Research on Feature Descriptors Used for Point Cloud Registration

Robotic Perception and Action: Homework

Team SLAMer: Ma Teng, Sharad Maheshwari, Mingxuan Liu

Table of Contents

1. Introduction

To help robots know our world better, we need to introduce 3D vision and teach computer how to process point cloud data. The main goal of this article is to explain how we process point cloud data and gain a good result in point cloud registration with the help of PCL (point cloud library) and Open3D (library for 3D data processing).

2. Problem description

In order to achieve SLAM, first we need to get the information of the environment, here we use 3d camera to generate PCD files in unity (find more in appendix). Then we want to match two 3d pictures come from the same environment with a different shooting angle and get the transformation, which is the main task of our work.

3. Implementation

There are 2 routes in general when it comes to point cloud registration: local pipelines and global pipelines (specific procedures shown in Figure 1).

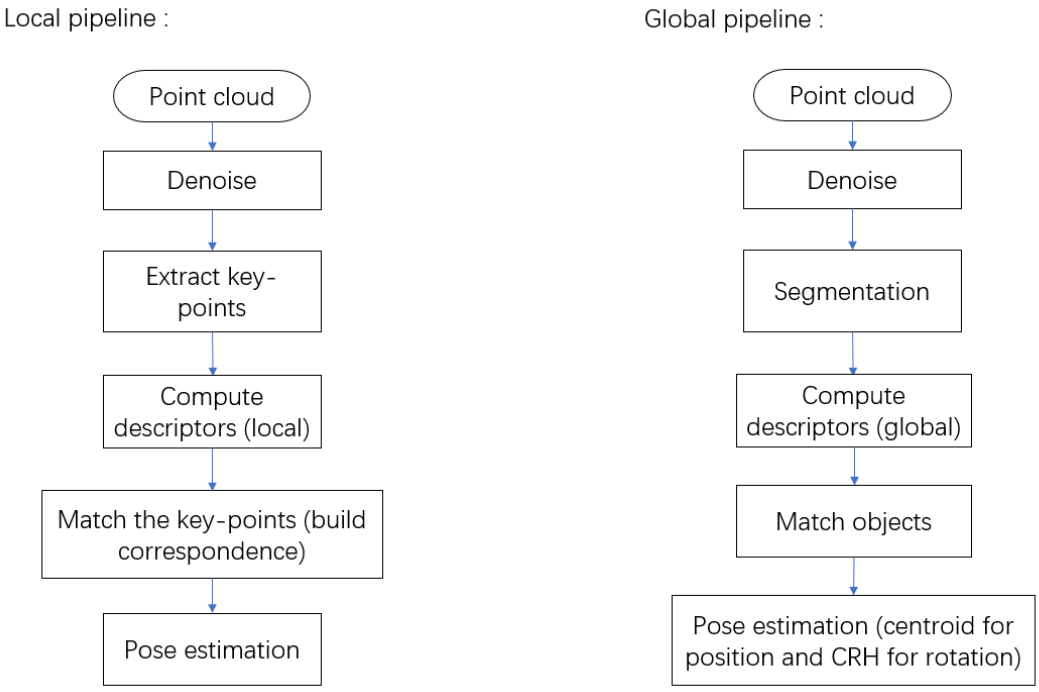


Figure 1. implementation flow charts of 2 pipelines

The main difference between them is the way they describe the point cloud, as a result, there are local descriptors and global descriptors. Local descriptors are computed for individual points that we give as input. They have no notion of what an object is, they just describe how the local geometry is around that point. Local descriptors are used for object recognition and registration. Global descriptors encode object geometry. They are not computed for individual points, but for a whole cluster that represents an object. Global descriptors are used for object recognition and classification, geometric analysis (object type, shape...), and pose estimation. Considering the scenarios in SLAM, for simplicity, we choose local pipelines (shown in Figure 2). Now we explain how we register point cloud step by step.

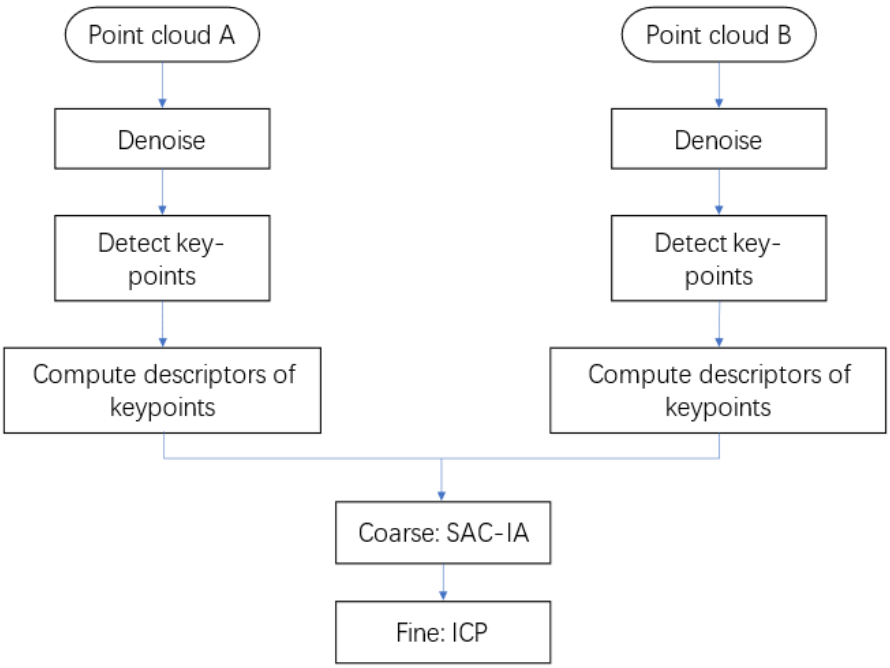


Figure 2. Local pipelines we implement

3.1 Preprocess

Before we implement those complex algorithms, it's necessary to preprocess the input point cloud data. Usually, we get the point cloud data from the real world, we need to deal with the noise. The most important part here is to denoise. We can use filters to remove outliers or set a specific remove condition (many filters you can choose, up to you), if the data in point cloud is too much, down-sampling is helpful. In order to get a more compact and clean point cloud, removing NaN (not a number) is a wise move. (You can find many other methods to preprocess a point cloud in PCL and Open3D, choose the one you need).

3.2 Keypoint detection

In this step, we choose 2 frequently-used methods: ISS (Intrinsic Shape Signatures) and SIFT (Scale-invariant feature transform). Other methods like Harris3d, NARF are not employed due to the lack of time.

3.21 SIFT3D (Scale-invariant feature transform)

This method includes a 3D interest point detector that is based on SURF, and a 3D descriptor that extends SIFT, which means a 3D version of the Hessian is used to find keypoints. [1]

3.22 ISS3D (Intrinsic Shape Signatures)

Keypoints determined by ISS are those that have large 3D point variations in their neighborhood, then implement PCA (Principal Component Analysis), the smallest eigenvalue of the covariance matrix should be large. [2]

1. compute scatter matrix in radius R, and get the eigenvalue λ_1, λ_2, λ_3 in decreasing magnitude.
2. if $\frac{\lambda_1}{\lambda_2} < r_{21}$, $\frac{\lambda_2}{\lambda_3} < r_{32}$, then we get initial keypoints, the threshold r is set by us.
3. Non-maximum suppression with λ_3, which represents the central point.

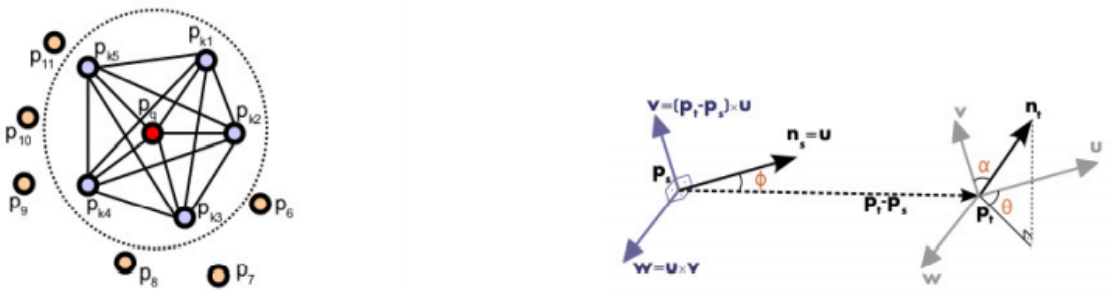
Notes: Scatter matrix is similar to covariance matrix, by definition, scatter matrix equals to covariance matrix multiply n-1, n is the sample number.
Scatter matrix is computed as: XX^T, where X is zero-centered matrix contains points.
As a result, scatter matrix is a simpler version when it comes to PCA. ISS3D ensures the detected keypoints are those different from their neighbours in three dimensions, it also requires a clean data source for the reason that it is sensitive to noise.

3.3 Computing descriptors

We implement 5 descriptors: FPFH, SI, SHOT, CSHOT, SIFT (both feature extraction and descriptor), and investigate a new data-driven descriptor: 3DMatch, now let's introduce them one by one.

3.31 PFH (Point Feature Histogram)

FPFH comes from PFH, so we start with PFH. PFH captures information of the geometry surrounding the point by analyzing the difference between the directions of the normals in the vicinity (as shown in Figure 3), [3]



(a) (b)

Figure 3: Point Feature Histogram. (a) Support region for a query point p_q (b) Darboux frame

Choose k nearest neighbors of point p, for every pair of points pi and pj in the neighborhood of p, where one point is chosen as p_s and the other as p_t, first, a Darboux frame, which is a natural moving frame, constructed on a surface is constructed at p_s as

$$u = n_s, v = u \times \frac{(p_t - p_s)}{||p_t - p_s||_2}, w = u \times v$$

Then, using the frame defined above, three angular features: α, Φ, θ, expressing the differences between normals n_t and n_s, and the distance d are computed
for each point pair in the support region.

$$\alpha = \langle v, n_t \rangle, \Phi = \frac{\langle u, p_t - p_s \rangle}{d}, \theta = arctan(\langle w, n_t \rangle, \langle u, n_t \rangle), d = ||p_t - p_s||_2$$

And the final PFH representation is created by binning these four features into a histogram with div^4 bins, where div is the number of subdivisions along each features' value range.
In this case, we need to compute $\frac{k(k+1)}{2}$ times for k neighbors.

3.32 FPFH (Fast Point Feature Histogram)

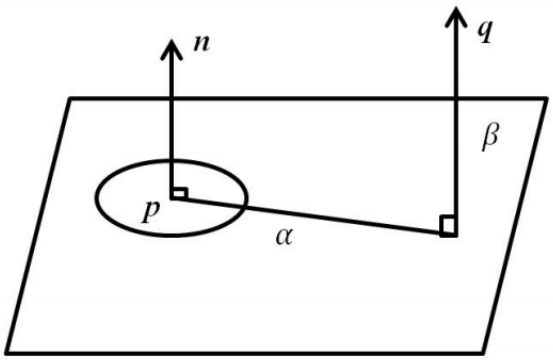
In order to reduce computational complexity, FPFH [4]
1. considers only the direct connections between the current keypoint and its neighbors, get the SPFH (Simplified Point Feature Histogram).
2. the SPFHs of a point's neighbors are "merged" with its own, weighted according to the distance.

$$FPFH(p) = SPFP(p) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{\omega_k} \cdot SPFH(p_k)$$

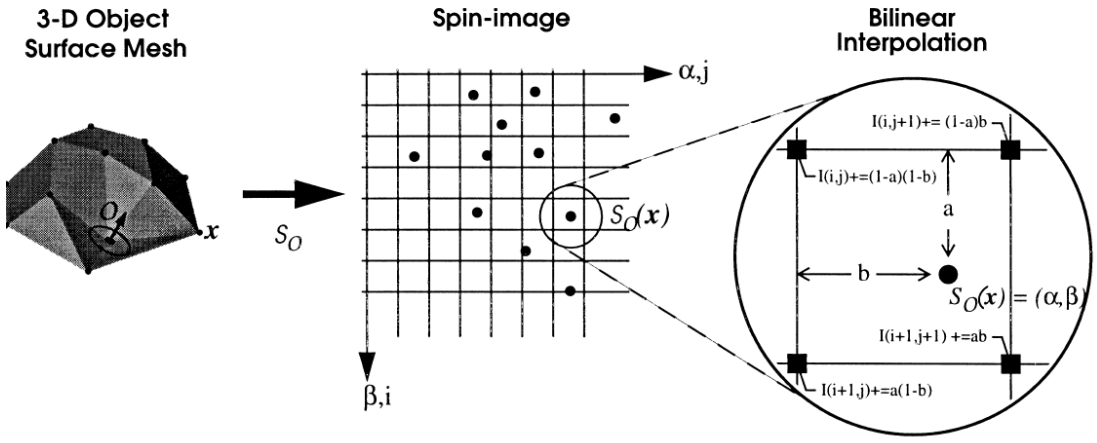
Where ω_k donates the distance between the query point and a neighbor in the support region.
In this case, we only need to compute k times for k neighbors when calculating the four values.
Besides, FPFH adds the neighbors' neighbors to represent a point, doubles the search radius of FPH.

3.33 SI (Spin Image)

The main idea of SI is to transform the point cloud in a certain area into a two-dimensional spin image, then we measure the similarity between two spin images. [5]
An oriented point is defined by a coordinate **p** and its surface normal **n**. Then the spin image attributes of each neighbor q of the feature point is defined as a pair of distances (α, β), where $\alpha = n_q \cdot (p - q)$ and $\beta = \sqrt{||p - q||^2 - \alpha^2}$ (shown in Figure 4(a)).
We generate a cylindrical coordinate system with the oriented point as the axis, set the radius R, then set the length and width of our spin image (usually the image is a square and one width parameter W, which represents the number of bins a row in the image, is enough), the resolution of the image is $\frac{R}{W}$.
Now we need to put the 3D points into our 2D image, one way is spinning the image 360° along the axis (oriented point) and let the points fall into the image bins, we can get the intensity of each bins just by calculating the number of points in the bins, in fact, in order to be more robust, a point is distributed into four pixels by bilinear interpolation, shown in Figure 4(b).



(a). Spin Image



(b). bilinear interpolation

Figure 4.

In the end, we need to measure the similarity between two spin images, according to the paper, the similarity function C is:

$$C(P,Q) = \left(atanh\left(R(P,Q)\right)\right)^2 - \lambda\left(\frac{1}{N-3}\right)$$

Where N is the number of pixels, R is:

$$R(P,Q) = \frac{N\sum p_i q_i - \sum p_i \sum q_i}{\sqrt{(N\sum {p_i}^2 - (\sum p_i)^2)(N\sum {q_i}^2 - (\sum q_i)^2)}}$$

The value range of R is [- 1,1]. The more similar the two spin images are, the closer R is to 1. When they are exactly the same, the value of R is 1.

The first part is the square of the value obtained by the inverse hyperbolic tangent function of C, and the second part is a weight λ multiplied by a smaller number. When two spin images are similar, the proportion of the second part should be smaller. When they are not close, the proportion of the second part should be larger. The function of λ is to limit spin images match when the overlap is low. In this paper, the way to select λ is to list the number of non-empty pixels in all spin images in order of size, and then take the median. This median is almost the expected value of pixel overlap. Then, considering the low overlap, we take half of the median as λ.

3.34 SHOT (Signature of Histogram of Orientation)

A combination of Signatures and Histograms. [6]

1. According to the neighborhood information of feature points, the local reference coordinate system LRF is established, and the neighborhood of feature points is divided along the radial direction (inner and outer spheres), longitude direction (time zone) and latitude direction (North and south hemispheres). Generally, there are 32 small areas, which are divided into 2 in radial direction, 8 in longitude and 2 in latitude (shown in Figure 5).
2. The distribution of cosine value of angle between normal vectors in each small area is calculated, and the normal vectors are divided into 11 bins. The length of the final shot is: 32x11 = 352.

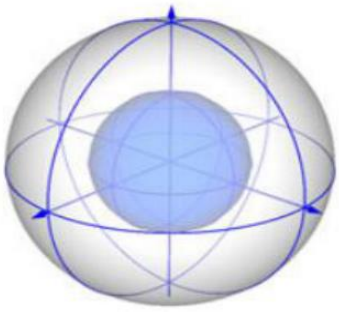


Figure 5. Signature structure for SHOT

3.35 CSHOT (Color Signature of Histogram of Orientation)

In order to improve the accuracy of feature matching, a variant that uses the texture information (CIELab color space) for matching called CSHOT is proposed. The histogram is 31 levels, so the length of color descriptor is 32x31 = 992; the SHOT descriptor with color information has 1344 dimensions. [7]

3.36 3DMatch

In recent years, deep learning plays an important role in 2D image recognition. In fact, when it comes to problems like pattern recognition, machine outperforms human with DNN (Deep Learning Network). 3DMatch is a data-driven model that learns a local volumetric patch descriptor for establishing correspondences between partial 3D data. It is a standard 3D ConvNet, inspired by AlexNet. The network learns the mapping from a volumetric 3D patch to a 512-dimensional feature representation that serves as the descriptor for that local region. Optimizing the mapping by minimizing the l_2 distance between descriptors generated from corresponding interest points (matches), and maximize the l_2 distance between descriptors generated from non-corresponding interest points (non-matches). [8]
The key idea is to amass training data by leveraging correspondence labels found in existing RGB-D scene reconstructions, which is a massive source of labeled correspondences between surfaces points of aligned frames (shown in Figure 6).

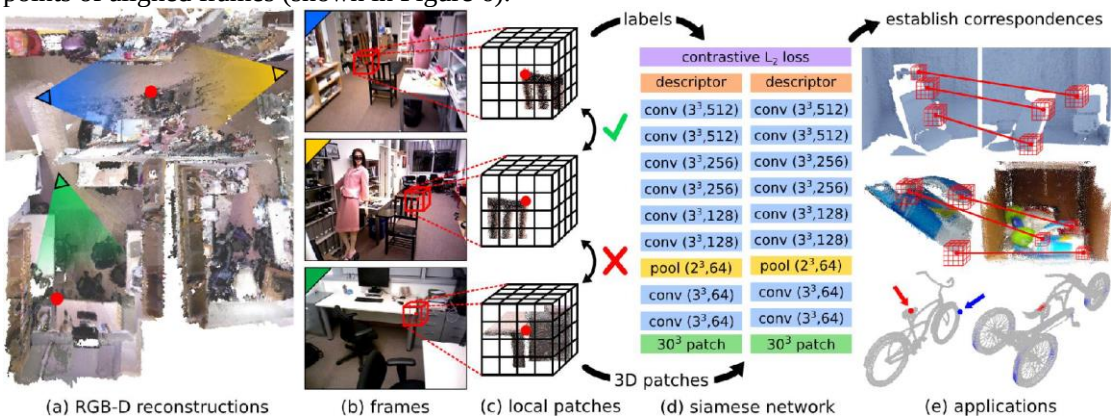


Figure 6. Learning 3DMatch from reconstructions. From existing RGB-D reconstructions (a), we extract local 3D patches and correspondence labels from scans of different views (b). We collect pairs of matching and non-matching local 3D patches and convert into a volumetric representation (c) to train a 3D ConvNet-based descriptor (d). This geometric descriptor can be used to establish correspondences for matching 3D geometry in various applications (e) such as reconstruction, model alignment, and surface correspondence.

3.4 Coarse registration

To get a good initial state for ICP, we use SAC-IA (Sample Consensus Initial Alignment) to register the point cloud before ICP. SAC-IA shares a similar idea with RANSAC (Random Sample Consensus), which is widely used in registration as well (with a known model to fit). First, we learn RANSAC.

3.41 RANSAC (Random Sample Consensus)

The RANSAC algorithm is a learning technique to estimate parameters of a model by random sampling of observed data. Given a dataset whose data elements contain both inliers and outliers (shown in Figure 7), RANSAC uses the voting scheme to find the optimal fitting result. Data

elements in the dataset are used to vote for one or multiple models. The implementation of this voting scheme is based on two assumptions: that the noisy features will not vote consistently for any single model (few outliers) and there are enough features to agree on a good model (few missing data). The RANSAC algorithm is essentially composed of two steps that are iteratively repeated:
In the first step, a sample subset containing minimal data items is randomly selected from the input dataset. A fitting model and the corresponding model parameters are computed using only the elements of this sample subset. The cardinality of the sample subset is the smallest sufficient to determine the model parameters. (find more on Wikipedia.)
In the second step, the algorithm checks which elements of the entire dataset are consistent with the model instantiated by the estimated model parameters obtained from the first step. A data element will be considered as an outlier if it does not fit the fitting model instantiated by the set of estimated model parameters within some error threshold that defines the maximum deviation attributable to the effect of noise.
The RANSAC algorithm will iteratively repeat the above two steps until the obtained consensus set (set of inliers obtained for the fitting model) in certain iteration has enough inliers.

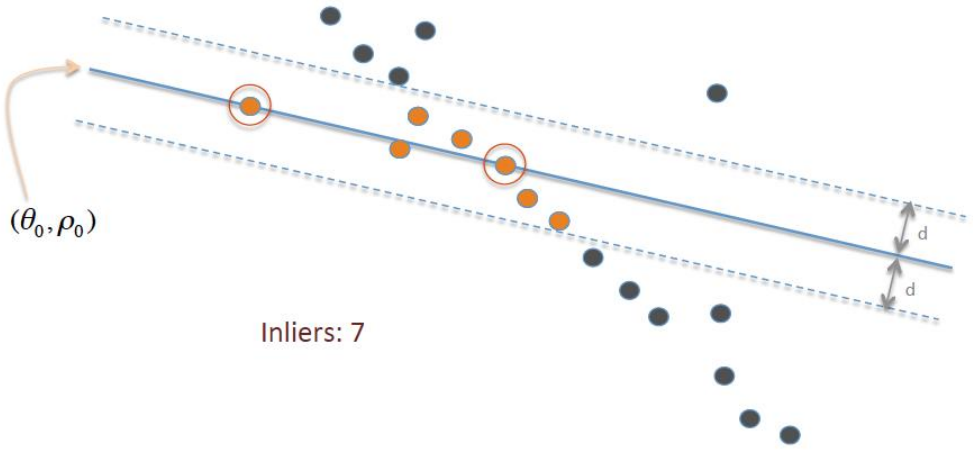


Figure 7. RANSAC: Inliers and Outliers.

3.42 SAC-IA (Sample Consensus Initial Alignment)

In order to get the initial transformation, inspired by RANSAC, SAC-IA estimates the transformation in a fast way. [9]
Instead of greedy search with a high computational complexity, SAC-IA collects a large number of samples from candidate correspondence, finds a good transformation quickly by looking at a large number of correspondences. This approach maintains the same geometric relations of the correspondences without having to try all combinations of a limited set of correspondences, by employing the following scheme:
1. pick n (n>=3) points in the source point cloud with different features (set a minimum distance between each two points).
2. find the points in target point cloud with similar feature to the point picked in step 1, and choose one point in random.
3. solve the rotation and displacement by SVD, calculate the loss function, iterate to get the best transformation.

3.5 Fine registration

Finally, we use ICP for fine registration. ICP is a local register method and there are global methods (e. g. 4PCS). There are many variations of ICP, here we take the standard point-to-point ICP and point-to-plane ICP to show the basic idea of ICP in general.

3.51 ICP (Iterative Closest Point)

After SAC-IA, we get the approximate transformation matrix which gives ICP a good initial state. ICP find the correspondence between points in a greedy way (closest point), as a result, when the correspondence is unknow and transformation between the source point cloud and the target point cloud is significant (far distance), ICP lost accuracy. Then we start ICP algorithm (shown in Figure 6). [10]
1. compute correspondences between the two scans
2. compute a transformation which minimizes distance between corresponding points.
Iteratively repeating these two steps typically results in convergence to the desired transformation. In practice, some points will not have any correspondence in the target point cloud, based on that fact, the algorithm contains a maximum matching threshold d_{max}. In most implementations of ICP, the choice of d_{max} represents a trade-off between convergence and accuracy. A low value results in bad convergence (the algorithm becomes "short sighted"); a large value causes incorrect correspondences to pull the final alignment away from the correct value.

input : Two pointclouds: $A = \{a_i\}, B = \{b_i\}$
An initial transformation: T_0
output: The correct transformation, T, which aligns A and B
$T \leftarrow T_0$;
while *not converged* **do**
  for $i \leftarrow 1$ **to** N **do**
    $m_i \leftarrow \texttt{FindClosestPointInA}(T \cdot b_i)$;
    if $||m_i - T \cdot b_i|| \leq d_{max}$ **then**
      $w_i \leftarrow 1$;
    else
      $w_i \leftarrow 0$;
    end
  end
  $T \leftarrow \underset{T}{\operatorname{argmin}} \{\sum_i w_i ||T \cdot b_i - m_i||^2\}$;
end

Figure 6. Standard ICP

There are two ways to find the solution in ICP: SVD and non-linear optimization.

1. SVD-based method:
Given: $P = \{p_i\},\ Q = \{q_i\}$
Optimization target: $\Sigma_i = \frac{1}{N}\sum_{i \in Q, j \in P} \quad (q_i - u_Q)(p - u_p)^T$
Algorithm
For i in range(iterations):

- Centering datasets

$$u_Q = \frac{1}{|C|}\sum_{(i,j)\in C} \quad q_i \qquad u_p = \frac{1}{|C|}\sum_{(i,j)\in C} \quad p_j$$
$$Q' = \{q_i - u_Q\} = \{q_i'\} \quad P' = \{p_j - u_p\} = \{p_j'\}$$

- Find correlative pairs for each $P = \{p_i\}\ in\ Q = \{q_i\}$ (nearest neighbors)
- Calculate cross-covariance along all pairs with kernel

$$Weight = kernel(p_j', q_i')$$

$$\Sigma_i = \frac{1}{N}\sum_{i\in Q', j\in P'} (q_i')(p_j')^T = \left[cov(p_x,\, q_x)\; cov(p_x,\, q_y)\; cov(p_y,\, q_x)\; cov(p_y,\, q_y)\right]$$

- Do SVD decomposition and get Roto.&Trans. Matrix

$$U,\, S,\, V^T = SVD(\Sigma_i)$$

$$R_i = UV^T \qquad t_i = u_Q - R_i u_p$$

- Use Roto.&Trans. transform P to match Q

$$P_{mi} = \{R_i p_i + t\} = \{p_{mi,j}\}$$

2. Non-linear Least-squares based method:

Given: $P = \{p_i\},\ Q = \{q_i\}$

Optimization target: $E = \sum_i \left[R_\theta p_i + t - q_j\right]^2 \to min$

Algorithm

Initialize rotation matrix R_θ and its derivative of rotation matrix $\dot{R}_\theta$

Initialize rot-translation pose $x = \{t, \theta_{rot}\}^T = \{x_t, y_t, \theta_{rot}\}^T$

For i in range(iterations):

- Find correlative pairs for each $P = \{p_i\}\ in\ Q = \{q_i\}$ (nearest neighbors)

  For m in ($Q = \{q_j\},\ P = \{p_i\}$):

  - Calc single pair error, update Loss function and Calc it's weight

$$e_{(i,j)}(x) = h_i(x) - q_j = R_\theta p_i + t - q_j$$

$$Weight = kernel(p_j',\, q_i')$$

  - Calc Jacobian matrix

$$T = \frac{\partial e_{i,j}(x)}{\partial x} = \frac{\partial h_i(x)}{\partial x} = \left(\frac{\partial h_i(x)}{\partial x}, \frac{\partial h_i(x)}{\partial y}, \frac{\partial h_i(x)}{\partial \theta}\right) = \left(I, \dot{R}_\theta p_i\right)$$

$$= \left[1\ 0\ \ -sin\, p_i^x sin\,\theta\ \ -cos\, p_i^y cos\,\theta\ \ 0\ 1\ p_i^x\ cos\ cos\,\theta\ \ -sin\, p_i^y sin\,\theta\ \right]$$

  - Calc Hessian matrix and update it

$$H \leftarrow H + H_m = H + T_{(i,j)}^T T_{(i,j)}$$

  - Calc Gradient of single pair error and update Gradient of loss function

$$g \leftarrow g + g'_{(i,j)} = g + T_{(i,j)}^T e_{(i,j)}$$

- Do Least-square solver to get the increment $\Delta x = LeastSquare(H, -g)$
- Update pose: $x \leftarrow x + \Delta x = x + \{\Delta x_t, \Delta y_t, \Delta\theta_{rot}\}^T$
- Use Roto.&Trans. transform P to match Q: $P_{mi} = \{R_i p_i + t\} = \{p_{mi,j}\}$

3.52 Point to plane ICP

Instead of minimizing $\sum \quad ||T \cdot b_i - m_i||^2$, the point-to-plane algorithm minimizes error along the surface normal (the projection of $(T \cdot b_i - m_i)$ onto the sub-space spanned by the surface normal). [11]

4. Comparison and analysis of our pipelines

To compare these different feature descriptors, we uniformly use SIFT to detect the key points from the source and target point clouds.

Then, we computed different feature descriptors (SI, FPFH, SHOT, CSHOT, SIFT) for the same detected key points of each pair of source and target point clouds.

Finally, we uniformly use RANSAC for coarse registration and ICP for fine registration for each pair of source and target with different computed feature descriptors.

Implementation of pipelines:

(1) SIFT3D + RANSAC + ICP
(2) SIFT3D + Spin Image + RANSAC + ICP
(3) SIFT3D + FPFH + RANSAC + ICP
(4) SIFT3D + SHOT + RANSAC + ICP
(5) SIFT3D + CSHOT + RANSAC + ICP

Dependencies:

- Point Cloud Library
- C++
- Visual Studio 2019
- Unity

4.1 Final delivery

(1) v7.3_ReleasedTookKit_MiroRGB3DCamera.zip

- Released_MiroRGB3DCameraV7.3: final version camera in Unity
- Setup Manual_MiroRGB3DCameraV7.3: Unity setup tutorial
- pcdVisualizer.py: Python visualizer
- pcd2fig: MATLAB visualizer
- Apartment: demo prefab
- Japanese Torii: demo prefab
- Miaomiao: demo prefab

(2) Pipelines+Manual-PointCloud_Registration.zip

- PCL_Setup Manual_PCL1.11.1+Windows+VS2019_x64: PCL setup tutorial
- f_pipeline_setup.h: header file that includes all parameters for pipelines
- fpip_ISS3D+CSHOT+RANSAC+ICP.cpp
- fpip_SIFT+FPFH+RANSAC+ICP.cpp
- fpip_SIFT+RANSAC+ICP.cpp
- fpip_SIFT+SHOT+RANSAC+ICP.cpp
- fpip_SIFT+SI+RANSAC+ICP.cpp

4.2 Performance

The number of registered point pairs between source and target point clouds can reflect the registration performance of these descriptors.
Figure 1 presents the number of registered points of each pipeline using different feature descriptors with increasing the noises of the point cloud from 0 (0%) to 4.0(100%) in our experiment. It can be clearly seen that first, SI descriptor has a relatively better performance with increasing noises. In contrast, SIFT descriptor that is the key points themselves detected by SIFT has the lowest performance. Moreover, when other geometry-based descriptors have lower performance, CSHOT descriptor that use color information has a better performance than others.

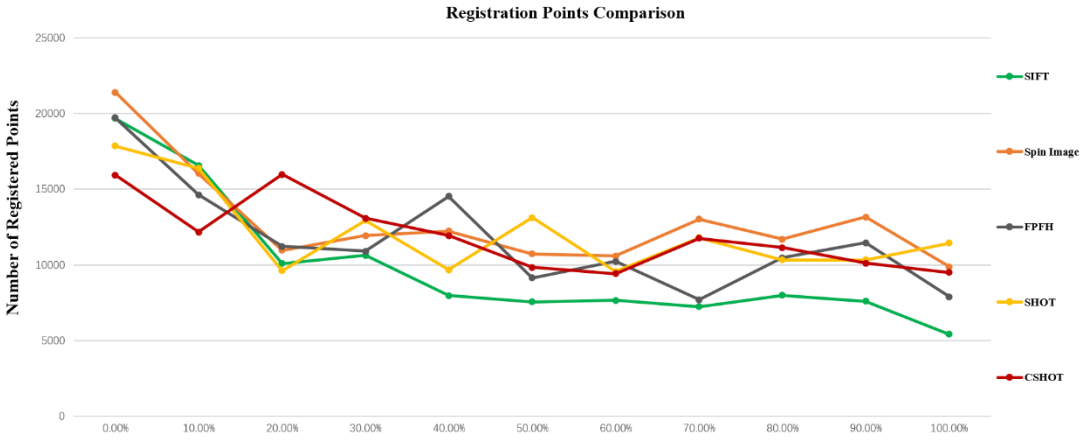


Fig1: Number of registered point pairs using different feature descriptors with increasing noises

4.3 Efficiency

During the experiments, we also recorded the computational cost of these descriptors. Figure 2 presents the running time of calculating descriptors for source and target key points with increasing noises of the point cloud from 0 (0%) to 4.0(100%). We can clearly see that SI descriptor is the most efficient descriptor. In contrast, SHOT and CSHOT become the most computationally expensive descriptors under current radius. However, if we increase the searching radius, the efficiency of these descriptors will decrease by different degree.

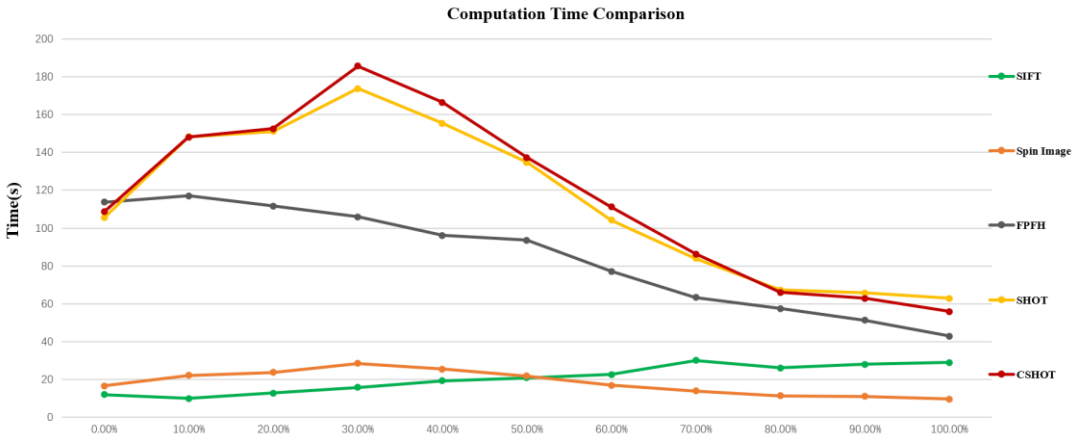


Fig2: Computation time required to generate the descriptor for the key points using different feature descriptors with increasing noises

4.4 Conclusion and future work

Overall, SI performs the best among these descriptors. SI has a good balance between performance and efficiency. Particularly, SI is suitable for real-time applications.

However, the performance and efficiency of these descriptors highly depends on the parameters setting. For instance, during the experiments, we found that the efficiency of FPFH decreased dramatically with increasing searching radius. If we set a larger searching radius for FPFH descriptor, it will become very slow, while other descriptors are not very sensitive to the increasing searching radius. In order to obtain a clearer and comprehensive understanding of these feature descriptors, we should carry out more experiments in tuning the parameters setting in the future.

Besides, in the current experiment, we used sparse point cloud data (object only), which is simple and has many NaN values. In the future, we will use the dense point cloud data like in the real world, having complex scene, as the experiment data.

The last but most interesting thing, although CSHOT is the only descriptor that takes into account the color information, it didn't perform better in the experiment. However, in some cases, CSHOT outperformed other descriptors. Therefore, in future work, we need to make more comparisons under different conditions and scenarios to find the most suitable application for CSHOT.

Appendix A

Miro RGB-D Camera v7.3 Setup

Build Platform

- Game Engine: Unity 2019.4.11f1 LTS
- IDE: Visual Studio 2019

Package Commissioning Steps

Opening Unity and enabling ‘unsafe programming

To use the Miro RGB-D Camera v7.3, you need to enable “Allow ‘unsafe’ code” in Unity, because we need to do casting in C#.

(1) Opening Unity and creating a new project(or open your project)
(2) Enabling “Allow ‘unsafe’ code” model by “Edit Project Settings Player tick “Allow ‘unsafe’ code”

Otherwise, there will be a warning in Unity after you import the Miro RGB-D Camera v7.3 package like shown below:

Importing packages

After downloading the released toolkit, you might have five tools:

- Released_MiroRGB3DCameraV7.3.unitypackage: Miro RGB-D Camera v7.3
- Apartment.unitypackage: example prefab
- JapaneseTorii. unitypackage: example prefab
- Miaomiao. unitypackage: example prefab
- pcd2fig.m: MATLAB PCD file visualizer
- pcdVisualizer.py: Python PCD file visualizer

In order to use these tools, steps as below:

(1) Uncompress “ReleasedToolKit_MiroRGB3DCameraV7.3”
(2) Import three unity package by “Assets Import Package Custom Package …”
    - “… Released_MiroRGB3DCameraV7.3.unitypackage”
    - “… Miaomiao.unitypackage”
    - “… JapaneseTorii. unitypackage”
    - “… Apartment.unitypackage”

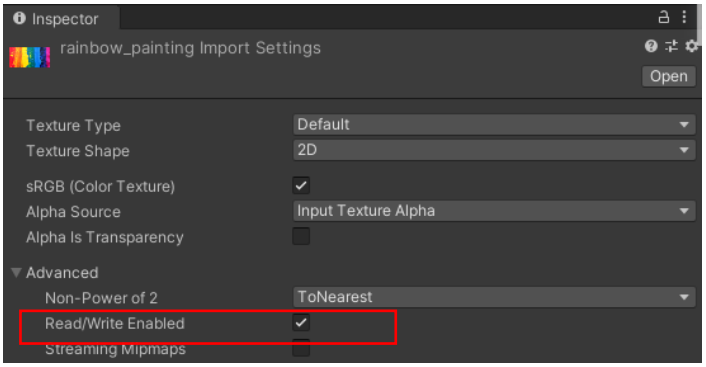
(3) Drag “RGB3DCameraV6” prefab inside the scene
(4) Drag “Torii” prefab inside the scene
(5) Drag “cat” prefab inside the scene
(6) Save the scene

How to setup your 3D object

To capture the XYZ information and color information of the texture of your 3D object in Unity, you should setup your object properly:

(1) Adding “Mesh Collider” component to your 3D object and assign the corresponding “Mesh” to it, like:
- Select “cat/cat” in the scene
- Add Mesh Collider in Inspector
- Find the mesh for the ‘cat’ object in “Project□RGB_Test_Miaomiao□Models□cat□cat”, and drag it into the ‘cat’ objects in the scene □ Inspector □ Mesh Collider □ Mesh

(2) Adding “texture” component to your 3D object and enabling the texture can be read/write by ticking the “Read/Write Enable” in “Import Settings □ Advanced □ Read/Write Enable” of your texture. This is a compulsory step if you want to capture the color(RGB) information from you object.



How to capture and save pointcloud as ‘.pcd’ file

To simply capture point cloud of your object with XYZ information and color information(if you want), you can refer the following steps:

(1) Run the game
(2) Select ‘RGB3DCameraV7’ in the ‘Hierarchy’

(3) Tick the RGB Enabler(if you want to add RGB info in your point cloud) on the ‘Inspector’

(4) Tune the ‘noise setting’(if you want to add different noises in you point cloud)) on the ‘Inspector’
(5) Click on ‘Capture Points’ on the ‘Inspector’
(6) Click on ‘Save Last Captured Points’ on the ‘Inspector’ to save the point cloud into ‘.pcd’ format

Besides, you definitely can design your own Unity project and play with the camera automatically by directly calling the functions or setting parameters in your code.

How to visualizer point cloud

In the toolkit, we also provide you with two simple point cloud visualizer.

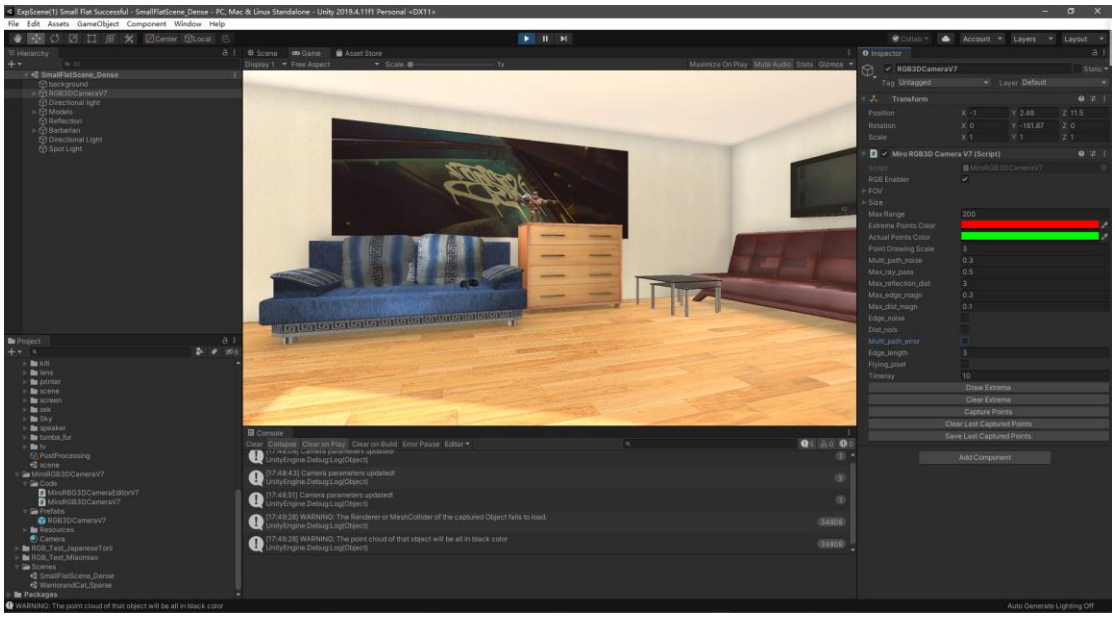
Option 1: Python + open3d library(http://www.open3d.org/docs/release/introduction.html)

- Open the Python file ‘pcdVisualizer.py’
- Change the path of the ‘.pcd’ file with yours
- Save the changed Python file
- Run the Python file ‘pcdVisualizer.py’ in the way you like
- P.S.: With this visualizer, you can also calculate the normal of your point cloud surface and visualize the normals

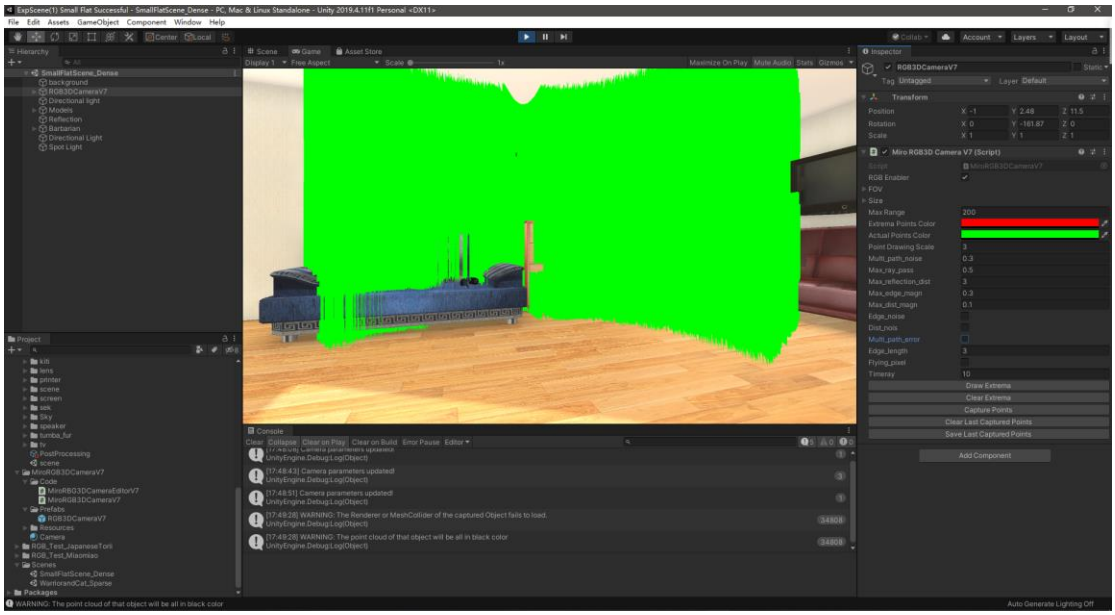
Option 2: Matlab

- Open the Matlab file ‘pcd2fig.m’
- Change the path of the ‘.pcd’ file with yours
- Run the script

Example 1

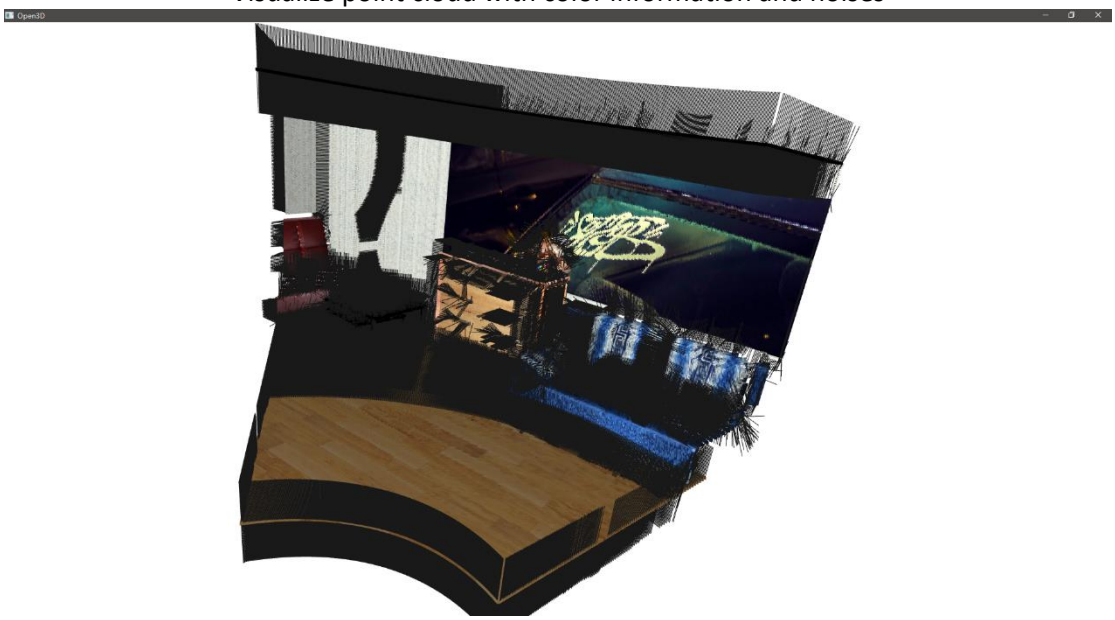


Unity interface



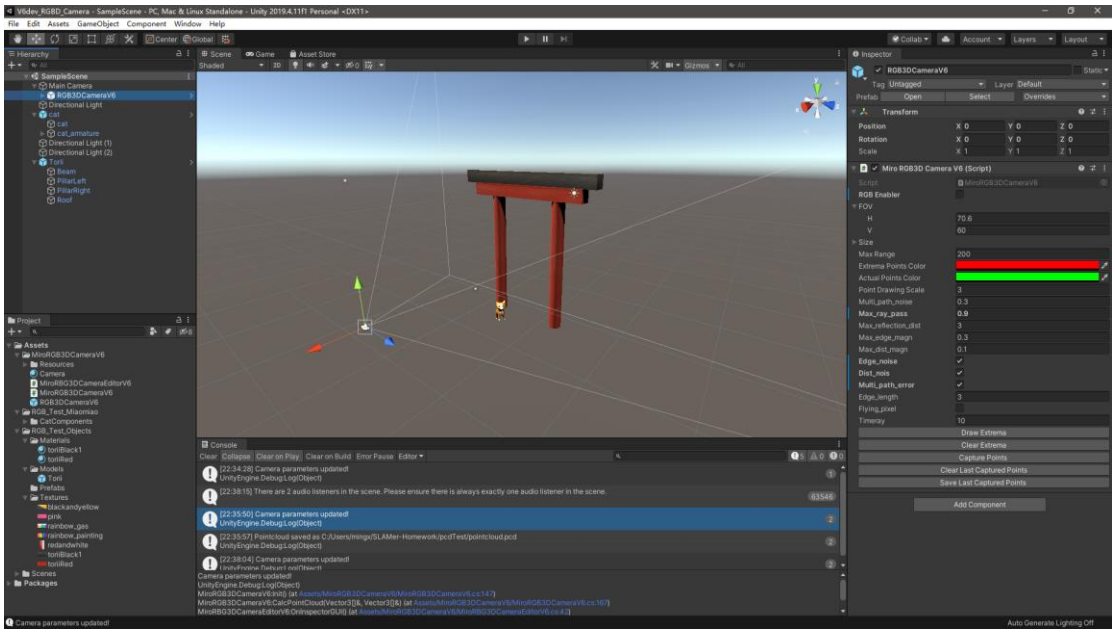
Draw point cloud in Unity

Visualize point cloud with color information and noises

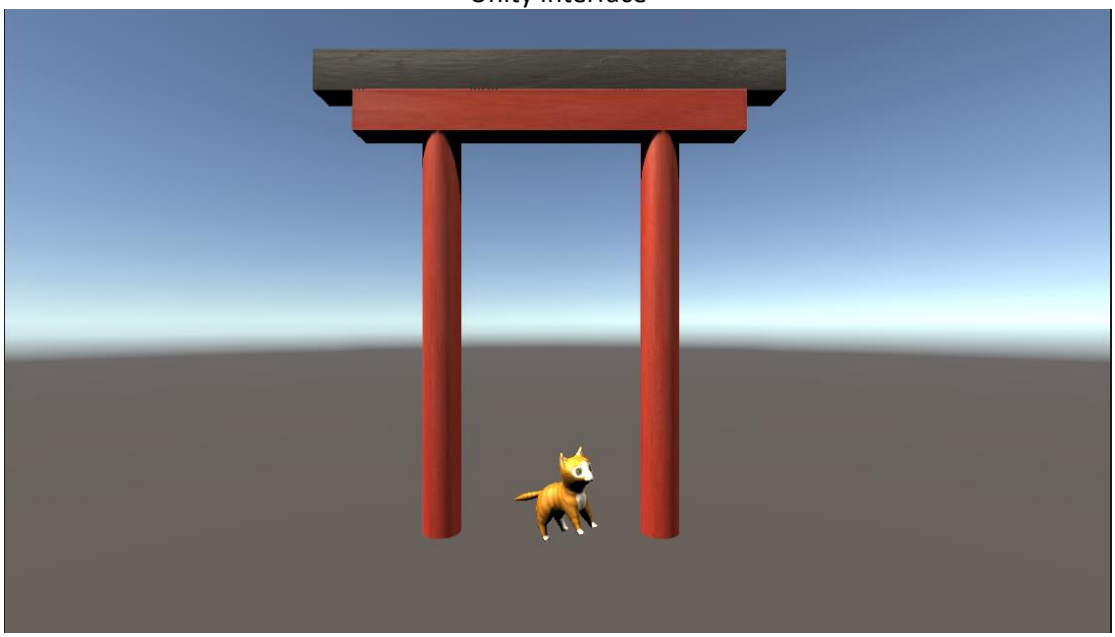


Visualize point cloud normals

Example 2



Unity interface



Camera viewpoint

Draw extrema and point cloud in Unity

Visualize point cloud with XYZ information only and noises

Visualize point cloud with color information and noises

Visualize point cloud with color information, no noises

Appendix B

Point Cloud Library Setup

Introduction

The Point Cloud Library (PCL) is a standalone, large scale, open project for 2D/3D image and point cloud processing. PCL is released under the terms of the BSD license, and thus free for commercial and research use.

Continuous integration

Build Platform	Status
Ubuntu	Ubuntu 18.04 GCC succeeded Ubuntu 20.04 Clang succeeded Ubuntu 20.10 GCC failed
Windows	Windows VS2017 x86 succeeded Windows VS2017 x64 succeeded
macOS	macOS Mojave 10.14 succeeded macOS Catalina 10.15 succeeded

Compiling

Linux: https://pcl-tutorials.readthedocs.io/en/latest/compiling_pcl_posix.html
Mac OS X: https://pcl-tutorials.readthedocs.io/en/latest/compiling_pcl_posix.html
Microsoft Windows: https://pcl-tutorials.readthedocs.io/en/latest/compiling_pcl_windows.html

PCL setup example

Below is a detailed tutorial about how to setup PCL 1.11.1 and its environments on Windows with VS2019 X64.

- **Build platform**: Windows VS2019 X64

- **PCL version**: PCL 1.11.1

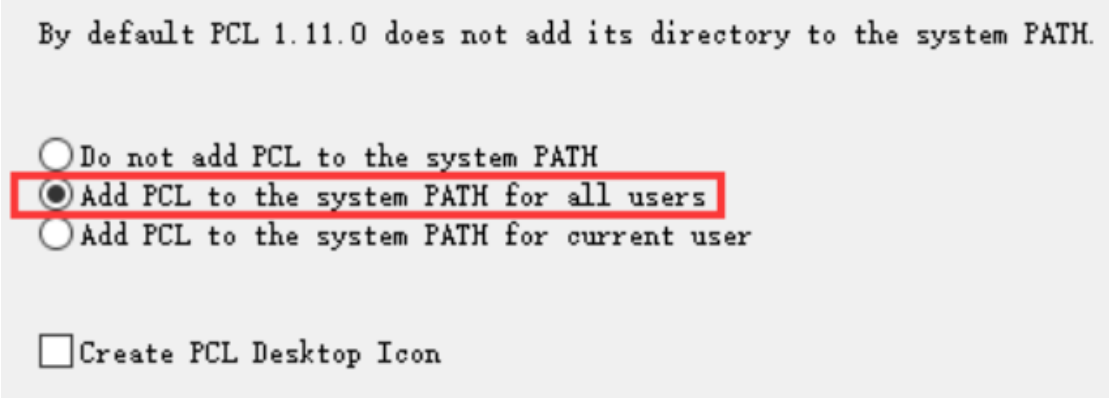
Downloading PCL

Accessing PCL from Github releases:
https://github.com/PointCloudLibrary/pcl/releases/tag/pcl-1.11.1
Then, downloading "PCL-1.11.1-AllInOne-msvc2019-win64.exe"

Installing PCL

Double click "PCL-1.11.1-AllInOne-msvc2019-win64.exe" to start install.

(1) Click "next" until below figure, and then choose "Add PCL to the system PATH for all users"



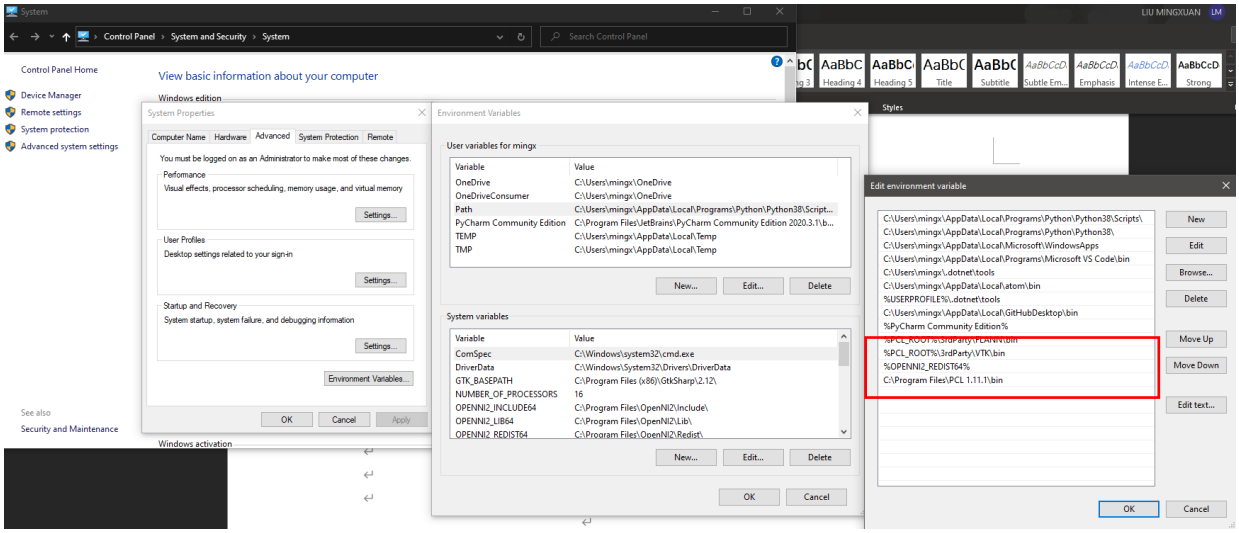
(2) Choose target file path(recommend to choose "**C:\Program Files\PCL 1.11.1**", since you can fully refer this tutorial)

(3) Click "next" until installation complete

Setup system environments

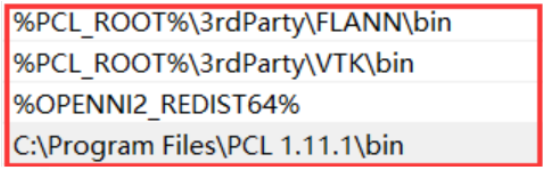
(1) Copy and paste "**pcl-1.11.1-pdb-msvc2019-win64**" file that you extracted from the PCL 1.11.1 releases file to "C:\PCL\1.11.1\bin" folder.

(2) Setup system environment variables: right click 'This PC' → 'Properties' → 'Advanced system settings' → 'Advanced' → 'Environment Variables' → 'User variables for name' → 'Path' → 'Edit'



(3) Add the four below

1	%PCL_ROOT%\3rdParty\FLANN\bin
2	%PCL_ROOT%\3rdParty\VTK\bin
3	%OPENNI2_REDIST64%
4	C:\Program Files\PCL 1.11.1\bin

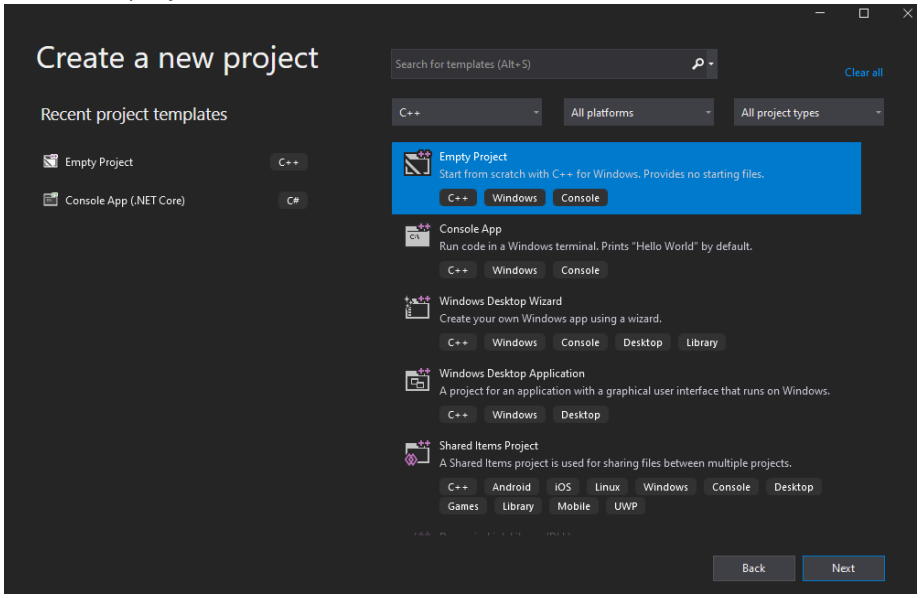


(4) Reboot your PC

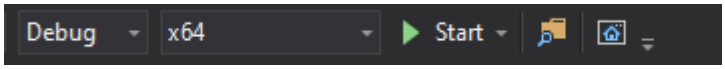
Setup Visual Studio 2019 environments

(1) Start your Visual Studio 2019

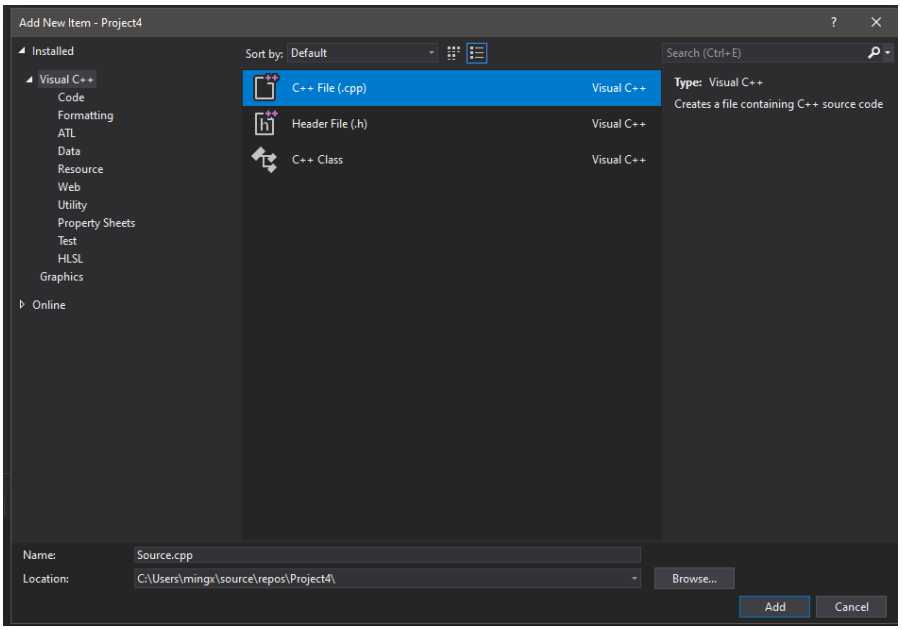
(2) Create a new project



(3) Change solution configuration to ‘Debug’ + ‘x64’



(4) Create a new C++ source file



(5) Right click your 'Project name', opening the 'properties'

(6) Click 'Configuration Properties' → 'Debugging' → 'Environment' → 'Edit'

(7) Adding the four below in your 'Environment'

1	**PATH=C:\Program Files\PCL 1.11.1\\bin;**
2	**C:\Program Files\PCL 1.11.1\\3rdParty\FLANN\bin;**
3	**C:\Program Files\PCL 1.11.1\\3rdParty\VTK\bin;**
4	**C:\Program Files\OpenNI2\Tools**

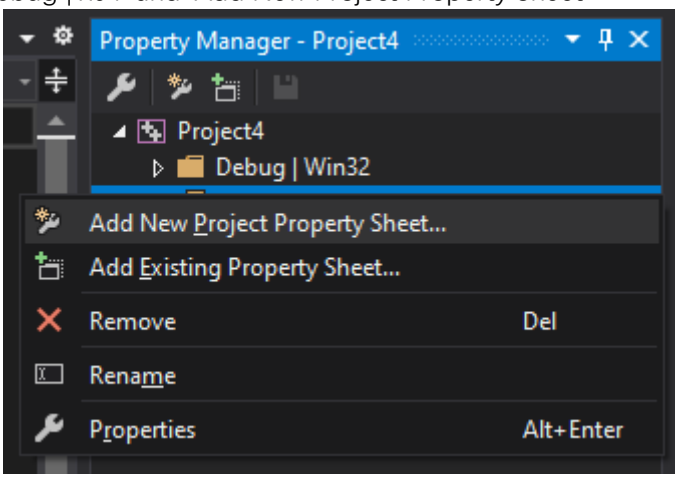
(8) Click 'C/C++' → 'Language' → 'Conformance mode' → choose '**NO**'

(9) Click 'C/C++' → 'General' → 'SDL checks' → choose '**NO**'

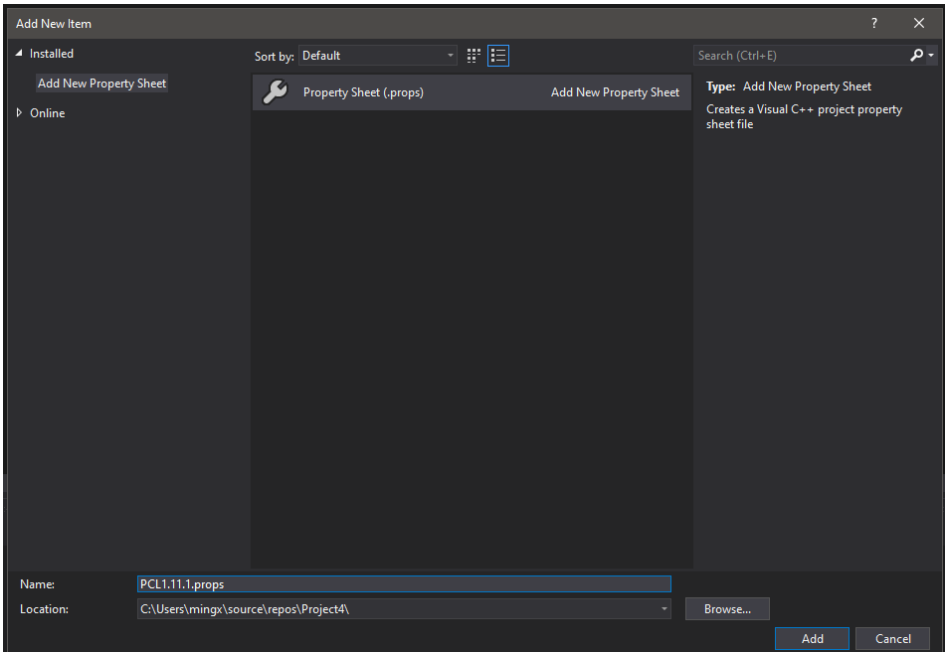
(10) Click 'Apply' and 'OK'

(11) Click 'Property Manager' section

(12) Right click 'Debug | x64' and 'Add New Project Property Sheet'



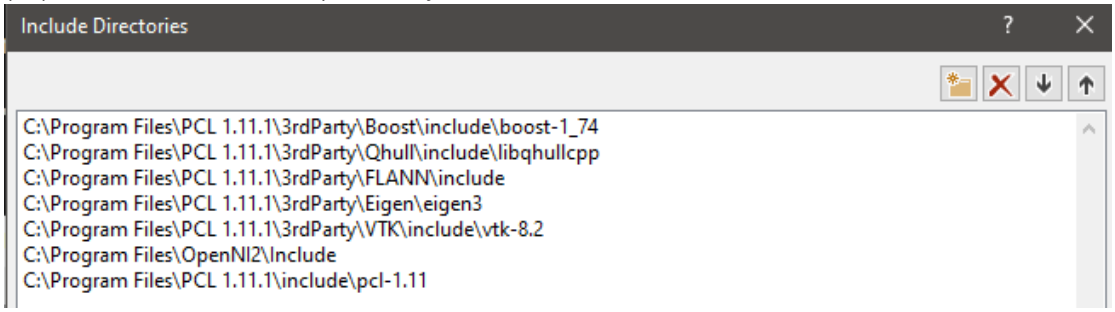
(13) Name your property sheet as 'PCL1.11.1.props' for future use and then 'Add'



(14) Double click the new property sheet you just added

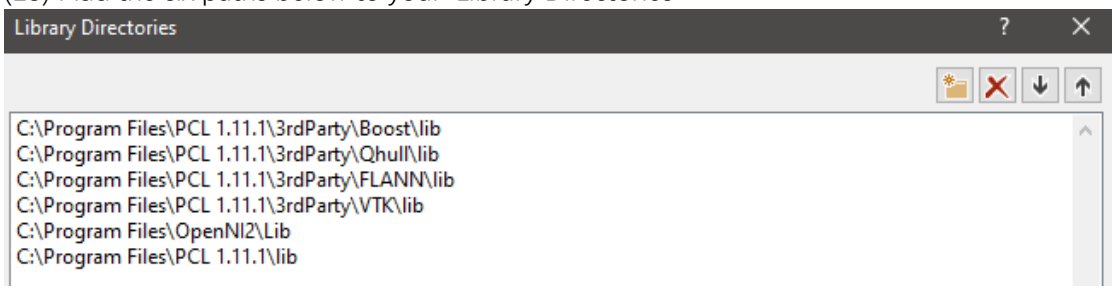
(15) Click ‘VC++ Directories’ → ‘Include Directories’ → ‘Edit’

(16) Add the seven below paths to your ‘Include Directories’



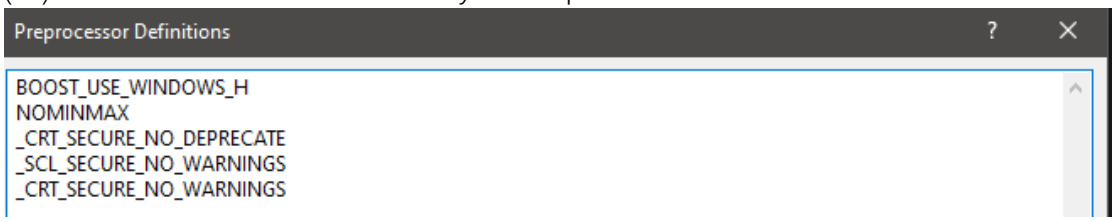
(17) Click ‘VC++ Directories’ → ‘Library Directories’ → ‘Edit’

(18) Add the six paths below to your ‘Library Directories’

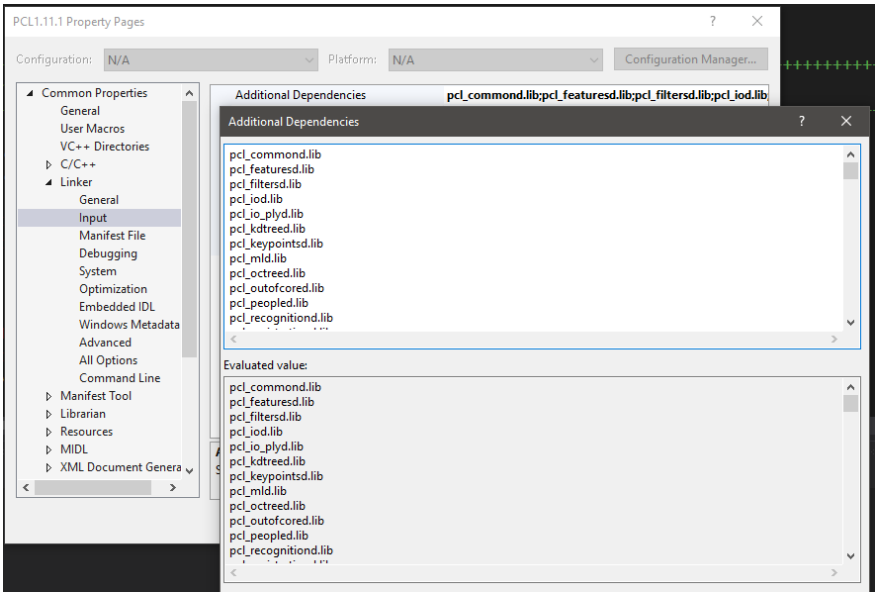


(19) Click ‘C/C++’ → ‘Preprocessor’ → ‘Preprocessor Definitions’ → ‘Edit’

(20) Add the five definitions below to your ‘Preprocessor Definitions’



(21) Click ‘Linker’ → ‘Input’ → ‘Additional Dependencies’ → ‘Edit’



(22) Add the dependencies below to your 'Additional Dependencies' (you can simply copy and paste below dependencies, but please double check the version of all dependencies)

```
pcl_commond.lib;
pcl_featuresd.lib;
pcl_filtersd.lib;
pcl_iod.lib;
pcl_io_plyd.lib;
pcl_kdtreed.lib;
pcl_keypointsd.lib;
pcl_mld.lib;
pcl_octreed.lib;
pcl_outofcored.lib;
pcl_peopled.lib;
pcl_recognitiond.lib;
pcl_registrationd.lib;
pcl_sample_consensusd.lib;
pcl_searchd.lib;
pcl_segmentationd.lib;
pcl_stereod.lib;
pcl_surfaced.lib;
pcl_trackingd.lib;
pcl_visualizationd.lib;
vtkChartsCore-8.2-gd.lib;
vtkCommonColor-8.2-gd.lib;
vtkCommonComputationalGeometry-8.2-gd.lib;
vtkCommonCore-8.2-gd.lib;
vtkCommonDataModel-8.2-gd.lib;
vtkCommonExecutionModel-8.2-gd.lib;
vtkCommonMath-8.2-gd.lib;
vtkCommonMisc-8.2-gd.lib;
vtkCommonSystem-8.2-gd.lib;
vtkCommonTransforms-8.2-gd.lib;
vtkDICOMParser-8.2-gd.lib;
vtkDomainsChemistry-8.2-gd.lib;
```

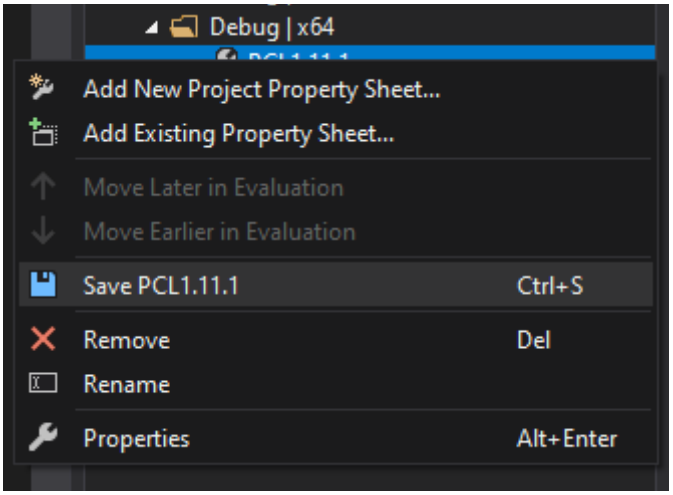
vtkDomainsChemistryOpenGL2-8.2-gd.lib;
vtkdoubleconversion-8.2-gd.lib;
vtkexodusII-8.2-gd.lib;
vtkexpat-8.2-gd.lib;
vtkFiltersAMR-8.2-gd.lib;
vtkFiltersCore-8.2-gd.lib;
vtkFiltersExtraction-8.2-gd.lib;
vtkFiltersFlowPaths-8.2-gd.lib;
vtkFiltersGeneral-8.2-gd.lib;
vtkFiltersGeneric-8.2-gd.lib;
vtkFiltersGeometry-8.2-gd.lib;
vtkFiltersHybrid-8.2-gd.lib;
vtkFiltersHyperTree-8.2-gd.lib;
vtkFiltersImaging-8.2-gd.lib;
vtkFiltersModeling-8.2-gd.lib;
vtkFiltersParallel-8.2-gd.lib;
vtkFiltersParallelImaging-8.2-gd.lib;
vtkFiltersPoints-8.2-gd.lib;vtkFiltersProgrammable-8.2-gd.lib;
vtkFiltersSelection-8.2-gd.lib;
vtkFiltersSMP-8.2-gd.lib;
vtkFiltersSources-8.2-gd.lib;
vtkFiltersStatistics-8.2-gd.lib;
vtkFiltersTexture-8.2-gd.lib;
vtkFiltersTopology-8.2-gd.lib;
vtkFiltersVerdict-8.2-gd.lib;
vtkfreetype-8.2-gd.lib;
vtkGeovisCore-8.2-gd.lib;
vtkgl2ps-8.2-gd.lib;
vtkglew-8.2-gd.lib;
vtkGUISupportMFC-8.2-gd.lib;
vtkhdf5-8.2-gd.lib;
vtkhdf5_hl-8.2-gd.lib;
vtkImagingColor-8.2-gd.lib;
vtkImagingCore-8.2-gd.lib;
vtkImagingFourier-8.2-gd.lib;
vtkImagingGeneral-8.2-gd.lib;
vtkImagingHybrid-8.2-gd.lib;
vtkImagingMath-8.2-gd.lib;
vtkImagingMorphological-8.2-gd.lib;
vtkImagingSources-8.2-gd.lib;
vtkImagingStatistics-8.2-gd.lib;
vtkImagingStencil-8.2-gd.lib;
vtkInfovisCore-8.2-gd.lib;
vtkInfovisLayout-8.2-gd.lib;
vtkInteractionImage-8.2-gd.lib;
vtkInteractionStyle-8.2-gd.lib;
vtkInteractionWidgets-8.2-gd.lib;
vtkIOAMR-8.2-gd.lib;
vtkIOAsynchronous-8.2-gd.lib;
vtkIOCityGML-8.2-gd.lib;
vtkIOCore-8.2-gd.lib;

vtkIOEnSight-8.2-gd.lib;
vtkIOExodus-8.2-gd.lib;
vtkIOExport-8.2-gd.lib;
vtkIOExportOpenGL2-8.2-gd.lib;
vtkIOExportPDF-8.2-gd.lib;
vtkIOGeometry-8.2-gd.lib;
vtkIOImage-8.2-gd.lib;
vtkIOImport-8.2-gd.lib;
vtkIOInfovis-8.2-gd.lib;
vtkIOLegacy-8.2-gd.lib;
vtkIOLSDyna-8.2-gd.lib;
vtkIOMINC-8.2-gd.lib;
vtkIOMovie-8.2-gd.lib;
vtkIONetCDF-8.2-gd.lib;
vtkIOParallel-8.2-gd.lib;
vtkIOParallelXML-8.2-gd.lib;
vtkIOPLY-8.2-gd.lib;
vtkIOSegY-8.2-gd.lib;
vtkIOSQL-8.2-gd.lib;
vtkIOTecplotTable-8.2-gd.lib;
vtkIOVeraOut-8.2-gd.lib;
vtkIOVideo-8.2-gd.lib;
vtkIOXML-8.2-gd.lib;
vtkIOXMLParser-8.2-gd.lib;
vtkjpeg-8.2-gd.lib;
vtkjsoncpp-8.2-gd.lib;
vtklibharu-8.2-gd.lib;
vtklibxml2-8.2-gd.lib;
vtklz4-8.2-gd.lib;
vtklzma-8.2-gd.lib;
vtkmetaio-8.2-gd.lib;
vtkNetCDF-8.2-gd.lib;
vtkogg-8.2-gd.lib;
vtkParallelCore-8.2-gd.lib;
vtkpng-8.2-gd.lib;
vtkproj-8.2-gd.lib;
vtkpugixml-8.2-gd.lib;
vtkRenderingAnnotation-8.2-gd.lib;
vtkRenderingContext2D-8.2-gd.lib;
vtkRenderingContextOpenGL2-8.2-gd.lib;
vtkRenderingCore-8.2-gd.lib;
vtkRenderingExternal-8.2-gd.lib;
vtkRenderingFreeType-8.2-gd.lib;
vtkRenderingGL2PSOpenGL2-8.2-gd.lib;
vtkRenderingImage-8.2-gd.lib;
vtkRenderingLabel-8.2-gd.lib;
vtkRenderingLOD-8.2-gd.lib;
vtkRenderingOpenGL2-8.2-gd.lib;
vtkRenderingVolume-8.2-gd.lib;
vtkRenderingVolumeOpenGL2-8.2-gd.lib;
vtksqlite-8.2-gd.lib;vtksys-8.2-gd.lib;

```
vktheora-8.2-gd.lib;
vktiff-8.2-gd.lib;
vtkverdict-8.2-gd.lib;
vtkViewsContext2D-8.2-gd.lib;
vtkViewsCore-8.2-gd.lib;
vtkViewsInfovis-8.2-gd.lib;
vtkzlib-8.2-gd.lib;
%(AdditionalDependencies)
```

(23) Click 'Apply' and 'OK'

(24) Save as the Property sheet as 'PCL1.11.1.props'. **When you want to create a new project, you can just simply import this 'Property sheet' to your new project rather than setup again.**



(25) Congratulations, the PCL setup is done. Start your PCL adventure now.

Appendix C

Experimental Point Cloud Data

Unity interface

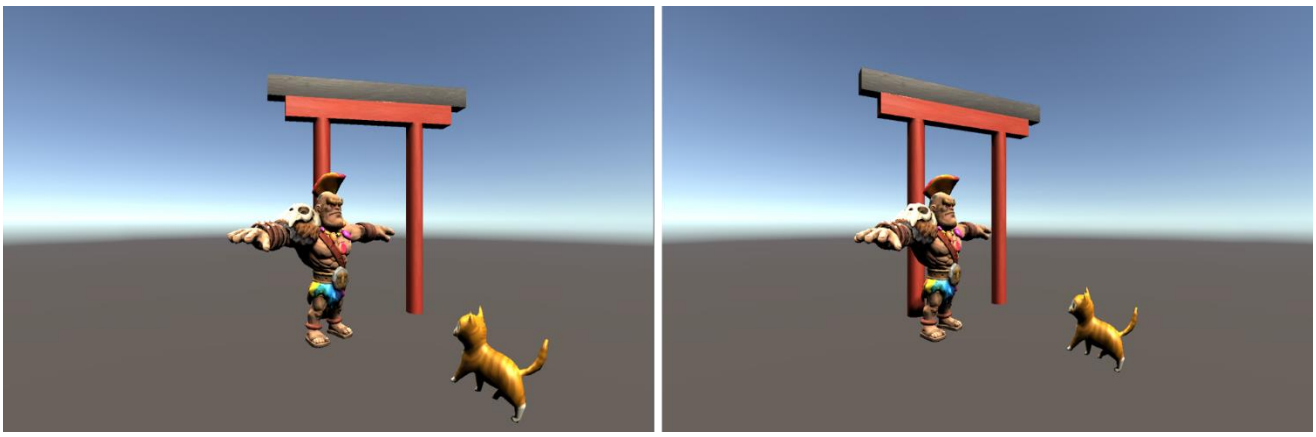
To test the general performance of different pipelines on point cloud registration with different noises, we collected experiment data in Unity with below rotation and translation:

- Translation:

$$\begin{pmatrix} x \\ y\ z \end{pmatrix} = \begin{pmatrix} +3 \\ 0\ 0 \end{pmatrix}$$

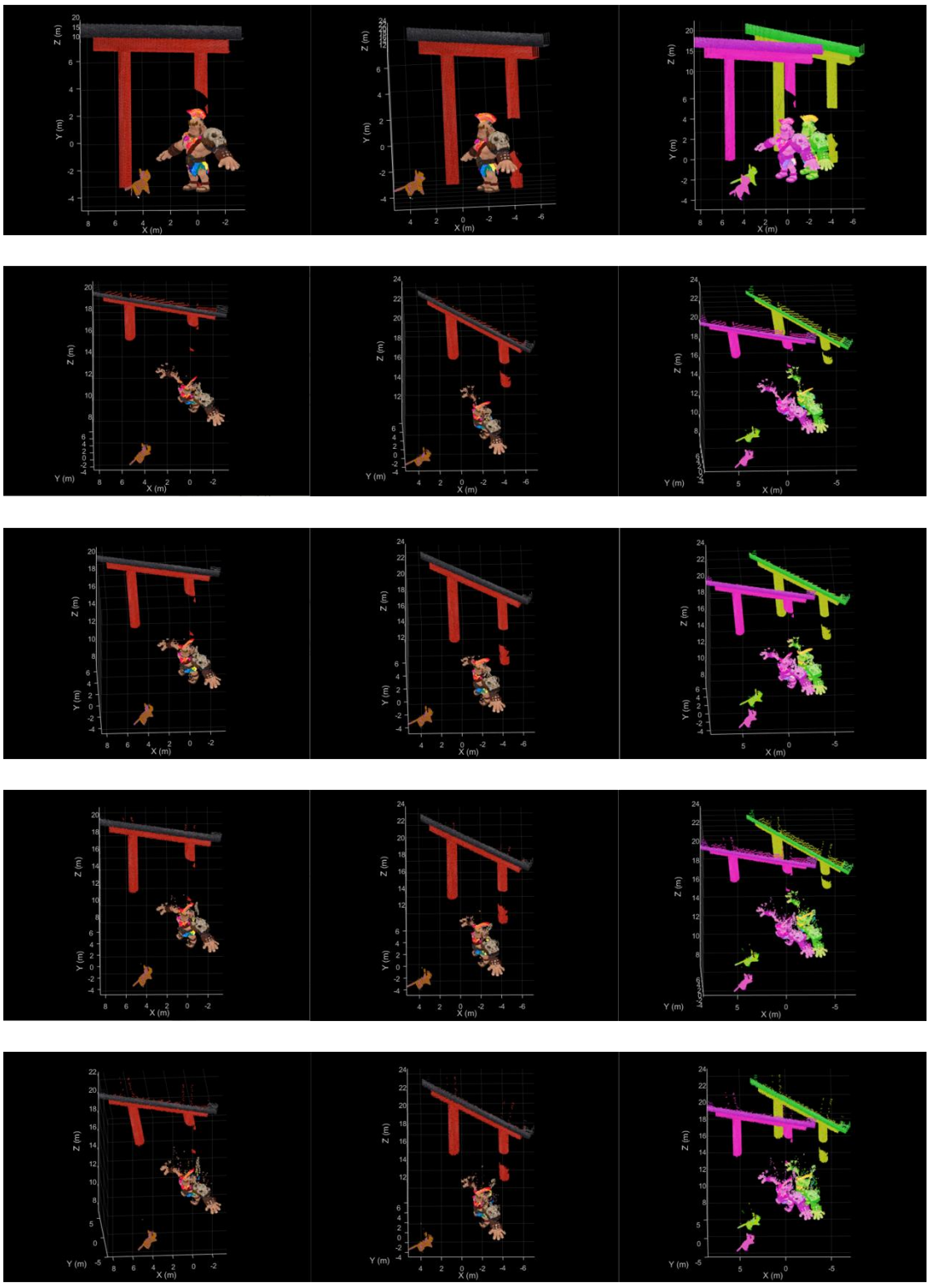
- Rotation:

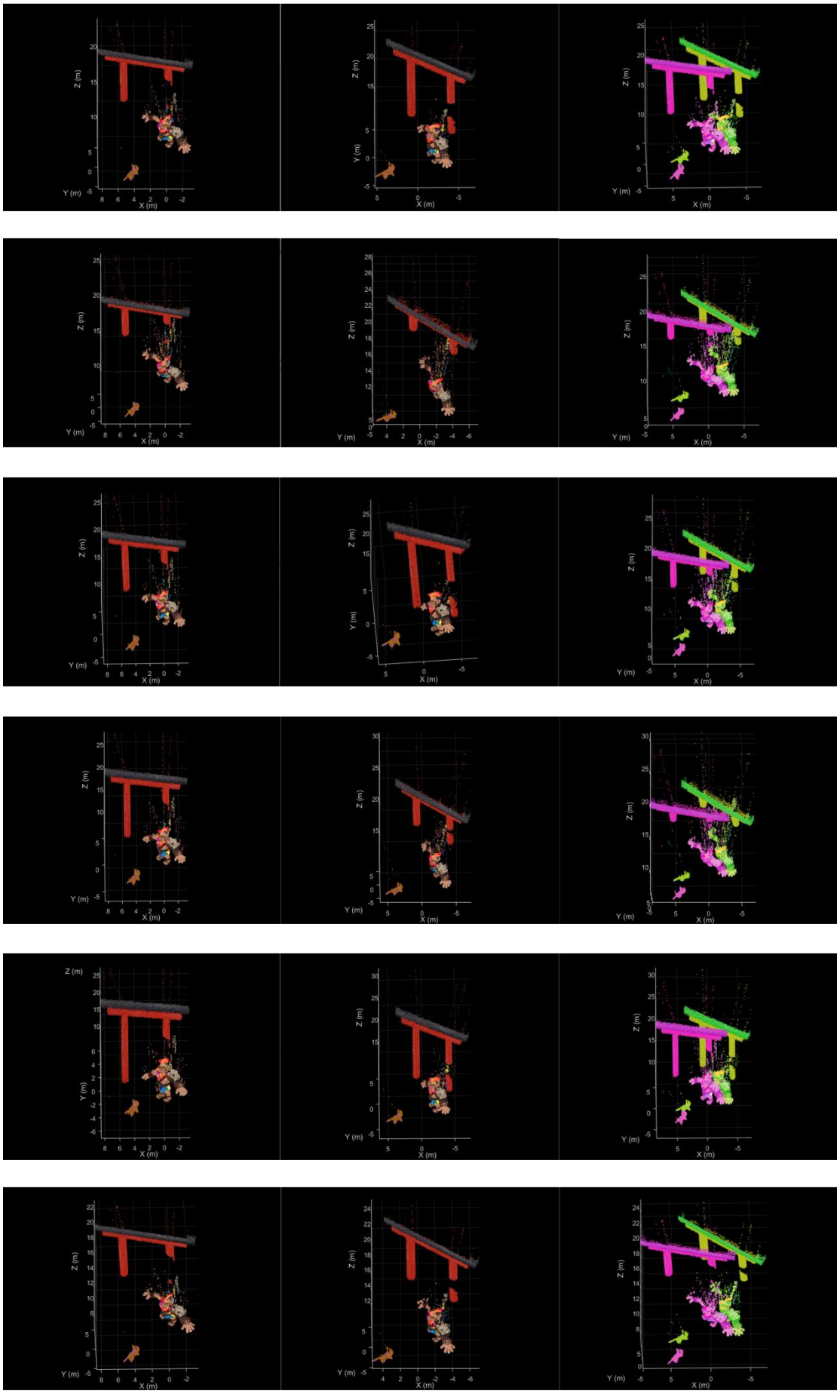
$$\begin{pmatrix} x \\ y\ z \end{pmatrix} = \begin{pmatrix} 0 \\ +30^{\circ}\ 0 \end{pmatrix}$$



Experimental data

Then, we captured point cloud data with different noises at the source viewpoint and target viewpoint as below:


Z (m)
Y (m)
X (m)


Z (m)
Y (m)
X (m)

References

[1] A. Flint, A. Dick, and A. Hengel, “Thrift: Local 3D Structure Recognition”, in 9th Biennial Conference of the Australian Pattern Recognition Society on Digital Image Computing Techniques and Applications, Dec. 2007, pp. 182–188.

[2] Yu Zhong, Intrinsic shape signatures: “A shape descriptor for 3d object recognition”, in: IEEE International Conference on Computer Vision Workshops, 2010, pp. 689–696.

[3] R. B. Rusu, N. Blodow, Z. C. Marton, M. Beetz, “Aligning point cloud views using persistent feature histograms”, in: Ieee/rsj International Conference on Intelligent Robots and Systems, 2008, pp. 3384–3391.

[4] R. B. Rusu, N. Blodow, M. Beetz, “Fast point feature histograms (fpfh) for 3d Registration”, in: IEEE International Conference on Robotics and Automation, 2009, pp. 1848–1853.

[5] A. E. Johnson, M. Hebert, “Surface matching for object recognition in complex 3-d scenes”, Image and Vision Computing 16 (9-10) (1998) 635–651.

[6] F. Tombari, S. Salti, L. D. Stefano, “Unique signatures of histograms for local surface description”, in: European Conference on Computer Vision Conference on Computer Vision, 2010, pp. 356–369.

[7] F. Tombari, S. Salti, L. D. Stefano, A combined texture-shape descriptor for enhanced 3d feature matching 263 (4) (2011) 809–812.

[8] A. Zeng, S. Song, Niener, Matthias, M. Fisher, J. Xiao, T Funkhouser. “3DMatch: Learning Local Geometric Descriptors from RGB-D Reconstructions”, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[9] Radu Bogdan Rusu, Nico Blodow, Michael Beetz. “Fast Point Feature Histograms (FPFH) for 3D registration”, in: IEEE International Conference on Robotics and Automation, 2009.

[10] A. Dai, M. Nießner, M. Zollh¨ofer, S. Izadi, and C. Theobalt. Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface re-integratio, 2016.

[11] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on, pages 127–136. IEEE, 2011.

[12] M. Nießner, M. Zollh¨ofer, S. Izadi, and M. Stamminger. Real-time 3d reconstruction at scale using voxel hashing. ACM Transactions on Graphics (TOG), 32(6):169, 2013.

[13] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D ShapeNets: A deep representation for volumetric shapes. 2015.

[14] J. Xiao, A. Owens, and A. Torralba. SUN3D: A database of big spaces reconstructed using SfM and object labels. 2013.

[15] A. V. Segal, D. Haehnel, and S. Thrun, “Generalized-ICP”, in: Robotics: Science and Systems 2009

[16] Y. Chen, G. Medioni. “Object Modeling by Registration of Multiple Range Images”, Proc. of the 1992 IEEE Intl. Conf. on Robotics and Automation, pp. 2724-2729, 1991.