3D Data Processing - $3^{rd} Assignement$

Luca Sambin

1 Problem Description

The goal of this assignment was to extend the provided C++ software to implement the Iterative Closest Point algorithm, which is employed to minimize the difference between two point cloud. So basically, given a source and a target point cloud roughly aligned, find the fine alignment transformation of the source to the target cloud, using two possible method **SVD** (Singular value decomposition) or **LM** (Levenberg–Marquardt).

2 Solution Structure

To reach the goal it was necessary to perform some tasks following the algorithm described in class. So the problem was divided in 4 different points:

- ***execute_icp_registration():*** The first task referred to the implementation main ICP loop. So basically, the first thing to do is check convergence criteria and and the current iteration. The convergence criteria that was chosen is essentially, this one: *if relative change (difference) of inliner RMSE score is lower than* ***relative_rmse****, the iteration stops.* The current and previous RMSE value is computed by the function *find_closest_point()*. Then given the output of the function *find_closest_point()*, which are used to calculate the transformation between the two point cloud, is called respectively:
  - If *mode=="svd"* use *get_svd_icp_transformation()*;
  - If *mode=="lm"* use *get_lm_icp_transformation()*;
  - If mode is something else then the algorithm stop.

  Finally, the variable *source_for_icp_* is used to store transformed source points.

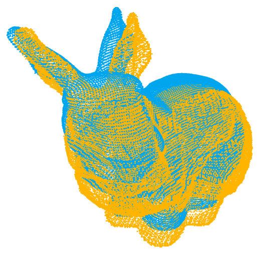
- ***find_closest_point():*** In this task is calculated the source and target indices. So, for each source point was found the closest one in the target and discarded if their distance is bigger than threshold, given as input parameter. Finally compute also the RMSE.

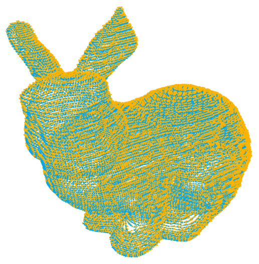
- ***get_svd_icp_transformation():*** The third task was essentially used to calculate transformation using SVD. So basically, the process is the following one: *find point clouds centroids and subtract them, then use SVD (****Eigen::JacobiSVD<Eigen::MatrixXd>****) to find best rotation and translation matrix.* It's used source_indices and target_indices variable, given by the *find_closest_point()* to extract point, to compute the 3x3 matrix to be decomposed. Also the special reflection case (**determinant**$(\mathbf{UV}^T)$ **= -1**) was managed by changing the sign of the last column of U.

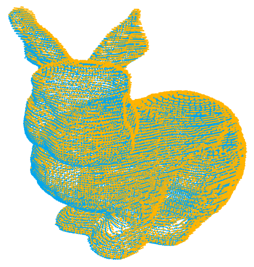
- ***get_lm_icp_registration():*** The last task was essentially used to calculate transformation using LM (Ceres). Also in this case was used source_indices and target_indices variable, given by the *find_closest_point()* to extract point, to compute the final transformation matrix.

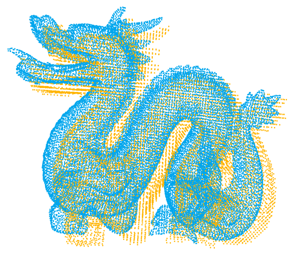
3 Results

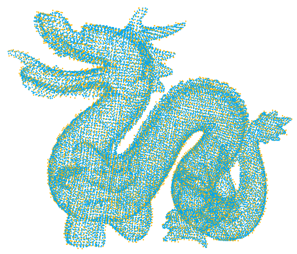
In this section is showed the results obtained with the two algorithms on the three given datasets, showing also the final RMSE error and the iteration in which the algorithm converge. You can find all the results into ***Result*** folder. The last thing that is important to note is that the time required to perform the task varies a lot, depending on the method that is chosen. In fact, the **SVD** method is much faster than the **LM** method.

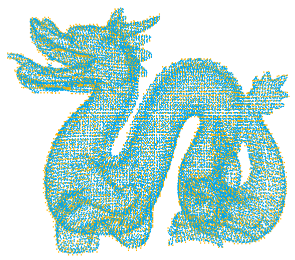

Bunny Dataset

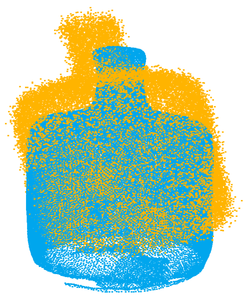

LM Output
RMSE: 0.00493606
$21^{st} Iteration$


SVD Output
RMSE: 0.00763901
$23^{rd} Iteration$

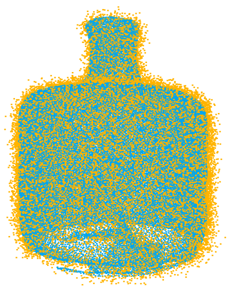

Dragon Dataset


LM Output
RMSE: 0.00569335
$19^{th} Iteration$

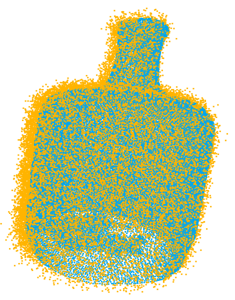

SVD Output
RMSE: 0.00568077
$13^{rd} Iteration$



Vase Dataset



LM Output
RMSE: 0.0165296
29th *Iteration*



SVD Output
RMSE: 0.0218471
25th *Iteration*