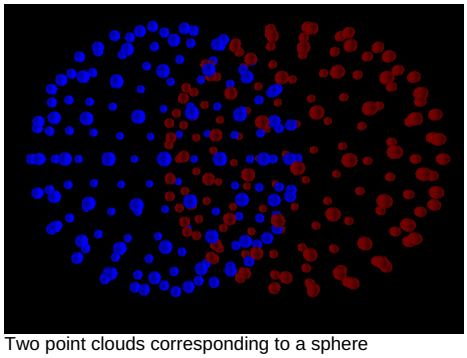
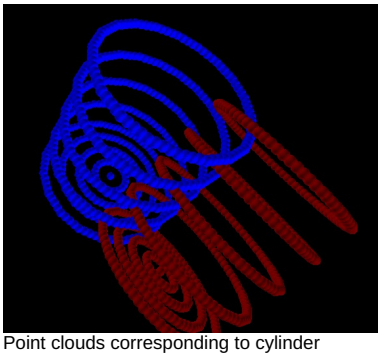
TD 5 - Iterative closest point

The goal of this TD session is to implement the iterative closest point algorithm (using both 3x3 matrices)


Two point clouds corresponding to a sphere


Point clouds corresponding to cylinder

Votre opinion sur le cours:

Votre opinion sur le TD:

Submit

1. Getting started (a Processing frame for rendering 3d point clouds)

Files to download today

Here are some files to download (and extract to the main folder of your Eclipse project):

- OFF.zip: examples of points clouds.

Libraries to add (project /LIB):

- Jcg.jar: library for the manipulation of 3d geometry (important: download the new version of the JCG library from this page)
- TC.jar: input/output from files.
- Jama.jar: linear algebra library.

For the implementation of ICP:

- src.zip: classes defining main methods for rendering point clouds and implementing ICP

TD5 documentation: instructions

Important: here is an interesting paper illustrating how to implement ICP (with both 3x3 matrices and quaternions).

1. Iterative closest point (with rotation matrices)

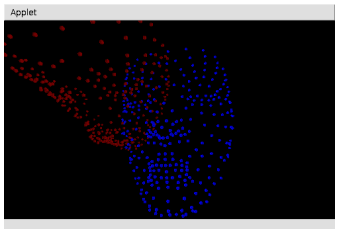
ICP with Matrices and SVD

Given two 3d point clouds P1 and P2 (class PointSet), we want to compute the rigid transformation that maps P2 to P1, by implementing the ***Iterative closest point method with rotation matrices***.

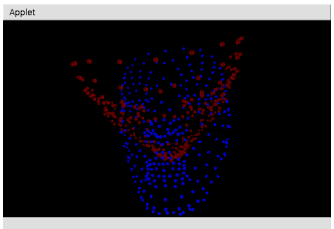
- complete the class IterativeClosestPoint (which extends class RigidTransformation_3).

Suggestion: in order to implement ICP, we suggest to you to follow the explanation given in this paper (section 3.1).

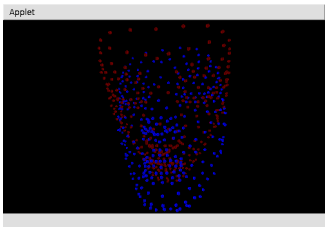
Testing your code: in order to test your implementation of ICP, you can run the class TestICP.



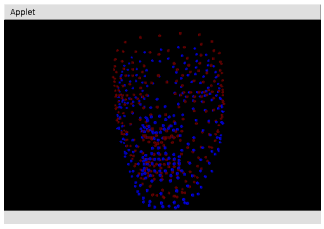
Initial point clouds



after 1 iteration



after 3 iterations



after 5 iterations

1.1 ICP: compute the rigid transformation for corresponded point clouds

- complete methods getRotation(...) and getTranslation(...), for the case of two point clouds P and Q whose points are matching (so we can assume point q[i]=p[i]).

1.2 ICP for arbitrary point clouds (no correspondence given)

Now we want to solve the problem above (computing a rigid transformation), for two point clouds whose points are not matched.

- modify methods `getRotation(...)` and `getTranslation(...)`, for dealing with arbitrary point clouds P and Q (now we cannot assume point q[i]=p[i]).

1.3 Check convergence of ICP

Now we want to check whether the ICP iterations converge.

- compute and print (at each iteration) the sum of distances between closest neighbors and check that such distances do decrease.

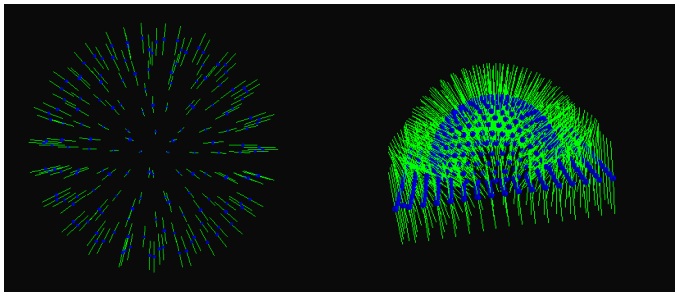
2. Normals estimation

2.1 Computing normals of point clouds

Given a 3d point cloud P1 (class `PointSet`), we want to compute an estimation of normals as explained in the lecture (using co-variance matrix).

- complete the class `PointSet` (complete the method `getKNearestNeighbors`).
- complete the class `NormalEstimator` (which contains the method `computeNormals`).

Use the class `PointCloudViewer` to test your code (press 'n' to compute normals)



3. Outliers computation

3.1 Computing outliers of point clouds

Given a 3d point cloud P1 (class `PointSet`), we want to compute the outliers as explained in the lecture (again using co-variance matrix).

- complete the class `OutliersComputation` (which contains the method `computeOutliers`).

Use the class `PointCloudViewer` to test your code

4 Iterative closest point, with quaternions

4.1 ICP with quaternions

Given two 3d point clouds P1 and P2 (class `PointSet`), we want to compute the rigid transformation that maps P2 to P1, by implementing the ***Iterative closest point method with quaternions***.
You can start by writing a

- class `QuaternionsICP`, which extends class `RigidTransformation_3`, allowing to compute rotations using quaternions.

Then modify the method `computeICP()` of the class `public void computeICP()` in order to test the new method (based on quaternions) for computing rotations and translations.

Suggestion: you can follow the same steps as above, taking a look to the slides of today Lecture and this paper (section 3.3)

Point-to-Plane ICP (bonus)

Improve ICP (bonus)

Given two 3d point clouds P1 and P2 (class `PointSet`), implement the point-to-plane variant of ICP (as described in the lecture).

- complete the class `PointToPlaneICP` and compare to the standard ICP method (compare convergence rates).

Modify the class `TestICP` to test your code