Acquisition and Processing of 3D Geometry
Assignment 1 – ICP registration
Report

Benjamin Barral

0. Questions attempted

I attempted all 6 of the required tasks :
Task 1 : I implemented point-to-plane ICP for $M_2 \rightarrow M_1$ registration.
I implemented both a single ICP iteration as well as the whole registration algorithm.
I derived the new formula for the weighted version of ICP; I provided a demonstration with this report.
I did not implement a way to visualize the non overlapping regions, but I found that using different colors and a reasonable point size gave quite intuitive results.
Task 2 : I ran my ICP algorithm until convergence for increasing initial angles, and reported the results on the number of iterations required for each configuration.
Task 3 : I implemented the noise addition (using Gaussian noise with controllable variance). Same as for task 2 and 3, I reported the results on the number of iterations before convergence for different values of standard deviation.
Task 4 : I implemented the subsampling step for ICP (with a controllable sampling rate). Same as for task 2, 3 and 4, I reported the result on the number of iterations before convergence for different values of the sampling rate.
Task 5 : I implemented a system to perform global registration over the 10 point clouds provided by the Stanford Bunny data set.
I use a strategic sequence of point cloud-to-point cloud registrations in order to register the whole data set. Therefore this task relies on some hard coded parameters (for the pairs of point clouds) and on user pre-alignment at each one-to-one step.
Task 6 : I implemented point-to-plane ICP.
For this, I implemented "Plane PCA" using K nearest neighbors for normal estimation.
I did not implement the new shading based on normals.
7. : I implemented this coursework in C++, using the LibIGL library.
I created three different classes for ICP algorithm, point cloud manipulation and user input; and a *main.cpp* file.

1. Task 1 : Basic ICP

I implemented the ICP described in the lecture.
For the point selection (step 2), I use a minimum threshold on the distance-to-nearest neighbor.

I use the ANN library for the nearest neighbor computation. I precompute the KD-tree of the registered-to point cloud, which remains static during the whole process.
I use Eigen for the SVD decomposition.
For the whole registration algorithm, I use a threshold on the averaged sum of squared distances to nearest neighbors over the selected points. To prevent from non converging configurations, I set a maximum value of iterations.
The following thresholds were set with fine-tuning :
- $D_{max} = 0.0028$ for point selection.
- $D_{conv} = 0.0007$ for convergence : average nearest neighbor distance.
- $N_{max} = 0.0028$

These are the default parameters, but for the converging distance
In the case of 'bun045'→'bun000' my algorithm converges for very rough initial pre-alignments (cf. figure 1).

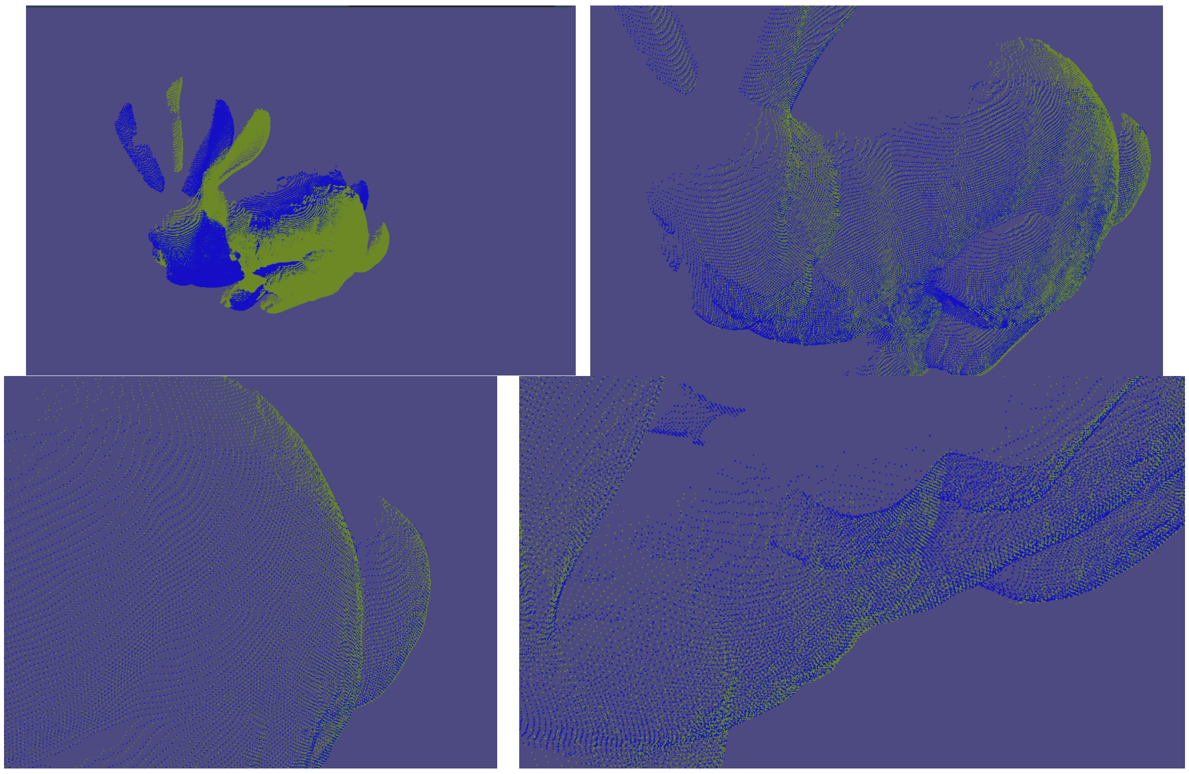
Figure 1 : An example of the basic ICP
Top left : the initial pre-alignment
Top right and bottom : the alignment
67 steps; final average error : 0.00049.

See the file 'Weighted ICP derivation' for the proof required.

2. Task 2 : Rotation

For this task, since the too point clouds are identical, the algorithm should in theory converge perfectly. Therefore I specifically decreased the value of the converging threshold, and increased the maximum number of iterations :
- $D_{conv} = 0.0002$.
- $N_{max} = 150$.

I manually rotate the second point cloud from the viewer with a slider (or a value input).
I increased values of α by 5 degrees for each experiment.
The value for which no convergence under 150 iterations was possible was 25 : it took 249 iterations.
I only studied positive values (empirically the behavior was symmetrical).
An observation I made is that, in this specific configuration, the error slowly decreases at first, and then converges drastically once getting a certain threshold (around 0.001).
I reported the results of the number of iterations before convergence for different values of α the rotation angle on figure 2.

Rotation angle (degrees)	Number of iterations
5	31
10	45
15	79
20	141
25	249

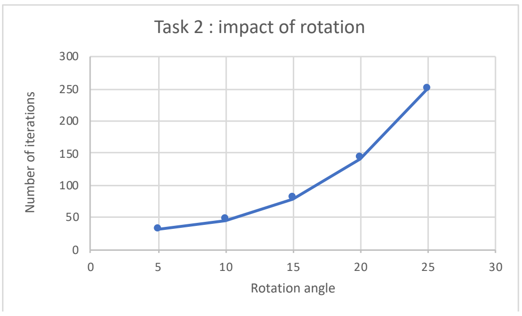

Figure 2 : Result of experimentation on angles and number of iterations before convergence

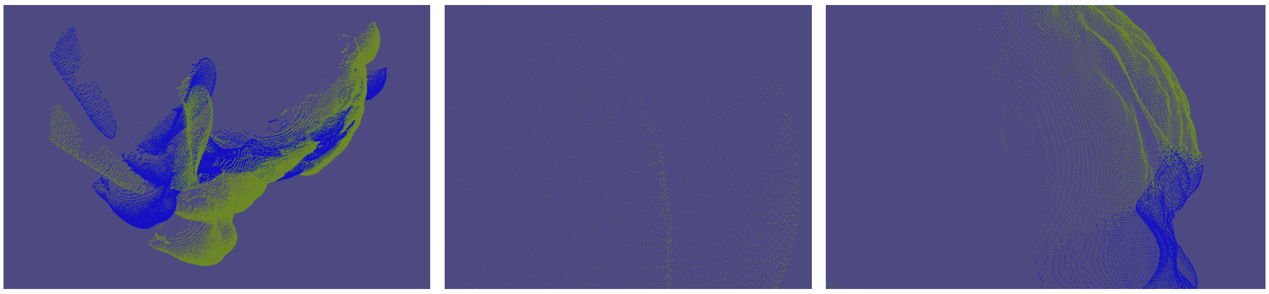
Figure 3 : Visual results of task 2
Left : the 25 degree rotation
Middle and right : Results after alignment : the absence of mixed blue and yellow parts indicates 'perfect alignment'
249 iterations; final error : 10^{-15}

3. Task 3 : Noise

For this task, I add noise to the data by simulating a zero-mean Gaussian distribution for each coordinate of each point, on the point cloud which is being registered to.
To set the variance of the Gaussian variable I calculate the dimensions of the bounding box in all the axes, and I set the per-axis variance as a ratio of the dimension.

I use the std::normal_distrubition method.

For my experiments, I pre-aligned the 'bunny045' on 'bunny000' (see figure 4 for the pre-alignment), and I modify the ratio manually.
Here I set :
- $D_{conv} = 0.0008$.
- $N_{max} = 120$.

Figure 5 shows examples of the data after noise addition.
Figure 6 shows results after alignment of noisy data.
Figure 7 shows results of number of iterations needed before convergence. The limiting value for which no convergence happens under 120 iterations is $ratio = 0.01$ (i.e. the variance in each axis is equal to 1% of the point cloud's bounding box dimensions).

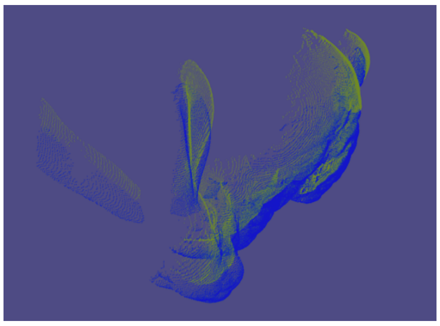
Figure 4 : Pre-alignment for the experiment

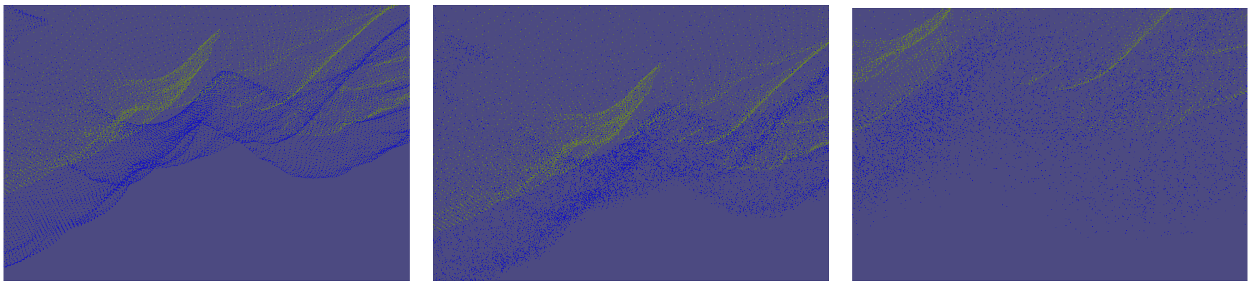
Figure 5 : Initial configuration : zoomed
Left : no noise

Middle : Noise ratio : 0.004
Right : Noise ratio : 0.01

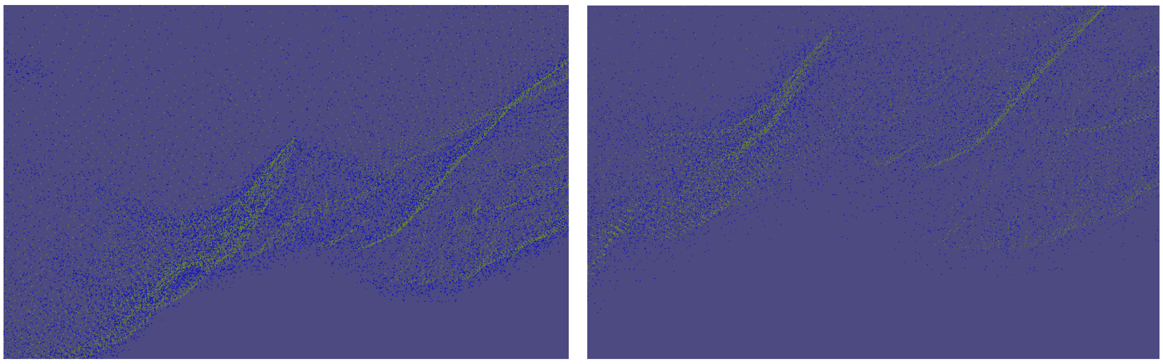

Figure 6 : Result after alignment of noisy data
Left : Noise ratio : 0.001
Right : Noise ratio : 0.004
Distance convergence threshold for the experiment : 0.008

Variance ratio	Number of iterations
0	12
0,003	14
0,004	15
0,005	17
0,006	19
0,007	23
0,008	30
0,009	44
0,01	>120

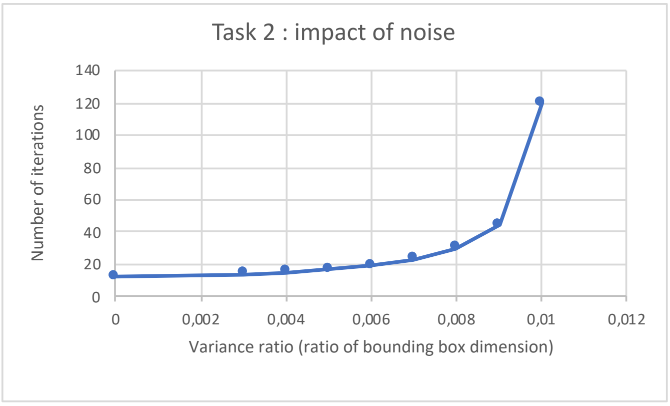

Figure 7 : Results of number of iterations before convergence for different values of noise

4. Task 4 : Subsampling

I implemented uniform subsampling.
My variable is the subsampling rate ε : : i.e. $n_{ICP} = \varepsilon * n$ where n_{ICP} is the number of points used for the algorithm, and n the total number of points.

I chose the same configuration as in the previous task (c.f. figure 4 for the pre-alignment).
I set :
- $D_{conv} = 0.0005$.
- $N_{max} = 120$.

Figure 8 shows the results of this experiment.
I found that a reasonable sampling rate is 0.1, as it provides similar results in registration accuracy and number of iterations.
I found that for a rate lesser than 0.002 (i.e. selecting 0.2% of the points) my algorithm stopped converging.

5. Task 5 : Global registration

For this task, I design a strategic 'sequence' of point-cloud-to-point-cloud ICP registrations in order to register the 10 point clouds into a common reference position (defined by bunny000).

For instance : I first register bunny045 to bunny000, then bunny090 to bunny045, bunny315 to bunny000, bunny270 to bunny315 etc… (using the updated point clouds at each step).
The strategy to define the sequence consists in finding pairs which have the most overlap.
Therefore I had to experiment with all of the point clouds before designing that sequence.
Look at the comments in the main code for the description of that specific sequence I designed.

Note that, as much as in the case of basic ICP, at each registration step, manual pre-alignment is required.
Also, note that : some point clouds don't have sufficient overlap with other ones for the 'convergence' condition being met : i.e. there are too instances when D_{conv} threshold is not reached; but the resulting registration looks reasonable overall.
In terms of user experience, I implemented the viewer such that :
- it shows the set of untreated point clouds with random colors, on the side.
- it shows the too currently processed point clouds, the registered one in white, the registering white in black.
- after each step is validated, the previously registered point clouds are shown in white.
(c.f. figure 8).

The advantage of this method is that it builds upon the pre-implemented IPC algorithm.
The drawback is that it 'propagates' the error : if a point cloud is inaccurately registered, then all the point clouds which will have that point cloud as its reference pair will be hit by that inaccuracy.
Another limitation is that it requires pre-existing knowledge on the point clouds (in that case I hardcoded the pair indices of the point clouds).

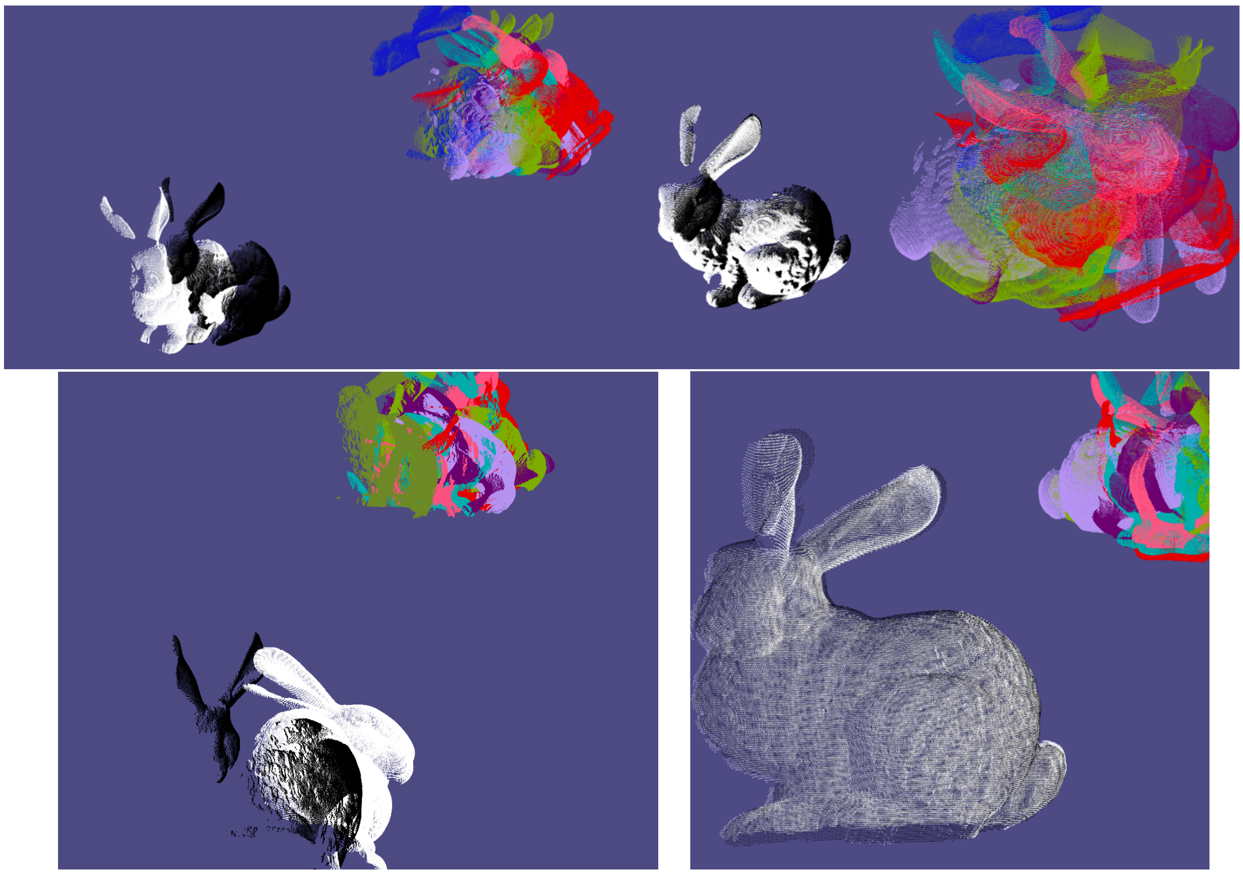
Figure 8 : The global registration, consecutive steps
Bottom right shows the registration step which leads to the least satisfying results

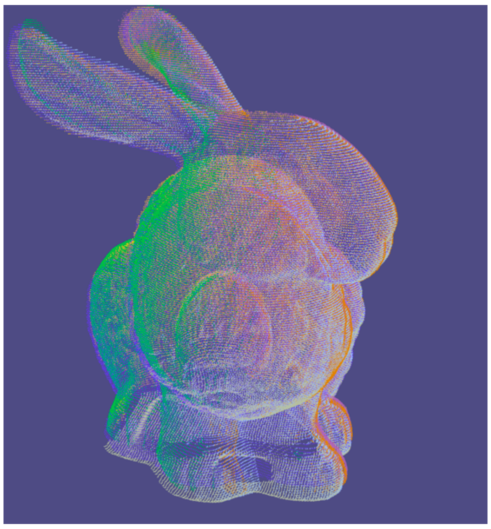
Figure 9 : A result of my global registration method, with 6 point clouds

6. Task 6 : Point-to-plane

For this task, I implemented Kok-Lim Low's Point-to-plane ICP **[1]**, and its linear approximation (small angles), using equations (8), (9) and (13).
To retrieve the normal information, I implemented the 'Plane PCA' algorithm in **[2]**, which fits a plane through Principal Component Analysis : by computing the eigen decomposition of the covariance matrix of the K nearest neighbors, and retrieving the normal as the eigen vector associated with the lowest eigen value.

I found that the point-to-plane method leads to more accurate registration results : both visually and quantitatively with the error values.
Note that I adapted my error threshold by multiplying it by 0.5 in order to take into account the impact of the dot product in the error formula.

Look at the code in `ICPHandler::ComputeRigidTransformation()` and `ICPHandler::ComputeLinearApproximationMatrix()` in 'ICPHandler.cpp' for more information about my implementation.

7. Code

The code for the ICP algorithm is in the class `ICPHandler`.
The class `PointCloudData` deals with transforming point clouds, storing their (initial and updated) vertices positions and their colors, computing noise etc…
The class `ICPViewerManager` deals with user input and calls the too classes above in order to update the viewer and perform the different variants of ICP.
Change the value of the global variable `task_Id` to either `TASK_1` `TASK_2` or `TASK_5`. All the other parameters and variables are controllable from the viewer.

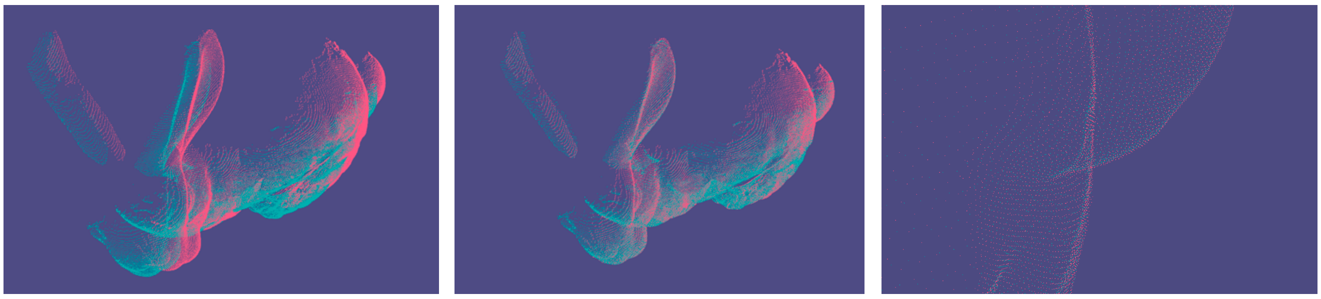

Figure 10 : Results of Point-to-plane ICP
Left : initial very rough configuration
Middle and right : Result
Convergence after 7 steps, error threshold : 0.0004

References

[1] LOW, K.-L. 2004. Linear least-squares optimization for point-to- plane icp surface registration. Tech. rep., Chapel Hill, University of North Carolina.

[2] K. Klasing, D. Althoff, D. Wollherr, et al., Comparison of Surface Normal Estimation Methods for Range Sensing Applications, Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA), 2009.