Assignment 1 Halfedge and ICP

February 20, 2024

1 Abstract

- 1.1 Generated a halfedge mesh for a cube as well as drew its correspounding diagram.
- 1.2 Wrote a function which can compute the x, y, z co-ordinates of the face‘s centroid
- 1.3 Wrote a function which can return the half-edge mesh for the dual polyhedron given a half edge mesh and tested the function by inputing cube, the dual graph of cube and the icosphere.
- 2.1 Set Meshlab as my mesh viewer, implemented point2point ICP and refined the derivation done to solve weighted optimization problem..
- 2.2 Progressively perturbed the initial rotation of R and evaluate the convergence behavior of ICP, progressively added noise and evaluate the convergence behavior of ICP
- 2.3 Speed up my computation by estimating the aligning transform using subsampled versions of models
- 2.4 Attempted to come up with the best strategy to align multi scans
- 2.5 Used the normal information to improve the ICP performance(Implemented point2plane ICP)

2 Halfedge Mesh Data Structure

2.1 Generate a Halfedge Mesh for a Cube

The diagram of the halfedge mesh for my cube is as follow:

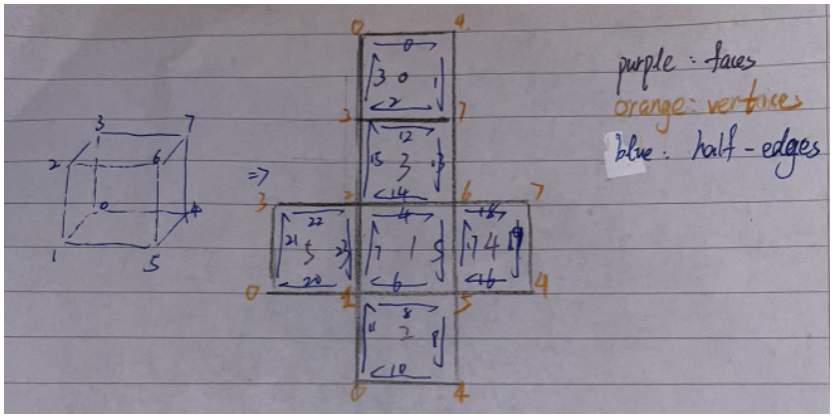


Figure 1: Diagram of my Cube

The cube I generated looked like this in MeshLab.

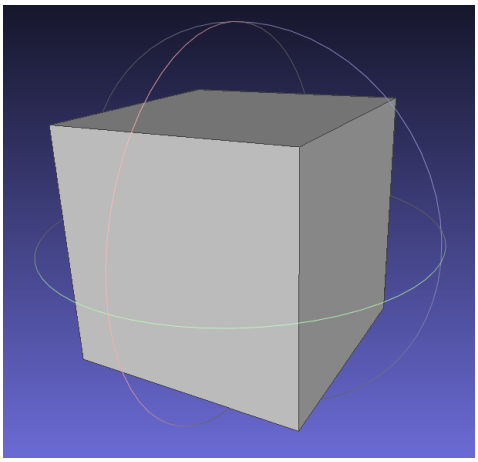


Figure 2: My halfedge data structured cube

2.2 Compute the Centroid of Facet

I computed the centroid of the input facet by using the method `'Facet.get_vertices()'`. So how the method `'Facet.get_vertices()'` works? It first finds the halfedge of the current facet, and then finds the starting points of all halfedges of the face by continuously calling next halfedge, and thus finds all the vertices of the face. After finding all the points, I computed the average of their coordinates on the xzy axis seperately to get the coordinates of the facet‘s centroid.

2.3 Compute our halfedge mesh for the Dual Polyhedron

To get a halfedge mesh for the dual polyhedron (M‘) from our original half-edge mesh(M), we need to follow the following principles:

- The facet of M will be transformed into a vertex of M‘.
- The vertex of M will be transformed into a facet of M‘

- For a facet f‘ of M‘, assuming it is transfored from the vertex v of M, then the halfedge he‘ of f‘ is transformed by the halfedge he whose start vertex is v Among them, the starting point and end point corresponding to he‘ are obtained from the facet attribute of he.

The more detailed description of the algorithm I used here is wrote in the comments of my code. To test my algorithm, I computed the dual of the cube defined previously and get the mesh with the shape of diamond which is correct:

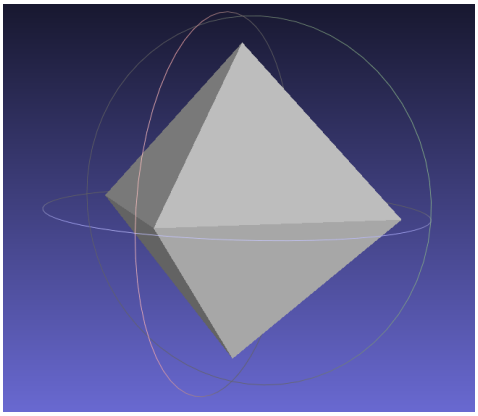


Figure 3: Dual Graph of the cube

I also computed the dual of the dual cube and get the mesh with the shape of cube which is correct:

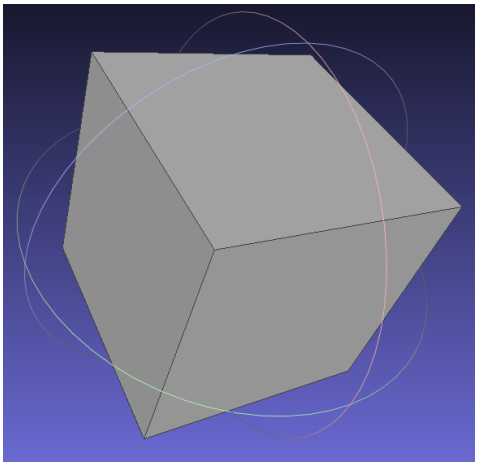


Figure 4: Dual Graph of the dual cube

I also computed the dual of the sphere1.off and get the mesh as following which is also correct:

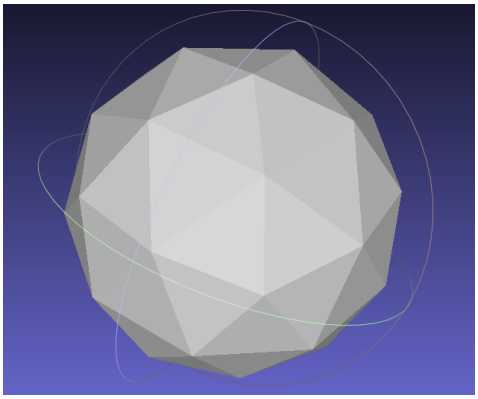


Figure 5: Dual Graph of sphere1.off

3 ICP

3.1 point to point ICP

I uesd MeshLab as my mesh viewer.

3.1.1 Implementation of Point2Point ICP

In this section, I implemented the traditional point2point ICP, used KDTree to speed up and visualized my results. I also refined the derivation done in class to solve the weighted ICP problem. The result of the alignment between "bunny000.ply" and "bunny045.ply" are as follows":

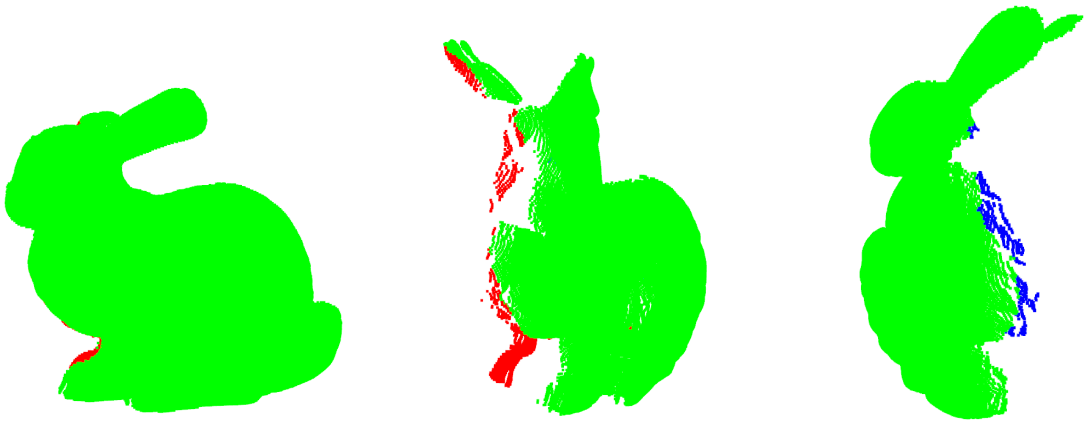


Figure 6: Visualization of Aligned Cloud Points from Different Angle(The green points represent aligned points, the red points represent unaligned points in mesh A and the red points represent unaligned points in mesh B.)

The output aligned mesh observed is as shown below

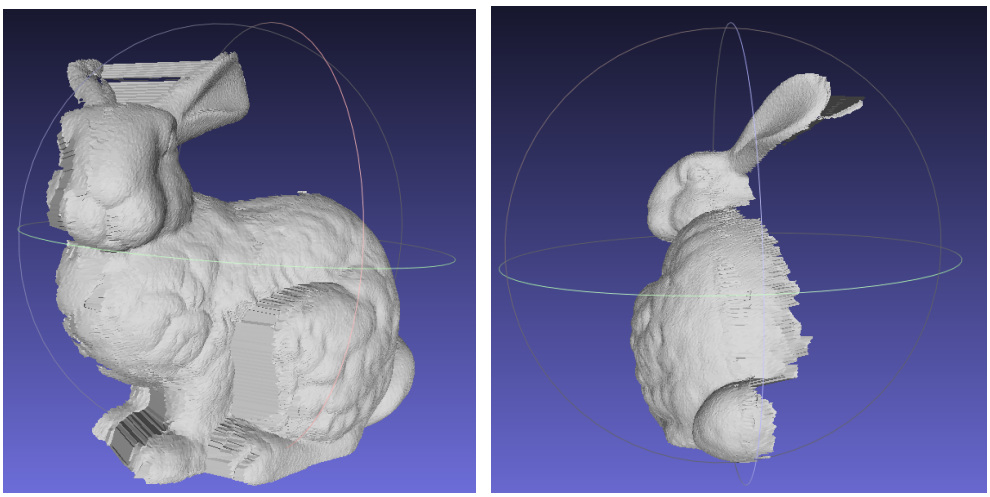


Figure 7: Visualization of aligned mesh in MeshLab from different angle

Judging from the two sets of pictures shown above, the point to point ICP alignment results here are good and the convergence is fast as well.

3.1.2 Derivation of Weighted ICP

Next, let's finish the second part of this section – refine the derivation done in class to solve the optimization $E(\mathbf{R}, \mathbf{t}) = \sum_i w_i \left\| (\mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i) \right\|^2$ over unknown rotation R and translation t.

First, we regularize points's weights w_i and compute the partial derivative of this expression with respect to t:

$$\begin{aligned} \frac{\partial F}{\partial t} &= \sum_{i=1}^{N} 2w_i \left(R \cdot p_i + t - q_i \right) \\ &= 2nt + 2R \sum_{i=1}^{N} w_i p_i - 2 \sum_{i=1}^{N} w_i q_i \end{aligned} \tag{1}$$

let equation(1) equals to 0 and we get:

$$t = \frac{1}{N} \sum_{i=1}^{N} w_i p_i - R \frac{1}{N} \sum_{i=1}^{N} w_i q_i \tag{2}$$

After the derivation of the optimal translation, we learnt that no matter how the rotation is valued, the optimal translation can be obtained by calculating the center of mass of the point cloud. So we no longer consider translation by define $\hat{p}_i = p_i - \bar{p}, \quad \hat{q}_i = q_i - \bar{q}$, then loss can be written as:

$$E(\mathbf{R}, \mathbf{t}) = \sum_{i=1}^{N} w_i \left\| R \cdot \hat{p}_i - \hat{q}_i \right\|^2 \tag{3}$$

Based on the derivation done in class(non-weighted ICP) we know that

$$w_i \|R\hat{p}_i - \hat{q}_i\|^2 = w_i \|\hat{p}_i\|^2 + w_i \|\hat{q}_i\|^2 - 2w_i \hat{p}_i^T R^T \hat{q}_i \tag{4}$$

$$R^* = \arg\max_R \left(\sum_{i=1}^{N} w_i \hat{p}_i R \hat{q}_i \right) = \arg\max_R \operatorname{trace}\left(W P^T R Q\right) \tag{5}$$

Let $H = WP^TQ$, similar to the derivation of non weighted ICP, we use svd decomposition of H, $H = U\Sigma V^T$, and we can get $R^* = VU^T$, by putting this result into equation (2), we can obtain the translation t.

3.2 Progressively perturb the model and Evaluate the ICP Algorithm Performance

3.2.1 Add Rotation

In this section, I simulated the effect of increasing misalignment (i.e., initial alignment) by rotating the object about z-axis over increasing rotation angles.

I printed mean error, times of iteration and time consumed to evaluate the result:

```
Angle: 0 degrees, Mean error: 2.0506390421834487e-17, Iterations: 2, time:0.18566608428955078
Angle: 5 degrees, Mean error: 0.0005169232970595079, Iterations: 9, time:0.8199942111968994
Angle: 10 degrees, Mean error: 0.0005283601477261675, Iterations: 12, time:1.4718983173370361
Angle: 15 degrees, Mean error: 0.0005476774148761764, Iterations: 13, time:1.9773929119110107
Angle: 20 degrees, Mean error: 0.0005504324757959595, Iterations: 14, time:2.0271763801574707
Angle: 25 degrees, Mean error: 0.0005548950706498226, Iterations: 15, time:2.8771872520446777
Angle: 30 degrees, Mean error: 0.0005842808150952362, Iterations: 16, time:4.0111987590789795
```

Figure 8: Printed Results of Adding Rotation

I also plotted figures to better visualize the effect of adding rotation angles:

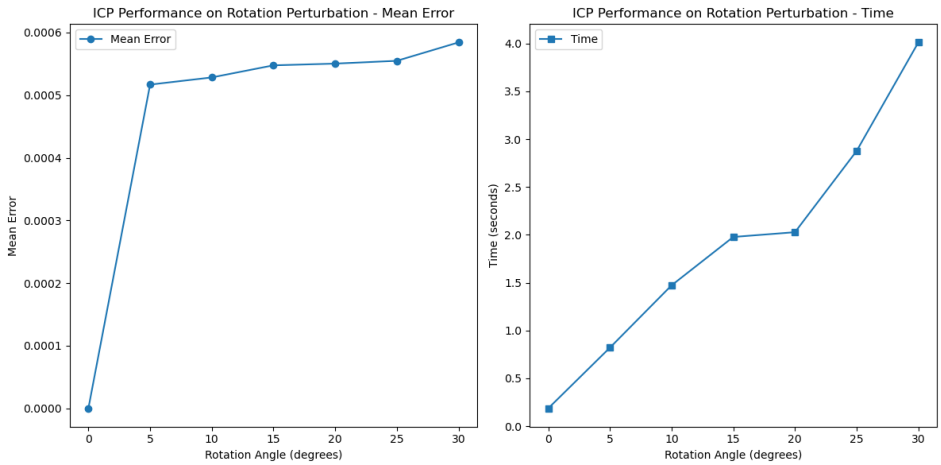


Figure 9: Plotted Results of Adding Rotation

Based on the visualization of the 2 sets of cloud points after alignment, I observed that my ICP can align the mesh well after perturbing different rotation angles.

Figure 10: Visualization of Aligned Cloud Points after Each perturbing

3.2.2 Add Gaussian Noise

In this section, I simulated the effect of increasing misalignment (i.e., initial alignment) by adding different amount of zero-mean Gaussian noises to the origin mesh.

I printed mean error, times of iteration and time consumed to evaluate the result:

```
Noise level: 0.0, Mean error: 2.0506390421834487e-17, Iterations: 2, time:0.17598485946655273
Noise level: 0.01, Mean error: 0.0013055074068654728, Iterations: 2, time:0.22258925437927246
Noise level: 0.02, Mean error: 0.0024956553960670666, Iterations: 2, time:0.3408243656158447
Noise level: 0.03, Mean error: 0.003686267928382393, Iterations: 2, time:0.47473645210266113
Noise level: 0.04, Mean error: 0.004883386164441198, Iterations: 2, time:0.6435079574584961
Noise level: 0.05, Mean error: 0.006062646153391346, Iterations: 2, time:1.0077323913574219
```

Figure 11: Printed Results of Adding Noise

I also plotted figures to better visualize the effect of adding different amount of zero-mean Gaussian noises:

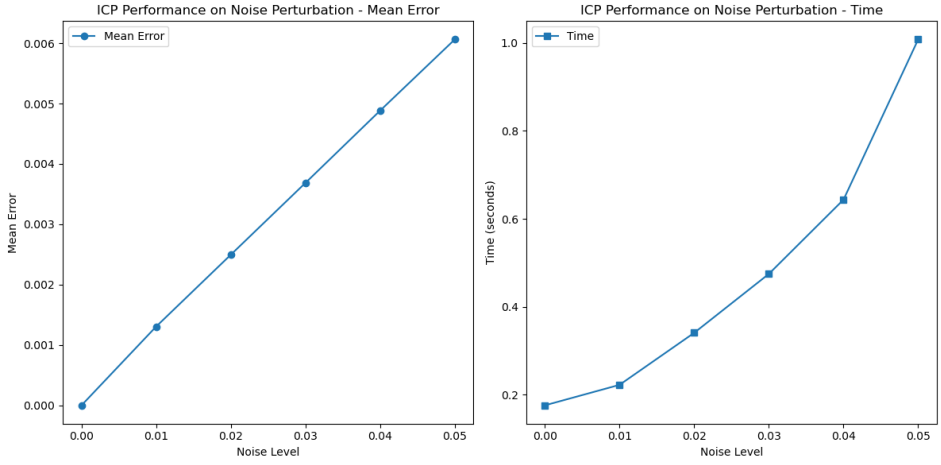


Figure 12: Plotted Results of Adding Noise

Based on the printed results and the plotted figures of this section, we can see that our ICP always converge to `INF_MIN` after adding different amount of zero-mean Gaussian noises, which indicate our ICP algorithm is robust enough to ignore different levels of noises.

3.3 Speed Up ICP by sub-sampling

In this section, instead of directly aligning M2 to M1, speed up your computation by estimating the aligning transform using subsampled versions of M1 and M2 as appropriate.

I didn't implement any fancy strategy for down-sampling, but directly use the api — np.random.choice.

I printed mean error, times of iteration and time consumed to evaluate the result:

```
Fraction: 1 degrees, Mean error: 0.001859828501122997, Iterations: 8, time:3.1211507320404053
Fraction: 0.5 degrees, Mean error: 0.0020056606419136857, Iterations: 8, time:0.968944787979126
Fraction: 0.25 degrees, Mean error: 0.002267793263826859, Iterations: 7, time:0.34894895553588867
Fraction: 0.1 degrees, Mean error: 0.002470187038845009, Iterations: 7, time:0.1726522445678711
Fraction: 0.05 degrees, Mean error: 0.003250332147782323, Iterations: 6, time:0.0581770957946777
Fraction: 0.01 degrees, Mean error: 0.004896361640893677, Iterations: 6, time:0.016521930694580078
```

Figure 13: Printed Results of Adding Levels of Sub-sampling

I also plotted figures to better visualize the effect of adding different amount of zero-mean Gaussian noises:

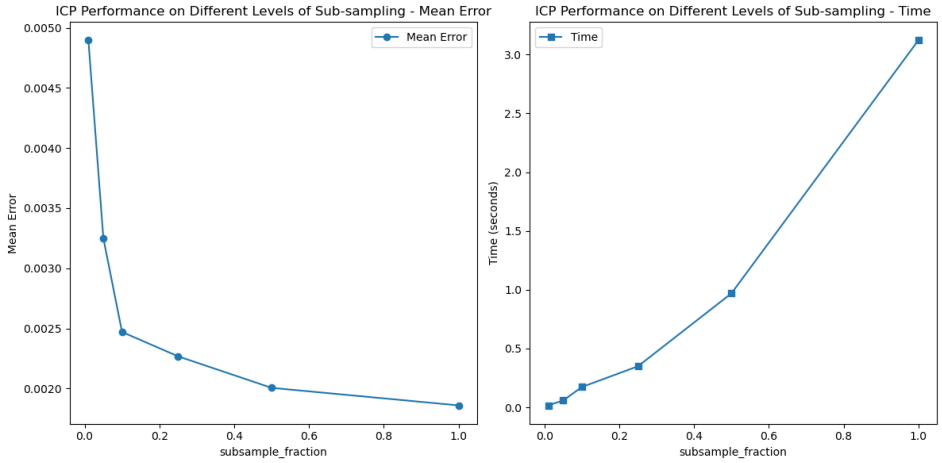


Figure 14: Plotted Results of Adding Levels of Sub-sampling

Visualization of aligned point clouds under different levels of sub-sampling are as follows:

(a) 0.5 Sub-sampling Level (b) 0.1 Sub-sampling Level (c) 0.01 Sub-sampling Level

Figure 15: Aligned Results under Different Levels of Sub-sampling Rate

Based on the visualization of the 2 sets of cloud points after alignment showed below as well as the printed results, We can observe that after gradually reducing the sub-sampling rate, the mean error of the ICP algorithm is still very small, the alignment effect is still very good, and at the same time, the convergence becomes significantly faster.

3.4 Align multiple scans to a common coordinate frame

Here are some strategies I implemented to align multiple scans.

3.4.1 Single Base Strategy

In this strategy, I use the mesh "bun000.ply" as the only base mesh, and then I align the rest of meshes to this base mesh one by one. This is the simplest strategy but the result it gets is far from satisfactory:

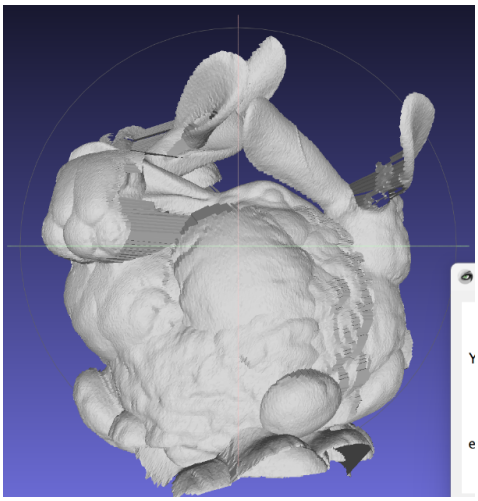


Figure 16: Result Generated by my First Strategy

3.4.2 Weighted ICP Strategy

In this strategy, I tried to use the weighted-ICP to optimize Single Base Strategy, using the distance and 1/distance weights of p_i corresponding to q_i queried by KdTree.

However the result is as disappointing as the one we get by 3.4.1.

3.4.3 Mean Error Priority Queue Strategy

We can observe that the mean error between some meshes is smaller and therefore easier to align correctly. So if we first align the meshes with smaller mean error to synthesize the merged mesh, then the merged mesh will have more information, and it is logically easier for the remaining meshes to be correctly aligned to this merged mesh.

In this strategy, before each merge of a new mesh, we will calculate the mean error of the merged mesh and all meshes that have not yet been merged, and put them into a priority queue.

After the calculation is completed, we select the smallest one in the priority queue. The mesh corresponding to the mean error is merged into the merged mesh.

Obviously, the time complexity of this strategy has increased significantly, from O(n) to O(n^2). Therefore, in order to make the algorithm converge faster, I also added the sub sampling idea of 3.3 to this strategy. After each merge, sample merged mesh is down sampled at a level of 0.5.

In the first three aligns and merges, this strategy achieved relatively good results:

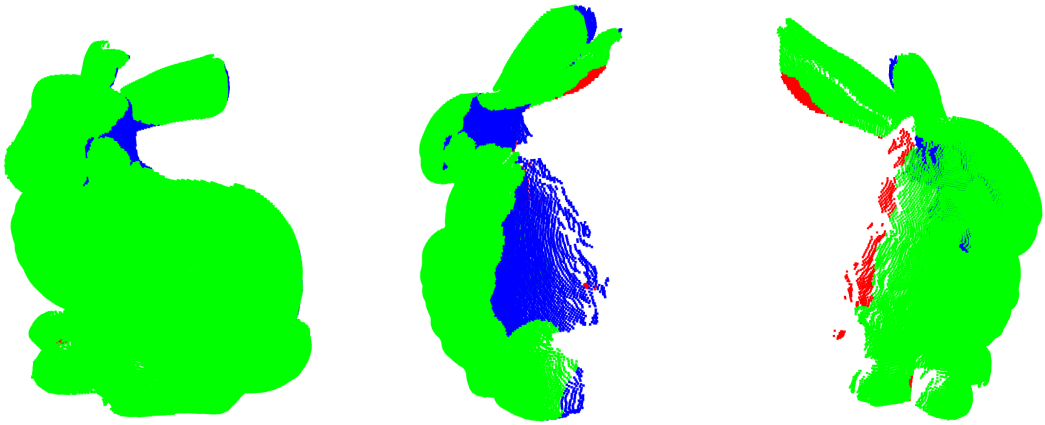


Figure 17: Aligned and Merged Result After First Three Iterations

But starting from the fourth merge, we started to get wrong align results again:

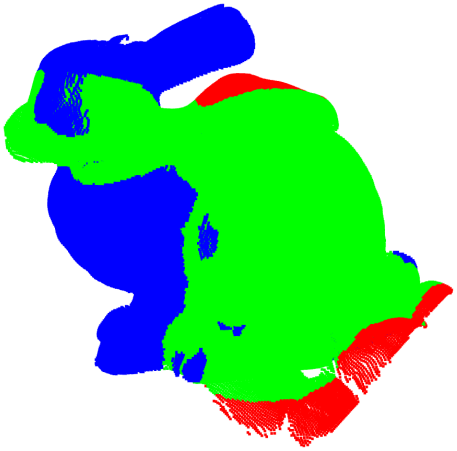


Figure 18: Aligned and Merged Result after the 4th iterations

3.4.4 Pre-alignment Strategy

In order to get absolutely correct align and merge result, the last strategy I used was to manually align the models that varies much using MeshLab — "bun000.ply", "bun090.ply", "bun180.ply", "bun270". ply". The pre-alignment result in MeshLab is as follow:

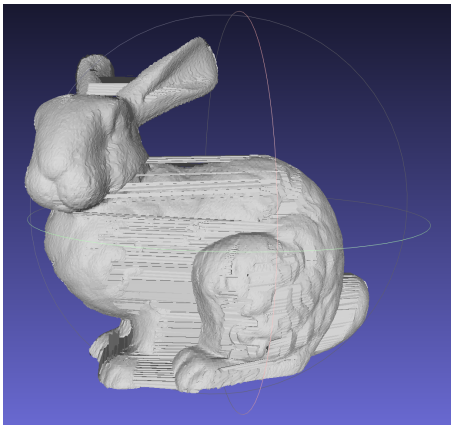


Figure 19: The Result of Pre-alignment

After pre-alignment, I use the strategy proposed in 3.4.4 to align and merge the remaining meshes into the base merged mesh.

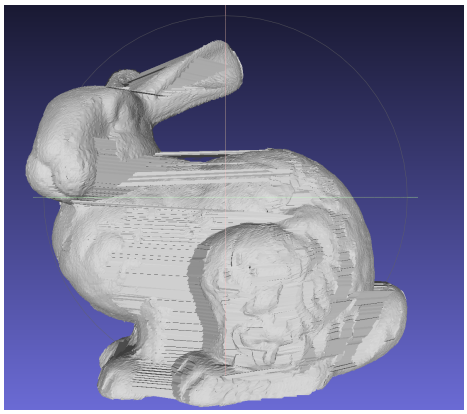


Figure 20: The Final Result of Multi Meshes Alignment

3.5 Point2Plane ICP

In this section, I computed normals at each vertex of the meshes and used the normal information to improve the ICP performance. Like discussed in class, I attempted to solve minimize the following objective function $E(\mathbf{R}, \mathbf{t}) = \sum_i \|(\mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i) \cdot \mathbf{n}_i^q\|^2$ instead of $E(\mathbf{R}, \mathbf{t}) = \sum_i \|(\mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i)\|^2$.

The formula derivation used in this section refers to this handout[PU].

The following figure shows the result obtained by using point2plane ICP align two meshes.

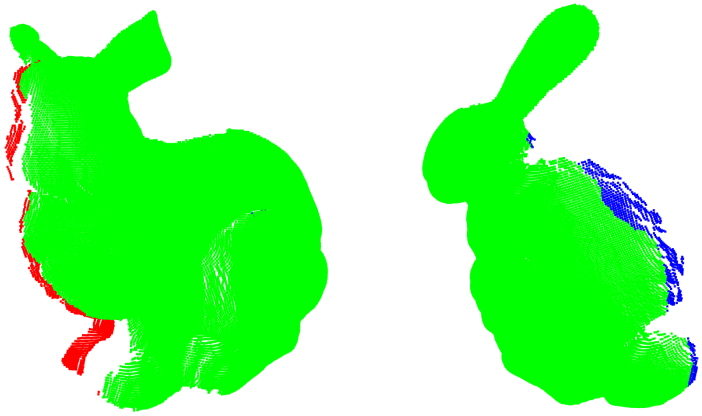


Figure 21: Visualization of Point2plane Aligned Cloud Points

We can see that it is basically correct.

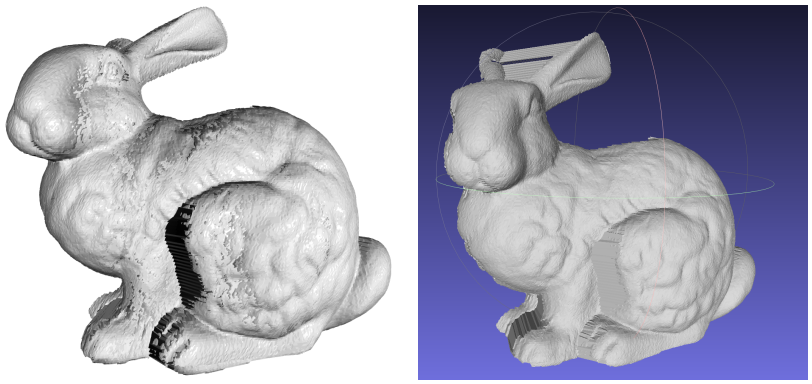
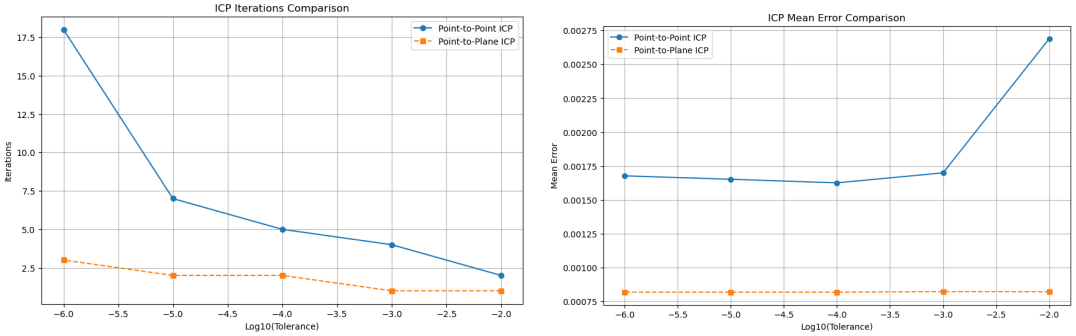


Figure 22: Visualization of Point2plane Aligned Meshes in MeshLab

We then compare the mean error and times of iteration of point2point ICP and point2plane ICP under different tolerence.



(a) ICP Iterations Comparison

(b) ICP Mean Error Comparison

Figure 23: Comparison Between Point2point ICP and Point2plane ICP

From the above plotted figure, we can observe that the method point2plane align converges significantly faster than using the method point2point align as well as has smaller mean error.

References

[PU] Computer Science Department Princeton University. Point-to-plane icp. https://www.cs.princeton.edu/~smr/papers/icpstability.pdf.