Student Name: Federico Gelain
Student ID: 2076737

UNIVERSITÀ DEGLI STUDI DI PADOVA
3D Data Processing
Lab 3: Iterative Closest Point

Note: in the first three tasks, when referring to the **source** point cloud, it's implied that **source_for_icp** is used, which is the source point cloud to which has been applied the transformation computed so far in the ICP main loop.

Task 1: Find Closest Point

The goal is to find for each point in the **source** point cloud the closest point in the **target** one. To do so, following the implementation of the method **compute_rmse()**, instead of using two nested loops (one to iterate through each point of source and one to iterate through each point of target), a KDTree is created from the target point cloud, in order to speed up the search process. The research of the nearest neighbour of each source point in the target KDTree is then performed using **SearchKNN()** (setting the *knn* parameter to 1). If the distance between a source point and the found nearest neighbour is below the threshold provided as parameter (not an outlier), then their indices are stored and their distance is used to compute the RMSE.

Task 2: Transformation using SVD

The aim of this task is to determine the transformation (rotation matrix and translation vector) between the **source** point cloud and the **target** one using Singular Value Decomposition (SVD). This is done in the following way:

1. using the vectors **source_indices** and **target_indices** of size n, provided as parameters, define the ordered set of correspondences $s = \{s_1, ..., s_n\}$, $m = \{m_{j(1)}, m_{j(n)}\}$, $s_i \leftrightarrow m_{j(i)}\ \forall i \in [1, n]$ (m is used instead of t to avoid any confusion with the translation vector t), by looping through the vectors and retrieving the correct points from both point clouds;

2. determine the centroid of each point cloud as:

$$s_c = \frac{1}{n}\sum_{i=1}^{n} s_i \qquad m_c = \frac{1}{n}\sum_{i=1}^{n} m_i \tag{1}$$

3. construct the matrix

$$W = \sum_{i=1}^{n} m_i^{'}(s_i^{'})^T \tag{2}$$

   where $s_i^{'} = s_i - s_c$ and $m_i^{'} = m_i - m_c$.

4. apply SVD to W. Since W is a 3x3 real (square) matrix, the factorization obtained is the following:

$$W = U\Sigma V^T \tag{3}$$

where U and V^T are 3x3 real (square) orthonormal matrices whose determinant is equal to either +1 or -1. The rotation matrix is computed as:

$$R = UV^T \tag{4}$$

However, the special reflection case $det(UV^T) = -1$ has to be handled. It can be easily solved by changing the sign of the determinant of one of the two matrices, which is achieved for example by multiplying the last column of U by -1. To do so, the following diagonal matrix is defined:

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

and finally

$$R = UTV^T \tag{5}$$

To summarize, the rotation matrix is computed as:

$$R = \begin{cases} UV^T & if \quad det(UV^T) \neq -1 \\ UTV^T & if \quad det(UV^T) = -1 \end{cases} \tag{6}$$

As for the translation vector, it can be computed in closed form as

$$t = m_c - Rs_c \tag{7}$$

5. reconstruct the transformation matrix with the obtained R and t.

Task 3: Transformation using LM

In this task the transformation between source and target point cloud is found using the **Levenberg-Marquardt** algorithm, which aims to minimize the registration error:

$$E(R,t) = \sum_{i=1}^{n} \left\| m_{j(i)} - (Rs_i + t) \right\|^2 \tag{8}$$

where $s_i \leftrightarrow m_{j(i)}$ is the i-th correspondence between the source and target point clouds (accessed, like done in the previous task, using the vectors **source_indices** and **target_indices** provided as parameter). To do so, an auto-differentiable cost function is created inside the class **PointDistance**, whose input is a correspondence source-target points. The residual is computed as the difference between the transformed source point (using the transformation parameters, which are going to be optimized) and the target point. Inside **get_lm_icp_registration()** a **ceres::Problem** is created, adding the cost function and the corresponding residual block for each pair of points. As loss function, in accordance to (8), *nullptr* is provided, which corresponds to the squared norm of the residuals. After solving the optimization problem, the triplet of Euler angles (α, β, γ) and the translation vector values, stored in **transformation_arr**, are retrieved to obtain R and t. The former is computed as:

$$R = R_x(\alpha) * R_y(\beta) * R_z(\gamma) \tag{9}$$

Where the function **Eigen::AngleAxisd()** was used to automatically define the rotation matrix along the specified axis and the corresponding angle. With R and t, the transformation matrix is then reconstructed.

Task 4: ICP Main Loop

To find the transformation between source and target point cloud, a while loop is performed until the convergence criteria is satisfied (the difference between the current RMSE **curr_rmse** and the one of the previous iteration **prev_rmse** is smaller than the **relative_rmse** provided as parameter) or until the maximum number of iterations is reached. At each iteration of the loop:

1. call **find_closest_point()** to get the vectors of indices for the correspondences in both clouds and the corresponding RMSE;
2. update prev_rmse and curr_rmse accordingly with the new value obtained;
3. retrieve the current transformation T_{curr} using either SVD or LM and use it to transform **source_for_icp**;
4. compute the new full transformation from **source** to **target** as $T_{full} = T_{curr} * T_{prev}$, where T_{prev} is the full transformation obtained in the previous iteration, accessed using **get_transformation()**, and set it using **set_transformation()**;

Results

All the RMSE values and the number of iterations taken using SVD and LM for ICP registration are listed in the following table

Data Item	Bunny	Dragon	Vase
RMSE with SVD	0.00341359	0.00564088	0.0162213
RMSE with LM	0.00341359	0.00564088	0.0162213

Both methods find the same transformation to align the two point clouds, as it can be seen by the value of RMSE, and converge at the same number of iterations. It's important to point out that LM takes noticeably more time than SVD, due to the optimization required at each iteration of the algorithm.
Below are shown some screenshots of the results with both methods, compared to the starting configuration, for the bunny, dragon and vase dataset respectively

Figure 1: Initial

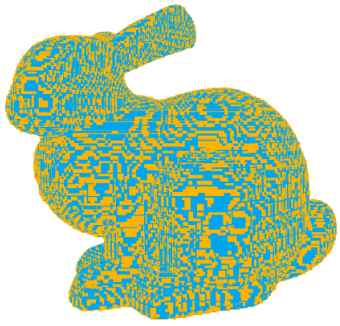

Figure 2: SVD

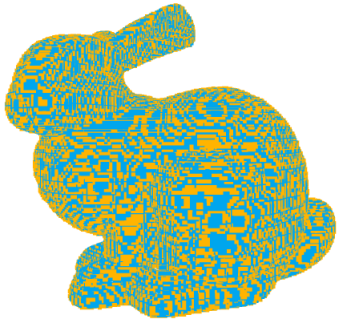

Figure 3: LM

Figure 4: Initial

Figure 5: SVD

Figure 6: LM

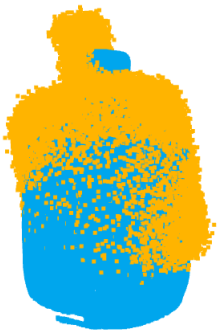

Figure 7: Initial

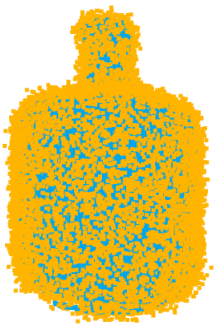

Figure 8: SVD

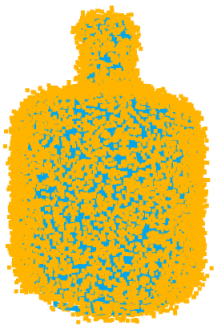

Figure 9: LM