Iterative Closest Point

Fabian van Stijn - 11906448, Venkat Mohit Sornapudi - 13552767

April 25, 2022

1 Introduction: Iterative Closest Point

In this assignment we implement the Iterative closest point algorithm (ICP) and try to make it more efficient using KDTree and a Z-buffer. The ICP algorithm tries to find the best transformation and rotation for 2 3D point clouds that contain points in a different orientation. The point clouds could have some points that represent the same points and some that are not. The points that correspond to eachother are not known. In this report we will also use ICP to merge multiple point clouds into one 3D model.

1.1 ICP Implementation

For the assignment we needed to implement the ICP algorithm to transform 2 different 3D point clouds. One point cloud contained a wave and the other contained the points of a bunny. The source and target point clouds of the wave had the same dimensions. Therefor we can assume these were exactly the same points but in a different orientation. The source and target point clouds of the bunny have different dimensions. This could mean that they were taken from different angles and contain some overlapping points but some new points. As you can see in figure 1 the ICP algorithm correctly transforms and rotates the 3D waves so that they overlap almost perfectly.

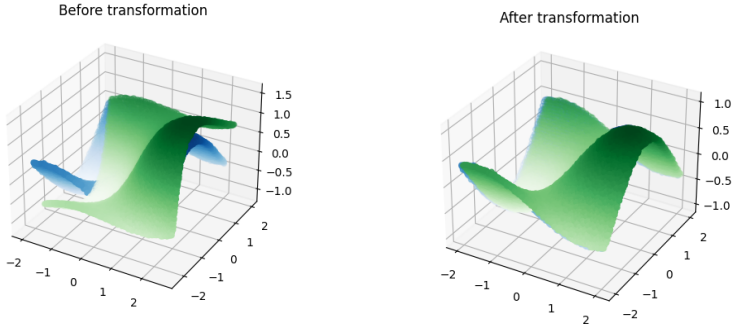

Figure 1: ICP Transformation of 2 3D point clouds of a curve

In figure 2 you can see that ICP brings the two pointclouds together but you can still see blue edges on the left. This in contrary to the wave point clouds where they completely overlap. We think this blue part is still visible because the blue point cloud contains points that were not in the green point cloud. This could be because the blue point cloud was made from a different angle than the green point cloud.

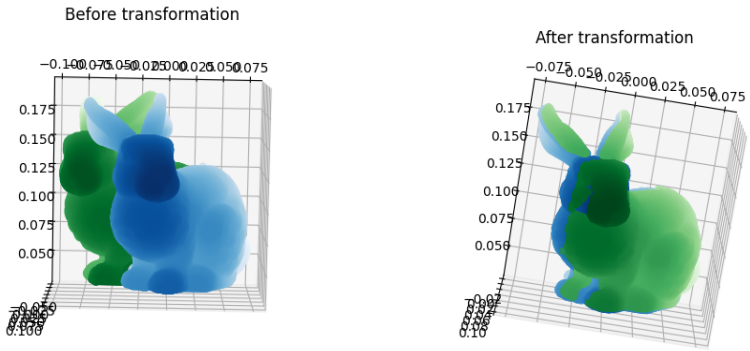

Figure 2: ICP Transformation of 2 3D point clouds of a bunny

2 Improving speed and Quality

ICP works well for finding the correct transformation and rotation to overlap the two point clouds in the most efficient way. But it is not that efficient at what it's doing. In this section we are going to look at ways to speed up the ICP algorithm using different forms of sampling, a kd-tree and a z-buffer.

2.1 Sampling

One way to make the ICP algorithm more efficient is by sampling points from the point cloud so the ICP algorithm doesn't have to perform calculations on every single point in the point cloud but only on a part of it, which when sampling correctly can be enough. For this assignment we will use uniform, random, multi-resolution and point-density sampling.

With uniform sub-sampling we draw samples from the point cloud with a uniform probability. Random sub-sampling draws samples at random from the point cloud.

Multi-resolution sub-sampling downsamples the data in the first few iterations and progressively increases the resolution of the data in the following iterations. In this way also the amount of iterations will be reduced because a lower resolution matching imply more important rotations and translations [JH02].

And finally informative points using point-density sub sampling. This kind of sampling looks at the densities of points in the point cloud as you can see in figure 3.

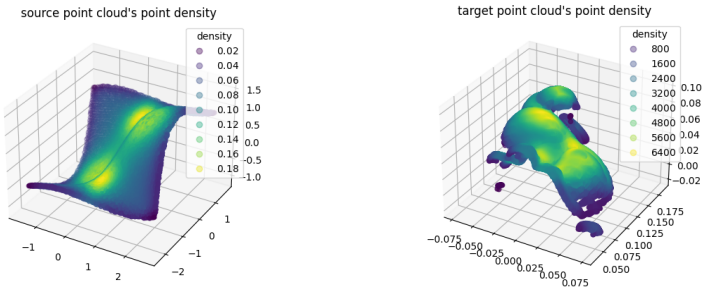

Figure 3: Point Densities of both source point clouds

As you can see clearly see in table 1 uniform sampling is already a lot faster and cuts the time in half. This while only slightly increasing the Root Mean Square error (RMS). Random sampling is the fastest sampling method but increases the RMS error more than the other sampling methods. Multi-resolution is fast but also increases the RMS error significantly. Point-density significantly slows down the ICP algorithm and is therefore not that interesting.

Time / Sampling	all	uniform	random	multi-resolution	point-density
run 1	0.3789	0.1885	0.0838	0.1336	1.4368
run 2	0.3798	0.1894	0.0759	0.1296	1.4751
run 3	0.3813	0.1894	0.0747	0.1296	1.5039
average	0.38	0.1891	0.0751	0.1309	1.4719

Table 1: Run time comparison for different sampling methods

Error / Sampling	all	uniform	random	multi-resolution	point-density
	0.036	0.035	0.136	0.084	0.040

Table 2: Root Mean Square error for different sampling methods

2.2 kd-tree

The ICP algorithm finds the closest points for the source and target 3D point clouds using nearest neighbours. To find these nearest points brute force is used, meaning that for each point in the source 3D point cloud the entire target set has to be searched through. This is not efficient at all. This is where KD-tree comes in. KD-tree splits the point cloud at each axes at the median and creates a binary tree from which the nearest neighbours can be easily found. Creating the KD-tree has a time complexity of $O(n * logn)$ and searching it has a time complexity of $O(m * logn)$. Therefor in theory it should be a lot faster than the brute force method.

Time / Sampling	all	uniform	random	multi-resolution	point-density
Default	7.658	2.811	0.634	2.086	50.182
KDTree	13.392	5.316	1.004	3.744	55.410

Table 3: Run time for all samplings with Default and KDtree

Looking at table 3 you can see that KD-tree didn't speed up our ICP algorithm but made it slower! We think this happened because the KD-tree was implemented within the iteration loop. Therefor constructing the KD tree every single time. We at first implemented it this way because the samples changed per iteration and therefore it seemed like a good thing to do. But it turned out otherwise. We did not have the time to reimplement this.

2.3 z-buffer

Mono-z-buffer has been effectively implemented. It has produced equally good results as kd-tree and the default nearest matching technique for both wave and bunny data. But, it took the longest amount of time to run.

Note that: In our implementation, we projected the point clouds on xy plane assuming that it gives the least enclosed area of the projection of their union. But, in the actual implementation this projection plane changes for every rotation.

Downsides of using a mono-z-buffer: Mono-z-buffer doesn't work well in the cases when the overlap between source and target surfaces cannot be approximated as planar.

Possible improvements: By using of multi-z-buffer.

Implementation ideas for change in the z-buffer referential: Every plane can be described in 4 parameters: 3 for the direction of normal (l,m,n) and 1 for the distance from the origin (d). We can fix centroid of the union of the point clouds as origin and d as a finite large value. Now, we should project the union of the point clouds over the plane. To find the best l,m,n values, we can take some initial random discrete combinations of l,m,n. Observing the least areas made by the projections of union of the point clouds at the different directions of normal, we can select best ones and make them as the limits of the next random combinations. Continuing this process for few iterations we can end up at approximately optimal combination of l,m,n.

3 Global registration

We were given 100 3D point clouds of a student made by a Xbox Kinect sensor. The Kinect sensor was rotated around the student to obtain the 3D points from multiple angles. Using the ICP algorithm between frames we stitched the 100 different point clouds together to form a single 3D model. To stitch the point clouds together we used two different methods, which we will address in the next two sections.

3.1 Estimate pose every N frames and merge at the end

In this first method we estimated poses between every N frames. N can be an arbitrary amount of frames. If we select $N = 10$ for example we take the first 3D point cloud (frame) and the one that is 10 frames further. This way we will use less frames and therefore the run time will be shorter. But the model will be less detailed and can contain more mistakes. If we take $N = 1$ we stitch every single frame in the dataset together, taking the longest possible time but granting the best 3D model.

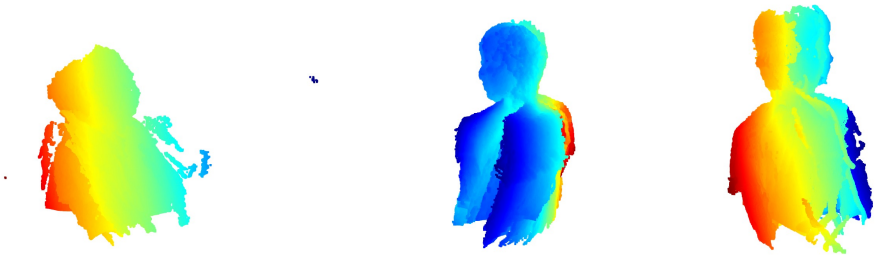

Figure 4: 3D models merging at the end for N=1, N=4 and N=10 respectively

Looking at figure 4 you can see the 3D models of the student that was constructed using the ICP algorithm with multi-resolution sub-sampling. To save time we used low time budget settings. We find that actually the model for $N = 1$ looks the worst and $N = 10$ the best. This is different to what we expected. We think this could be explained by a cumulative error which is worse if you perform the calculation 100 times than 10 times. Therefore the $N = 10$ 3D model looks the best.

3.2 Estimate pose and merge every N frames

Now we will repeat what we did in the previous section but instead of merging at the end we will merge after every pose estimation. We expect that this will introduce even more errors into constructing the model. Because if one frame matches incorrectly to it's target then the next one will also be misaligned and the model will contain many incorrectly added 3D point clouds.

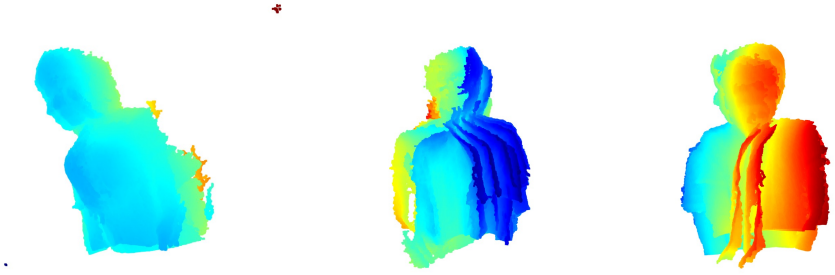

Figure 5: 3D models merging after every pose estimation for N=1, N=4 and N=10 respectively

As expected the 3D models seem to be worse than before. It seems that the point clouds have many different rotations, with the exception for $N = 1$. We think this can be explained by the frames following eachother up in the correct order. Therefor they match closely to one another and they are

stitched together correctly. In the case of $N = 4$ or $N = 10$ the frames are further apart and therefor the 3D point clouds don't match as easy and therefor the result is worse as we expected.

4 Additional Questions

4.1 What are the drawbacks of the ICP algorithm?

The ICP algorithm has multiple drawbacks. Underneath a list of these drawbacks.

1. ICP has quadratic time complexity $O(N^2)$
2. Preprocessing is needed to clean the point clouds from noisy data.
3. ICP is very demanding computationally because point clouds can contain a lot of points and therefor need a lot of calculations.
4. A first guess is needed for ICP. This could be a very wrong guess but starting from a transformation matrix containing only zeros slows ICP down in finding the transformation.
5. To speed up the ICP algorithm Kd-tree is often used. But because a Kd-tree is a binary structure with a node per point this is very memory hungry. Therefor this method of speeding up ICP is only usable on small point clouds for a normal computer.
6. When using a mono z-buffer to speed up the ICP algorithm curves are not approximated that well. It would be better to use a multi-z-buffer technique.

4.2 How do you think the ICP algorithm can be improved, besides the techniques mentioned in [7], in terms of efficiency and accuracy?

Outliers could be removed beforehand to improve on the accuracy. Outliers can cause the transformation to be off and not reach the best possible transformation.

In order to improve on the efficiency we could try to make a 2D projection of the 3D point cloud so we can find key points in 2D. If we can find the same key points in 2D we could match them on both the source point cloud and the target cloud in 3D and then the ICP algorithm could have multiple points for which it knows that they have a high probability to belong together. And then it could find the correct transformation and rotation a lot faster. A method of finding keypoints for ICP has also been done with Multiplex Dynamic Graph Attention Networks [SCH+21].

An improvement in efficiency might be able to be achieved with another structuring algorithm for nearest neighbour search. Instead of using KD Tree we could use a Ball Tree. This is one of the three options given by Scikit learn's K.NeighborsClassifier algorithm. A KD Tree on the contrary partitions the data along Cartesian axes. The ball-tree algorithm is a metric tree. Metric trees organize and structure data points taking the metric space into account. It divides the data points into clusters of nesting spheres. Using this method of structuring is more efficient on highly structured data, even in very high dimensions.[16N] The actual performance is highly dependent on the structure of the training data, in some cases it might still be more efficient to use the KD-tree algorithm. It would be good to try both algorithms to find out whether one outperforms the other.

5 Conclusion

Iterative Closest Point is a good way to transform and rotate 3D point clouds to their overlapping orientations. In a relatively short time the algorithm can find the ideal transformation using a straight forward approach. There are methods to speed up the ICP algorithm, like using sub-sampling or kd-tree. Sub-sampling can make the ICP algorithm run up to 4 times faster for the fastest sampling method while only suffering minor degradation in RMS error. KDtree is a good solution to make ICP scalable if you can implement it correctly. Using point-density to sub-sample points from the point cloud was not viable for our models because it took very long. But it might be interesting in certain models with a lot of noise or extreme outliers.

References

[16N] 1.6. nearest neighbors — scikit-learn 1.0.2 documentation. https://scikit-learn.org/stable/modules/neighbors.html. (Accessed on 04/23/2022).

[JH02] Timothee Jost and Heinz Hügli. A multi-resolution scheme icp algorithm for fast shape registration. pages 540– 543, 02 2002.

[SCH+21] Chenghao Shi, Xieyuanli Chen, Kaihong Huang, Junhao Xiao, Huimin Lu, and Cyrill Stachniss. Keypoint matching for point cloud registration using multiplex dynamic graph attention networks. *IEEE Robotics and Automation Letters*, 6(4):8221–8228, 2021.