Technical University of Munich

TUM Department of Civil, Geo and Environmental Engineering

Chair of Computational Modeling and Simulation

Localizing and Matching CAD Model in Point Cloud Using Semantic Registration Method

Master Thesis

for the Master of Science Course Civil Engineering

Author:	
Student ID:	
Supervisor:	Prof. Dr.-Ing. André Borrmann
	M.Sc. Yuandong Pan
	M.Sc. Florian Noichl
Date of Issue:	01. February 2021
Date of Submission:	01. August 2021

Preface

With the increasing application of computer technology in various fields, the combination of BIM and computer vision technology has become a current research hotspot. For the 3D reconstruction work of the existing scenes, it used to rely on tedious manual modeling, which often consumes a lot of time and labor costs. However, the emergence of point clouds makes this work simple and efficient. By scanning the real scene with lidar or camera to obtain the point cloud data, we can further process the point cloud data, so as to achieve the purpose of 3D scene reconstruction. In this process, how to accurately identify a specific object from the point cloud data is a key factor in 3D reconstruction.

Semantic registration is an important step when processing 3D point clouds. The applications of point cloud registration range from object modeling and tracking to simultaneous localization and mapping. The overall objective of registration is to align individual point clouds and fuse them to a single point cloud, so that subsequent processing steps like object reconstruction can be applied. This article studies the point cloud semantic registration in 3D reconstruction, and also studies other point cloud processing algorithms. This article is also my first attempt in the field of point cloud processing and 3D reconstruction.

I would like to thank my supervisor Mr. Yuandong Pan and Mr. Florian Noichl for the amazing guidance and mentoring that he showcased during the entire process of the project.

Chen Liu

Munich, 01-08-2021

Abstract

Development of as-built BIM presents an ongoing challenge for the global BIM and computing engineering communities. The manual process for constructing as-built BIMs is time-consuming and requires skilled workers, the modeling time can be several weeks or months for a normal building. Therefore, the development of automated as-built BIM from point cloud data is necessary and it´s the focus of current research. Point cloud refers to the type of data obtained through a 3D scanner. The scanned data is recorded in the form of points. Each point contains three-dimensional coordinates, color information and intensity. Due to these characteristics of point clouds, more and more researchers are considering applying point clouds in the field of as-built BIM 3D reconstruction. A very important part of the 3D scene reconstruction is to identify the specific CAD model in the point cloud. Semantic registration is an important approach to achieve this purpose.

This thesis aimed to complete a workflow of point cloud registration with Point Cloud Library(PCL). We first reviewed the mature point cloud processing algorithms, and introduced the relevant knowledge background in our thesis. Then a variety of point cloud processing algorithms in PCL were used in the registration process, such as noise filter, keypoint detection, feature descriptor computation, correspondence grouping and so on. Through this workflow, the CAD model can be accurately identified from the point cloud. And in the experiment part, we input multiple point clouds in different indoor scene to verify the point cloud registration process and fully analysis the influence of some critical factors. Finally, combined with the analysis of the experimental results, we put forward some suggestions for future research in this direction.

Zusammenfassung

Die Entwicklung von as-built-BIM stellt eine ständige Herausforderung für die globale BIM und Computing-Engineering-Community dar. Der manuelle Prozess zur Erstellung von Bestands-BIMs ist zeitaufwändig und erfordert Facharbeiter, die Modellierungszeit kann für ein normales Gebäude mehrere Wochen oder Monate betragen. Daher ist die Entwicklung von automatisiertem Bestands-BIM aus Punktwolkendaten notwendig und steht im Fokus der aktuellen Forschung. Punktwolke bezieht sich auf die Art von Daten, die durch einen 3D-Scanner erhalten werden. Die gescannten Daten werden in Form von Punkten aufgezeichnet. Jeder Punkt enthält dreidimensionale Koordinaten, Farbinformationen und Intensität. Aufgrund dieser Eigenschaften von Punktwolken erwägen immer mehr Forscher den Einsatz von Punktwolken im Bereich der As-Built-BIM-3D-Rekonstruktion. Ein sehr wichtiger Teil der 3D-Szenenrekonstruktion besteht darin, das spezifische CAD-Modell in der Punktwolke zu identifizieren. Die semantische Registrierung ist ein wichtiger Ansatz, um dieses Ziel zu erreichen.

Diese Arbeit zielte darauf ab, einen Workflow der Punktwolkenregistrierung mit der Point Cloud Library (PCL) zu vervollständigen. Wir haben zuerst die ausgereiften Punktwolkenverarbeitungsalgorithmen überprüft und den relevanten Wissenshintergrund in unsere Diplomarbeit eingeführt. Dann wurden eine Vielzahl von Punktwolken-Verarbeitungsalgorithmen in PCL im Registrierungsprozess verwendet, wie z. Durch diesen Workflow kann das CAD-Modell aus der Punktwolke genau identifiziert werden. Und im experimentellen Teil geben wir mehrere Punktwolken in verschiedenen Indoor-Szenen ein, um den Punktwolkenregistrierungsprozess zu überprüfen und den Einfluss einiger kritischer Faktoren vollständig zu analysieren. Zusammen mit der Analyse der experimentellen Ergebnisse unterbreiten wir schließlich einige Vorschläge für zukünftige Forschungen in dieser Richtung.

Contents

List of Figures

List of Table

List of Abbreviations

BIM	Building information modelling
2D	Two-dimensional
3D	Three-dimensional
CMS	Chair of Computational Modeling and Simulation
PCL	Point cloud library
CAD	Computer-aided design
BSD	Berkeley Software Distribution
RGB-D	Red green blue + Depth Map
NURBS	Non-uniform rational B-spline
FPFH	Fast Point Feature Histograms
SIFT	Scale-Invariant Feature Transform
SHOT	Unique Signatures of Histograms for Surface
ICP	Iterative Closest Point
FLANN	Fast Library for Approximate Nearest Neighbors
DOG	Difference-of-Gaussian
VTK	The Visualization Toolkit
CAM	Computer-aided manufacturing
CAE	Computer-aided engineering
PCB	Printed circuit board
SVD	Singular value decomposition
CV	Computer vision
KD	K-dimensional
RA	Reference axial

LRF	Local reference frame
SPFH	Simple Point Feature Histograms
PFH	Point Feature Histograms

1 Introduction and Motivation

1.1 Introduction

Point cloud refers to the type of data obtained through a 3D scanner. The scanned data is recorded in the form of points. Each point contains three-dimensional coordinates, and some may contain color information (R, G, B) or the intensity of the reflective surface of the object. Due to these characteristics of point clouds, more and more researchers are considering applying point clouds in the field of as-built BIM 3D reconstruction.

An as-built BIM refers to a digital representation of the facility as it was actually built or as it currently exists. Rich semantic information is contained in a BIM (NIBS, 2015). It covers geometry, spatial relationships, geographic information systems, and the nature and quantity of various building components. The building information model can be used to show the product life cycle of the entire building, including the construction process and the operation process. It is very convenient to extract information about the materials in the building. All parts and systems in the building can be presented.

However, the manual process for constructing as-built BIMs is time-consuming and requires skilled workers, the modeling time can be several weeks or months for an normal building. Therefore, the development of automated as-built BIM from point cloud data is necessary and it´s the focus of current research. A very important part of the 3D scene reconstruction is to identify the specific CAD model in the point cloud. There are currently two mainstream methods, semantic segmentation and semantic registration. Both methods have their own advantages and disadvantages and different usage scenarios.

In regard to modelling, the concept of semantic segmentation has arisen in different research domains, mainly in computer vision, and robotics[1]. As an important notion towards complete scene understanding, semantic segmentation is applied to numerous application such as autonomous driving, augmented reality, image search engines, and computational photography [2]. For semantic segmentation in indoor modelling, the research is usually performed on RGB-D sensor depth images for small indoor scenes[3]. Figure1.1 shows the semantic segmentation result from point cloud.

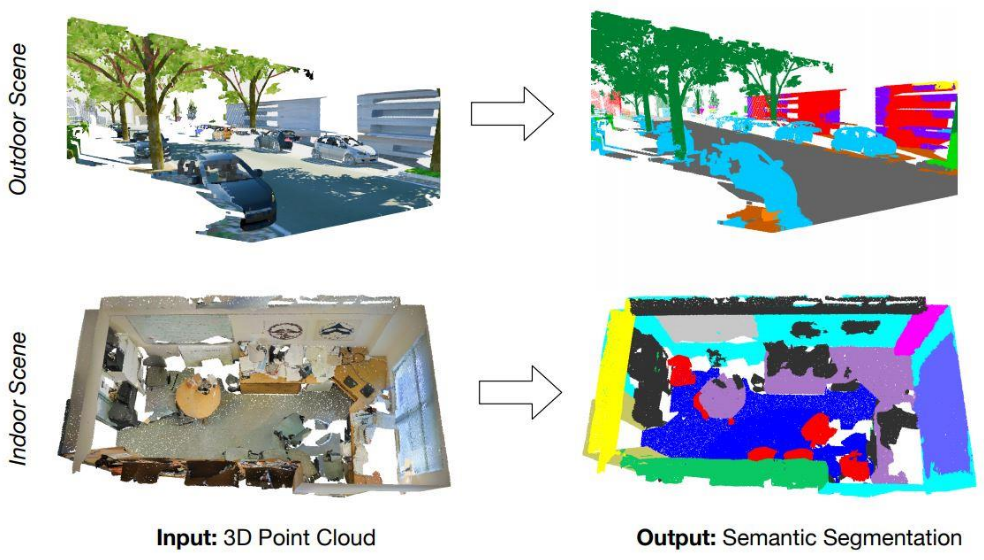


Figure1.1: Point clouds from indoor and outdoor scenes on the left. Semantic segmentation results produced by the presented method on the right.[11]

Nevertheless, although the semantic segmentation method has a wide range of applications in the field of computer version, the semantic segmentation methods are hard to implement, time-consuming, and in some cases inaccurate, especially for complex point cloud scenes[4]. For example, Babacan et al.'s (2017) deep learning model exhibited over 90% precision and 90% recall in segmenting walls and floors, but only roughly 40% precision for beams and 55% recall for doors. The SVM model in Koppula et al. (2011) succeeded in office point clouds with 80% precision and 70% recall, but unsatisfactory result for home point clouds. Wang et al.'s (2018) model was effective on dense point clouds (about 90% recall), but less effective on sparse and noisy data (about 40% recall). In summary, the semantic segmentation method is not effective when faced with the fundamental challenge of segmenting complex point clouds. Uncontrolled real-life environments characterized by diversity of labels, irregular geometries, and topological relationships all make semantic segmentation even less satisfactory[5].

Compared with semantic segmentation methods, semantic registration has more advantages when faced some complex point cloud scenes. The overall objective of registration is to align individual point clouds and fuse them to a single point cloud, so that subsequent processing steps like object reconstruction can be applied. Usually the point clouds are captured by 3D sensors from different viewpoints, then the registration finds the relative position and orientation between views in a global coordinate frame, such that the overlapping areas between the point clouds match as well as possible[6].

And semantic registration has wide application in computer vision, computer graphics, robotic perception, photogrammetry, cultural heritage modeling, digital archaeology, and architecture. Figure1.2 shows an example of point cloud registration.

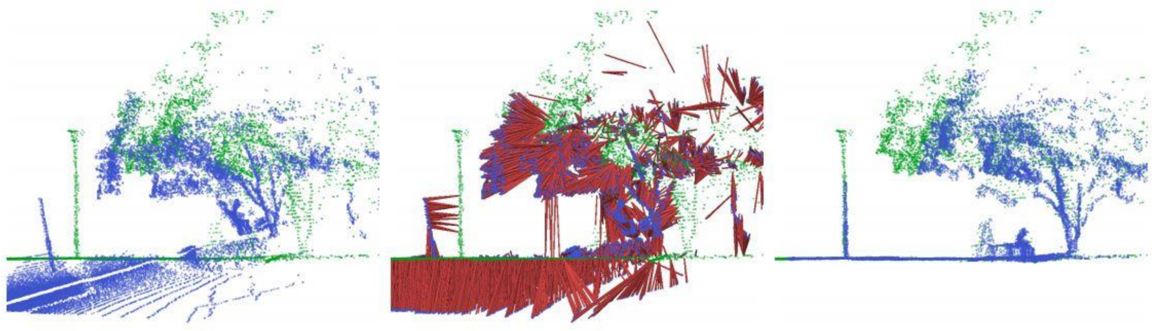


Figure1.2: Examples of registration between a reference point cloud (light green points) and a reading point cloud (dark blue points). Left: Initial position of the two point clouds. Middle: Alignment error (dark red lines). Right: Final alignment of the two point clouds.[19]

In this thesis, align one point cloud A (3D model) with another point cloud B (3D scene) to generate the posture information of point cloud A relative to point cloud B is the main focus. The tools of registration in this article are mainly from open-source Point Cloud Library(PCL). PCL is a stand-alone, large scale, open project for 3D point cloud processing. It is released under the BSD license and contains numerous state-of-the-art algorithms for various applications including noise filtering, object feature estimation, 3D surface reconstruction, semantic segmentation, visualization, point cloud registration, as well as higher-level tools and applications for performing mapping and object recognition.

1.2 Related literature

The problem of indoor scene automated modeling by semantic registration has been extensively studied, there are several ways to process raw point cloud data. Barazzetti[12] used NURBS curves and surfaces to reconstruct complex and irregular objects, but the user has to identify the different structural objects from point clouds data and extract the discontinuity lines of constructive elements manually. Xue[13] present an optimization-based model generation method which semantic BIM components could be organized automatically. A fitness function was generated from target measure in point cloud then the fittest model was output with necessary inverse transformation. This method has a higher automation level and a cost saving could also be expected, but the accuracy and availability of component libraries match the target measure not perfectly. Wang[14] propose a new approach to improve efficiency of automatic reconstruction by fully exploring the regularity and variability of man-made structure in indoor scene. Furthermore, a key point extraction algorithm was designed to match the scanned objects’ point cloud data and database models. Corsia[15] designed a novel segmentation algorithm based on region growing and edge detection, then each segment is computed keypoints and multi-scale features to match the CAD models from database. This method has higher efficiency and accuracy in processing large-scale scenes. Xue[4] present a semantic registration approach to recognize BIM components from 3D point clouds directly, then in order to optimize the suitability of CAD Models an objective function was designed to minimize the Root Mean Square Error between the as-built BIM and the input 3D point cloud. Xu[16] developed a framework of reconstruction of indoor scenes, the method is on the basis that objects are movable and environment-independent. However it´s not suitable for large-scale scene (e.g.Lecture Hall) with immovable chairs or desks. Nan[17] develop a approach for object-level scene reconstruction, over segmented input point cloud data was classified and templated were deformed to fit to the classified point cloud, finally the best matching templates were selected. Kerber[18] designed a novel symmetry detection method for large-scale point cloud data. The feature descriptors is designed for each sample point in order to locate all symmetry points. The author used this method in the overall modeling of the building outlook and achieved good results.

1.3 Goal of thesis

This thesis mainly realizes the correct registration of the target model point cloud from the scene point cloud, which combines a variety of algorithms in the PCL. The target model point cloud data is converted from CAD model. The scene point cloud data used in the thesis is collected by the CMS department. The flow chart of point cloud registration is shown in Figure1.3.

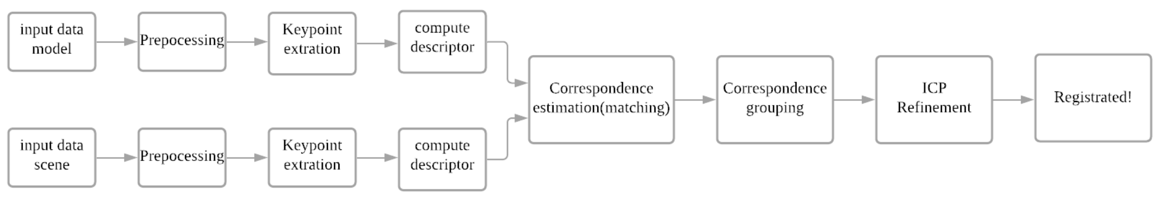


Figure1.3: flow chart of point cloud registration in this thesis

PCL is an important assistant for this thesis, and many algorithms in the PCL are used. For example, downsampling VoxelGrid filter in the prepocessing of point cloud, SIFT algorithm and uniform sampling algorithm in point cloud keypoint extraction, FPFH method and SHOT descriptor in point cloud descriptor computation, correspondence estimation and correspondence grouping, etc. The Point Cloud Library provides specific implementations of these algorithms.

In order to ensure that the algorithm has better robustness, a variety of different algorithms have been tested, and various point cloud scenarios have been used to verify the algorithm. Although the final model is not fully optimized, this study provides a relatively comprehensive and meticulous methodology with plenty of experiments and illustrations.

1.4 Layout of this thesis

Chapter 1 points out the significance why 3D point cloud registration is needed based on current situation and trend, the robustness of the semantic registration method in some complex scenes is clarified compared with semantic segmentation. Here the feasibility of this technique is shown in described scenarios. And this chapter announces the objective of this study and a brief procedure for evaluation and optimization.

Chapter 2 introduces the theoretical background of the algorithms used in the point cloud registration process. For example, Harris detector and Scale-invariant Feature Transform(SIFT) detector used in key points extraction, Fast Point Feature Histograms (FPFH) descriptor and SHOT descriptor used in descriptor computation, Local Surface Patches and Hough Voting in correspondence grouping, and a variant of Iterative Closest Point(ICP) algorithm in registration refinement.

Chapter 3 first lists the tools involved in this study and then demonstrate the workflow of point cloud registration. Explains in detail how to achieve the successful registration of the target point cloud and the scene point cloud through the step-by-step processing of the point cloud.

Chapter 4 shows some specific test cases, including different algorithm, different parameter combinations and their corresponding results. By analyzing the result of registration, a better parameter combination is to find, and the robustness of methodology is to test and verify.

Chapter 5 discusses some further influence on point cloud registration, and analyzes the drawback of this study. Some ideas for current method are proposed to achieve a more accurate and stable optimization in the future.

2 Theoretical Background

This chapter explores the theoretical basis for studying into the research goal. To lay the foundation, required knowledge in data structure and Computer Vision will be introduced.

2.1 KD-tree

A KD-tree is a data structure used in computer science for organizing some number of points in a space with K dimensions, first introduced by Bentley (1975)[7]. The KD-tree is a binary tree in which each node is a k-dimensional point. All non-leaf nodes can divide the space into two half spaces as a hyperplane. Figure 2.1 is an example of the subdivisions of a 2D space[8]. The corresponding binary tree is on the right side of the plot.

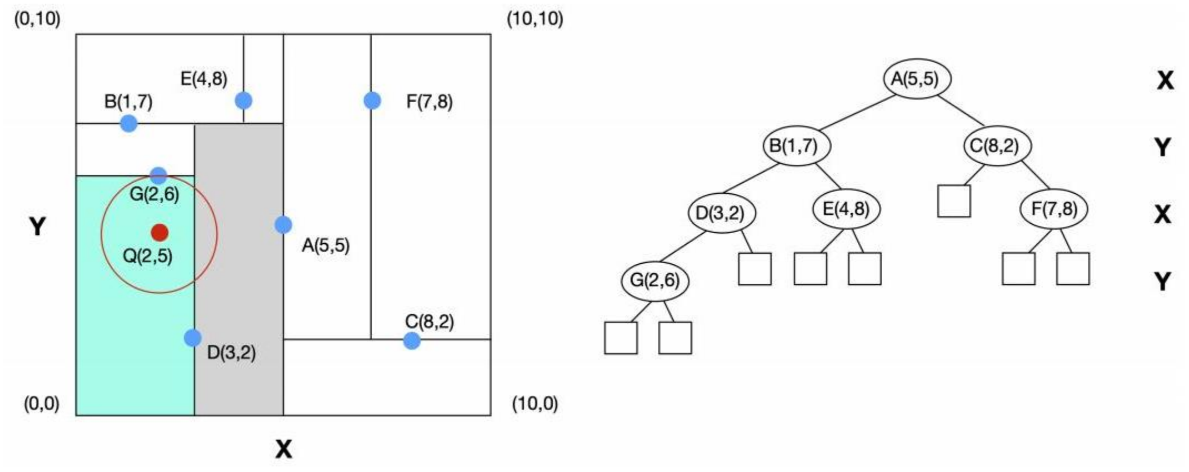


Figure 2.1: 2D feature space shown on the left corresponds to the tree on the right [7].

Because we will generally only deal with point clouds in three dimensions, so all of the k-d trees will be three-dimensional. In this thesis, we used KD-tree to perform a nearest neighbor search. Compared with exhaustive search, KD-tree nearest neighbor search can be more efficient, saving time and computing resources. It has been empirically proven to be one of the best performing solutions[9]. In order to initialize the algorithm faster, PCL depends on Fast Library for Approximate Nearest Neighbors(FLANN)[10], an open-source library for fast nearest neighbor searches.

2.2 SIFT Keypoints

Scale Invariant Feature Transform(SIFT) is a scale-based Spatial image local feature description algorithm[25]. Although it can only describe the local features of the image, but this feature is very important for viewing angle changes and affine changes. For change of rotation, scale scaling, and brightness, it will maintain a certain degree of stability. The principle of SIFT is as follows:

2.2.1 Generation of scale space

First of all, we have to understand a concept, what is scale space. From the perspective of cognition, in an image, even if we have no concept of an object or we are not familiar with it, people can still perceive the structure of the object. If you want to know the meaningful of an image, you must first clarify the question: In an image, only within a certain scale range, an object is meaningful. To give an example, the concept of a branch can only be perceived as a branch by observing it at a distance of a few centimeters to a few meters. If you observe at the micrometer level or the kilometer level, you will not be able to perceive the concept of branches. In this way, you can perceive the concept of cells or forests.

Therefore, the blur degree of each image becomes larger in the scale space gradually, which can simulate the formation process of the target on the retina when the distance between people and target from close to far. And the larger the scale, the more blurred the image.

The scale space $L(x, y, \sigma)$ is defined by the convolution of a variable-scale Gaussian $G(x, y, \sigma)$ with an input image $I(x, y)$. The function is as following:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

$G(x, y, \sigma)$ is variable-scale Gaussian function, σ is factor of scale space, it determines the smoothness of the image, (x, y) is coordinate of image pixel.

Figure 2.2.1 shows a practical method of establishing DOG. The initial image is convolved with Gaussian functions with different σ values to obtain a set of blurred images, and then this set of blurred images is subtracted from each other to obtain the corresponding DOG. These blurred images are separated in the scale space by a factor of K, and the highest scale in the set should be twice the lowest scale. In order to carry out follow-up work and meet the above requirements, each set needs to get $s + 3$ blurred images through convolution, and $k = 2^{1/s}$.

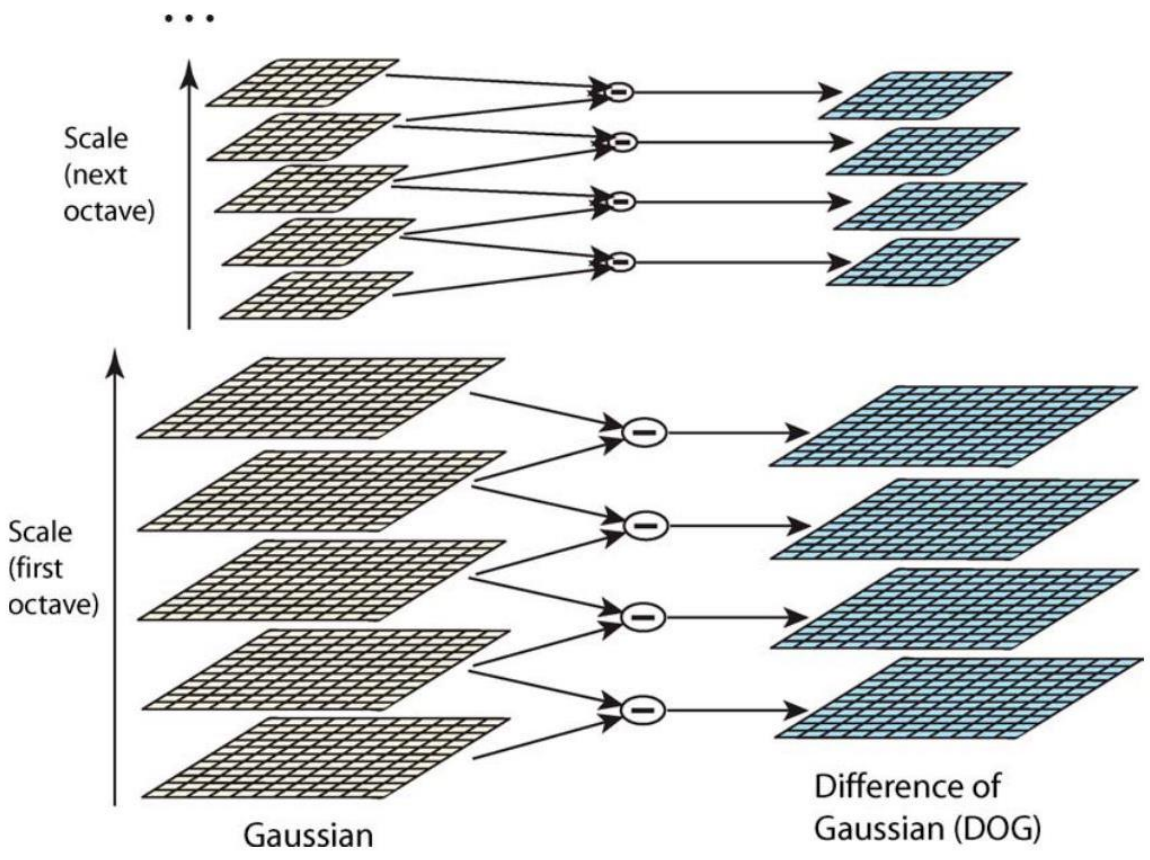


Figure 2.2: The process to construction of DOG[25]

After a stack of images is created, downsampling is needed to get the DOG of the next stack of images. In actual operation, a fuzzy image is created with the σ value twice that of the first image, and then the image is downsampled, that is, one pixel is extracted from every two pixels, and the DOG of the next image can be obtained.

2.2.2 Detect extreme points in scale space

To efficiently detect stable key point locations in scale space, Lowe[26] proposed using scale-space extrema in the difference-of-Gaussian function convolved with the image.

$$D(x,y,\sigma) = \big(G(x,y,k\sigma) - G(x,y,\sigma)\big) * I(x,y) = L(x,y,k\sigma) - L(x,y,\sigma)$$

$D(x,y,\sigma)$ is difference-of-Gaussian function, this function is introduced for effective detection of stable feature points in scale space. When detecting extreme points, each pixel must be compared with 26 points (8 adjacent pixels of the same scale and its upper and lower adjacent scales 9×2 points), which is shown in figure 2.3. If the DOG (difference of Gaussian) operator of one pixel is the biggest or smallest in these 26 points, then this pixel is the feature point.

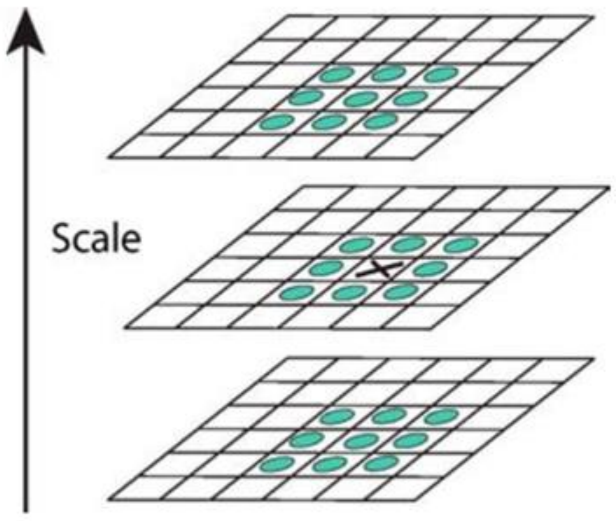


Figure 2.3: Maxima and minima of the difference-of-Gaussian images are detected by comparing a pixel (marked with X) to its 26 neighbors in 3 × 3 regions at the current and adjacent scales (marked with circles)[25]

2.2.3 Locate extreme points precisely

By fitting a three-dimensional quadratic function to accurately locate the feature points position and scale (get sub-pixel accuracy), while removing low contrast feature points and unstable edge response points to enhance stability of matching and improve the ability of anti-noise. Performing the curve fitting of the scale space DOG function, using the Taylor expansion of the DOG function in the scale space, we can get the following formula:

$$D(x) = D + \frac{\partial D^T}{\partial x} x + \frac{1}{2} x^T \frac{\partial^2 D}{\partial x^2} x$$

where D and its derivatives are evaluated at the sample point and $x = (x, y, \sigma)^T$ is the offset from this point. The location of the extremum is determined by taking the derivative of this function with respect to x and setting it to zero:

$$\hat{X} = -\frac{\partial^2 D^{-1}}{\partial X^2} \frac{\partial D}{\partial X}$$

Add $\hat{X}$ to the location of its sample point to get the interpolated estimate for the location of the extremum.

2.2.4 Specify direction parameters for each feature point

Using the gradient direction distribution characteristics of the neighboring pixels of the feature point specifies the direction parameter for each feature point, so that the operator process the ability of rotation invariance. The gradient mode and direction of each pixel are:

$$m(x,y) = \sqrt{(L(x+1),y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = tan^{-1}(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)})$$

Sampling in the neighborhood window centered on the feature point, and using histograms to count the gradient direction of neighboring pixels. The range of gradient histogram is $0^{\circ}{\sim}360^{\circ}$, every 10° is a column, and the gradient modulus m is used as the contribution weight, that is, the more far away of the neighborhood from the center point, the smaller of its contribution to the histogram.

So far, the feature points of the image have been detected, and each feature point has three pieces of information: location, scale and direction. The SIFT keypoint class in PCL converts the two-dimensional image SIFT operator into 3D space, this class can calculate out the SIFT keypoint of point cloud, to realize the direct application of SIFT keypoint in point cloud.

2.3 FPFH descriptor

Fast Point Feature Histograms (FPFH) is a feature descriptor based on the normal angle between points and their neighboring points, and the angle between points. FPFH descriptor is first introduced by Rudu[23]. It is a optimized algorithm from Point feature Histograms(PFH). The improved algorithm retains the main geometric characteristics of the point description in PFH, and reduces the computational complexity from O($n \cdot k^2$) to O($n \cdot k$), where n is the midpoint of the point cloud data The number of points, k is the number of points contained in the neighborhood of each point.

To understand the principle of FPFH, we need to first understand the working mechanism of PFH. The PFH descriptor forms a multi-dimensional histogram to describe the geometric properties of the k-neighborhood of the point by parameterizing the spatial difference between the query point and the neighboring point. The high-dimensional hyperspace where the histogram is located provides a measurable information space for the feature representation. It is invariant to the 6-dimensional pose of the corresponding surface of the point cloud, and is robust under different sampling densities or neighborhood noise levels. The point feature histogram (PFH) representation is based on the relationship between a point and its k neighbors and their estimated normal. It considers all the interactions between the normal directions and tries to capture the best sample surface changes. Describe the geometric characteristics of the sample. Therefore, the quality of surface normal estimation is an important factor for PFH[24]. The calculation principle of PFH is shown in Figure 2.4. For any point p_q in space (marked in red), p_{k1}~p_{k5} are the neighborhood points with p_q as the center and radius. In order to describe the relationship between any two points p_s, p_t and the corresponding normals of the points, a local coordinate system needs to be established with one of the points as the origin, as shown in Figure 2.5.

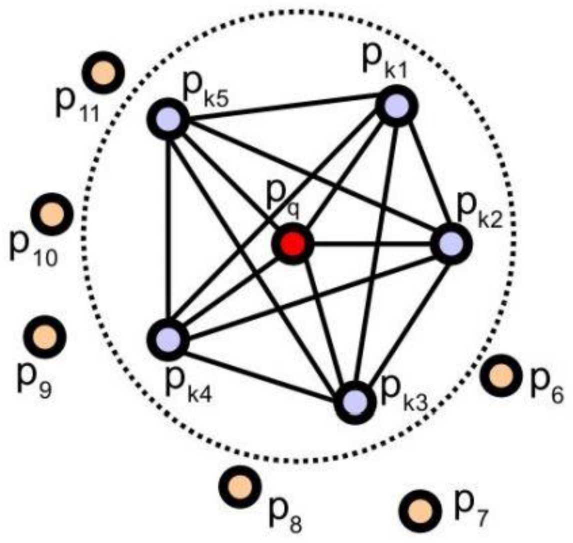


Figure 2.4: The influence region diagram for a Point Feature Histogram[23].

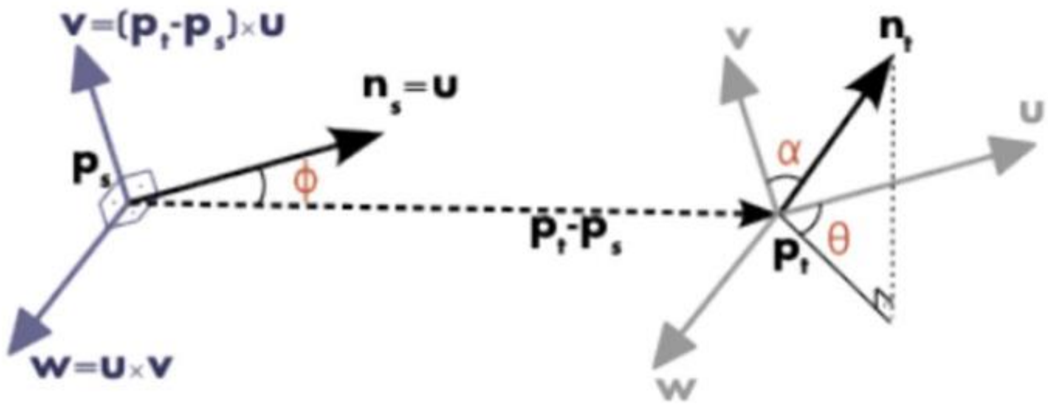


Figure 2.5: The local reference frame and the three angles α, φ and θ computed by FPFH at each point pair[6].

The PFH feature is further optimized and upgraded to obtain the Fast Point Feature Histogram (FPFH). The calculation process of FPFH is similar to that of PFH. For any point p_q (marked in red) and its k-neighboring points p_{k1}~p_{k5}, construct a local coordinate system according to each neighboring point, and obtain a quadruple with each k-neighboring point, and obtain the point feature histogram through statistics The graph is called Simple Point Feature Histograms (SPFH) due to the lack of the pairwise interconnection between neighboring points and neighboring points. As shown in Figure 2.6, then use p_{k1}~p_{k5} as the target points to find the k-neighboring points, calculate the normal vector, construct the local coordinate system, and obtain the SPFH.

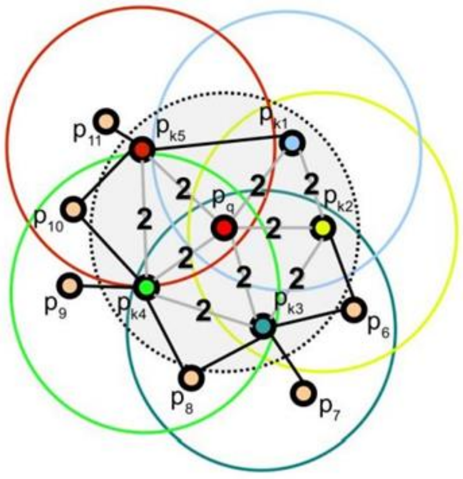


Figure2.6: The influence region diagram for a Fast Point Feature Histogram. Each red point is connected only to its direct k-neighbors (enclosed by the gray circle). Each direct neighbor point is connected to its own neighbors and the resulted histograms are weighted together with the histogram of the red point to form the FPFH. The connections marked with 2 will contribute to the FPFH twice[23].

In this way, the SPFH(p_k) with p_q and its neighboring points as the target point is obtained, and the FPFH feature calculation formula of the final point p_q is shown in formula[23]:

$$FPFH(p) = SPF(p) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{w_k} \cdot SPF(p_k)$$

2.4 SHOT descriptor

The traditional point cloud local feature descriptor (3D Local Feature Descriptor) can be divided into Signature and Histograms, which is shown in Figure 2.7.

Signatures describe the 3D surface neighborhood (support) of a given point by defining an invariant local reference frame and according to the local coordinates, encoding one or more geometric measurements computed individually at each point of a subset of the support[20]. It encodes local spatial geometric information by defining a local reference coordinate system (LRF). Due to the existence of LRF, the eigenvalues of its local geometric space are ordered.

Histograms describe the support by encoding counters of local topological entities into histograms according to a specific quantized domain such as point coordinates[20]. Histogram divides the eigenvalues into intervals, and encodes them in the way of histogram statistics, and the statistics are disordered.

Signature of Histograms of Orientations (SHOT) is a 3D descriptor that encodes histograms of basic first-order differential entities, which are more representative of the local structure of the surface compared to plain 3D coordinates. It's first introduced by Tombari and get a good result[20] and then merged with texture-based measurement[21]. The use of histograms defined a unique and robust 3D local reference frame, it can bring in the effect that filtered the noise. By introducing the geometric information concerning the location of the points within the support, making it possible to enhance the discriminative power of the descriptor.

On the one hand, by defining an invariant local coordinate system (RF) and separately encoding and calculating the geometric characteristics of a point's support area according to the local coordinates, the 3D surface neighborhood of a given point (here

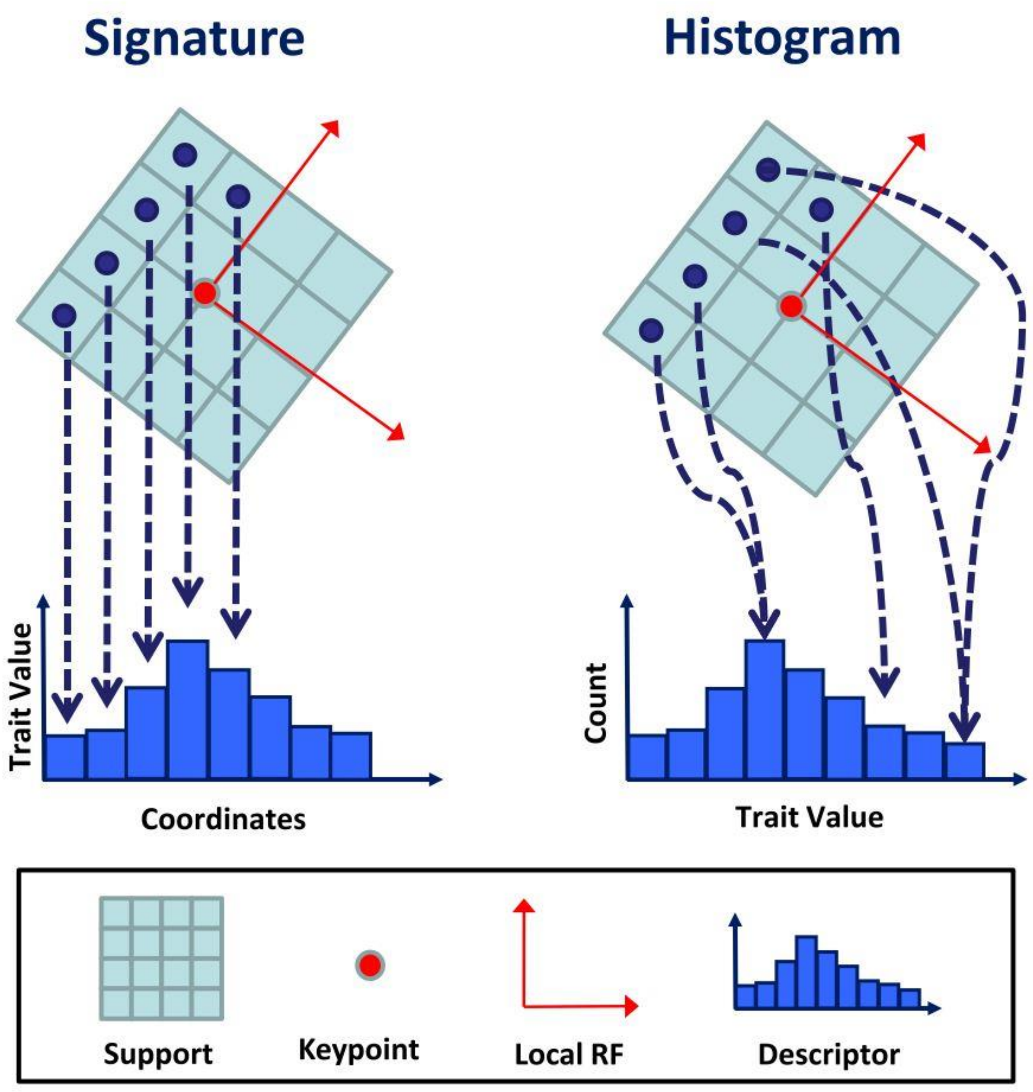


Figure 2.7: Signatures and Histograms are the two main 3D descriptors[20].

inafter referred to as support) is described. On the other hand, the histogram-based method describes the support area by accumulating local geometric or topological measurements (such as the number of points, grid triangle area) into a histogram according to a specific quantization domain (such as point coordinates, curvature), which requires definition Reference Axis (RA) or local RF.

The steps to generate a SHOT descriptor are following:

(1).The local reference coordinate system LRF is established according to the neighborhood information of the feature point sphere, and the sphere neighborhood of the feature point is divided into the radial direction (inner and outer sphere), longitude (time zone) and latitude direction (north and south hemisphere). Usually the radial is divided into 2, the longitude is divided into 8, and the latitude is divided into 2, a total of 32 Small Areas.

(2).Building a distribution histogram for each Small Areas separately. A set of local histograms over the 3D volumes defined by a 3D grid superimposed on the support are computed. For each of the local histograms, cosθ_i as a function to accumulate point counts into bins by the angle θ_i, which located between the normal at each point within the corresponding part of the grid n_{vi}, and the normal at the feature point n_u. That is, the function can be expressed as cosθ_i=$n_{vi} \cdot n_u$. According to the value of cosθ_i on the local surface, the corresponding histogram interval is accumulated. In order to overcome the boundary effect in the histogram construction process, when each point is accumulated to a specific interval of the histogram, quadrilinear interpolation is performed on the adjacent interval of the histogram and the corresponding interval of the adjacent Small Area histogram. The dimension of the orientation histogram feature depends on the number of histogram intervals, and its dimension is $n \cdot 32$. Tom et al. used experiments to prove that when n=11, that is, the feature dimension is 352, the orientation histogram feature achieves the best recognition rate [22].

As for the structure of the signature, we use an isotropic spherical grid that encompasses partitions along the radial, azimuth and elevation axes, as described in Figure 2.8 [20].

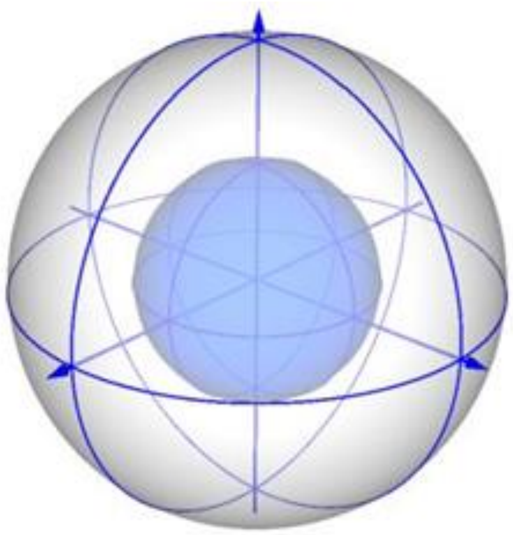


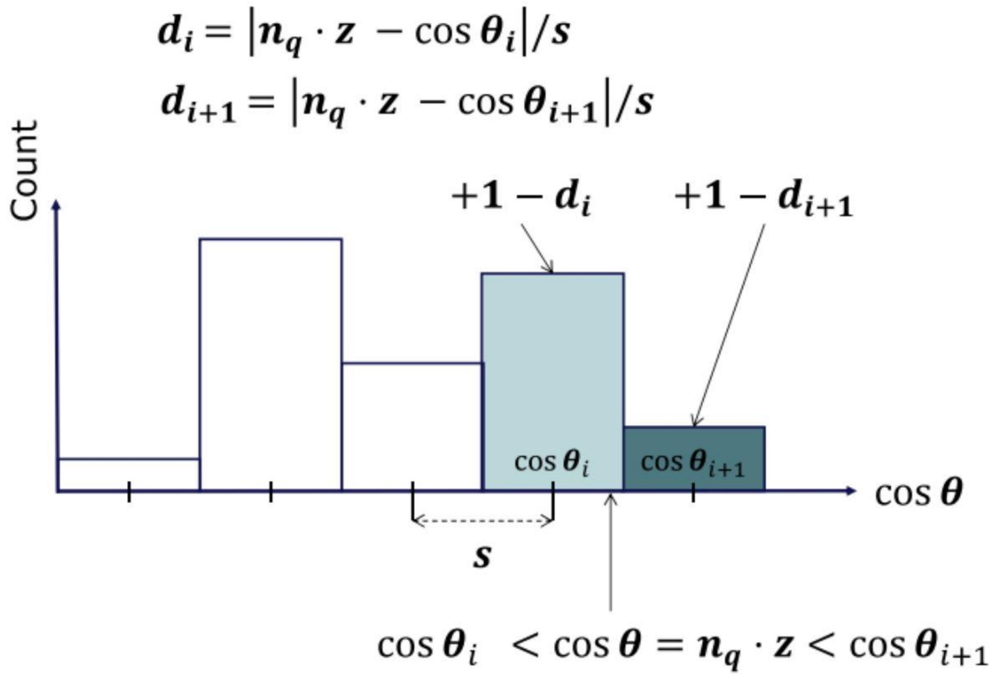
Figure 2.8: The 3D grid deployed by SHOT, which is repeatably oriented by the local reference frame denoted by the blue arrows.

For the Quadrilinear Interpolation method to overcome the boundary effect, that is, when voting for each point, interpolation voting is carried out from four dimensions according to the distance weight. First, for the same small area (volume), 11 bins are divided, and interpolation voting is performed from the $\cos\theta$ dimension, as shown in the Figure 2.9 below; then, for its adjacent small areas (volumes) based on the weight d (angle distance or Euclidean distance) performs corresponding interpolation voting, as shown in the following Figure (2.10, 2.11, 2.12).

In other word, each point performs interpolation voting on 8 small regions (volume) in the three dimensions of radial, azimuth, and elevation, and each small region (volume) performs interpolation voting on two intervals (bins) for the $\cos\theta$ dimension.. Therefore, a total of 16 bins are interpolated voting for each point.

(1). Normal Cosine Interpolation

Assuming that the characteristic value of a point in the support area is $\cos\theta$, which is in the interval of ($\cos\theta_i$, $\cos\theta_{i+1}$), first calculate the normalized distance (divided by the interval length S) from $\cos\theta$ to $\cos\theta_i$ and $\cos\theta_{i+1}$, marked as d_i and d_{i+1}, and then vote for $\cos\theta_i$ interval as $+1 - d_i$, and vote for $\cos\theta_{i+1}$ interval as $+1 - d_{i+1}$. The function of linear interpolation is to distribute the current value to adjacent discrete intervals in a linear proportion.



$d_i = |n_q \cdot z - \cos\theta_i|/s$
$d_{i+1} = |n_q \cdot z - \cos\theta_{i+1}|/s$
Count
$+1 - d_i$
$+1 - d_{i+1}$
$\cos\theta_i$
$\cos\theta_{i+1}$
$\cos\theta$
s
$\cos\theta_i < \cos\theta = n_q \cdot z < \cos\theta_{i+1}$

Figure 2.9: Interpolation on normal cosines[22]

(2). Azimuth interpolation

For azimuth, the weight d is calculated as the angular distance. The interpolation method is the same as above.

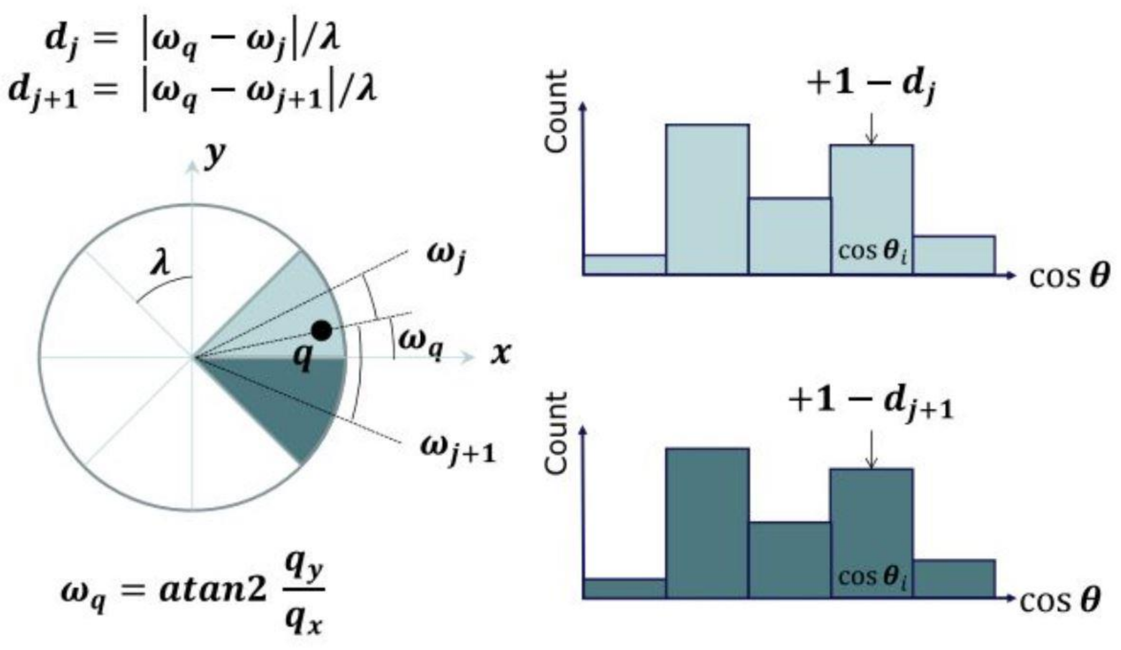


Figure 2.10: Interpolation on azimuth[22]

(3). Elevation Interpolation

For elevation, the weight d is also calculated as the angular distance. The interpolation method is the same as above.

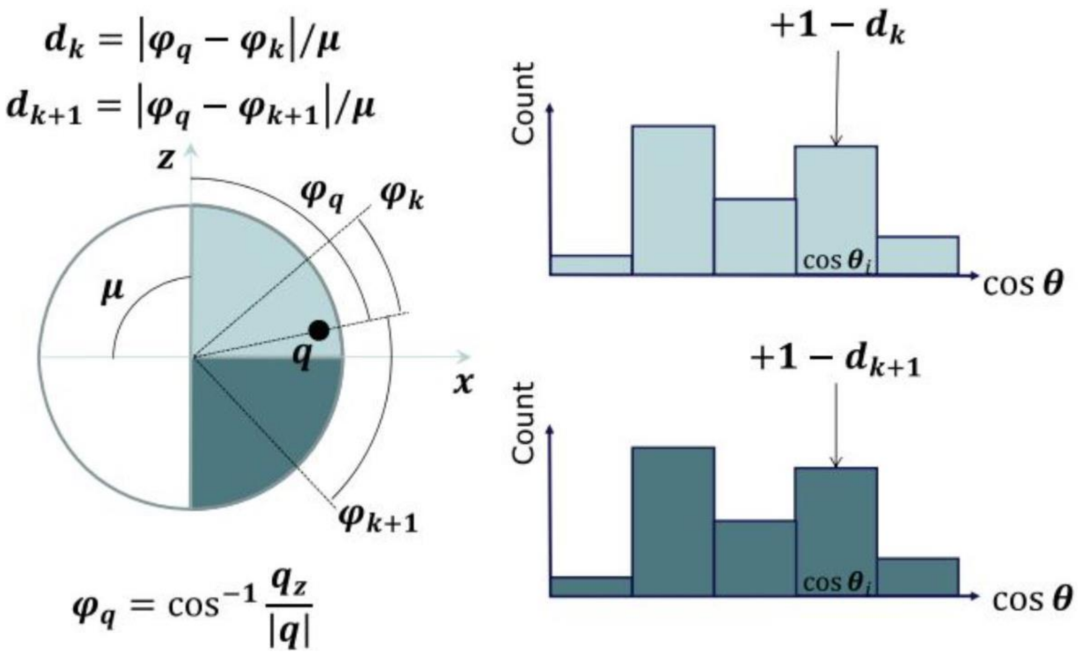


Figure 2.11: Interpolation on elevation[22]

(4). Distance Interpolation

For the distance dimension, the weight d is calculated based on Euclidean distance.

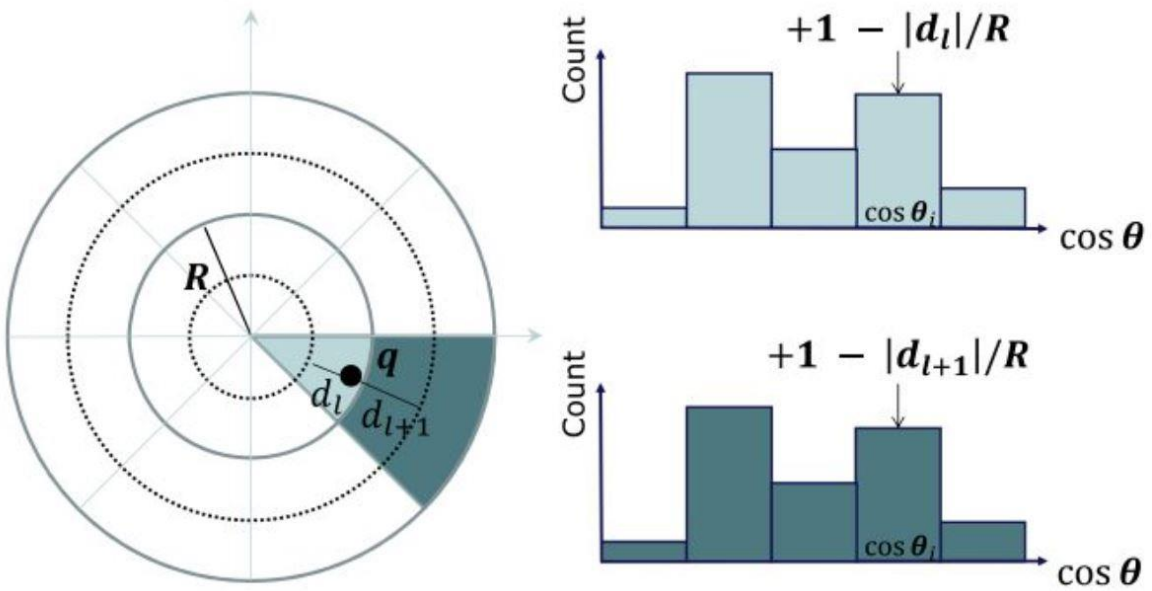


Figure 2.12: Interpolation on distance[22]

According to the above-mentioned academic principles, the Point Cloud Library provides users with specific SHOT descriptor algorithm implementation. Using the Point Cloud Library, we can easily and faster use the SHOT descriptor in the point cloud registration algorithm.

2.5 VoxelGrid Downsampling

Since the amount of point cloud data obtained is relatively large, if all the point cloud data are used to do the experiments, it will affect the speed of subsequent experiments, so we need to use part of point cloud to replace all point clouds to improve operational efficiency[26]. In PCL the VoxelGrid class can achieve this purpose, through the input point cloud data to create a three-dimensional voxel grid (think of the voxel grid as a collection of tiny three-dimensional cubes in space), and then in each voxel the center of gravity of all points in the voxel is used to approximate the other voxels point. The steps of making voxel grid are as follows:

2.5.1 Determine the length L

L is very important for the voxelgrid downsampling, if L is too large, the search efficiency will be reduced. If L is too small, there will be many empty grids. Side length of small cube grid is: $L = \alpha\sqrt[3]{s/c}$, where α is used to adjust the small cube grid of the side length, s is the scale factor, c is the number of point clouds in small grid.

2.5.2 Determine the volume of the 3D voxel grid

The volume of 3D voxel grid is $V = L_x L_y L_z$, among them, L_x, L_y, L_z are the maximum range of the point cloud on the X, Y, and Z axes. The number of point clouds contained in the unit small grid is $c = N/V$ (N is the point cloud total), then we can get $L = \alpha\sqrt[3]{sL_x L_y L_z/N}$. Then we can divide the point cloud data into small cube grids.

2.5.3 Find the gravity center

The gravity center of each voxel grid is: $X_0 = \frac{1}{C}\sum_{i=1}^{c} x_i\,, Y_0 = \frac{1}{C}\sum_{i=1}^{c} y_i\,, Z_0 = \frac{1}{C}\sum_{i=1}^{c} z_i$. Using the X_0, Y_0, Z_0 to represent all the points in the small grid to achieve the downsampling of the point cloud. The purpose of simplification of point cloud can be achieved.

After VoxelGrid Downsampling, the number of point clouds is reduced, but the shape characteristics of the point clouds is not changed. Therefore, this method is used to improve the efficiency of point cloud registration.

2.6 Iterative Closest Point

Iterative closest point is one of the widely used algorithms in aligning three dimensional models in the field of point cloud registration. It is a high-level registration method based on free-form surfaces, which was first introduced by Besl and McKay[28]. In the iterative closest point algorithm, we need a source point cloud and a target point cloud. In usual condition, the target point cloud is kept fixed, while the other one, the source point cloud, is transformed to best match the reference. The algorithm iteratively revises the transformation (combination of translation and rotation) needed to minimize a distance from the source to the reference point cloud, such as the sum of squared differences between the coordinates of the matched pairs. In the Figure 2.13, we can see the transformation process of the ICP algorithm intuitively. After multiple iterations, the blue curve is matched with the closest point corresponding to the red line.

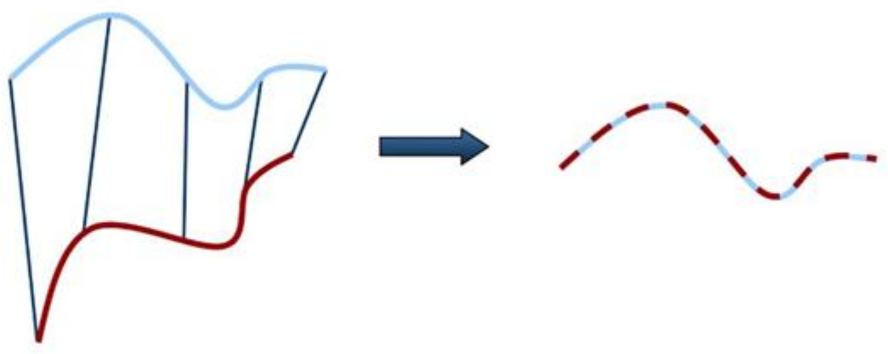


Figure 2.13: Transformation between blue line and red line.

In subsequent research, many researchers improved the ICP algorithm and proposed many variants of the ICP algorithm. For instance, Chen er al. [29] and Bergevin et al. [30] proposed an accurate registration method for point-to-plane search for nearby points. Rusinkiewicz [31] proposed weighted point-to-plane error metric to assign a different weight to each correspondence. In addition, Andrew and Sing [32] extracted a data registration method based on the texture information of the color 3D scan data points, and mainly considered the texture color information of the 3D scan points in the ICP algorithm to search for nearby points. Natasha et al. [33] analyzed the quality of point cloud data registration in the ICP algorithm.

Although there are many variants of the ICP algorithm, they are all optimizations of the classic point-to-point ICP algorithm, so the following is an introduction to the theory of ICP algorithm. The theoretical basis of other variants of ICP algorithm is similar to the theory to be introduced.

Suppose we have two sets of point clouds: Source point cloud P and target point cloud Q, their expressions are as follows:

$$P = \{p_1, p_2, p_3, \dots p_n\}, Q = \{q_1, q_2, q_3, \dots, q_n\}$$

Then we need to calculate the corresponding near point of each point in Q in the P point set. In order to speed up the search process, the KD-tree algorithm can be used in this step to speed up the search. Compared with ordinary brute force search, KD-tree search has higher search efficiency.

After finding the corresponding near point, we need to find a rotation matrix R and transformation matrix T to minimize the following error function:

$$E(R,T) = \frac{1}{n}\sum_{i=1}^{n} ||p_i - q_i||^2$$

Bring the obtained transformation and rotation matrix into the point cloud P, we can get a new set of point cloud P.

If the distance between the new point cloud set P and the reference point set Q is less than a given threshold, the iterative calculation is stopped, otherwise the new transformation point set will continue to iterate as a new P until it reaches the requirement of the objective function.

Through this continuous iterative method, we can find the transformation matrix and rotation matrix that minimize the distance between the two sets of point clouds, so as to achieve the purpose of accurately registering the two sets of point clouds.

The original ICP algorithm occupies too much computing resources, is sensitive to the initial transformation, and easily falls into a local optimal solution. Since the introduction of ICP, there have been quite a few ICP improved algorithms, and a few are briefly listed:

(1). Point-to-Plane ICP[29]: The point-to-point distance used in the cost function of the original ICP algorithm. Point-to-plane considers the distance from the source vertex to the surface where the target vertex is located. Compared with calculating the point-to-point distance directly, taking into account the local structure of the point cloud, the accuracy is higher, and it is not easy to fall into the local optimum; but it should be noted that the point-to-plane optimization is a nonlinear problem, the speed is relatively slow, and its linear approximation is usually used;

(2). Plane-to-Plane ICP[41]: Point-to-plane only considers the local structure of the target point cloud, plane-to-plane, as the name implies, also considers the local structure of the source point cloud, and calculates the distance from the surface to the surface;

(3). Generalized ICP[42]: Considering the point-to-point, point-to-plane and plane-to-plane strategies comprehensively, the accuracy and robustness are improved;

(4). Normal ICP[43]: Considering the normal vector and local curvature, and further using the local structure information of the point cloud, the experimental results in the paper have better performance than Generalized ICP.

2.7 Hough Voting

Hough voting is a popular computer vision technique designed for sparse collections, so it is naturally suitable for point clouds. It's first introduced by Tombari [34]. In his paper, he proposed a classic method of using Hough Voting ideas for target recognition in three-dimensional scenes, and achieved good results in cluttered scenes and occlusions. This idea has been cited many times in scientific papers in recent years, and some deep learning methods also have the shadow of this voting idea.

The algorithm uses the feature descriptors of the point cloud to calculate a series of matching pairs of model points and scene points. Firstly we need the feature points of the model and scene, which could be computed by FPFH descriptor or SHOT descriptor. At the same time, each model feature point has a relative positional relationship with the model centroid, model centroid is the red circle in Figure 2.14.

Then the vector between each feature and the centroid is computed and stored, which is the blue arrows in Figure 2.14. The matched scene feature points can correspond to the position of a centroid, and the position information is used to vote related parameters in the Hough space. If enough features vote for the presence of the object in a given position of the 3D space, then the object is detected and its pose is determined by means of the computed correspondences.

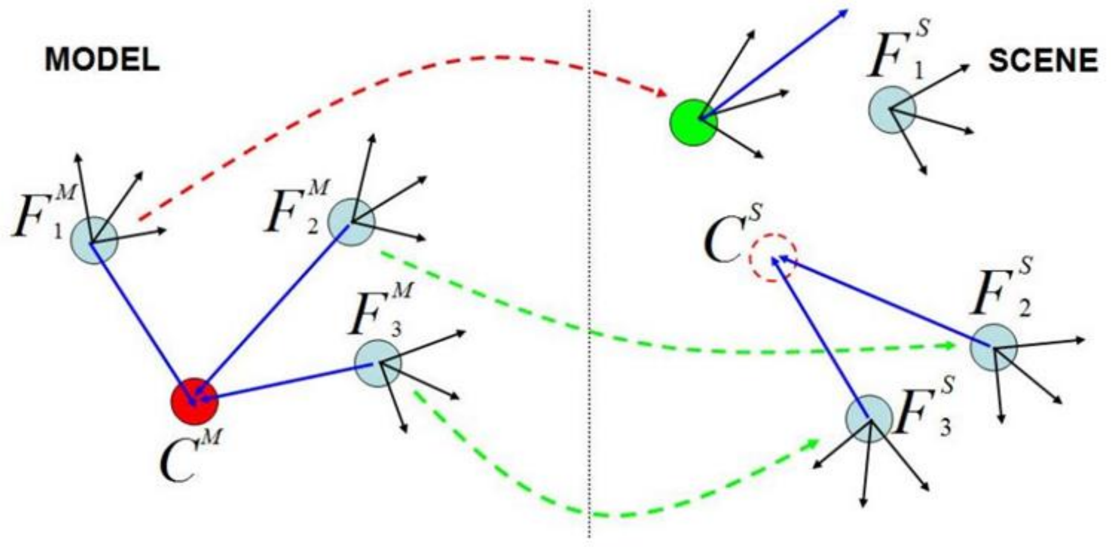


Figure 2.14: Example of 3D Hough Voting based on local reference frame.

In addition, we want this method to be rotation and translation invariant, so the vector between each feature and the centroid can't be stored in the coordinates of the global reference frame. Hence, as sketched in Figure 2.15[35], the local reference frame need to be computed both in the model and in the scene. In particular, because we need to compute local reference frame for each feature, so the reference frame has to be efficiently computable and very robust to disturbance factors.

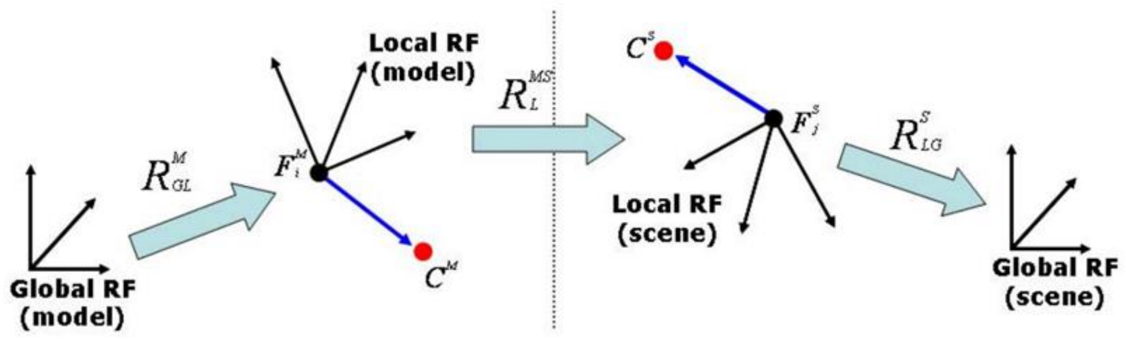


Figure 2.15: Transformations induced by the use of local reference frame

In summary, this method can be used in correspondence grouping in point cloud registration. We selected the bin in the Hough space having the maximum number of votes, only one instance of object can be sought from scene.

3 Methodology

3.1 Overview

The workflow of the proposed point cloud registration is illustrated in Figure 3.1.

The first step is a preprocessing of the raw point cloud. The scene point cloud is filtered to remove most of noise, such as outliers points. And some details is trimmed by using software CloudCompare. Then the resolution of the model point cloud and the scene point cloud is calculated, because resolution is a very important parameter in registration. After that, the voxelgrid downsampling is applied to reduce the amount of data and adjust the resolution of model and scene to the same.

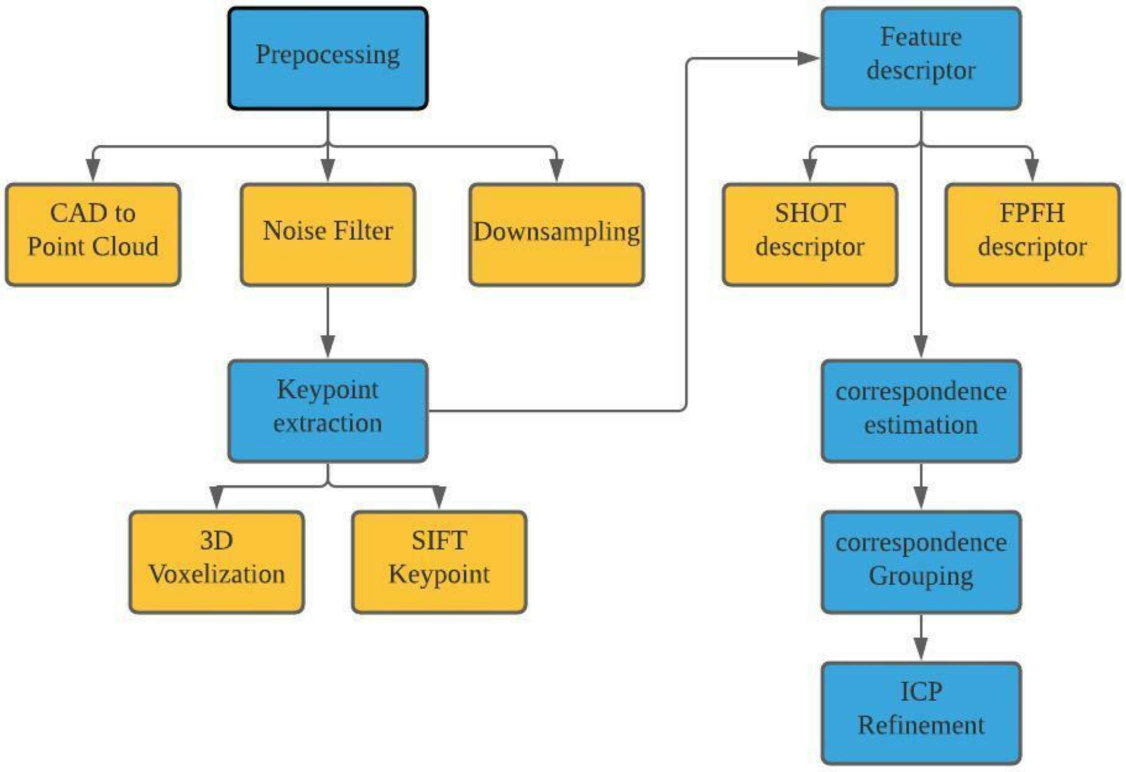


Figure 3.1: The workflow of point cloud registration

The second step is key point extraction both in model and scene. PCL implements many key point detectors such as Harris corner detector, Intrinsic Shape Signature, SIFT Keypoint, 3D voxelization and so on. In this thesis, the SIFT Keypoint and 3D voxelization are applied to extract the key points.

In third step, feature descriptor is computed for each detected keypoint. We use local descriptor in this step because it try to resemble shape and appearance only in a local neighborhood around a point and thus are very suitable for representing it in terms of matching. Hence, FPFH descriptor and SHOT descriptor are used to compute feature descriptor.

The next step is to find corresponmdence between model and scene based on searching similar feature descriptor. K-d tree is applied for rapid searches. This data structure has logarithmic search times but take longer to initialize. For this purpose, PCL depends on FLANN [10], an open-source library for fast nearest neighbor searches.

As a result of the matching stage, correspondences are determined by associating pairs of model-scene descriptors that lie close in the descriptor space. Next we need to make a correspondence grouping, to find out instance from scene. For this purpose, hough voting algorithm is used to get the correspondence grouping. In this step, the rotation and translation matrix between model and instance from scene should be calculated out.

The final step is the refinement using ICP algorithm. Through the ICP, we can further register the model and the instances found in the scene. So that we can get an accurate transformation relationship between the model and the instance from scene.

3.2 Toolkit

3.2.1 Point cloud library

PCL (Point Cloud Library) is a large cross-platform open source C++ programming library established on the basis of absorbing previous point cloud related research. It implements a large number of general point cloud related algorithms and efficient data structures, involving point cloud acquisition, filtering, segmentation, registration, retrieval, feature extraction, recognition, tracking, surface reconstruction, visualization, etc. As shown in the Figure3.2, the PCL architecture diagram for 3D point cloud processing, It supports multiple operating system platforms, and can run on Windows, Linux, Android, MacOS, and some embedded real-time systems. PCL is a BSD authorization method that can be used for commercial and academic applications for free.

PCL is completely a modular modern C++ template library. It is based on the following third-party libraries:

Boost: This set of C++ libraries is used for threading and mainly for shared pointers, so there is no need to re-copy data that is already in the system.

Eigen: It is an open-source template library for linear algebra (matrices, vectors). Most mathematical operations in PCL are implemented with Eigen.

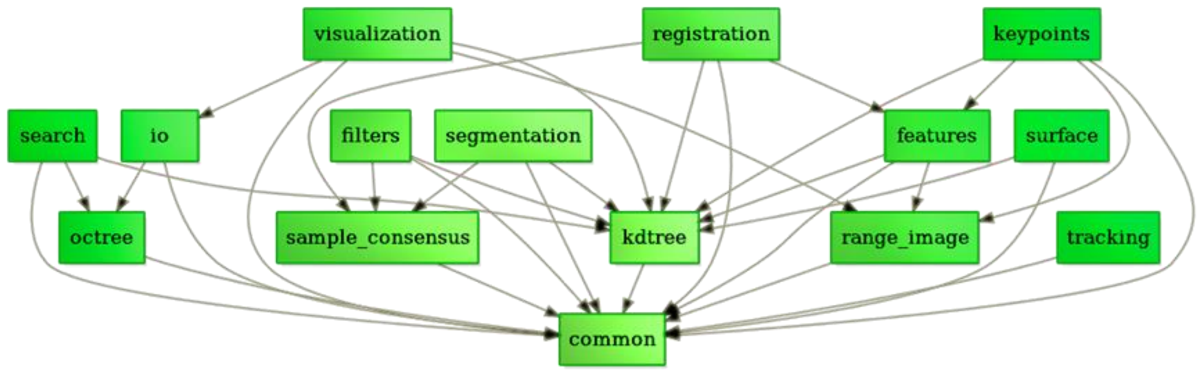


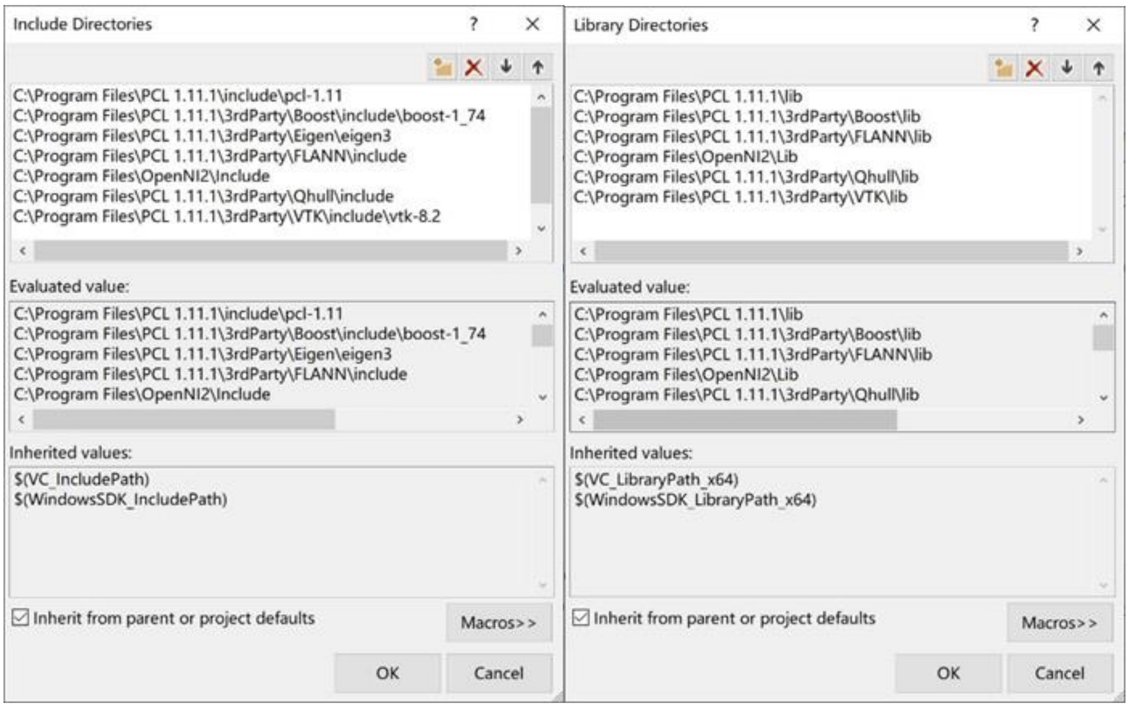
Figure3.2: Applications and relationships of each algorithm in PCL

FLANN: It is a library that performs a fast approximate nearest neighbor search in high dimensional spaces. In PCL, it is especially important in the kd-tree module for fast k-nearest neighbor search operations.

VTK: Used in visualization module for point cloud rendering and visualization.

OpenNI: It is used to retrieve point clouds from devices.

In this thesis, the PCL version is PCL1.11.1. We downloaded the PCL code library to the local computer, and completed the relevant configuration through visual studio. After that, we can call the functions provided in PCL when developing with visual studio. Figure 3.3 shows some configuration pages in visual studio.



Include Directories

```
C:\Program Files\PCL 1.11.1\include\pcl-1.11
C:\Program Files\PCL 1.11.1\3rdParty\Boost\include\boost-1_74
C:\Program Files\PCL 1.11.1\3rdParty\Eigen\eigen3
C:\Program Files\PCL 1.11.1\3rdParty\FLANN\include
C:\Program Files\OpenNI2\Include
C:\Program Files\PCL 1.11.1\3rdParty\Qhull\include
C:\Program Files\PCL 1.11.1\3rdParty\VTK\include\vtk-8.2
```

Evaluated value:

```
C:\Program Files\PCL 1.11.1\include\pcl-1.11
C:\Program Files\PCL 1.11.1\3rdParty\Boost\include\boost-1_74
C:\Program Files\PCL 1.11.1\3rdParty\Eigen\eigen3
C:\Program Files\PCL 1.11.1\3rdParty\FLANN\include
C:\Program Files\OpenNI2\Include
```

Inherited values:

```
$(VC_IncludePath)
$(WindowsSDK_IncludePath)
```

☑ Inherit from parent or project defaults

Library Directories

```
C:\Program Files\PCL 1.11.1\lib
C:\Program Files\PCL 1.11.1\3rdParty\Boost\lib
C:\Program Files\PCL 1.11.1\3rdParty\FLANN\lib
C:\Program Files\OpenNI2\Lib
C:\Program Files\PCL 1.11.1\3rdParty\Qhull\lib
C:\Program Files\PCL 1.11.1\3rdParty\VTK\lib
```

Evaluated value:

```
C:\Program Files\PCL 1.11.1\lib
C:\Program Files\PCL 1.11.1\3rdParty\Boost\lib
C:\Program Files\PCL 1.11.1\3rdParty\FLANN\lib
C:\Program Files\OpenNI2\Lib
C:\Program Files\PCL 1.11.1\3rdParty\Qhull\lib
```

Inherited values:

```
$(VC_LibraryPath_x64)
$(WindowsSDK_LibraryPath_x64)
```

☑ Inherit from parent or project defaults

Figure 3.3: Configuration pages in visual studio

3.2.2 Autodesk Fusion 360

Fusion 360 is a three-dimensional visual modeling software launched by Autodesk Inc. in the United States. It integrates industrial design, mechanical design, collaboration, processing and other elements into one. Fusion 360 is also a cloud-based 3D modeling, CAD, CAM, CAE, and PCB software platform for product design and manufacturing. It supports .dwg .obj .stl file format. The more popular technologies in the software include direct modeling technology, T-spline modeling technology, connection-based assembly technology, top-down parametric modeling technology, cloud data management, etc. For students, teachers and educational institutions, it is a free software, and other commercial uses need to be paid for.

Because when converting the CAD model to the point cloud model, we need to use the .obj format file, so we use Autodesk Fusion 360 to convert the .dwg file format to the .obj file format, which is shown in Figure3.4.

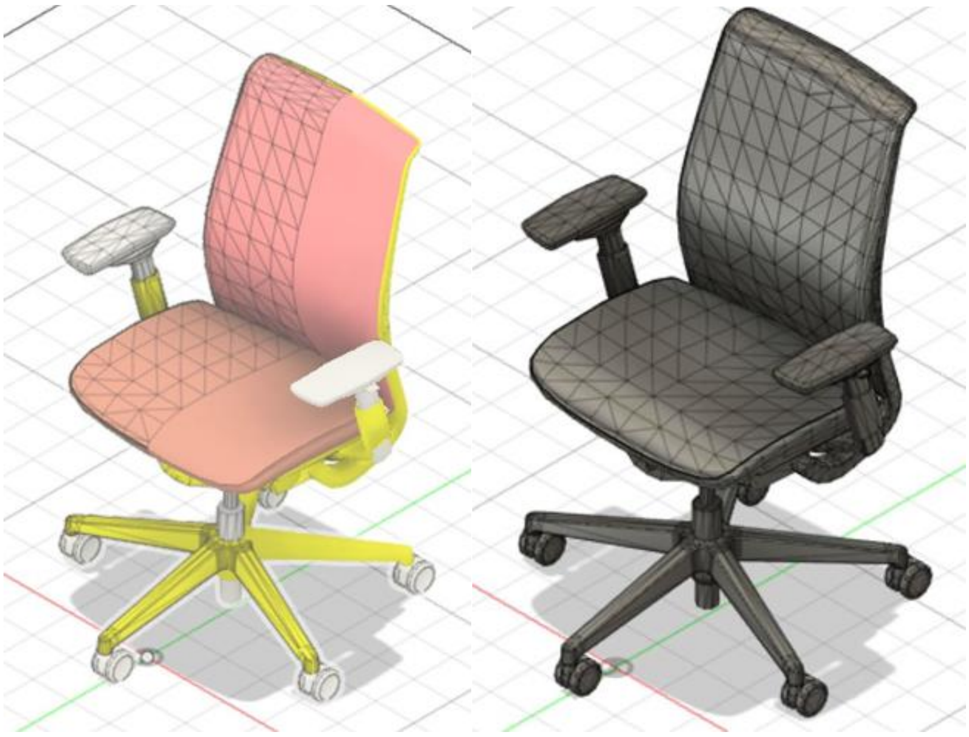
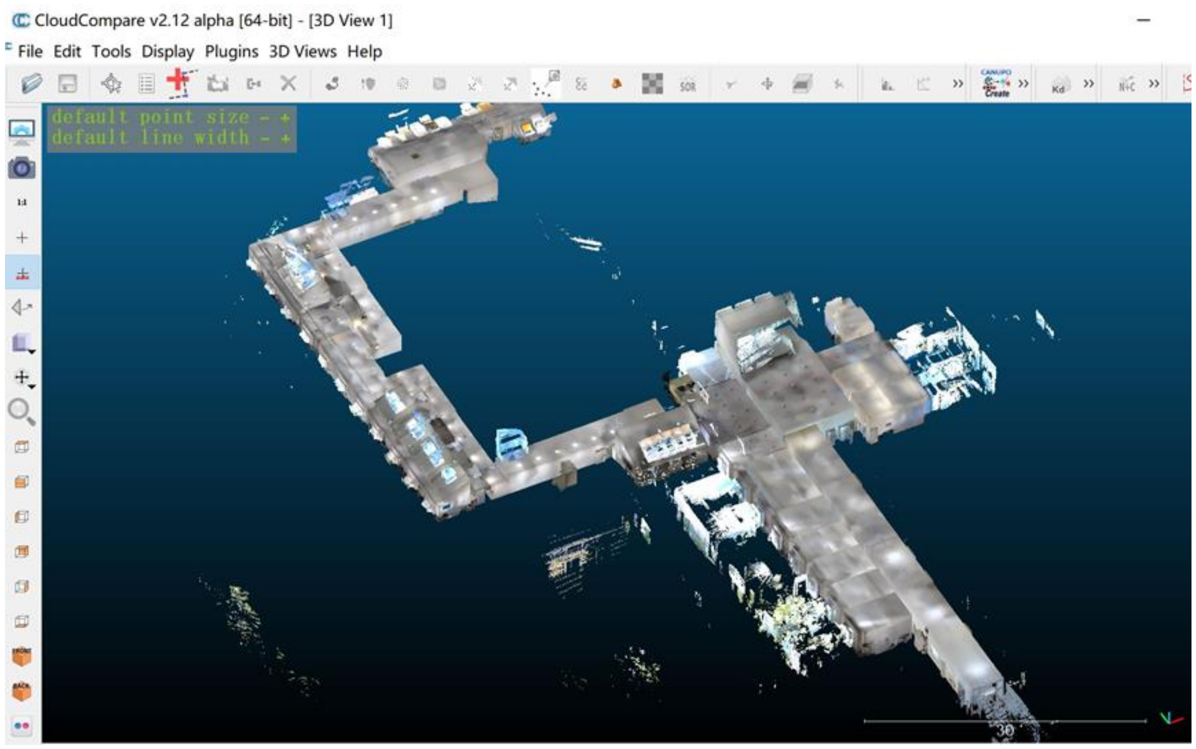


Figure 3.4: .dwg CAD model converted to .obj file format in Fusion 360

3.2.3 CloudCompare

CloudCompare is an open source 3D point cloud processing software. It has been originally designed to display large point cloud and perform comparison between two dense 3D points clouds. It relies on a specific octree structure dedicated to this task. Afterwards, many advanced algorithms has been included into CloudCompare (registration, resampling, color/normal vector/scale, statistical calculation, sensor management, interactive or automatic segmentation, etc.). It is also a display enhancement tools (custom color gradient, color and normal vector Processing, calibration image processing, OpenGL shaders, plug-ins, etc.). In this study, we mainly use this software to display the result after point cloud prepocessing. We also use it to adjust the scale of model and scene, trim some details of model, or separate part of the point cloud from the big point cloud.


CloudCompare v2.12 alpha [64-bit] - [3D View 1]
File Edit Tools Display Plugins 3D Views Help
default point size - +
default line width - +

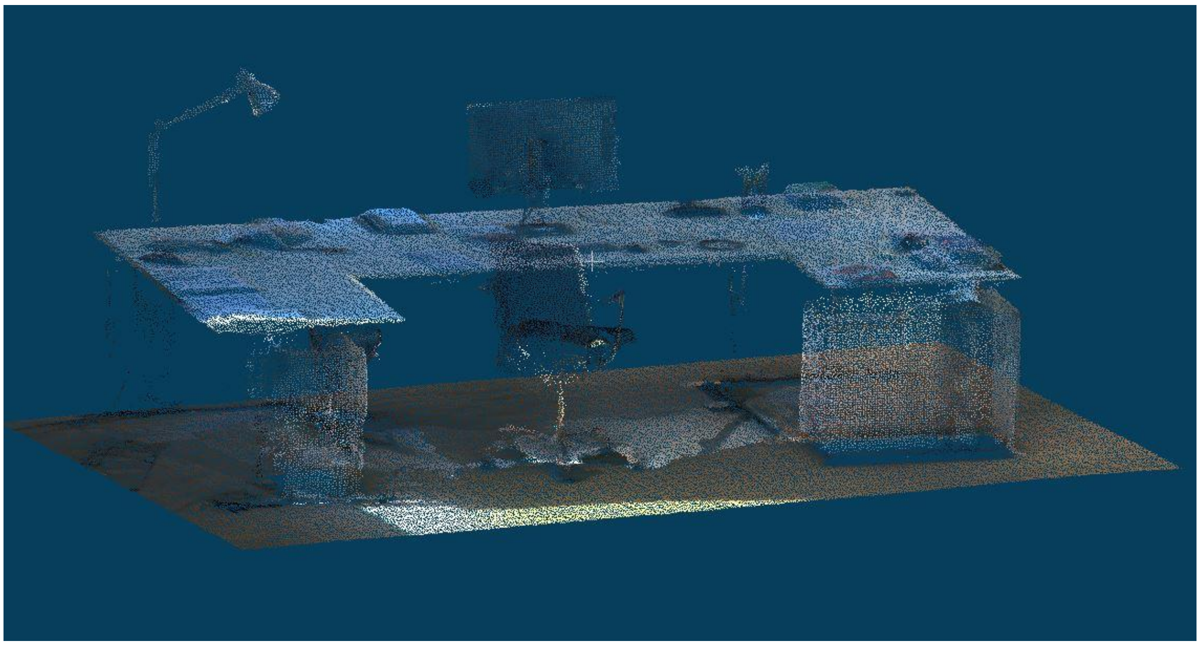


Figure 3.5: point cloud display in CloudCompare

3.3 Prepocessing

3.3.1 Noise Filter

Point cloud collection can be collected by equipment such as RGBD camera or lidar. Due to the accuracy of the collection equipment, environmental factors, lighting factors, and surface properties of the object, noise will inevitably appear in the point cloud data. The filtering process is to solve the problems of irregular and unsmooth point cloud data density, outliers, and noisy data.

In this thesis, we use the *StatisticalOutlierRemoval* provided by PCL to remove the outlier points in scene. Statistical analysis techniques are used to centrally remove measurement noise points from a point cloud data. Perform statistical analysis on the neighborhood of each point, and eliminate the neighborhood points that do not meet certain standards.

For each point, calculate its average distance to all adjacent points. Assuming that the distribution obtained is Gaussian, we can get a average value μ and a standard deviation σ.

All points in this neighborhood point set whose distance from its neighborhood is greater than $\mu + std\_mul * \sigma$ outside the interval can be regarded as outliers and can be removed from the point cloud data. std_mul is a threshold of multiples of standard deviation, which can be specified by yourself.

Here is the corresponding code segment:

```
pcl::StatisticalOutlierRemoval<pcl::PointXYZ> filters;
filters.setInputCloud(cloud);
filters.setMeanK(10);
filters.setStddevMulThresh(1.0);
filters.filter(*filtered);
pcl::io::savePCDFile("afterfilter.pcd", *filtered);
```

As shown in the code above, we first create a statistical analysis filter, and then input the cloud, which is the point cloud we want to process, and then set the number of adjacent points analyzed for each point to 10, and the multiple of standard deviation is set to 1.0, which means that if the distance of a point exceeds the average distance plus 1.0 times standard deviation, the point is marked as an outlier and it is removed. After statistical analysis and filtering, the output result is *cloud_filtered*. Finally, we store the result in a pcd file *afterfilter.pcd.*

3.3.2 Voxelgrid Downsampling

The Voxelgrid method is used to achieve downsampling, that is, to reduce the number of points, while maintaining the shape of the point cloud, which is very practical in improving the speed of registration, surface reconstruction, shape recognition and other algorithms.

The Voxelgrid class implemented by PCL creates a three-dimensional voxel grid (think of the voxel grid as a collection of tiny spatial three-dimensional cubes) through the input point cloud data. The center of gravity of all points in the voxel is approximated to display other points in the voxel, so that all points in the voxel are finally represented by a center of gravity point, and the filtered point cloud is obtained after processing all the voxels. This method is slower than the method of approximating the center of the voxel, but it is more accurate for the representation of the corresponding surface of the sample point.

Here is the corresponding code segment:

```
float leftSize = 0.02f;

pcl::VoxelGrid<pcl::PointXYZ> downsample;

downsample.setInputCloud(cloud);

downsample.setLeafSize(leftSize, leftSize, leftSize);

downsample.filter(*downsampling);

pcl::io::savePCDFile("afterfilter.pcd", *downsampling);
```

As shown in the code above, we first set the size of grid to 0.02, this parameter needs to be adjusted according to the density of the point cloud to ensure that a voxel grid of appropriate size is generated. Then we create a voxelgrid filter and input the cloud, which is the point cloud we want to process. After downsampling, the output result is $cloud\_filtered$. Finally, we store the result in a pcd file $afterfilter.pcd$.

After downsampling the point cloud, the amount of data has been significantly reduced. Taking the point cloud used in the experiment as an example, the number of point clouds has been reduced from 177992 to 55917. We can also intuitively see the point cloud comparison before and after processing from the Figure3.6.

From the visualization results, it can also be clearly seen that the density of points is different from the degree of neatness. Although the amount of data is greatly reduced after processing, it is obvious that the shape features and spatial structure information contained in it are similar to the original point cloud.

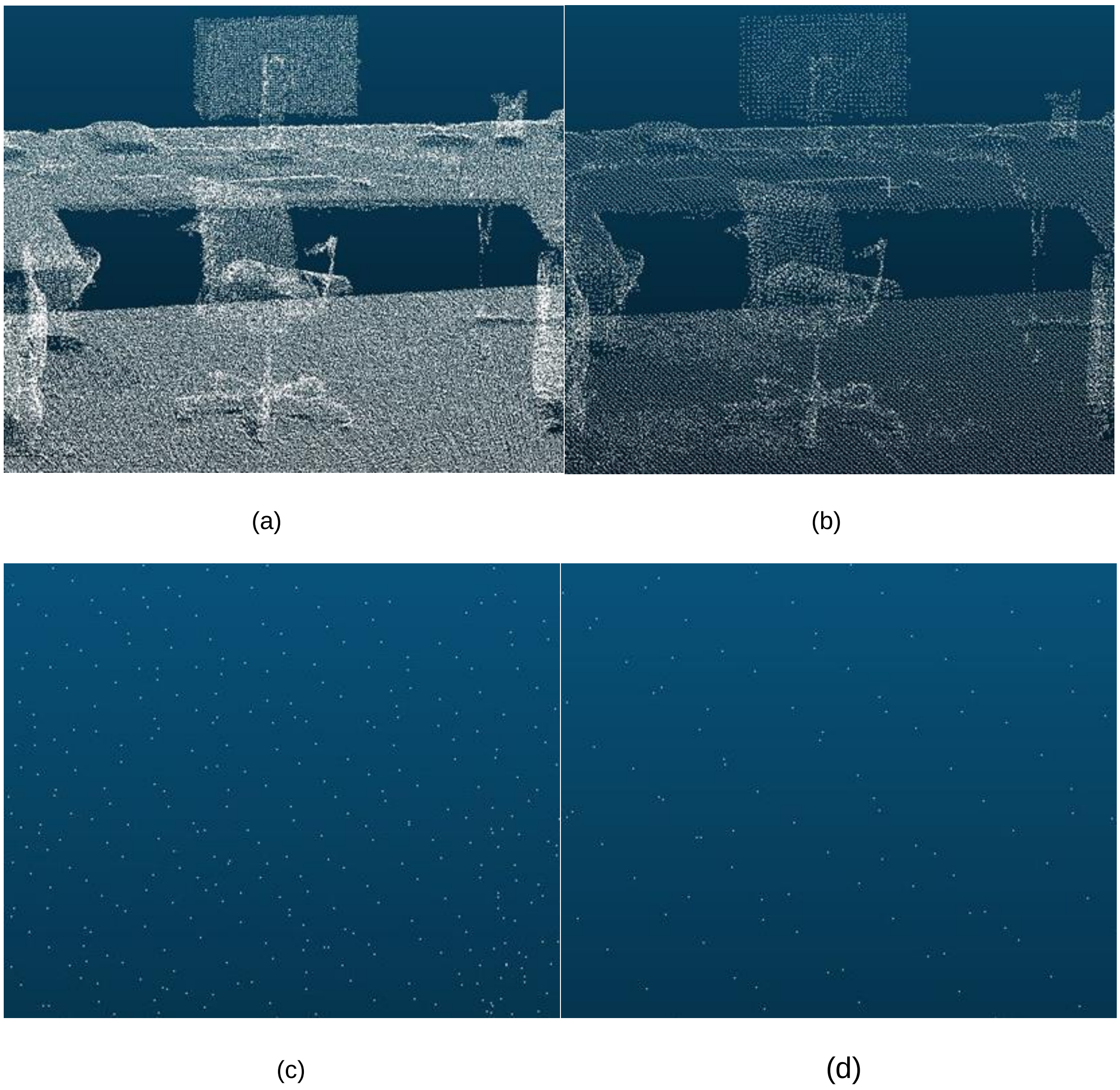


Figure 3.6: visualization result of point cloud downsampling: (a) original point cloud section; (b) downsampled point cloud; (c) smaller section of the original point cloud; (d) smaller section of downsampled point cloud.

3.4 Keypoint extraction

3.4.1 3D Voxelization

The same as the above mentioned downsampling method, by generating multiple spherical grids in the point cloud, each grid contains a certain number of points, and extracting the center of gravity of the grid as a key point. As shown in Figure 3.7, blue points are key points extracted from point choud. In simple experimental scenarios, this method is practical and efficient, and it can better maintain the shape characteristics of the original point cloud.

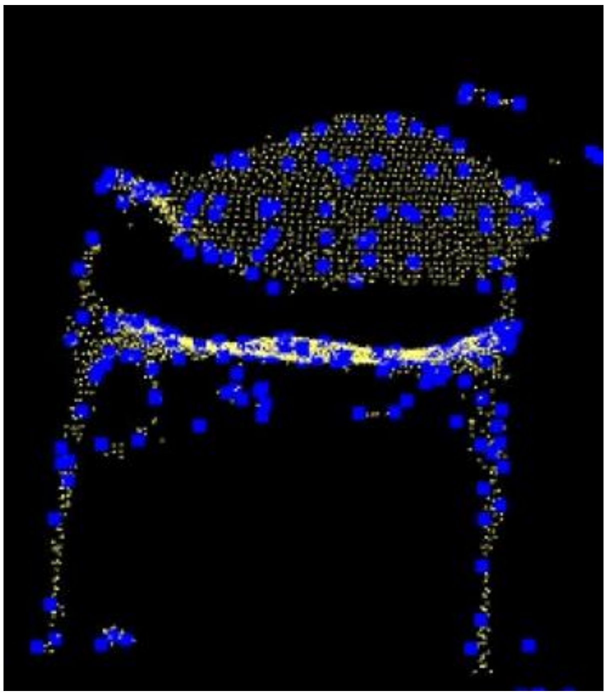


Figure 3.7: Key points extracted by 3D voxelization

3.4.2 SIFT keypoint

According to the introduction in the chapter II, we can know that the premise of locating and extracting sift keypoints is to construct a Difference of Gaussians(DOG) scale space, in order to further find extreme points in the Gaussian scale space. Each point must be compared with all neighboring points in its point domain (same scale space) and scale domain (adjacent scale space). When it is greater than (or less than) all adjacent points, the point is an extremum point.

Because the local extremum points of the DOG come from a discrete space, so the extreme point found in the discrete space is not necessarily the true extreme point. So we must try to eliminate the points that do not meet the conditions, such as low-contrast keypoints.

The class SIFT Keypoint is the implementation of SIFT keypoint that transplants the SIFT keypoint from the 2D image to the 3D space after adjustment. The input is a point cloud with XYZ coordinates, and the output is the SIFT keypoints.

Here is the corresponding code segment:

```
pcl::SIFTKeypoint<pcl::PointXYZRGBA, pcl::PointWithScale> sift;

pcl::PointCloud<pcl::PointWithScale> result_model;

pcl::PointCloud<pcl::PointWithScale> result_scene;

pcl ∷ search ∷ KdTree<pcl ∷ PointXYZRGBA> ∷ Ptr  tree(new  pcl ∷ search ∷
KdTree<pcl∷PointXYZRGBA>());

sift.setScales(min_scale, n_octaves, n_scales_per_octave);

sift.setSearchMethod(tree);

sift.setMinimumContrast(min_contrast);

sift.setInputCloud(model);

sift.compute(result_model);

pcl::copyPointCloud(result_model, *model_keypoints);
```

We interpret some of the key functions:

(1). sift.setScales(min_scale,n_octaves,n_scales_per_octave);

Set the parameters related to the scale when searching, min_scale is the standard deviation in the point cloud voxel scale space, the point cloud corresponds to the minimum size of the voxel in the voxel grid, $nr\_octaves$ is the number of voxel space scales when detecting key points, $n\_scales\_per\_octave$ is the parameter required to calculate the Gaussian space scale at each voxel space scale.

(2). sift.setSearchMethod(tree);

Specify the search method as KD-tree search, create an empty KD-tree object, and pass it to the sift detection object.

(3). sift.setMinimumContrast(min_contrast);

Set the lower limit of contrast that candidate key points, to eliminate the extremum points that do not meet the conditions.

Finally, we store the results in $result\_model$, and convert point type $pcl::PointWith Scale$ data to point type $pcl::PointXYZ$ data. And we store the converted result in $model\_keypoints$.

As shown in the Figure3.8, the input is a point cloud of a chair. After processing, the SIFT Keypoint are successfully extracted. The blue points in the figure are the key points.

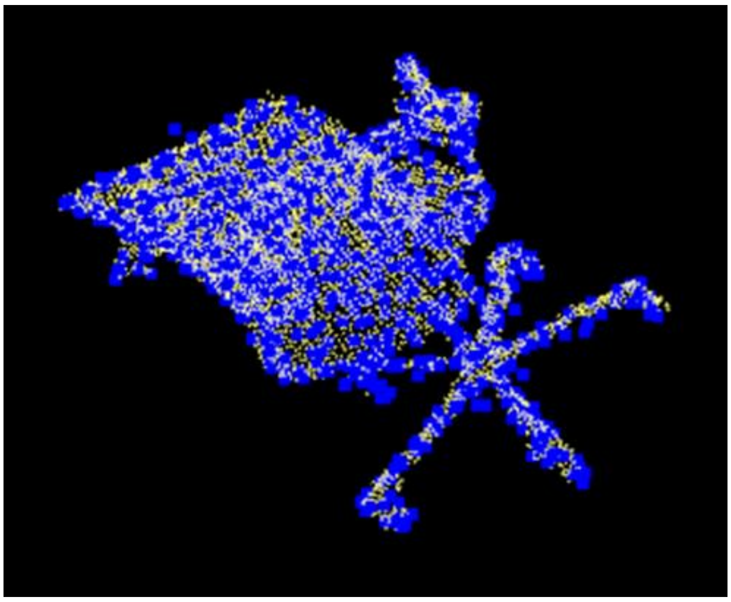


Figure 3.8: SIFT Keypoint extract from a chair point cloud

3.5 Compute feature descriptor

3.5.1 SHOT descriptor

Before this, we have successfully extracted the key points from the model point cloud and the scene point cloud. In the next step, we need to calculate the feature descriptors of the key points.

Based on the introduction in Chapter II, SHOT descriptor counts the topological features around the feature points in the constructed local reference system, saves the features in the histogram, and normalizes them. According to this concept, first we have to construct a local reference system for each key point. Then we generate a spherical support area for each key point. According to the constructed local coordinate system, we can divide the support area to 32 small area, which is radial direction(inner and outer), latitude direction(northern and southern hemisphere) and longitude direction(eight area). In this way, a spherical support area can be divided into 32 small areas. For each small area, construct a local histogram, vote based on the cosine value of the angle between the normal vector of the feature point and the local reference frame z-axis. Divide each local histogram into 11 bins, then the SHOT descriptor has 352 dimension. It should be noted that because the local reference system is used, boundary problems will inevitably occur. In order to solve this problem, SHOT descriptor adopted quadrilinear interpolation weaken the boundary effect.

The encoding of the SHOT descriptor is very difficult, but fortunately the PCL provides the implementation of the SHOT descriptor, here is the code segment:

```
pcl∷NormalEstimationOMP<PointType, NormalType> normal;
normal.setKSearch(10);
normal.setInputCloud(model);
normal.compute(*normals);
pcl∷SHOTEstimationOMP<PointType, NormalType,DescriptorType>shot;
shot.setRadiusSearch(0.08f);
shot.setInputCloud(model_keypoints);
shot.setInputNormals(normals);
shot.setSearchSurface(model);
shot.compute(*model_descriptors);
```

As shown in the code above, we first need to compute the normals for each point of both the model and the scene cloud with the $pcl :: NormalEstimationOMP <$ $pcl::NormalEstimationOMP>$ estimator, using the 10 nearest neighbors of each point. Then in order to generate the SHOT descriptor object, we use the implementation $pcl :: SHOTEstimationOMP < PointType, NormalType, DescriptorType >$. Then we set the radius for descriptor computation, the radius that defines the support area on which the descriptor is computed (i.e. the subset of keypoint neighbors being described). It should be enough large to include at least a few tens of points, but not too large not to include clutter in the description of keypoints close to the object border.

Usually a value between 10-20 times of point cloud resolution does the job. Then we input the key points and normal vectors, and use $setSearchSurface$ function to input neighborhood area at its original point cloud. In the end, SHOT descriptor is computed and stored in vector $model\_descriptors$.

3.5.2 FPFH descriptor

Surface normal and curvature estimation are the basic representation methods of geometric features around a certain point. Although it is relatively easy to calculate, there is not much information that can be provided. Because they only use a few parameter values to approximate the k-neighborhood geometric features of a point. In most scenarios, there will be many identical or similar feature values, so only using point features will reduce a lot of global feature information. So we can collect global feature information through the Fast Point Feature Histogram(FPFH).

According to the introduction in Chapter II, FPFH forms a multi-dimensional histogram to describe the geometric characteristics of the k-neighborhood of the point by parameterizing the spatial difference information between the query point and the neighboring point. The high-dimensional hyperspace where the histogram is located provides a measurable information space for the representation of the feature, which makes the 6DOF (degree of freedom) attitude of the point cloud invariant, and it has good robust at different sampling densities or neighborhood noise levels.

First of all, we find the three characteristic element values between query point and its k neighborhood, and then calculate it into a SimplePFH. Then we determine the k-neighborhood for each point in the k-neighborhood respectively, and form its own SPFH according to the first step. Finally, weighted statistics on each SPFH in the neighborhood is performed, histogram can be calculated out.

According to the concept of FPFH descriptor and progress of construction, PCL provides implementation in $pcl\_feature$ library. The default FPFH implementation uses 11 binning subdivisions and a decorrelated scheme which results in a 33-byte array of float values. These are stored in a $pcl::FPFHSignature33$ point type.

For users with demanding calculation speeds, PCL provides another implementation of FPFH estimation, which uses multi-core/multi-thread specifications and the OpenMP development model to improve calculation speed. The name of this class is $pcl::FPFHEstimationOMP$, and its application programming interface (API) is 100%

compatible with single-threaded $pcl::FPFHEstimation$, which makes it suitable as a replacement component. In an 8-core system, the implementation of OpenMP can perform the same calculation on a single-core system in 6-8 times faster computing time.

Here is the corresponding code segment:

```
pcl::FPFHEstimation <pcl::PointXYZ, pcl::Normal, pcl::FPFHSignature33>fpfh;

fpfh.setInputCloud (cloud);

fpfh.setInputNormals (normals);

fpfh.setSearchMethod (tree);

pcl::PointCloud<pcl::FPFHSignature33>::Ptr fpfhs (new pcl::PointCloud

<pcl::FPFHSignature33> ());

fpfh.setRadiusSearch (0.1);

fpfh.compute (*fpfhs);
```

As shown in the code above, first we create the FPFH estimation class, and pass the input cloud and normal to it. Then we create an empty KD-tree representation, and pass it to the FPFH estimation object. Its content will be filled inside the object, based on the given input dataset. Then we set the radius of sphere with all neighbors is 10cm, this radius should be larger than the radius used to estimate the surface normal. Finally, the feature is computed.

3.6 Correspondence estimation

Before that, no matter which feature descriptor computation method is used, we should already get the feature descriptor of model and scene. Now we need to determine point-to-point correspondences between model descriptors and scene descriptors. This is the process of pairing points p_i from the source point cloud P to their closest neighbors q_j in the target cloud Q. We need to search for each point in the target point cloud to find a feature descriptor similar to the corresponding point of the origin point cloud.

A primary approach of searching for the nearest neighbor is to perform an exhaustive search through all the points in target point cloud. However, PCL provides us with a better way, we cloud use K-d tree data structure to make a rapid search. There is a third party FLANN included in PCL, by using the FLANN library, we can directly use the k-d tree data structure in the code.

Here is the code about correspondence estimation

```
pcl::CorrespondencesPtr findcorrespondence(new pcl::Correspondences());
pcl::KdTreeFLANN<DescriptorType> match;
match.setInputCloud(model_descriptors);
for (std::size_t i = 0; i < scene_descriptors->size(); ++i)
{……
if (!std::isfinite(scene_descriptors->at(i).descriptor[0]))
continue;
if (found_neighs == 1 && neigh_sqr_dists[0] < 0.25f)
 findcorrespondence->push_back(correspondence);
  ……}
```

As shown in the code above, we first set a vector of correspondence estimation, which is named by $model\_scene\_corrs$, then we use the method of corresponding point searching is KD- tree nearest neighbor region. Descriptor of model as the input data for KD-tree search. Then we can find the closest point corresponding to the model descriptor in the scene descriptor, and set an appropriate matching threshold according to the resolution of the point cloud and the radius of the field set by the previous SHOT descriptor computation. Finally, we store the matching corresponding pairs points in the vector $model\_scene\_corrs$.

3.7 Correspondence Grouping

In the previous step, correspondences are determined by searching pairs of model and scene descriptors that lie close in the descriptor space. However, most of the correspondence between model and scene descriptors contained in the correspondence vector does not exist in the instance from scene we want to find. Hence, we need to filter these descriptor pairs, to find out the correspondence from instance in scene. A relatively common approach within 3D point cloud registration methods is usually referred to as correspondence grouping.

In this step, we have to classify initial feature correspondences between two 3D point clouds obtained by matching local geometric descriptors into inliers and outliers, inliers means that these descriptor comes from model instance in scene which we want. Due to a number of factors, such as key points localization errors, and point cloud noise, limited overlap, clutter and occlusion, heavy outliers are generated in the initial correspondence set. Hence, it's very challenging to find the suitable group.

Existing 3D correspondence grouping methods can be divided into two categories: group-based and individual-based[36]. Group-based method assume that inliers combine a cluster in a particular domain and try to recover the cluster. In contrast, individual-based first assign scores to correspondence based on feature distance and constraints of geometry, then select the top-scored correspondence.

Most commonly group-based methods are Random sampling consensus, Game theory matching, Geometric consistency, Hough voting. Random sampling consensus iteratively estimates a model from correspondences and verifies its rationality. Game theory matching use spectral analysis and dynamic evolution on the affinity matrix to calculate out the inlier cluster. Geometric consistency forms a cluster for each correspondence by ensuring correspondences in the cluster are compatible with the query correspondence; the cluster with the maximum element count is served as the inlier cluster[37]. Hough voting transforms correspondences to 3D points in a 3D Hough space and then finds the cluster in Hough space[34].

In this thesis, we perform Hough voting to make correspondence grouping. We have introduced the theory of Hough voting in Chapter II, the important process of Hough voting is: first we need to set the local reference frame both for model and scene, since we want the method could be rotation and translation invariant. Then the hough space is created, voting will be conducted in hough space.

Here is the code segment about Hough voting:

pcl::Hough3DGrouping<PointType, PointType, RFType, RFType> hough;

hough.setHoughBinSize(size);

hough.setHoughThreshold(threshold);

hough.setUseInterpolation(true);

hough.setUseDistanceWeight(false);

hough.setInputCloud(model_keypoints);

hough.setInputRf(model_rf);

hough.setSceneCloud(scene_keypoints);

hough.setSceneRf(scene_rf);

hough.setModelSceneCorrespondences(correspondence);

hough.recognize(rototranslations, clustered_corrs);

As before, PCL provides us the implementation of Hough voting, we can use several function in PCL directly. *pcl::Hough3DGrouping* make a Hough voting class named hough, *hough.setHoughBinSize* set the sampling interval in Hough space, it defines the spatial length of each Hough bin. And it should be enough large to encompass oscillations due to noise of the 3D Hough votes but not too large to create spurious peaks in the Hough space. *hough.setHoughThreshold* set the threshold for the minimum number of votes to determine whether there is an instance in the Hough space. *hough.setUseInterpolation* set whether to interpolate the vote score between neighboring bins. *hough.setUseDistanceWeight* set whether the vote casting procedure uses the correspondence's distance as a score. *hough.setInputCloud* input the data into the Hough space. *hough.setInputRf* provide a pointer to the input dataset's reference frames, each point in the reference frame should be the reference frame of the correspondent point in the input dataset. *hough.setModelSceneCorrespondences* provide a pointer to the precomputed correspondences between points in the input dataset and points in the scene dataset. *hough.recognize* recognizes instances of the model from the scene. It also create a vector containing the correspondences for each instance of the model, and a vector containing one transformation matrix for each instance.

As sketched in Figure 3.9, we use an indoor environment as the scene point cloud, and a chair as the model point cloud, the correspondence grouping is found. The blue point is the key points extracted from point cloud and the green line is the correspondence between model and instance in scene, the red chair is the instance found in scene.

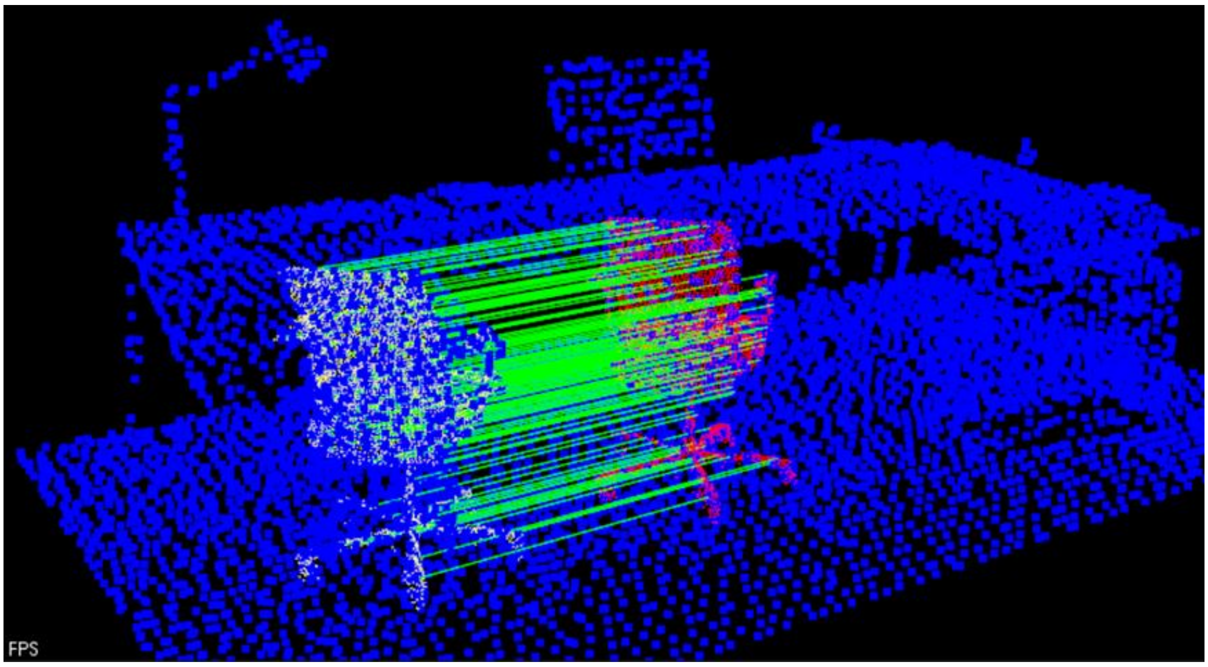


Figure 3.9: the result of correspondence grouping

3.8 Iterative closest point refinement

We have introduced the principle of Iterative Closest Point(ICP) algorithm in ChapterII, In point cloud registration, the ICP algorithm is a common algorithm for precise registration, which can calculate the nearest transformation of two point clouds. The ICP algorithm uses least squares estimation to calculate the transformation matrix. The principle is simple and has good accuracy. However, due to the iterative calculation, the calculation speed of the algorithm is slow. Moreover, when the ICP is used for the registration calculation, it has special requirements for the initial position of both point cloud. If the selected initial position is unreasonable, the algorithm will fall into a local optimum.

The selection of the initial position is very important. The correct initial position can improve the success rate of the ICP calculation, and it can also greatly improve the calculation speed of the ICP algorithm. However, in the previous step, we not only need to calculate the model instance in the scene point cloud, but also need to calculate the location information of the model. In the Hough Voting class, *hough.recognize* can create a vector containing the translation matrices and rotation matrices. Hence, these translation matrices and rotation matrices can be used as position information, as the initial transformation position of the ICP algorithm.

Regarding the implementation of the ICP algorithm, we can still find it from the PCL. And PCL provides a variety of functions to realize the multiple functions of the ICP algorithm. It should be noted that the ICP algorithm in PCL is implemented based on Singular Value Decomposition(SVD, an important method of matrix factorization in linear algebra).

In addition, before using ICP, a few parameters need to be set:

(1). setMaximumIterations:

ICP is an iterative method, before using this method, we should set the maximum number of iterations, when this number is reached, ICP stops iterating.

(2). setEuclideanFitnessEpsilon:

Set the maximum allowed Euclidean error between two consecutive steps in the ICP iteration, before the algorithm is considered to have converged. The error is estimated as the sum of the differences between correspondences in an Euclidean sense, divided by the number of correspondences.

(3). setTransformtionEpsilon:

Set the transformation epsilon (maximum allowable translation squared difference between two consecutive transformations) in order for an optimization to be considered as having converged to the final solution.

(4). setMaxCorrespondenaceDistance:

Set the maximum distance between corresponding point pairs (this value may have a greater impact on the registration result).

After setting the initial parameters, we can use the function about ICP in PCL to finely register the point cloud.

Here is the corresponding code segment:

```
std::vector<pcl::PointCloud<PointType>::ConstPtr> instance;
pcl::transformPointCloud(*model, *rotated_model, rototranslations[i]);
instance.push_back(rotated_model);
pcl::IterativeClosestPoint<PointType, PointType> icp;
icp.setMaximumIterations(max);
icp.setMaxCorrespondenceDistance(distance);
icp.setInputTarget(scene);
icp.setInputSource(instance[i]);
pcl::PointCloud<PointType>::Ptr registered(new pcl::PointCloud<PointType>);
icp.align(*registered);
```

As shown in the code above, *pcl::transformPointCloud* could obtain the position information of the translation matrices and rotation matrices of the model that has been roughly transformed in the previous step and pass the transformation to model as initial position. The model with the initial position can be called *rotated_model* and then stored in vector *instance. pcl::IterativeClosestPoint* create a class of ICP. First, we set the initial parameters of the ICP algorithm. Then we input the source point cloud and target point cloud that need to be matched, the source point cloud is the instance model with initial position, and the target point cloud is the scene point cloud. Finally, we create a vector to store the result after ICP registration.

4 Experiments and Results

4.1 Hardware Information

Since the subsequent efficiency and some restrictions are hardware-relevant, some of representative information are hereby listed briefly.

Point cloud data is collected by Navvis VLS, which is a wearable mobile scanning system suitable for high-quality reality capture in complex areas such as construction sites, stairs, and technical rooms.

The algorithm and code in this thesis are developed and tested on a laptop with a 64-bit Windows 10 operating system. The specifications of the used laptop are listed as follows:

CPU: Intel(R) Core(TM) i5-7267U CPU @ 3.10GHz

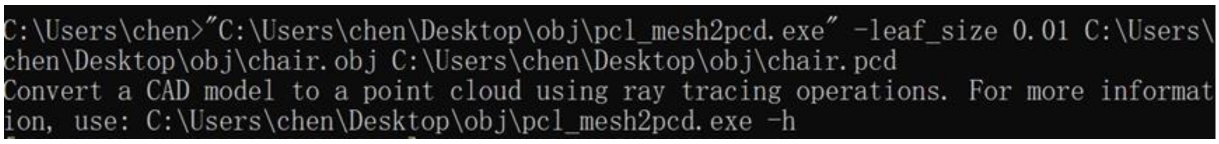
RAM: 8GB

GPU: Intel(R) Iris(R) Plus Graphics 650

4.2 Point Cloud Dataset

4.2.1 Model

The point cloud of model is converted from the CAD model. In order to be able to transform the CAD model into a point cloud model, we have tried a variety of methods. First of all, we used Cloudcompare for conversion. The Cloudcompare software contains tools to convert mesh to point cloud, but the conversion result is very unsatisfactory. The point cloud is very sparse and part of the point cloud is missing. So we used the surface sampling method provided in PCL. First, the file format .dwg is converted to the .obj format in Autodesk Fusion 360, and then the mesh surface is sampled by the method of surface sampling, *pcl_mesh2pcd.exe* in PCL is used. The code to call this program is shown in the Figure 4.1, where the *-leaf_size* parameter is the sampling density. The CAD models used in the experiment and the visualization of point cloud data after conversion are exhibited in Figure 4.2.



```
C:\Users\chen>"C:\Users\chen\Desktop\obj\pcl_mesh2pcd.exe" -leaf_size 0.01 C:\Users\
chen\Desktop\obj\chair.obj C:\Users\chen\Desktop\obj\chair.pcd
Convert a CAD model to a point cloud using ray tracing operations. For more informat
ion, use: C:\Users\chen\Desktop\obj\pcl_mesh2pcd.exe -h
```

Figure 4.1: the code to call the pcl_mesh2pcd.exe

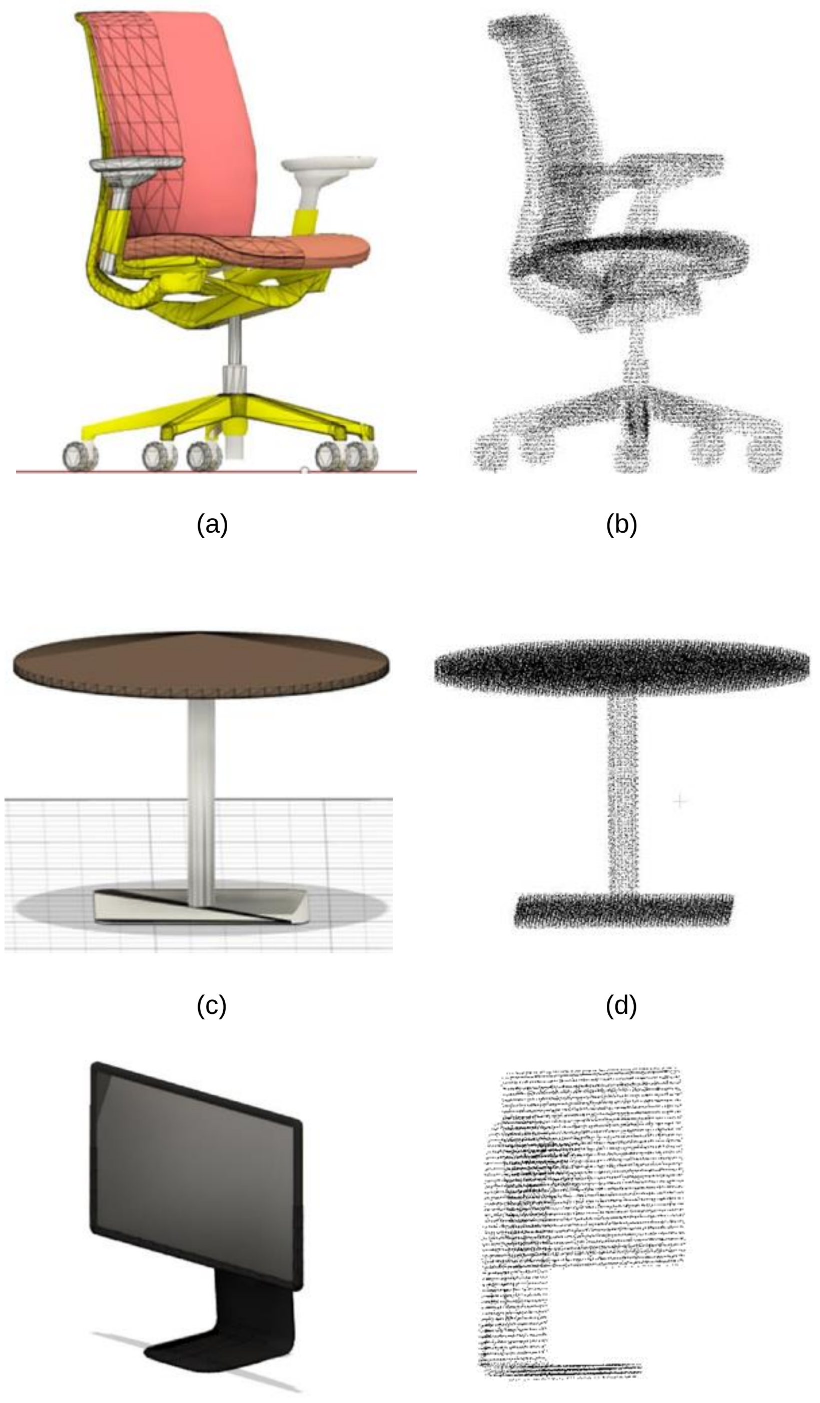


Figure 4.2: (a),(c),(e) are the CAD model being converted and (b),(d),(f) are the point cloud after conversion.

It should be noted that although the CAD model we are used is very similar to the physical object in real scene, but there is still a big gap with the model in the scene point cloud. This is because when the real scene is scanned, some parts are not collected in the point cloud data due to the problems of noise and occlusion. Therefore, in order to make the model point cloud and the scene point cloud as similar as possible to avoid negative impact on the registration, the model point cloud need to be trimmed details by using Cloudcompare.

4.2.2 Scene

These three scene point clouds are all intercepted from a large point cloud, which is acquired using a wearable mobile scanning equipment Navvis VLS. These three sets of scene point clouds all contain the target model we want to register such as the chair in scene.1, the table in scene.2, and the computer screen in scene.3. The visualization of these three sets of scene point cloud is shown in Figure 4.3. Table 4.1 presents the information of all point cloud, which includes the resolution, number of points, and box dimensions. Because all the point cloud data are collected by the same instrument, the resolution of the point cloud is the same. In the following experimental validation, we will change the resolution of the point cloud by downsampling and upsampling, in order for testing the performance of algorithm with different point cloud resolution.

Table 4.1 Point Cloud Datasets

Nr.	Nr. of Points	Box Dimensions[m]			Resolution
		x	y	z	
Model.1	7108	0.69192	0.66246	0.976044	0.01
Scene.1	177992	4.11433	2.3951	1.51406	0.01
Model.2	3831	0.729336	0.667839	0.742377	0.01
Scene.2	56128	2.1815	2.45214	0.861658	0.01
Model.3	2951	0.648599	0.239517	0.503199	0.01
Scene.3	16267	1.1283	1.66858	1.2341	0.01

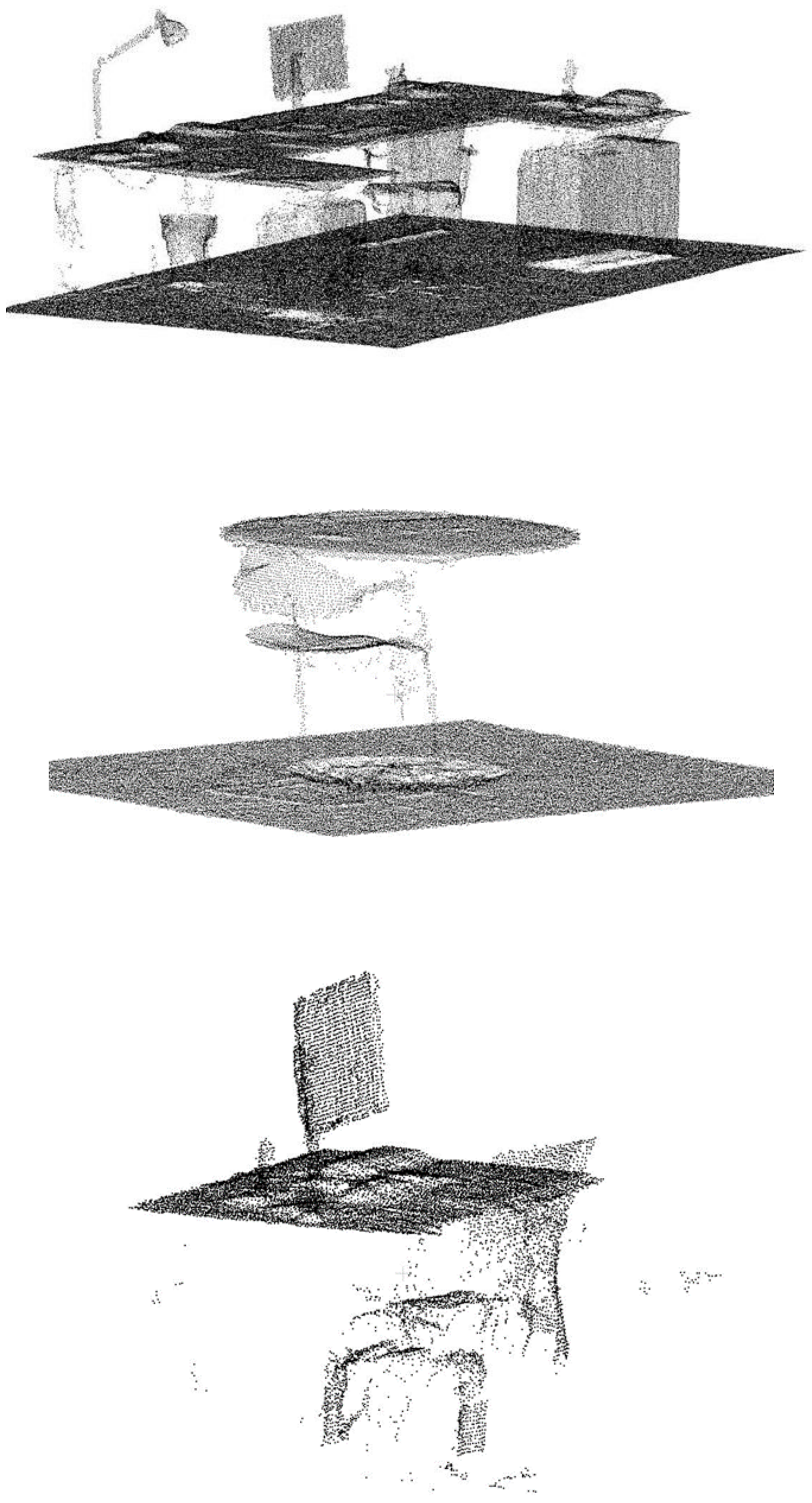
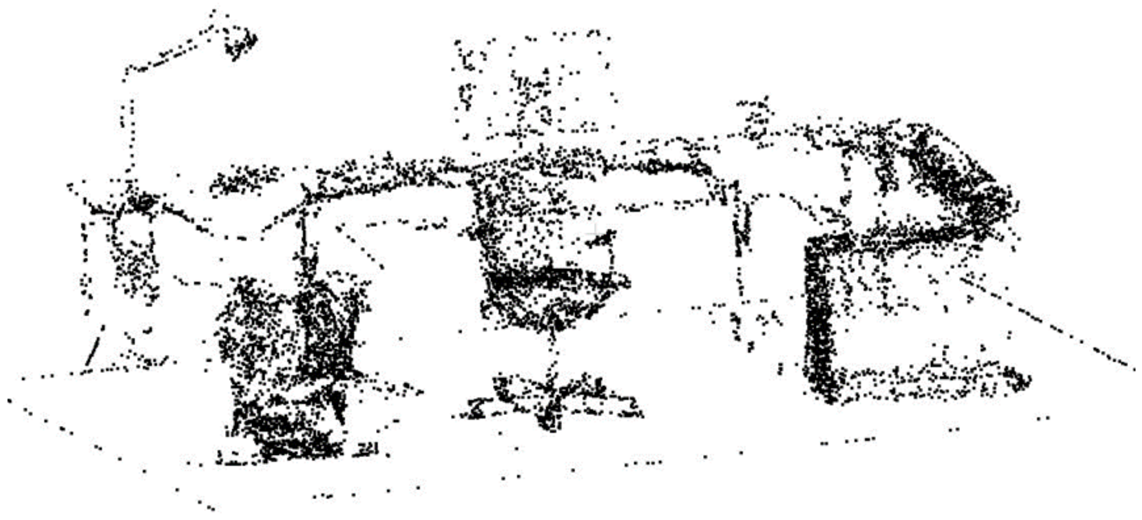


Figure 4.3: From top to the bottom: scene.1, scene.2 and scene.3 of point cloud

4.3 Result and Analysis

4.3.1 Result of Noise Filter

As we mentioned before, in order to reduce the result error in the registration process, we filter the original point cloud using *StatisticalOutlierRemoval* method providing by PCL. We filtered the point clouds of the three scenes point cloud separately. The number of filtered points is also listed in table 4.2. We also visualize the filtered points in Figure 4.4, which can intuitively show which points in the point cloud are filtered out.



(a)

(b)

(c)

Figure 4.4: The removed points during noise filter

Table 4.2: Noise filter information

	Scene.1	Scene.2	Scene.3
Points before filter	177992	56182	16267
Points after filter	169539	53910	15742
Removed points	8453	2272	525

4.3.2 Key Points and feature descriptor

First, in order to test the performance of two different key point extraction algorithms, 3D Voxelization and SIFT Keypoint, we input the Scene.1 from our point cloud dataset. Because Scene.1 contains a lot of details, for instance, the chair besides the table has a lot of curved surfaces, the table and monitor have a lot of flat surfaces. It can be measured that the two different algorithms deal with the curved surface and the flat surface respectively. In addition, there is a certain occlusion between the table and the chair, which can also test the effect of the algorithm on the occlusion situation.

According to the previous introduction, radius of grid should be set in advance before input point cloud data when we use 3D Voxelization. Because the density of point cloud is 0.008, so we choose 0.08 as the radius. Scene.1 has 177992 points totally, 3751 key points are extracted with 3D Voxelization, and 5885 key points are detected with SIFT keypoint. The extracted key points are shown in the Figure 4.5.

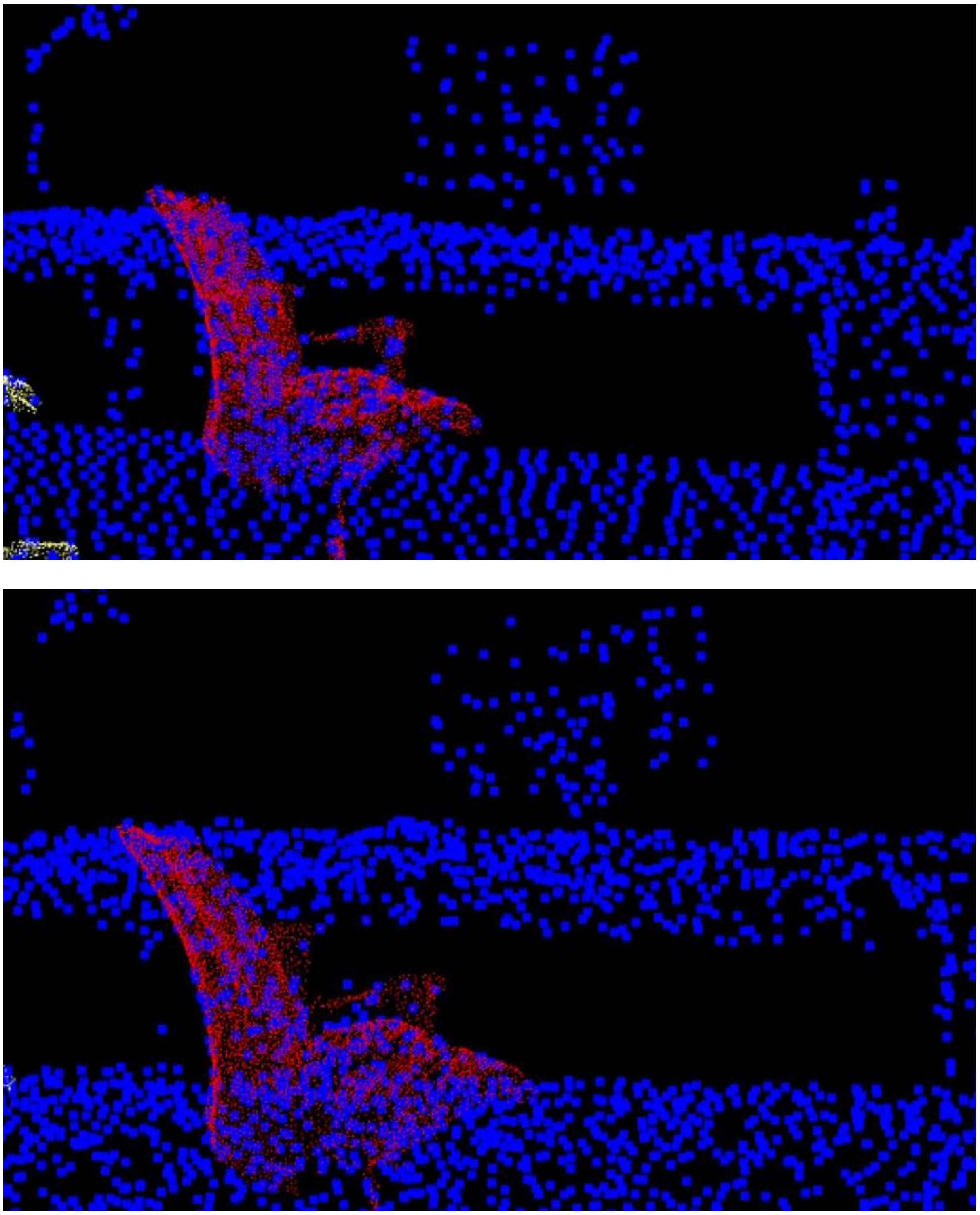


Figure 4.5: Key points (blue points) extracted from point cloud. Above figure is key points with 3D voxelization algorithm, bottom figure is key points with SIFT Keypoint algorithm.

From the figure, we can conclude that the key points extracted by the SIFT Keypoint algorithm did not maintain the shape of the original point cloud, while the key points extracted by the 3D Voxelization algorithm maintained the original shape of the point cloud.

The reason for this phenomenon is related to the principle of the algorithm, 3D Voxelization create voxel grid in point cloud with a certain radius which we have set in advance, and in each voxel the center of gravity of all points in the voxel is used to approximate the other voxels point. The key points generated from this can well maintain the shape characteristics of the original point cloud. However, SIFT Keypoint algorithm extracts key points by detecting extreme points in Gaussian space, and extreme points contain less point cloud shape information.

In addition, the computation time of the two algorithms is also different. In the key point extraction process of this point cloud, the computation time of the SIFT key point algorithm is twice that of the 3D Voxelization algorithm. The calculation time of the 3D Voxelization algorithm is relatively short, thanks to its simpler calculation principle, and the calculation process is relatively straightforward.

4.3.3 Registration Result

We input three sets of models to test whether it can successfully locates and match the corresponding point cloud model in the scene. But the first two sets of models have matching errors, mainly because the algorithm parameters are not set to the best values. Therefore, we have made many attempts to optimize the parameter settings. These parameters have a great relationship with the density of the point cloud.

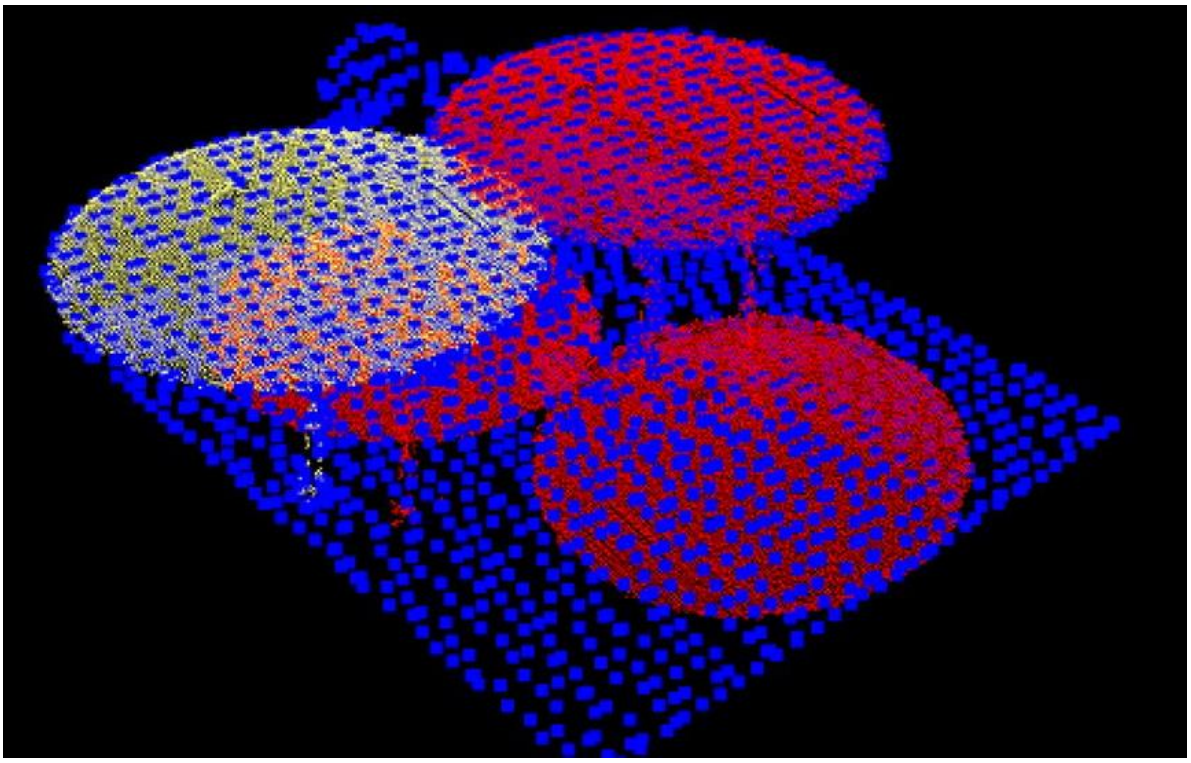
For instance, in the SHOT feature descriptor computation algorithm, we have to set the radius of local reference frame and the radius for descriptor computation. The size of the radius depends on the density of the point cloud. If the density of the point cloud is larger, then within a certain radius, the number of points involved in the calculation will be more. On the contrary, if the point cloud is relatively sparse, then within a certain radius, the number of points involved in the calculation is small.

The algorithm requires us to take a suitable radius to allow a suitable number of point clouds to participate in the calculation. If the radius is larger, the calculation time will be prolonged and the calculation efficiency will be reduced, but the feature points can better express the local features. If the radius is reduced, the calculation time is shortened and the calculation efficiency is improved, but the calculated feature points can not well represent the local features. After constant attempts, we found that when the radius is 10 to 20 times the resolution of the point cloud, the calculation effect of the algorithm is ideal.

In addition, in the correspondence grouping algorithm, we use the Hough voting method to group the found correspondence. According to the definition of the Hough voting, we need to preset the length of each Hough bin, which it can also be considered as the size of cluster. It should be enough large to encompass oscillations due to noise of the Hough votes but not too large to create spurious peaks in the Hough space. Then we also need to set the threshold of Hough voting to stipulate the minimum number of votes. It should be noted that these two parameters have a great influence on the results of the registration. If the parameter size is set incorrectly, a matching error will occur.

For example, we set the radius for key points extraction is 0.06m, the radius of feature descriptor is 0.12m, the size of cluster is 0.5m and the threshold of Hough voting is 3. This is obviously a very unreasonable combination of parameters. The size of the cluster group is too large, or the threshold of votes is set too small, then too many meaningless feature points will be included. This will result in multiple model instances in the scene as shown in Figure 4.6.

It can be seen from the Figure 4.6 that due to the inappropriate cluster size, only part of the feature points in the model are matched, that is, the feature points on the seat plate of the chair are matched with the corresponding feature points in the scene. Part of the feature points involved in the matching also caused the model's pose estimation error in the scene.



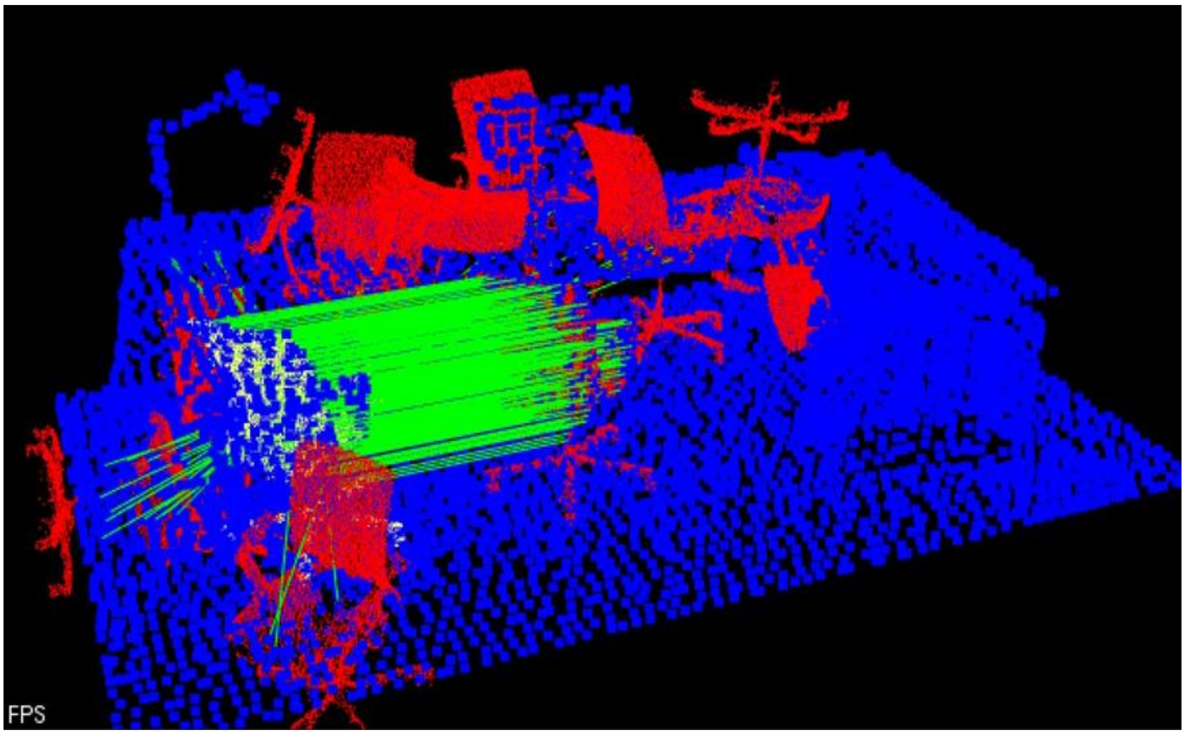


Figure 4.6: Not ideal matching result with multiple instance in scene.

After many experiments, the appropriate algorithm parameters were found, and all three sets of point cloud models can be registered in the scene point cloud through this algorithm. The registration result is shown in the Figure 4.7.

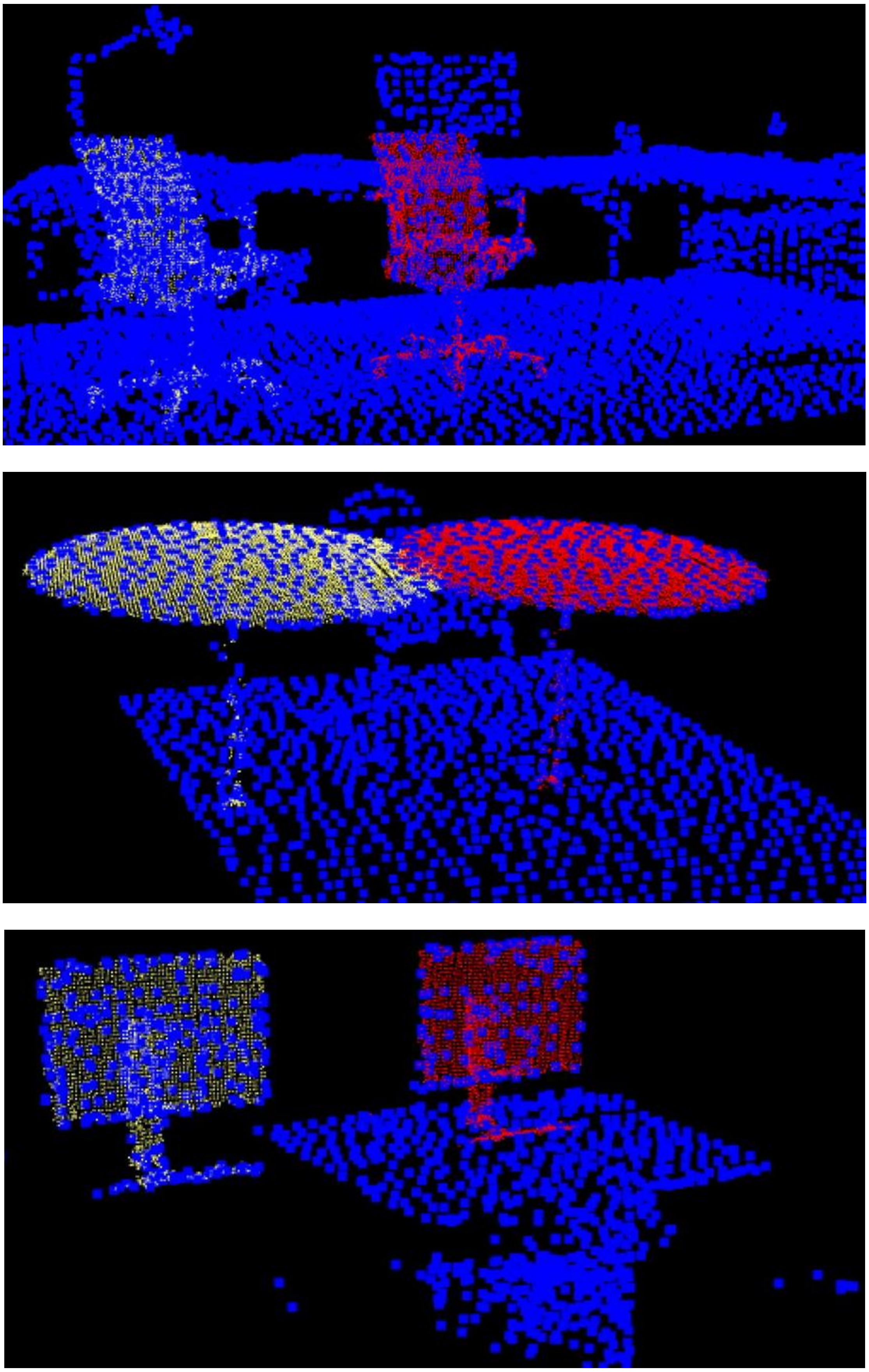


Figure 4.7: Ideal matching result, red part is instance model located from scene.

From the matching results, we can see that all three sets of model point clouds can be correctly localized and matched in the scene. It should be noted that before we mentioned two feature point computation algorithms, SHOT descriptor and FPFH descriptor. In the experiment, the two algorithms were tested separately, and it was found that after setting the algorithm parameters, both algorithms can complete the task, but the calculation time of the SHOT descriptor is faster than the time of the FPFH descriptor. The specific computation time will be detailed in the following algorithm computation time table. It can be concluded from the above experimental results that the algorithm has good accuracy and robustness in simple indoor scenes.

Table 4.3: information in registration, amount and consumed time(s).

	Model.1	Scene.1	Model.2	Scene.2	Model.3	Scene.3
Key points	430	6771	227	1731	139	699
Key points extraction time	25.6		8.4		3.2	
SHOT descriptor	56.2		14.3		5.6	
FPFH descriptor	62.1		19.1		9.4	
Correspondence	2288		1315		105	
Correspondence estimation time	3.4		0.6		0.2	
Correspondence grouping time	55.7		14.1		5.2	
ICP time	2.1		0.8		0.3	

4.4 Experiment in complex indoor scene

But how does the algorithm perform in complex scenes, such as indoor scenes with a large range, or there is more interference in the scene. In consideration of the robustness of the test algorithm, we selected a more complex indoor scene, as shown in Figure 4.8.

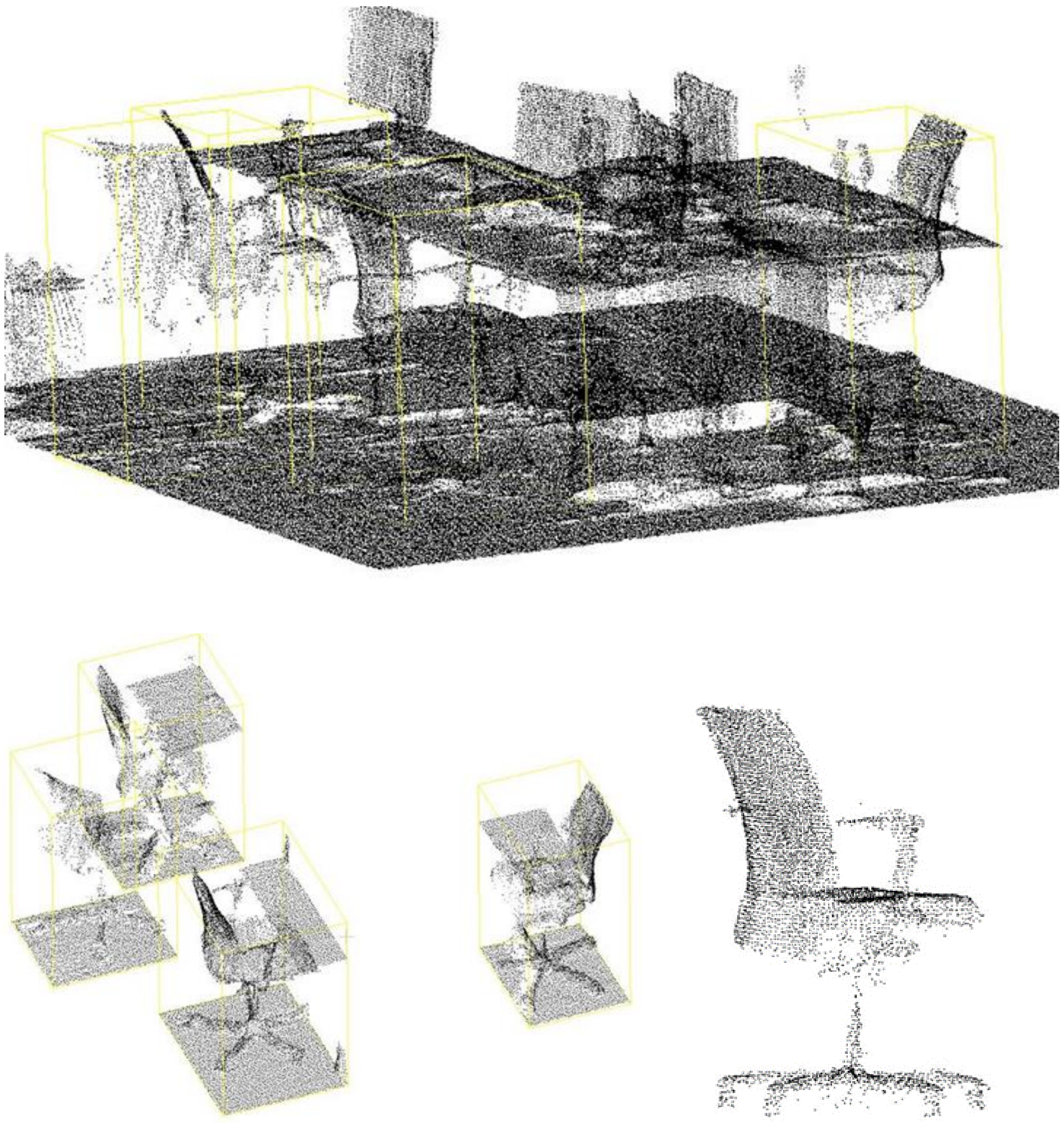
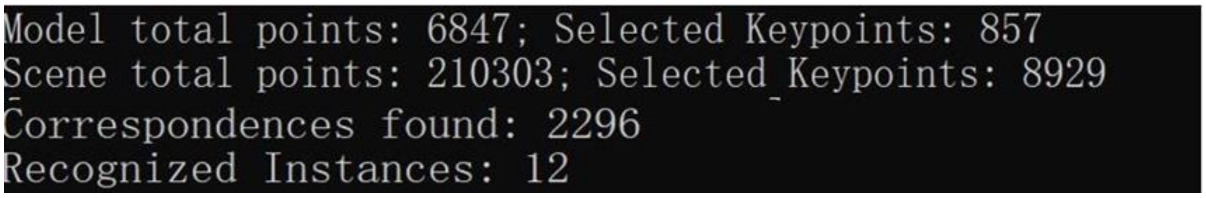


Figure 4.8: point cloud dataset with complex indoor scene. (a) is the scene, (b) is the target model in scene and (c) is the model.

Because scene(b) is too messy, it is difficult to see the target model from the image. For the convenience of the reader, we cut out the target model from the scene. The four chairs in (c) are the target models we want to match. It can be seen that the target model is very difficult to identify in the scene, such as the occlusion between the chair and the table, or the lack of some points of the chair point cloud in scene. So this is a big challenge for our algorithm. Unfortunately, the algorithm cannot handle the point cloud matching problem in complex scenes. Because there are too many interference items in the scene point cloud, the algorithm is greatly disturbed during feature descriptor matching, resulting in a lot of false matches. The number of key points and correspondence found by the algorithm are shown in the Figure 4.9. The visualization of the matching results is shown in Figure 4.10.



```
Model total points: 6847; Selected Keypoints: 857
Scene total points: 210303; Selected_Keypoints: 8929
Correspondences found: 2296
Recognized Instances: 12
```

Figure 4.9: computation result of key point extraction, correspondence and grouping

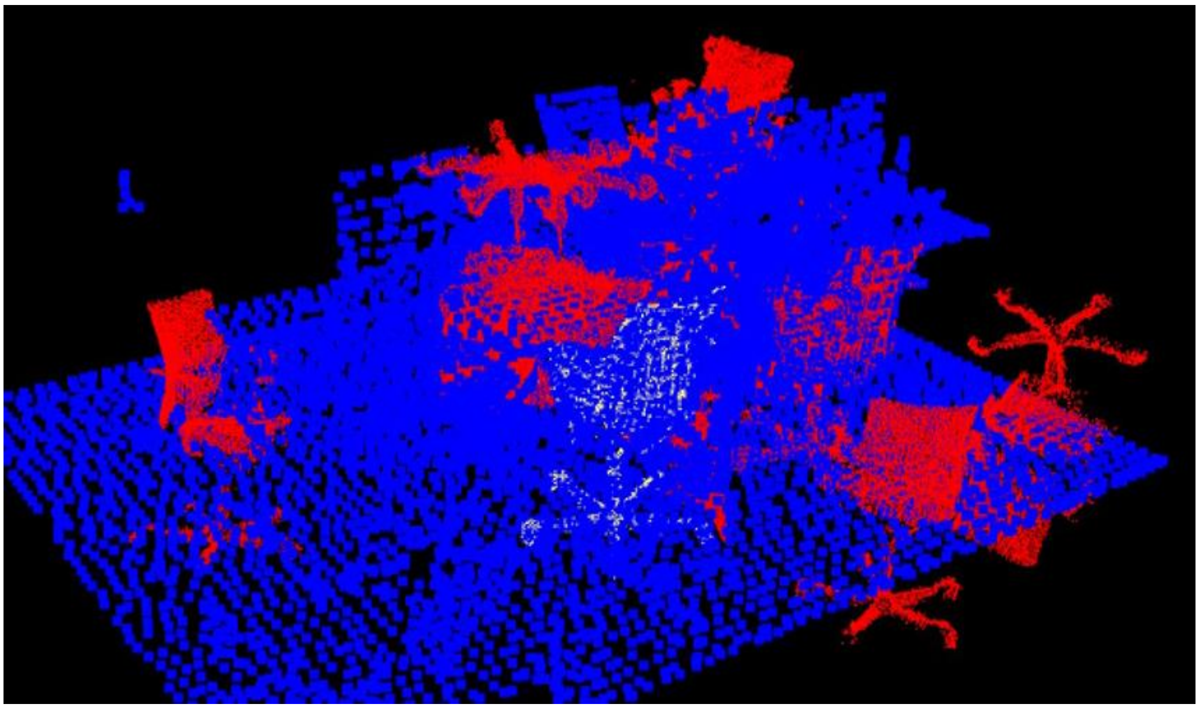


Figure 4.10: visualization of matching result with errors.

A total of 12 instance models were found in the scene point cloud, which is obviously unrealistic, indicating that the algorithm will produce large matching errors when dealing with complex scenes.

It can also be seen from the figure that a lot of key points are incorrectly matched. This is because there are variants similar parts and too many occlusions between objects in the scene, resulting in wrong feature descriptors of key points. And because of occlusion, it is also difficult to assign appropriate sizes for Hough voting during correspondence grouping, resulting in many false voting peaks in the Hough space. This also reflects the limitations of the algorithm, unable to handle more complex scenes.

4.5 Influence of point cloud density

According to the previous description, the three sets of point clouds used in the experiment have the same resolution, so in order to validate the performance of algorithm in different resolution of point cloud, we use downsampling to artificially reduce the density of first set point clouds. We take Model.1 and Scene.1 as examples, and use downsampling to reduce the point cloud density. Table 4.4 introduce the density information.

Table 4.4: Point cloud datasets after downsampling

		Downsampling								
		0	1	2	3	4	5	6	7	8
Model	Points	7208	4319	2946	1983	1508	1143	936	717	610
	Resolution	0.01	0.015	0.02	0.025	0.03	0.035	0.04	0.045	0.05
Scene	Points	177992	91470	55917	34371	28310	21322	16148	12820	9130
	Resolution	0.01	0.015	0.02	0.025	0.03	0.035	0.04	0.045	0.05

It can be seen from the above table that the density of the point cloud of the model and the scene is reduced in the same ratio while downsampling. In other word, it is necessary to ensure that the density of the model is the same with the scene, in order to avoid the distribution caused by different densities. In order to intuitively reflect the effect of downsampling, we intercepted a group of point clouds after downsampling, which is shown in Figure 4.11.

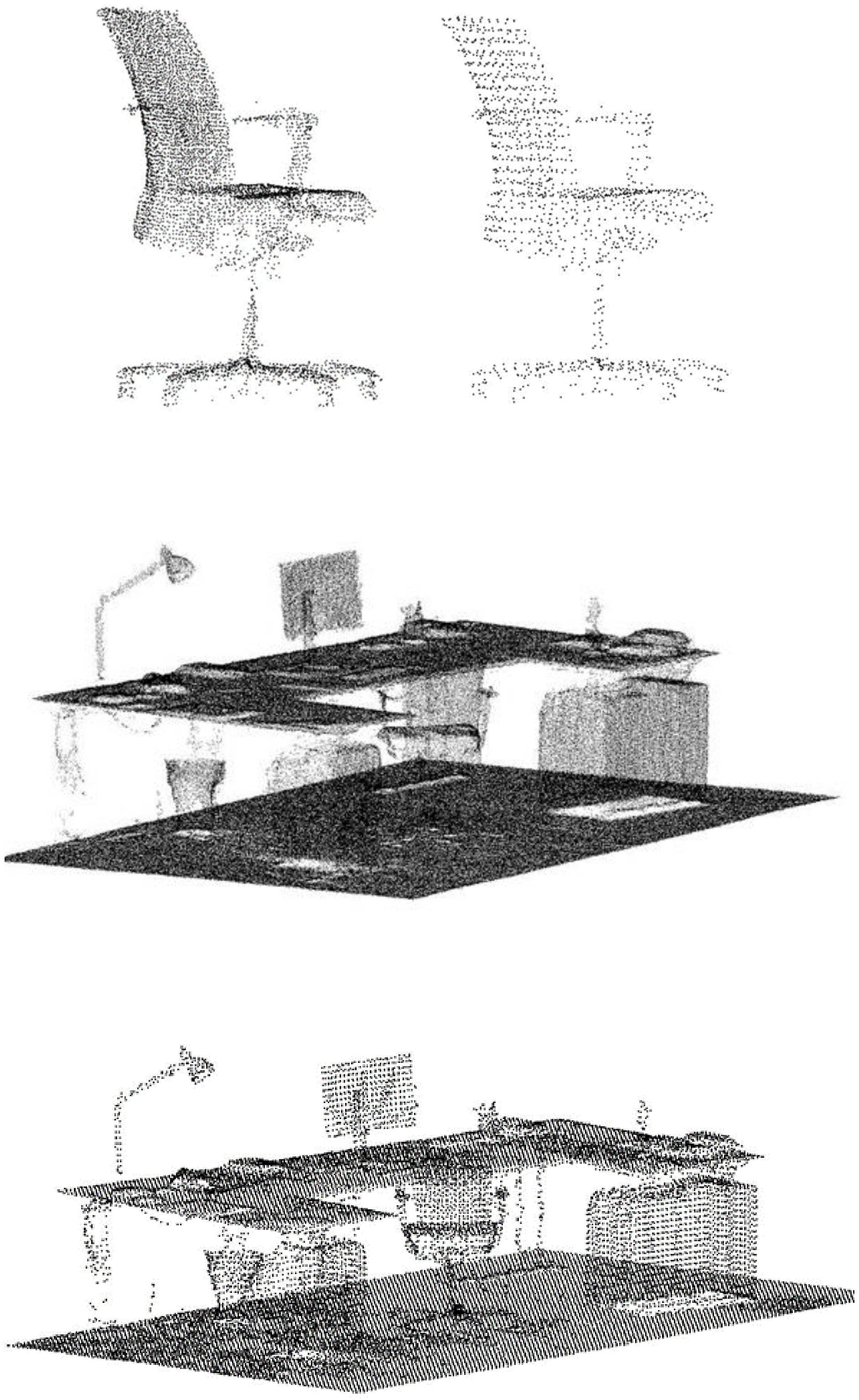


Figure 4.11: point cloud visualization after downsampling, point cloud from above are the comparison between original chair and dowmsampling chair, point cloud from bottom are the original scene and downsampling scene.

From the figure, we can intuitively see that the density of the point cloud is lower than before. And through our test, after the density is reduced, the density of point cloud does not have a significant impact on the point cloud registration result. But it should be noted that we need to re-find the appropriate algorithm parameters according to the density of the new point cloud. It is not feasible to directly use the previous algorithm parameters. But since we only use the indoor environment scale, the point cloud density at this scale will roughly change within the range we set. If the scale is expanded to a building, the point cloud density at this scale can continue to decrease. Hence, do not rule out the possibility that too sparse point cloud will affect the computation results of the algorithm. The registration result of point cloud after downsampling is shown in Figure 4.12 and Figure 4.13. We selected one of results from all result for display. It is obvious from the figure that the key points are relatively sparse. And the data of time consumed is shown in Table 4.5.

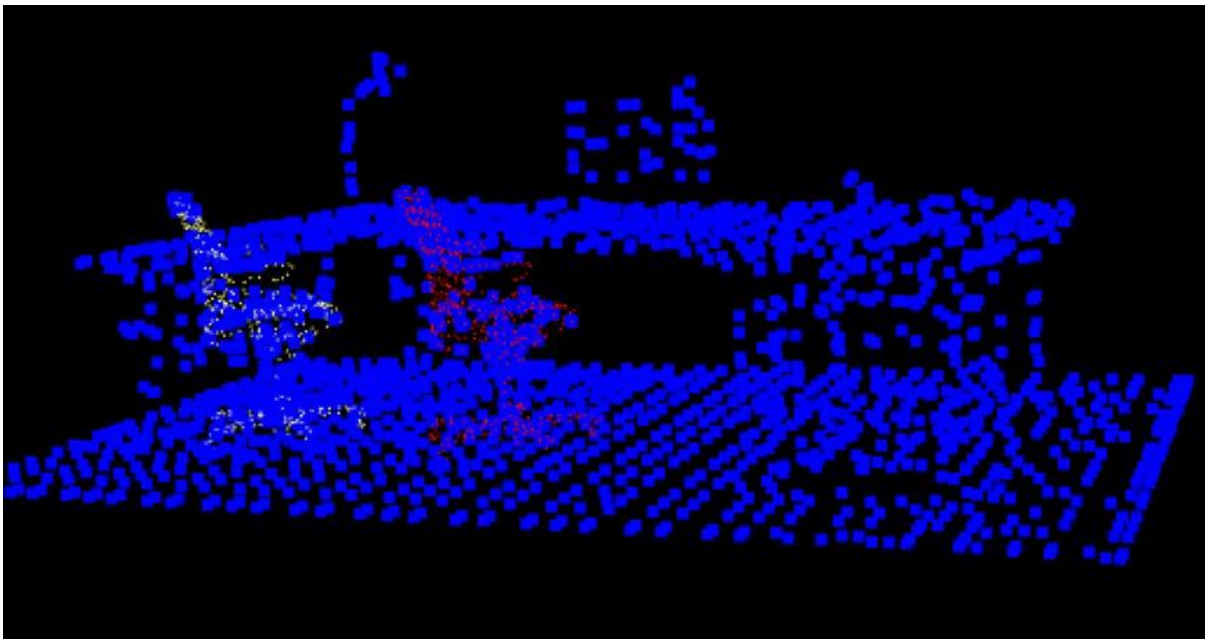
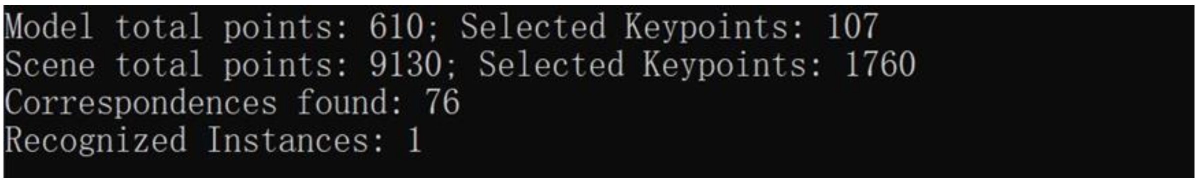


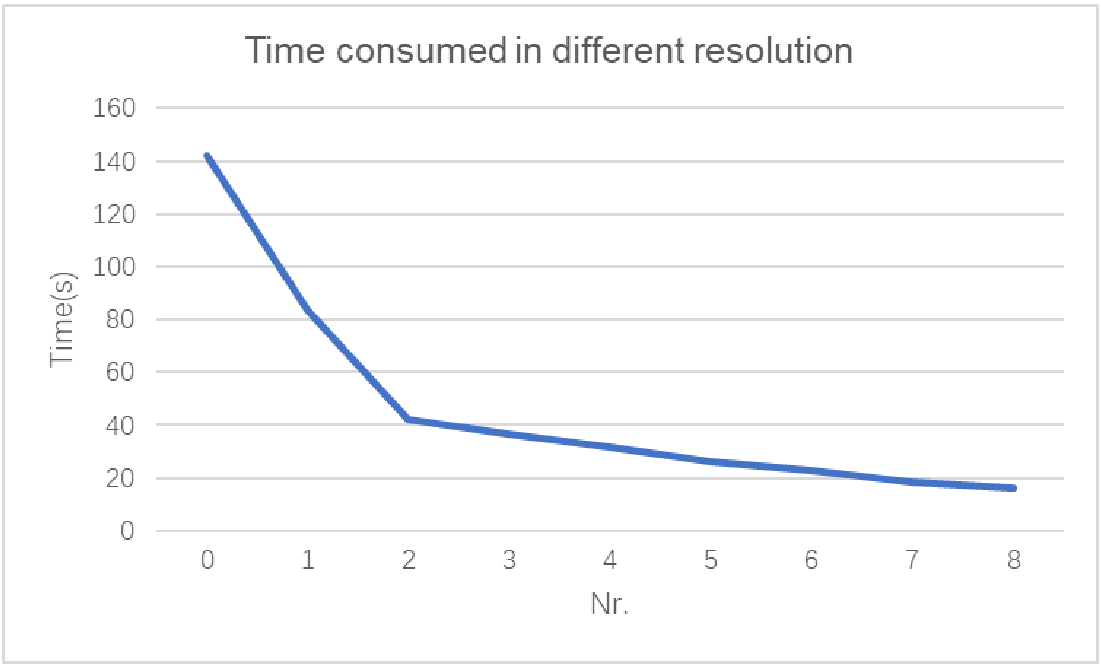
Figure 4.12: registration result of point cloud with sparse density.



```
Model total points: 610; Selected Keypoints: 107
Scene total points: 9130; Selected Keypoints: 1760
Correspondences found: 76
Recognized Instances: 1
```

Figure 4.13: result about key points extraction, correspondence estimation and grouping

Table 4.5: Statistics about consumed time



5 Conclusion and Outlook

5.1 Conclusion

This thesis aimed to complete a workflow of point cloud registration with Point Cloud Library. A variety of algorithms were used in the registration process, and the performance of some algorithms was tested. The models used are all point clouds of indoor environments.

First, we preprocessed the point cloud, such as point cloud conversion, filtering and downsampling. Then we used SIFT Keypoint and 3D Voxelization algorithm to extract the key points from point cloud and compare the key points extracted by the two algorithms. It is found that the key points extracted by the 3D Voxelization algorithm are more complete, the original shape of the point cloud is better preserved and less computation time needed. In the test, we tried to achieve variable scale through the sift algorithm, that is, the scale of the model point cloud and the target model in the scene can be changed. However, this function has not been implemented. We still need to adjust the ratio of both model and target model to ensure the success of subsequent point cloud registration when processing the point cloud in the early stage. Then, we use the SHOT descriptor and FPFH descriptor to describe the feature information of the key points, and compare the two algorithms. Both algorithms are based on feature distribution histograms, so they all have very good robustness, but in terms of computational efficiency, the SHOT descriptor is better, and the SHOT descriptor has faster calculation speed and fewer computation resource occupation. Then we used the Kd-tree data structure to take a rapid match the calculated feature descriptors between the model and the scene. Compared with the traditional brute force search, the Kd-tree data structure perform much more efficient. After getting the matching point pairs, we use the Hough voting algorithm to group these matching point pairs accordingly, so that the target model can be selected out of the scene and get the location information of the target model. However, sometimes because of the mutual occlusion and overlap between the models in the scene point cloud, and the presence of noise in the point cloud, sometimes the position information obtained is not accurate, so we next use the ICP algorithm. The position information obtained from Hough voting is used as the initial pose of ICP, the source model and target model can be accurately registered. Through this workflow, semantic registration between point clouds in simple indoor scene can be achieved.

According to Huang’s summary [27]. The registration of point cloud faces many challenges, these challenges can be mainly divided into four fields:

(1). Noise and outliers. Because the acquisition environment, sensor noise are different at different acquisition time, the captured point clouds will contain noise and outliers around the same 3D object.

(2). Partial overlap. Because of different viewpoint and acquisition time, the captured point cloud is only partial overlapped.

(3). Density difference. Due to different imaging mechanisms and different resolutions, the captured point clouds usually contain different density.

(4). Scale variation. Since different imaging mechanisms may have different physical metrics, the captured point clouds may contain different scale.

In order to test the performance of the algorithm facing these challenges, we selected point clouds in simple scenes, point clouds in complex scenes, and point clouds with different densities for testing. The accuracy and robustness of the algorithm in simple indoor scene are verified with different sets of point cloud, but many shortcomings of the algorithm are also found.

5.2 Limitations

When testing the registration algorithm, we found that the algorithm is very sensitive to parameters, and appropriate parameters need to be set for each experimental scenario. Therefore, a lot of attempts are required for each parameter before the registration. For instance, the computation radius in the SHOT descriptor algorithm determines how many adjacent points around each key point will be considered in the feature calculation. It should be enough large to include at least a few tens of points, but not too large not to include clutter in the description of key points close to the object border. And due to the influence of the boundary effect, the calculation of the feature points at the boundary will lose accuracy. In addition, the cluster size and threshold in Hough voting also have an great impact on the result of registration. It Should be enough large to encompass oscillations due to noise of the 3D Hough votes but not too large to create spurious peaks in the Hough space.

In addition, this algorithm is poor in processing complex indoor scenes. If the input scene has more point cloud objects, more noise, and more mutual occlusion between objects, the algorithm will produce larger errors. For example, in Chapter IV, we mentioned that in a very complex indoor point cloud environment, it is difficult to find suitable algorithm parameters to make the point cloud registration successful.

Finally, the algorithm cannot achieve scale variability. In the algorithm, we use the SIFT Keypoints algorithm, and strive to be able to register two point cloud models with different scale but not success. Scale Invariant Feature Transform(SIFT) is a scale-based Spatial image local feature description algorithm which is first introduced by Lowe[25], The algorithm was originally used to process images, for change of rotation, scale scaling, and brightness, it will maintain a certain degree of stability. PCL transplants it to process 3D point cloud data, and implements the SIFT key point extraction algorithm, but ignores the SIFT feature descriptor algorithm, which may cause the SIFT algorithm to be unable to achieve variable scale in the point cloud data.

5.3 Recommendation for Further Development

Based on the above experiments and analysis, as well as the current application scenarios for point clouds. The high accurate and robust registration should have the fast running speed with the guarantee of high accuracy. In this section, we suggest future research directions.

The point cloud is a record of the 3D environment. However, the real scene is much complicated because of the occlusion and overlap between objects, which makes noise and outliers variations. Firstly, the future direction could be robust to handle the challenging variations of noise and outliers in real-world point clouds. Our algorithm still requires a lot of pre-processing work before registering, to remove the noise and outliers manually, and does not realize the automation of the whole process. The point cloud noise removal algorithm in the existing PCL library still cannot effectively remove all point cloud noise, and the accuracy and speed are far behind the requirement of real applications.

Secondly, in the actual registration process, the scale of the two sets of point clouds is often inconsistent. It also takes time to adjust the scale of the two sets of point clouds. Although we have tried using the SIFT algorithm, because the SIFT algorithm is transplanted from 2D image processing, so further optimization is still needed. The use of more advanced OpenCV may solve this problem.

Thirdly, the algorithm can be further optimized to be able to handle point clouds of complex large scenes. Since the point cloud of a large scene contains more noise and outliers, it often causes the algorithm to fail to correctly register the target model. Combining with neural networks and deep learning may be a good research direction. Several pieces of literature have already started research in this area, such as registration using PointNet[38], self-supervised learning[39] and feature-metric registration[40]. They are trying to merge the conventional mathematical theories and deep neural network into the point cloud semantic registration workflow, in order to obtain both high accuracy and efficiency. This area is just the beginning, and there needs much research to develop fantastic registration algorithms.

Reference

[1] Thoma, M. (2016). A Survey of Semantic Segmentation. 1–16.

[2] Garcia-Garcia, A., Orts-Escolano, S., Oprea, S., Villena-Martinez, V., & Garcia-Rodriguez, J. (2017). A Review on Deep Learning Techniques Applied to Semantic Segmentation. 1–23.

[3] Babacan, K., Chen, L., & Sohn, G. (2017). SEMANTIC SEGMENTATION of INDOOR POINT CLOUDS USING CONVOLUTIONAL NEURAL NETWORK. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 4(4W4), 101–108.

[4] Xue, F., Lu, W., Chen, K., & Zetkulic, A. (2019). From Semantic Segmentation to Semantic Registration: Derivative-Free Optimization–Based Approach for Automatic Generation of Semantically Rich As-Built Building Information Models from 3D Point Clouds. Journal of Computing in Civil Engineering, 33(4), 04019024.

[5] Andreopoulos, A., & Tsotsos, J. K. (2013). 50 Years of object recognition: Directions forward. Computer Vision and Image Understanding, 117(8), 827–891.

[6] Holz, D., Ichim, A. E., Tombari, F., Rusu, R. B., & Behnke, S. (2015). Registration with the point cloud library: A modular framework for aligning in 3-D. IEEE Robotics and Automation Magazine, 22(4), 110–124.

[7] Bentley, J. L. (1975). Multidimensional Binary Search Trees Used for Associative Searching. Communications of the ACM, 18(9), 509–517.

[8] Cunningham, P., & Delany, S. J. (2020). k-Nearest Neighbour Classifiers: 2nd Edition (with Python examples). 1, 1–22.

[9] Elseberg, J., Magnenat, S., Siegwart, R., & Andreas, N. (2012). Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration. Journal of Software Engineering for Robotics (JOSER), 3(1), 2–12.

[10] M. Muja and D. G. Lowe, “Fast approximate nearest neighbors with automatic algorithm configuration,” in International Conference on Computer Vision Theory and Application (VISAPP), 2009, pp. 331–340.

[11] Engelmann, F., Kontogianni, T., Schult, J., & Leibe, B. (2019). Know what your neighbors do: 3D semantic segmentation of point clouds. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 11131 LNCS, 395–409.

[12] Barazzetti, L. (2016). Parametric as-built model generation of complex shapes from point clouds. Advanced Engineering Informatics, 30(3), 298–311.

[13] Xue, F., Chen, K., Liu, D., Niu, Y., & Lu, W. S. (2018). An optimization-based semantic building model generation method with a pilot case of a demolished construction. Proceedings of the 21st International Symposium on Advancement of Construction Management and Real Estate, 2016, 209889, 231–241.

[14] Wang, J., Wu, Q., Remil, O., Yi, C., Guo, Y., & Wei, M. (2018). Modeling indoor scenes with repetitions from 3D raw point data. CAD Computer Aided Design, 94, 1–15.

[15] Corsia, M., Chabardès, T., Bouchiba, H., & Serna, A. (2020). Large scale 3D point cloud modeling from CAD database in complex industrial environment. International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives, 43(B2), 391–398.

[16] Xu, K., & Sun, W. (2015). Autoscanning for Coupled Scene Reconstruction and Proactive Object Analysis Kai. ACM Transactions on Graphics, 34(6), 177.

[17] Nan, L., Xie, K., & Sharf, A. (2012). A search-classify approach for cluttered indoor scene understanding. ACM Transactions on Graphics, 31(6), 1–10.

[18] Kerber, J., Bokeloh, M., Wand, M., & Seidel, H. P. (2013). Scalable symmetry detection for urban scenes. Computer Graphics Forum, 32(1), 3–15.

[19] Pomerleau, F., Colas, F., & Siegwart, R. (2015). A Review of Point Cloud Registration Algorithms for Mobile Robotics. A Review of Point Cloud Registration Algorithms for Mobile Robotics.

[20] Tombari, F., Salti, S., & Stefano, L. Di. (2010). Unique Signatures of Histograms for Local Surface Description. CVLab - DEIS, University of Bologna, Viale Risorgimento, 2 - 40135 Bologna, Italy, 356–369.

[21] Tombari, F., Salti, S., & Di Stefano, L. (2011). A combined texture-shape descriptor for enhanced 3D feature matching. Proceedings - International Conference on Image Processing, ICIP, 809–812.

[22] Salti, S., Tombari, F., & Di Stefano, L. (2014). SHOT: Unique signatures of histograms for surface and texture description. Computer Vision and Image Understanding, 125, 251–264.

[23] Rusu, R. B., Blodow, N., & Beetz, M. (2009). Fast Point Feature Histograms (FPFH) for 3D registration. IEEE International Conference on Robotics and Automation, 3212–3217.

[24] Rusu, R. B., Blodow, N., Marton, Z. C., & Beetz, M. (2008). Aligning point cloud views using persistent feature histograms. 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 3384–3391.

[25] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, 60(2), 91–110.

[26] Lowe, D. G. (1999). Object Recognition from Local Scale-Invariant Features. International Conference on Computer Vision, Corfu, Greece, pp. 1150–1157.

[27] Huang, X., Mei, G., Zhang, J., & Abbas, R. (2021). A comprehensive survey on point cloud registration. 1–17.

[28] P. J. Besl and N. D. McKay, “A method for registration of 3-D shapes,” IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), vol. 14, no. 2, pp. 239–256, 1992.

[29] Y. Chen, G. Medioni. Object Modeling by Registration of Multiple Range Images[J]. Image and Vision Computing, 1992. 10: 145-155.

[30] R. Bergevin, M Soucy, H. Gagnon, et al. Towards a general multi-view registration technique[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1996. 18(5): 540-547.

[31] S. Rusinkiewicz and M. Levoy, “Efficient variants of the ICP algorithm,” in International Conference on 3-D Digital Imaging and Modeling (3DIM), 2001.

[32] Andrew Edie Johnson, Sing Bing Kang. Registration and Integration of Textured 3D Data[J]. Image and Vision Computing, 1999. 17: 135-147.

[33] Natasha Gelfand, Leslie Ikemoto, Szymon Rusinkiewicz, et al. Geometrically Stable Sampling for the ICP Algorithm[A]//. Proceedings of Fourth International Conference on 3-D Digital Imaging and Modeling[C]. Stanford University, CA, USA, 2003: 260-267.

[34] Tombari, F., & Di Stefano, L. (2010). Object recognition in 3D scenes with occlusions and clutter by Hough voting. Proceedings - 4th Pacific-Rim Symposium on Image and Video Technology, PSIVT 2010, 349–355.

[35] Qi, C. R., Litany, O., He, K., & Guibas, L. (2019). Deep hough voting for 3D object detection in point clouds. Proceedings of the IEEE International Conference on Computer Vision, 2019-October(Iccv), 9276–9285.

[36] Yang, J., Chen, J., Huang, Z., Quan, S., Zhang, Y., & Cao, Z. (2020). 3D Correspondence Grouping with Compatibility Features. d.

[37] Hui Chen and Bir Bhanu. 3d free-form object recognition in range images using local surface patches. Pattern Recognition Letters, 28(10):1252–1262, 2007.

[38] Yasuhiro Aoki, Hunter Goforth, Rangaprasad Arun Srivatsan, and Simon Lucey. Pointnetlk: Robust & efficient point cloud registration using pointnet. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 7163–7172, 2019.

[39] Yue Wang and Justin M Solomon. Prnet: Self-supervised learning for partial-to-partial registration. In Advances in Neural Information Processing Systems, pages 8814–8826, 2019.

[40] Xiaoshui Huang, Guofeng Mei, and Jian Zhang. Feature-metric registration: A fast semi-supervised approach for robust point cloud registration without correspondences. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 11366–11374, 2020

[41] K.-L. Low, “Linear least-squares optimization for point-to-plane ICP surface registration,” in Technical Report TR04-004, Department of Computer Science, University of North Carolina at Chapel Hill, 2004.

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

[43] Jacopo Serafin and Giorgio Grisetti. “NICP: Dense Normal Based Point Cloud Registration,” International Conference on Intelligent Robots and Systems (IROS), 2015, pages:742-749

Declaration of Originality

With this statement I declare, that I have independently completed this Master Thesis. The thoughts taken directly or indirectly from external sources are properly marked as such. This thesis was not previously submitted to another academic institution and has also not yet been published.

München, 30. Juli 2021

Firstname Lastname

Name

Address

Email