Lab 3: Iterative Closest Point Cloud Registration

Mohammadhossein Akbari Moafi

Task 1: find_closest_point()

A KD-Tree is used to organize points in a k-dimensional space for efficient query execution, such as nearest neighbor searches. The implementation uses open3d::geometry::KDTreeFlann to build the KD-Tree for the target point cloud.

The find_closest_point function iterates through each point in the source point cloud, finds the nearest neighbor in the target cloud using the KD-Tree, and includes the pair in the result if their distance is within the threshold. It also computes the RMSE based on the squared distances.

Task2: get_svd_icp_registration()

The first step in the get_svd_icp_transformation function is to compute the centroids of the source and target point clouds. The centroids are calculated using formula (1) by averaging the coordinates of the corresponding points in each point cloud.

$$\mathbf{d}_c = \frac{1}{n}\sum_{i=1}^{n}\mathbf{d}_i \qquad \mathbf{m}_c = \frac{1}{n}\sum_{i=1}^{n}\mathbf{m}_i \tag{1}$$

Next, the centroids are subtracted from each point in their respective point clouds. This step centers the point clouds around the origin, simplifying the subsequent calculations.

The centered point clouds are used to compute the matrix A, which is decomposed using SVD. The decomposition provides matrices U and V, which by utilizinf formula (2) they are used to compute the rotation matrix R.

$$\hat{\mathbf{R}} = \arg\min_{\mathbf{R}} \sum_{i=1}^{n} \|\mathbf{m}'_i - \mathbf{R}\mathbf{d}'_i\|^2 = \mathbf{U}\mathbf{V}^\top$$

A special reflection case is managed by adjusting the sign of the third column of U if the determinant of R is negative, ensuring a proper rotation matrix by utilizing formulas (3).

$$\det(\mathbf{U}\mathbf{V}^\top) = -1$$

then (3)

$$\hat{\mathbf{R}} = \mathbf{U}\,\mathrm{diag}(1,1,-1)\mathbf{V}^\top$$

The translation vector t is computed using the centroids of the target and rotated source point clouds(formula (4)). This vector, along with the rotation matrix R, forms the final transformation matrix.

$$\hat{\mathbf{t}} = \mathbf{m}_c - \hat{\mathbf{R}}\mathbf{d}_c \tag{4}$$

Task 3: get_lm_icp_registration()

PointDistance struct, encapsulates the auto-differentiable cost function required by Ceres Solver. This struct computes the residuals based on the difference between the transformed source points and the target points.

The main function get_lm_icp_registration uses the PointDistance struct to set up and solve the optimization problem using the Ceres Solver. The goal is to find the best rotation and translation that align the source points with the target points.

In this function:

1. Initialize the transformation matrix to the identity matrix.
2. Set up the Ceres Solver options.
3. Create a vector to hold the transformation parameters (3 for rotation and 3 for translation).
4. Iterate over the point correspondences, adding residual blocks to the problem using the PointDistance struct.
5. Solve the optimization problem using the ceres::Solve function.
6. Convert the optimized rotation vector to a rotation matrix.
7. Update the transformation matrix with the optimized rotation and translation.

Task 4: execute_icp_registration()

The execute_icp_registration function manages the main loop of the ICP algorithm. It uses the class variable source_for_icp_ to store transformed source points and updates the class variable transformation_ with the calculated transformation matrix.

The main ICP loop iterates until the maximum number of iterations (max_iteration) is reached. In each iteration, the following steps are performed:

1. **Finding Closest Points**:

   The find_closest_point function is called with a specified threshold to identify corresponding points between the source and target point clouds.

2. **Convergence Check**:

   The algorithm checks for convergence by comparing the current RMSE with the previous RMSE. The loop terminates if:

- The RMSE is increasing (rmse > previous_rmse).
- The improvement in RMSE is less than a specified threshold (relative_rmse), indicating minimal progress.

3. **Transformation Calculation**:

   Depending on the specified mode, the function computes the transformation matrix using either SVD or LM methods:

   - If mode is "svd", get_svd_icp_transformation is called.
   - If mode is "lm", get_lm_icp_registration is called.
   - If an unknown mode is specified, an error message is printed, and the function exits.

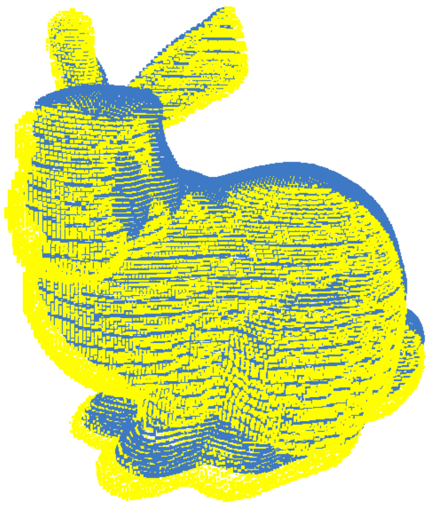
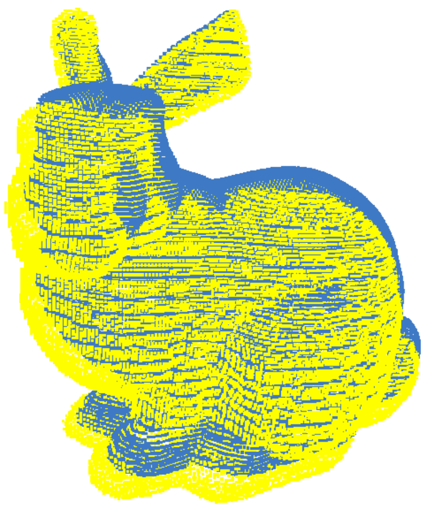
4. **Updating the Source Point Cloud**:

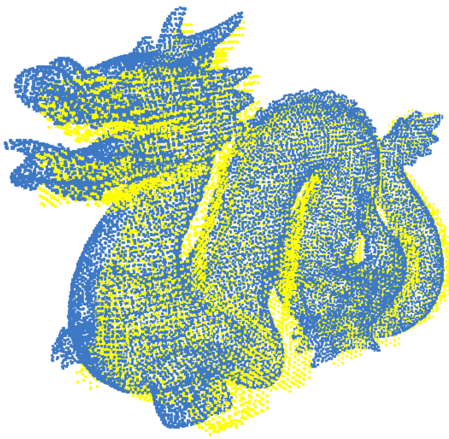
   The source point cloud (source_for_icp_) is transformed using the computed transformation matrix. This transformation aligns the source points closer to the target points.

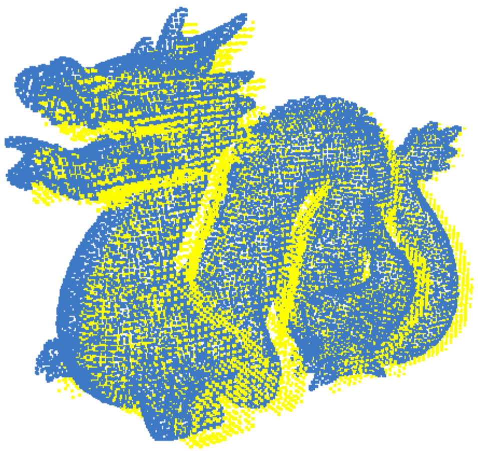
5. **Iteration Update**:

   The previous_rmse is updated with the current RMSE. The iteration counter is incremented.

Result:

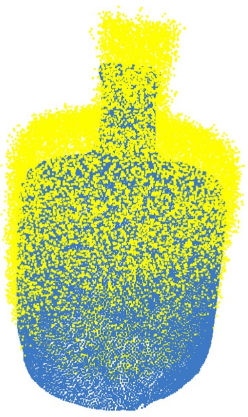
	Screenshot		**Dataset**
	 *SVD*	 *LM*	**Bunny**
RMSE	**0.0339172**	**0.0332919**	

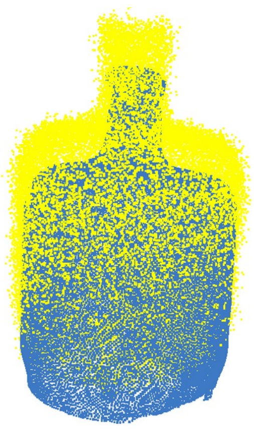

SVD


LM

Dragon

	SVD	*LM*
RMSE	**0.0254031**	**0.0235279**


SVD


LM

Vase

	SVD	*LM*
RMSE	**0.059053**	**0.0590594**