6 November 2018

General Information:

You have to submit your solution via Moodle. We allow **groups of two students**. Please list all names in the submission comments, upload one solution per group. You have **one week of working time** (note the deadline date in the footer). To get the 0.3 bonus, you have to pass at least three exercises, with the fourth one being either completely correct or borderline accepted. The solutions of the exercises are presented the day after the submission deadline respectively. Feel free to use the Moodle forum or visit us during office hours (every Thursday 10:00 – 11:00) to ask questions!

To inspect your result, you need a 3D file viewer like **MeshLab** (http://www.meshlab.net/). The exercise zip-archive contains a Visual Studio 2017 solution. The required libraries are located in the libs folder and the required data in the data folder.

For this project we use **Eigen** (http://eigen.tuxfamily.org/), **FLANN** (https://www.cs.ubc.ca/research/flann/). The provided libraries are pre-compiled on Windows for Visual Studio 2017, but they can be compiled for other environments following instructions on the provided websites (for Linux instructions are added).

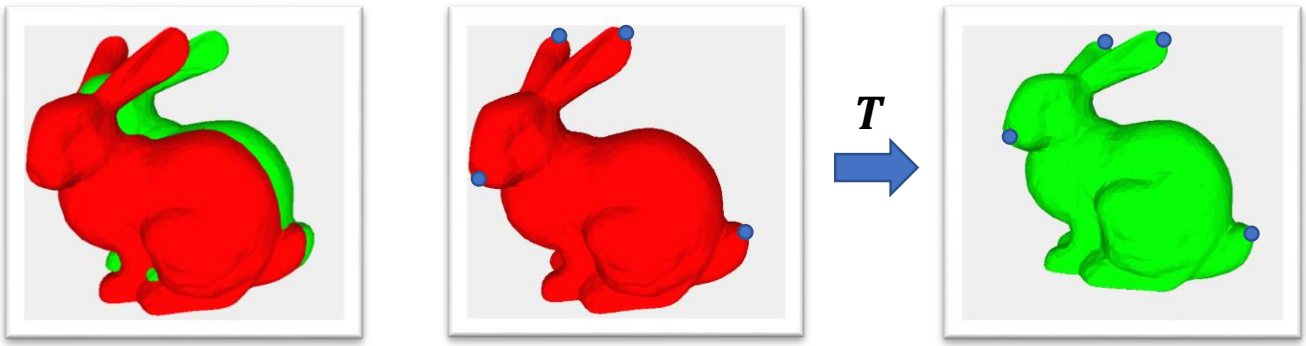
Expected submission files: ProcrustesAligner.h, ICPOptimizer.h, bunny_procrustes.off, bunny_icp.off

Exercise 3 – Registration – Procrustes & ICP

In this exercise we want to align 3D shapes. We will explore two different algorithms and test them on shapes provided either as meshes or with depth maps.

Tasks:

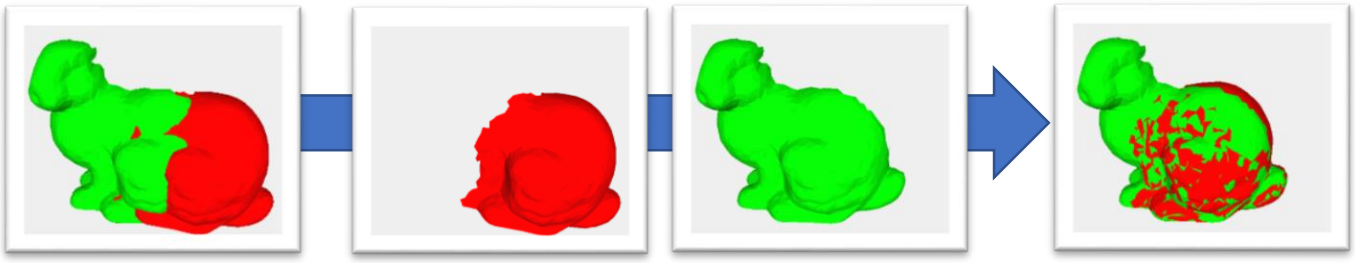
1. Coarse Registration – Procrustes Algorithm



In this task you have to implement the Procrustes Algorithm to align two bunny meshes (red & green). Exploiting the sparse point correspondences shown as blue dots, we are able to estimate the transformation T that transforms the red bunny to match the green bunny.

The sparse point correspondences were extracted by picking similar points using MeshLab, and are provided in the code. Your task is to fill in the missing parts of the algorithm in the ProcrustesAligner.h. More details about the tutorial can be found in the tutorial slides. The resulting mesh bunny_procrustes.off should almost perfectly align green and red bunny.

2. Fine Registration – Iterative Closest Point (ICP)



The Iterative Closest Point (ICP) is used for fine registration. Thus, the input meshes or point clouds have to be roughly aligned. Our partial bunny meshes are close enough to directly apply ICP. Two variants of ICP were discussed during the lecture. One is the point-to-point variant which is already implemented and the other is the point-to-plane variant. You will have to implement a combination of both.

The goal of the ICP is to align source point cloud to the target point cloud by estimating the rigid transformation (rotation and translation) that transforms source point cloud into the target point cloud. It is an iterative algorithm, where each iteration consists of two steps:

a) Finding the nearest (closest) target point for each source point, using the current rigid transformation (using approximate nearest neighbor algorithm);
b) Estimating the relative rigid transformation by minimizing the distance between the matched points. For the point-to-point variant, this can be done by using Procrustes. The point-to-plane variant involved non-linear optimization. Fortunately, there is a way to linearize this and solve a linear system like you did in Task 1. This linearization is explained in the following paper:
https://www-new.comp.nus.edu.sg/~lowkl/publications/lowk_point-to-plane_icp_techrep.pdf

After each iteration the current relative rigid transformation is updated. The procedure is repeated until convergence (in our examples we use a fixed number of iterations).

We use two variants of the distance function: point-to-point and point-to-plane distance. If s is a source point with position p_s and t is a target point with position p_t and normal n_t, then we define the distance functions as:

a) Point-to-point: $d(s,t) = ||Tp_s - p_t||^2$
b) Point-to-plane: $d(s,t) = (n_t^T(Tp_s - p_t))^2$

The first distance function corresponds to the Procrustes algorithm which you already implemented. Your task is to implement a combination of both the first and the second distance function in ICPOptimizer.h. The optimization variable is relative camera pose (rigid transformation), which can be parametrized with 6 parameters: 3 parameters for rotation, represented in axis-angle (SO3) notation, and 3 parameters for translation, given as a 3D vector. Implementation of the point-to-plane distance function is discussed in the paper linked above. To also add the point-to-point distance function to the system, add the corresponding residuals to the system matrix. Note that when written in matrix form, there is only one residual for the point-to-plane function, but three residuals for the point-to-point function (one for each coordinate).

The final mesh bunny_icp.off should be computed using both point-to-point and point-to-plane cost functions. The main difference between using only point-to-point distance compared to using also point-to-plane distance is the speed of convergence.

3. Submit your solution

a. Upload the resulting meshes: bunny_procrustes.off, bunny_icp.off
b. Submit the following files: ProcrustesAligner.h, ICPOptimizer.h