Assignment 1

Attempted Questions

1. Implementation of point-to-point ICP to align two meshes.
    a. Loading the meshes and experimentally applying transformation in order to test the ability to use a mesh viewer
    b. Writing the algorithm for point-to-point ICP and presenting the results
    c. Writing the derivation of the error for weighted correspondences between points
2. Iteratively rotate a copy of a mesh using different angles in order to observe the results of the ICP algorithm applied to the rotated copy and the original mesh
3. Iteratively perturb a mesh M2 by adding noise to its vertices in order to examine the effects on the ICP algorithm
4. Speed up the alignment computation of two meshes M1 and M2 by subsampling appropriately and report the effect subsampling has on the performance of the ICP algorithm
5. Given multiple scans M1, M2, … M5 align all of them to a global coordinate frame

Goals Achieved

1. I managed to successfully align two meshes using the point-to-point ICP algorithm
    a. Managed to upload meshes and apply trivial transformations such as translation. I observed the results using MeshLab
    b. I implemented the point-to-point algorithm successfully, achieving a good alignment of the meshes observed using MeshLab, and I examined the evolution of residual error
    c. I found the equation of the derivation of the error by “t” (translation) having weights corresponding to the pairs of points from P and Q
2. I applied a transformation of rotation to a mesh with multiple angle values and I observed the evolution of the ICP algorithm for every iteration by examining the convergence speed and the residual errors
3. I applied gaussian noise with different standard deviation values to the vertices, and I observed the performances of the ICP algorithm as the level of noise increased. I also adjusted the value of noise I added to each vertex to be inside the initial bounding box of the mesh
4. I observed the evolution of the results provided by the point-to-point algorithm, as I used different amounts of samples. The samples are the indices of points from the mesh that I use to find their correspondents and therefore establish the <pi, qi> pairs. Therefore, as the number of samples decreases, the number of correspondent pairs also decreases by the same amount
5. I uploaded the meshes taken at 0, 45, 90, 270 and 315 degrees and I aligned them 2 by 2 in order to obtain a final set of 5 meshes that are aligned and form a big proportion of the 3D model of a bunny. Moreover, I tried to align the mesh taken at 180 degrees with the one taken

at 90 degrees getting unsatisfactory results, and I also tried to align the 180 degrees mesh to the 3D model obtained previously in order to exemplify that a mesh that varies by such a big angle cannot be aligned with the other meshes

6. I determined the normal to each vertex and I colored the vertices according to their normals. I implemented a point-to-plane ICP algorithm by introducing the normals in the equation of the error, and modifying the process of obtaining the transformation matrix.

Libraries and tools used

- numpy: used for matrix operation, norm of vectors and initialization of matrices
- matplotlib: used to plot the results as linear graphs
- trimesh: used to load the meshes in “ply” format, get information about the mesh, view the meshes using the “Scene” method and save the meshes
- scikit-learn: used to create an instance of KDTree
- MeshLab: tool used to view the meshes and their alignments

Solutions

Question 1

a. In order to check the functionality of my environment I used the trimesh library to manage meshes. I loaded 2 meshes using the method “trimesh.load_mesh()” to load meshes from files. To get familiar with the manipulation of meshes I applied a transformation on a mesh using “trimesh.apply_transform()”, passing as argument a transformation matrix.
The transformation is supposed to translate the mesh by 0.2 on the x axis, and is presented below:

$$T1 = \begin{pmatrix} 1 & 0 & 0 & 0.2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

To the second matrix I applied a translation of -0.2 on the z axis.

$$T2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

After translating a mesh, I used “trimesh.export()” method to save the meshes. To view the result I imported the meshes in MeshLab. Below I am attaching a screenshot of the result.

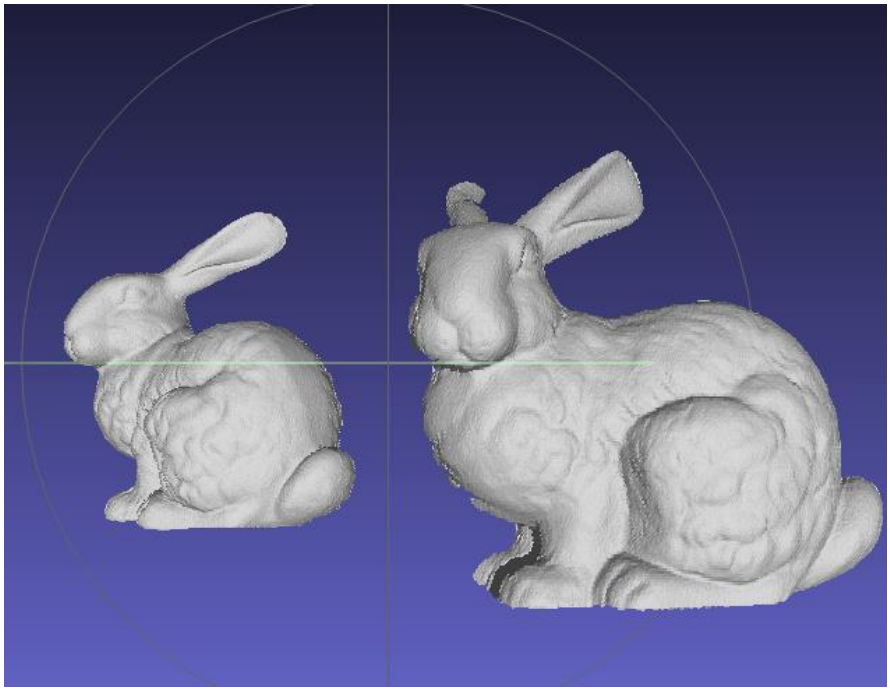

Figure 1: Viewing 2 bunnies using translation

As the bunnies do not overlap, we can observe that one of them is translated.

b. In order to implement the point-to-point ICP algorithm to align the second mesh (Q) to the first one (P), I followed the following steps:
    - Firstly, I chose two meshes that are nearly aligned and have a significant overlap: the mesh of the bunny at 0 degrees and the one at 45 degrees.
    - Secondly, I wanted to establish pairs of <pi, qi> points that correspond. In order to do that I created an instance of a KDTree on the vertices of the first mesh, and I extracted 5000 samples from the second mesh in order to find their matching pair from the first mesh. To find their match I searched in the KDTree for the closest point from the first mesh to each of the sampled points from the second mesh. However, the pairs are not always accurate so I had to reject some of the pairs. To reject the pairs I used a least squares plane fitting algorithm to find the normal of every vertex, corresponding to the plane a vertex forms with his 2 closest neighbours. If we know the difference of the angles between the 2 meshes is 45, I used a threshold of 47 degrees for the angles between the normal of points pi and qi to reject pairs.

- After establishing the <pi, qi> pairs, I normalize the sampled points by subtracting the centroids of P and Q, the 2 sets of points for the 2 meshes. Having normalised the points, now I can create the A matrix,

$$A = Q^T x P$$

- Having the matrix A, I use the SVD algorithm in order to create the rotation matrix and the translation vector.

$$R = V\ x\ U^T$$
$$t = centroidP - R\ x\ centroidQ$$

- Now I can form the transformation matrix, and I apply the transformation to the mesh. I calculate the error at this iteration and the difference between the values of samples before and after this iteration.
- If the difference between the values of the samples is bigger than 1e-8, then it performs another iteration. The maximum number of iterations is 200. The function returns the errors for each iteration and the 2 meshes.

The resulted alignment of the two meshes taken at 0 and 45 degrees has a residual error of 0.021. The error for the alignment of bunny270 to bunny315 is 0.34.

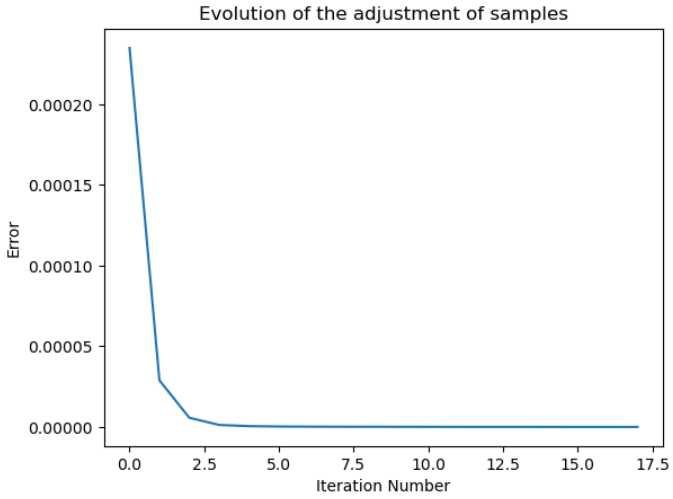

Figure 2: Difference between the values of samples between the start and the finish of the iteration for bunny000 and bunny045

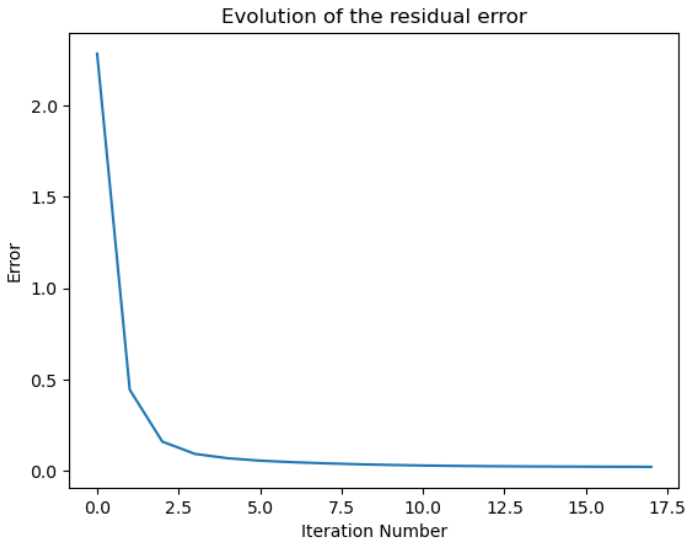

Figure 3: Evolution of the residual error for bunny000 and bunny045

Figure 4: Alignment of bunny045 to bunny000

Figure 5: Alignment of bunny045 to bunny000, from another perspective

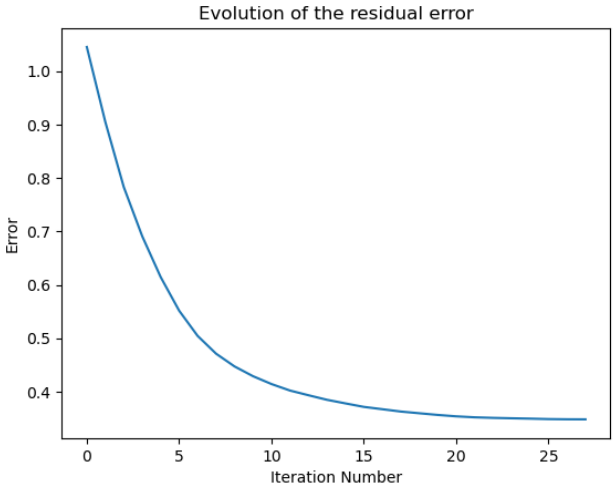

Figure 6: Evolution of residual error for bunny270 and bunny325

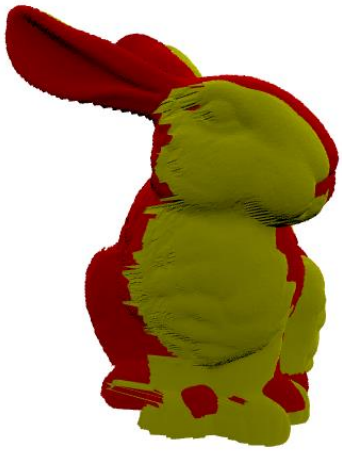

Figure 7: Alignment of bunny315 to bunny270

Figure 8: Alignment of bunny315 to bunny270 from another perspective

c. Having the optimization:

$$E(R,t) = \sum_{i=1}^{n} \left(wi * \left|\left| Rpi + t - qi \right|\right|^2 \right)$$

To minimize the error the derivative by R and the derivative by t should both be 0, with the constraints:

$$\det(R) = 1 \; and \; R \; x \; R^T = 1$$

Therefore:

$$\frac{dE}{dt} = \sum_{i=1}^{n} \left(2 * wi * \left|\left| Rpi + t - qi \right|\right| \right) = 0$$

$$\frac{\sum_{i=1}^{n}(wi * Rpi)}{n} + \frac{(\sum_{i=1}^{n}(wi)) * t}{n} - \frac{\sum_{i=1}^{n}(wi * qi)}{n} = 0$$

$$t = \left(\frac{\sum_{i=1}^{n}(wi * qi)}{n} - \frac{\sum_{i=1}^{n}(wi * Rpi)}{n} \right) * \frac{n}{\sum_{i=1}^{n}(wi)}$$

Notations:

$$p^- = \frac{\sum_{i=1}^{n}(wi * pi)}{n} * \frac{n}{\sum_{i=1}^{n}(wi)} = \frac{\sum_{i=1}^{n}(wi * pi)}{\sum_{i=1}^{n}(wi)}$$

$$q^- = \frac{\sum_{i=1}^{n}(wi * qi)}{n} * \frac{n}{\sum_{i=1}^{n}(wi)} = \frac{\sum_{i=1}^{n}(wi * qi)}{\sum_{i=1}^{n}(wi)}$$

$$\widetilde{pi} = pi - p^-$$

$$\widetilde{qi} = qi - q^-$$

Introducing t in the initial equation we get:

$$E(R,t) = \sum_{i=1}^{n} \left(wi * \left|\left| Rpi + q^- - p^- - qi \right|\right|^2 \right)$$

$$= \sum_{i=1}^{n} \left(wi * \left|\left| R(pi - p^-) - (q^- - qi) \right|\right|^2 \right) = \sum_{i=1}^{n} \left(wi * \left|\left| R\widetilde{pi} - \widetilde{qi} \right|\right|^2 \right)$$

$$= E(R)$$

From this point onwards the calculus is similar with the calculus for the initial error equation.

Question 2

To solve this task, I used the mesh taken at a 0 angle, and I rotated it with 0, 5, 10, 15, 20, 25 and 30 degrees. To rotate the mesh, I created a transformation matrix where I changed the parameters of rotation with the corresponding angle.

Now, I am going to present the result for every value of the rotation angles:

0 degrees: it's already aligned, the algorithm stops after the first iteration and the error is 0.0

5 degrees: the error is 0.00159

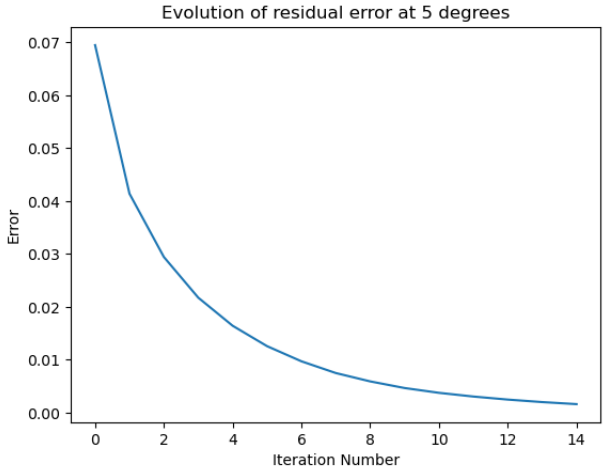

Figure 9: Evolution of residual error for 5 degrees

10 degrees: the error is 0.00215

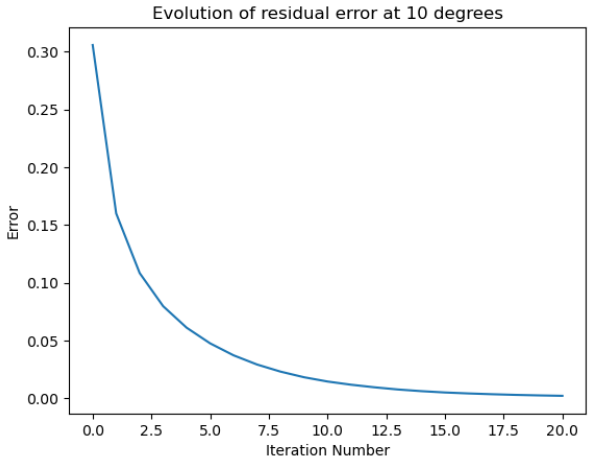

Figure 10: Evolution of residual error for 10 degrees

15 degrees: the error is 0.00258

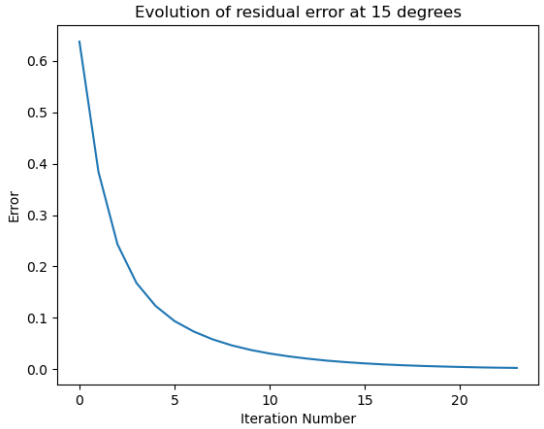

Figure 11: Evolution of residual error for 15 degrees

20 degrees: the error is 0.00269

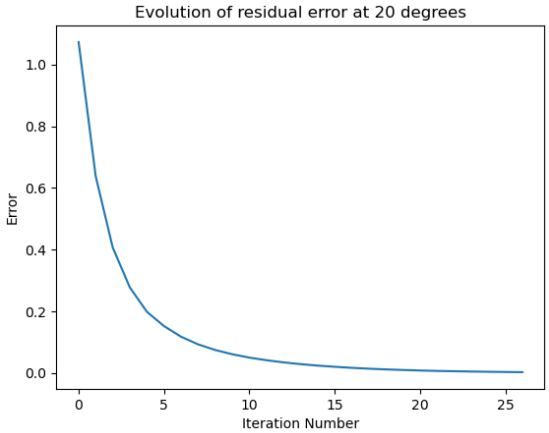

Figure 12: Evolution of residual error at 20 degrees

25 degrees: the error is 0.00254

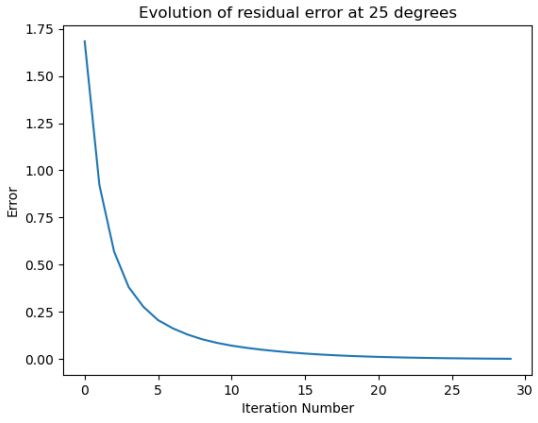

Figure 13: Evolution of residual error at 25 degrees

For all of the rotated meshes, the same vertex indices were used when sampling. The error has as upwards trend as we increase the value of the angle. The only exception is between 20 and 25 degrees rotation but this can be influenced by the random sample.

Another observation is that the error of the first error is bigger as we increase the angle, which is caused by moving further the corresponding points by rotating the mesh.

Moreover, the convergence tends to be achieved in more iterations as the angle increases. For the rotation of 5 degrees the convergence is achieved in 14 iterations, whereas for an angle of 25 degrees, almost 40 iterations are needed.

Question 3

For exemplification I used two meshes, a mesh M1 that remains unchanged and a mesh M2 that is modified. The mesh M2 is perturbed with different levels of noise in order to study the impact of noise on the point-to-point ICP algorithm. To make these observations I added gaussian noise with mean 0 and standard deviation, in turns, 0.0002, 0.001, 0.005, 0.01, 0.05.

Figure 14: Adding 0.0002 std noise to bunny000

Without the addition of noise, the error is 0.025

For 0.0002 noise the error is 0.023.

For 0.001 noise the error is 0.032.

For 0.005 noise the error is 0.135.

For 0.01 noise the error is 0.496.

For 0.05 noise the error is 5.30.

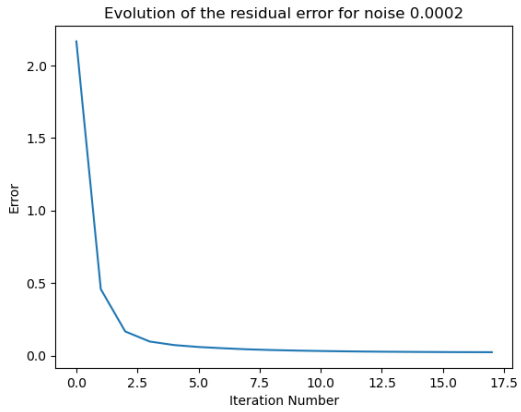

Figure 15: Evolution of residual error for noise 0.0002

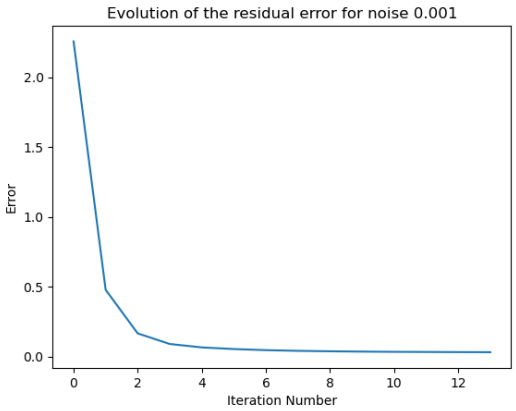

Figure 16: Evolution of residual error for noise 0.001

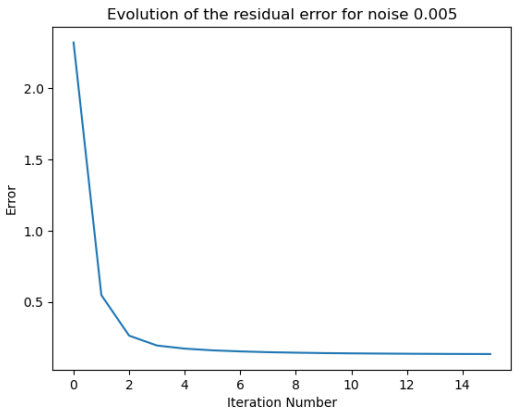

Figure 17: Evolution of residual error for noise 0.005

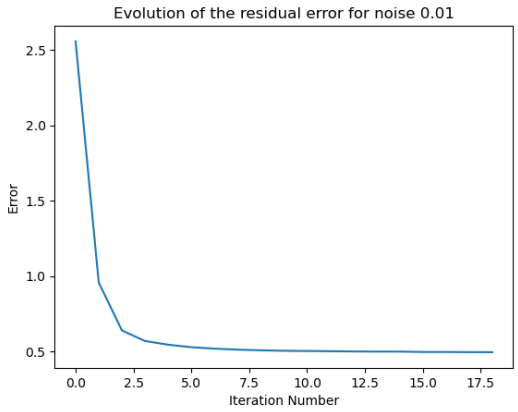

Figure 18: Evolution of residual error for noise 0.01

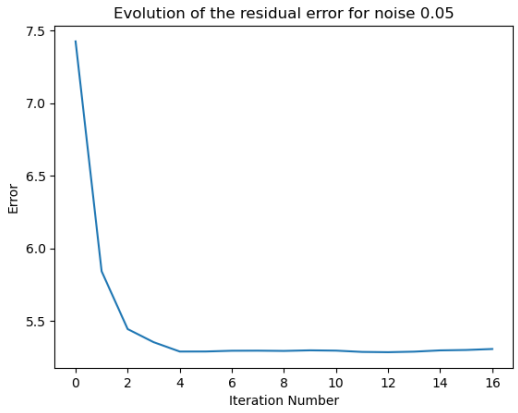

Figure 19: Evolution of residual error for noise 0.05

Question 4

My point-to-point ICP algorithm method is designed to get passed as argument the samples that correspond to the indices of points from the second mesh for which I find the appropriate points from the first mesh. Therefore, in order to solve this task, I generated arrays of different sizes of samples and I used them to call the ICP method.

In order to study the performances of the algorithm for different number of samples, I chose to run on 10, 50, 100, 500, 1000, 3000 and 5000 samples.

The more samples are to be aligned, the more difficult is to find a perfect mapping to align the samples from 2 different meshes. The higher the number of samples, the higher is the chance to randomly pick a point that is only visible in the second mesh, as it is not part of the first mesh. This type of points are further than the first mesh even though we align the 2 meshes, and it adds higher values to the residual error.

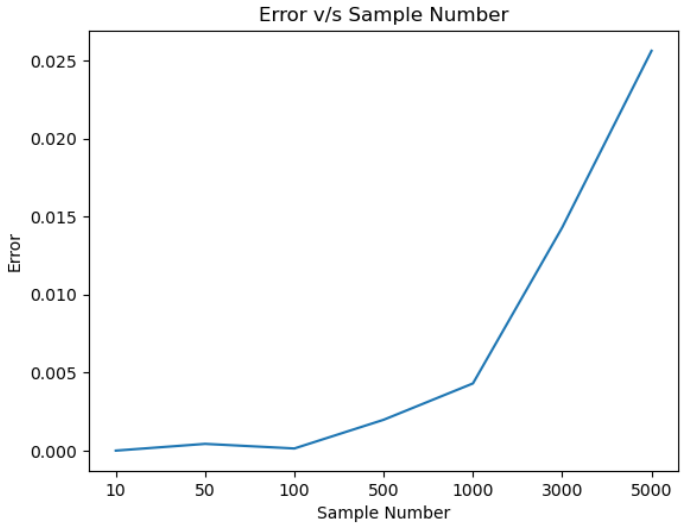

Figure 20: Residual error compared to the number of samples

Question 5

In order to align 5 meshes M1, M2, … M5 I picked the 5 meshes that are on a smaller than 45 degrees angle from other meshes: bunny000(M1), bunny045(M2), bunny090(M3), bunny270(M5), bunny315(M6).

My strategy is to align the meshes 2 by 2 by picking the closest one. I used bunny 270 as reference, and I aligned all the other meshes to this one. To present the sequence of aligned meshes I will use the notation Mx_to_y to reference the mesh resulted from aligning mesh x to mesh y.

The sequence of alignments I did is:

- M6 **aligned to** M5 resulting in M6_to_5
- M1 **aligned to** M6 resulting in M1_to_6
- M1_to_6 **aligned to** M6_to_5 resulting in M1_to_5
- M2 **aligned to** M1 resulting in M2_to_1
- M2_to_1 **aligned to** M1_to_6 resulting in M2_to_6
- M2_to_6 **aligned to** M1_to_5 resulting in M2_to_5
- M3 **aligned to** M2 resulting in M3_to_2
- M3_to_2 **aligned to** M2_to_1 resulting in M3_to_1
- M3_to_1 **aligned to** M2_to_6 resulting in M3_to_6
- M3_to_6 **aligned to** M2_to_5 resulting in M3_to_5

After these alignments, all the 5 meshes are aligned. The result is viewed by adding the meshes M5, M6_to_5, M1_to_5, M2_to_5, M3_to_5 in the Scene.

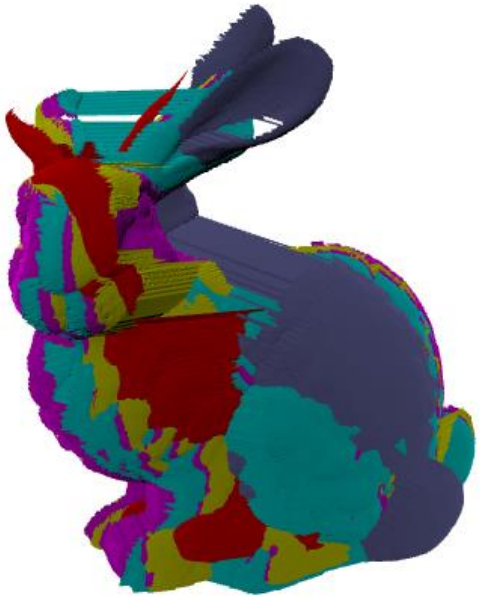

Figure 21: Result of the alignment of the 5 meshes

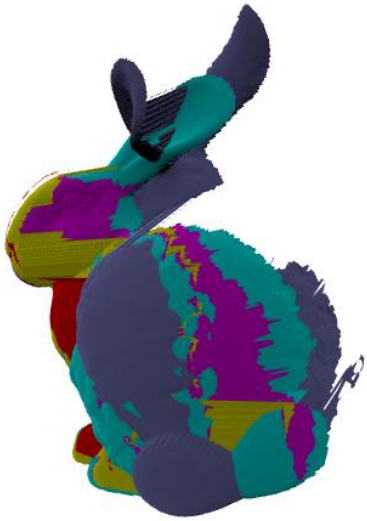

Figure 22: Result of the alignment of the 5 meshes from other perspective

While the alignment manages to bring together the 5 meshes, the dark blue bunny doesn't perfectly fit the rest of the alignment (M3_to_5).

In order to prove that the bunny180 cannot be aligned with the rest of the meshes, I generated the alignment of bunny180 to bunny270. This huge difference in the alignments causes bad results.

Figure 23: bunny180 aligned to bunny270

Figure 24: bunny180 aligned to bunny270 from other perspective

Therefore, introducing bunny180 in the previous alignment, as reference mesh, the following disastrous results are obtained:

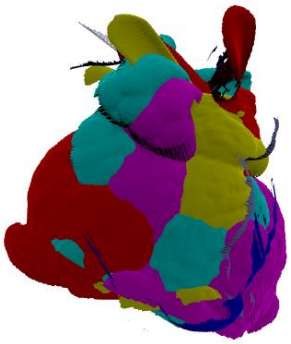

Figure 25: Alignment of all the meshes

The pros of this method is that is easy to understand and to use. If the meshes are initially pretty much aligned than the method produces good result. However, a disadvantage is that we need to know the angle of the meshes before we use them (information such as bunny045 is taken at 45 degrees). Also, the method is not performant if one of the meshes has a significant difference in its alignment compared to the other meshes (such as bunny180). Also, this method requires a lot of calls of the ICP function because it aligns each mesh multiple times until it reaches the position of the reference mesh.

Question 6

To obtain the normal for each vertex I used the least square based normal estimation as presented in the second tutorial. For the visualisation of the normal I coloured each vertex with the value of it's normal, getting the following scene:

Figure 26: Coloured bunny000 based on normals

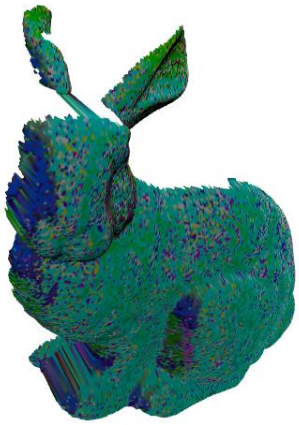

Figure 27: Coloured bunny000 based on normal from other perspective

To implement the point-to-plane ICP algorithm I followed the steps presented in the "Point-to-Plane ICP Algorithms for Surface Registration" by K. L. Low. (https://www.comp.nus.edu.sg/~lowkl/publications/lowk_point-to-plane_icp_techrep.pdf)

The main differences between the point-to-point and point-to-plane algorithms were:

- Use of the normals, getting the normal values from the vertex colour.
- For point to plane we write the difference between corresponding points as M*pi – qi, where M is the transformation matrix, while for the point-to-point implementation we write it as R*pi + t – qi
- For the point-to-plane approach we are trying to solve an equation Ax=b, where A and b have a specific format described in the paper
- SVD elements are used to find the pseudo-inverse of A, which helps to identify the x vector that contains all the parameters of the transformation M

The error for aligning bunny045 to bunny000 is 0.0106.

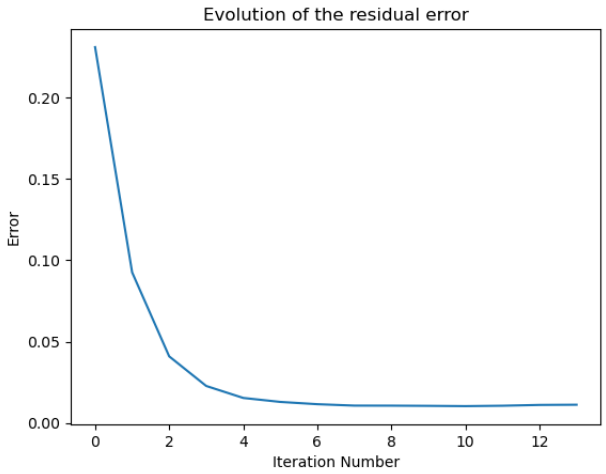

Figure 28: Evolution of the residual error for bunny045 to bunny000

Figure 29: Bunny045 aligned to bunny000

Figure 30: Bunny045 aligned to bunny000 from another perspective

The error for aligning bunny315 to bunny270 is 0.0978.

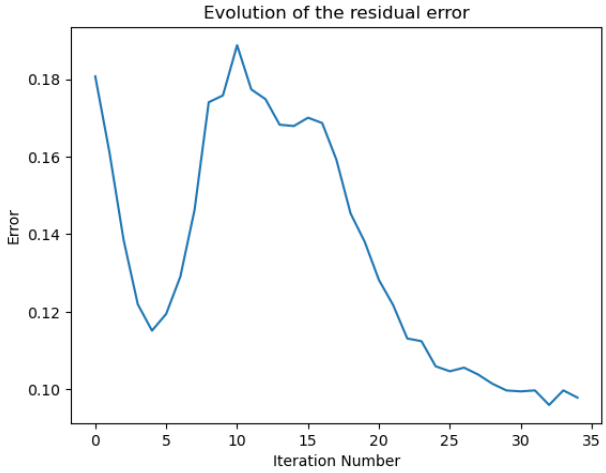

Figure 31: Evolution of the residual error for bunny315 to bunny270

Figure 32: Bunny315 aligned to bunny270

Figure 33: Bunny315 aligned to bunny270 from another perspective

The point-to-plane implementation has a significant smaller error for these 2 alignments. For the first alignment 0.1 compared to 0.2, and for the second alignment 0.09 compared to 0.34. It also seems for the first scenario that the solution converges faster. However, the error converges smoother in the point-to-point ICP algorithm, as we can observe a strange shape in the plot for the second alignment.