# 3D DATA PROCESSING LAB 3

Giuseppe Labate mat.2095665

**Topic:** Iterative Closest Point Cloud Registration

**Goal:** Given a source and a target point cloud roughly aligned, find the fine alignment  
transformation of the source to the target cloud.

## Work description

### Task 1 – **find\_closest\_point**

For this task, the objective was to find the nearest target\_point for each source\_point.  
To do so, I used a KD-tree to store the target\_ point cloud, which accelerates the process of finding the nearest source\_point.  
Next, a source\_clone point cloud was created to save the source\_for\_icp\_ point cloud, which is the source point cloud transformed by the current estimation of transformation\_.

To find the closest point, the SearchKNN function with K = 1 was exploited.

When a neighbor is found, its square distance is checked. If it’s below a certain arbitrary threshold (0.2 in this case), the source and target indices are saved into two std::vector and the mean square error (MSE) is updated to keep track of the current error.

After the for loop is completed, the root mean square error (RMSE) is calculated by taking the square root of the final MSE.

Finally, a tuple composed of the source\_indices, target\_indices and RMSE is returned.

### Task 2 – **get\_svd\_icp\_registration**

Here it was requested to apply the singular value decomposition (SVD) to find the transformation from source to target.

To achieve this, the transformed source point cloud, source\_for\_icp\_ was saved to source\_clone.  
The source\_for\_icp\_ is the initial source point cloud to which the current estimation of the transformation\_ was applied.

Next, the centroids for source and target point clouds were calculated and saved in the source\_centroid and target\_centroid 3D vectors, respectively.

With the centroids identified, the difference between each point in the point cloud and its centroid was retrieved, following the nearest neighbor (NN) matching indices order, returned by find\_closest\_point.  
In other words, for each iteration ***i***, the source\_clone.points\_[source\_indices[i]] and target\_clone.points\_[target\_indices[i]] were picked to compute the difference between them and their respective centroids:  
source\_point = source\_point - source\_centroid;   
target\_point = target\_point - target\_centroid;  
applying the following formulas:

Subsequentially, the 3x3 **W** matrix was computed by multiplying the and the transpose of and summing the result for each iteration:

Note that *m* variables represent for target point and *d* variables represent source points.

The **W** matrix is crucial for finding the transformation between two point clouds.  
In fact, the rotation matrix can be written as:

Given:

The U and V matrices were found to derive the rotation matrix.  
This was achieved by applying the singular value decomposition (SVD) to **W** by using Eigen::JacobiSVD<Eigen::MatrixXd> svd(W, Eigen::ComputeFullU | Eigen::ComputeFullV);

The special case in which the was also handled, by computing the rotation matrix as follows:

Note that to check if the determinant was negative, it was needed only to compute and multiply and because

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente  
*Figure 1: implementation of SVD*

Next, the translation vector was calculated as the difference between the target\_centroid and the rotated source\_centroid.

Finally, the rotation matrix and translation vector were concatenated into a 4D transformation matrix, which was then returned.

### Task 3 – PointDistance

In task 3 the objective was to complete the PointDistance struct to include an auto-differentiable cost function.  
This part was similar to the Bundle Adjustment in the previous assignment.  
The main difference here was that the function now had to optimize only the transformation array, which is composed by:

* 3 cells for the rotation (expressed in angle-axis representation)
* 3 cells for the translation (tx, ty, tz)

The operator() function was used for two purposes:

* to transform the source\_point first by rotating it with the AngleAxisRotatePoint function and then by translating it by exploiting the last 3 cells of transf array;
* to retrieve the residual by computing the difference between transformed source\_point and target\_point (Note that the residual is 3 dimensional).

### Task 4 – **get\_lm\_icp\_registration**

In this task, it was requested to use the Levenberg-Marquardt algorithm (LM) to find the best roto-translation matrix.

Similar to task 2, the source\_for\_icp\_ point cloud was stored in source\_clone to easily access to the point cloud with the current estimated transformation.

Next, for each point in source\_clone, the cost\_function was created using the coupled source and target points as parameters. In the same iteration, the residual block was added to the problem, with the cost\_function and the transformation\_arr (the array containing the roto-translation transformation) as parameters.

After exiting the loop, the ceres::Solve function was executed to obtain the transformation.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

*Figure 2: Ceres process to use LM to get the estimated transformation*

Once Ceres solved the problem, the transformation\_arr vector was accessed to compute the 4D roto-translation matrix transformation.  
The rotation matrix **R** was obtained from the first three angle-axis cells using the Eigen::AngleAxisd method.   
The translation vector was instead directly extracted from the last three cells.

Finally, the rotation matrix and translation vector were concatenated into a 4D transformation matrix, which was then returned.

### Task 5 – **execute\_icp\_registration**

Task 5 was the most important one because it was the function from which all the other functions were called.  
A primary for loop iterates up to a maximum of max\_iteration (100) times to execute the following steps:

* **Find Closest Points:** Launch find\_closest\_point to get coupled points (the source and target indices) and current RMSE.
* **Check Convergence:** If the absolute difference between previous and current RMSE is less than relative\_rmse (which is equal to 1e-6 in this case), return because the execution converged.
* **Update RMSE:** Update the prev\_rmse to current\_rmse
* **Perform ICP Registration:** Launch get\_lm\_icp\_registration if mode == “lm” or launch get\_svd\_icp\_registration if mode == “svd”
* **Save Transformation:** Save the returned transformation into new\_transformation
* **Update transformation\_ and source cloud:** Update the transformation\_ 4D matrix and transform source\_for\_icp\_ using new\_transformation to avoid recalculating the transformed source cloud each time.

If after max\_iteration the algorithm hasn’t converged, a "Diverged: MAX\_ITERATION surpassed." message is sent.

## Encountered problems

During the assignment, no major problems were encountered.

## Quantitative results

|  |  |  |
| --- | --- | --- |
|  | SVD RMSE | LM RMSE |
| Bunny | 0.00401621 | 0.00341366 |
| Dragon | 0.00568867 | 0.00564134 |
| Vase | 0.0162243 | 0.0162217 |

|  |  |  |
| --- | --- | --- |
|  | SVD ITERATIONS | LM ITERATIONS |
| Bunny | 23 | 21 |
| Dragon | 13 | 19 |
| Vase | 25 | 29 |

## Qualitative results

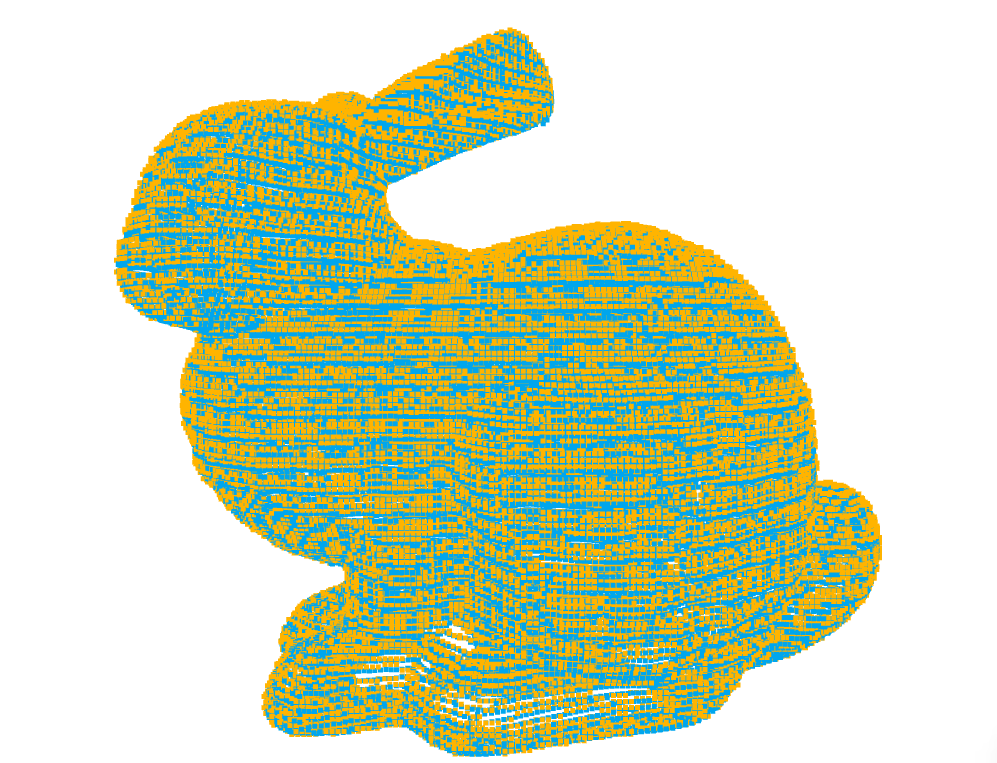
### 4.1) Bunny

Immagine che contiene clipart, disegno, cartone animato, creatività

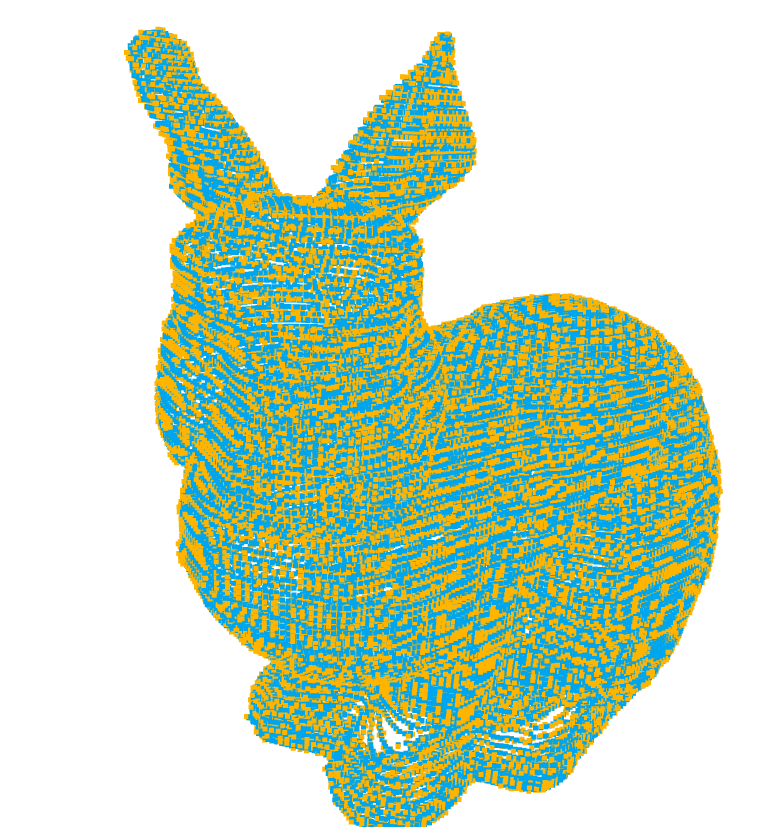
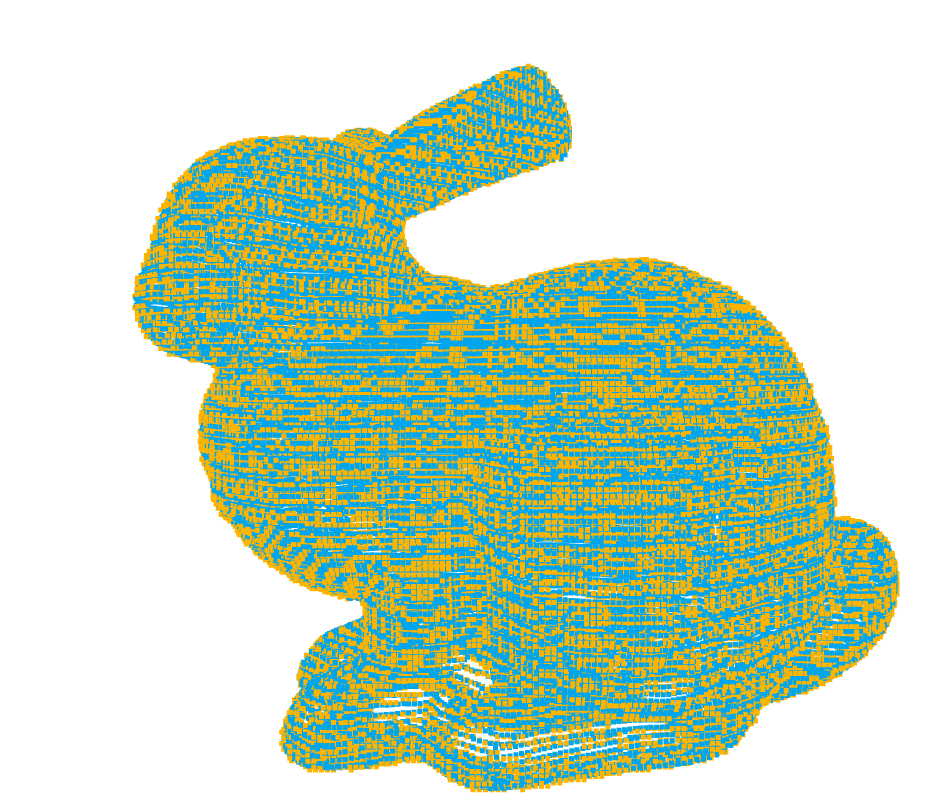
Descrizione generata automaticamente

#### 4.1.1) Bunny SVD 🡪 0.00401621 at iteration 23

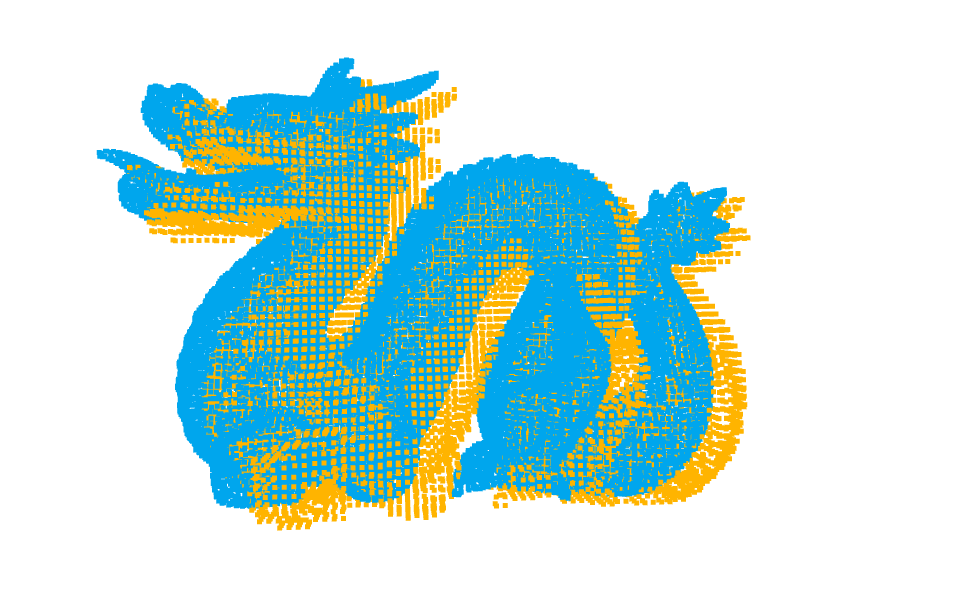
Immagine che contiene disegno, Arte bambini, illustrazione, creatività

Descrizione generata automaticamente 

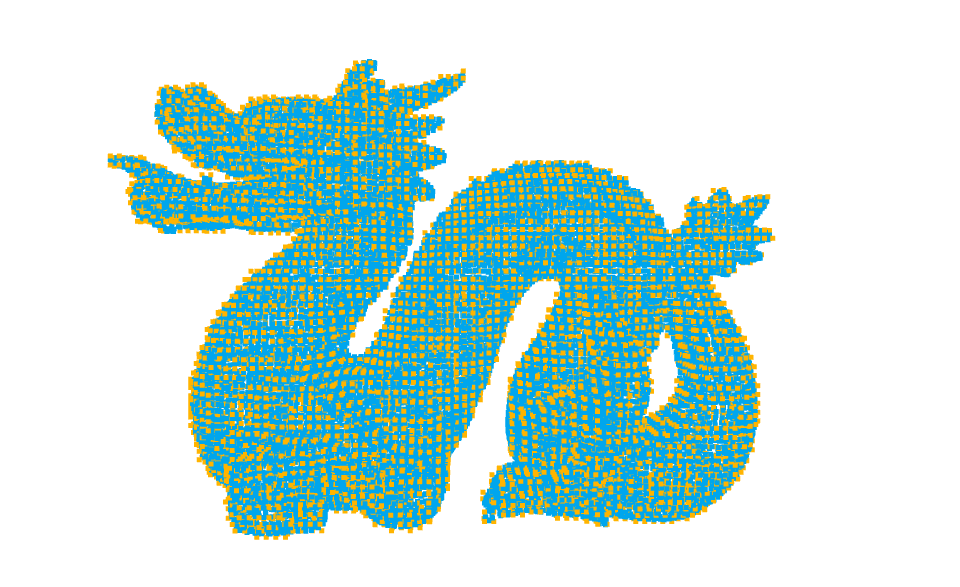
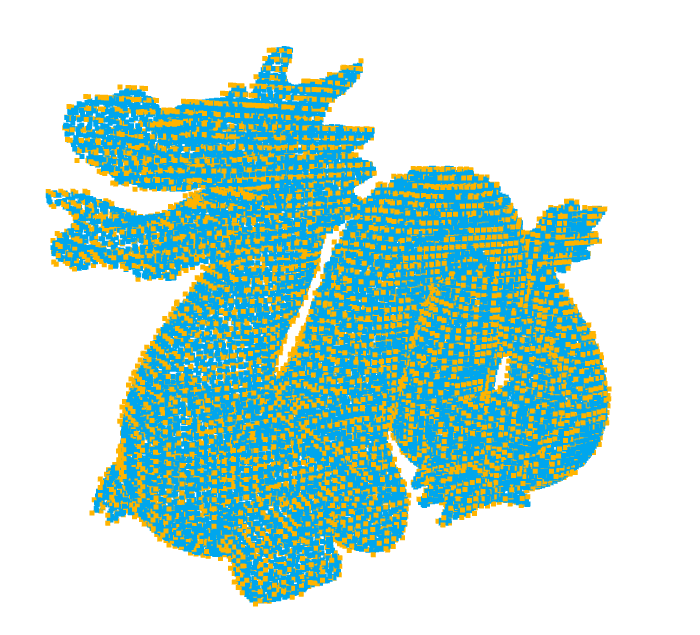
#### 4.1.2) Bunny LM 🡪 0.0341366 at iteration 21

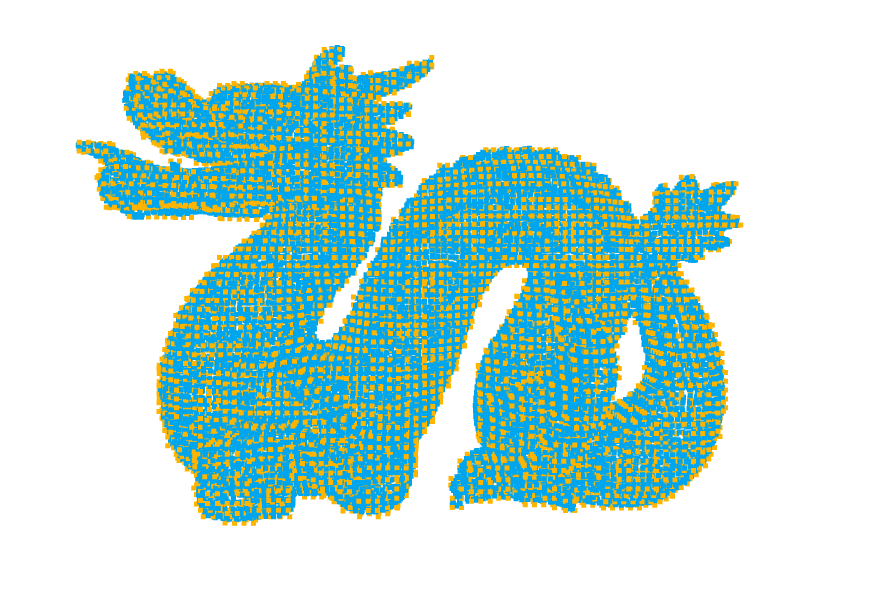
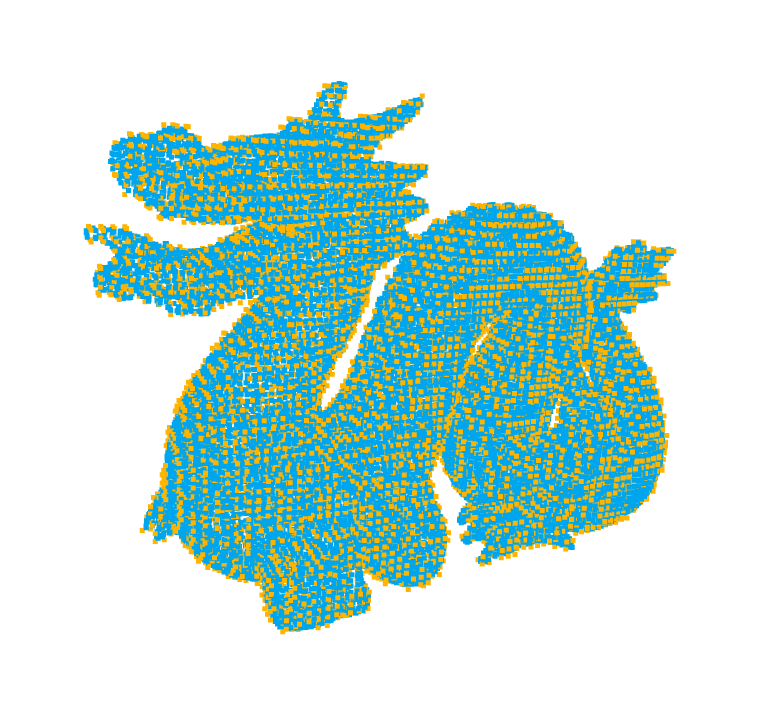
### 4.2) Dragon



#### 4.2.1) Dragon SVD 🡪 0.00568867 at iteration 13

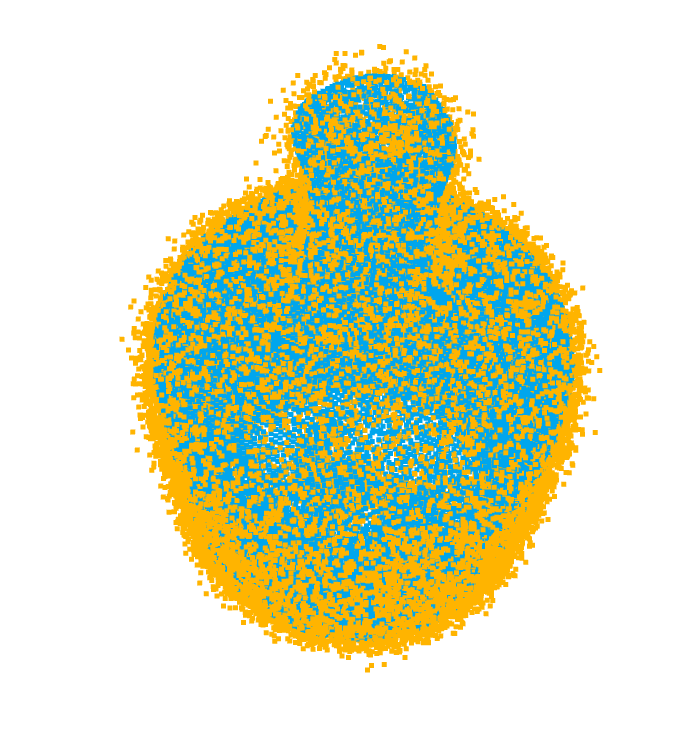
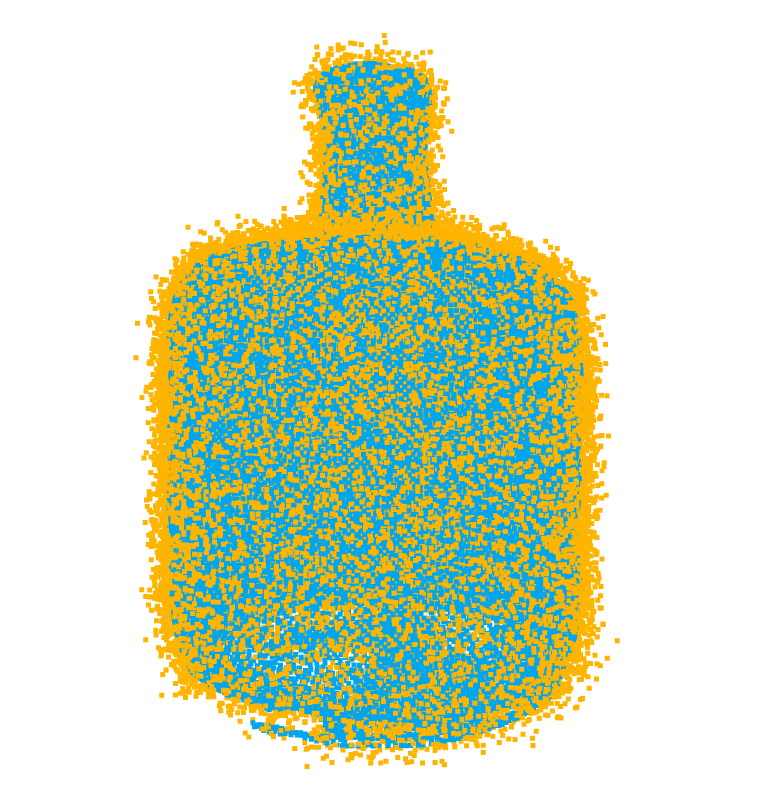
#### 4.2.2) Dragon LM 🡪 0.00564134 at iteration 19

### 4.3) Vase



#### 4.3.1) Vase SVD 🡪 0.0162243 at iteration 25



#### 4.3.2) Vase LM 🡪 0.0162217 at iteration 29

