2 Dezember 2019

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 four exercises, with the fifth 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 (Monday 14:00 – 15:00) to ask questions!

To inspect your result, you need a 3D file viewer like **MeshLab** (http://www.meshlab.net/). The exercise zip-archive contains the source files and a CMake configuration file. The required libraries and data 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://github.com/mariusmuja/flann) and **FreeImage** (http://freeimage.sourceforge.net/).

If you work under Linux you have to install the freeimage lib using:

```
$ sudo apt-get install libfreeimage3 libfreeimage-dev
```

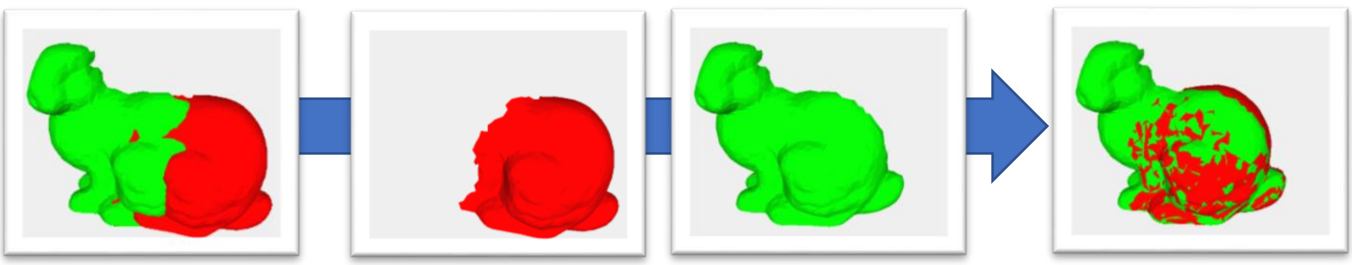
Expected submission files: ICPOptimizer.h, PointCloud.h, bunny_icp.off, mesh_merged.off

Exercise 5 – Registration – ICP, CeresSolver

In this exercise we want to align 3D shapes. We will explore different variants of the Iterative Closest Points (ICP) algorithm and test them on shapes provided either as meshes or with depth maps.

Tasks:

1. Fine Registration – Iterative Closest Point with Ceres



The Iterative Closest Point (ICP) is used for fine registration. Thus, the input meshes or point clouds 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.

2 Dezember 2019

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

a) Finding the nearest (closest) destination point for each source point, using the current rigid transformation (using the approximate nearest neighbor algorithm);
b) Estimating the relative rigid transformation by minimizing the distance between the matched points with one iteration of the Levenberg-Marquardt algorithm.

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 d is a destination point with position p_d and normal n_d, then we define the distance functions as:

a) Point-to-point: $d(s,t) = ||Mp_s - p_d||^2$
b) Point-to-plane: $d(s,t) = (n_d^T(Mp_s - p_d))^2$

Your task is to implement both distance functions as cost functions in Ceres solver (in ICPOptimizer.h). The optimization variable is the 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.

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 distances compared to using also point-to-plane distances 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.

2. Linearized Iterative Closest Points

You might have noticed that the point-to-point variant from the first task is a linear least-squares problem which can be solved with the Procrustes algorithm from the previous exercise. On the other hand, the point-to-plane metric is non-linear in the rotation. However, there exists a method to linearly approximate the point-to-plane metric which would then allow us to use a linear solver for the alignment.

2 Dezember 2019

Let $M = T(t_x, t_y, t_z) \cdot R(\alpha, \beta, \gamma)$ be a transformation matrix that transforms the source point cloud to the destination point cloud. For small rotation angles $\alpha, \beta, \gamma \approx 0$ we can use the following approximation for the rotation matrix $R(\alpha, \beta, \gamma)$:

$$\hat{R}(\alpha, \beta, \gamma) = \begin{pmatrix} 1 & -\gamma & \beta & 0 \\ \gamma & 1 & -\alpha & 0 \\ -\beta & \alpha & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

We now want to find a matrix $\widehat{M} = T(t_x, t_y, t_z) \cdot \hat{R}(\alpha, \beta, \gamma)$ that minimizes:

$$\sum_i \left((\widehat{M} s_i - d_i) n_i \right)^2$$

The expression $(\widehat{M} s_i - d_i) n_i$ can be written as a linear expression of the unknown rotation and translation parameters:

$$(\widehat{M} s_i - d_i) n_i = (\widehat{M} \begin{pmatrix} s_{ix} \\ s_{iy} \\ s_{iz} \\ 1 \end{pmatrix} - \begin{pmatrix} d_{ix} \\ d_{iy} \\ d_{iz} \\ 1 \end{pmatrix}) \begin{pmatrix} n_{ix} \\ n_{iy} \\ n_{iz} \\ 0 \end{pmatrix}$$

This can further be rewritten in the form $Ax - b$ where x is the vector of unknowns:

$$x = (\alpha, \beta, \gamma, t_x, t_y, t_z)^T$$

Thus our goal is to minimize $|Ax - b|^2$ which is a linear least-squares problem and can be solved using singular value decomposition. It is important to note that the approximation $\hat{R}(\alpha_{opt}, \beta_{opt}, \gamma_{opt})$ may not be a valid rotation matrix and one should use $R(\alpha_{opt}, \beta_{opt}, \gamma_{opt})$ instead.

Your task is to implement the point-to-plane constraint function in the LinearICPOptimizer class. For a more detailed analysis please refer to the "Linear Least-Squares Optimization for Point-to-Plane ICP Surface Registration" paper attached to the exercise and to the exercise slides. Note that the system has $4n$ rows where n is the number of points. For each point, we expect you to add both a point-to-plane constraint (one row) as described in the paper, and a point-to-point constraint (three rows, one for each coordinate). To determine the elements of the rows that correspond to the point-to-point constraints, try expanding $\widehat{M} s_i = d_i$ by hand and take a look at which coefficients are multiplied by the unknowns and which are free.

You should experiment with different number of iterations for all ICP variants. How does the convergence speed change when switching from one method to another?

2 Dezember 2019

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. You still need to implement the computation of point cloud normals in the PointCloud.h header file and set the macro in main.cpp to also run this task. Make sure you run the code in Release mode, otherwise the optimization will take too long.

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 frames 1, 11 and 21 and put them into a common mesh mesh_merged.off in MeshLab (Filters -> Mesh Layer -> Flatten Visible Layers).

4. Submit your solution

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