1 HALFEDGE MESH DATA STRUCTURE

1.1 Halfedge mesh for a cube

The halfedge_mesh Python package is used to construct a half-edge mesh for a cube. Initially, a HalfedgeMesh object was created, and 8 vertices of the cube were defined. After that the half-edge connectivity for each of the cube's six facets are structured. This process included assigning attributes such as vertex, next, facet, and opposite to each half-edge, ensuring proper connectivity among vertices, edges, and facets. The final step is focused on defining facet and vertex connectivity. Each facet was linked to one of its boundary half-edges, while each vertex was connected to one of its outgoing half-edges.

1.2 Centroid of facets

Centroid of facets can be simply calculated by averaging x, y and z coordinates of every vertices on same facet. With this function we can calculate vertex of dual dual polyhedron in task 1.3.

1.3 Halfedge mesh for the dual polyhedron

Constructing a dual polyhedron using a halfedge mesh can be acheived with three key steps. In the first step, the centroids of the facets of the original polyhedron are calculated using the `get_centroids` function. These centroids serve as the vertices of the dual polyhedron. Subsequently, the number of facets, vertices, and halfedges for the dual polyhedron are defined. Notably, the number of facets in the dual polyhedron equals the number of vertices in the original polyhedron, and vice versa. The number of halfedges remains the same.

In the second step, the `find_onering` function which is used in tutorial is employed to determine the one-ring neighbor vertices of each vertex in the original polyhedron. These neighbor vertices correspond to adjacent facets in the dual polyhedron, which share halfedges. By identifying these one-ring neighbors, the facets, halfedges, and vertices of the dual polyhedron can be effectively mapped and constructed.

Finally, the third step involves refining the dual polyhedron's structure. For each vertex in the original polyhedron, its one-ring neighbors are used to define the corresponding facets in the dual polyhedron. The halfedges are then updated to reflect their correct relationships in the dual structure, ensuring that each halfedge correctly points to its vertex, next halfedge, facet, and opposite halfedge.

2 CORE SECTION

2.1 ICP

To acheive ICP [1], corresponding points between the source and target point clouds are identified. This is achieved by utilizing a KDTree structure from `scipy.spatial` for efficiently finding the closest point in the target point cloud for each point in the source point cloud.

Crucial to the ICP process is the rejection of 'bad pairs', which are point correspondences that are likely to be incorrect. This is achieved by considering the normals at each point. The normals of the corresponding points are compared, and a pair is rejected if the angle between their normals exceeds a certain threshold. This step ensures that only points with similar surface orientations contribute to the transformation calculation, thereby improving the accuracy and robustness of the algorithm

$$P'_s = P_s - \mu_s$$
$$P'_t = P_t - \mu_t$$

Figure 1: Centering to 0 for all source and target points

$$H = P'^T_s \cdot P'_t$$
$$U\Sigma V^T = H$$
$$R = V \cdot U^T$$
$$t = P'^T_t - R \cdot P'^T_s$$

Figure 2: Calculation for transformation matrix R and t

To calculate transformation matrix, the filtered source and target points are centered at the origin, as shown in Figure 1. This step is important for aligning the points correctly. Then, these centered points to is used create a covariance matrix.

SVD is applied to this covariance matrix to find the rotation matrix **R** and the translation vector **t**, as illustrated in Figure 2. These two components - **R** and **t** - are then combined into a single transformation matrix. This combination is done in the function we implemented, using something called homogeneous coordinates. This approach makes the transformation process simpler and more efficient.

After applying this transformation, the error is calculated with formula below. This error measures how well the source points have been aligned with the target points. It is calculated as the sum of the squared differences between the transformed source points and the target points.

$$\sum_i \|\mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i\|^2$$

These steps are repeated until difference of current error and previous error is smaller than certain value. Following figure 3 is result of aligning two mesh which are bun000_v2.ply and bun045_v2 by using ICP algorithm.

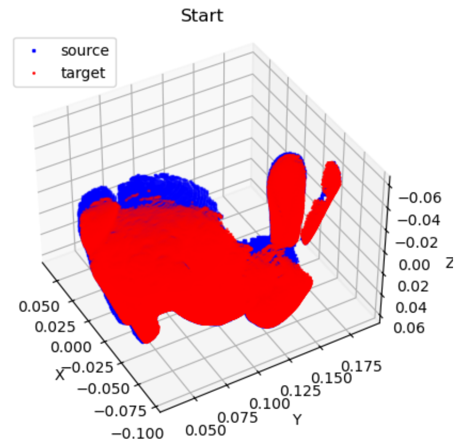


Figure 3: Result of ICP

2.1.1 Weighted ICP

In the context of the Iterative Closest Point (ICP) algorithm, the introduction of weights into the transformation matrix computation enhances the registration process by assigning varying levels of importance to each point pairing. This is expressed mathematically as:

$$\sum_i w_i \left\| \mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i \right\|^2$$

In this formula, w_i represents the weight for the i-th pair, which is often chosen to be inversely proportional to the distance between the source point $\mathbf{p}_i$ and its corresponding closest target point $\mathbf{q}_i$. This approach reduces the influence of point pairs that are spatially distant, thereby attenuating their effect on the transformation matrix that comprises the rotation $\mathbf{R}$ and the translation $\mathbf{t}$.

Through the implementation of Weighted ICP, where the weighting factor is the inverse of the point-to-point distances, reducing alignment error can be achieved. However, this accuracy improvement gives computational performance and longer computation time.

2.2 Experiment

2.2.1 ICP with Rotated Source points

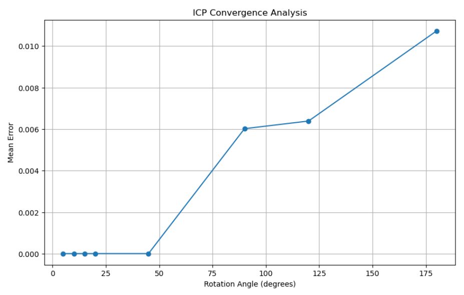


Figure 4: Mean error with different angles

This section examines the effect of rotation on the Iterative Closest Point (ICP) algorithm's accuracy. As figure 4 represents, when the rotation angle of the source points increases, there is a notable rise in the error. This trend is linked to the escalating number of iterations required for the algorithm to converge. A higher count of iterations introduces greater uncertainty, leading to less precise alignment results.

2.2.2 ICP with noisy model

In this part, the model is modified by adding normally distributed noise with a mean of zero and a standard deviation equal to the bounding box dimensions of the model in each axis (x, y, z). This addition of noise results in significantly worse outcomes compared to simple rotation (more than 100 times higher error). The rationale behind this degradation is the algorithm's reliance on rigid transformations, which maintain constant distances between points. The introduction of noise alters these distances, impeding the ICP algorithm's ability to accurately align the point sets.

2.3 ICP with Sub-sampling

As the number of iterations required for the Iterative Closest Point (ICP) algorithm increases, so does the computation time. To mitigate this, sub-sampling techniques are employed. This implementation adopts normal-space sampling [2], which contrasts with uniform sampling that distributes points without regard to surface curvature. Normal-space sampling is particularly advantageous for our purposes as it selects more points around areas of high curvature.

Normal-space sampling commences by segmenting the normal vector space into a predetermined number of bins, with each bin representing a specific range of angles. Normal vectors are categorized based on their orientation in the XY-plane, computing the 2D angle for each and then mapping these angles from $[-\pi, \pi]$ to the corresponding bin indices. This mapping process becomes robust with excluding any abnormal values by filtering out NaN or Inf.

After categorizing into bins, the point sampling process ensues. Unlike uniform sampling, this method is weighted; the number of points sampled from each bin is proportional to the number of points it contains. A predefined number of points are randomly chosen from each bin without replacement, which means bins with a higher density—often indicative of regions with more intricate geometry—contribute a greater number of points to the sample set. This selective emphasis on complex regions is optimal for ICP, as these areas typically require closer examination. The sampled points, along with their corresponding normals, are then aggregated into subsets. These subsets are used in the next computational steps, enhancing the efficiency of the overall process by significantly reducing the computational cost.

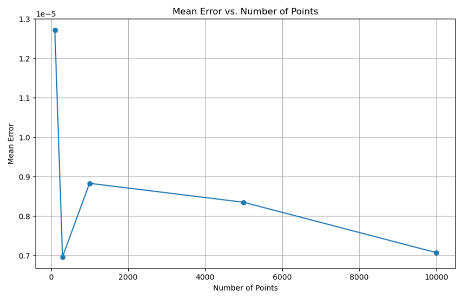


Figure 5: Mean error with different number of points

As Figure 5 illustrates, there is a clear correlation between the number of points sampled and the decrease in error. However sampling contains randomly choosing bin, graph can be changed for each computation, but given graph shape is most commonly shown. This trend can be attributed to the fact that normal-space sampling, by focusing more on areas with higher curvature, inherently samples more points from these complex regions. Consequently, as the sample size increases, the representation of these critical areas becomes more detailed and accurate, leading to a reduction in the overall alignment error.

The graph depicted in Figure 5 underscores the importance of sample size in achieving precision in the ICP algorithm. It reveals that while normal-space sampling efficiently targets areas of high curvature, the quantity of points sampled from these areas is still a crucial factor in determining the accuracy of the alignment. Therefore, an optimal balance must be struck between the number of points sampled and the computational resources available.

2.4 Align all together

To achieve precise alignment of all meshes, I selected the bun000_v2.ply mesh as the primary reference due to its comparatively higher vertex count and more distinctive features, such as ears and face, surpassing those of bun0180_v2.ply. Recognizing that the mere removal of poor pairings post-nearest neighbor computation does not assure perfect match alignment, certain meshes are applied manual rough preliminary adjustments to enhance the accuracy of the subsequent alignment process.

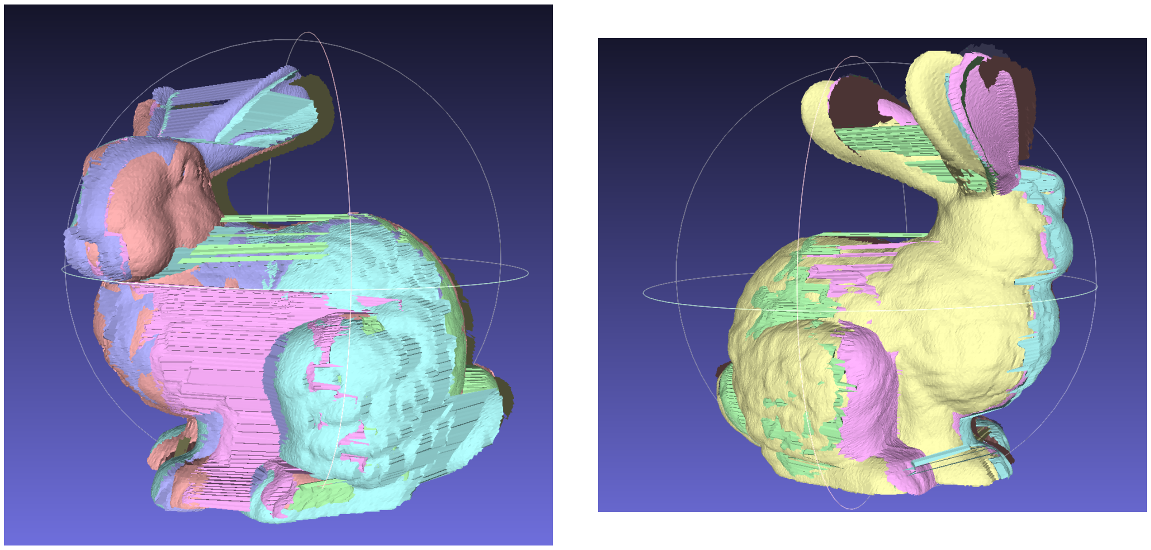


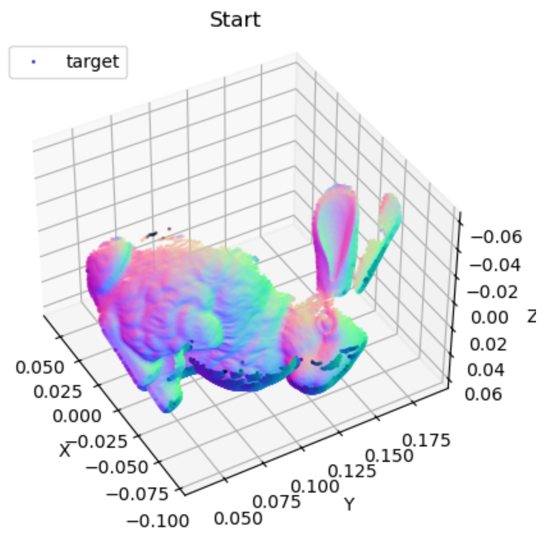
Figure 6: All aligned meshes

The alignment procedure commenced with the bun045_v2.ply mesh, which was aligned to the bun000_v2.ply using the ICP algorithm with sum-sampling. This aligned mesh was then merged with the bun000_v2.ply mesh. Following this, the bun090_v2.ply mesh was aligned to this newly combined mesh using the same ICP method and sum-sampling, and subsequently merged. This process was repeated for the remaining meshes.

For pre-processing, specific rotations were applied: bun090_v2.ply was rotated 90 degrees clockwise around the y-axis; bun180_v2.ply, 180 degrees clockwise; bun270_v2.ply, 90 degrees counterclockwise; and bun315_v2.ply, 70 degrees counterclockwise. This pre-processing significantly improved the results, as evidenced in Figure 6. However, due to the randomness in sub-sampling, certain inaccuracies may occur.

While this method generally gives more precise outcomes than the direct application of the ICP algorithm, it requires manual work for rough alignment. To overcome this limitation, future work could explore the development of a deep learning model can be trained of performing these preliminary mesh adjustments automatically to achieve end-to-end model.

2.5 Point to Plane ICP



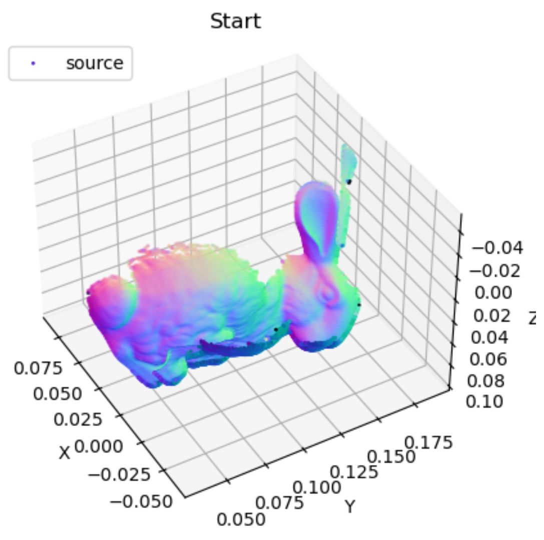


Figure 7: Normal based shading for M1 and M2

As figure 7, meshes with different angles get similar normal for same location such as eye or ear. Based on this we can idea to use point to plane ICP which minimizes the distance from the points of one surface to the plane of the corresponding points on the other surface by using normal.

$$\text{Point to Point:} \quad \sum_i \|\mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i\|^2$$

$$\text{Point to Plane:} \quad \sum_i \|(\mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i) \cdot \mathbf{n}_i^q\|^2$$

Figure 8: Comparison of error functions in ICP algorithms

As the point-to-point method minimizes the closest distance between two corresponding points, point to plane ICP [3] minimizes the distance from the points of one surface to the plane of the corresponding points on the other surface. This approach can lead to more accurate alignments in scenarios where the surfaces have more complex geometries like rabbit, as it takes into account the surface's orientation and not just the position of points.

Compared to point-to-point ICP, the key difference of point-to-plane ICP is the different error function it optimizes as figure 8. Therefore, until the step of computing the transformation matrix, the process is the same as the sub-sampling point-to-point ICP algorithm. In this case A and b vectors are calculated as figure 9 to solve Ax=b which gives x contains information of R and t.

$$v_i = \begin{bmatrix} \text{cross}(\text{P}_i^s, \text{n}_i^t) \\ \text{n}_i^t \end{bmatrix}$$

$$A = \sum_{i=1}^{N} \text{outer}(v_i, v_i)$$

$$b = \sum_{i=1}^{N} \left(\text{dot}(\text{P}_i^t - \text{P}_i^s, \text{n}_i^t) \times v_i \right)$$

Figure 9: Calculation of A and b for Ax=b

After getting A and b, we can solve Ax=b. Solution vector x can be represented as $x = \begin{bmatrix} \omega \\ t \end{bmatrix}$, where ω is the axis-angle representation and t is the translation vector. Also the rotation matrix R can be obtained with ω in implementation expm from `scipy.linalg` is used:

$$R = \exp\left(\begin{bmatrix} 0 & \omega_3 & -\omega_2 \\ -\omega_3 & 0 & \omega_1 \\ \omega_2 & -\omega_1 & 0 \end{bmatrix} \right)$$

After obtaining **R** and **t**, we can have same process as point to point method which are move **R** and **t** into a single matrix to facilitate the calculation of the transformation in one step using homogeneous coordinates. However, the key difference is as this method optimize different error function as point to point method mean error should be calculated as figure 8.

This method benefits compared to the point-to-point approach because it utilizes normal vectors to align the surfaces more accurately. By considering the orientations of the surfaces through normal vectors, the point-to-plane ICP algorithm effectively aligns surfaces with complex geometries. This is particularly advantageous in cases where surface features are not just clear in position but also in their orientation relative to the surrounding geometry, as seen in complicate shapes like a rabbit.

References

[1] Besl, P. J., & McKay, N. D. (1992). A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2), 239–256.

[2] Rusinkiewicz, S., & Levoy, M. (2001). Efficient Variants of the ICP Algorithm. In *Proceedings of the International Conference on Recent Advances in 3-D Digital Imaging and Modeling* (pp. 145-152). IEEE. doi:10.1109/IM.2001.924423

[3] Low, K.-L. (2004). Linear Least-Squares Optimization for Point-to-Plane ICP Surface Registration. Technical Report TR04-004, Department of Computer Science, University of North Carolina at Chapel Hill, February 2004.