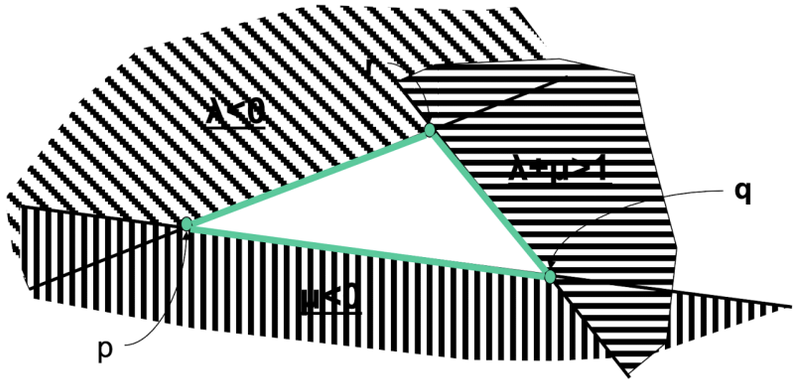
assume (**c**-**p**)=**v**+**u**

As (**a**-**p**)≈lam(**q**-**p**)+mu(**r**-**p**)

Thus, by solving this least square, we got lam and mu. Hence we can compute **c**

**c**=**p**+lam(**q**-**p**)+mu(**r**-**p**)

From the **Fig.8**, we need to judge whether the closest point is inside of the triangle by comparing the value of lam and mu. Therefore, If λ>=0, μ>=0, λ+μ<=1, **c** lies within the triangle and is the closest point.




| Region | Closest point |
|---|---|
| λ<0 | *ProjectOnSegment(***c,r,p***)* |
| μ<0 | *ProjectOnSegment(***c,p,q***)* |
| λ+μ>1 | *ProjectOnSegment(***c,q,r***)* |

**Fig.8 Region and ProjectOnSegment**

if the region of **c** is outside of the triangle bounder, if the region lam<0, the input vertex points of the ProjectOnSegment is (c,r,p), if mu<0, the input vertex points of the ProjectOnSegment is (c,p,q), if lam+mu>1, the input vertex points of the ProjectOnSegment is (c,q,r).

Substitute the vertex coordinate to compute new lam and lam(seg)

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

$$\lambda^{(seg)} = Max(0, Min(\lambda, 1))$$

$$\mathbf{c}^{*} = \mathbf{p} + \lambda^{(seg)} \times (\mathbf{q}-\mathbf{p})$$

Therefore, we got the actual closest point **c*** which is the closest point we aimed to find.

### *1.4. Bounding Sphere*

Bounding Sphere is a kind of algorithm which is used to improve the search speed.

suppose you have a point **p** and try to find the matching closest triangle(ak,bk,ck) to **p**. and the previous closest point **rj.** Check if it satisfied the criterion

$$\left\|\vec{\mathbf{p}}-\vec{\mathbf{q}}_k\right\|-\rho_k<\left\|\vec{\mathbf{p}}-r_j\right\|.$$

if not, return the previous closest point and check the next bounding sphere. Otherwise, search this bounding sphere to check if there is a point within the current triangle mesh. First, compute **f**=(**a**+**b**)/2. Define

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

Compute the sphere center q

$$\vec{\mathbf{q}}=\vec{\mathbf{f}}+\lambda\vec{\mathbf{d}}$$
$$\lambda\geq\frac{\vec{\mathbf{v}}^2-\vec{\mathbf{u}}^2}{2\vec{\mathbf{d}}\bullet\left(\vec{\mathbf{v}}-\vec{\mathbf{u}}\right)}=\gamma$$

If gamma<=0, then just pick lam=0.Else pick lam=gamma.

### *1.6. OCtree*

The OCtree is another kind of data structure to speed up the searching speed, the main idea is to separate the whole 3-D structure bounded by a cube. Then we should compute the median of the vertex points as the syncopated center to separate the cube to 8 subcubes in space.

- Step1. Compute the median value of whole vertex points.
- Step2. Separated the cube into 8 subcubes.
- Step3. Compute the mean value of each subcube, check the distance between the corresponding pointing point with mean value points. Select the subcube which is closest to the it.
- Step4. Iterate the above steps until the last cube only contains one mesh triangle.

### *1.7. ICP Algorithm*

ICP is a kind of algorithm method to match point clouds under the same frame and find the optimal matching solution. Generally, this method could be used to find the point pairs and match them together. in the target point cloud P and source point cloud Q with matching, the nearest point (pi,qi) if found according to certain constraints, and then the optimal matching parameters R and t are calculated to minimize the error function.

step1 Start with an initial guess **R0**, for **R**.

step2 At iteration k

1. For each sampled point **qi** belong to point cloud **Qi**, find the point **pi** belong to point cloud **Pi** that is closest to **Rkqi**
2. Then compute **Rk+1** as the transformation that minimizes

$$D_{k+1} = \sum_i \left\| \mathbf{v}_i - \mathbf{T}_{k+1} \cdot \mathbf{f}_i) \right\|^2$$

3. Physical Analogy

step3 transformation update

1. n ← n + 1
2. Fn ← FindBestRigidTransformation(A,B)
3. Transformation update

Since the closest points cloud after registration, has't matched the mesh point cloud perfectly, Hence, we hold the points cloud, $c(k + 1) = F * ck$, after registration, for c(k+1), we could find another closest points cloud sk located on matching surface, where sk=$\Delta F * c(k + 1)$. Hence the Freg can be represented as

$$Freg = F * \Delta F$$

step4 Termination Test

After several updates of ck and Freg, set tolerance of iteration as the termination test condition.

$$Tolerance = |Freg(k + 1) - Fregk|$$

if the tolerance is smaller then 0.01, return Fregk as the orientation and translation of the matching frame transformation Freg.

# II. Algorithm Approach

***2.1 Scenario approach***

In this scenario, we are given a 3-D CT scan model with triangle mesh corresponding to the true bone. The bone was screwed with a probe fixed with LED markers respected to the optical tracker. By processing the CT imaging, we built the triangle mesh 3-D model, with the vertex coordinates and triangle index data.

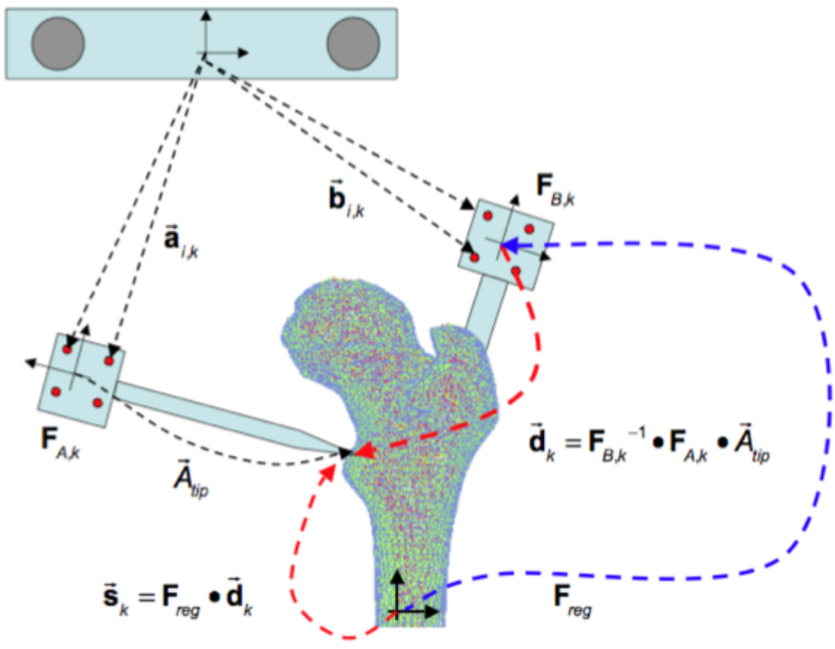



**Fig.10 Scenario chart**

Since the bone was scanned with probe B, thus, the orientation and translation between bone and probe was clear, which means the probe can be shown on the CT scan and doing frame transformation together with imaging.

As the graph shows, The optical tracker can read the marker's positions ai,k and bi,k, hence, the frames of probes FA,k and FA,k can be defined. Also for probe A, Since the length of the probe has already been measured, vector Atip, which indicates the probe's pointing place, can be early delivered based on probe's frame FA,k. Thus, the transform relationship dk between local frame B and the point which pointed by the probe A can be known in real-time, where

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

For the frame transformation between local frame FB,k and CT frame, which is represented by Freg as shown in **Fig.10**. vector ck indicates the position of the pointing point in the CT scan frame, which corresponds to sk in real actual coordinates. We assume the initial guessed frame transformation Freg=I,

$$\vec{s}_k = F_{reg} \bullet \vec{d}_k$$

Therefore, we can get sk, where sk-ck illustrates the matching error between the pointing point on the bone and the matching mesh in the CT scan.

In order to find pairs of pointing points and CT mesh, we need to match every mesh to the corresponding relative closest pointing point. Thus, FindClosestPoint algorithm is applied to every pointing point coordinate to find the matching closest mesh. According to the given data files, each pointing point and triangle vertices coordinates have been indicated.

- step1. Label pointing point's coordinate as dk, triangle mesh vertices as pi, qi and ri, the closest point as ci.
- step2. Compute the **v**=mu(**r-p**), **u**=lam(**q-p**), (**c-p**)=**v+u**
- step3. Solve the least square (**a-p**)≈lam(**q-p**)+mu(**r-p**)
- step4. Hence, we got the closest point coordinate **c**=**p**+lam(**q-p**)+mu(**r-p**)
- step5. Judge if the lam>=0, mu>=0 and lam+mu<=1, if it's true, then the closest within the triangle mesh, which has been solved in step4. If not we need to check the region from the **Table. 1** to decide the input points of the PojectOnSegment.

**Table. 1 Region and ProjectOnSegment**

| Region | Closest point |
|---|---|
| **λ<0** | *ProjectOnSegment(**c,r,p**)* |
| **μ<0** | *ProjectOnSegment(**c,p,q**)* |
| **λ+μ>1** | *ProjectOnSegment(**c,q,r**)* |

- step6. Compute the new lam and lam(seg) using the selected three point in step5

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

$$\lambda^{(seg)} = Max(0, Min(\lambda, 1))$$

- step7. By using the lam(seg), compute the c true coordinate, which is the closest point we aimed to find.

$$\mathbf{c}^{*} = \mathbf{p} + \lambda^{(seg)} \times (\mathbf{q}-\mathbf{p})$$

Nevertheless, the linear search algorithm has to search every triangle mesh, we need to search 3135 triangle meshes and find the closest point in everyone, which means a huge amount of calculation.

### *2.2 Speed up methodes.*

In order to implement the most suitable algorithm, we tried 3 different speed up methodes, which are Bounding sphere, Covariance tree, OC tree.

#### *2.2.1 The Bounding Sphere*

Aim to reduce the calculated time, we need to implant data structure to speed up the whole code running speed. The Bounding Sphere algorithm can be a good solution to process the data and increase the searching speed.

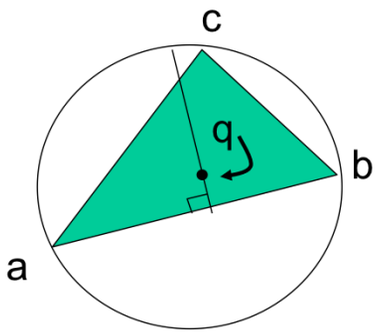



**Fig.11 Bounding sphere**

- Step1. Label the pointing points pi, and the closest triangle coordinate(ak, bk, ck) to **p**. the previous closest point rj.
- Step2. check if the distance between pointing point and the sphere is smaller then the distance between pointing point and previous closest. If it's true, search the triangle mesh inside of the current bounding sphere. If not, return the previous closest point and check the next bounding sphere.

$$\left\|\vec{\mathbf{p}}-\vec{\mathbf{q}}_k\right\|-\rho_k<\left\|\vec{\mathbf{p}}-r_j\right\|.$$

- Step3. For the sphere who satisfied the criterion, assume edge(a,b) is the longest side of the triangle. Compute f=(a+b)/2, define

$$\vec{\mathbf{u}}=\vec{\mathbf{a}}-\vec{\mathbf{f}};\vec{\mathbf{v}}=\vec{\mathbf{c}}-\vec{\mathbf{f}}$$

$$\vec{\mathbf{d}}=(\vec{\mathbf{u}}\times\vec{\mathbf{v}})\times\vec{\mathbf{u}}$$

  then determine if the lam satisfied the following condition

$$\lambda\geq\frac{\vec{\mathbf{v}}^2-\vec{\mathbf{u}}^2}{2\vec{\mathbf{d}}\bullet\left(\vec{\mathbf{v}}-\vec{\mathbf{u}}\right)}=\gamma$$

  if gamma <=0, then just pick lam=0. Else pick lam=gamma.
- step4. compute the sphere center q lies somewhere along the line.

$$\vec{\mathbf{q}}=\vec{\mathbf{f}}+\lambda\vec{\mathbf{d}}$$

Then combine the linear search and bounding sphere together to increase the searching speed.

- step1. If it satisfied the following criterion

$$\left\|\vec{\mathbf{p}}-\vec{\mathbf{q}}_k\right\|-\rho_k<\left\|\vec{\mathbf{p}}-r_j\right\|.$$

- step2. compute the closest point the distance h, check the determination **h-a<p-rj**. If true, then **c**=**h**, update the bound as **h-a**.

### *2.2.3 OCtree*

The OCtree is another kind of data structure to speed up the searching speed, the main idea is to separate the whole 3-D structure bounded by a cube. Then we should compute the median of the vertex points as the syncopated center to separate the cube to 8 subcubes in space. The significant point is we shouldn't compute the mean point as a syncopated center, since if the

vertex points are not uniform, most of the points may be squeezed together which may cause the subcubes to be unbalanced.

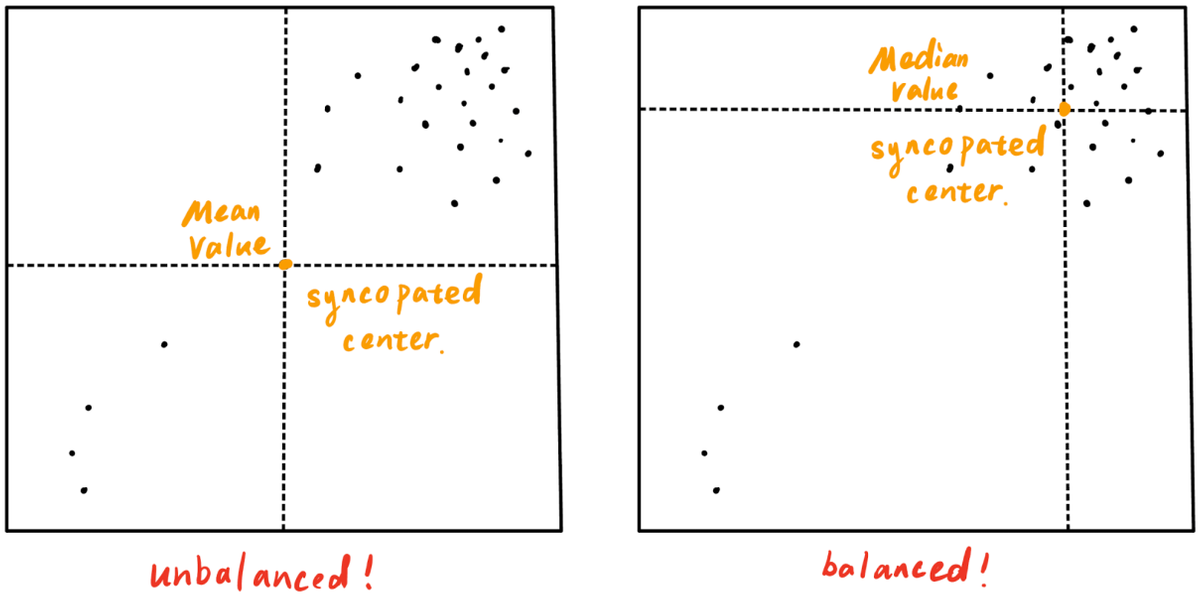



**Fig.14 The difference of choosing the syncopated center.**

In each subcube, calculate the mean value of each cube, compare the distance between each mean point with the corresponding pointing point P, Select the closest one, the closest matching point should be inside the corresponding box, then we focus on this leaf, separate it into 8 leaves, iterate the above steps, until the last leaf only contain one point.

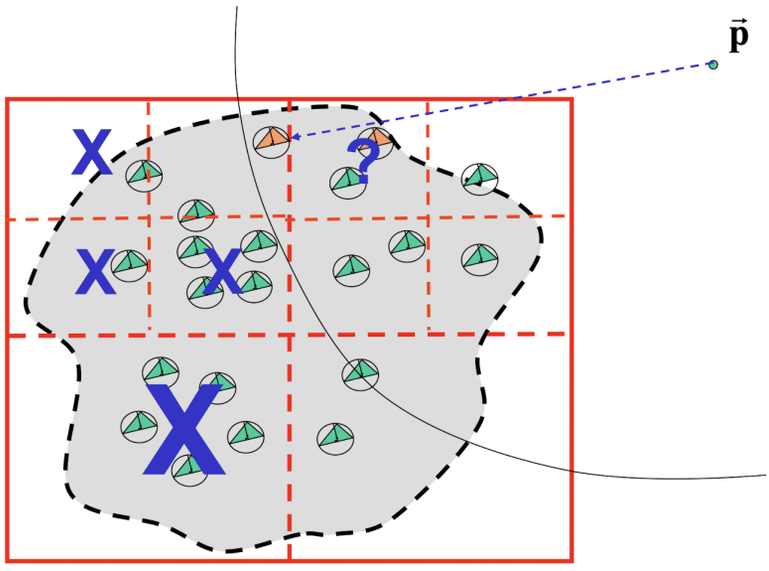



**Fig.15 The chart of OC tree**

***2.3 ICP matching and Iterations.***

After combining the linear search and bounding sphere and finding one of the pointing points of the dk, ICP is used to match every point to the corresponding triangle meshes. Hence, we matched all given triangle mesh with corresponding pointing points.

First, as we had already got the closest point ck at initial state, thus we could finish the first matching step by

$$sk = F * ck$$

Since the closest points cloud after registration, has't matched the mesh point cloud perfectly, Hence, we hold the points cloud, $c(k + 1) = F * ck$, after registration, for c(k+1), we could find another closest points cloud sk located on matching surface, where sk=$\Delta F * c(k + 1)$. Hence the Freg can be represented as

$$Freg = F * \Delta F$$

After several updates of ck and Freg, set tolerance of iteration as the termination test condition.

$$Tolerance = |Freg(k + 1) - Fregk|$$

if the tolerance is smaller then 0.01, return Fregk as the orientation and translation of the matching frame transformation Freg.

For PA5 the mesh triangles are deformable, the vertex points are different in 6 models. Suppose the vertex indices of this triangle are $\{s, t, u\}$. The corresponding coordinates of the deformed vertices can be represented as

$$\vec{\mathbf{m}}_s = \vec{\mathbf{m}}_{0,s} + \sum_{m=1}^{N\text{modes}} \lambda_m^{(t)} \vec{\mathbf{m}}_{m,s}$$

$$\vec{\mathbf{m}}_t = \vec{\mathbf{m}}_{0,t} + \sum_{m=1}^{N\text{modes}} \lambda_m^{(t)} \vec{\mathbf{m}}_{m,t}$$

$$\vec{\mathbf{m}}_u = \vec{\mathbf{m}}_{0,u} + \sum_{m=1}^{N\text{modes}} \lambda_m^{(t)} \vec{\mathbf{m}}_{m,u}$$

Therefore, combined with the barycentric coordinates, we could compute the closest point qk.

$$\vec{\mathbf{q}}_{m,k} = \zeta_k \vec{\mathbf{m}}_{m,s} + \xi_k \vec{\mathbf{m}}_{m,t} + \psi_k \vec{\mathbf{m}}_{m,u}$$

Also for ck, which equals the closest point q0,k at model 0 pulsed with some distortion qm,k.

$$\vec{\mathbf{c}}_k^{(t)} = \vec{\mathbf{q}}_{0,k} + \sum_{m=1}^{N_{\text{modes}}} \lambda_m^{(t)} \vec{\mathbf{q}}_{m,k}$$

Also for the we got sk and dk, hence we can use the ICP algorithm to do iteration and deliver the Freg.

$$\vec{\mathbf{s}}_k^{(t)} = \mathbf{F}_{reg}^{(t)} \bullet \vec{\mathbf{d}}_k$$

Hence, we got the equation

$$\mathbf{F}_{reg}^{(t)} \bullet \vec{\mathbf{d}}_k \approx \vec{\mathbf{q}}_{0,k} + \sum_{m=1}^{N_{\text{modes}}} \lambda_m^{(t)} \vec{\mathbf{q}}_{m,k}$$

By solving this least square equation,

$$\begin{bmatrix} & \cdots & & \\ q1 & q2 & q3 & \cdots & qn \\ & \cdots & & \end{bmatrix} \cdot \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \cdots \\ \lambda_n \end{bmatrix} = \begin{bmatrix} \cdots \\ F_{reg}d_k - q_0 \\ \cdots \end{bmatrix}$$

we can compute the weight and Freg.

$$\Lambda^{(t)} = \left\{\lambda_1, \cdots, \lambda_{N_{modes}}\right\}^{(t)}$$

As we know the weight, substitute it into the vertex coordinates, hence, we could iterate the mesh triangles' vertices and the closest points's coordinates, and improve the Freg.

# III. Overview of Program Structure

### *3.1 Overview of Program Structure*

There are 4 major parts for our program, including Input data, PROGRAMS, Output and Utility functions. We group all of these four parts as a zip file and that is what we submit.

```
└── PROGRAMS
    ├── PA5              # Test scripts are contained in this directory
    │   ├── Data         # This DIR contains all the provided data
    │   ├── output       # This DIR contains all the results of our program
    │   └── pa5_main.py      # Main script that
    └── cispa                                  # Utility Functions are in this directory
        ├── CarteFrame.py
        ├── ComputeExpectValue.py
        ├── CorrectDistortion.py
        ├── DataProcess.py
        ├── FindBoundingSphere.py
        ├── FindClosestPoint2Mesh.py       # Find the closest point on the mesh to point
        ├── FindClosestPoint2Triangle.py
        ├── HomoRepresentation.py
        ├── Octree.py
        ├── PivotCalibration.py
        ├── IterClosestPoint.py              # ICP Implementation
        ├── DeformICP.py                     # Perform Deformable ICP Implementation
        └── Registration.py
```

**Fig.16 Overview of Program Structure Chart**

### *3.2 Input Data*

Under the folder named “PA5/Data”, there are all files we use to debug and generate the output. Directly import from PA5 dataset.

### *3.3 Program*

Under the folder named “PROGRAMS”, there are 1 py script which is used to compute our Freg for PA5 and Readme.txt file which contains how to run our program and all the explanation for every py file in this folder. In the above **Fig.16**, we have already explained the functioning for each file.

We got all the data reading from the “PA5/Data” folder using DataProcess.py. After getting all the input data, we instantialize a DeformICP class to interatively solve the deformation mode coefficients lambda and update the mesh accordingly. First, we compute the initial guess about the transformation between the given tip point and the initial mesh using the ICP algorithm we developed in PA4. Afterwards, we update the mesh and use the updated mesh to again apply ICP to update the corresponding transformation. Repeat the above steps and set a proper terminal condition will give out a relatively good result about the deformable transformation.

### *3.4 Output*

Under the folder named “Output”, we stored all the results of our output files generated for filename from A to K. For the dataset from A-F, we have compared our result with the given “Answer” and our error between sk and ck is slightly bigger than the standard result. For the dataset G,H,J,K, we have saved the output to prepare for future inspection.

# IV. Verification of Program

### *4.1 Verify by Testing with Debug Sets*

After we developed the Iterative Closest Point algorithm, we need to test the errors between the output in the given debug files and the output we generated. Hence, we visualized computed results and compared them with the corresponding output debug files(A,B,C,D,E,F) by calculating the mean error, which are shown as the **Table.2**.

**Table.2 The mean error of our error and debug file error.**

| DataSet | Our method - Mean Error | Answer - Mean Error |
|---|---|---|
| A | 0.0236 | 0.000 |
| B | 0.0257 | 0.000 |
| C | 0.0314 | 0.000 |
| D | 0.0176 | 0.000 |
| E | 0.1979 | 0.097 |

| | | |
|---|---|---|
| F | 0.2080 | 0.114 |

As **Table.2** showed, the error of our implementation is greater than the given answer. We will discuss the reason in the Section V.

The code is simply running by the following command:

```
(cispa) jeremy ProgramAssignment main ± python3 PA5/pa5_main.py -n PA5-E-Debug
(1568, 3)
(400, 3)
-----------------------------
Iteration:  0
Max Error:  9.366169479492118
Sigma:  0.007896264570111054
Eps_bar:  2.993381179933183
Convergence Flag:  False
(399, 3)
-----------------------------
Iteration:  1
Max Error:  8.450062944710456
Sigma:  0.007003700762947228
Eps_bar:  2.4027317858475867
Convergence Flag:  False
(398, 3)
-----------------------------
```

… intermediate output…

And the results are as following:

```
-------------------------------
[23:41:44] INFO     Deformed ICP time using Octree=                    pa5_main.py:136
                    238.60326194763184 seconds
           INFO     Deformed ICP transformation matrix :               pa5_main.py:137
                     R = [[ 0.99676096 -0.06048789  0.0529981 ]
                     [ 0.06421865  0.99535135 -0.07177499]
                     [-0.04841022  0.07494598  0.99601182]]
                     t = [[0.26613546]
                     [0.09215843]
                     [0.4250458 ]]
```

Here we visualize the corresponding point pairs before and after computation for dataset PA5-A-Debug.

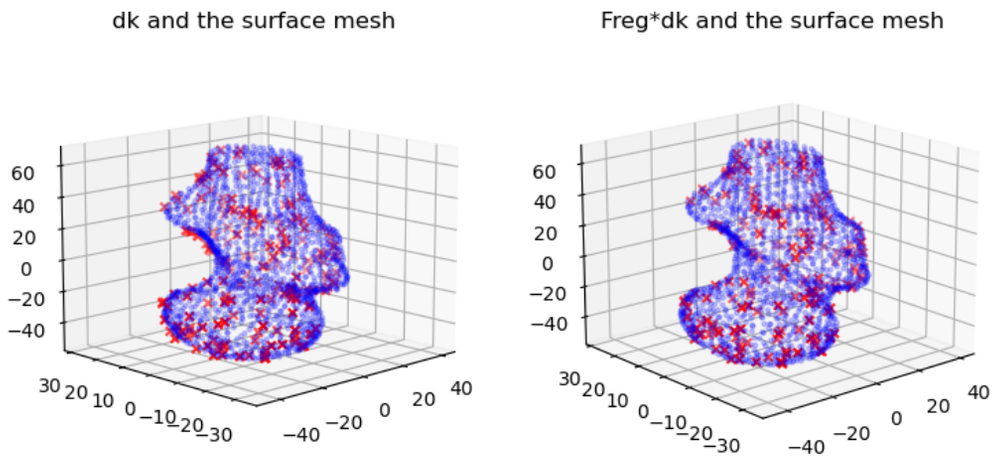



**Fig 17. The comparison of the original mesh between after ICP.**

# V. Result and Discussion

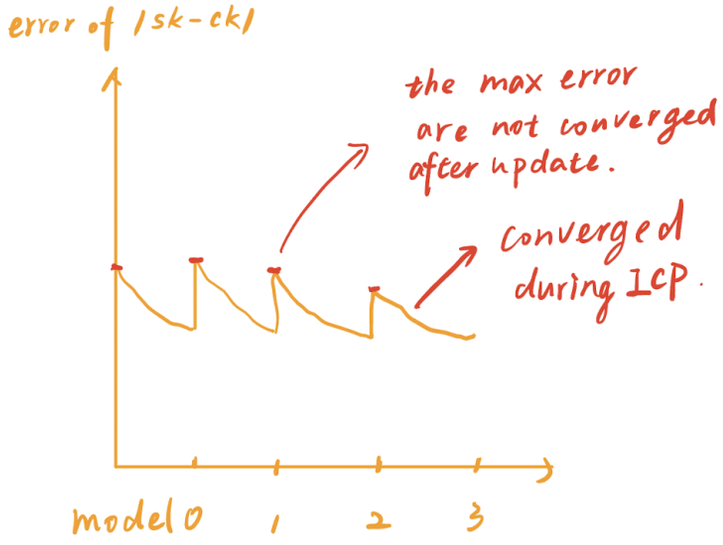
### *5.1 Discussion of Results*

- By comparing the actual value(debug files), we analyzed errors and came up with a possible affection of the results. Since all of the errors mainly begin showing after the percentile, Therefore, we believed the error may have come for the numerical rounding of the final results. As we round off the numbers after the percentile, hence, it affected the result value. For this scenario, the unit of each coordinate is millimeter, the percentile perspective error is so small that we can completely ignore it.
- As the output shown, our Freg have some error with the ground truth, After analyze, we suspect it may be caused by the update method of the vertice coordinates. From the scenario, the the vertices update should obey the given formulas as following

$$\vec{\mathbf{m}}_s = \vec{\mathbf{m}}_{0,s} + \sum_{m=1}^{Nmodes} \lambda_m{}^{(t)} \vec{\mathbf{m}}_{m,s}$$
$$\vec{\mathbf{m}}_t = \vec{\mathbf{m}}_{0,t} + \sum_{m=1}^{Nmodes} \lambda_m{}^{(t)} \vec{\mathbf{m}}_{m,t}$$
$$\vec{\mathbf{m}}_u = \vec{\mathbf{m}}_{0,u} + \sum_{m=1}^{Nmodes} \lambda_m{}^{(t)} \vec{\mathbf{m}}_{m,u}$$

  The formulas indicated it's should plus with the vertices' coordinates under model 0, However, we are about the symbol of the correct term $\lambda m$. If it obeys the formulas, it will affect the results of sk-ck, which couldn't converge after we iterated several times. But if we change it to a negative term, which means using the coordinates at model 0 to minus the correct term, the results of Freg could match our expected convergence. Hence, we thought that could be the source of the error.

- For the termination test at calculation of λ, after we updated the vertex coordinates, the error of sk-ck would also change. However, With the updating, the error of sk-ck is not converging, it is always jumping around as the **Fig** shown, which means we couldn't set a specific stopping condition for the output of the sk-ck




- 

**Fig 18. The error jumps after every update.**

Hence, we set a stopping time for the iteration, after 6 times, break the loop and output results. It could be a problem affecting the accuracy.

### *5.2 Summary of Results for Unknown Data*

| 150, PA5-G-Unknown-Output.txt | | | | | | |
|---|---|---|---|---|---|---|
| lambda | -7.1196 | 30.1401 | -16.3410 | -6.8035 | -34.7362 | 8.6751 |
| … | | | | | | |
| 33.2342 | 21.3315 | -16.9266 | 33.3008 | 21.2836 | -16.9427 | 0.0836 |
| … | | | | | | |

| 150, PA5-H-Unknown-Output.txt | | | | | | |
|---|---|---|---|---|---|---|
| lambda | 4.4780 | -18.4939 | 6.7814 | -22.3074 | 33.2305 | -0.7161 |
| … | | | | | | |
| -9.6142 | 30.0447 | -32.5286 | -9.5405 | -30.0026 | -32.6094 | 0.1171 |
| … | | | | | | |

| 150, PA5-J-Unknown-Output.txt | | | | | | |
|---|---|---|---|---|---|---|
| lambda | 13.8920 | -10.7430 | 7.1381 | 8.8698 | 2.7237 | -2.7372 |
| … | | | | | | |
| 9.8934 | -3.2893 | 64.2173 | 9.9089 | 9.9089 | -3.5031 | 0.2145 |
| … | | | | | | |

| 150, PA5-K-Unknown-Output.txt | | | | | | |
|---|---|---|---|---|---|---|
| lambda | -9.8757 | 4.4090 | 0.6432 | -10.5327 | 15.3431 | -7.3362 |

| … | | | | | | |
|---|---|---|---|---|---|---|
| 30.9428 | 22.1934 | 0.8542 | 30.9218 | 22.1913 | 0.7990 | 0.0591 |
| … | | | | | | |

# VI. Teamwork Contribution

Jiaming Zhang: Writing programming coding part and the overview of structure part in the report.

Jiawei Liu: Writing almost all of the report and also contributes to the coding part.

# VII. Reference


[1]Taylor, R. (2022) Calibration p.10.
https://ciis.lcsr.jhu.edu/doku.php?id=courses:455-655:2022:fall-2022-schedule
[2]Taylor, R. (2022) Basic Mathematical Methods for CIS p.22.
https://ciis.lcsr.jhu.edu/doku.php?id=courses:455-655:2022:fall-2022-schedule
[3]Taylor, R. (2022) Basic Mathematical Methods for CIS p.21.
https://ciis.lcsr.jhu.edu/doku.php?id=courses:455-655:2022:fall-2022-schedule
[4]Taylor, R. (2022) Finding point pairs for Iterative Closest Point algorithms p11,12,17,18,19,20
https://ciis.lcsr.jhu.edu/doku.php?id=courses:455-655:2022:fall-2022-schedule
[5]Taylor, R. (2022) Registration-part1 p71,72,73,74
https://ciis.lcsr.jhu.edu/doku.php?id=courses:455-655:2022:fall-2022-schedule
[6]Taylor, R. (2022) Hand out programming assignment #3 and #4
https://ciis.lcsr.jhu.edu/doku.php?id=courses:455-655:2022:fall-2022-schedule