Edoardo Bastianello

ID: 2053077

REPORT LAB 3

Function *find_closest_point()*:

First, I created a KD-tree using the points of the target point cloud. Next, I applied the current estimate of the transformation matrix on a copy of the source point cloud. Then, I looped over all the points of the transformed source point cloud to do the following computation:

- For each of the point of the transformed point cloud, I found the closest point in the target point cloud by using the function *SearchKNN()* (of all K neighbors I only kept the closest point). Next, if the distance between the two corresponding points was lower or equal than the threshold specified in the input parameters of the function, I saved the indices of the corresponding points and I updated the mean square error.

Finally, I computed the current root mean square error and I returned the required parameters.

Function *get_svd_icp_registration()*:

First, I applied the current estimation of the transformation matrix to a copy of the source point cloud. Next, I computed the centroids of the target point cloud and of the transformed copy of the source point cloud. Then, I created two empty matrices of dimension *3xN*, named *subtracted_source* and *subtracted_target*, where *N* is the number of indices found with the function *find_closest_point()*.

At this point, for each index of the arrays of indices found with the function *find_closest_point()*, I did the following computation:

- I retrieved the point of the transformed source point cloud and the point of the target point cloud corresponding to the current index;
- I subtracted the centroid of the relative point cloud to the two points just retrieved and I added the results as a column to the matrices *subtracted_source* and *subtracted_target*.

Then, following the theory, I computed the product between the matrix *subtracted_target* and the transpose of the matrix *subtracted_source* to get the matrix *W*.

Next, I applied the SVD decomposition to the W matrix to get the matrices *U* and *V*. At this point, I computed the rotation matrix *R_matrix* as the product between the *U* matrix and the transpose of the *V* matrix:

$$\boldsymbol{R\_matrix} = U\,V^T$$

I also handled the special reflection case (*det(R_matrix) = -1*) by computing the rotation matrix *R_matrix* as the product of the following three matrices instead:

$$\boldsymbol{R\_matrix} = U\,diag(1,1,-1)\,V^T$$

Then, I computed the translation vector *t_vec* as:

$$\boldsymbol{t\_vec} = centroid\_target\_cloud - R\_matrix * centroid\_source\_cloud$$

Finally, I concatenated the rotation matrix *R_matrix* and the translation vector *t_vec* to obtain the transformation matrix.

Function *get_lm_icp_registration()*:

First, I applied the current estimation of the transformation matrix to a copy of the source point cloud.

Then, for each index in the vector containing the indices found with the function *find_closest_point(),* I did the following computation:

- I retrieved the point of the source point cloud and the point of the target point cloud corresponding to the current index;
- I called the function *AddResidualBlock()* for the Ceres solver.

Next, I solved the optimization problem using Ceres solver to recover the rotation matrix and the translation vector. Finally, I created the transformation matrix by concatenating the rotation matrix with the translation vector.

To complete this function, I also defined a templated functor to compute the residual by adapting the Ceres tutorial to this problem. In particular, in this case there are only 6 parameters to be optimized (*rx, ry, rz, tx, ty, tz*) and the residual is 3-dimensional. In fact, the residual is given by the difference between the point in the transformed source point cloud and the corresponding point in the target point cloud.

Function *execute_icp_registration()*:

This function contains the man ICP loop. For each iteration:

- I find the closest correspondences between the points of the two point clouds and the current root mean square error by calling the function *find_closest_point(threshold);*
- I check if the algorithm converged with the following condition:

  $$|previous\_error - current\_error| < relative\_rmse$$

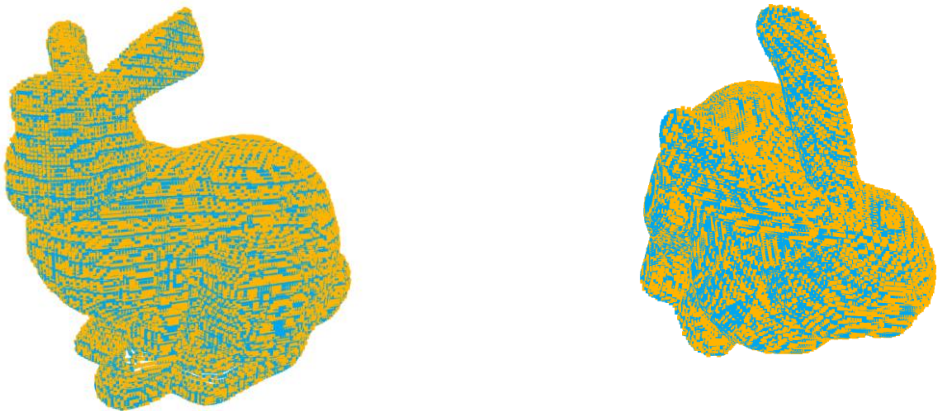
  If this condition is true, I stop the algorithm. I also check if *previous_error < current_error*; if this condition is true, I restore the estimation of the transformation matrix that I obtained in the previous iteration since it means that the current estimation got worse;
- If the algorithm didn't converge, I call either the function *get_svd_icp_tranformation()* or *get_lm_icp_registration(),* depending on the input parameter of the function, in order to get the new estimate of the transformation matrix;
- I update the overall transformation by multiplying the transformation matrix that I just computed by the transformation matrix up to this iteration. This is due to the fact that the transformation that I compute with *get_svd_icp_registration()* or *get_lm_icp_registration()* doesn't relate the original source point cloud with the target point cloud; instead, it only relates the source point cloud transformed with respect to the previous estimation of the transformation matrix with the target point cloud. For this reason, the overall transformation between the original source point cloud and the target point cloud is equal to the product between the overall transformation matrix up to that iteration and the current transformation between the transformed source point cloud and the target point cloud. In fact, applying multiple transformation matrices in sequence is equivalent to applying the product of all these matrices just once.

Results:

For all datasets, the computation of the transformation matrix with LM required more time than with SVD, but the resulting RMSE with LM is slightly better.

Note: the obtained clouds are contained in the “output_clouds” folder.

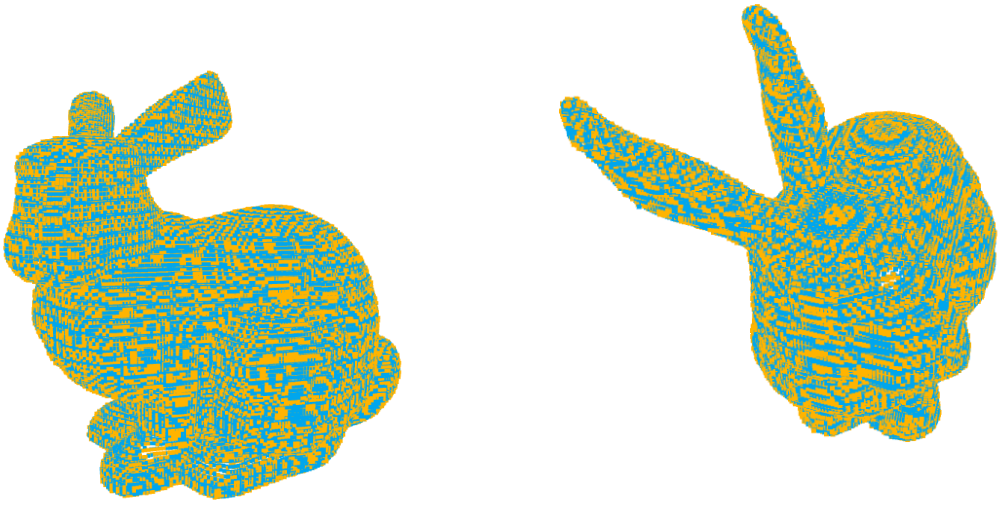
Bunny SVD



Number of Iterations = 27

RMSE = 0.00401692

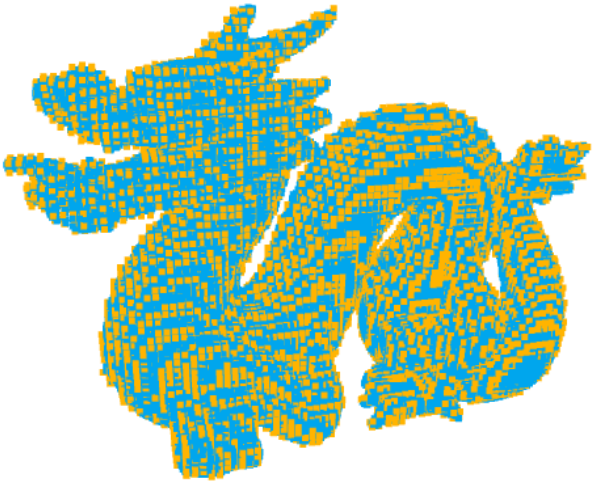
Bunny LM



Number of Iterations = 23

RMSE = 0.00341361

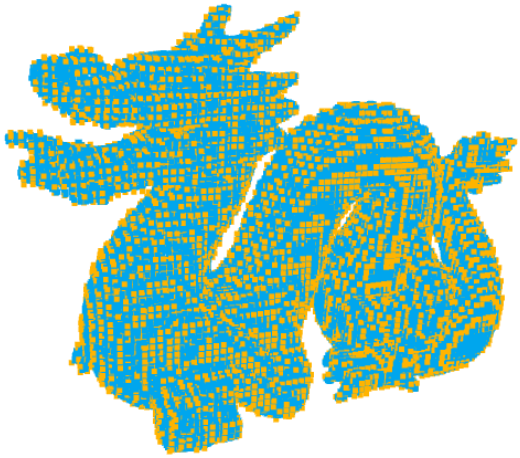
Dragon SVD



Number of Iterations = 13

RSME = 0.00568868

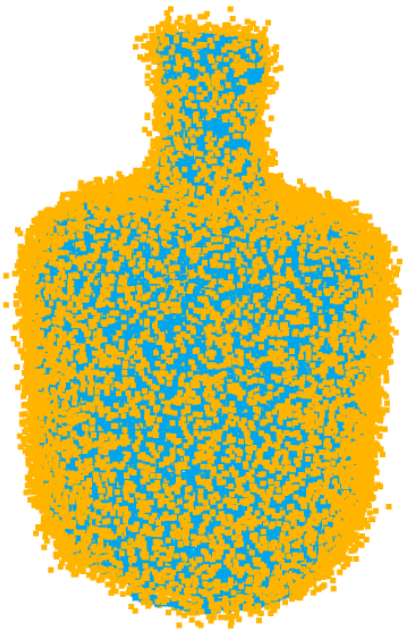
Dragon LM



Number of Iterations = 20

RMSE = 0.00564154

Vase SVD



Number of Iterations = 26

RMSE = 0.0162231

Vase LM

Number of Iterations = 29

RMSE = 0.0162209