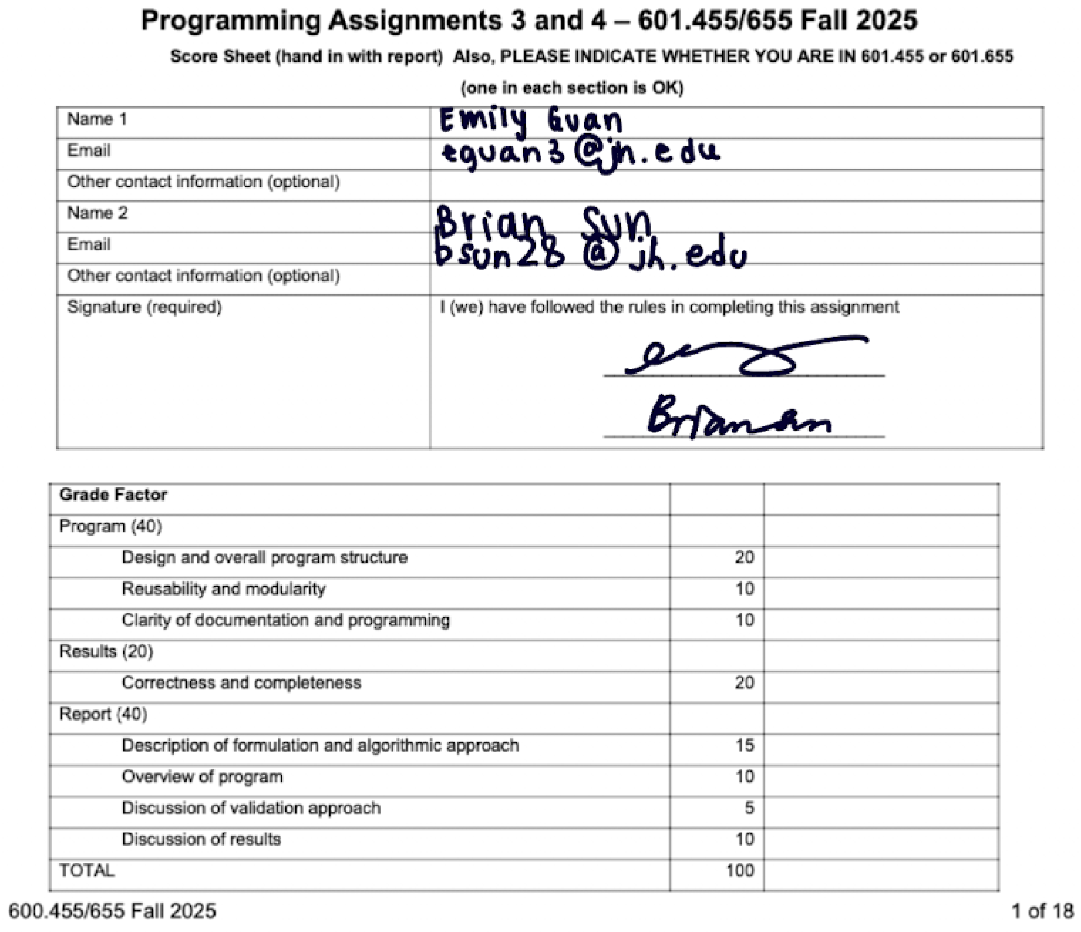
CIS PHW 3

Emily Guan, Brian Sun

November 2025



Programming Assignments 3 and 4 – 601.455/655 Fall 2025

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	Emily Guan
Email	eguan3@jh.edu
Other contact information (optional)	
Name 2	Brian Sun
Email	bsun28@jh.edu
Other contact information (optional)	
Signature (required)	I (we) have followed the rules in completing this assignment [illegible] [illegible]

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	

600.455/655 Fall 2025 1 of 18

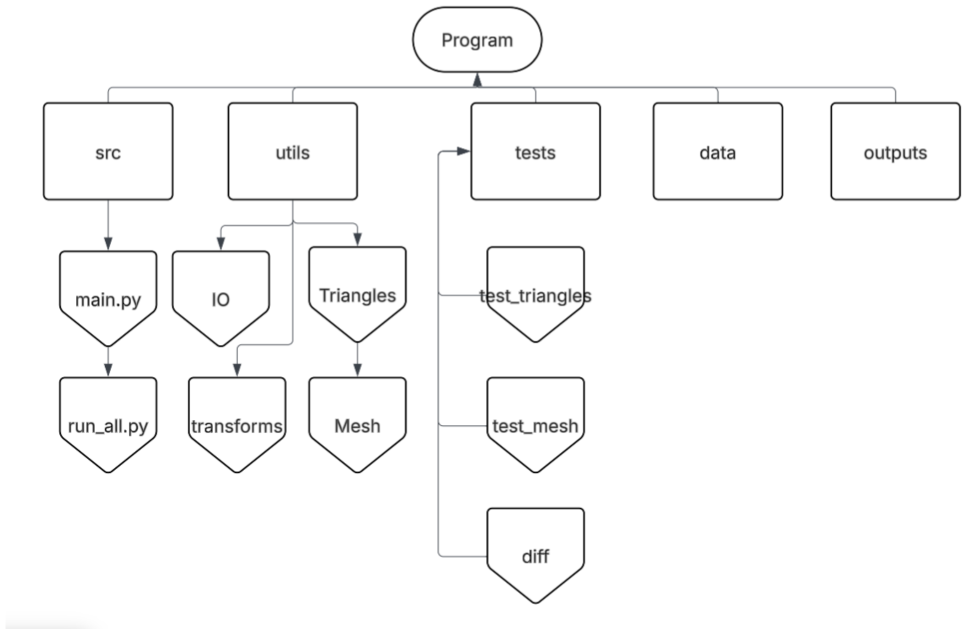
1 Contributions

Report: 1, 3-5 (Brian Sun), 2/6 (Emily Guan)
Code: Emily Guan

2 Summary

In this assignment, we implement a Iterative Closest Point (ICP) registration algorithm for surgical navigation applications. In this scenario, we have a rigid bone surface represented as a triangular mesh in CT coordinates, two optical tracking rigid bodies (body A serving as a pointer and body B rigidly attached to the bone), and an optical tracker that records LED marker positions. Our goal is to determine where the pointer tip contacts the bone surface in CT coordinates. We accomplish this through two primary methods. First, we use 3D point-to-point registration (Arun's method) to compute the pointer tip position d_k in the bone's local coordinate frame for each sample k. Second, we apply a closest-point-on-mesh algorithm to project each d_k onto the triangular surface to find the corresponding surface point c_k.

3 Program Structure



3.1 Triangles Module: `utils/triangles.py`

Function	Description	Input	Output
compute_normal	Computes unit vector of triangle.	N/A	normal: Unit vector, numpy array (3,)
build_bounds	Builds bounds for box bounding.	N/A	lb: Lower bounds, numpy array (3,) ub: Upper bounds, numpy array (3,)
barycentric_coords	Build barycentric coordinates.	p: 3D point.	u, v, w: Barycentric coordinates, floats
project_to_plane	Projects point onto base of triangle.	p: Point in space.	p_proj: Orthogonal projection, numpy array (3,)
in_box	Checks if a point is within bounding box.	p: Query point. margin: float.	True if p in [lb - margin, ub + margin], else False
closest_point_on_edge	If closest point is on an edge, finds that point.	p: Query point. a, b: Endpoints of segment (numpy arrays (3,))	c: Closest point on segment AB, numpy array (3,)
closest_point	Finds closest point on a triangle.	p: Query point.	closest: numpy array (3,) Closest point on the triangle.

3.2 Mesh Module: `utils/mesh.py`

Function	Description	Input	Output
__init__	Initializes and builds internal Triangle list.	vertices: array-like (N_vertices $\times$ 3) indices: array-like (N_triangles $\times$ 3)	Mesh object.
build_mesh	Builds a list of Triangle objects using vertices and index.	vertices: array (N $\times$ 3) indices: array (M $\times$ 3)	`self.triangles`.
find_closest_point_linear	Linear search for closest point on mesh surface.	p: Query point (3D)	closest_point: numpy array (3,) Closest point on any triangle.
find_closest_point	Optimized search using bounding boxes..	p: Query point (3D)	closest_point: numpy array (3,)

3.3 IO Module: `utils/IO.py`

Function	**Description**	**Input**	**Output**
read_body	Reads a body definition file.	filepath: str Path to the body file.	markers: (N_markers $\times$ 3) float array tip: (3,) float array N_markers: int name: str
read_mesh	Reads a surface mesh file.	filepath: str Path to mesh file.	vertices, triangle_indices, neighbors: (N_vertices $\times$ 3) float array N_vertices N_triangles: int
read_sample	Reads tracker sample data for bodies A and B over multiple frames.	filepath: str N_A: markers on body A N_B: markers on body B	A_samps,B_samps: (N_samps $\times$ N_A $\times$ 3) array N_s,N_samps: total markers and samples
write_output	Writes formatted PAHW3 output file.	filename: str D: (N_samps $\times$ 3) estimated tip positions C: (N_samps $\times$ 3) closest mesh points	Writes output file with: N_samples and filename Rows of dk, ck, and $\|\|d_k - c_k\|\|$

3.4 Rigid Transform and Registration Module: `utils/transform_register.py`

Function	**Description**	**Input**	**Output**
aruns_method	Computes the rigid transform using Arun's SVD method.	A: (N $\times$ 3) array, source points B: (N $\times$ 3) array, target points	R: (3 $\times$ 3) rotation matrix t: (3,) translation vector
compute_d	Computes pointer tip positions d_k in B-frame using $d = F_{Bk}^{-1} F_{Ak} A_{\text{tip}}$.	A_body_markers: (N_A $\times$ 3) B_body_markers: (N_B $\times$ 3) A_tip: (3,) A_samps: (N_samples $\times$ N_A $\times$ 3) B_samps: (N_samples $\times$ N_B $\times$ 3)	d: (N_samples $\times$ 3) array of pointer tip positions in B frame
apply_Freg	Applies rigid transform [R, t] to points. Identity transform used if R or t are None.	points: (N $\times$ 3) or (3,) array R: (3 $\times$ 3) or None t: (3,) or None	Transformed points (same shape as input)
compute_ck	Computes closest mesh points c_k for transformed series $F_{reg} d_k$. Uses linear or bounding-box search.	mesh: Mesh object d_series: (N_samples $\times$ 3) R, t: optional registration transform linear: bool (linear vs. optimized search)	C: (N_samples $\times$ 3) array of closest mesh points

3.5 Main Script: `src/main.py`

Function	**Description**	**Input**	**Output**
main	Executes the full PA3 workflow: reads data, constructs mesh, computes d_k, computes c_k, and writes results.	A_file: body A definition file B_file: body B definition file mesh_file: surface mesh (.sur) sample_file: tracker readings outfile: path for output file linear: bool (linear vs. optimized search)	Writes a formatted output file containing: d_k: transformed tip positions in B frame c_k: closest surface points to each d_k

3.6 Automation Script: `src/run_all.py`

Component	**Description**	**Input**	**Output / Behavior**
Debug Batch Runner	Automatically executes `main.py` on all PA3 debug and demo sample files. Intended for local testing.	None (file paths hard-coded).	Runs `main.py` for: Debug sets: A–F Demo sets: G, H, J Produces output files: `output/pa3-X-output.txt` for each dataset.

4 Mathematical Approach

4.1 Triangle Methods

We created a `Triangle` object to hold the base shape of our mesh. Each triangle consists of three points upon initialization (a, b, c).

For ease of calculation, when projecting a point $p = (x, y, z)$ onto our triangle, we first convert p to barycentric coordinates (u, v, w)[1]. These coordinates represent the relative areas of the subtriangles formed with the triangle's vertices.

Using the vector approach[2], we define:

$$u = \frac{\Delta BCP}{\Delta ABC} = \frac{||(B-P) \times (C-P)||}{||(B-A) \times (C-A)||}$$
$$v = \frac{\Delta ACP}{\Delta ABC} = \frac{||(C-P) \times (A-P)||}{||(B-A) \times (C-A)||}$$
$$w = \frac{\Delta ABP}{\Delta ABC} = \frac{||(A-P) \times (B-P)||}{||(B-A) \times (C-A)||}$$

We can derive this from area formula

$$\Delta ABC = \frac{||(B-A) \times (C-A)||}{2}$$

A projected point lies inside the triangle if and only if $u \geq 0$, $v \geq 0$, and $w \geq 0$.

When projecting a point onto the plane of a triangle, we use the dot product to find the distance to the plane d. The unit normal vector is computed as:

$$\mathbf{n} = (b-a) \times (c-a), \qquad \hat{\mathbf{n}} = \frac{\mathbf{n}}{\|\mathbf{n}\|}.$$

The signed distance[3] from p to the plane is:

$$d = (p-a) \cdot \hat{\mathbf{n}}$$

The projection is therefore:

$$p_{\text{proj}} = p - d\,\hat{\mathbf{n}}.$$

4.2 Point Cloud Registration

For 3D point set registration, we utilized Arun's Method for 3D registration[4]. Arun's method utilizes singular value decomposition instead of iteration, making it easy to implement.

[1]Russell Taylor, "Segmentation and Modeling Lecture".

[2]scratchapixel, "Ray-Tracing: Rendering a Triangle", *https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/barycentric-coordinates.html.*

[3]Russell Taylor, "Registration Lecture 2".

[4]K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-Squares Fitting of Two 3-D Point Sets", *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-9, no. 5 (1987): 698–700, `https://doi.org/10.1109/TPAMI.1987.4767965`.

We are trying to minimize the following equation:

$$\sum_{i=1}^{N} \|p_i' - (\mathbf{R}p_i + \mathbf{t})\|^2$$

Where p_i, p_i' represent corresponding points in each point set, and $\mathbf{R}$ and $\mathbf{t}$ are the rigid transformation needed to align the two point sets.

Following the proof from Jingnan Shi[5], we can see that this is equivalent to singular value decomposition of

$$H = \sum_{i=1}^{N} q_i\, q_i'^{\top} = U\Lambda V^{\top}.$$

From this we can calculate

$$X = VU^{\top}$$

which is our rotation matrix if $\det(X) = 1$.

If SVD returns $\det(X) < 0$, we flip the sign of the matrix using

$$X' = V'U^{\top}$$

where

$$V' = [\, v_1,\ v_2,\ -v_3\,].$$

4.3 Finding d_k

For each sample frame k, the optical tracker provides the positions of LED markers on bodies A and B in tracker coordinates. Applying Arun's method to the marker data yields the rigid transformations:

$$F_{A,k} = (R_A, t_A), \qquad F_{B,k} = (R_B, t_B). \tag{1}$$

The pointer tip in tracker coordinates is:

$$p_{\text{tracker}} = R_A A_{\text{tip}} + t_A. \tag{2}$$

To express this tip position in the coordinate frame of body B, we apply the inverse of $F_{B,k}$:

$$F_{B,k}^{-1} = (R^T,\, -R^T t).$$

Thus,

$$d_k = F_B k^{-1} * F_{A,k} * A_{tip} \tag{3}$$

[5]Jingnan Shi, "Arun's Method for 3D Registration", *https://jingnanshi.com/blog/arun*$_m$*ethod*$_f$*or*$_3$*d*$_r$*eg.html*, 2022,

4.4 Finding c_k

Given the mesh with triangles $\{T_i\}$ and the registration transform F_{reg}, we compute:

$$s_k = F_{reg} \cdot d_k \tag{4}$$

In PA3, $F_{\text{reg}} = I$, so $s_k = d_k$.
The find the closest point on the mesh, we implemented a bounded box speed up.
In this, each triangle stores an axis-aligned bounding box:

$$a = (x_1, y_1, z_1)$$

$$b = (x_2, y_2, z_2)$$
$$c = (x_3, y_3, z_3)$$
$$Lower = (\min(x1, x2, x_3), \min(y_1, y_2, y_3), \min(z_1, z_2, z_3))$$
$$Upper = (\max(x1, x2, x_3), \max(y_1, y_2, y_3), \max(z_1, z_2, z_3))$$

Given the current best distance, triangles for which s_k lies outside the bounding box are skipped, ensuring only potentially closer triangles are evaluated.

5 Algorithmic Approach

5.1 Tools Used

For this assignment, we worked in Python. Our main package used was NumPy[6]. This package was mainly used for linear algebra. We also utilized the typing[7] library to ensure clean type check.

5.2 Triangle Methods

Triangle-level operations form the core of our algorithm. These include computing triangle normals, barycentric coordinates, and the closest point on a single triangle.

Normal Vector Computation

The normal is used both for projection onto the triangle plane and for orientation consistency.

Algorithm 1 Compute Normal Vector

```
1: **Input:** Triangle vertices $a, b, c \in \mathbb{R}^3$
2: **Output:** Unit normal vector $\hat{n}$
3: $v_1 \leftarrow b - a$
4: $v_2 \leftarrow c - a$
5: $n \leftarrow v_1 \times v_2$
6: $\hat{n} \leftarrow n/\|n\|$
7: **return** $\hat{n}$
```

Barycentric Coordinates

Barycentric coordinates allow us to determine whether a point lies inside a triangle.

Closest Point on a Triangle

To determine the closest point to an arbitrary point p, we project p onto the triangle plane and use barycentric coordinates to test whether the projection lies within the triangle. If not, we compute the closest point on each of the three edges.

5.3 Mesh Methods

- **Linear Search:** Checks every triangle.
- **Bounded Box Search:** Uses bounding boxes to reject triangles that cannot be closest.

Algorithm 2 Barycentric Coordinates

```
1: Input: Triangle vertices $a, b, c$; query point $p \in \mathbb{R}^3$
2: Output: Coordinates $(u, v, w)$
3: $v_0 \leftarrow b - a,\ v_1 \leftarrow c - a,\ v_2 \leftarrow p - a$
4: $d_{00} \leftarrow v_0 \cdot v_0$
5: $d_{01} \leftarrow v_0 \cdot v_1$
6: $d_{11} \leftarrow v_1 \cdot v_1$
7: $d_{20} \leftarrow v_2 \cdot v_0$
8: $d_{21} \leftarrow v_2 \cdot v_1$
9: $\text{denom} \leftarrow d_{00}d_{11} - d_{01}^2$
10: $v \leftarrow (d_{11}d_{20} - d_{01}d_{21})/\text{denom}$
11: $w \leftarrow (d_{00}d_{21} - d_{01}d_{20})/\text{denom}$
12: $u \leftarrow 1 - v - w$
13: return $(u, v, w)$
```

Algorithm 3 Closest Point on Triangle

```
1: Input: Triangle $T = (a, b, c)$; query point $p \in \mathbb{R}^3$
2: Output: Closest point $c$ on $T$
3: $\text{dist} \leftarrow (p - a) \cdot \hat{n}$
4: $p_{\text{proj}} \leftarrow p - \text{dist}\,\hat{n}$
5: $(u, v, w) \leftarrow \text{BarycentricCoords}(T, p_{\text{proj}})$
6: if $u \geq 0$ and $v \geq 0$ and $w \geq 0$ then
7:     return $p_{\text{proj}}$
8: end if
9: $c_1 \leftarrow \text{ClosestPointOnEdge}(p, a, b)$
10: $c_2 \leftarrow \text{ClosestPointOnEdge}(p, b, c)$
11: $c_3 \leftarrow \text{ClosestPointOnEdge}(p, c, a)$
12: $c \leftarrow \operatorname{argmin}_{x \in \{c_1, c_2, c_3\}} \|p - x\|$
13: return $c$
```

Algorithm 4 Linear Search for Closest Point on Mesh

```
1: Input: Mesh $M$ with $n$ triangles; query point $s$
2: Output: Closest point $c$
3: $\text{minDist} \leftarrow \infty,\ c \leftarrow \text{null}$
4: for all triangles $T_i$ in $M$ do
5:     $x \leftarrow T_i.\text{ClosestPoint}(s)$
6:     $d \leftarrow \|x - s\|$
7:     if $d < \text{minDist}$ then
8:         $\text{minDist} \leftarrow d,\ c \leftarrow x$
9:     end if
10: end for
11: return $c$
```

Linear Search

Bounded Box Search

This method accelerates search by skipping triangles whose bounding boxes lie outside the current best distance.

Algorithm 5 Bounded Box Search for Closest Point

```
1: **Input:** Mesh $M$; query point $s$
2: **Output:** Closest point $c$
3: bound $\leftarrow \infty,\ c \leftarrow$ null
4: **for** $i = 0$ **to** $\min(5, n-1)$ **do**
5:     $x \leftarrow T_i$.ClosestPoint($s$)
6:     $d \leftarrow \|x - s\|$
7:     **if** $d <$ bound **then**
8:         bound $\leftarrow d,\ c \leftarrow x$
9:     **end if**
10: **end for**
11: **for all** triangles $T_i$ in $M$ **do**
12:     **if** not $T_i$.InBox($s$, bound) **then**
13:         **continue**
14:     **end if**
15:     $x \leftarrow T_i$.ClosestPoint($s$)
16:     $d \leftarrow \|x - s\|$
17:     **if** $d <$ bound **then**
18:         bound $\leftarrow d,\ c \leftarrow x$
19:     **end if**
20: **end for**
21: **return** $c$
```

[6] "NumPy", *https://numpy.org/.*
[7] "Typing", *https://docs.python.org/3/library/typing.html.*

5.4 Finding d_k

Before querying the mesh, we must determine the pointer tip position in body–B coordinates for every recorded sample.

Algorithm 6 Compute d_k for All Samples

```
1: Input: $A_{\text{body}}, B_{\text{body}}, A_{\text{tip}}, A_{\text{samps}}, B_{\text{samps}}$
2: Output: $d \in \mathbb{R}^{N_{\text{samps}} \times 3}$
3: for $k = 0$ to $N_{\text{samps}} - 1$ do
4:     $(R_A, t_A) \leftarrow \text{ArunsMethod}(A_{\text{body}}, A_{\text{samps}}[k])$
5:     $(R_B, t_B) \leftarrow \text{ArunsMethod}(B_{\text{body}}, B_{\text{samps}}[k])$
6:     $\text{tip}_{\text{tracker}} \leftarrow R_A A_{\text{tip}} + t_A$
7:     $d[k] \leftarrow R_B^T(\text{tip}_{\text{tracker}} - t_B)$
8: end for
9: return d
```

5.5 Finding c_k

Finally, we compute the closest point on the mesh surface to each pointer tip sample. Since PA3 assumes $F_{\text{reg}} = I$, we directly treat d_k as the search point unless a registration transform is supplied.

Algorithm 7 Compute c_k for All Samples

```
1: Input: Mesh mesh; points $d$; optional $F_{\text{reg}}$; flag linear
2: Output: $c \in \mathbb{R}^{N_{\text{samps}} \times 3}$
3: for $k = 0$ to $N_{\text{samps}} - 1$ do
4:     if $F_{\text{reg}}$ provided then
5:         $s_k \leftarrow R_{\text{reg}} d[k] + t_{\text{reg}}$
6:     else
7:         $s_k \leftarrow d[k]$
8:     end if
9:     if linear then
10:        $c[k] \leftarrow \text{mesh.FindClosestPointLinear}(s_k)$
11:    else
12:        $c[k] \leftarrow \text{mesh.FindClosestPoint}(s_k)$
13:    end if
14: end for
15: return c
```

6 Validation

6.1 Unit Tests for Triangle Class: Passed

1. `test_normal_unit_length()`, `test_normal_direction()` For these tests (and following ones) we use the triangle with vertices $A = (0, 0, 0), B = (1, 0, 0), C = (0, 1, 0)$. This triangle lies in the xy-plane. The normal vector computed from the cross product $(B - A) \times (C - A)$ is $(1, 0, 0) \times (0, 1, 0) = (0, 0, 1)$. The tests therefore verify that a) the normal vector is unit length 1 and b) the normal orientation agrees with the right-hand rule for the ordering $A \to B \to C$, which is in the positive z direction.

2. `test_bounds()` This test checks the correctness of the axis-aligned bounding box. Given vertices $A = (1, 2, 3), B = (4, -1, 10), C = (0, 5, 7)$, the lower bound is $\big(\min(x),\ \min(y),\ \min(z)\big) = (0, -1, 3)$, and the upper bound is $\big(\max(x),\ \max(y),\ \max(z)\big) = (4, 5, 10)$.

3. `test_barycentric_center()` $A = (0, 0, 0), B = (2, 0, 0), C = (0, 2, 0),\ P = \left(\frac{2}{3}, \frac{2}{3}, 0\right)$. The barycentric coordinates (u, v, w) should satisfy $u + v + w = 1$, and because P is inside the triangle, all coordinates must satisfy $u \geq 0, v \geq 0, w \geq 0$.

4. `test_project_to_plane()` We use the triangle $A = (0, 0, 0), B = (1, 0, 0), C = (0, 1, 0)$, whose supporting plane is the xy-plane. For the point $P = (0.3,\ 0.3,\ 5.0)$, the orthogonal projection onto this plane simply removes the z-component: $\Pi(P) = (0.3,\ 0.3,\ 0.0)$.

5. `test_in_box_true()` and `test_in_box_false()` These tests verify the bounding-box function with margin. For the triangle $A = (0, 0, 0), B = (1, 2, 3), C = (-1, -2, -3)$, the bounding box spans from $(-1, -2, -3)$ to $(1, 2, 3)$.

- In the "true" case, the point $P = (0.5,\ 0.5,\ 0.5)$ lies inside the bounding box, even with a small margin. The test checks that the function correctly returns `True`.
- In the "false" case, the point $P = (10,\ 10,\ 10)$ lies far outside the bounding region. The test ensures that the function correctly returns `False`.

6. `test_closest_point_inside_triangle()` For the triangle $A = (0, 0, 0),\quad B = (1, 0, 0),\quad C = (0, 1, 0)$, the point $P = (0.2,\ 0.2,\ 1.0)$ projects orthogonally onto $\Pi(P) = (0.2,\ 0.2,\ 0)$, which lies strictly inside the triangle. Thus, the closest point on the triangle is simply the projection itself.

7. `test_closest_point_near_edge()` For the triangle $A = (0, 0, 0),\quad B = (2, 0, 0),\quad C = (0, 2, 0)$, consider the point $P = (1,\ -1,\ 0)$. The projection of P onto the triangle's plane is $(1, -1, 0)$, but this projected point lies below the edge joining A and B. The closest point on the triangle is therefore the orthogonal projection onto that edge, which is $C_{\text{edge}} = (1,\ 0,\ 0)$.

8. `test_closest_point_near_vertex()` Using the same triangle, the point $P = (-1,\ -1,\ 0)$ projects to $(-1, -1, 0)$, which lies outside all edges and is closest to the vertex $A = (0, 0, 0)$. Thus the expected closest point is $C_{\text{vertex}} = (0, 0, 0)$.

6.2 Unit Tests for Mesh Class: Passed

1. `test_mesh_build()` This test verifies that the `Mesh` constructor correctly builds the internal list of triangles from a list of vertices and triangle index tuples. Given the index list $(0, 1, 2), (0, 1, 3)$, the mesh should contain exactly two triangles.

2. `test_getitem()` For a mesh containing only one triangle with vertices $A = (0, 0, 0), B = (1, 0, 0), C = (0, 1, 0)$, the expression `mesh[0]` should return a `Triangle` object whose vertex fields match the input vertices.

3. `test_find_closest_point_linear_inside_triangle()` A point with a projection that lies inside the triangle, following linear closest-point search should return the projected point itself.

4. `test_find_closest_point_linear_outside_edge()` Projecting onto the triangle's plane leaves the same coordinates, but the closest point on the triangle is the projection onto an edge.

5. `test_find_closest_point_linear_outside_vertex()` Using the same triangle, the point $P = (-1,\ -1,\ 0)$ is closest to the vertex $A = (0, 0, 0)$. Thus the closest point returned should be $(0, 0, 0)$.

6. `test_find_closest_point_multiple_triangles()` This test checks that the mesh chooses the closest triangle among multiple ones. The mesh contains two triangles:

$$(0, 0, 0),\ (1, 0, 0),\ (0, 1, 0),$$
$$(5, 5, 0),\ (6, 5, 0),\ (5, 6, 0)$$

The point $P = (0.2,\ 0.2,\ 1.0)$ is close to T_1 and far from T_2. Both the search methods should return (0.2, 0.2, 0).

7. `test_find_closest_point_matches_linear()` A mesh of three triangles arranged at different positions is used, and for each test point P, we check that `find_closest_point`(P) = `find_closest_point_linear`(P).

6.3 Speedup

To test that our speedup does in fact have significant results, we used the `time` command while running `run_all.py`.

Linear Search

	Time
real	0m13.432s
user	0m19.404s
sys	0m11.337s

Bounded Box Search

	Time
real	0m4.809s
user	0m15.453s
sys	0m6.378s

As you can see, there was a significant speedup using our bounded box method.

7 Results

7.1 Debug Dataset Validation

We compared our outputs with the provided reference files for debug datasets A through F. For each sample k, we compute the position error:

$$\varepsilon_k = \|c_k^{\text{ours}} - c_k^{\text{ref}}\|_2$$

We then report the mean and maximum errors across all samples in each dataset:

$$\bar{\varepsilon} = \frac{1}{N}\sum_{k=1}^{N} \varepsilon_k, \qquad \varepsilon_{\max} = \max_k \varepsilon_k$$

Validation Script. We used `tests/diff.py` to compute our d, c errors with respect to the debug files e.

Dataset	Mean Error	Max Error
A	0.0046	0.0141
B	0.0043	0.0141
C	0.0066	0.0141
D	0.0063	0.0200
E	0.0049	0.0141
F	0.0048	0.0173
Average	**0.0053**	**0.0156**

These small errors in results could be a noise from rounding, or from noise in implementation in regards to our method chosen for barycentric modeling. Either way, we believe these results to be minimal enough to indicate our code achieves high precision.

7.2 Unknown Dataset Results

Table 1: Unknown dataset G results (all values in mm).

d_k			c_k			$\|d_k - c_k\|$
-11.33	-27.57	-20.11	-10.66	-28.86	-19.76	1.495
32.48	12.85	3.57	35.25	15.64	3.89	3.944
-45.95	-12.07	-24.95	-44.08	-11.83	-25.16	1.899
0.01	-13.79	7.89	0.22	-11.62	7.59	2.203
25.91	-4.55	15.59	26.09	-5.03	15.72	0.520
11.28	-10.38	12.58	11.09	-9.68	12.33	0.765
-3.84	-14.30	0.87	-3.78	-13.24	0.42	1.157
18.47	0.39	46.12	20.16	-1.50	46.66	2.592
-14.09	-5.48	-49.79	-14.16	-5.99	-47.72	2.132
-24.49	-25.88	-12.61	-24.50	-26.84	-11.60	1.392
-44.76	-15.29	-22.15	-42.97	-14.97	-22.54	1.857
11.35	20.47	23.64	11.43	23.42	24.23	3.014
-32.10	-31.23	-23.23	-31.32	-30.00	-23.43	1.471
-2.27	-0.22	61.04	-2.28	-0.22	63.43	2.389
2.46	-14.73	-25.19	2.59	-14.78	-25.27	0.160
4.42	18.91	33.52	4.63	21.71	34.07	2.863
-1.53	-5.82	-29.22	-0.99	-5.67	-29.67	0.715
-33.92	-29.95	-22.13	-33.11	-28.67	-22.40	1.543
-37.66	3.29	-19.48	-37.51	3.13	-19.55	0.237
35.24	-3.29	-10.87	35.89	-3.89	-10.63	0.911

Table 2: Unknown dataset H results (all values in mm).

d_k			c_k			$\|d_k - c_k\|$
-30.72	-3.04	-43.00	-31.56	-2.35	-44.45	1.816
-4.01	10.24	58.27	-5.75	11.11	58.12	1.944
-36.50	-19.51	-10.24	-35.89	-18.79	-11.21	1.359
0.92	-6.40	44.26	0.78	-6.72	44.28	0.347
-17.43	-24.22	-45.49	-17.78	-23.37	-44.57	1.303
19.57	17.26	56.32	18.33	16.30	56.27	1.566
-9.69	-1.49	15.42	-10.22	-1.68	15.62	0.599
-13.06	-5.28	5.53	-13.28	-5.39	5.66	0.276
-31.05	-28.14	-13.20	-29.93	-26.94	-14.10	1.871
25.97	-14.65	-20.76	25.43	-12.59	-19.54	2.452
27.96	20.97	-9.66	28.63	22.01	-9.72	1.240
24.76	22.55	5.51	25.27	23.65	5.74	1.231
1.19	-4.62	48.53	0.33	-6.34	48.52	1.929
4.19	22.39	21.49	4.24	23.83	21.70	1.457
-21.44	-31.12	-11.71	-21.07	-28.65	-13.39	3.010
-2.46	5.61	-15.45	-2.32	8.41	-16.18	2.894
-35.92	-4.09	-13.66	-37.13	-3.28	-12.48	1.875
-32.31	-4.54	-42.92	-33.09	-3.90	-44.30	1.705
22.32	2.57	52.40	21.69	2.83	52.30	0.689
-5.96	22.83	31.83	-5.64	23.85	32.35	1.191

Table 3: Unknown dataset J results (all values in mm).

d_k			c_k			$\|d_k - c_k\|$
-22.89	-16.34	-7.61	-22.90	-16.39	-6.12	1.490
-32.48	3.32	-31.06	-34.82	6.14	-32.10	3.818
19.52	23.06	6.06	20.46	25.37	6.71	2.574
-36.41	-3.11	-23.13	-41.04	-0.59	-22.17	5.365
31.22	2.84	-29.92	30.16	2.73	-28.73	1.599
-4.19	13.82	8.25	-4.67	14.37	7.71	0.908
-38.70	-5.68	-30.93	-41.68	-6.84	-32.77	3.689
-30.10	-32.85	-21.93	-29.15	-30.88	-22.43	2.243
13.75	22.30	-7.39	11.65	24.33	-8.21	3.033
8.99	-8.71	52.88	8.84	-7.03	52.90	1.685
16.51	22.50	12.06	17.29	24.86	12.65	2.565
-21.80	6.90	-27.43	-22.29	12.21	-28.28	5.405
-31.48	-29.82	-18.98	-31.12	-29.18	-19.27	0.788
-4.05	-5.92	37.58	-3.49	-5.35	37.48	0.799
27.14	7.92	28.56	26.78	7.89	28.45	0.384
8.39	-8.60	62.53	8.28	-7.36	62.12	1.316
33.71	5.28	-27.38	31.88	5.42	-26.33	2.114
14.35	-8.67	34.43	13.77	-6.03	34.20	2.716
-6.84	8.47	9.81	-8.85	10.59	9.15	2.997
15.14	7.84	62.28	15.12	7.83	63.15	0.869