COMPGV18 Assignment 1

Student Name: Yifei Gao Student ID:13028702

Task 1

This part of the code is corresponding to the performICP() function in the code, the aim of this function is to align the mesh pair by minimise the point to point error. I have chosen bun000 and bun045 as mesh parir for testing, as they have large overlapping region. I've used nanoflann to build the k-d tree structure of bun000 for speeding up the nearest neighbor computation. There are two testing button to perform the ICP algorithm for 1 iteration and another button to perform the ICP algorithm till the meshes pair converge. The stopping criteria of it is either the alignment error change is smaller than the threshold I've set (change in error<1e-8), or the maximum iteration is reached. For this testing set, the point to point ICP algorithm runs for 83 iterations are the resulting alignment error is 0.0679945. Figure 1. shows the meshes before alignment and result after alignment.

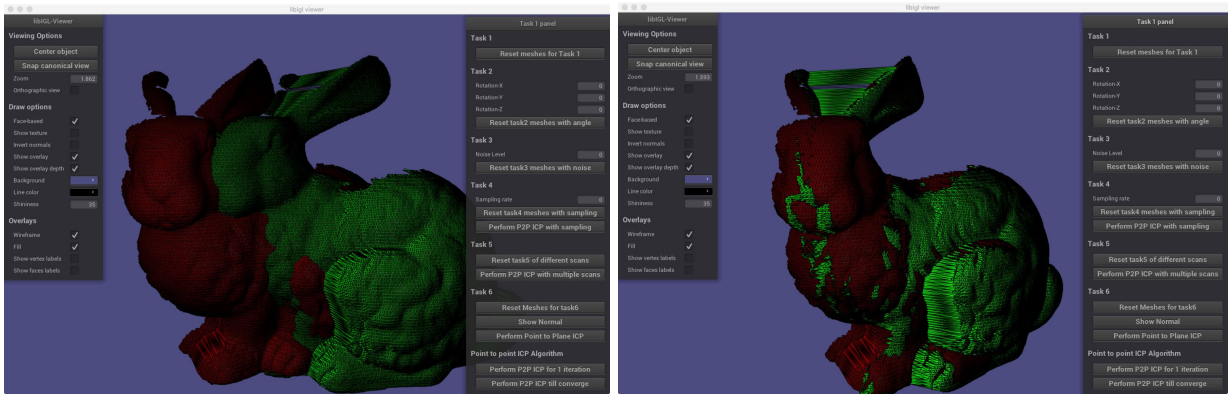


Figure 1. Result of point to point ICP.
(Left: Raw meshes; Right: Resulting meshes)

This function assumes the pairs of points have constant weight, but we can improve the optimization by giving different weight to these pairs based on confidence of the pairs. There are a few ways to compute this weight, for example:

- Assigning higher weights to pairs with smaller point-to-point distance. Where we compute and store the square distance of all pairs.
  $Dist(p_i, q_i) = \|p_i - q_i\|^2 \quad w_i = 1 - \frac{Dist(p_i, q_i)}{maxDist}$
- Weight the pairs based on the compatibility of their normals (Requires pre-calculation of the normal), where $w_i = n_p \bullet n_q$

The center of mass becomes: $\overline{P} = \frac{\sum w_i p_i}{\sum w_i}$, and the error matrix $A = \sum w_i (\hat{q}_i)(\hat{p}_i)^T$

Rest of the routine is the same as ICP with constant weight, where we uses SVD of A to build the rotation matrix R=UV' and estimate the translation using the R and the center of mass after

rotation. There is a tradeoff between the preprocessing of the weights and run-time without confidence weighting here.

Task 2

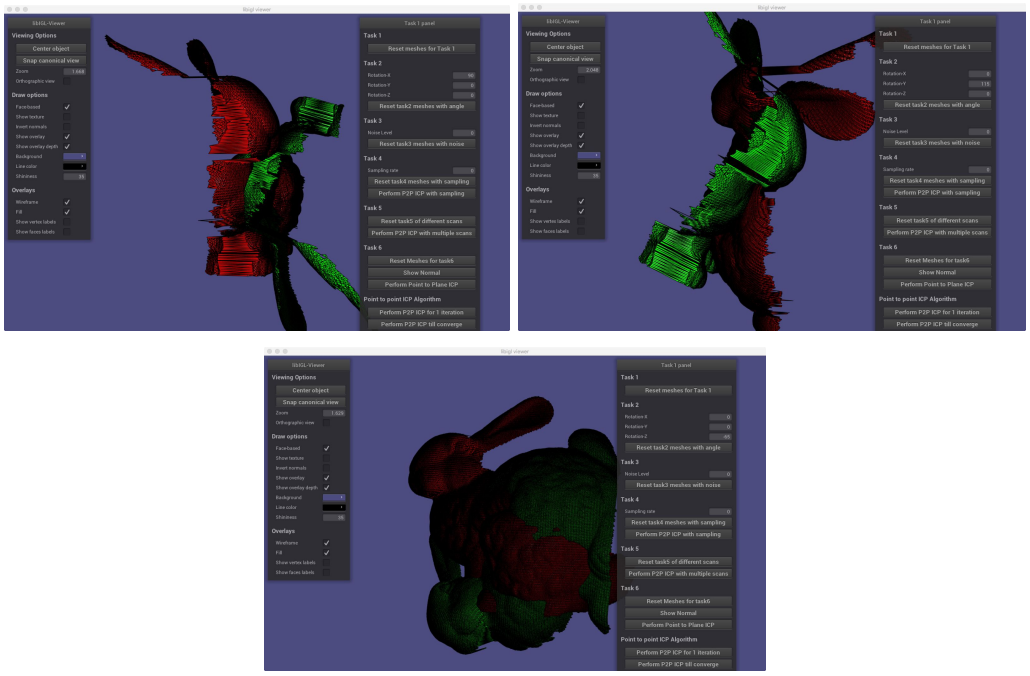
This part of the code uses the apply_rotation() function to rotate the object about different axis. This object I used for testing here is bun000, I've tried to experiment with different degrees of rotation on different axis, the resulting are:

- X-axis: -175<angle<90 the mesh and its rotation can still converge correctly, in general the number of iteration it takes for the mesh to align is growing slowly as the angle gets larger, the alignment error is similar to error on Y-axis. For any angle outside this range, the meshes results in incorrect alignment.

- Y-axis: -125<angle<115 the mesh and its rotation can still converge correctly, in general the number of iteration it takes for the mesh to align is growing as the absolute angle of rotation is growing, but the alignment error is similar. For any angle outside this range, these two meshes results in incorrect alignment.

- Z-axis: -65<angle<80 the meshes and its rotation can still coverage correctly, in general the number of iteration it takes for the mesh to align is similar around small rotation, as the rotation angle gets closer to the boundary, the number of iteration increasing dramatically. The resulting alignment error are similar for all correct conversions. For any angle outside this range, the meshes results in incorrect alignment.

Figure 3. Shows the incorrect alignment around different axis.
(Top left: Incorrect Alignment around X-axis; Top right: Incorrect Alignment around Y-axis
Bottom: Incorrect Alignment around Z-axis)



Task 3

This part of the code is corresponding to apply_noise() function is the code, the aim of this function is to perturb the model with zero-mean Gaussian noise based on the bounding box dimension of the object. I've used bun000 and bun045 as the mesh pair here and the noise is added to bun045 for testing. Figure 4. Shows the graph of the error size change with different noise level and number of iteration it takes.

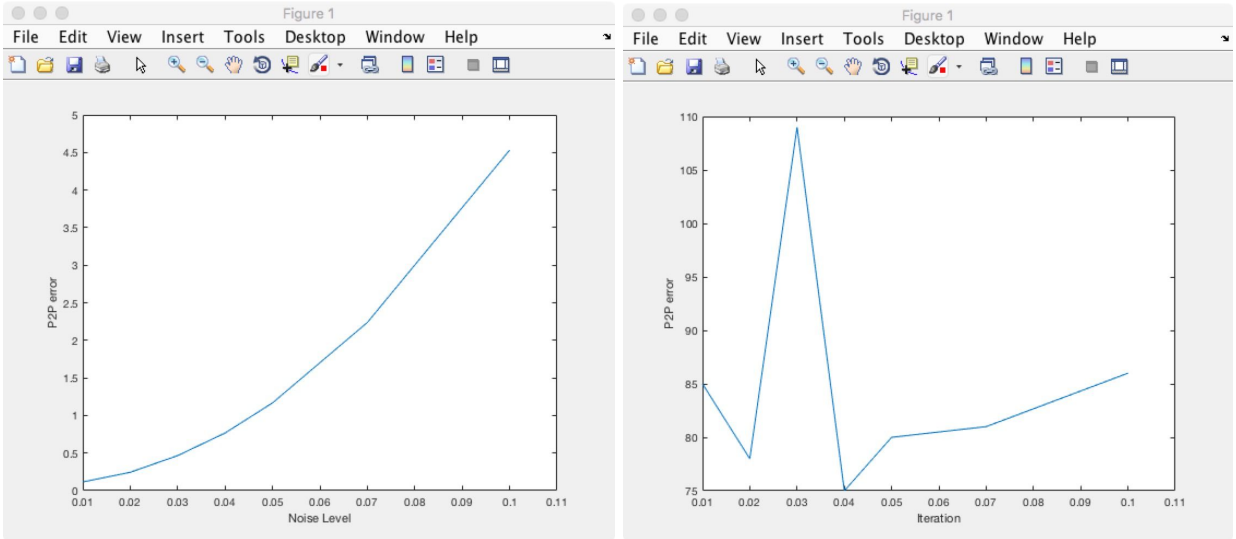


Figure 4. Left: x-axis noise level, y-axis total p2p error
Right: x-axis noise level, y-axis total iteration

As we can see from the graph, the p2p error is directly related to the noise level, as the noise level increases the total error increases as well. The number of iteration it takes for the ICP algorithm to terminate does not dependent on the noise level, as the iteration change didn't follow any pattern. Figure 5. Shows the Mesh with low and high noise level.

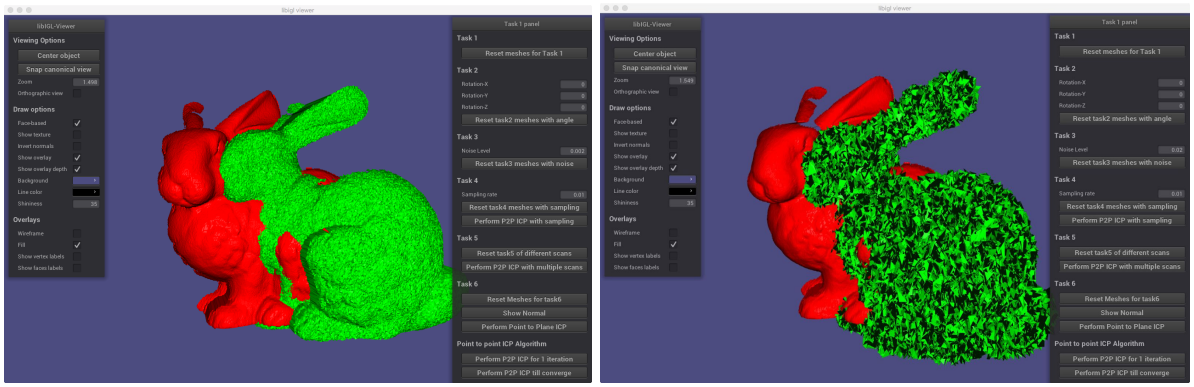


Figure 5. Left: Noise level = 0.002
Right: Noise level = 0.02

Task 4

This part is corresponding to uniform_sampling() function in the code, the aim of this function is to subsample the M2 with uniform random index. As the meshes are scanned from bottom to top with the same direction, by sampling the index of vertices uniformly we can sample the object uniformly as well. I've used the random engine and uniform_int_distribution to generate uniform random integer between 0 to the number of vertices. Figure 6. Shows the selected vertices at different sampling rate.

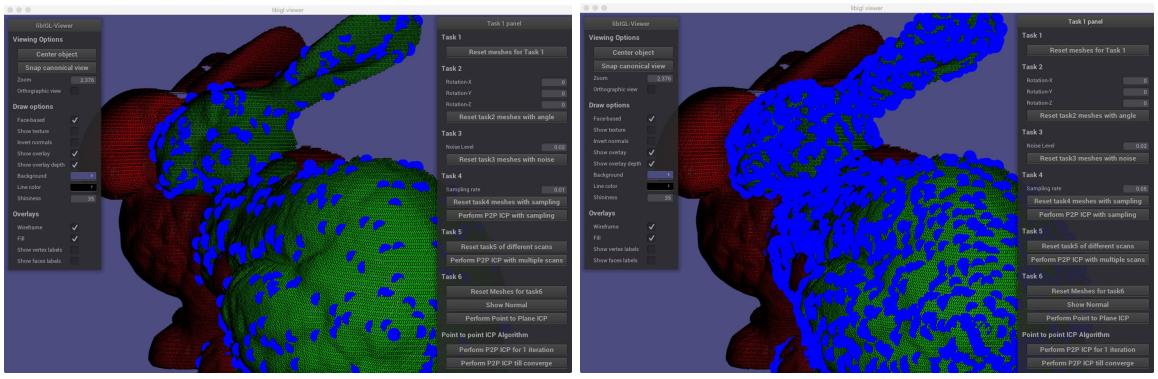


Figure 6. Left: Sampling rate = 0.01
Right: Sampling rate = 0.05

The run time of the ICP algorithm drops significantly as the the sampling rate get smaller. Figure 7. Shows how the accuracy and iteration change with increasing sampling rates.

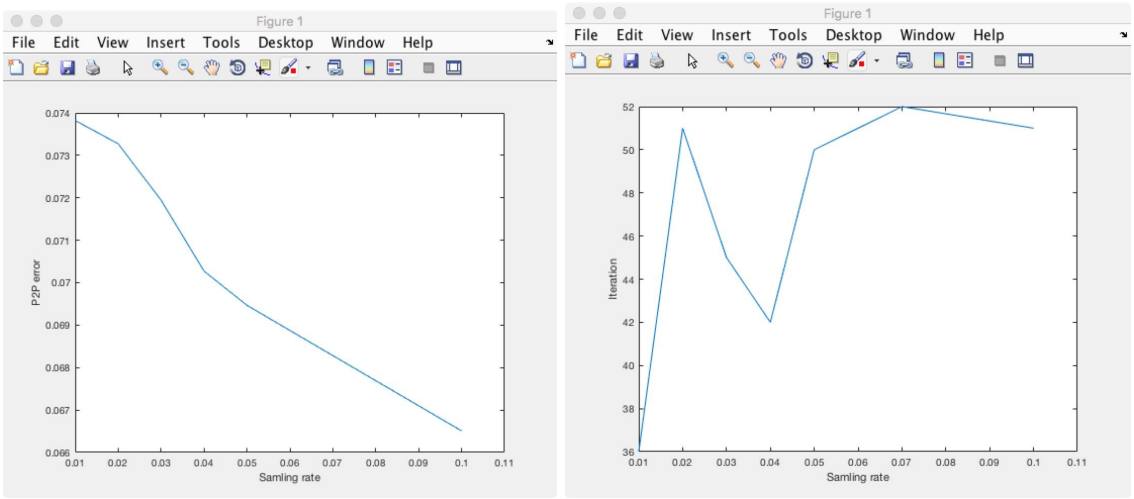


Figure 7. Left: x-axis Sampling rate, y-axis Total p2p Error
Right: x-axis Sampling rate, y-axis Total Iteration

In general, the total error is inversely proportional to the sampling rate, the total error decrease as we increases the sampling rate. Again the iteration is independent to the sampling rate, the number of iteration didn't follow any pattern here.

Task 5

This part of the code corresponding to the perform_icp_with_multiple_scans() function, the aim of this function is to align multiple meshes (n>2) to a common global coordinate frame. My approach is to find the closest points of current mesh with every other scans, optimise the transformation of current scan while the other scans are fixed. I subsampled current mesh and find the nearest point in all other scans, then build the matrix A with all the nearest points from all other meshes with the difference to their centre of mass. Then after we compute the optimise transformation for each scan, apply them to the meshes and do the next iteration. Figure 8. Shows the alignment result of 3 meshes and 5 meshes.

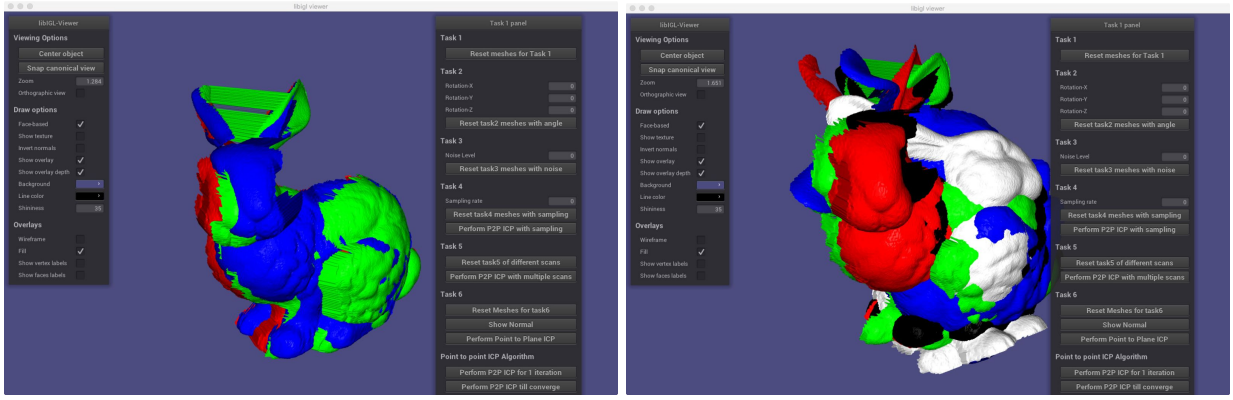


Figure 8. Left: Alignment result of 3 meshes (000,045 and 315) for 50 iterations;
Right: Alignment result of 5 meshes (000, 045, 180, 270, 315) for 50 iterations

The pros of this approach is that is does not registered the meshe pairwisely, so the alignment errors does not add up accumulatively. But the cons of this method is that the alignment here is greedy, so for some meshes pair like 000 and 180 will lead to incorrect alignment because their local alignment error is small that dirven the transformation towards it. I think this can be fixed by doing a pairwise alignment before my approach to get a rough alignment, then uses this greedy global alignment to reduce the accumulated error from pairwise alignment.

Task 6

This part of the code is corresponding to the performpoint2planeICP() function in my code, the aim of this function to minimise the alignment error by minimising the point to plane error of corresponding pairs. We first estimate the normal of each vertices based on the point cloud, which is done by estimateNormal() function in my code, in this part I uses a neighborhood of 8 samples to estimate the normal. Figure 9. Shows the estimate normal of the mesh.

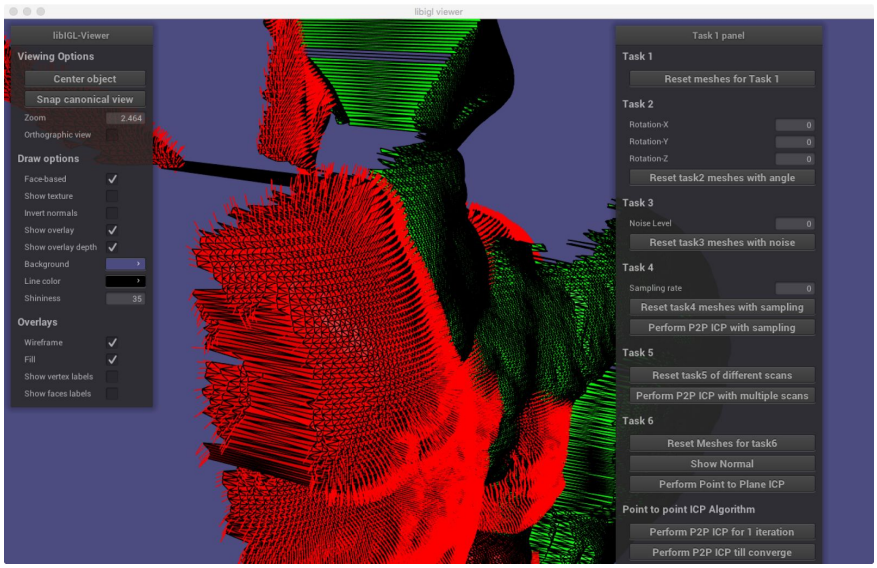


Figure 9. Estimated normal of vertices, show as red edges.

I used the same meshes pair to test the performance of point to point and point to plane ICP algorithm. The point to plane ICP takes 11 iteration to align the meshes and the resulting alignment error (In term of point to point error) is 0.067067. It has better performance than point to point ICP in this case with fewer iteration and better alignment error. Figure 10. Shows the alignment result of point to point and point to plane ICP.

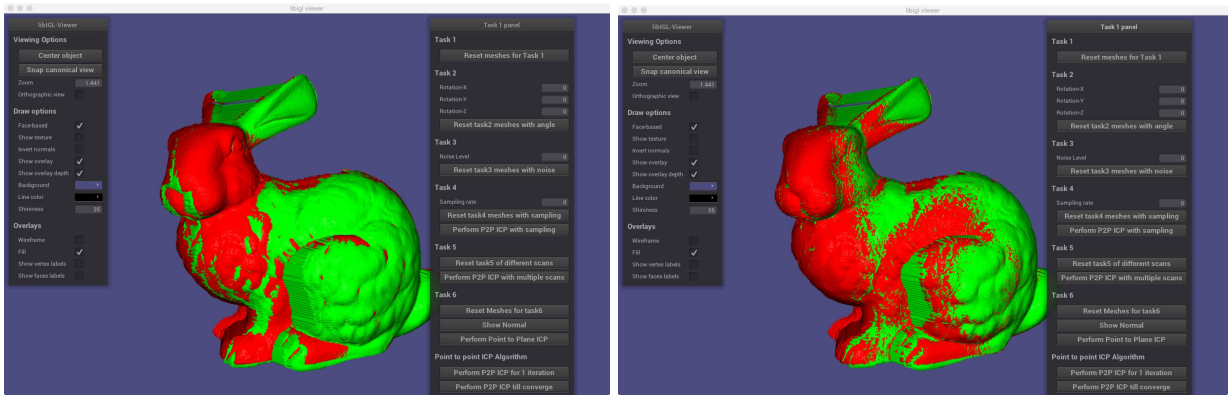


Figure 10. Left: Result of point to point ICP
Right: Result of point to plane ICP