Assignment 1

January 23, 2024

INSTRUCTIONS

This is the first coursework for course COMP0119 on Acquisition and Processing of 3D Geometry. The subject of the coursework is implementing iterative closest point (ICP) alignment. The coursework is worth *100 points*.

You are encouraged to use Python for this assignment, but can alternatively use C/C++ or Matlab. 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 such as Scikit-kdtree[1] or FLANN[2].

Please **note**, that the coursework is *individual work*, we are expecting to see your solution (please do not attempt to look up answers from external sources or friends), including the code you hand in. For libraries you used, please reference them properly in your report (no need to include them in the zip). Please write your code, do **not** call a library's function, that does ICP for you.

LATE POLICY See Moodle for submission details. For late policy, please refer to departmental guidelines. If you are suspected of plagiarism, you might be requested to present your work in a one-to-one session (date to be decided) demonstrating what you have implemented.

SUBMISSION GUIDELINES

- Please submit a report (PDF) on your work (to be submitted via Moodle). The report should start with a brief list of the questions you have attempted and to what extent you have achieved each goal.
- Next, write a short description of the methods you used for each part together with any conclusions you have drawn from your experiments.
- Present plots or tables where appropriate (tasks 2,3 and 4 in the Core Section).
- Where appropriate visualize meshes from at least 2 different directions and provide screenshots.
- Color meshes differently for better visibility.

In summary, you are expected to submit (a) a full report (pdf) and (b) code (as a zipped file along with documentation and instructions for running the code). Please do *not* upload build directories, executables or 3D models you used as input.

[1] `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KDTree.html`
[2] `http://www.cs.ubc.ca/research/flann/`, `https://github.com/jlblancoc/nanoflann`

1 HALFEDGE MESH DATA STRUCTURE (10 POINTS)

In the tutorial, you have learnt/will learn how to construct a shape using the halfedge mesh data structure. In this section you will investigate the halfedge mesh data structure further.

1. Using the simple halfedge mesh library from the tutorial, write a function to generate a halfedge mesh for a cube. (The facets should be quads - it's important that you don't subdivide them into triangles.)

For any graph G, we can construct the **dual graph** G' by replacing every face of G by a vertex, and connecting two vertices only when the two original faces shared an edge. By doing this, it's also true that every vertex of G will turn into a face of G'. In addition, the 'dual' operation is self-inverse - i.e. the dual of the dual of G is the same as G.
When the graph represents a polyhedron (e.g. a cube) then we can create a 'dual polyhedron' by replacing every face by a vertex at the centroid of that face. For example, the dual of a cube is an octahedron, the dual of a dodecahedron is an icosahedron, and the dual of a tetrahedron is another tetrahedron. To see illustrations, look at the Wikipedia pages for dual graphs (this link) and dual polyhedra (this link.)

2. Write a function that takes a pointer to a facet as input, and returns the x, y, z co-ordinates of the centroid. (The centroid of a facet is the mean of the co-ordinates of its vertices.) If you use the method 'Facet.get_vertices()' then you should explain how it works.

3. Write a function that takes a halfedge mesh and returns a halfedge mesh for the dual polyhedron. (To get started, how many facets, vertices and halfedges there should be? Then work out how to define the connections between halfedges, vertices and facets.)

   Test your function by finding the dual and the double dual of the cube, and the icosphere 'sphere1.off'. Visualise the results by exporting as an 'obj' file and opening it in Meshlab, and provide screenshots in your report.

If you use pointers correctly, your code will be very efficient. For example, your code will not need to search through any lists. You do not need to use any functions from the library except for those used in the tutorial.
If the dual mesh is inside out (in Meshlab it will look dark) you can use the HalfedgeMesh.flip() method to fix this.

2 CORE SECTION (90 POINTS)

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 Moodle[3]. Select two models as M_1 and M_2 for the subsequent tests. Try to select models with sufficient overlap (you will need a viewer described 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. Geometry processing is usually a computation-heavy task, hence it is important to write efficient code. Try to modularize your code into re-usable objects/files/modules.

1. The source data is in PLY format. The format is relatively simple and you can write your own parser, or use existing code (see the first tutorial[4] or libIgl's readPLY[5]). You may also convert the models to another format (e.g., OBJ, OFF, or xyz) using Meshlab.

   Your first task is to make sure you have access to a simple mesh viewer (e.g., libIgl, or Meshlab) 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 the example framework provided in our tutorial or similar (e.g., Meshlab) to start from. You will need this viewer throughout the course.

[3] original data source :`http://graphics.stanford.edu/data/3Dscanrep/`
[4] `https://github.com/smartgeometry-ucl/COMP0119_20-21`
[5] `https://github.com/libigl/libigl/blob/master/include/igl/readPLY.h`

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 Numpy, Eigen or other libraries. You should use KDTree or similar to speed up nearest neighbor computations. (25 points)

In class, we discussed the solution to the optimization to minimize alignment error of the form $E(\mathbf{R},\mathbf{t}) = \sum_i \|(\mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i)\|^2$ over unknown rotation $\mathbf{R}$ and translation $\mathbf{t}$. Now, assume we have a confidence score for each measurement, *i.e.*, for each point $\mathbf{p}_i$ we have an associated weight $w_i \in [0,1]$ with higher weight denoting higher confidence (or reliability). Refine the derivation done in class to solve the optimization $E(\mathbf{R},\mathbf{t}) = \sum_i w_i \|(\mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i)\|^2$ over unknown rotation $\mathbf{R}$ and translation $\mathbf{t}$. Note that this part only requires a derivation – you are not required to implement this version. (10 points)

2. a) Assume that the model M_1 is at the origin (*i.e.*, centroid of M_1 is at the origin). Now assume $M_2 = \mathbf{R}(M_1)$, *i.e.*, a rotated version of model M_1 about the origin. Progressively perturb the initial rotation of $\mathbf{R}$ and evaluate the convergence behavior of ICP trying to align $M_2 \rightarrow M_1$. $\mathbf{R}$ can be rotation about any of the coordinate axes for example. One way to simulate the effect of increasing misalignment (*i.e.*, initial alignment) is to rotate the object about z-axis over increasing rotation angles (say ± 5, ± 10, ± 15,.. degrees). (5 points)

   b) Perturb model M_2 (without the extra rotation) to simulate a noisy model say M_2'. Evaluate how well ICP performs as you continue to add more noise. As for noise, add zero-mean Gaussian noise – you can simply perturb each vertex of the mesh M_2 under this Gaussian model. Adjust the amount of noise based on the bounding box dimensions of M_2. (5 points)

3. Instead of directly aligning M_2 to M_1, speed up your computation by estimating the aligning transform using subsampled versions of M_1 and/or M_2 as appropriate. Report accuracy with increasing subsampling rates. (10 points)

4. Now given multiple scans $M_1, M_2, \ldots M_5$ align all of them to a common global coordinate frame. Make sure to think about the best strategy to do this. Describe the method you propose and discuss its pros/cons. (10 points)

5. Now assume that you have access to normals at each vertex of the meshes. Update your viewer to shade the models based on these normals. Now, use the normal information to improve the ICP performance. Like discussed in class, you are attempting to solve minimize the following objective function $E(\mathbf{R},\mathbf{t}) = \sum_i \|(\mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i) \cdot \mathbf{n}_i^q\|^2$ instead of minimizing $E(\mathbf{R},\mathbf{t}) = \sum_i \|(\mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i)\|^2$ as implemented in Q1. Note that points $\mathbf{p}_i \in M_1$ and $\mathbf{q}_i \in M_2$ with $\mathbf{n}_i^q$ denoting normal at point $\mathbf{q}_i$ to the mesh M_2. This method is commonly referred to point-to-plane ICP. (25 points)