COMP0119: Acquisition and Processing of 3D Geometry

Coursework 1: Iterative Closest Point

This report will demonstrate the algorithm implemented for the Iterative Closest Point.

Task 1: Point-to-Point Alignment

The basic ICP algorithm can be a breakdown of following steps:

1. Select source points
2. Match points in the other mesh
3. Reject bad pairs
4. Compute rigid transform
5. Detect error and check if stop iteration

The first task requires us to align M2 to M1 using such an algorithm. After loading two OFF meshes which are converted from PLY using MeshLab, we perform the KNN search for M2 to find nearest neighbour pointsS2 on M1 using all available vertexS1 (subsample will be utilised later in task 4) and then reject distant vertexS3. The rejection can be done using normal compatibility, boundary condition and distant points. In this task we simply reject distant vertex by using a threshold - ***k*** times median distance.[3] After this step we obtain a refined version of M2 which contains matched vertex that are 1-to-1 mapping to the M1. Once we have the mapping, we start calculating the rigid transformS4 which contains the rotation **R** and translation **T**. The rigid transform can be found when is minimised, where and represent pairs of matched vertex on each mesh; this process can be estimated using SVD. First, we calculate the mean vertex value and from matched point sets. Then are able to compute their covariance A, which is given by . Provided that , we are able to find that and use this result to obtain . By applying rigid transform to M2, the position and rotation of it will be updated and then we may repeat the steps above until we reach the reasonably accurate alignment. However, in most cases, there is an extra stepS5 that needs to be done before we apply the rigid transform. We would like to know whether the alignment has reached a point where it is already aligned correctly, and the next rigid transform brings the error to misalign it again. In this task, we have tried implemented such a function that calculates the sum of squared differences between a matched set and a pre-transformed set so that we know how much it changes compared with the previous transform. The difference will gradually decrease and if it increases again then we may terminate the iteration. It works but still requires some adjustment so we have commented out the code segment, but they can still be used if required.

Additionally, this task requires us to assign a weight to neighbour points in so that prioritises those more important points. The alignment error function is provided as

By using similar method provided in the best rigid transform handout:

Thus, we have:

Therefore, we have:

And:

Similar to the A without weight, we are now able to construct a new matrix A:

Because the q and p changed position, we swap U and V in R expression and also change the t expression respectively.

The results from task 1 are provided below:

A picture containing yellow, indoor, orange, sky

Description automatically generated A picture containing yellow, artichoke, sky, orange

Description automatically generated

*Result at 10th and 50th iteration*

A picture containing sky

Description automatically generated A picture containing yellow, sky

Description automatically generated

*Result at 100th iteration with non-overlapping area marked out and at 150th iteration*

Task 2: Rotation Matching

This task requires to produce a rotated version of M1, which can be simply done with the GUI interaction. The GUI provides three text fields for x, y, z-axis rotation in degree, and our goal is to check how well the algorithm can handle different degrees of misalignments. For ease of demonstration, the model used for this task is “bun000.off”, and the initial rotation degrees for x, y, z-axis are 0, 0, 0 respectively. The results are provided below:

|  |  |
| --- | --- |
| Z-Rotation | Aligned with M1 |
| -75 | No |
| -60 | No |
| -45 | No |
| -35 | Yes |
| -30 | Yes |
| -15 | Yes |
| 15 | Yes |
| 30 | Yes |
| 35 | Yes |
| 45 | No |
| 60 | No |
| 75 | No |

A picture containing sky, yellow, table

Description automatically generated A picture containing sky, fruit, indoor

Description automatically generated

*M2 (rotated M1, 45 degree at z-axis) and the alignment result*

From the result, we can notice that the ICP algorithm is able to align slightly rotated version of itself successfully. However, once the rotation exceeds specific values, it will not be able to recover it no matter how many iterations we set. Thus, we can conclude that the ICP algorithm is sensitive to initial misalignment and in this case, the maximum initial misalignment is approximately 35 degree of rotation on z-axis either clockwise or anticlockwise.

Task 3: Adding Noise

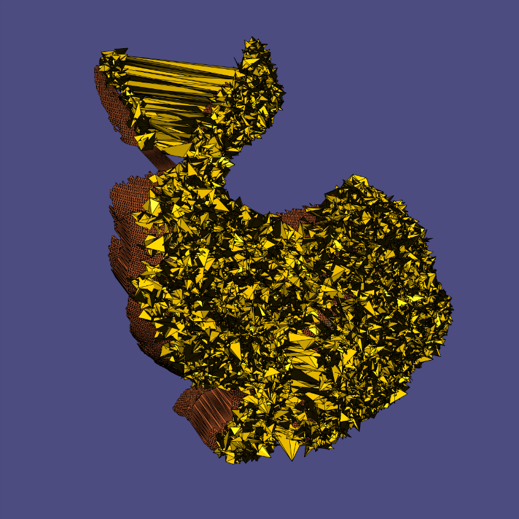
This task requires us to add zero-mean Gaussian noise to M2 and use the noised M2’ to match M1 using ICP algorithm. The standard deviation is used to obtain the overall noise level and it will be scaled and distributed to each axis based on the bounding box dimension.

A picture containing sky, holding, indoor

Description automatically generated A close up of a yellow wall

Description automatically generated with low confidence

*Noised M2 and matched M2 (sd = 0.5)*



*Matched M2 (sd = 3.0)*

The results are provided below:

|  |  |
| --- | --- |
| Standard Deviation | Aligned with M1 |
| 0.1 | Yes |
| 0.25 | Yes |
| 0.5 | Yes |
| 1.0 | Yes |
| 2.0 | Yes |
| 3.0 | Yes |

By observing the results, we can find that this ICP algorithm is capable of dealing with distorted meshes. However, the accuracy deceases significantly when we add more noises to the model. We can conclude that this algorithm has a very good tolerance of noise.

Task 4: Subsampling

This task requires us to subsample our vertex in order to speed up the matching process, and compare the results obtained with increasing subsample rate. The subsampling rate can be specified using the GUI, the higher the subsample rate uses, the smaller the point set becomes, for example, 99% subsample rate will remove 99% of the vertex from the original mesh.

There are many different algorithms to subsample the point set, such as random sampling, uniform sampling or even more advanced stable sampling. In this task, we implemented random and uniform sampling, but we will focus on using random sampling and see how well it works with the ICP algorithm.

The random sampling will randomly pick vertex from the mesh, for example, using 70% subsample rate will preserve approximately 30% of the vertex on the given mesh without having any repeating index. In order to benchmark its speed versus accuracy, we set the iteration number as a constant value of 300 and only adjust the subsample rate. The results are provided below:

|  |  |  |  |
| --- | --- | --- | --- |
| Subsample Rate | Aligned with M1 | Accurate | Time Taken |
| 1% | Yes | Yes | 17.64s |
| 15% | Yes | Yes | 15.69s |
| 35% | Yes | Yes | 12.93s |
| 50% | Yes | Yes | 11.02s |
| 65% | Yes | Yes | 9.08s |
| 85% | Yes | Yes | 6.58s |
| 99% | Yes | Yes | 4.65s |

By observing the benchmarking result, we may suggest that subsampling significantly increases the speed of the alignment process, and at the same time, maintains the accuracy. We are not able to confirm that subsampling always returns accurate results given the fact that the meshes we use to perform this benchmark contains more than 40,000 points, and even 0.1% of then can provide enough reference points for the matching process.

Task 5: Multiple Meshes Alignment

This task requires us to align at least five meshes together to form a complete mesh. As suggested in task 1, our basic ICP is tested on meshes that have large overlapping area. However, in this task, it is not quite possible to align multiple meshes together perfectly without any manual adjustment. For example, it is doubtful to align the left side of a rabbit with the right side of the same rabbit if there is less or no overlapping area at their boundary. Therefore, one possible solution is cherry-picking the provided mesh files and put them into a specific alignment sequence so that they have a higher chance of aligning together. However, this is not a general solution and may not apply to other models such as cats.

In this task, we have tried multiple ways to align multiple meshes. The first way I tried here is to align V2 to V1 together (for about 300 iterations) then we align V3 to the merged mesh of V1 and V2 and keep doing this process for the following meshes. (File sequence: 000, 045, 090, 180, 270)

A close up of a colorful background

Description automatically generated

*Method 1*

The problems that this method has are quite obvious. First, as suggested in the lecture note, if we have multiple meshes and we align them with the previous one, it will accumulate errors. For example, if V1 and V2 are not correctly aligned, and we immediately align V3 to the previous result, this will produce a chain effect on the following alignments: if V3 is not correctly aligned, so is V4.

Another method I tried here is cherry-picking. This somehow destroy the purpose of using the algorithm to automate such a process, but this does provide a better result:

A picture containing aircraft

Description automatically generated

*Method 2*

In conclusion, the basic ICP algorithm heavily relies on the overlapping areas between different meshes. Combined with the finding in previous tasks, we may also say that if the mesh is highly misaligned and contains minimal regions for point matching, it is not possible to align them automatically, human intervention is necessary in such cases.

Task 6: Point-to-Plane Alignment

This task requires us to use the point-to-plane ICP algorithm instead of the basic point-to-point approach. This algorithm is based on the error equation with normal information:

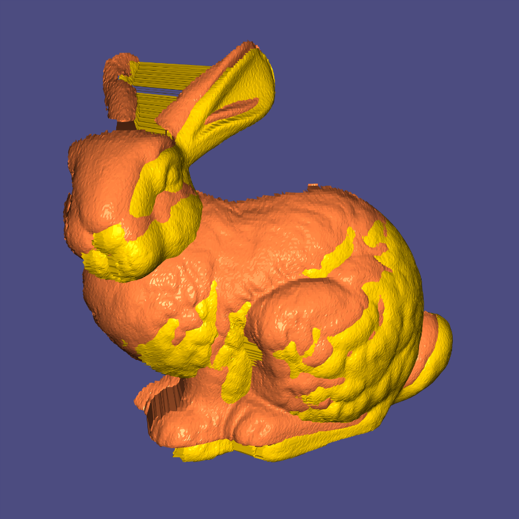
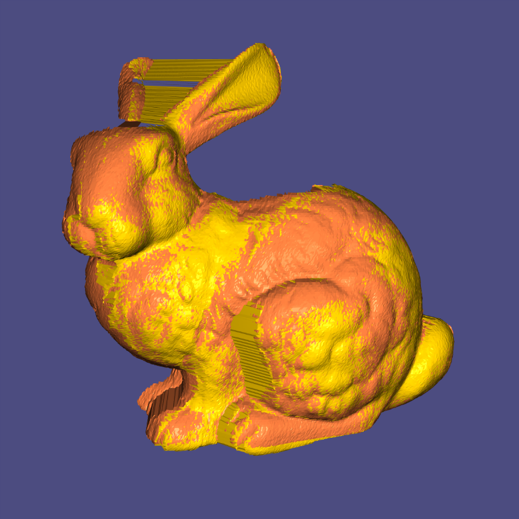
The new variable n represents the vertex normal of q, which can be computed as following steps:

1. For each point in mesh q, find nearest x neighbors.
2. Use igl::fit\_a\_plane() taught in tutorial 2 to calculate the normal using obtained points.

The ICP algorithm is similar to the basic version, we still need to select a set of points and match those points in the other mesh and do the rejection. The next step is a bit different, we will also use the normal information of matched points (after the rejection) and use them to calculate the rigid transform. This process can be turned into a SLE question [2]:

1. Construct matrix A, which consists of n by 6 elements:
2. Construct matrix b, which contains:
3. Solve the equation and get:
4. Construct matrix R and vector T
5. Apply rigid transform

The results are provided below:

Result at 5th an 10th iteration

As we can see from the results, the normal-based ICP algorithm yields a similar alignment compared with the basic approach. However, it is significantly efficient given the fact that it only takes around 10 iterations (compared with the one with 150 iterations using basic ICP) to reach the similar result while the time taken for each iteration increases slightly.

Reference

1. Gelfand. N. et al. (2003). Geometrically Stable Sampling for the ICP Algorithm, *Forth International Conference on 3-D Digital Imaging and Modelling*, 260-267. doi: 10.1109/IM.2003.1240258
2. Low, K. (2004). Linear Least-Squares Optimization for Point-to-Plane ICP Surface Registration. Retrieved from <https://www.comp.nus.edu.sg/~lowkl/publications/lowk_point-to-plane_icp_techrep.pdf>
3. Planck Institut Informatik. (n.d.). Pairwise, Rigid Registration: The ICP Algorithm and Its Variants. Max Retrieved from <http://resources.mpi-inf.mpg.de/deformableShapeMatching/EG2011_Tutorial/slides/2.1%20Rigid%20ICP.pdf>