22 November 2017

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 **two weeks of working time** (note the deadline date in the footer). To get the 0.3 bonus, you have to pass all three exercises. The solutions of the exercises are presented the day after the submission deadline respectively. There is a Q&A session every second week (Wednesday Tutorial slot). Feel free to use the Moodle forum 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 in a separate zip-archive that can be downloaded from a separate link, provided on Moodle.

For this project we use **Eigen** (http://eigen.tuxfamily.org/), **CeresSolver** (http://ceres-solver.org/), **FLANN** (https://www.cs.ubc.ca/research/flann/), **FreeImage** (http://freeimage.sourceforge.net/). 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.

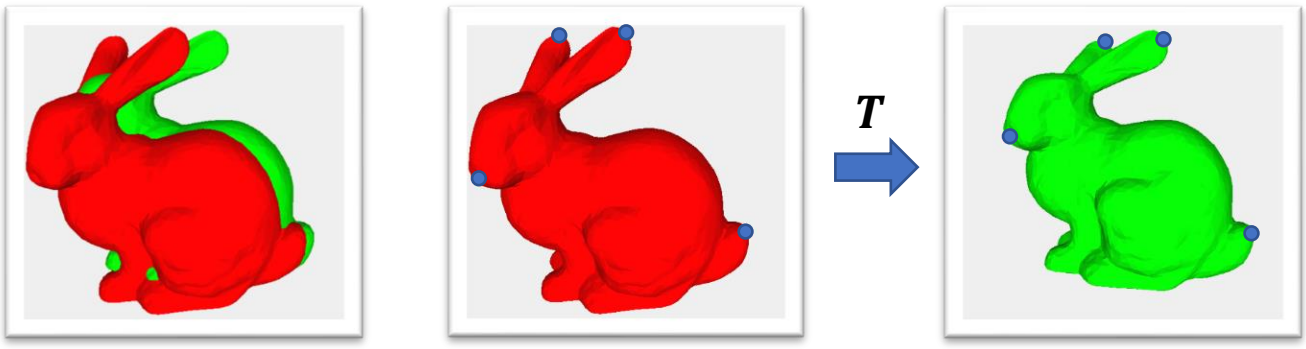
Expected submission files: ProcrustesAligner.h, ICPOptimizer.h, PointCloud.h, bunny_procrustes.off, bunny_icp.off, mesh_merged.off

Exercise 3 – Registration – Procrustes, ICP, CeresSolver

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

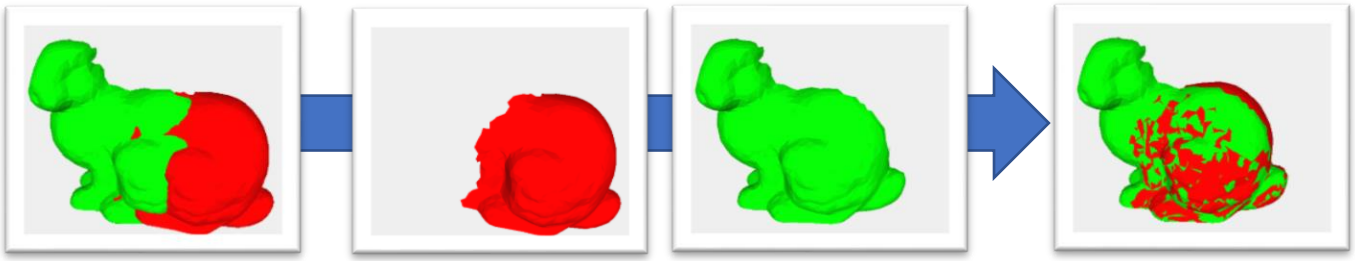


In this task you have to implement the Procrustes Algorithm to align two bunny meshes (red & green). Exploiting the sparse point correspondences shown as blue dots, we are able to estimate the transformation T that transforms the red bunny to match the green bunny.

22 November 2017

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 pointclouds have to be roughly aligned (e.g. using the Procrustes Algorithm). Our partial bunny meshes are close enough to directly apply ICP. Your task is to implement two variants of the ICP algorithm using the CeresSolver library to optimize for the best rigid transformation.

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 with one iteration of Levenberg-Marquardt algorithm.

After each iteration the current relative rigid transformation is updated. The procedure is repeated until convergence (in our examples we used 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 ps and t is a target point with position pt and normal nt, then we define the distance functions as:

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

Your task is to implement both distance functions as cost functions in Ceres solver (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.

22 November 2017

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. For our partial shapes the variant with only point-to-point constraints needs 20 ICP iterations to converge, while the addition of point-to-plane constraints reduces the number of ICP iterations to 10.

3. Camera Tracking using Frame-to-Frame ICP

We now want to test your algorithm on real-world data, that you already used in the first exercise sheet. If you successfully implemented the ICP algorithm in the previous task, then you are almost done. The only thing that needs to be added is the computation of point cloud normal in the PointCloud.h. The procedure to compute the normals from depth images was mentioned during the lectures.

The image presents the expected result. The Frame-to-Frame ICP produces a depth map mesh with current camera pose every 5 frames. You need to take meshes of frame 1, 11 and 21 and put them into a common mesh mesh_merged.off in MeshLab.

4. Submit your solution

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

>>> Submission Deadline: 5.December 2017 23:55 <<<