Assignment 1

January 22, 2016

INSTRUCTIONS

This is the first coursework for course COMPGV18/COMPM080 on Acquisition and Processing of 3D Geometry. The subject of the coursework is implementing a basic version of point-to-point iterative closest point (ICP). The coursework is worth a maximum of 100 points.

You are encouraged to use C/C++ for this assignment. For efficient implementation you will need a nearest neighbor data structure. You may use a quad tree, or a BSP tree, or use a library like ANN (`http://www.cs.umd.edu/~mount/ANN/`) or FLANN (`http://www.cs.ubc.ca/research/flann/`)

Hand in a short report on your work (to be submitted via Moodle). The short report should start with a brief list of what you have attempted and to what extent you have achieved each item. Next, write a short description of the methods you used for each part together with any conclusions you have drawn from your experiments. It is encouraged to augment your experiments (tasks 3,4 and 5) with plots or tables.

You will be required to present your work in a one to one session (date to be decided) demonstrating what you have implemented. Please upload a pdf report and a zip of your source code in your submission.
Please **note**, that the coursework is individual work, we are expecting to see your own solution, including the code you hand in. For libraries you used, please reference them properly in your report. For tasks 2-7, please write your own code, do **not** call a library's function, that does ICP for you.

LATE POLICY The coursework is due February 12th (Friday midnight). We will use the following late policy:
Submissions after the deadline but within 24 hours of it will be marked down to 90% of the number of points achieved.
Submissions later than 24 hours, but within 48 hours from the deadline will be marked down to 80% of the number of points achieved.
Submissions later than 48 hours will receive 0%.

1 Core Section

In this assignment we will explore different aspects of the ICP algorithm. Please refer to the lecture notes and also the original ICP papers (see lecture notes for relevant references). Start by downloading the bunny models from `http://graphics.stanford.edu/data/3Dscanrep/`. Select two models as M_1 and M_2 for the subsequent tests. Try to select models with sufficient overlap (you will need the viewer that you develop in the first task to roughly judge this).
For all the tasks you are expected to write your own code, although looking up the original paper(s) is encouraged.

1. The source data is in PLY format. The format is relatively simple and you can write your own parser, or download a library (e.g., `http://paulbourke.net/dataformats/ply/`). Your first task is to make sure you have access to a simple Mesh Viewer where you can visualize and inspect your intermediate results. You should be able to load meshes, and then interactively rotate or translate them. It should be possible to load two (or more) meshes at the same time. Suppose, you load mesh M_1 along with transformation T_1, and similarly M_2 along with transformation T_2, then your viewer should show $T_1(M_1)$ and $T_2(M_2)$ in a common coordinate system. By default, you may assume T_1 to be the trivial transform with the rotation component as the identity and translation being zero. You can use `http://openmesh.org/svnrepo/OpenMesh/trunk/src/OpenMesh/Apps/QtViewer/` or similar to start from. You may use libraries as TriMesh, OpenMesh etc.. You will need this viewer throughout the course. (5 points)
2. In this part, you will align model M_2 to model M_1 using only rigid transforms. Assume that the models are nearly aligned to start with and have a large (but unknown) overlap. Implement point-to-point ICP to align $M_2 \rightarrow M_1$ and also mark out the non-overlapping regions (after alignment). You will need to have access to an eigen solver. You can try Eigen, GSL or other libraries. You can use ANN to speed up nearest neighbor computations. (35 points)
3. Assume that the model M_1 is at the origin (i.e., centroid of M_1 is at the origin). Now assume $M_3 = R(M_1)$, i.e., a rotated version of matrix M_1 about the origin. Progressively perturb the initial rotation of R and evaluate the convergence behavior of ICP trying to align $M_3 \rightarrow M_1$. R can be rotation about about any of the coordinate axes for example (or for general rotation you can lookup quaternion based rotation). (10 points)
4. Perturb model M_2 to simulate a noisy model say M_2'. Evaluate how well ICP performs as you continue to add more noise. As noise, add zero-mean Gaussian noise – you can simply perturb each vertex of the mesh M_2 under this model. (10 points)
5. Instead of directly aligning M_2 to M_1, speed up your computation by commutating the aligning transform using subsampled versions of M_1 and/or M_2 as appropriate. Report accuracy with increasing subsampling rates. (10 points)
6. Now given multiple scans $M_1, M_2, \ldots M_k$ align all of them to a common global coordinate frame. Make sure to think about the best strategy to do this. What strategy gives you best alignment. Is this also the fastest to compute? (15 points)
7. For each point $p \in M_1$ estimate its local normal as follows. (i) Find its k-nearest neighbors, say $q_1, \ldots, q_k$ (take k to be 5-10). Then compute the covariance matrix as $C = \sum_{i=1:k}(q_i - p)(q_i - p)^T$. The normal at p can be approximated as the eigenvector with the lowest eigenvalue when decomposing C. Now, you can compute normals at each points in M_1 and M_2. Update your viewer to shade the models based on these normals. Now, use the normal information to improve the ICP performance. Are you improving speed or accuracy? Is it correct to compare normals directly? (15 points)