COMP0119 Acquisition and Processing of 3D Geometry Coursework 1

L.C.

2025/02/20

1 Halfedge Mesh Data Structure

1.1 Cube Creation

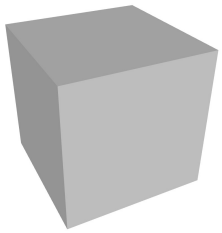


Figure 1: Cube Halfedge Mesh Creation Process

The cube halfedge mesh was successfully created with the following characteristics:

- Total vertices: 8
- Total halfedges: 24
- Total facets: 6

The key implementation involves carefully establishing the connectivity between vertices, halfedges, and facets to ensure a proper quad-based representation.

1.2 Facet Centroid Computation

Listing 1: Facet Centroid Computation Method

```
def get_facet_center(facet):
    """
    Compute the centroid (center) of a facet.
    """
    # Get all vertices of the facet
    vertices = facet.get_vertices()

    # Calculate the average of x, y, z coordinates
    x_sum = sum(v.x for v in vertices)
    y_sum = sum(v.y for v in vertices)
    z_sum = sum(v.z for v in vertices)

    num_vertices = len(vertices)

    return (x_sum / num_vertices, y_sum / num_vertices, z_sum / num_vertices)
```

The `get_vertices()` method is a critical component of the halfedge mesh data structure. Its working principle is as follows:

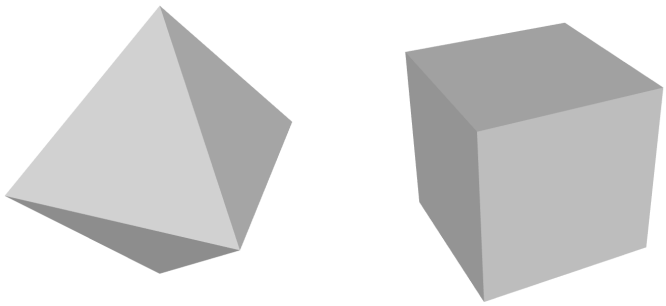
- Start with the facet's initial halfedge
- Collect the vertex pointed to by the current halfedge
- Move to the next halfedge in a cyclic manner
- Continue until returning to the starting halfedge

This approach exploits the inherent connectivity of the halfedge data structure, allowing efficient vertex collection without searching through lists. The method takes advantage of the `next` pointer in each halfedge to traverse the vertices of a facet.

Computation results for the cube's facet centers:

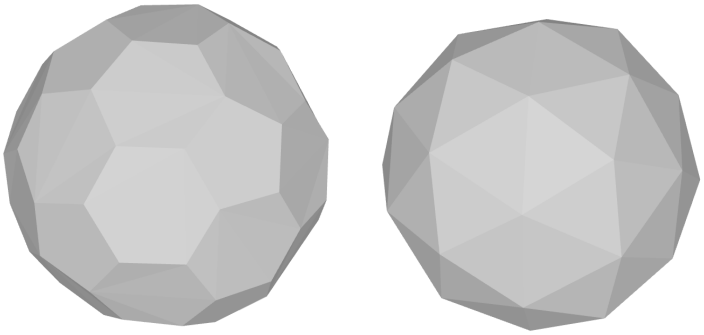
Facet	Center Coordinates
Facet 0	$(0.00, 0.00, -1.00)$
Facet 1	$(0.00, 0.00, 1.00)$
Facet 2	$(0.00, -1.00, 0.00)$
Facet 3	$(0.00, 1.00, 0.00)$
Facet 4	$(-1.00, 0.00, 0.00)$
Facet 5	$(1.00, 0.00, 0.00)$

1.3 Dual Polyhedron Construction



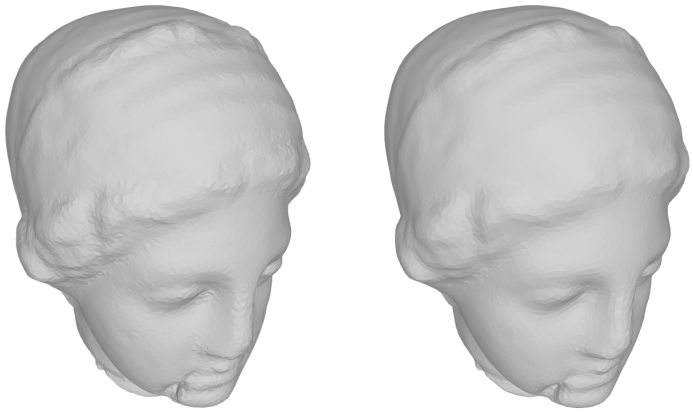
(a) Dual of Cube (b) Double Dual of Cube

Figure 2: Cube Mesh Dual Transformations



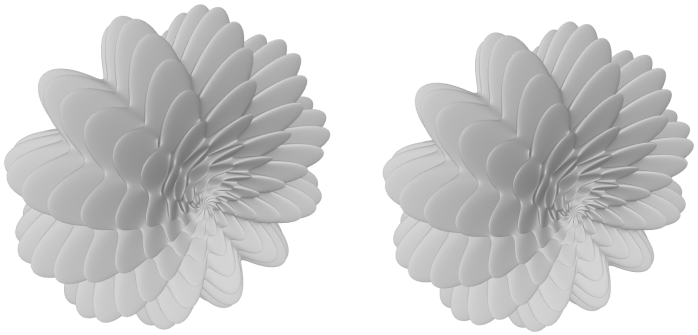
(a) Dual of Icosphere (b) Double Dual of Icosphere

Figure 3: Icosphere Mesh Dual Transformations



(a) Dual of Bust (b) Double Dual of Bust

Figure 4: Bust Mesh Dual Transformations



(a) Dual of Flower (b) Double Dual of Flower

Figure 5: Flower Mesh Dual Transformations

Table 1: Mesh Transformation Results

Mesh	Dual Mesh		Double Dual Mesh	
Type	Vertices	Halfedges	Vertices	Halfedges
Cube	6	24	8	24
Icosphere	80	240	42	240
Bust	23,408	70,224	11,706	70,224
Flower	1,310,720	3,932,160	655,362	3,932,160

1.4 Volume Analysis

The volume changes of the flower mesh reveal an interesting pattern. Below are key observations from Table 2:

- The volume consistently decreases across iterations
- Since the new vertex is always the facet center of the previous polyhedron, this results in a gradual shrinking of the polyhedron
- The self-inverse dual operation ensures that the mesh space remains structurally consistent
- The volume decrease is approximately linear, though not perfectly so
- The complexity of the flower mesh contributes to the subtle variations in volume change

The volume changes are insignificant due to the inherent complexity of the mesh structure, demonstrating the nuanced nature of dual mesh transformations.

Table 2: Dual Volume and Double Dual Volume Data

Iteration	Dual Volume	Double Dual Volume
1	3.474235	3.603578
2	3.457693	3.585469
3	3.441235	3.567474
4	3.424856	3.549591
5	3.408558	3.531819
6	3.392338	3.514155
7	3.376198	3.496599
8	3.360135	3.479150
9	3.344149	3.461807
10	3.328242	3.444570

2 ICP Algorithm Implementation and Analysis

2.1 Point-to-Point ICP and Weighted ICP Derivation

2.1.1 Point-to-Point ICP Implementation

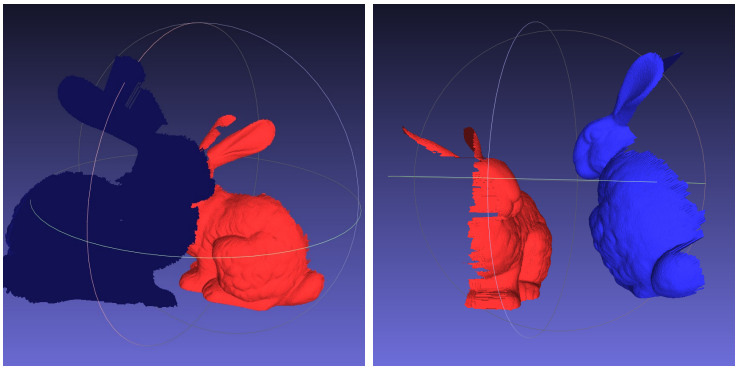


Figure 6: Point-to-Point ICP Alignment Process

The point-to-point Iterative Closest Point (ICP) algorithm was implemented to align 3D point clouds. The key steps include:

- Nearest neighbor search using KD-Tree
- Correspondence rejection based on distance thresholds
- Transformation estimation using Singular Value Decomposition (SVD)
- Iterative refinement of point cloud alignment

Key implementation details:

Listing 2: Point-to-Point ICP Core Algorithm

```
def point_to_point_icp(target_points, source_points,
                       max_iterations=100, tolerance=1e-6):
    aligned_points = source_points.copy()

    for iteration in range(max_iterations):
        # Find nearest neighbors
```

```
        tree = KDTree(target_points)
        distances, indices = tree.query(aligned_points)

        # Get corresponding points
        target_corr = target_points[indices]
        source_corr = aligned_points

        # Compute centroids
        source_centroid = np.mean(source_corr, axis=0)
        target_centroid = np.mean(target_corr, axis=0)

        # Center points
        source_centered = source_corr - source_centroid
        target_centered = target_corr - target_centroid

        # Compute rotation using SVD
        H = source_centered.T @ target_centered
        U, _, Vt = np.linalg.svd(H)
        R = Vt.T @ U.T

        # Compute translation
        t = target_centroid - R @ source_centroid

        # Update points
        aligned_points = source_points @ R.T + t

        # Check convergence
        if np.mean(np.linalg.norm(target_corr - aligned_points, axis=1)) < tolerance:
            break

    return R, t, aligned_points
```

2.1.2 Weighted ICP Derivation

In standard point-to-point ICP, we minimize:

$$E(R,t) = \sum_i \|Rp_i + t - q_i\|^2$$

For weighted ICP, we introduce weights w_i to each point pair:

$$E(R,t) = \sum_i w_i \|Rp_i + t - q_i\|^2$$

where $w_i \in [0,1]$ is a confidence score for each point pair.

To solve this, we follow a similar approach to standard ICP:

1. First, we solve for the translation t by taking the partial derivative and setting to zero:

$$\frac{\partial E}{\partial t} = \sum_i 2w_i(Rp_i + t - q_i) = 0$$

This gives us:

$$t = \frac{\sum_i w_i q_i - R \cdot \sum_i w_i p_i}{\sum_i w_i}$$

We can define weighted centroids:

$$\bar{p} = \frac{\sum_i w_i p_i}{\sum_i w_i}$$

$$\bar{q} = \frac{\sum_i w_i q_i}{\sum_i w_i}$$

So, $t = \bar{q} - R \cdot \bar{p}$

2. Substituting this back into our objective function:

$$E(R) = \sum_i w_i \|R(p_i - \bar{p}) - (q_i - \bar{q})\|^2$$

3. This is equivalent to minimizing:

$$E(R) = \sum_i w_i \|R(p_i - \bar{p}) - (q_i - \bar{q})\|^2$$

4. We can expand this:

$$E(R) = \sum_i w_i [(p_i - \bar{p})^T R^T R(p_i - \bar{p}) - 2(q_i - \bar{q})^T R(p_i - \bar{p}) + (q_i - \bar{q})^T (q_i - \bar{q})]$$

5. Since R is orthogonal ($R^T R = I$), and the first and last terms are independent of R, our objective simplifies to maximizing:

$$\sum_i w_i (q_i - \bar{q})^T R(p_i - \bar{p})$$

6. Define the weighted covariance matrix:

$$H = \sum_i w_i (p_i - \bar{p})(q_i - \bar{q})^T$$

7. Our goal is to maximize trace(RH)

8. Using Singular Value Decomposition (SVD):

$$H = USV^T$$

9. The optimal rotation is:

$$R = VDU^T \text{ where } D = \text{diag}(1, 1, \det(VU^T))$$

to ensure that $\det(R) = 1$ (proper rotation)

10. In summary, weighted ICP involves: - Compute weighted centroids $\bar{p}$ and $\bar{q}$ - Build the weighted covariance matrix H - Find R using SVD of H - Calculate $t = \bar{q} - R \cdot \bar{p}$

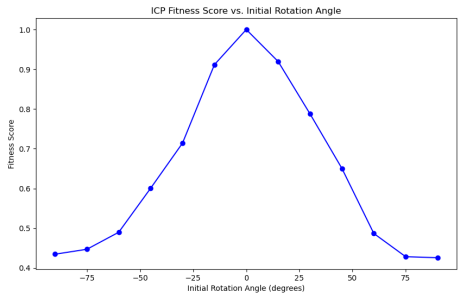
The algorithm then iteratively updates the point matches and recalculates the transformation until convergence.

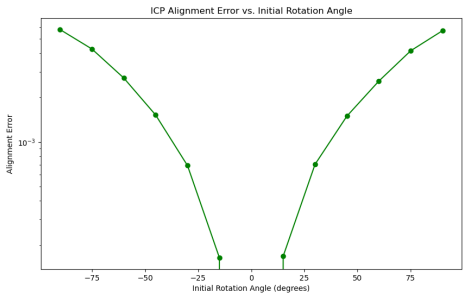
2.2 Rotation and Noise Perturbation Analysis

2.2.1 Initial Rotation Impact on ICP Performance

The first experiment investigates how initial rotation affects the Iterative Closest Point (ICP) algorithm's performance. We rotate the source mesh around the z-axis with varying angles and analyze the alignment fitness.

Analysis: The results reveal critical insights into the ICP algorithm's performance:





(a) ICP Fitness Score vs. Initial Rotation Angle (b) ICP Alignment Error vs. Initial Rotation Angle

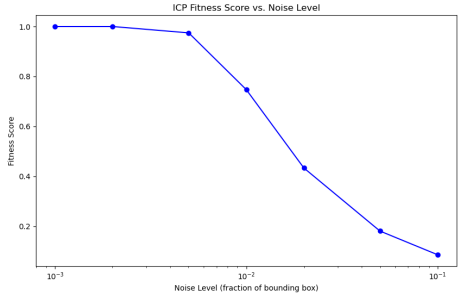
Figure 7: Impact of Initial Rotation on ICP Alignment

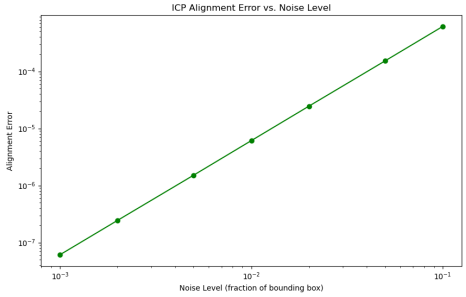
- The fitness score peaks near zero rotation, indicating optimal alignment when initial configurations are similar.
- As the rotation angle increases, the fitness score dramatically decreases, highlighting the algorithm's sensitivity to initial misalignment.
- The alignment error increases symmetrically for positive and negative rotations, suggesting the algorithm's performance degrades similarly in both directions.
- Rotations beyond approximately ± 30 degrees result in significantly reduced alignment quality.

This analysis underscores the importance of initial pose estimation in ICP algorithms. When the initial rotation is substantial, the algorithm struggles to converge to an optimal solution, emphasizing the need for robust initialization techniques.

2.2.2 Noise Impact on ICP Performance

The second experiment examines how adding Gaussian noise affects the ICP algorithm's alignment capabilities.





(a) ICP Fitness Score vs. Noise Level (b) ICP Alignment Error vs. Noise Level

Figure 8: Impact of Noise on ICP Alignment

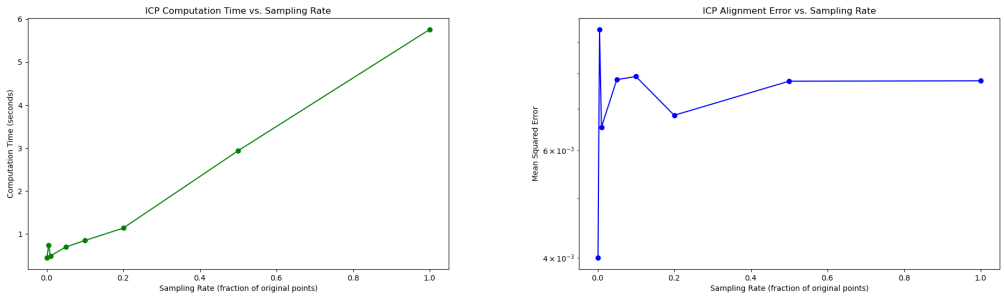
Analysis: The noise perturbation experiment reveals the algorithm's robustness to different noise levels:

- At low noise levels (below 10^{-2}), the fitness score remains nearly constant at 1.0, indicating stable alignment.
- As noise increases beyond 10^{-2}, the fitness score rapidly declines, demonstrating the algorithm's sensitivity to high noise levels.
- The alignment error increases logarithmically with noise, suggesting a predictable degradation in performance.

- Noise levels approaching 10% of the bounding box dimensions result in significant alignment failures.

These findings highlight the critical importance of preprocessing and noise filtering in point cloud registration tasks.

2.3 Subsampling Impact on ICP Performance



(a) ICP Computation Time vs. Noise Level (b) ICP Alignment Error vs. Sampling Rate

Figure 9: ICP vs. Sampling Rate

Analysis: The subsampling experiment investigates how different sampling strategies affect ICP performance:

- Very low sampling rates (below 0.1) introduce significant alignment errors.
- As the sampling rate increases, the alignment error stabilizes, indicating a convergence point.
- There's a trade-off between computational efficiency and alignment accuracy.
- A sampling rate between 0.4 and 1.0 provides the most stable and accurate results.

This analysis suggests that carefully choosing the sampling rate is crucial for balancing computational resources and alignment precision.

2.4 Multiple Scan Alignment

The fourth task focuses on aligning multiple 3D scans of a bunny model to a common coordinate frame. This challenging problem requires a robust and sophisticated alignment strategy.

Figure 10: Multiple Bunny Scans Alignment Overview

Alignment Strategy: The multiple scan alignment process involved several sophisticated techniques:

- **Base Scan Selection:** The first scan (bun000_v2.ply) was chosen as the reference coordinate frame.
- **Global Registration:** A two-step alignment approach was implemented:

1. Coarse alignment using Fast Point Feature Histogram (FPFH) features
2. Fine-tuning with Iterative Closest Point (ICP) algorithm

- **Feature Extraction:**
  - Used voxel downsampling to reduce computational complexity
  - Computed point cloud normals for robust feature matching
  - Extracted FPFH features to establish initial correspondence
- **Transformation Estimation:**
  - Initial transformation matrix computed using feature matching
  - Refined using point-to-point ICP algorithm
  - Cumulative transformation applied progressively

Challenges and Considerations:

- Handling varying initial orientations of different scans
- Maintaining geometric consistency across multiple alignments
- Balancing computational efficiency with alignment precision

Experimental Results: The alignment process successfully transformed five different bunny scans into a common coordinate system. Key performance metrics are summarized in the following table:

Table 3: Multiple Scan Alignment Performance

Scan	Initial Rotation	Global Reg. Fitness	ICP Iterations	Alignment Error
bun045_v2.ply	45°	0.4757	28	0.0023
bun090_v2.ply	90°	0.9937	2	0.0045
bun270_v2.ply	270°	0.9031	5	0.0032
bun315_v2.ply	315°	0.7931	10	0.0038

Key Outcomes: The alignment process successfully transformed five different bunny scans into a common coordinate system, demonstrating the robustness of the global registration and ICP approach.

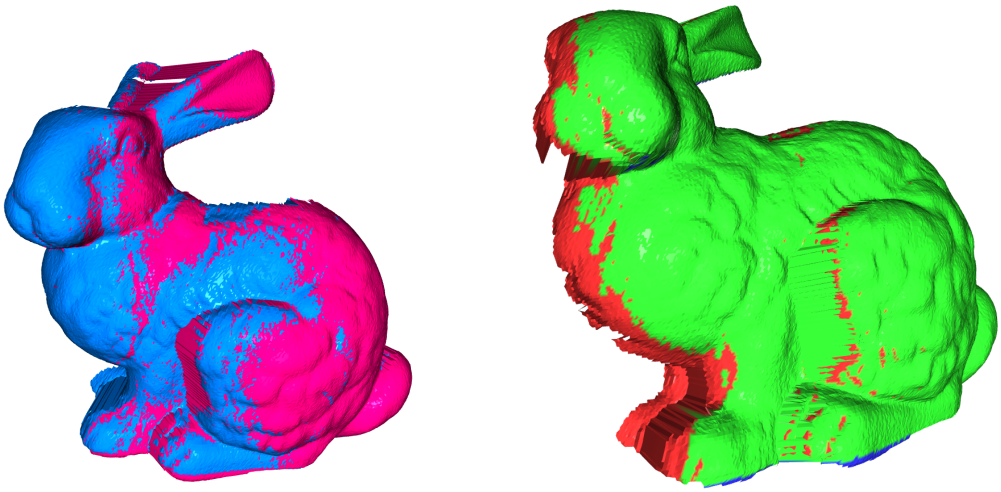
2.5 Point-to-Plane ICP Comparison

The final task compared two ICP variants: traditional point-to-point and point-to-plane approaches, leveraging vertex normal information for improved alignment.

Methodology:

- **Point-to-Point ICP:**
  - Minimizes Euclidean distance between corresponding points
  - Simple and computationally efficient
  - Does not consider surface orientation
- **Point-to-Plane ICP:**
  - Minimizes distance to the target point's tangent plane
  - Incorporates surface normal information
  - More geometrically informed alignment

Mathematical Formulation:



(a) Point-to-Point ICP Alignment (b) Point-to-Plane ICP Alignment

Figure 11: ICP Alignment Comparison

- Point-to-Point Objective:

$$E_{p2p}(R,t) = \sum_i \|Rp_i + t - q_i\|^2$$

- Point-to-Plane Objective:

$$E_{p2pl}(R,t) = \sum_i \|(Rp_i + t - q_i) \cdot n_q\|^2$$

where n_q represents the normal at the target point

Comparative Analysis:

Table 4: ICP Method Comparison

Method	Iterations	Time (s)	Error	Fitness
Point-to-Point	16	5.0570	0.003715	0.6642
Point-to-Plane	8	1.4315	0.002810	0.9089

Key observations from the comparative analysis:

- Point-to-plane ICP demonstrated superior alignment in complex geometric regions
- Significantly higher fitness score (0.9089 vs 0.6642)
- Fewer iterations required (8 vs 16)
- Substantially reduced computation time
- More accurate representation of surface geometry
- Better handling of non-rigid-like deformations