Towards 3D Matching of Point Clouds Derived from Oblique and Nadir Airborne Imagery

by

Ming Zhang

B.S., Tianjin University, 2007

M.S., Beijing University of Post & Telecommunication, 2010

A thesis submitted in partial fulfillment of the requirements

for the degree of Master of Science

in the Chester F. Carlson Center for Imaging Science,

College of Science,

Rochester Institute of Technology

May, 1st, 2014

Signature of the Author____

Accepted by____

Coordinator, M.S. Degree Program Date

CHESTER F.CARLSON CENTER FOR IMAGING SCIENCE
COLLEGE OF SCIENCE
ROCHESTER INSITITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK, UNITED STATES OF AMERICA

CERTIFICATE OF APPROVAL

M.S DEGREE THESIS

The M.S. Degree Thesis of Ming Zhang
has been examined and approved by the
thesis committee as satisfactory for the
thesis requirement for the
M.S. Degree in Imaging Science

Dr. John Kerekes, Thesis Advisor

Dr. Carl Salvaggio, Committee Member

Dr. David Messinger, Committee Member

Date

Towards 3D Matching of Point Clouds Derived from Oblique and Nadir Airborne Imagery

by

Ming Zhang

Submitted to the Chester F. Carlson Center for Imaging Science
in partial fulfillment of the requirements
for the Master of Science Degree
at the Rochester Institute of Technology

Abstract

Because of the low-expense high-efficient image collection process and the rich 3D and texture information presented in the images, a combined use of 2D airborne nadir and oblique images to reconstruct 3D geometric scene has a promising market for future commercial usage like urban planning or first responders. The methodology introduced in this thesis provides a feasible way towards fully automated 3D city modeling from oblique and nadir airborne imagery.

In this thesis, the difficulty of matching 2D images with large disparity is avoided by grouping the images first and applying the 3D registration afterward. The procedure starts with the extraction of point clouds using a modified version of the RIT 3D Extraction Workflow. Then the point clouds are refined by noise removal and surface smoothing processes. Since the point clouds extracted from different image groups use independent coordinate systems, there are translation, rotation and scale differences existing. To figure out these differences, 3D keypoints and their features are extracted. For each pair of point clouds, an initial alignment and a more accurate registration are applied in succession. The final transform matrix presents the parameters describing the translation, rotation and scale requirements.

The methodology presented in the thesis has been shown to behave well for test data. The robustness of this method is discussed by adding artificial noise to the test data. For Pictometry oblique aerial imagery, the initial alignment provides a rough alignment result, which contains a larger offset compared to that of test data because of the low quality of the point clouds themselves, but it can be further refined through the final optimization. The accuracy of the final registration result is evaluated by comparing it to the result obtained from manual selection of matched points.

Using the method introduced, point clouds extracted from different image groups could be combined with each other to build a more complete point cloud, or be used as a complement to existing point clouds extracted from other sources. This research will both improve the state of the art of 3D city modeling and inspire new ideas in related fields.

Acknowledgement

Foremost, I would like to express my sincere gratitude to my advisor Prof. John Kerekes for his continuous support of my MS study and research, his patience, motivation, enthusiasm, and immense knowledge. He helped me in all the time of research and writing of this thesis. Without his persistent guidance this dissertation would not have been possible.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Carl Salvaggio and Prof. David Messinger for their insightful comments, and hard questions, which encouraged me to make continuous progress throughout the research.

My sincere thanks also go to Nina Raqueno and Michael Richardson. Nina helped me to access more pre-processed Pictometry imagery as well as the meta data. Michael offered me the chance to better understand the research using simulated building models and muti-view images.

I also thank Stephen Schultz and Yandong Wang from Pictometry. They not only provide me with a chance to work on such an exciting project, but also enlighten me with ideas from an industry point of view.

I thank Prof. Harvey Rhody, Erin Ontiveros, David Nilosek and Kate Salvaggio. They help me with the usage and modifying of the RIT 3D Extraction Workflow and the understanding of epipolar geometry and photogrammetry.

Also I thank the other two students Jie Zhang and Ming Li, who also work on Pictometry data. The cooperation with them gave me a better understanding of the other parts involved in the entire 3D modeling. We discussed possible methods and solved the problems we met together all the time.

Last but not the least, I would like to thank Shaohui Sun and Lei Hu, who helped me out with many difficulties I met in coding.

Contents

List of Figures

Chapter 1

Introduction

3D city modeling has become a hot research topic in the last decade. This technology has been widely used in various areas like visualization, urban planning, first responders, visual military, and even insurance (Notargiacomo, 2012). The generation of highly accurate large scale 3D scenes is, however, a time-consuming process which usually depends or partially depends on manual work. Fully automatic modeling is still a developing field.

1.1 Motivation

A popular computer vision technique to build a 3D model is extracting "structure from motion" of a calibrated camera with respect to a target. Combining this computer vision technique with photogrammetry and applying them to the geographic scenes makes it feasible to build a 3D city model from 2D imagery automatically. The earliest images are obtained from ground-based photography. These images could provide every detail of the buildings, but the collection process takes a long time and the contents of the images are largely limited to façade information. Nowadays, most conventional images are taken from much further distances, using a spaceborne or airborne platform in a quasi-vertical perspective. With this remote sensing viewpoint, images could be acquired in a relatively short time period, while covering a much larger area. There are also limitations, that although the nature of nadir imagery makes the registration process easier, it trades off limited viewing of structures under roofs, especially when occlusion happens. In this case, a new data type is needed to keep the remote sensing advantages and to alleviate its limitations, and airborne oblique imagery might be the one to solve the problem.

However, due to the nature of oblique imagery, it is difficult to match images taken from different viewing directions or integrate them with other information like traditional nadir images. An alternative method is to build a 3D point cloud for each group of images taken from the same viewing direction. Then these point clouds could be combined with each other to build a more complete point cloud using a 3D registration method.

The methodology introduced in this thesis will improve the state of the art of 3D city modeling and inspire new ideas in related fields. Using 2D airborne oblique images to reconstruct 3D scenes has a promising market for future commercial usage in various areas. The usage of oblique images can overcome the limitation of traditional vertical images. More detailed structures of the side facets can be better represented even when occlusion happens. Besides providing more complete information, it makes faster response possible compared to the ground-based method and costs much less expense than using images taken by satellite or data generated by LiDAR. In addition, the oblique images captured from different directions are ideal for generating building texture, based on further research in surface identification. The 3D registration method proposed avoids the difficulty of 2D registration originating from projective distortion. It provides an automatic way to integrate point clouds extracted from different image collections. It can also be used to improve the existing models by adding additional parts extracted from other sources.

1.2 Objectives

The research in this thesis has two main aspects: (1) Extract dense point clouds from oblique and nadir airborne images and refine them. (2) Extract 3D features and implement automatic 3D registration for different point clouds with partial overlap.

The research begins with dividing multi-view Pictometry imagery (http://pictometry.com/) into different groups to reduce the disparity between images because of the projective distortion. For each group, the geometry extraction process is based on the modification of the RIT 3D Extraction Workflow (http://dirsapps.cis.rit.edu/3d-workflow/index.html), which obtains better 2D keypoint extraction and matching especially for oblique images.

The original point clouds reconstructed by the workflow are quite noisy. Some noise points are sparse points spreading all over the space, while some are floating miscalculated clusters. They need to be eliminated using different methods respectively. So far, since these point clouds are extracted from different image groups, they are represented in independent coordinate systems. 3D matching is needed before integrating them.

The 3D registration procedure is based on a robust 3D feature extraction. We need to find a method to simplify the calculation by extracting 3D keypoints from the huge point clouds. We also need to find a suitable way to describe the features of the keypoints efficiently. The 3D registration can be achieved by two alignment steps. The initial alignment should obtain a rough matching of the point clouds by an approximate estimation of the translation, rotation and scale difference. After that, a more accurate registration process is used to optimize the result. The output transformation matrix consists of the parameters for scale, rotation and translation changes. Since the ground truth of the primary data set studied is unknown., another test data set will be used instead to estimate the robustness of our method under noisy condition. The accuracy of the final registration result can be evaluated by comparing it to the compound point cloud obtained from manual selection of corresponding points.

1.3 Layout of the Thesis

The second section of the thesis provides a brief instruction to 3D city modeling. It starts with the concept of 3D modeling, includes the characteristics of 3D building models, and ends with a literature review of current research in this field.

The third section introduces the data used in this project and the general methodology. A detailed statement of related algorithms is present in this chapter, including the original point cloud extraction, the point cloud refinement, and the 3D registration.

The fourth section presents results the author has obtained for both the test data and Pictometry data step by step. The robustness and accuracy of the methodology is also discussed in this section.

The last section of the thesis summarizes the work of the research. The limitations of the work are listed. The plans for the future research are suggested in the end.

Chapter 2

Background

2.1 3D Modeling

3D modeling is the process of developing a mathematical representation of the three-dimensional characteristic of a certain object. The product of the 3D modeling process is called a 3D model. It simulates the original object using a collection of points, or points connected by various geometric entities such as lines, triangles, or curved surfaces in 3D space. A 3D model can be created manually, algorithmically, or by scanning. To display a 3D model, we can use two-dimensional image sequences through a 3D rendering process in computer, or physically represent it using 3D printing devices.

2.1.1 Categories

Generally, most 3D models fall into two categories, solid models and boundary models. Solid models are defined by the volume of the object. These models are more closely related to realistic objects, but difficult to build. These models are mostly used for non-visual simulations for some specialized applications, such as medical investigation, ray tracing and engineering simulation. On the contrary, boundary models are defined only by the surface of the object. These models are more popular for commercial usage and much easier to work with. Nowadays, boundary models have become an important part of video game and film designs to build virtual scenes. In this thesis, we aim to build boundary models for buildings using an image-based 3D exaction method.

2.1.2 Structure from Motion

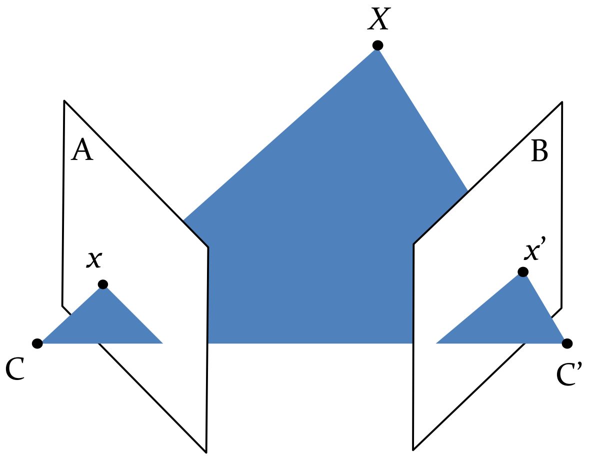


Fig. 2-1 Geometry of two views

A popular technology to generate a 3D model for a real object is to extract "Structure from Motion". It refers to the process of finding the 3D structure of an object by analyzing the local motion signals over time. In Fig. 2-1, two photos A and B are taken for the interested point X from different directions respectively. Draw a line passing through the camera center C, and the corresponding image x. Similarly, draw another line passing through C', x'. Then two lines will intersect right at the point X. That is to say, we can track the original position of X from the position of its images x, x' in two different photos and the corresponding cameras C, C'. The calculation is based on epipolar geometry (Hartley, 2004), which will be introduced in detail in the next chapter.

For a more complicated real object, the 3D model reconstruction is achieved by manual or automatic analysis of the corresponding points in 2D images acquired by two or more cameras in a similar way. The points generated in 3D space reflect the depth information of objects present in the scene. Fig. 2-2 shows an example of image-based 3d modeling. The 2D images are selected from the Middlebury "Dino" data set. The images are taken by a fixed camera while the object spins.

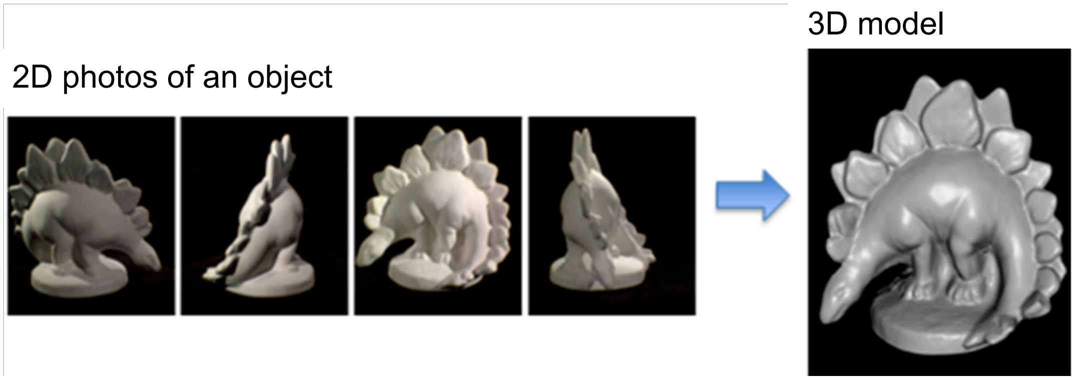


Fig. 2-2 Image-based 3D modeling (http://vision.middlebury.edu/mview/data/)

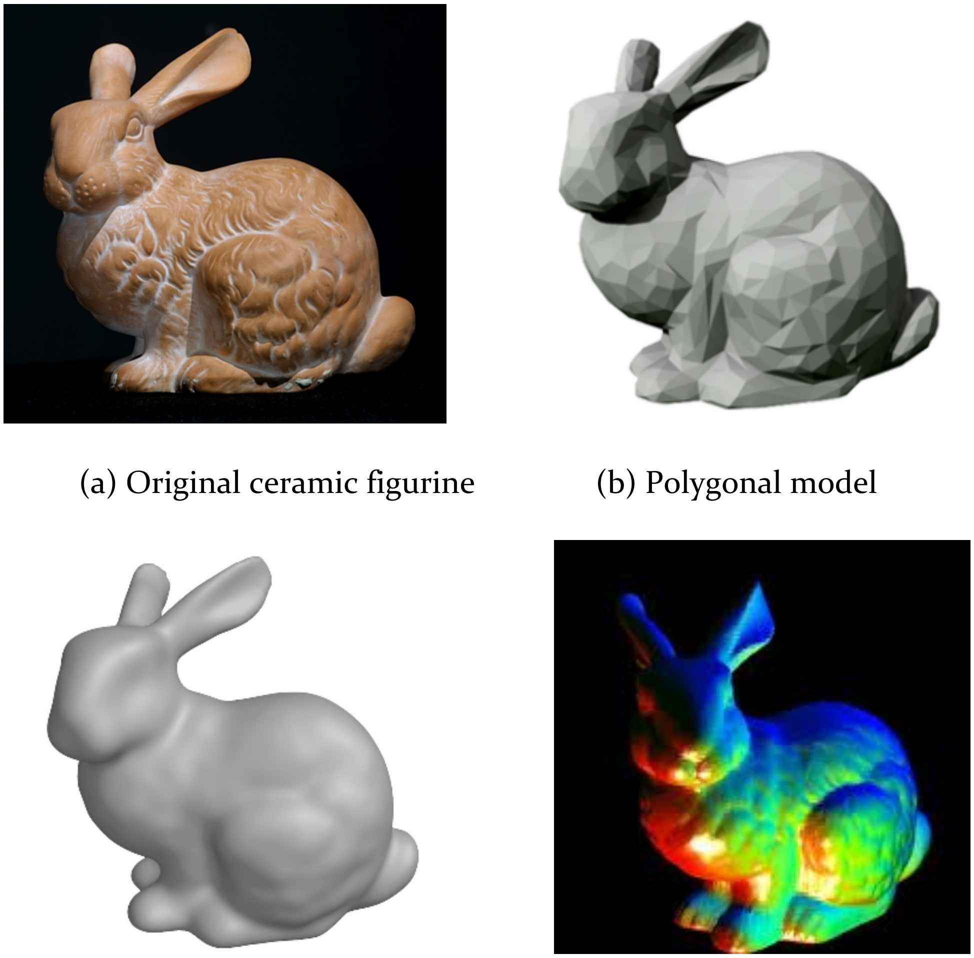
2.1.3 Presentation of 3D Models

The original point cloud extracted from the images consists of isolated points, while the final model should use points with continuous coordinates to fully describe every part of the object. Three popular ways to represent the final 3D model are shown below:

- Polygonal modeling (See Fig. 2-3(b)): Vertices are connected by line segments to form a polygonal mesh. The majority of 3D models today are built as textured polygonal models. They are very flexible and can be rendered quickly by computers. However, since polygons are planar, it can only approximate curved surfaces using many polygons.

- Curve modeling (See Fig. 2-3(c)): Surfaces are defined by curves influenced by control points. Increasing the weight for a point will pull the curve closer to that point.

- Sculpt modeling (See Fig. 2-3(d)): Using software to manipulate (push, pull, smooth, grab, pinch, etc.) a digital object as if it were made of a real-life substance, such as clay. It can be realize by displacement, volumetric, or dynamic tessellation and allows for very artistic exploration.



(a) Original ceramic figurine

(b) Polygonal model

(c) Curve model

(d) Sculpted model

Fig. 2-3 Stanford bunny and its 3D models,

http://graphics.stanford.edu/data/3Dscanrep/

2.1.4 Applications

Today, 3D models are used in a wide variety of fields. The medical industry uses detailed models to simulate real organs. The movie industry uses them as characters and objects for animated and real-life motion pictures. The video game industry uses them as basic assets for game design. The chemical industry uses them as highly detailed models of chemical compounds. The architecture industry uses them to demonstrate proposed buildings and landscapes. The engineering community uses them as designs of new devices, vehicles and structures. In recent decades the earth science community has started to construct 3D geological models as a standard practice. In addition, online marketplaces for 3D contents allow individual artists to sell contents that they have created, and companies can save money by buying pre-made models instead of paying an employee more to create one from scratch.

2.2 3D Building Models

3D building modeling has been an active research area in digital photogrammetry for a decade and a number of methods and systems have been developed for creating 3D city models from digital images and other auxiliary data automatically or semi-automatically.

2.2.1 Unique Characteristics

Different from extracting 3D model from a single object, 3D city modeling has its unique characteristics. Most buildings are designed with simple geometry shapes, consisting of straight edges, rectangular facades, planar or smooth surfaces. The walls are usually built vertically, while most roofs are built horizontally. The majority of the surface textures are several common materials. These characteristics can simplify the modeling process. But there are also some challenges for building reconstruction: more complicated environments lead to more noise; non-rigid objects like people or trees need to be ignored; shadows are always changing and bring difficulty in 2d registration. Repeated patterns may lead to failure. Local terrain needs to be considered. The final result needs to be presented in the world coordinate system.

2.2.2 Major Steps

As shown in Fig. 2-4, generating 3D city models involves many steps, including the point cloud extraction, filtering, segmentation, surface reconstruction and texture fitting. These steps can be divided into two major aspects which are creation of building models and adding textures to the building models. Various methods have been developed for creating building models from digital images automatically or semi-automatically. Since digital aerial images and LiDAR data supplement each other, accurate and reliable building extraction can be achieved ideally by fusing digital images and LiDAR data (You, 2011). Adding texture to the created building models makes 3D models more realistic. Different approaches have been developed to create building textures automatically from aerial vertical and oblique images.

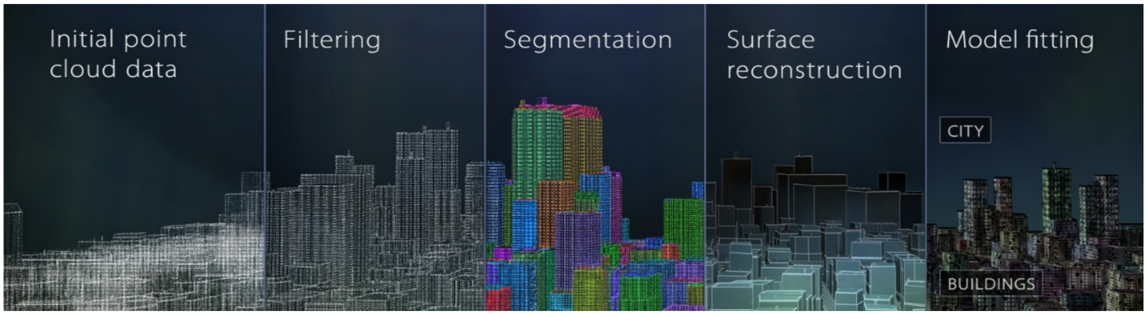


Fig. 2-4 Textured building modeling procedure (http://pointclouds.org/)

2.2.3 Current Research and Problems

In recent years 3D city models have been used for many applications such as:

- To visualize the cities for various purposes (e.g. virtual tours, visual military).
- In navigation or intelligent transportation systems.
- Build viewshed for urban planning.
- First responders or insurance estimation.

Techniques for 3D digitizing and modeling have been rapidly advancing over the past few years although most focus on single objects or specific applications. The ability to capture details and the degree of automation vary widely from one approach to another. One can safely say that there is no single approach that works for all types of environments and at the same time is fully automated and satisfies the requirements of every application. Automatically creating geometrically correct and complete 3D models of complex environments remains a difficult problem.

The most straightforward way to obtain 3D information is using LiDAR. Modern LiDAR systems are capable of receiving multiple returns with some penetrating vegetation, and thus, it can even reduce the effect of occlusions by combining the information from different returns. However, a problem with extraction of building models from LiDAR data is that the extracted models may not be very accurate because of point spacing, scanning angle, the performance of line extraction algorithm, etc. Therefore, building models derived from LiDAR data need to be refined, in order to create accurate 3D city models (Wang, 2008). To correct building models, they are projected back on the vertical image triangulated with accurate ground control points.

Although 3D data can be acquired using LiDAR systems directly, it can cost too much since it requires expensive devices and large memory. Compared to LiDAR, extracting 3D models from 2D images are more popular for commercial usage.

The community photo collections based 3D city reconstruction involves a series of state-of-the-art algorithms. In "Building Rome in a Day" (Agarwal, 2009) a parallel distributed system was suggested. It downloads millions of images from the web, matches them, computes the pose of the cameras that captured these images, and forms the 3D structure of the city automatically in one day. A huge community photo collection guarantees a detailed dense point cloud. As shown in Fig.2-5, the result model shows not only the detailed scene outside the building, but also the decorations from the inside, which is ideal for photo tourism for the famous sites.

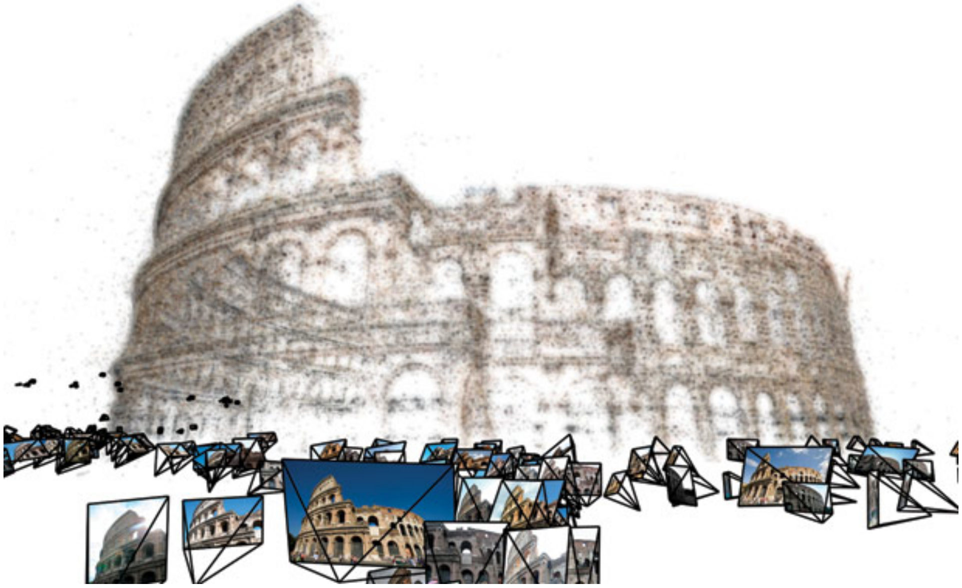


Fig. 2-5 3D reconstruction of the Colosseum from tourist photographs,

(http://grail.cs.washington.edu/rome/)

The community photo collection is a powerful type of image dataset. The images provide incredibly comprehensive information, but such a detailed result is not necessary for large scale 3D mapping. The extremely dense point clouds generated are computationally expensive and need large memory. Another problem is that these images have extreme variability, having been taken by numerous photographers from a myriad viewpoint with varying lighting and appearance, and often with significant occlusions and clutter. A third problem is that these photos limit the results to only places of interest. For a more general mapping demand, or where conditions make the ground-based photo impossible, like first responders after a disaster, we need a well planed data collection method that uses limited time and memory to obtain enough information for any target.

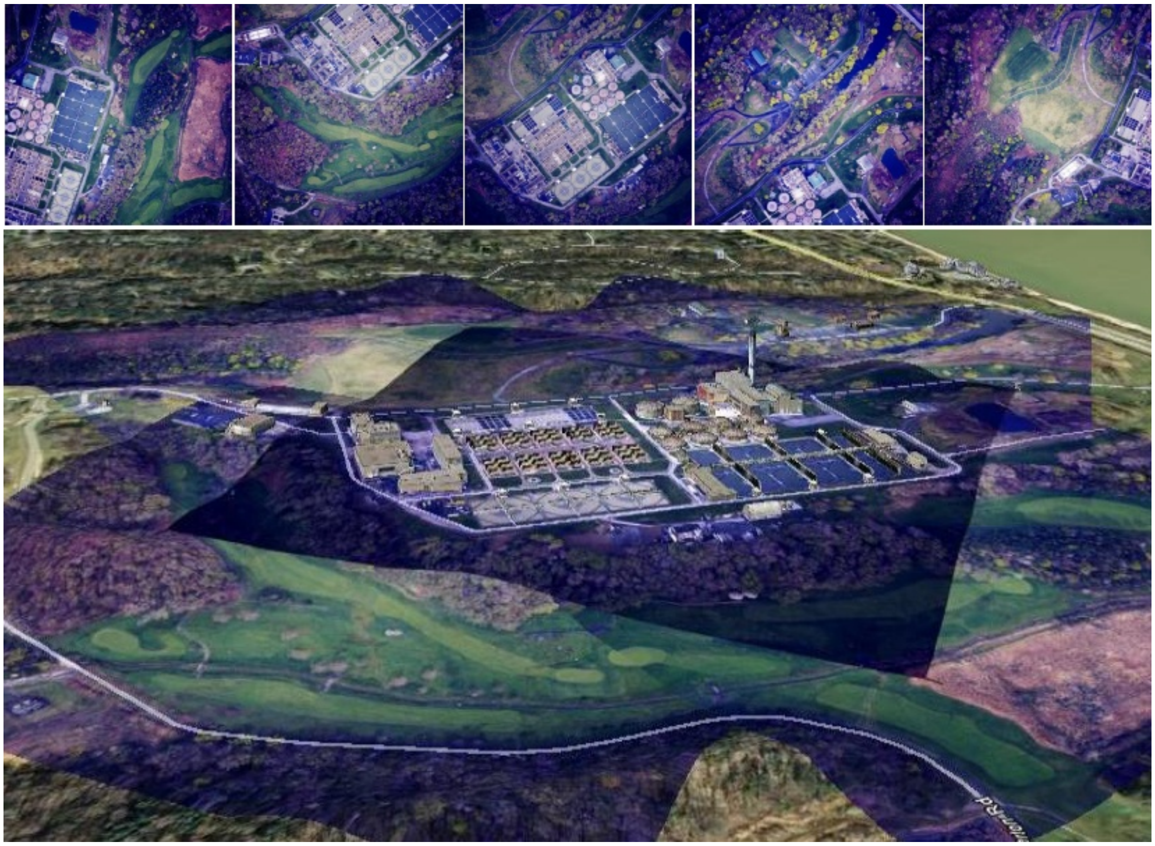
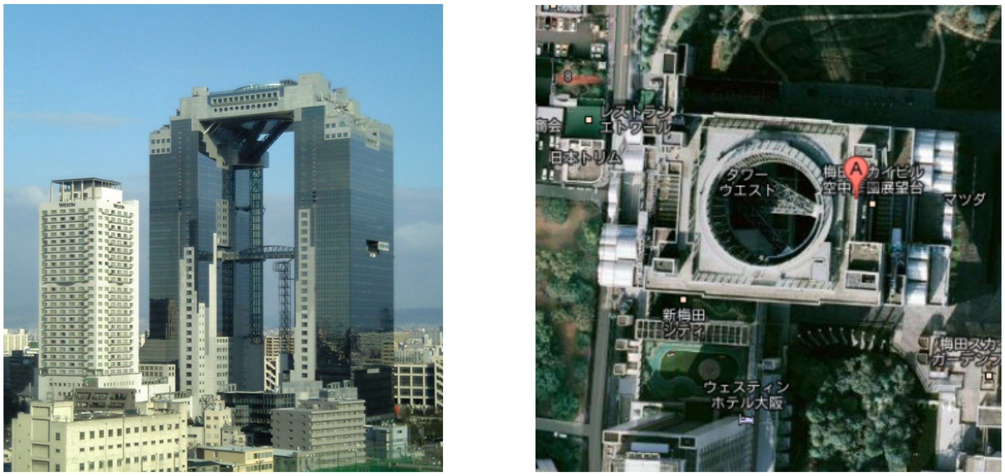


Fig. 2-6 3D reconstruction of Van Lare from nadir remote images (Nilosek, 2009)

Instead of community photo collection, the remotely sensed images use much smaller but well planed image collections to generate 3D models for any target area at a much larger scale. Well planed nadir image collections shorten the entire processing time by using less images and making the photogrammetry easier. Since remote images are taken from a much further distance, the resolution may not be as good as those of ground-based photos. But the resulting models still preserve the general geometry of buildings. Fig. 2-6 shows a 3D model of the Van Lare area (lower row), near Rochester, New York, derived from 5 nadir WASP images (upper row) by RIT researchers (Nilosek, 2009). The preliminary 3d point cloud of this model is extracted using a modified version of Agarwal's workflow, which will be introduced in details in the following chapter.



(a) Side view (b) Satellite view from Google Map

Fig. 2-7 Different views of the Omeda Sky Building,

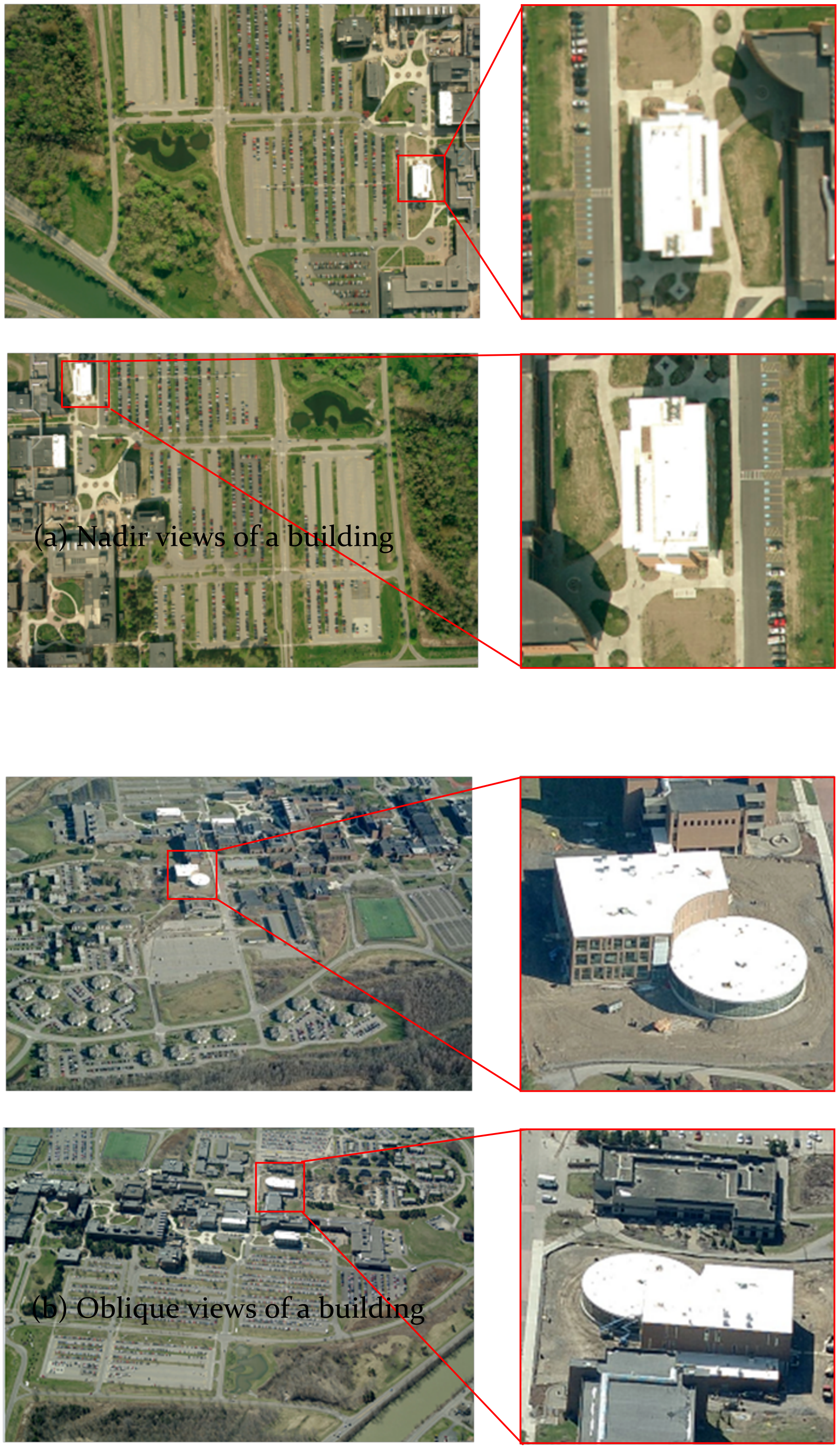
http://en.wikipedia.org/wiki/Umeda_Sky_Building

The results using only nadir remote imagery describe the roof tops well but have limited information about the side facets. Accordingly, a major problem with automatic approaches is that the extraction may fail when occlusions occur in the images. As shown in Fig. 2-7 (a), it is easy to tell that the blue building has an open structure under the roof from the side views. But there is no clue of this structure from the nadir views (as shown in Fig.2-7 (b)). If we try to extract a 3D model for this building but only use vertical images, the correct structure will definitely be missed. In this case, a new data type is needed to inherit the advantage and alleviate the limitation of nadir imagery. Airborne oblique imagery might be the one to solve this problem.

2.2.4 Oblique Imagery

Oblique images exhibit rich 3D like information of objects on the ground. So they could be used with nadir images for creation of detailed 3D city models. Furthermore, oblique images have advantages compared to vertical images in creating building textures (Frueh, 2004). They provide a better side view of building facades.

However, because of the nature of oblique imagery, it is difficult to match images taken from different viewing directions or integrate them with other information like the traditional nadir images. As shown in Fig. 2-8(a), the disparity between two nadir images is small, so it is not too hard for registration. But for oblique images (see Fig. 2-8(b)), because of the projective distortion, the disparity is huge. The scale is not constant in oblique images. The top level has a much lower resolution than the bottom level. And the shapes of facets, angles between edges, and neighbors in the scene change all the time. As a result, the features of images from different viewing direction are pretty hard to be matched to each other.



(a) Nadir views of a building

(b) Oblique views of a building

Fig. 2-8 Comparison of nadir and oblique images

An alternative method is to build a 3D point cloud for each group of oblique images taken from the same viewing direction. Then these point clouds could be combined with each other into a complete point cloud through a 3D registration method. They could also be used as a complement of point clouds extracted from nadir imagery or other source like LiDAR scanned data.

Thus, the work in this thesis has two main aspects: (1) Extract dense point cloud from oblique images automatically; (2) Implement automatic 3D registration to different point clouds with partial overlap. This research provides a practical approach to automatically reconstruct 3D building models with airborne oblique imagery, which has a promising market for future commercial usage in various areas like virtual tourism, navigation system, urban planning, visual military, and even insurance. The usage of oblique images for 3D city modeling can break the limitation of traditional vertical images. It not only preserves more detailed structure of the buildings compared to the nadir imagery, but also makes faster response possible compared to the ground-based method, and costs much less expense than using images taken by satellite or LiDAR scanning. Furthermore, the oblique images captured from different directions are ideal for generating building texture, based on further research in the 3D key feature extraction and surface identification. The 3D registration method proposed can also be used to improve the existing models by adding additional parts extracted from different sources.

Chapter 3

Methodology

In this thesis, we aim to apply the muti-view Pictometry airborne imagery to the extraction of a 3D scene of the RIT campus area. This process includes two main steps: 3D point cloud extraction and registration. The former involves 2D feature matching, sparse 3D point cloud extraction, dense point cloud reconstruction, and cloud refinement. The latter one involves 3D feature extraction, initial alignment and final registration. The theoretical basis of each part is stated in detail in this chapter.

3.1 Data

3.1.1 Pictometry Imaging System

At Pictometry International Corporation (http://pictometry.com/), a medium format digital imaging system has been well developed. It has been widely used for acquisitions of both vertical and oblique digital images. As shown in Fig. 3-1, this imaging system consists of five digital cameras, an integrated unit of Global Positioning System (GPS) and Inertial Measurement Unit (IMU), and a flight management system. Each camera has a CCD array with about 4.9k × 3.2k pixels. The five digital cameras are arranged in such a way that one of them looks straight down and the other four of them look into forward, backward, left and right directions respectively at a certain viewing angle (≈ 40°). The camera in the vertical direction captures high-resolution vertical images and the other four acquire oblique images at different view directions at the same time. The onboard GPS and IMU provide an accurate position and attitude of each sensor at the exposure time, thus the images produced by Pictometry imaging system are directly geo-referenced images.

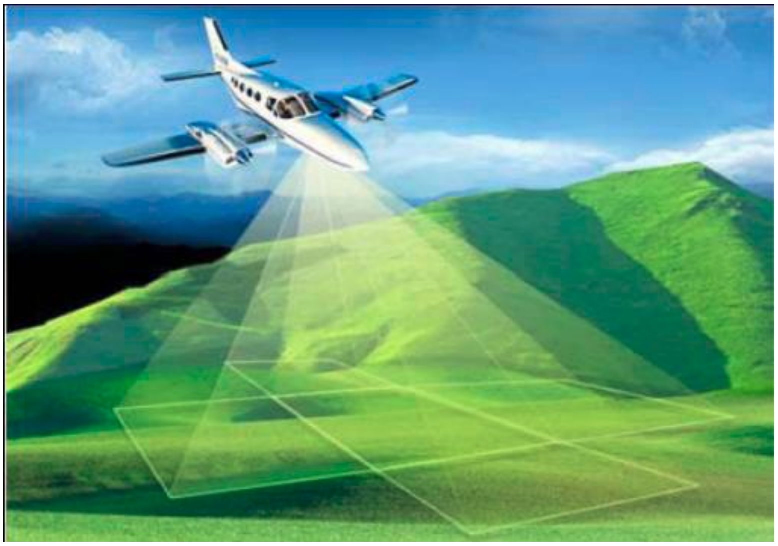


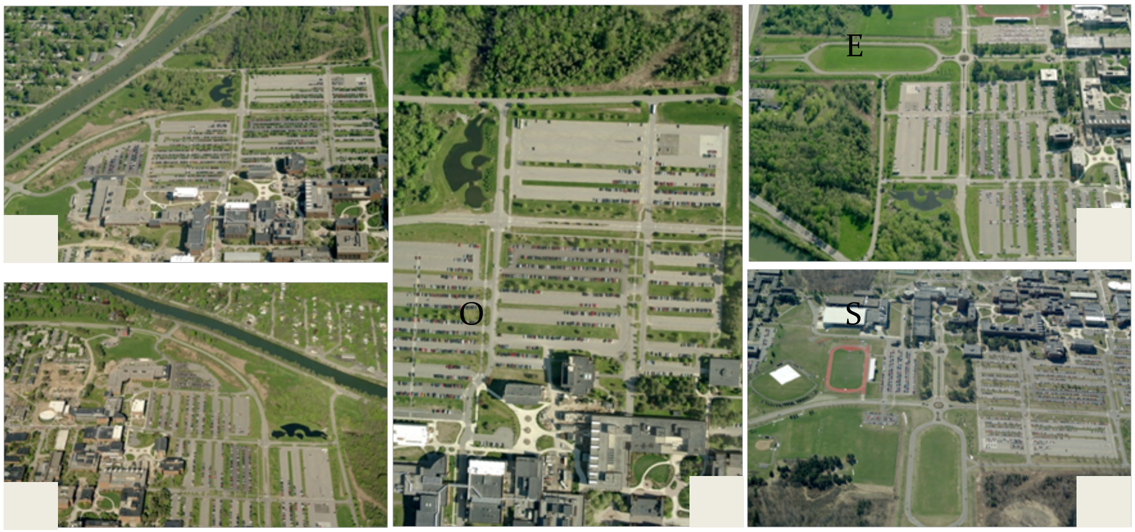
Fig. 3-1 Pictometry camera sensor systems (Lemmens, 2007)

Like traditional aerial images, vertical images provide vertical views of the terrain surfaces, while oblique images show the side views of objects on the ground, like buildings. Vertical images can be used for creation of accurate large scale ortho photos and oblique images can be utilized for visualization, measurement and 3D modeling. Up to now, more than 50 Pictometry imaging systems have been developed in the USA and tens of systems are being used around the world. These images have been widely used in various applications such as public safety, tax assessment, urban planning, 3D city modeling, etc. The flight management system is a flight planning software which determines flight lines, control image overlaps, etc. before and during the flight for both vertical and oblique images. In order to better use and visualize both oblique and ortho images, a software package called Electronic Field Study (EFS) has been developed at Pictometry. Both vertical and oblique images can be easily viewed in EFS, and spatial measurements such as distance and height of objects on the ground can be easily performed on both oblique and vertical images. The results can be exported into ArcGIS (a geographic information system) directly to update the existing geo-spatial information in the database.

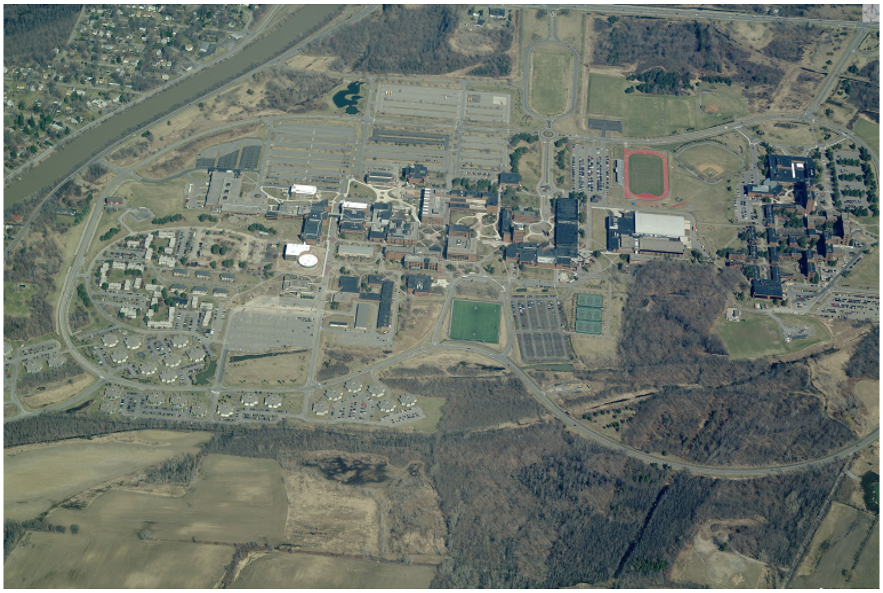
3.1.2 RIT Campus Images

There are two kinds of Pictometry airborne imagery with different resolutions. They are called neighborhood imagery and community imagery respectively.

The neighborhood imagery is taken at the altitude of around 4500 feet. Fig. 3-2(a) shows an example of the neighborhood images taken over the same area from five



(a) Neighborhood images



(b) Community image

Fig. 3-2 Pictometry imagery of RIT campus area

different viewing directions. These 8bit color images have the resolution of 4872× 3248 pixels, and are stored in ".psi" format. The oblique images are taken at a focal length of around 85mm with a resolution of around 0.55 feet/pixel and 60%

overlap, while the orthogonal images are taken at a focal length of around 65mm with a resolution of around 0.46 feet/pixel and 30% overlap. The focal plane is 36.053 mm wide × 24.035 mm high.

The community imagery is taken using the same system, but at a higher altitude. As shown in Fig. 3-2(b), these images have a larger scale than the neighborhood images. For this project, only the neighborhood imagery is used.

3.1.3 Meta Data

There is a ".txt" file provided along with the each image. It includes the following additional information about the image:

- Creation Date & viewing direction
- Per pixel resolution
- GPS coordinates of four shot corners
- Camera location, altitude and average Elevation
- Camera pitch, roll and azimuth angle
- Focal length and focal Plane size
- Principal point offset

3.2 Proposed Method

The main steps of the proposed method for 3D Point matching are shown in the following chart in Fig. 3-3.

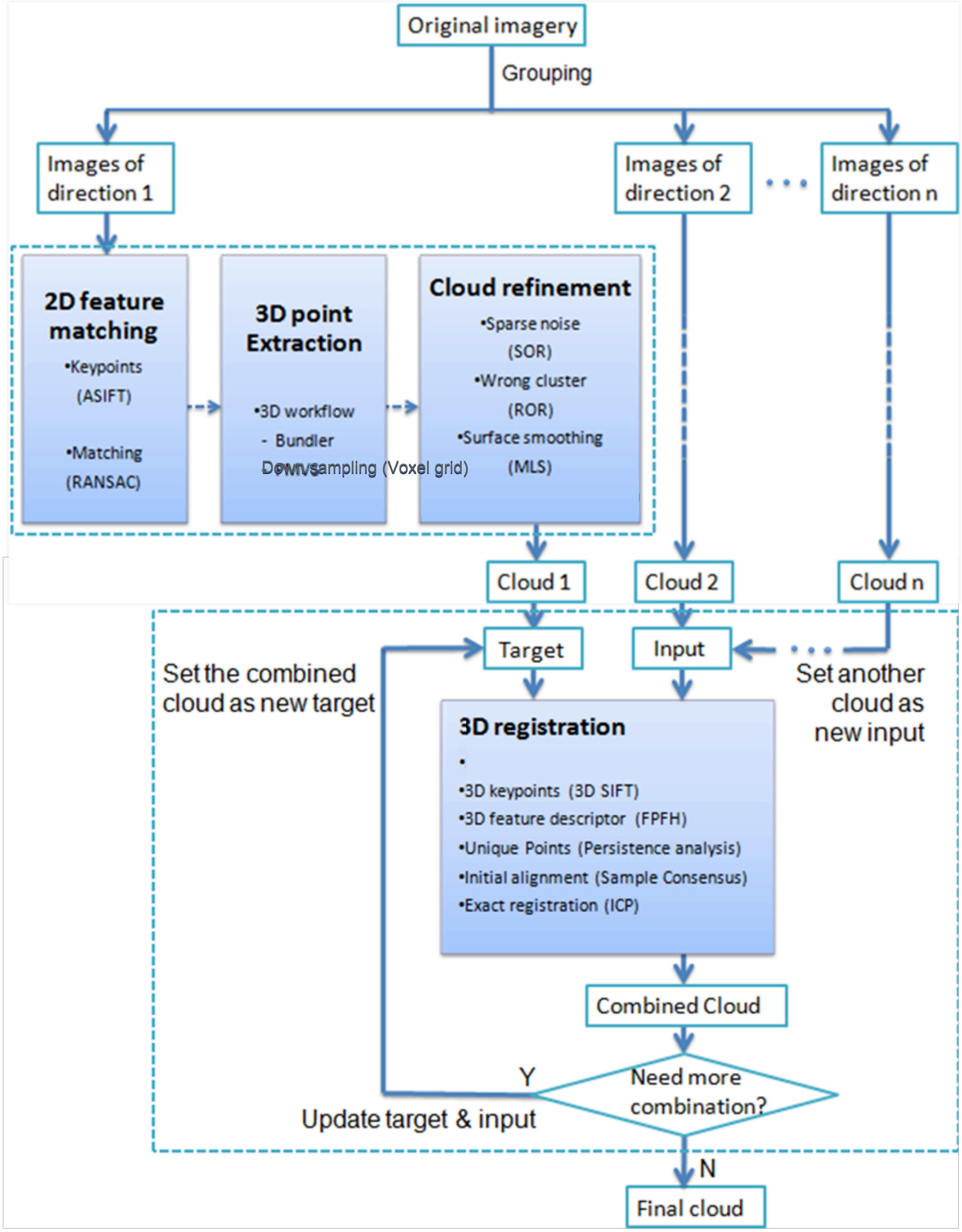


Fig. 3-3 Main steps for 3D point extraction and matching

The research will begin with building several 3D point clouds. The multi-view imagery provided by Pictometry International Corporation is divided into different groups according to the viewing direction. For each group, the geometry extraction process is based on a modified version of the RIT 3D Extraction Workflow. First, it uses the Affine Scale Invariant Feature Transform (ASIFT, Morel, 2009) algorithm for the 2D feature extraction. Then the RANdom SAmple Consensus (RANSAC, Fischler, 1981) is used to match the keypoints. After that, the Bundler calculates the camera parameters and reconstructs the sparse scene using Sparse Bundle Adjustment (SBA, Lourakis, 2004). Finally, the Patch-Based Multi-View Stereo (PMVS, Furukawa, 2007) is used to reconstruct a dense scene. The resulting point clouds are presented in independent coordinate systems.

The next step is to refine the original point clouds generated by the above workflow. The clouds here are quite noisy. First, a Statistical Outlier Removal (SOR) method is used to eliminate the sparse noise spreading all over the space. Then, a Radius Outlier Removal (ROR) method is used to further remove the remained miscalculated floating clusters. In the end, a Moving Least Square (MLS) method is used to smooth the surface. These filters are from the Point Cloud Library (PCL, http://pointclouds.org/documentation).

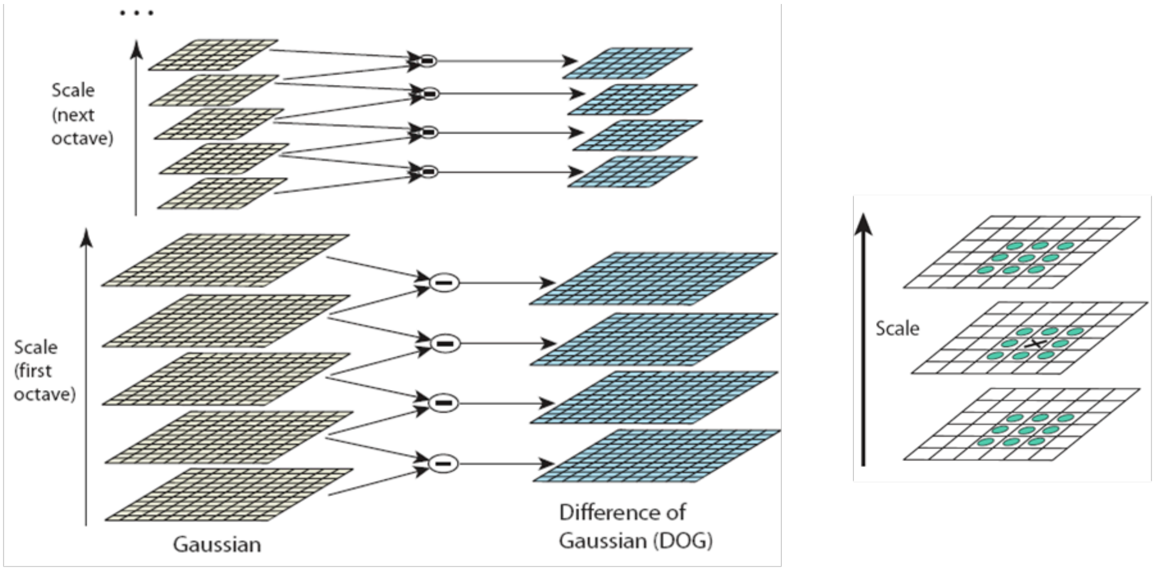
The last step is to perform the 3D registration for each refined point cloud pair. First, the original clouds are downsized for faster processing. Then the 3D SIFT method is used to extract 3D keypoints from the point clouds, and the Fast Point Feature Histograms (FPFH, Rusu, 2009) method is used to describe the features of the keypoints. A multi-scale feature persistence analysis process is performed to find the points that have unique features. The preliminary registration uses the sample consensus method to find point pairs with similar features from those points and achieve a rough alignment. Once the initial positions of the point clouds are well estimated, the Iterative Closest Points (ICP, Zhang, 1993) algorithm will be used to obtain a more exact registration. The resulting transformation matrix contains the information of scale, rotation and translation changes. Using the same 3D registration method, more point clouds can be combined together.

3.3 Algorithms and Implementation

3.3.1 2D Feature Matching

3.3.1.1 Affine Scale Invariant Feature Transform

As mentioned in chapter 2.1.2, the 3D geometry extraction from multi-view imagery is based on accurately matching of 2D features. There are many state-of-the-art algorithms that could be used to detect and describe features in an image. All of these algorithms are translation invariant. Some are also rotation invariant, like the Harris corner point detector (Harris, 1988). Some are even invariant to changes of scale, like Hessian-Laplace region detector (Mikolajczyk, 2004). Some are designed to be affine invariant, like edge detector (Tuytelaars, 2004) and maximally stable extremal regions (MSER, Matas, 2004). Among these methods, the scale invariant feature transform (SIFT, Lowe, 2004) is proved to be robust to scaling and rotation changes, and partially invariant to illumination and viewpoint changes. By adding two parameters, its updated form affine-SIFT (ASIFT, Morel, 2009) becomes fully affine invariant. Since we aim to find corresponding features between images taken from different positions at different time, we choose ASIFT for the feature extraction.



(a) Octave of scale space (b) Local neighbors

Fig. 3-4 Scale-space extrema detection (Lowe, 2004)

1. Scale Invariant Feature Transform

The Scale Invariant Feature Transform (SIFT) algorithm is currently one of the most popular feature detectors used to provide reliable matching between different views of a scene. The method mainly includes the following 4 steps:

i) Detect the scale-space extrema.

As shown in Fig. 3-4(a), according to Eq. 3-1~3-3, for each octave of scale space, the initial image $I(x,y)$ is repeatedly convolved with Gaussians $G(x,y,\sigma)$ to produce the set of scale space images $L(x,y,\sigma)$ shown on the left. Adjacent Gaussian images are subtracted to produce the difference-of-Gaussian images $D(x,y,\sigma)$ on the right. After each octave, the Gaussian image is down-sampled and the process repeated.

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y) \tag{3-1}$$

$$G(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \tag{3-2}$$

$$D(x,y,\sigma) = \big(G(x,y,k\sigma) - G(x,y,\sigma)\big) * I(x,y) = L(x,y,k\sigma) - L(x,y,\sigma) \tag{3-3}$$

where σ is the scale parameter, k is a constant multiplicative factor. As shown in Fig. 3-4(b), local maxima and minima of the difference-of-Gaussian images are detected by comparing a pixel (marked with ×) to its 26 neighbors in 3x3 regions at the current and adjacent scales (marked with circles).

ii) Locate the keypoints accurately and reject poor candidates.

A detailed fit is performed to the nearby data for location, scale, and ratio of principle curvatures. Points having low contrast or localized along an edge will be rejected. The Taylor expansion (shift the origin to the sample point) of D is

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T \frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x} \tag{3-4}$$

By take the derivative of D with respect to $\mathbf{x} = (x, y, \sigma)^{\mathrm{T}}$ and set it to zero, the location of the extreme $\hat{\mathbf{x}}$ is obtained.

$$\hat{\mathbf{x}} = -\frac{\partial^2 D^{-1}}{\partial \mathbf{x}^2}\frac{\partial D}{\partial \mathbf{x}} \tag{3-5}$$

All extrema with the function value $\mathrm{D}(\hat{\mathbf{x}})$ lower than the threshold will be discarded.

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2}\frac{\partial D^T}{\partial \mathbf{x}}\hat{\mathbf{x}} \tag{3-6}$$

iii) Assign the orientation.

At the scale of the keypoint, the gradient magnitude $\mathrm{m}(\mathrm{x}, \mathrm{y})$ and orientation $\theta(\mathrm{x}, \mathrm{y})$ of each image sample $\mathrm{L}(\mathrm{x}, \mathrm{y})$ are computed by Eq. 3-7. Peaks in the orientation histogram formed from the gradient orientations of sample points around the keypoints correspond to dominant directions of local gradient.

$$\begin{aligned} m(x,y) &= \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \\ \theta(x,y) &= \tan^{-1}((L(x,y+1) - L(x,y-1)) / (L(x+1,y) - L(x-1,y))) \end{aligned} \tag{3-7}$$

iv) Present the descriptor.

As shown in Fig. 3-5, keypoint descriptors are created by computing the gradient magnitude and orientation, Gaussian weighted by the pixels location surrounding a keypoint. These samples are then accumulated into 8 bin orientation histograms, which summarize a 4×4 sub-region. The final descriptor for a keypoint is a vector consists of 128 elements.

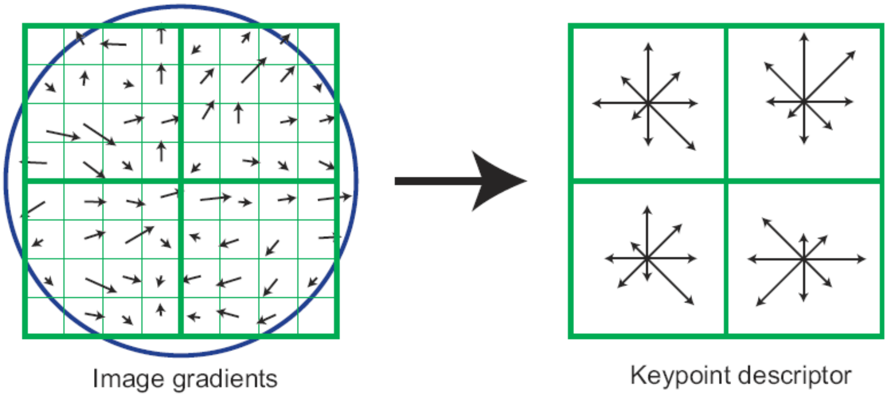
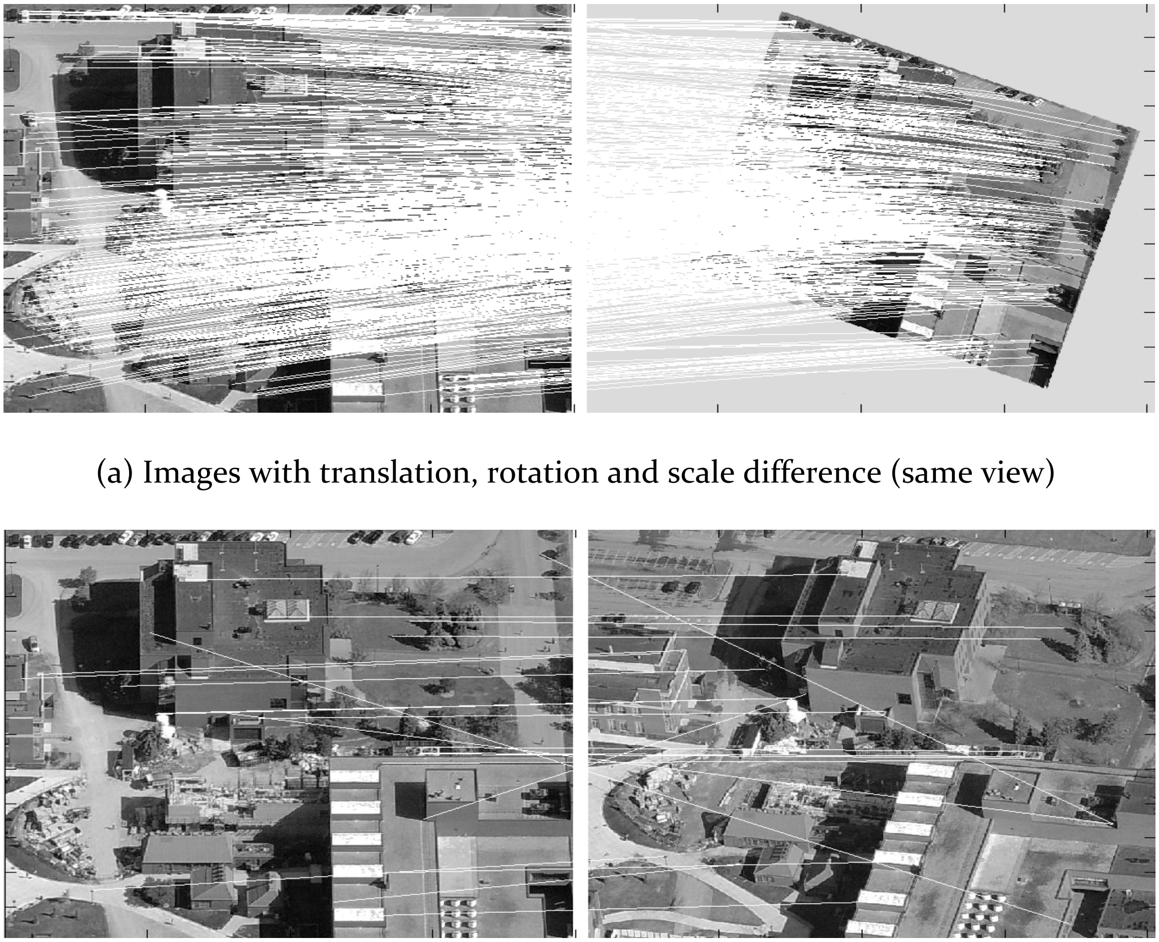


Fig. 3-5 SIFT Keypoint descriptor (Lowe, 2004)



(a) Images with translation, rotation and scale difference (same view)

(b) Images with project distortion (two different views)

Fig. 3-6 Corresponding keypoints extracted from two images using SIFT

By comparing the descriptors of keypoints in two images, matched points are found. As shown in Fig. 3-6. SIFT method is robust to translation, rotation and scale difference between images (see Fig. 3-6(a), matched points are connected by lines), but not to affine difference (see Fig. 3-6(b)), which exists between the airborne images we used. So when there is bigger disparity, SIFT might fail.

2. Affine Scale Invariant Feature Transform

Because SIFT normalizes rotations and translations, and simulates all zooms out of the query and of the search images, it is invariant for zoom, rotation and translation, with respect to four out of the six parameters of an affine transform. To achieve fully affined invariant image comparison, the Affine Scale Invariant Feature Transform (ASIFT, Morel, 2009) treats the two left out parameters: the angles defining the camera optical axis orientation.

According to Fig. 3-7, an affine map A can be expressed as

$$A = \lambda \begin{bmatrix} cos\psi & -sin\psi \\ sin\psi & cos\psi \end{bmatrix} \begin{bmatrix} t & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} cos\phi & -sin\phi \\ sin\phi & cos\phi \end{bmatrix} = \lambda R(\psi) T_t R(\phi) \qquad (3\text{-}8)$$

where ϕ and θ are the camera viewpoint angles, ψ parameterizes the camera spin. $\lambda > 0$, $\phi \in [0, 180°)$, $R(\psi)$denotes the planar rotation with angle ψ, and T_t is called the tilt. The absolute tilt t is defined as $t = f/cos\theta$. If the absolute tilt of another image from different view is marked as $t^{'}$, then the transition tilt is defined as

$$\tau = t'/t \qquad (3\text{-}9)$$

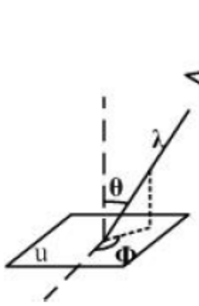


Fig. 3-7 Camera motion(Morel, 2009)

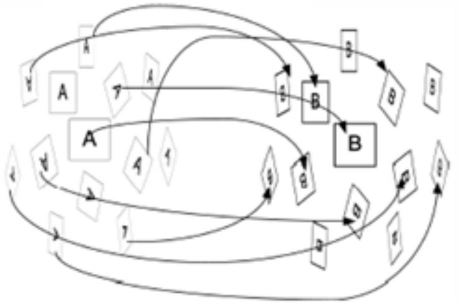


Fig. 3-8 The idea of ASIFT (Morel, 2009)

As described in Fig. 3-8, ASIFT algorithm is based on the comparison of many pair of rotated and tilted images obtained from A and B by SIFT. It mainly includes:

i) Transform each image by simulating many possible affine distortions caused by the change of camera optical axis orientation from a frontal position. The distortions depend on the longitude ϕ and the latitude θ. The images undergo ϕ-rotations followed by tilts t.

ii) These rotations and tilts are performed for a finite and small number of ϕ and θ. The sampling steps of these parameters ensures that the simulated images keep close to any other possible view generated by values of ϕ and θ.

iii) All simulated images are compared by SIFT. Since SIFT normalizes the translation of the camera parallel to its focal plane and the rotation of the camera around its optical axis, but simulates the scale change, all six camera parameters are either normalized or simulated by ASIFT. So the descriptor of ASIFT keypoints is similar to that of SIFT, which consists of 128 vectors.

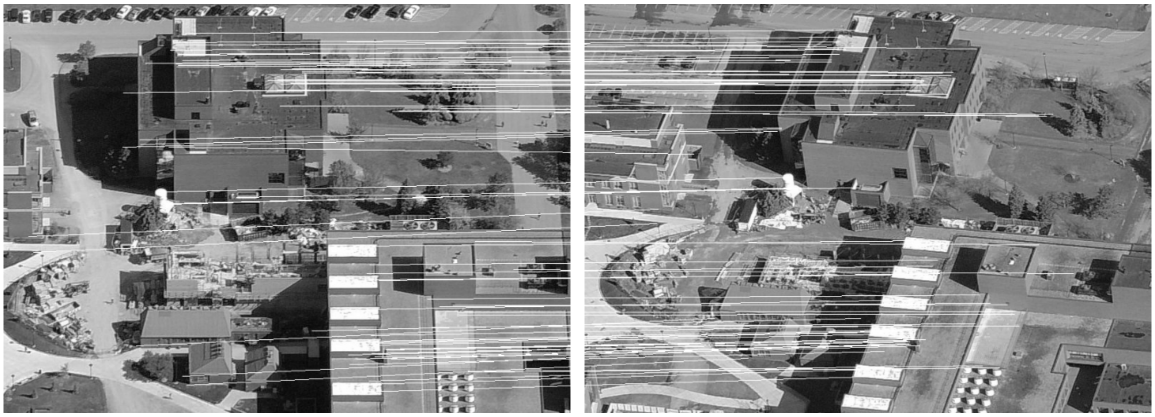


Fig. 3-9 Corresponding keypoints extracted from two images using ASIFT

As shown in Fig. 3-9, ASIFT does obtain a better result than SIFT (see Fig. 3-6). There are much more correct matched points found between this image pair when the affine distortion is considered.

3.3.1.2 RANdom SAmple Consensus

A huge number of keypoints could be extracted from ASIFT. We need to find an efficient way to find the correct correspondences and eliminate the bad ones. Unlike that of conventional smoothing techniques, RANdom SAmple Consensus (RANSAC, Fishler, 1981) procedure uses as small initial data set as feasible and enlarges this set with consistent data when possible, rather than using as much of the data as possible to obtain an initial solution and then attempting to eliminate the invalid data points. The main steps of RANSAC are shown as follows:

i) Given a model that requires a minimum of n data points to determine its free parameters and a set of points P (the number of points in P is greater than n), randomly select a subset $S1$ of n points from P and estimate the instantiated model $M1$. Determine the consensus set $S1^*$ (subset of P) that are within some error tolerance of $M1$.

ii) If the number of points in $S1^*$ is greater than some threshold t, use $S1^*$ to compute a new model $M1^*$. Otherwise, randomly select a new subset $S2$ and repeat the above process.

iii) After some predetermined number of iteration, if no consensus set includes more members than t, terminate in failure. Otherwise, solve the model with the largest consensus set found.

In this thesis, the target model is set according to epipolar geometry to find the correct corresponding pairs.

3.3.2 3D Point Cloud Extraction

3.3.2.1 Epipolar Geometry

The epipolar geometry is the basis of stereo reconstruction. The geometry between two views is essentially the geometry of the intersection of the image planes with the pencil of planes having the baseline as axis. The baseline is the line joining the camera centres. The epipole is the point of intersection of the baseline with the image plane. An epipolar plane is a plane containing the baseline. An epipolar line is the intersection of an epipolar plane with the image plane.

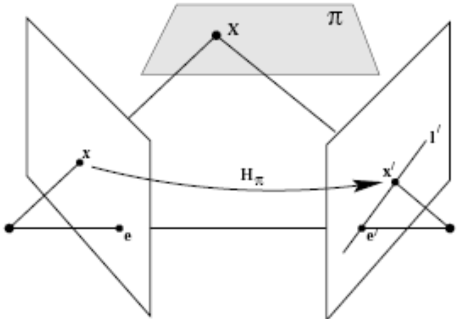


Fig. 3-10 Epipolar geometry (Hartley, 2004)

1. Fundamental Matrix

As shown in Fig. 3-10, a point x in one image is transferred via the plane π to a matching point x' in the other image. The epipolar line l'through x' is obtained by joining x' to the epipole e'. In symbols one may write

$$l' = Fx \tag{3-10}$$

where F is a 3×3 homogenous matrix (Luong, 1996), called the Fundamental Matrix. It satisfies the condition that for any pair of corresponding points $x \leftrightarrow x'$

$$x'^{\mathrm{T}}Fx = 0 \tag{3-11}$$

So, for a given point, the preliminary match point must lie along the epipolar line in order for it to be valid. That is to say, the matches that do not fit this epipolar constraint described by Eq. 3-11 are then eliminated.

Even with the presence of several outliers, these relationships can be utilized in concert with RANSAC to develop a robust Fundamental Matrix. Once this is accomplished, the Fundamental Matrix can then be used to constrain the ASIFT match set to remove most outliers. Unfortunately, it is possible that an erroneous set of correspondences may still fulfill the Fundamental Matrix constraints, so additional constraints may be required to further cull the data.

2. Camera Matrix

The ray back-projected from point x in an image by camera matrix P to point X in the world coordinate system is obtained by solving

$$\mathrm{PX} = x \qquad (3\text{-}12)$$

where $\mathrm{P} = K[R|t]$. K is the camera calibration matrix, containing the internal camera parameters, focal length f and the coordinates of principal point (p_x, p_y)). $t = -R\tilde{\mathrm{C}}$, R is the rotation matrix, $\tilde{\mathrm{C}}$ is the camera center. The parameters contained in R and $\tilde{\mathrm{C}}$ are called the external camera parameters.

3.3.2.2 Bundler

The Bundle Adjustment (BA, Triggs, 2000) algorithm is almost invariably used as the last steps of every feature-based structure and motion estimation vision algorithm to obtain 3D structure and viewing parameters. Its name refers to the bundles of light rays originating from each 3D feature and converging on each camera center, which are adjusted optimally with respect to both structure and viewing parameters under certain assumptions regarding the noise pertaining to the observed image features. It amounts to minimizing the re-projection error between the observed and predicted image points, which can be achieved using the Levenberg-Marquardt (LM, More, 1978) method.

By iteratively linearizing the function to be minimized in the neighborhood of the current estimate, LM involves the solution of linear systems known as the normal equations. These equations are solved repeatedly and LM can be computationally demanding. Consider the sparse block structure of the normal equation matrix owing to the lack of interaction among parameters for different 3D points and cameras, Lourakis (2004) developed a tailored sparse variant of LM. The so-called Sparse Bundle Adjustment (SBA) software package explicitly takes advantage of the normal equations zero patterns.

Bundler is a software package based on the Photo Tourism work (Snavely, 2008). It is a structure-from-motion system for unordered image collections (for instance, images from the Internet). Bundler takes a set of images, image features, and image matches as input, and produces a 3D reconstruction of the camera and scene geometry (presented in sparse point cloud) as output. The system reconstructs the scene incrementally, a few images at a time, using a modified version of SBA as the underlying optimization engine.

Bundler has a number of internal parameters. The input of bundler is images, keypoints (in SIFT style), and matches. The outputs are camera parameters and sparse scene geometry.

3.3.2.3 Patch-Based Muti-View Software

Patch-Based Muti-View Software (PMVS, Furukawa, 2007) is a multi-view stereo software package that takes a set of images and camera parameters and reconstructs the 3D structure of an object or a scene (presented in dense point cloud) visible in the images. Only rigid structures are reconstructed (i.e. the software automatically ignores non-rigid objects such as pedestrians in front of a building).

The software outputs a set of oriented points instead of a polygonal (or a mesh) model, where both the 3D coordinate and the surface normal are estimated at each oriented point. PMVS has various parameters and flags for the software in the option file. The input files for PMVS are images and camera parameters. The output are colored, oriented point cloud stored in ‘ .ply’ file.

3.3.2.4 RIT 3D Extraction Workflow

The RIT 3D Extraction Workflow is a packaged software developed by RIT researchers David Nilosek and Harvey Rhody (http://dirsapps.cis.rit.edu/3d-workflow/index.html). It integrates SIFT, bundler and PMVS, and is coded in Python. This workflow is designed for extracting 3D dense point cloud from nadir images. To apply it to the Pictometry imagery, ASIFT is used instead of SIFT in this thesis to obtain more corresponding point pairs. The focal length is also fixed at the known value, so bundler can skip the focal length estimation, which leads to more accurate results.

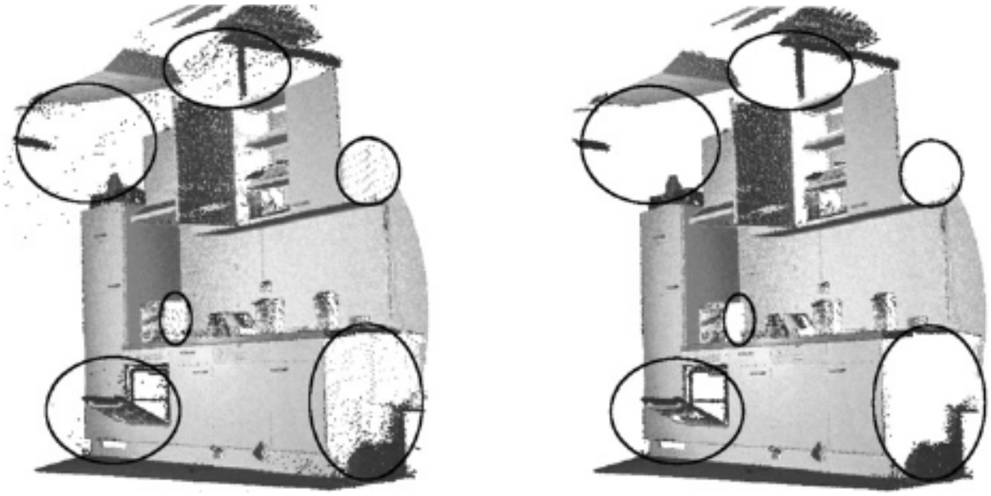
3.3.3 Cloud Refinement

The point clouds generated from PMVS are quite noisy because of the calculation errors. The existing sparse outliers and wrong floating clusters need to be eliminated using different methods. In the meantime, to better estimate the feature of points in the clouds, especially for surface normals, a surface smooth process is necessary.

3.3.3.1 Noise Removal

1. Statistical outlier removal

As shown in Fig. 3-11(a), there can be lots of sparse noise points in the cloud. These sparse outliers of a data set can be filtered by performing a statistical analysis on each point's neighborhood, and trimming those which do not meet a certain criteria (see Fig. 3-11(b)). The Statistical Outlier Removal (SOR) method is based on the computation of the distribution of point to neighbors' distances in the input



(a) Before removal

(b) After removal

Fig. 3-11 Statistical outlier removal (Rusu, 2008)

dataset (PCL API documentation 1.7.0). For each point, the mean distance from it to all its neighbors is computed. By assuming that the resulted distribution is Gaussian with a mean μ and a standard deviation σ, all points whose average neighbor distances are outside an interval $\mu \pm \alpha \cdot \sigma$ defined by the global distances mean and standard deviation can be considered as outliers and trimmed from the dataset. The value α depends on the size of the analyzed neighborhood. The algorithms iterates through the entire input twice. During the first iteration it will compute the average distance that each point has to its nearest k neighbors. Then, the mean and standard deviation of all these distances are computed in order to determine a distance threshold. During the next iteration each point will be classified as an inlier or outlier if its average neighbor distance is below or above this threshold respectively.

2. Radius outlier removal

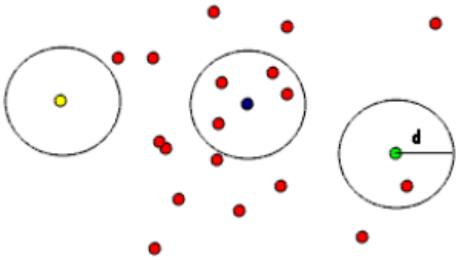


Fig. 3-12 Radius outlier removal (PCL, 2012)

Besides the sparse point noise, there are also some misestimated small clusters floating far away from the main cluster in the original point cloud. The point density in the cluster is as dense as the other correct estimated points, so this kind of noise cannot be eliminate by SOR. Radius Outlier Removal (ROR) decides outliers in a cloud based on the number of neighbors they have (PCL API documentation 1.7.0). If we set the search radius larger than that of the noise cluster, the outliers will have much less neighbors than the inliers. The algorithm iterates through the entire input once, and for each point, retrieves the number of neighbors within a certain radius. The point will be considered an outlier if it has too few neighbors, as determined by a certain minimum neighbor radius. As shown in Fig. 3-12, the yellow point is definitely an outlier. If we set the threshold at “2 neighbors”, then the green point is also an outlier.

t

3.3.3.2 Surface Smoothing

The goal of 3D modeling is to render the surface of buildings. The points in the original cloud are located around but not exactly on the "true surface". These errors will lead to failure in normal or curvature estimation for the surface. Moving Least Squares (MLS) algorithm can be used to mitigate this problem by data smoothing (PCL, 2012), which relies on the idea that the given point set implicitly defines a surface.

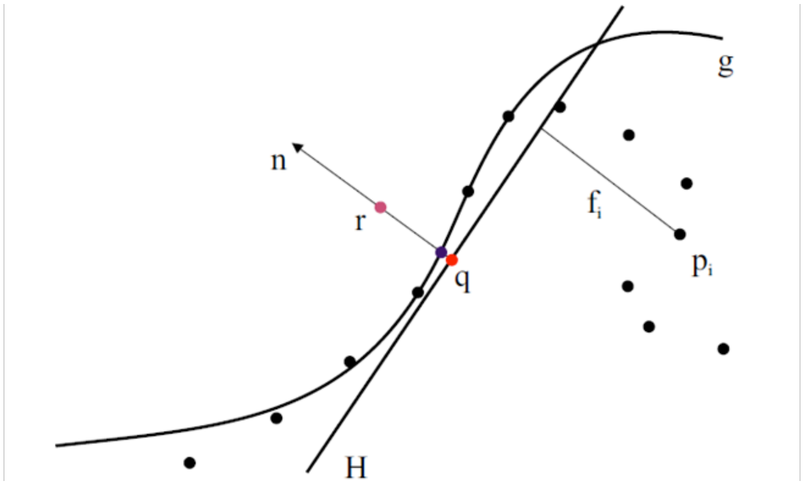


Fig. 3-13 MLS projection procedure (Alexa, 2003)

MLS projects the points close to the original surface on to a new smoothed surface, based on local maps from differential geometry. The approximation error is bounded and can be controlled by increasing or decreasing the density of the result point cloud. Fig. 3-13 shows the procedure of MLS. Points pi are sampled from the original surface and the goal is to project the purple point r near the original surface onto a new surface that approximates the pi. First, a local reference plane H for r is generated by minimizing a local weighted sum of square distances of pi to H. The projection of r onto H defines its origin q (the red point). The distance between each p_i and q is used as the weight function. Let points qi be the projection of pi onto H, and fi the heights of points p_i over H. Then, the local polynomial approximation g is obtained by minimizing a weighted least squares error between g and fi. The blue point t shows the result of the MLS projection procedure, which is the projection of r onto g.

3.3.3.3 Normal Estimation

Surface normals are important properties of a geometric surface. The estimation of the normal at each point in the point cloud is based on its relationships with the nearby *k* points surrounding it. This information is then used for computing persistent features and registration. A fast and accurate estimation requires both a method for determining the best *k*-neighborhood support for the query point and a way of estimating the surface normal at the query point.

The estimated normal $\boldsymbol{n}$ of the point p can be approximated with the normal to the *k*-neighborhood surface by performing PCA (Principal Component Analysis) on the neighbors' covariance matrix (Paully, 2002). The eigenvector corresponding to the smallest eigenvalue gives an estimate of $\boldsymbol{n}$'s direction. The MLESAC (Maximum Likelihood Estimation SAmple Consensus, Torr, 2000) technique is used to robustly estimate the best support for a plane and discard the outliers. As shown in Fig. 3-14, the normal of a point is calculated from the normal of the plane fitted by its neighbors. The covariance matrix from the points p_i of the support neighborhood is defined by (Rusu, 2008)

$$C = \sum_{i=1}^{k} \xi_i \cdot (p_i - \bar{p})^t \cdot (p_i - \bar{p}), i = 1 \dots k \tag{3-13}$$

The eigenvector V and eigenvalue λ is computed for C. The term ξ_i represents the weight for point p_i

$$\xi_i = exp\,(-\frac{d_i^2}{\mu^2}) \tag{3-14}$$

If p_i is outlier, $\xi_i = 1$. If not, $\xi_i = -1$. μ is the mean distance from the query point p to all its neighbors p_i, and d_i is the distance from point p to a neighbor p_i.

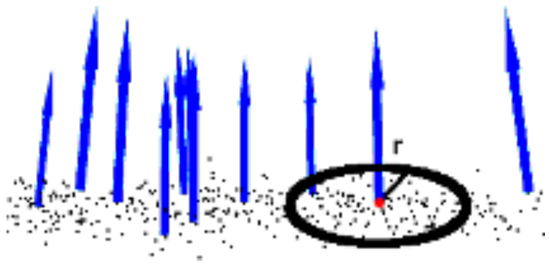


Fig. 3-14 Normal Estimation (PCL, 2012)

3.3.4 3D Feature Extraction

3.3.4.1 Resampling

The original point cloud consists of a large amount of points. To save the memory and to speed up the calculation, we perform down sampling using a voxelized grid (see Fig. 3-15). A 3D voxel grid is like a set of tiny 3D boxes. In each voxel, all the points present will be approximated with their centroid. In this case, though the number of points in the cloud is shrunk, most characteristics are preserved.

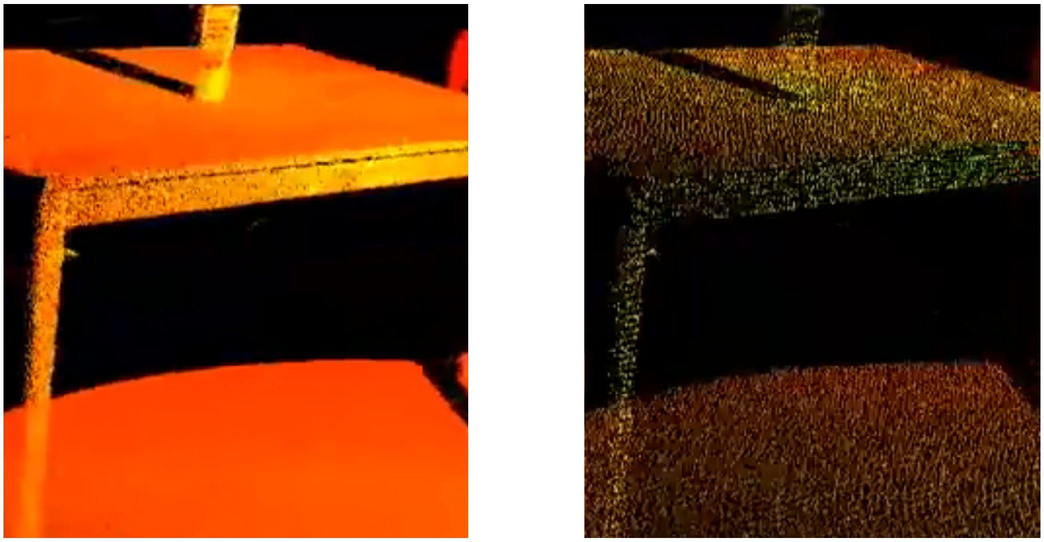


Fig. 3-15 Before and after downsampling of 3D table data (PCL, 2012)

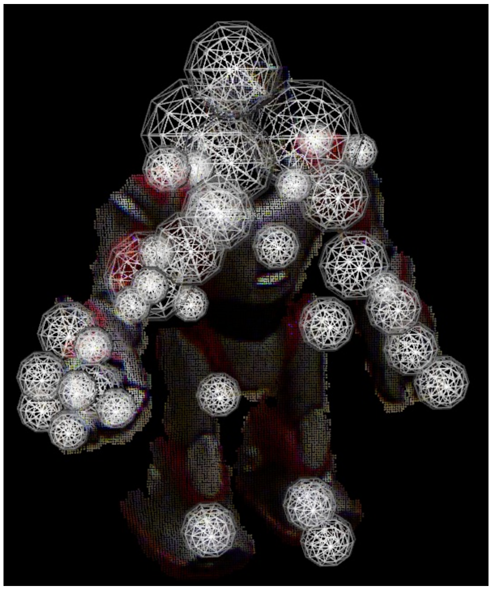


Fig. 3-16 3D SIFT keypoints (Michael, 2011)

3.3.4.2 3D SIFT Keypoints

As shown in Fig. 3-16, the 3D version of SIFT keypoint extraction is similar to that of 2D (Michael, 2011). To blur a point in a 3D point cloud, we just find all of its neighbors within a fixed-sized radius (based on the scale of the Gaussian) and assign the new intensity value as the Gaussian weighted sum of that of neighbors. Do this for all points at several blurring scales and subtract subsequent scales from each other, a 4-dimensional $D(x, y, z, \sigma)$ scale space is obtained.

3.3.4.3 Fast Point Feature Histograms

As we have extracted the keypoints from the point clouds, the next step is to find a suitable way to describe the feature of each point. As a result, corresponding points of 2 point clouds can be found by comparing their feature. Point Feature

Histograms (PFH, Rusu, 2008) are robust multi-dimensional features that describe the local geometry around a certain point in 3D point cloud datasets. In this section, we introduce the mathematical expressions of PFH, and their optimized version, called Fast Point Feature Histograms (FPFH, Rusu, 2009).

1. Point Feature Histograms

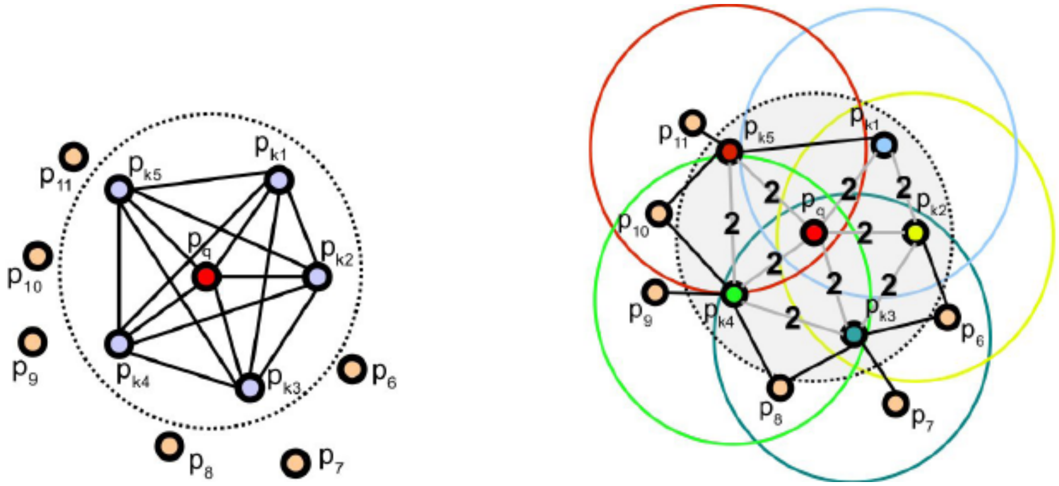
Point Feature Histograms (PFH) are informative pose-invariant local features that represent the underlying surface model properties at a point p . These features are scale and pose invariant. Their computation relies on the combination of certain geometrical relations between p 's nearest k neighbors. They incorporate 3D point coordinates $< x, y, z >$ and estimated surface normals $< nx, ny, nz >$, but are extensible to the use of other properties such as curvature, 2nd order moment invariants, etc.

For each neighboring point pair p_i and p_j ($i \neq j$, $j < i < k$) in the k -neighborhood of p and their estimated normals n_i and n_j (p_i being the point with a smaller angle between its associated normal and the line connecting the points), we define $u = n_i$; $v = (p_j - p_i) \times u$; $w = u \times v$ with the origin in p_i and compute 4 features f_x that measure the angle differences between the points' normals and the distance vectors between them as follows:

$$\left.\begin{aligned} f_0 &= \langle v, n_j \rangle \\ f_1 &= \langle u, p_j - p_i \rangle / \| p_j - p_i \| \\ f_2 &= \| p_j - p_i \| \\ f_3 &= a\tan\left(\langle w, n_j \rangle, \langle u, n_j \rangle \right) \end{aligned}\right\} i_{hist} = \sum_{x=0}^{x \le 3} \left\lfloor \frac{f_x \cdot d}{f_{x_{\max}} - f_{x_{\min}}} \right\rfloor \cdot d^x \tag{3-15}$$

where $x \in \{0,1,2,3\}$, "$\langle\ \rangle$" denotes the scalar product, "$\lfloor\ \rfloor$" denotes the floor function, and d is the number of subdivisions of the features' value range . For each point-pair and its i_{hist} index, we increment the histogram value at that index by 1, and at the end, normalize each bin with the total number of point pairs $k(k+1)/2$ to achieve point density invariance. The number of histogram bins that formed using these four geometric features is d^4. If we divide each feature definition range in 2 parts, we obtain a total of $2^4 = 16$ bins in total.

Fig.3-17(a) presents an influence region diagram of the PFH computation for a query point p_q, marked with red and placed in the middle of a circle (sphere in 3D) with radius r, and all its k neighbors (points with distances smaller than the radius r) are fully interconnected in a mesh.



(a) PFH (Rusu, 2008)

(b) FPFH (Rusu, 2009)

Fig.3-17 Influence region

The top left part of Fig.3-18 illustrates the PFH of a set of query points located on various geometric surfaces, which are synthetically generated. The results show that the different geometrical properties of each surface around the query point produce unique signatures in the feature histograms space. So PFH can be used to search corresponding for registering multiple clouds of the same model. The right part of Fig. 3-18 presents corresponding histogram features for similar points in two different overlapping point clouds (shown in the bottom left part of Fig. 3-18).

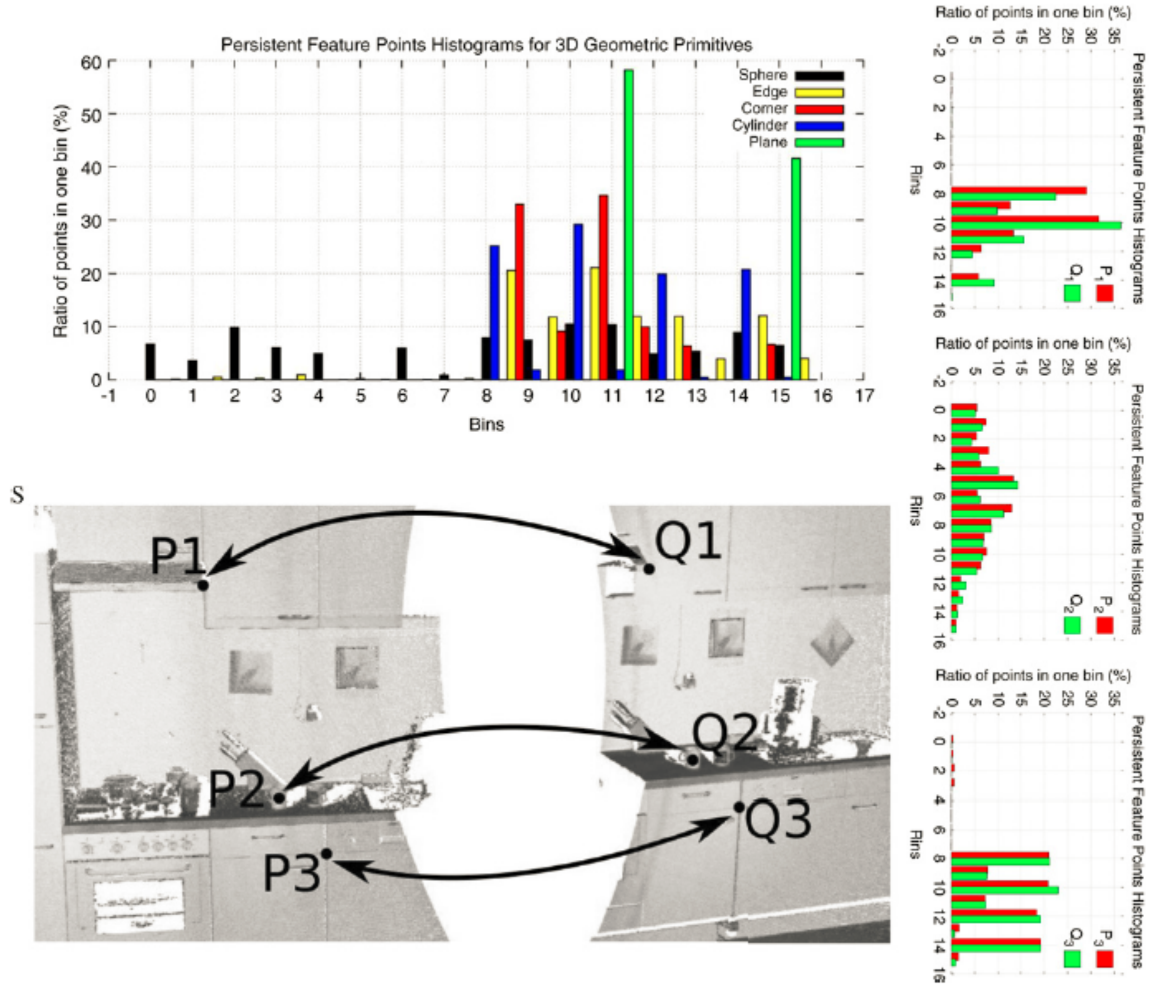


Fig.3-18 PFH of points on different surfaces (Rusu, 2008)

2. Fast Point Feature Histogram

The theoretical computational complexity of the Point Feature Histogram for a given point cloud P with n points is O(n·k^2), where k is the number of neighbors for each point p in P. Since the computation of Point Feature Histograms in dense point neighborhoods can represent one of the major bottlenecks in the registration framework for realtime or near realtime applications. A simplified version, Fast Point Feature Histograms (FPFH), is used instead of PFH. It reduces the computational complexity of the algorithm to O(nk), while still retaining most of the discriminative power of the PFH. The FPFH is calculated by:

i) For each query point p we compute only the relationships (see Eq. 3-15) between itself and its neighbors inside a r radius sphere – we will call this the Simplified Point Feature Histogram (SPFH).

ii) For each point we re-determine its k neighbors and use the neighboring SPFH values to weight the final histogram of p (called FPFH):

$$FPFH(p) = SPE(p) = \frac{1}{k}\sum_{i=1}^{k}\frac{1}{\omega_k}\cdot SPF(p_k) \qquad (3\text{-}16)$$

where the weight ω_k represents the distance between query point p and a neighbor point p_k.

An influence region diagram illustrating the FPFH computation is presented in Fig. 3-17(b). For a given query point p_q, we first estimate its SPFH values by creating pairs between itself and its neighbors. We repeat this for all the points in the dataset, and then we re-weight the SPFH values of p_k using the SPFH values of its neighbors, thus creating the FPFH for p_q. As the diagram shows, some of the value pairs will be counted twice (marked with 2 in the figure).

Recent experiments showed that presence of f_2 makes no significant difference, so we only use f_0, f_1 and f_3 to calculate the FPFH. A further optimization can be pursued if we tackle the correlation in the feature histogram space. So far, the resulting number of histogram bins was given by q^d, where q is the number of quantums (i.e. subdivision intervals in a feature's value range) and d is the number of features selected (in our case: $5^3 = 125$ bins). The resulting histograms contain a lot of zero values, and can thus contribute to a certain degree of information redundancy in the histogram space, as some of the subdivision cells of the cube will never contain any values. A simplification of the above is to simply create d separate feature histograms, one for each feature dimension, and concatenate them together (see Fig. 3-19, there are 15 bins in total).

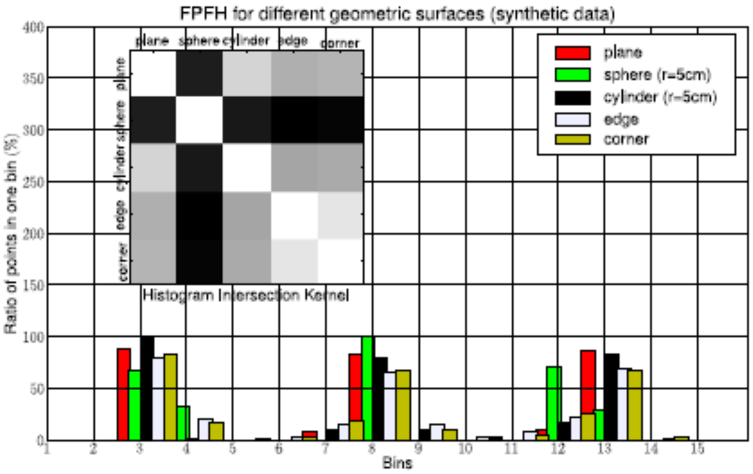


Fig. 3-19 FPFH of points on different surfaces (Rusu, 2009)

3. Persistent Analysis

In large datasets, the number of points with similar FPFH might be large and could lead to ambiguous correspondences. A solution is to neglect all points with features that are considerably dominant in the dataset and thus concentrate on more prominent points, which can be achieved by performing a persistence analysis (Rusu 2008): that is to observe which histograms are salient at each scale.

At a given scale, compute the distances from the mean FPFH of a dataset to all the features of that dataset. This distance distribution can be approximated with a Gaussian distribution, and using simple statistical heuristics, features whose distances are outside the $\mu \pm \beta \cdot \sigma$ interval can be selected as less common, where μ represents the mean FPFH, σ represents the standard deviation of the distance distribution, and β controls the width of the interval and acts as a band-stop filter cut-off parameter. To account for density variations but also different scales, the above is repeated over a discrete scaling interval (i.e. each point is enclosed in spheres with varying radii and its FPFH values recomputed), and points which are marked as unique over the entire interval are marked as persistent:

$$P_f = \bigcup_{i=1}^{n-1} [P_{f_i} \cap P_{f_{i+1}}] \tag{3-17}$$

where P_{f_i} represents the points which are selected as unique for radius r_i. The values of r_i are selected based on the size of the features that need to be detected.

3.3.5 3D Registration

3.3.5.1 Transformation Estimation

Let there be N corresponding points. Their coordinates in the source and target are denoted by $p = \{p_i\}$ and $q = \{q_i\}$ respectively, $i = 1,2,3 \ldots N$. We are looking for a transformation of the form

$$q_i = sRp_i + T + V_i \tag{3-18}$$

where s is the scale factor, R is a standard 3×3 matrix, T is a translation vector and V_i is a noise vector. Solving for the optimal transformation $[\hat{S}, \hat{R}, \hat{T}]$ that maps the set $\{p_i\}$ onto $\{q_i\}$ typically requires minimizing a least square error criterion:

$$e^2 = \sum_{i=1}^{N} \left\| q_i - \hat{s}\hat{R}p_i - \hat{T} \right\|^2 \tag{3-19}$$

Ideally, perfect matched corresponding point clouds should have the same centroid. So the translation T could be estimated from the offset between the original centroids, which are defined by

$$\begin{aligned} \overline{p} &= \frac{1}{N}\sum_{i=1}^{N} p_i \\ \overline{q} &= \frac{1}{N}\sum_{i=1}^{N} q_i \end{aligned} \tag{3-20}$$

The translation is found by

$$\hat{T} = \overline{q} - \overline{p} \tag{3-21}$$

If we move both centriods to the original, the new coordinates can be expressed by

$$\begin{aligned} p_{c_i} &= p_i - \overline{p} \\ q_{c_i} &= q_i - \overline{q} \end{aligned} \tag{3-22}$$

Then Eq.3-19 can be rewritten and reduce to

$$\begin{aligned} e^2 &= \sum_{i=1}^{N} \left\| q_{c_i} - \hat{s}\hat{R} p_{c_i} \right\|^2 \\ &= \sum_{i=1}^{N} \left\| p_{c_i} - (1/\hat{s})\hat{R}^T q_{c_i} \right\|^2 \end{aligned} \qquad (3\text{-}23)$$

or

$$\begin{aligned} e^2 &= \sum_{i=1}^{N} \left\| (1/\hat{s}) p_{c_i} - \hat{R} q_{c_i} \right\|^2 \\ &= \frac{1}{\hat{s}} \sum_{i=1}^{N} p_{c_i}{}^T p_{c_i} - 2\sum_{i=1}^{N} p_{c_i}{}^T \hat{R} q_{c_i} + \hat{s} \sum_{i=1}^{N} q_{c_i}{}^T q_{c_i} \\ &= \frac{1}{\hat{s}} S_p - 2D + \hat{s} S_q \\ &= \left(\sqrt{\hat{s}} \sqrt{S_q} - \frac{1}{\sqrt{\hat{s}}} \sqrt{S_p} \right)^2 + 2\left(\sqrt{S_q S_p} - D \right) \end{aligned} \qquad (3\text{-}24)$$

This is minimized with respect to the scale *s* when the first term is zero or when $\hat{s} = \sqrt{S_p / S_q}$ (Horn, 1988), that is

$$\hat{s} = \left(\sum_{i=1}^{N} \left\| p_{c_i} \right\|^2 \Big/ \sum_{i=1}^{N} \left\| q_{c_i} \right\|^2 \right)^{1/2} \qquad (3\text{-}25)$$

The above result could determine the scale without knowledge of the rotation. However, the estimation of the rotation is not affected by the choice of the value of the scale factor. The remaining error is minimized when D is as large as possible, which is equivalent to maximizing $Trace(\hat{R}H)$ (Eggert, 2004), where

$$H = \sum_{i=1}^{N} p_{c_i} q_{c_i}{}^T \qquad (3\text{-}26)$$

If the singular value decomposition of H is given by $H = U \Lambda V^T$, then the optimal rotation matrix that maximizes the desired trace is

$$\hat{R} = VU^T \tag{3-27}$$

Eq.3-27 is called the orthogonal Procrustes problem, which can be solved using SVD method (Schonemann, 1966).

3.3.5.2 SAmple Consensus (SAC)

The registration of a pair of 3D point clouds is easily solvable if the point to point correspondences are perfectly known. We implemented the SAC method which maintains the geometric relations of the correspondences without having to try all combinations. The large numbers of correspondence candidates are sampled and ranked quickly by employing the following scheme:

i) Select s sample points from P while making sure that their pairwise distances are greater than a user-defined minimum distance *dmin*.

ii) For each of the sample points, find a list of points in Q whose histograms are similar to the sample points' histogram. From these, select one randomly which will be considered that sample point's correspondence.

iii) Compute the rigid transformation defined by the sample points and their correspondences and compute an error metric, determined using a Huber penalty measure L_h, for the point cloud that computes the quality of the transformation.

$$L_h(e_i) = \begin{cases} \frac{1}{2} e_i^2 & \|e_i\| \leq t_e \\ \frac{1}{2} t_e (2\,\|e_i\| - t_e) & \|e_i\| > t_e \end{cases} \tag{3-28}$$

This scheme finds a good transformation fast by looking at a very large number of different correspondences. These three steps are repeated, and the transformation that yielded the best error metric is stored and used to roughly align the clouds. Finally, a non-linear local optimization is applied using a Levenberg-Marquardt algorithm. Since SAC only considers the sample points, the initial alignment result will not be quite accurate. A final registration is needed to optimize the result.

3.3.5.3 Iterative Closest Point

The final registration is achieved using the Iterative Closest Point (ICP, Zhang, 1994) registration, which is an efficient and reliable method for registration of free-form curves and surface. This algorithm is based on the prior knowledge of the initial alignment. It iteratively matches points in one set to the closest points in the other and estimate the final registration with high accuracy.

The inputs of ICP are two frames containing m and n 3D points. The output is the optimal motion between two frames. It aims to minimize the criterion

$$\mathcal{F}(R, t) = \frac{1}{N} \sum_{i=1}^{N} \|Rx_i + t - y_i\|^2 \tag{3-29}$$

where N is the number of pairs, R and t is the motion rotation and translation,

x_i and y_i are paring points. The procedure is:

i) Initialization: Set a value for the maximum tolerable distance in the first iteration D_{max}^{0} . Every point in the first frame whose distance to its closest point in the second frame is bigger than D_{max}^{0} is discarded during the first iteration.

ii) Preprocessing: a) Compute the tangent at each point of the two frames; b) Build the k-d tree representation of the second frame.

iii) Iteration: a) Find the closest points satisfying the distance and orientation constrains; b) Update the recovered matches through statistical analysis of distance; c) Compute the motion between the two frames from the updated matches; d) Apply the motion to all points in the first frame; e) Iteration until convergence of the computed motion.

3.3.5.4 Point Cloud Library

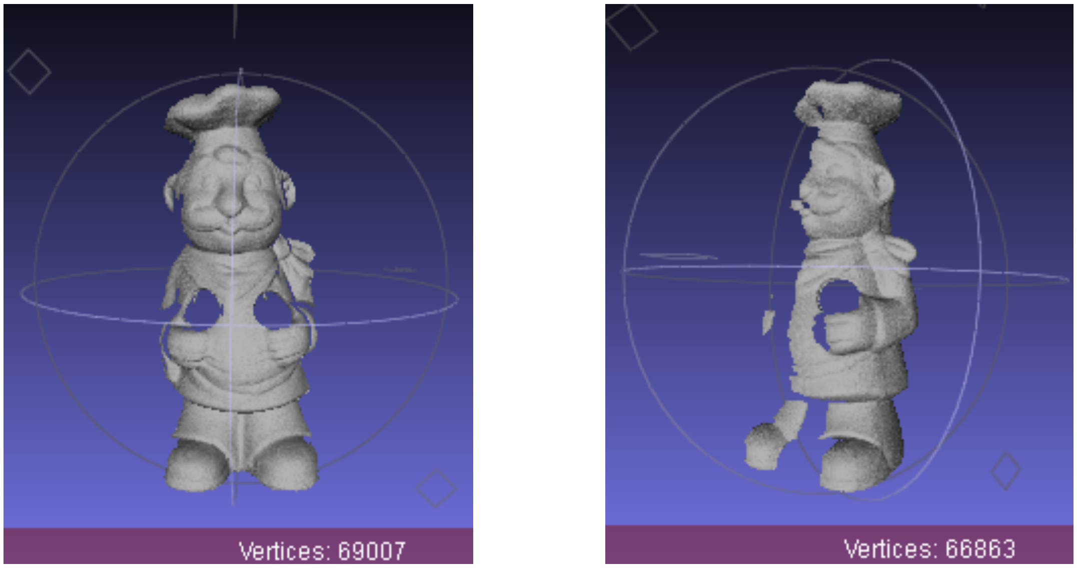
The Point Cloud Library (PCL, http://www.pointclouds.org) is a large scale, open source software for 2D/3D image and point cloud processing. It covers numerous state-of–the-art algorithms including filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation. In this thesis, PCL is used as an external library for C++ to manipulate the 3D point clouds.

Chapter 4

Results and Discussion

4.1 Testing Data

To test the registration method proposed in the thesis, a 3D Chef data (point clouds) set is used. Two point clouds from this data set are shown in Fig.4-1. This data set is scanned with the Minolta scanner and can be downloaded from http://www.csse.uwa.edu.au/~ajmal/3Dmodeling.html.



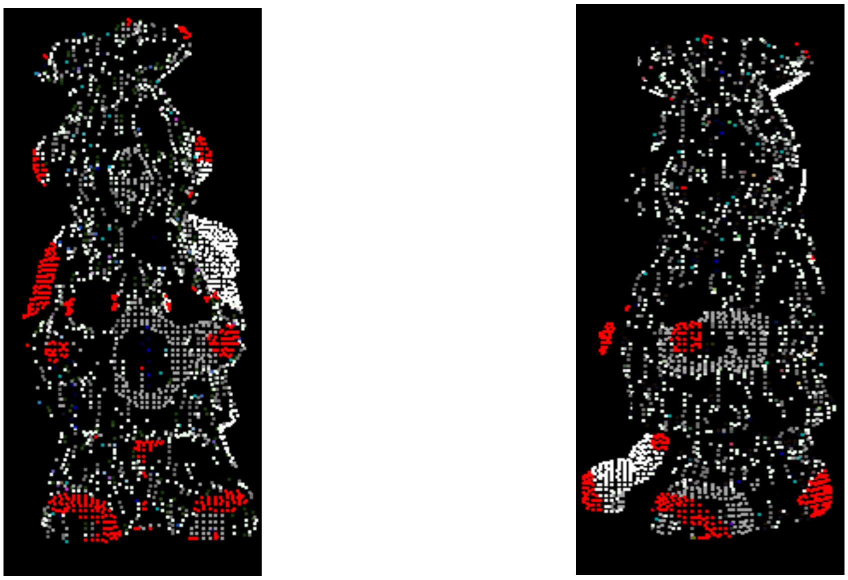
(a) Cloud 1

(b) Cloud 2

Fig. 4-1 3D Chef data

Compared to the Pictometry 3D data generated by the 3D workflow we used, the Chef data consist of much denser 3D points, but less noise. The points in the data locate accurately on the outline of the statue. The surface itself contents much more complicated structures. Since the noise in the Chef data set can be ignored and the point refinement step is skipped. We simply down sample the point clouds by leaf size = 3.0 (i.e. the average distance d_0 in the result point cloud) before we extract the FPFH for each point.

4.1.1 No Scale Difference



(a) Cloud 1

(b) Cloud 2

Fig. 4-2 Point with persistent features (shown in red)

Fig. 4-2 shows the points before (shown in white, 5213 points and 4875 points respectively) and after (shown in red, 424 points and 303 points respectively) the multi-scale persistence analysis. These remained red points have unique features. The corresponding points used later for the initial alignment will be selected from these candidates.

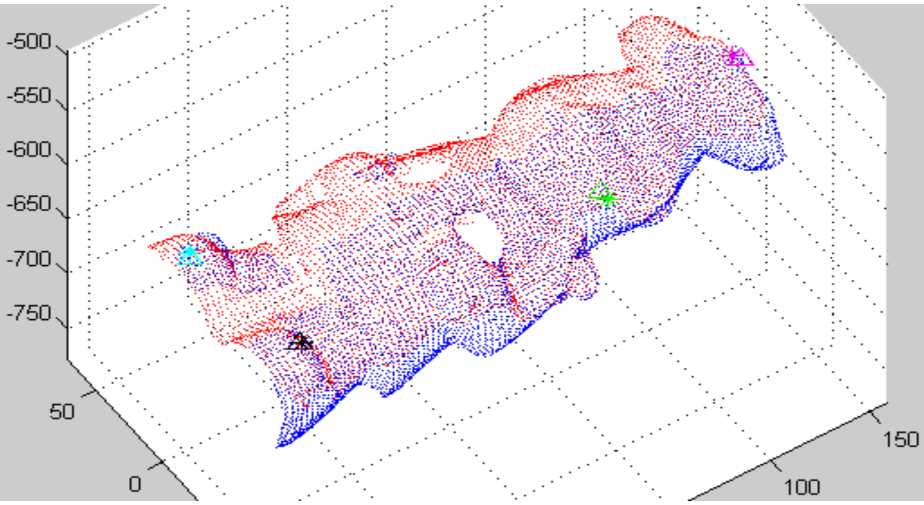


Fig. 4-3 Initial alignment result

(red: points in Cloud 1, blue: points in Cloud 2, black/green/cyan/magenta: Corresponding points (*: points in Cloud 1, ^: points in Cloud 2))

The initial alignment result using the sample consensus method is shown in Fig. 4-3. Four corresponding pairs we found are highlighted in four different colors respectively. The two points in each corresponding pair are marked in the same color but different shape according to which cloud it belongs to. We can see that these corresponding points overlap well. Based on these corresponding points, a satisfying initial alignment is achieved.

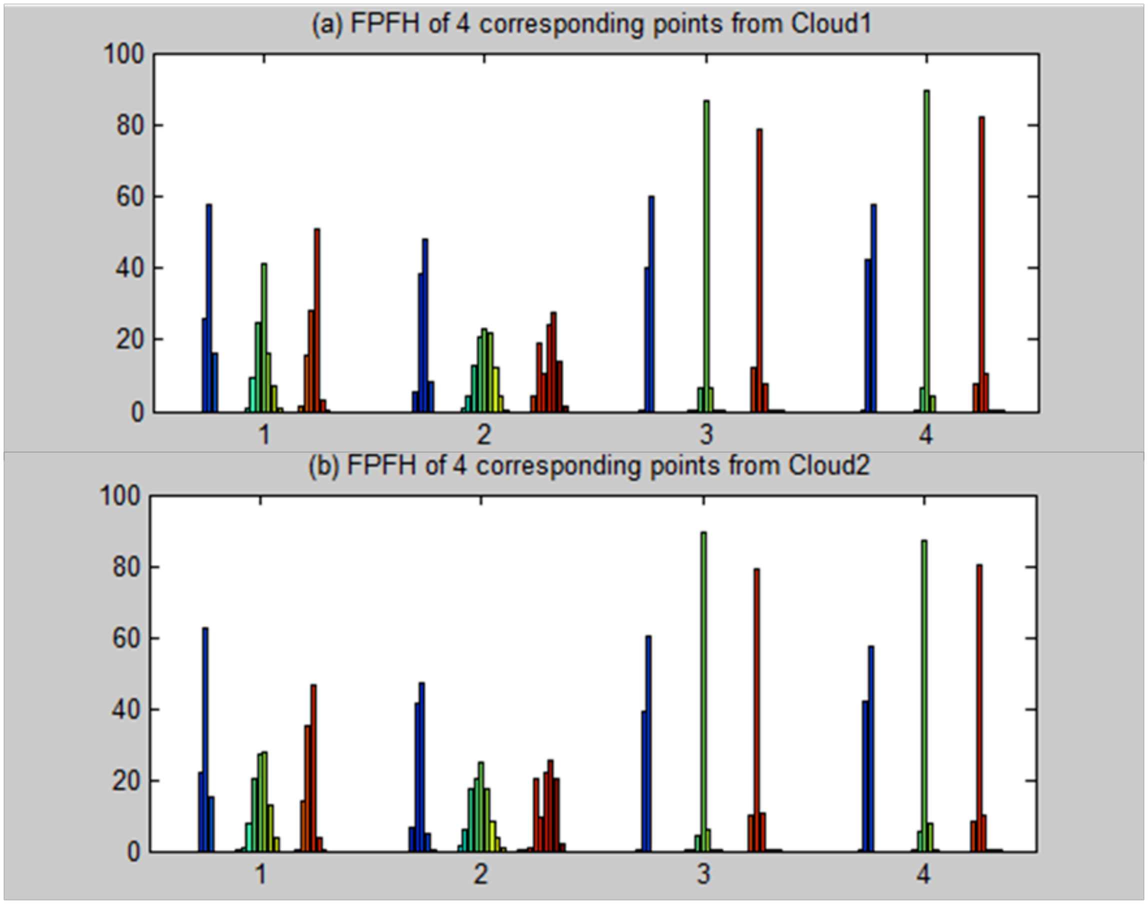
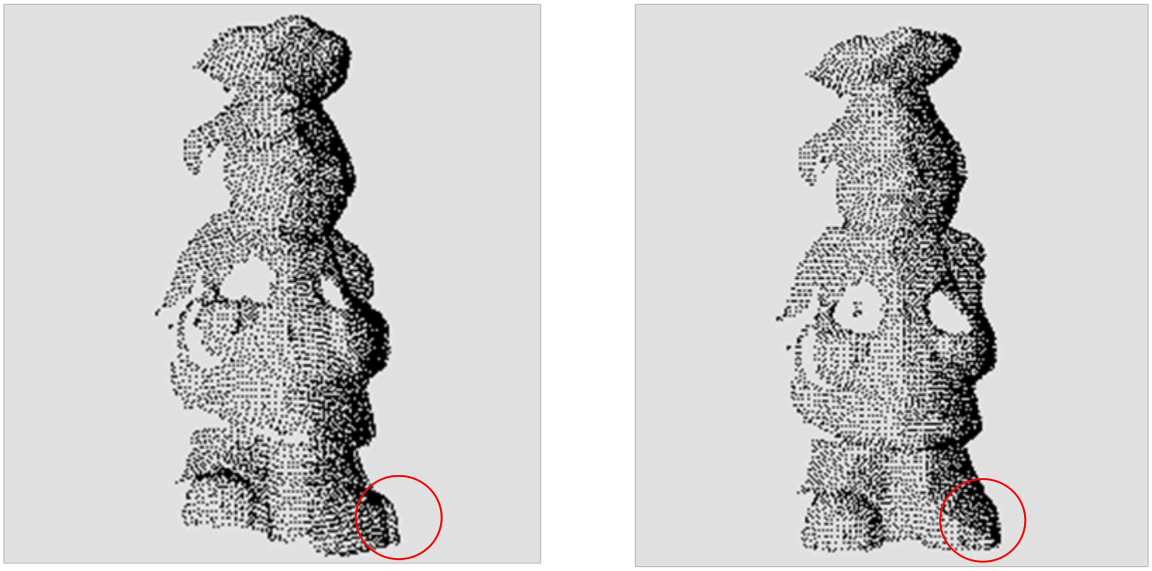


Fig. 4-4 FPFH of corresponding points

(blue: f_0, green: f_1, red: f_3)

The FPFH of the corresponding points are shown in Fig. 4-4. For each corresponding pair (shown in the same column), their FPFH are quite similar to each other, which proves that the corresponding points are selected reasonably and correctly as expected.

The final ICP registration result is shown in Fig. 4-5 (b). Compared with the initial alignment result shown in Fig. 4-5 (a), the offset around the shoe in the right (circled in red) is revised after the final registration.

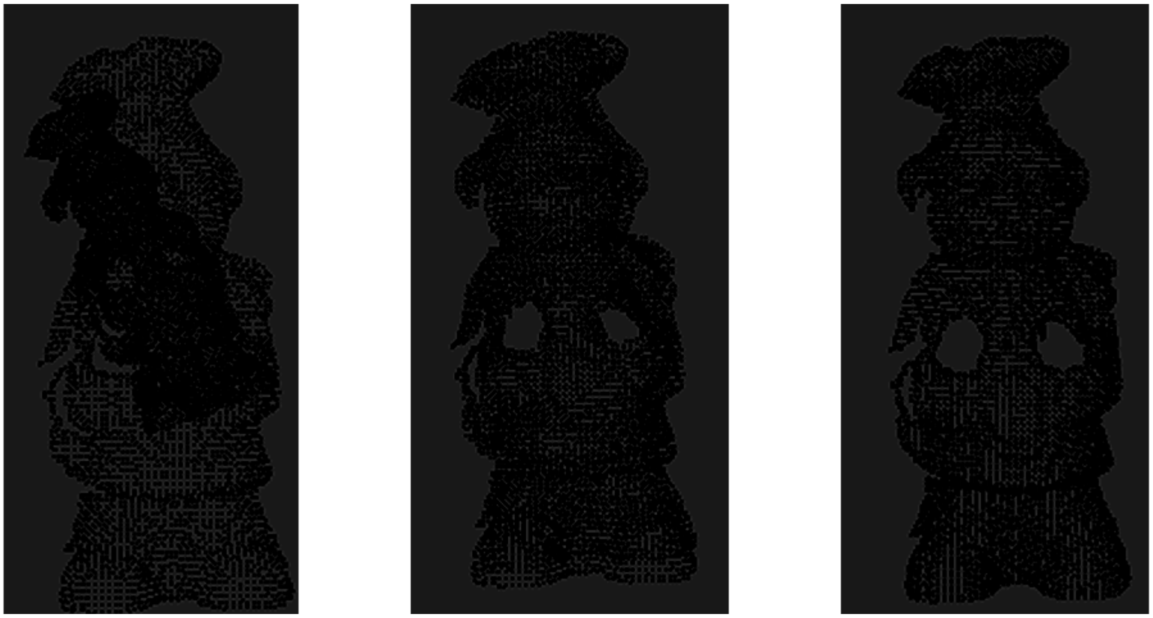


(a) Initial alignment (b) Final registration

Fig. 4-5 Comparison of initial alignment and final registration results

4.1.2 With Scale Difference

As shown in Fig. 4-6, we test the algorithms under different scale differences between input target clouds. When we use one point cloud and its rescaled point cloud for registration, the algorithm can handle different scale ranging from 0.5~2 (scale of input scale is 1). But when different point clouds are used for registration, this range is very narrow (0.9~1.1). To obtain a satisfying result, we should better set the point clouds to similar scale first. In reality, most point clouds have preliminary information about its scale somehow, so this won't be a big problem.



(a) Original Clouds (b) Initial alignment (c) Final registration

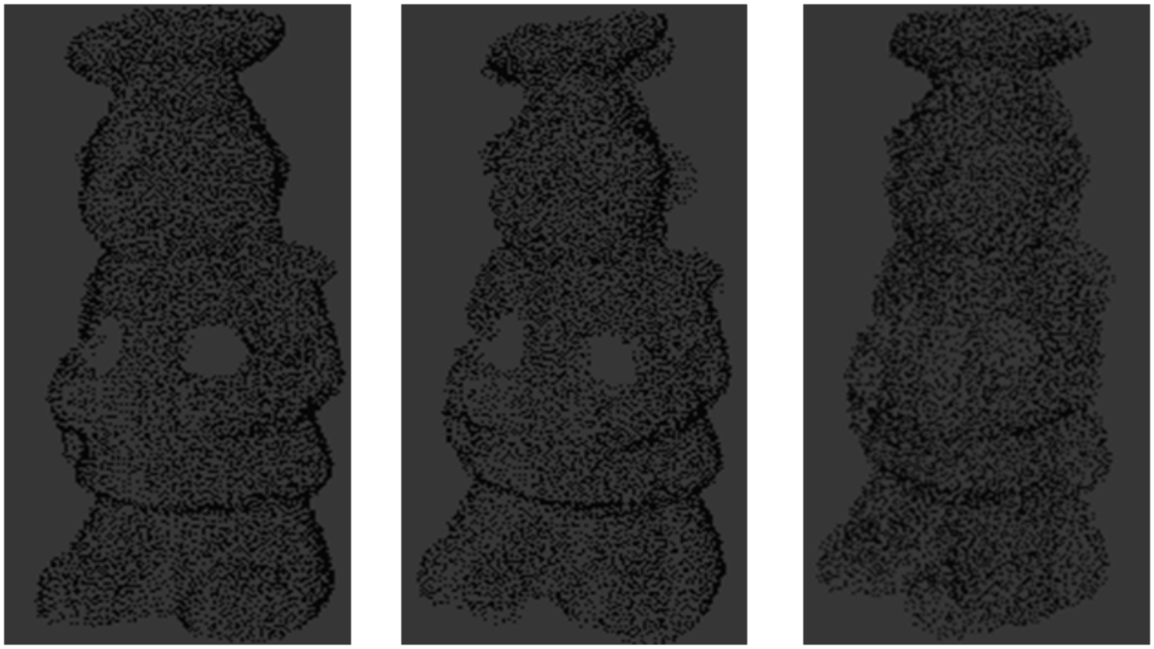
Fig. 4-6 Registration result for clouds with different scale (scale = 1.6)

4.1.3 With Noise

To test the robustness of the method introduced, artificial noise is added to the chef point clouds before normal and feature estimation procedure.

When sparse noise points (i.e. random points whose coordinate are uniformly distributed in the 3d space) exist, they can be easily removed as long as they are much less dense than the point cloud itself (i.e. $d_n < 0.1d_0$). d_n is the average distance between the noise points, while d_0 is the average distance between the original points.

When wrong clusters (i.e. small noise clusters that has a similar density as the original point cloud) exist, they can be removed correctly if their size is small enough (i.e. $N_n < 100N_0$ and $D_n < 3D_{ist}$). N_n is the number of points in a noise cluster, $N_0 \; is$ the number of points in the original cloud, D_n is the diameter of the noise cluster, D_{ist} is the smallest distance from a point in the noise cluster to a point in the original cloud.



(a) β= 0.3, success (b) β= 0.35, small offset (c) β=0.5, fail

Fig. 4-7 Registration result with shifting noise presence

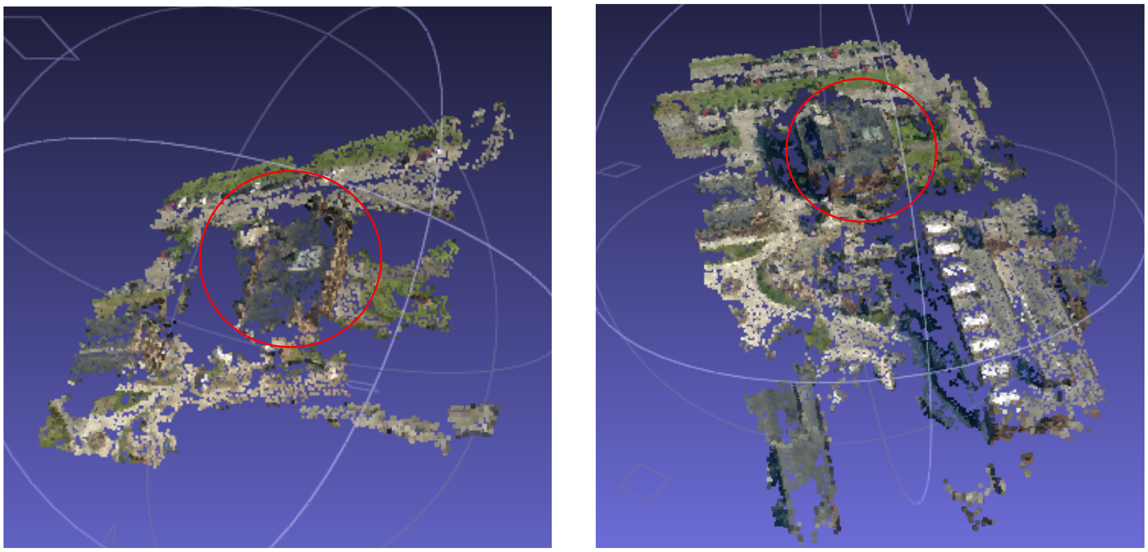
The noise that influences the final registration result significantly is the shift of points originating from the error which always exists when we estimate their 3D positions. The x, y, z coordinates of each points are shifted by three random values in range $[-\beta d_0, \beta d_0]$, where β is a control factor. With the influence of severe shifting noise (i.e. $\beta \geq 0.5$), the details of the original surface will be lost in the smoothing procedure, which trades off the benefit from MLS. The registration result is shown in Fig. 4-7. We can see that a successful registration can be achieved if $\beta \leq 0.3$.

4.2 Pictometry Data

4.2.1 Image Grouping

The research in this thesis is based on the application of the RIT 3D Extraction Workflow, which is designed to reconstruct dense points from nadir images. The imagery used in this project is provided by Pictometry, including airborne images taken from five different viewing directions. If images of different viewing directions are simply thrown into the original workflow, the result will be poorer than that only using a portion of these images all in the same viewing direction.

The following results in Fig. 4-8 are extracted from sub-images of the original Pictometry images. Each image is a 800×600 pixels area around the RIT Building 76 (highlighted by red circles in the point clouds).



(a) Using images of all directions

(b) Using only north images

Fig. 4-8 The reconstruction results of using different image groups

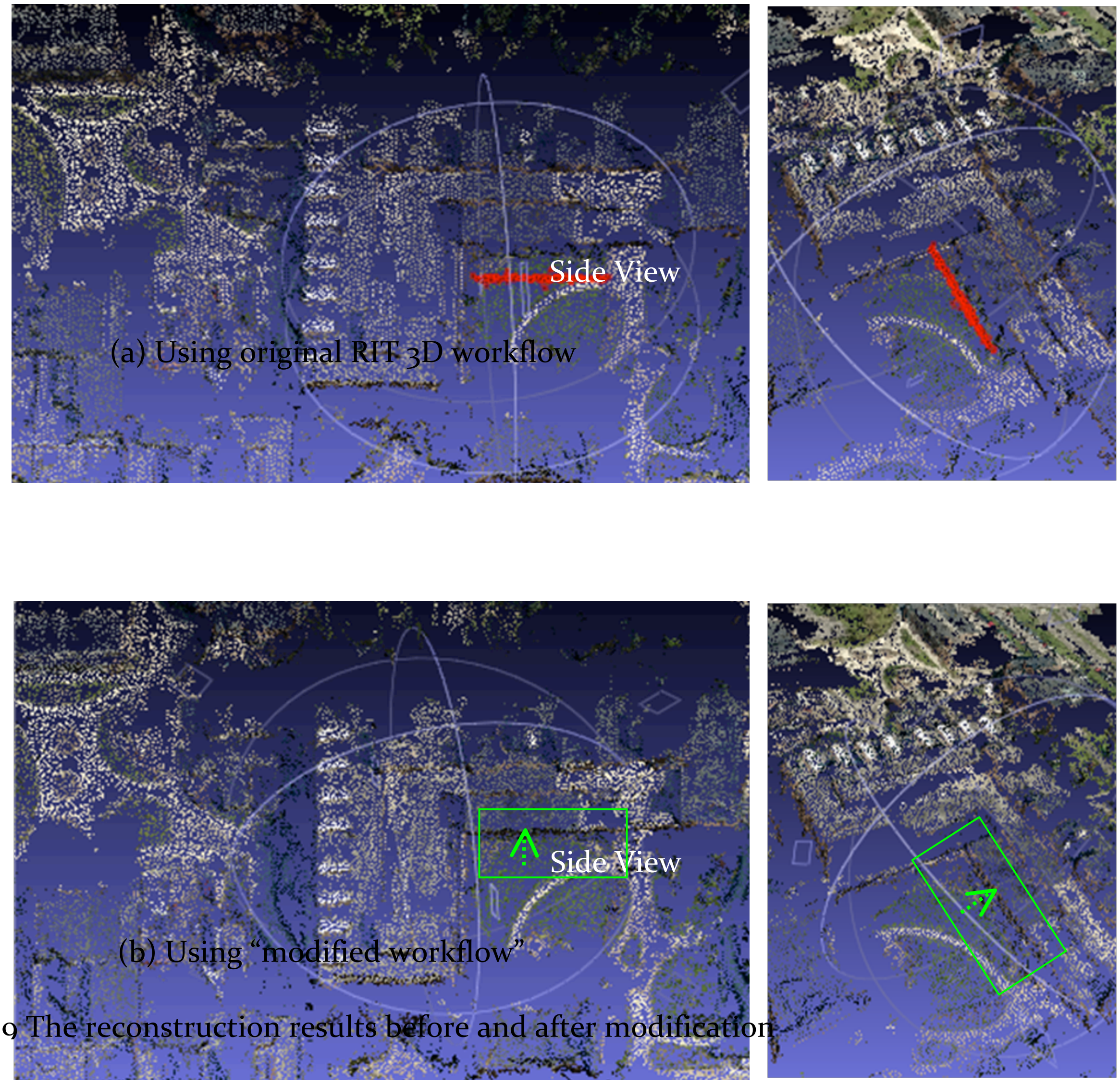
The reconstruction result on the left (Fig.4-8(a)) includes 9882 points, using 28 oblique images of different directions (6 east images, 11 north images, 4 south images, and 7 west images). When we checked the “bundler.out” file, we found that lots of images are abandoned because of bad 2D matching results before the PMVS process. Since they didn’t participate in the final dense point reconsruction, the 3D information they carried is missing, leading to large blank areas in the resulting point cloud.

The reconstruction result on the right (Fig.4-8(b)) included 23901 points, using only 11 north images. Because every image is taken from the same direction, they are easier to be matched. From the corresponding “bundler.out” file, most images are used for the dense point reconstruction, except one image that covers a small overlap area compared to the others. As a result, the set of north images is utilized more efficiently and the output point cloud is even better (includes much more points) than the former.

From the above comparison, we can conclude that more comprehensive results could be obtained by grouping the images according to the viewing direction before the 3D extraction procedure. Actually, the name of each Pictometry image file already includes the information about the viewing direction. It is easy to sort these images without determining from their content.

4.2.2 Modification of the RIT 3D Workflow

Since the ASIFT algorithm can produce better matching result than the SIFT algorithm when affine transformation exists, the SIFT part in the RIT 3D workflow is substituted by the ASIFT. This process is realized by using the ASIFT keypoints instead of the SIFT keypoints. That is to save the 128-digit descriptors of ASIFT keypoints in the "SIFT style" (which means to change the original orders of these digits to be consistent with SIFT descriptors) and use them instead of the original ".key" files generated from SIFT. In addition, since the focal lengths of the cameras are known, the focal length estimation procedure in bundler of the RIT 3D Workflow is also skipped. The comparison of the results before and after these modifications is shown in Fig. 4-9.



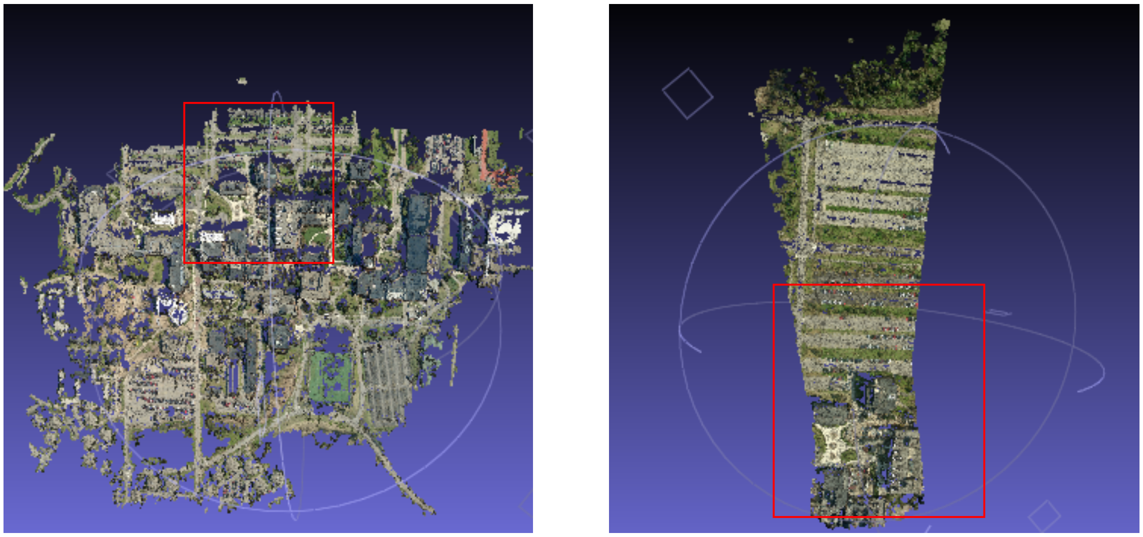
(a) Using original RIT 3D workflow

(b) Using "modified workflow"

Fig. 4-9 The reconstruction results before and after modification

The point cloud shown in Fig. 4-9(a) are extracted from 11 full size north images using the original RIT 3D Workflow, which includes 366422 points in total. The highlighted red points present the wall of a nearby building but shift to a wrong location. Fig. 4-9(b) shows the point cloud extracted from the same image set but using the “modified workflow”, which includes 435908 points in total. In the highlighted area, the floating wall (i.e. red points in Fig. 4-9(a)) disappears. These points which belong to a nearby wall have moved to the correct position (highlighted by green arrow). The whole point cloud is denser than the former one because of the better matching of images using ASIFT. And the position of the shifting cluster in the former point cloud is estimated correctly because the error has been eliminated by fixing the true focal length.

4.2.3 Original Point Clouds extracted

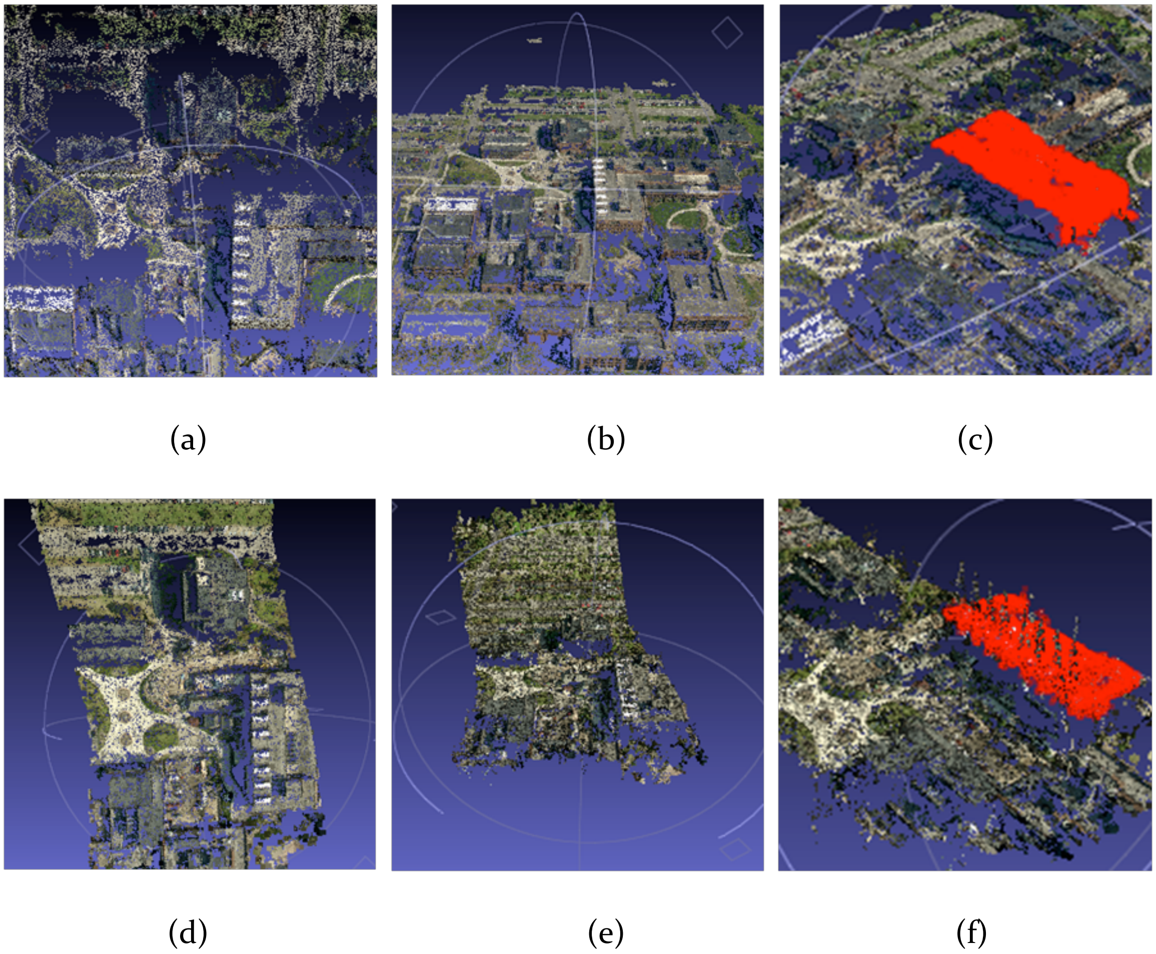


(a) From north image set

(b) From nadir image set

Fig. 4-10 Point clouds extracted from the “modified workflow”

Using the “modified workflow”, two point clouds extracted from 11 north and 9 nadir images respectively are shown in Fig. 4-10. The areas highlighted in red squares show the geographical overlap of two point cloud. Since two image sets cover different geographical areas, the resulting point clouds represent scenes of different areas also.



(a) (b) (c)

(d) (e) (f)

Fig. 4-11 Different views of the point clouds

(a)(b)(c): North cloud; (d)(e)(f): Nadir cloud

(a)(d): Nadir view; (b)(e): North view; (c)(f): Side view

Red: Points on the facets of a same building

Fig. 4-11 shows the different views of the point clouds extracted. From these figures, we can see that the north cloud looks better in the north view, while the nadir cloud looks better in the nadir view. This is because the north image set includes more information about the north facets of buildings (e.g. the highlighted points in 4-11(c)); as a result the extracted cloud includes more points presenting the north facets. Similarly, the nadir image set includes more information that could be found when viewing from top, so the extracted cloud includes more points describing the roofs.

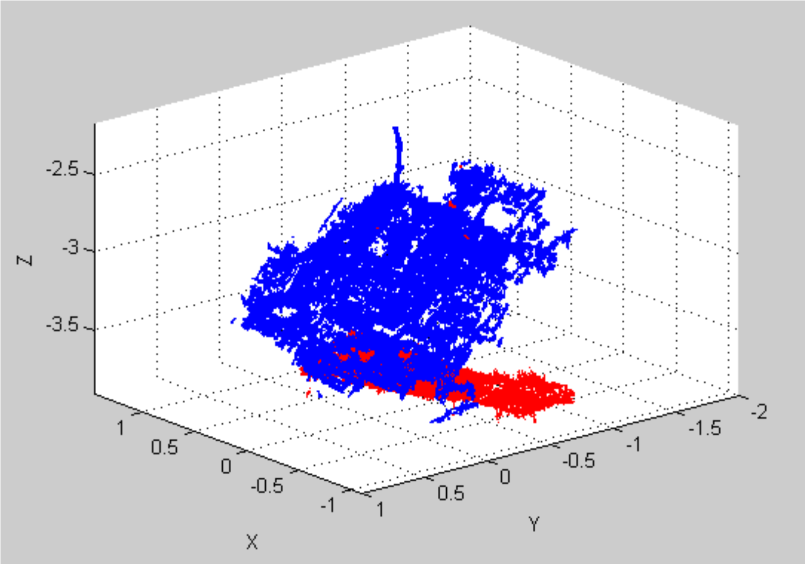
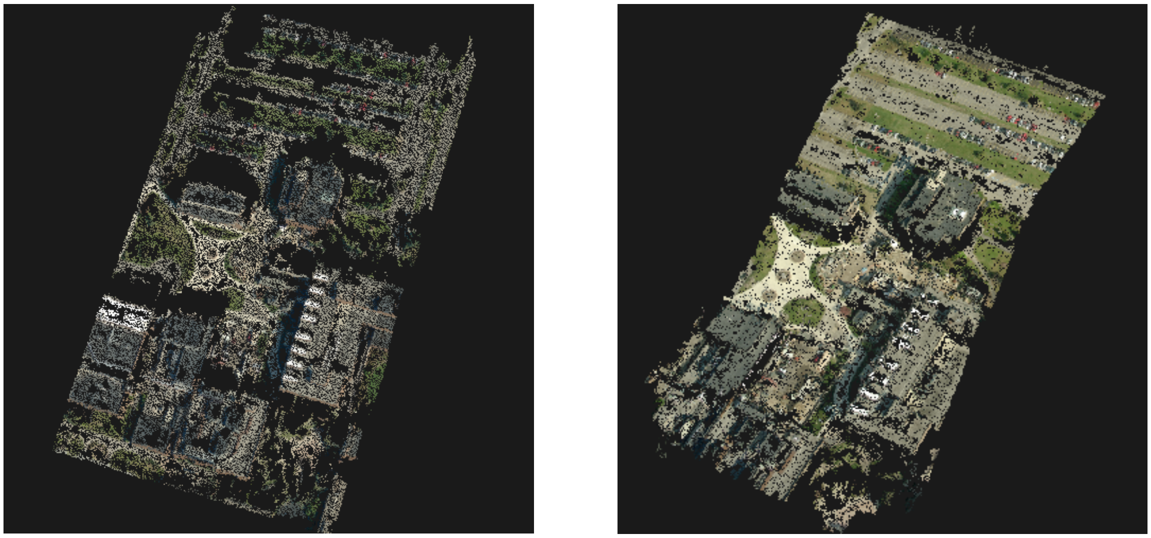


Fig. 4-12 Point clouds shown in the same coordinate system

(blue: north cloud, red: nadir cloud)

Since different point clouds focus more on different facets, the combination of these point clouds will provide richer information and render the scene better. But these point clouds extracted from the 3D workflow use independent coordinate systems. As shown in Fig. 4-12, there is translation, rotation and scale difference between the point clouds. To perform a 3D registration, we need to find out the corresponding points between two clouds and to estimate the transform matrix.

4.2.4 Refined Point Clouds



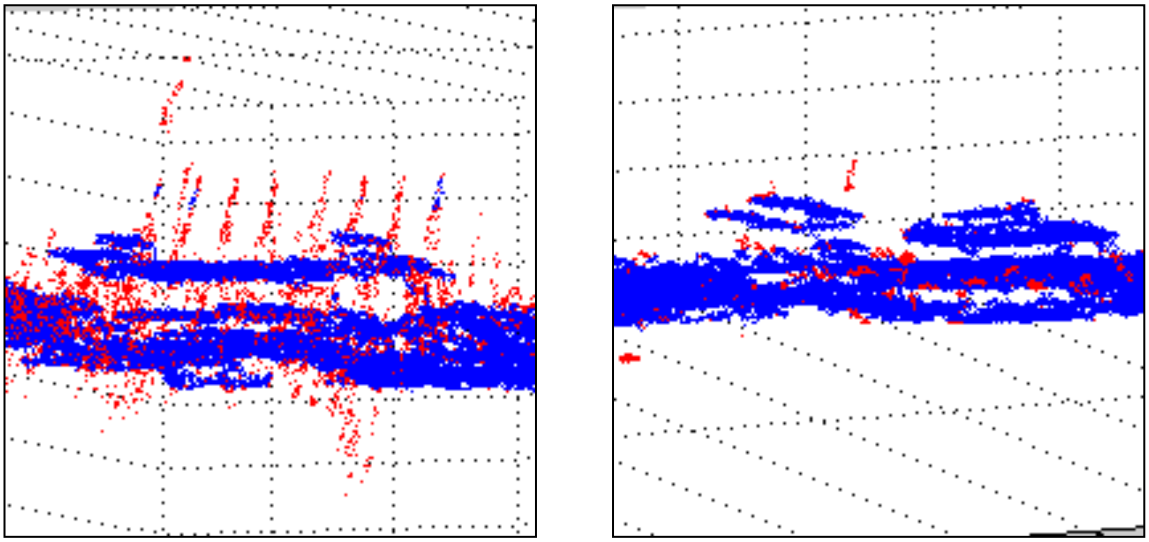
(a) North cloud

(b) Nadir cloud

Fig. 4-13 Sub-clouds used for registration

To simplify the calculation, the point clouds are cut into smaller size to only contain the areas that have overlap for the later registration process. The two point clouds which will be used for registration are shown in Fig. 4-13. The original point clouds here are quite noisy, especially for the nadir point cloud. As shown in Fig. 4-13, some sparse outliers distributing all over the space, some are mis-estimated floating point clusters, and points presenting the same surface are not exactly on the “true surface”. With all these errors, it’s hard to extract accurate 3D features. So the noise removal and surface smoothing are necessary.

The filtering results are shown in Fig.4-14. The Statistical Outlier Removal method decides that a point is an inlier or outlier according to the point density of its neighborhood, while the Radius Outlier Removal method decides according to the size of the cluster it belonging to. The final remaining points (about 80% of the original points) represent the real scene better without too much noise.

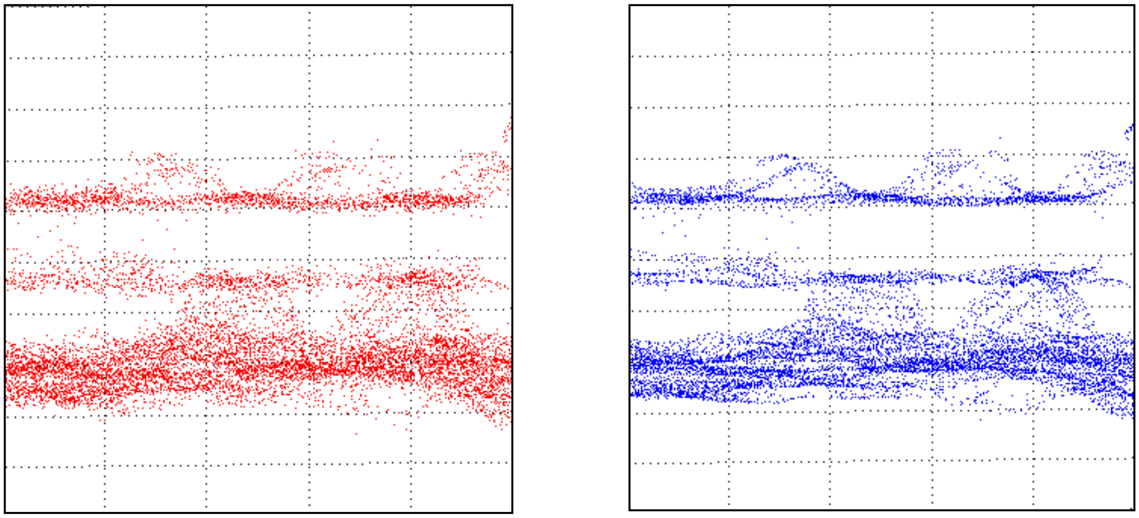


(a) Floating point removal

(b) Wrong clusters removal

Fig. 4-14 Noise removal results (red: outliers, blue: remained points)

The surface smoothing results are shown in Fig.4-15. The re-distributed points are concentrated around actual surfaces to better avoid wiggling borders.



(a) Original points

(b) Re-distributed points

Fig. 4-15 Surface smoothing result

An overall view of the nadir point cloud before and after the refinement process is shown in Fig. 4-16. We can see that most of the floating points and wrong clusters are eliminated and the surface now is much smoother than before.

A comparison of the normal distribution before and after the refinement process is shown in Fig. 4-17. In Fig. 4-17(a), the normals derived from the 3D workflow look like a bunch of random vectors, while the updated normal estimation shown in Fig. 4-17(b) are much more reasonable. According to this result, we can see that the refining procedure obtained the expecting effects. The normals of the points estimated are closer to the true surface normals after the refinement now.

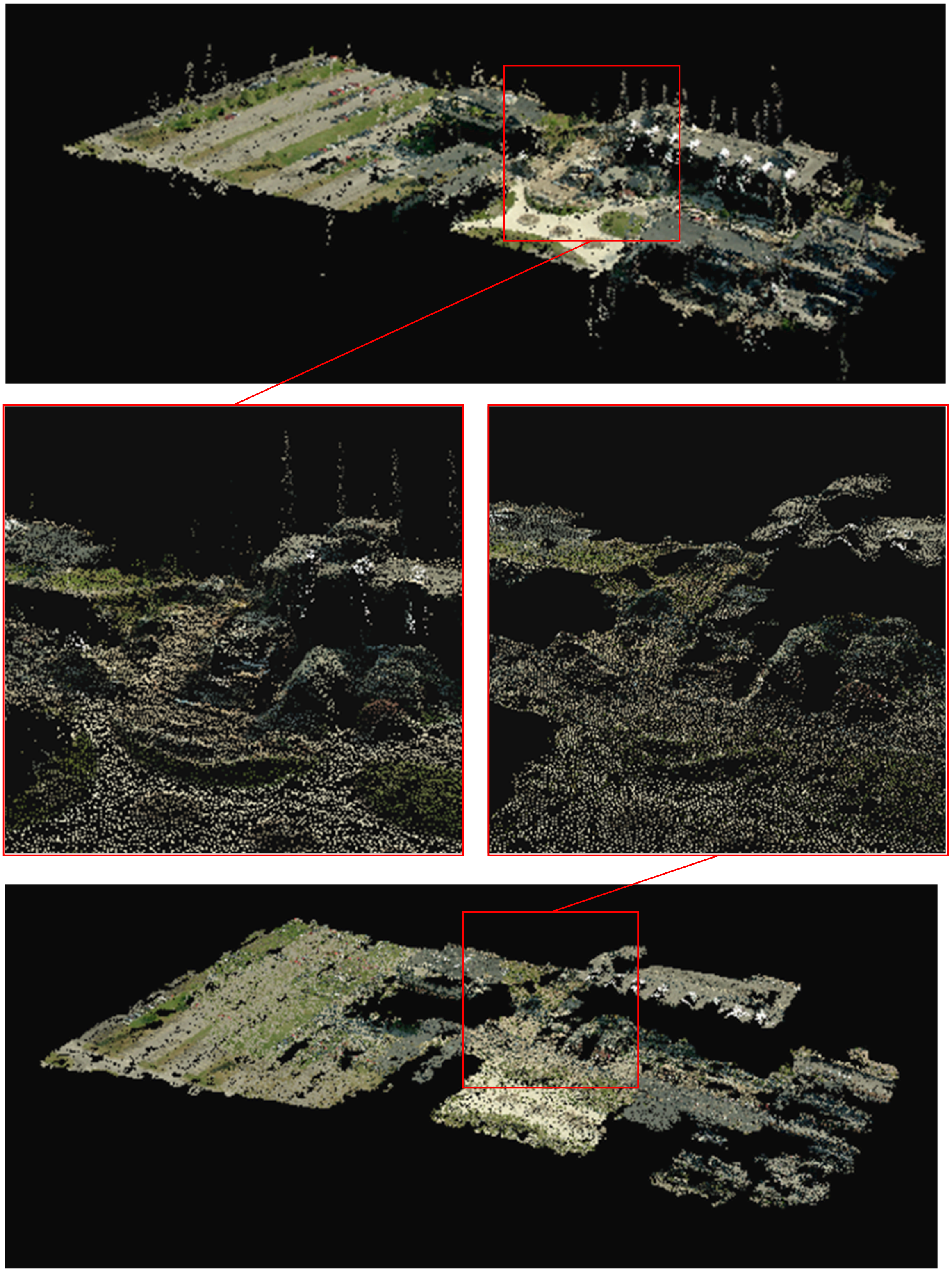
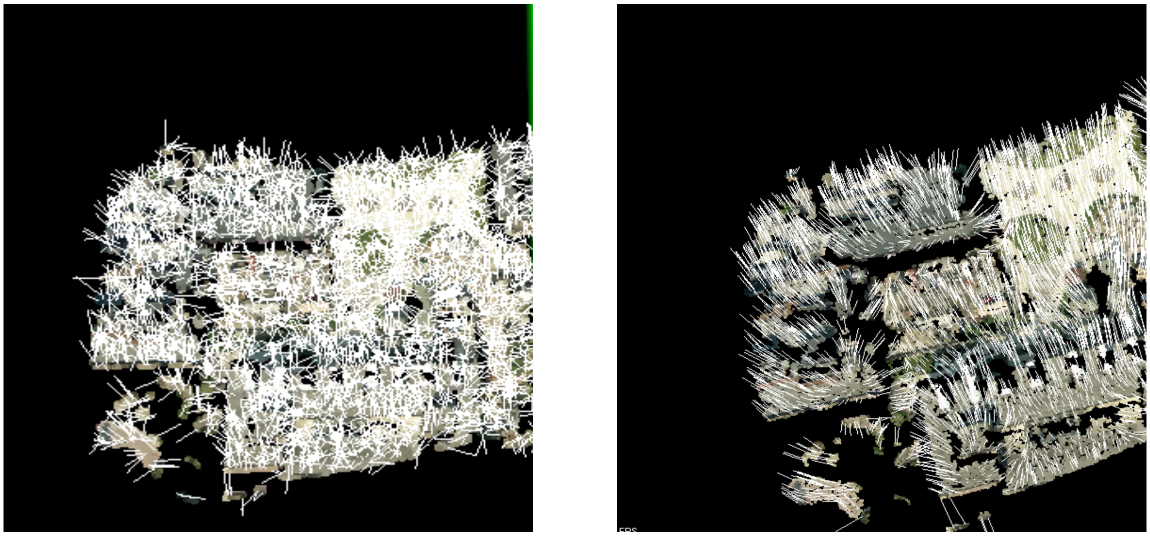


Fig. 4-16Nadir point cloud before and after refinement

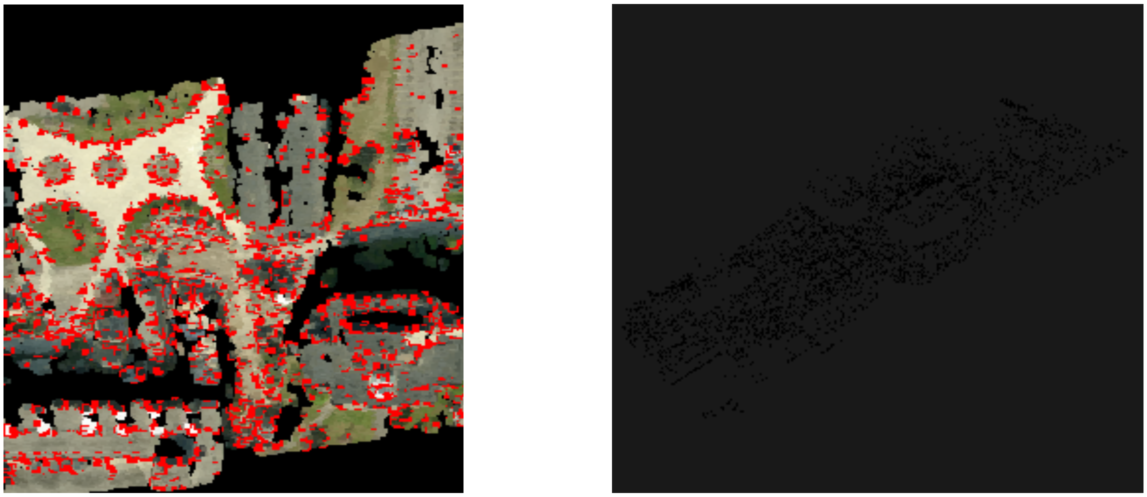


(a) Before refinement

(b) After Refinement

Fig. 4-17 Normals of the points

4.2.5 3 D Keypoints



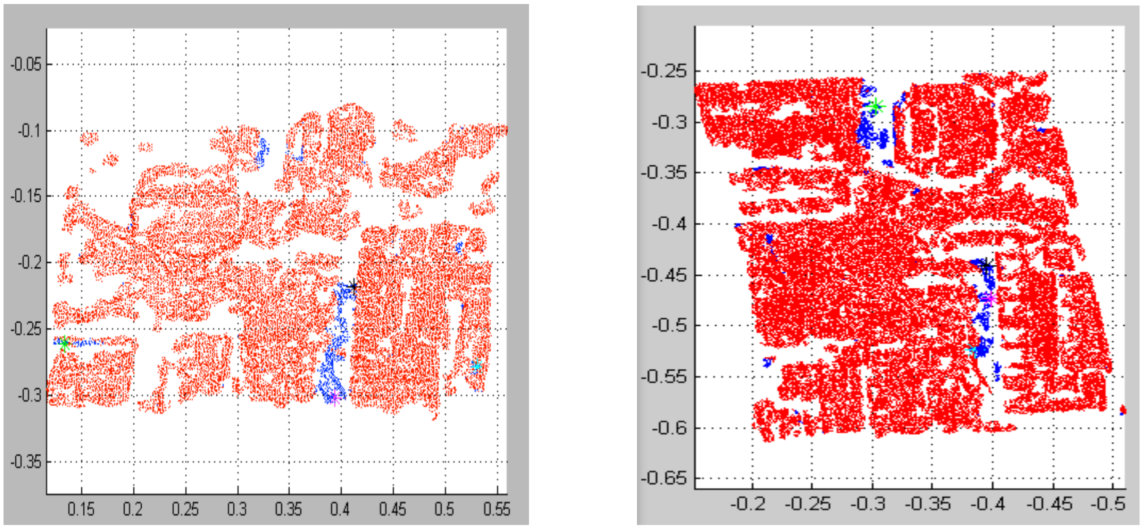
(a) Highlighted keypoints

(b) 3606 keypoints for the nadir cloud

Fig. 4-18 3D SIFT keypoints selected

According to the method mentioned in 3.3.4.2, 2% points of the entire clouds are selected as keypoints according to the color distribution (See Fig. 4-18). Actually, this keypoint extraction step is not necessary for our Pictometry data, which is not as dense as scanned data. The number of points is already shrunk to tens of thousands after the downsampling process. So we skipped this step here (But for a larger data set, this step will be necessary.).

4.2.6 Point with Unique FPFH



(a) North cloud

(b) Nadir cloud

Fig. 4-19 Points with unique FPFH (shown in blue)

Follow the algorithm introduced in 3.3.4.3, the FPFH features are calculated for the entire point clouds under different scales, and a multi scale persistence analysis process is performed to focus on those points with unique features. Fig. 4-19 shows the points remained in each point cloud after persistence analysis. There are about one thousand unique points found for each cloud.

4.2.7 Initial Alignment

Using the method mentioned in 3.3.5.1 and 3.3.5.2, we randomly select several samples and their nearest corresponding points to guess the transformation. Choose the best one that produce smallest errors as the rough initial alignment result. Here we re-scale the target cloud first (scale $\approx$ 1.5) to make sure it is comparable to the input cloud. The selected corresponding point pairs are shown in Fig. 4-20.

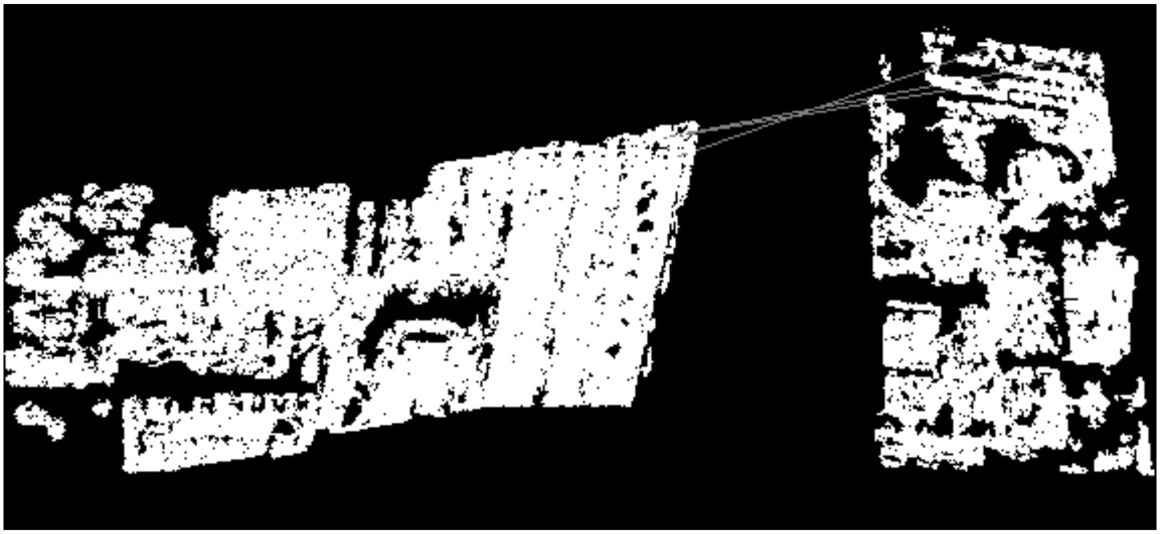


Fig. 4-20 Corresponding points selected

The FPFH of the corresponding points are shown in Fig. 4-21. We can see that the FPFH of pair 1 match each other the best (i.e. a good match is found), while the FPFH of pair 3 have the largest difference (i.e. this pair might not be a good match). But it does not influence the success of initial alignment result since we are not expecting a high accurate result in this step anyway.

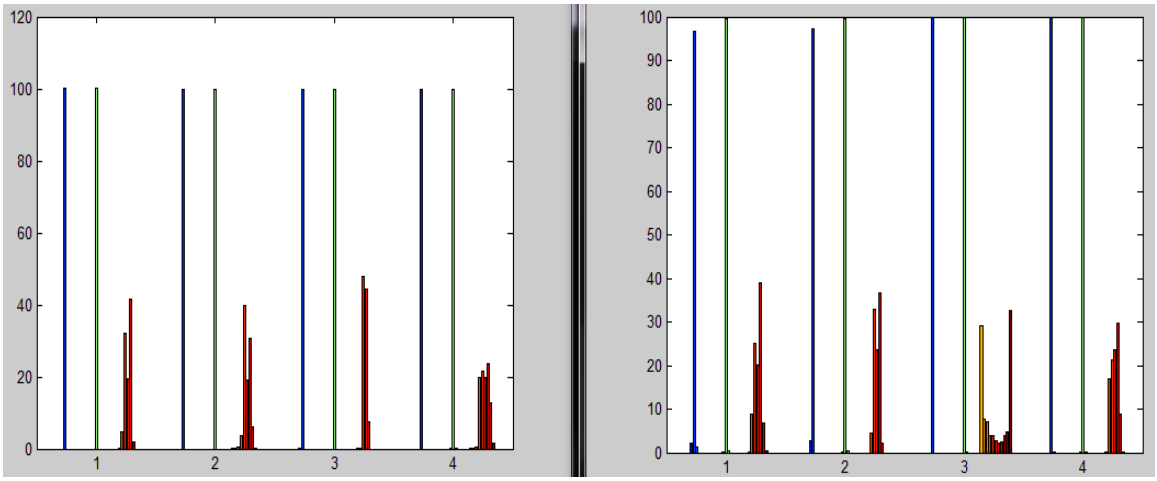


Fig. 4-21 FPFH of corresponding points

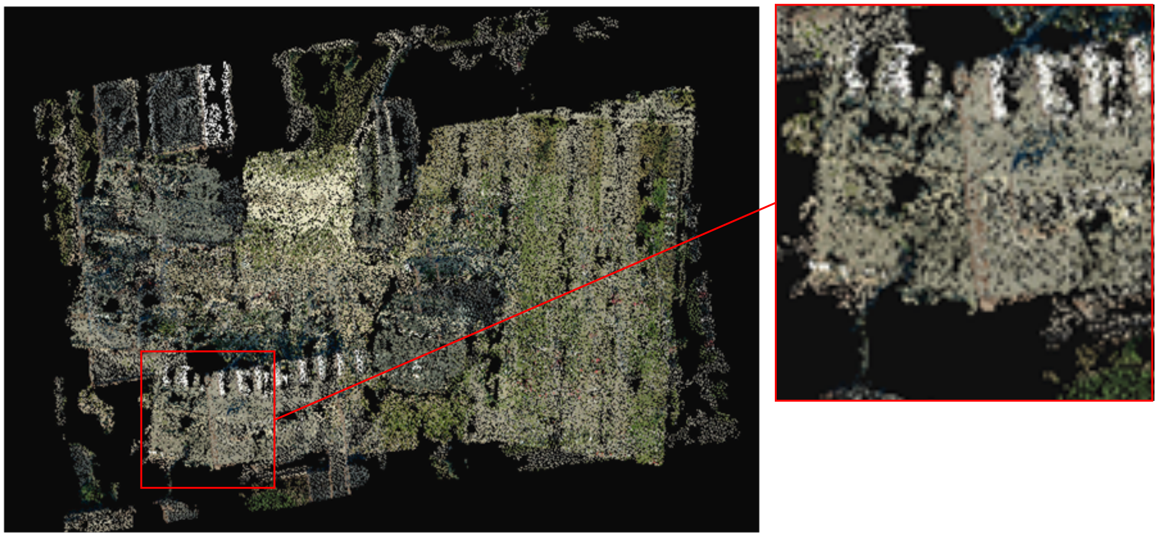


Fig. 4-22 Result of initial alignment

Before

The initial alignment result is shown in Fig 4-22. In general, a rough registration is achieved. Two point clouds have been moved and rotated to the similar position; but since the corresponding points are not perfect, there is still a slight shift between them. An improvement is required for further alignment.

After

4.2.8 Final Alignment

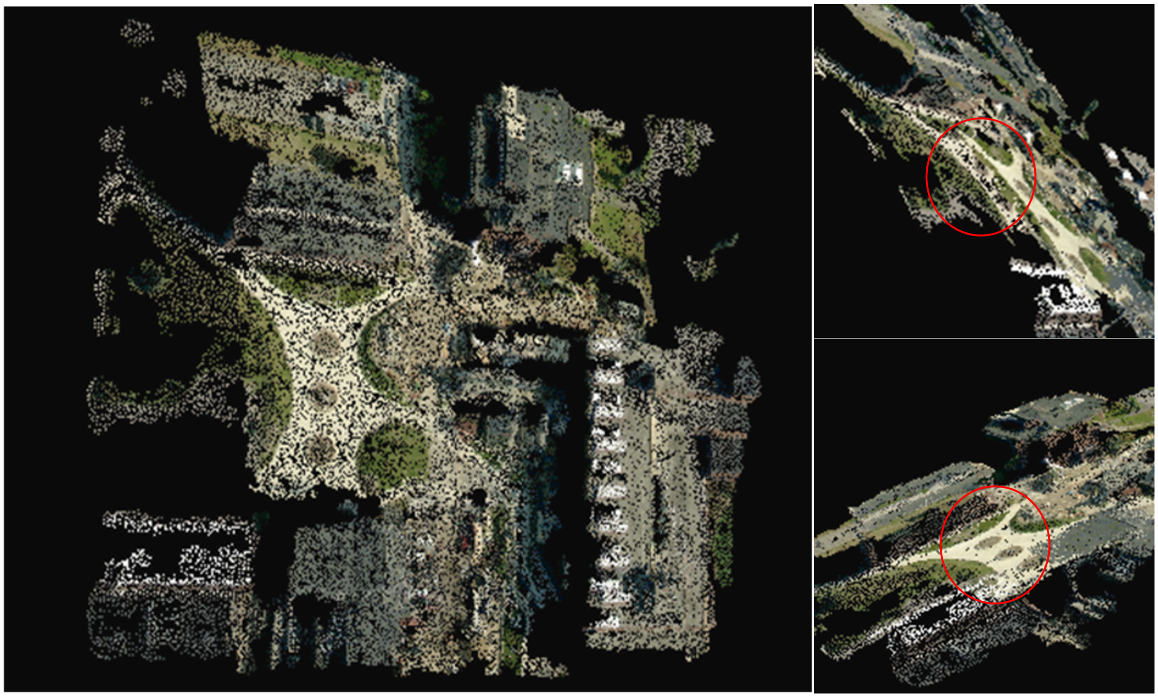


Fig. 4-23 Result of final alignment

To optimize the initial alignment result, a final alignment is performed using the ICP algorithm. The translation, rotation and scale parameters are all slightly modified in this step and the registration result is shown in Fig. 4-23. The points representing the same building, highlighted in Fig.4-22, overlap better now. Similarly, the offset between the points representing the same grass area, highlighted in Fig. 4-23, has also disappeared after the optimization. We can see that a more accurate result is finally achieved.

4.2.9 Compare to Manual Registration

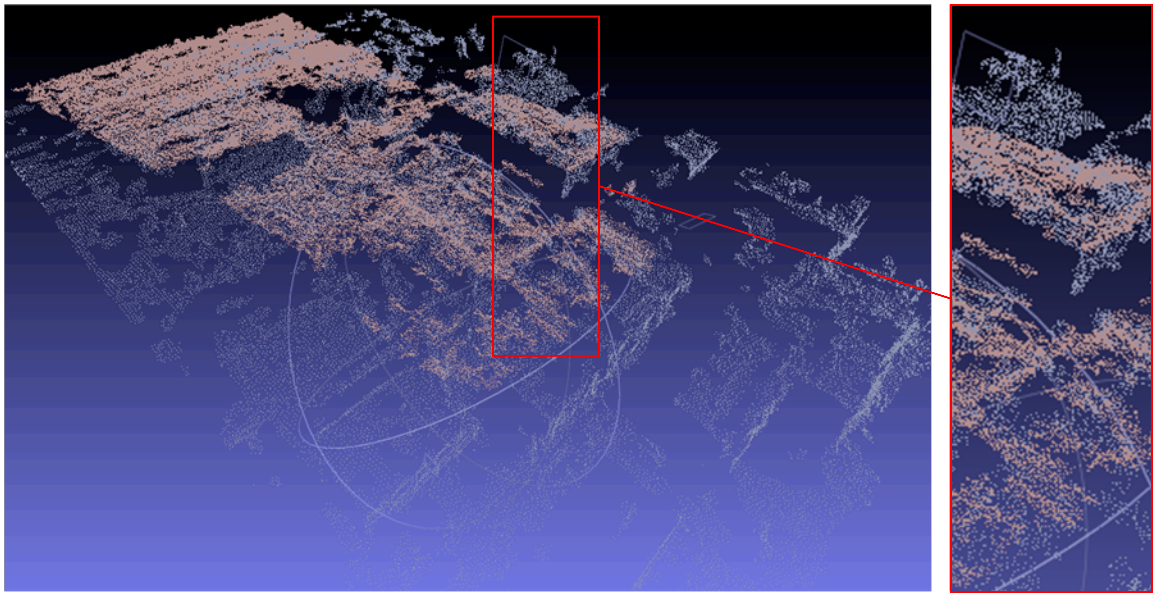


Fig. 4-24 Registration based on corresponding points selected manually (two point clouds are shown in pink and blue respectively)

The ground truth of the 3D points generated from Pictometry is unknown. So we just compare the registration result to a manual registration result based on manual selection of corresponding points. Compare the magnified details in Fig. 4-

23 and Fig. 4-24, we can see that both result achieved satisfying accuracy. From these figures, no offset between the point clouds is perceptible. The successful registration of point clouds with large amount of noise generated from Pictometry imagery proves the reliability of the method introduced in this thesis.

Chapter 5

Conclusion

5.1 Summary of work

The research presented in this thesis suggested a practical approach to automatically reconstruct a 3D building model from airborne oblique and nadir imagery. The multi-view Pictometry imagery used is obtained using a calibrated five-camera system. This set of images provides enough coverage and overlap for building a 3D model for the RIT campus area, which includes a tilted ground floor, buildings of different shapes and many other facilities. It is a good start for trying to reconstruct a complicated city scene. The ASIFT and focal length control have been used to adapt the RIT 3d point cloud extraction workflow to the Pictometry imagery. The modified version has been tested to perform well for both nadir and oblique images used in our research. The point cloud refinement methods used in the research have achieved the expected results in eliminating different kinds of noises and smoothing the surface. The 3d feature descriptors are defined based on the innovative usage of the color information. Through the multi scale persistence analysis, points with unique 3D FPFH feature are selected as candidates, from which corresponding pairs for the initial sample consensus alignment are found.

The initial alignment obtains a rough matching of two point clouds. Then a further improved final registration using ICP is realized. Since the ground truth of the point cloud generated is unknown, the result is only compared to that obtained from manually registration. A Chef data set is used to test the robustness of our method with artificial noise present. Overall, the 3D point cloud reconstruction and matching method applied to the oblique and nadir imagery in this thesis has reached the expectation in a limited condition. In the future research, the current approach will be optimized and a complete 3d model with meshed surface and texture information will be generated.

5.2 **Contribution** to the field

The research in this thesis presents a method to realize 3D building modeling using multi-view imagery and combination of different point clouds. Using 2D airborne oblique images to reconstruct 3D geographic scene has a promising market for future commercial usage in various areas like virtual tourism, navigation system, urban planning, visual military, and even insurance. The usage of oblique images for 3D city modeling can break the limitation of traditional vertical images. More detailed structures of the side facets can be better represented even when occlusion happens. Modeling results from oblique imagery can be used independently or integrated with existing models from other sources. Besides providing more complete information, it makes faster response possible compared to ground-based methods and costs much less expense than images taken by satellite or data generated by LiDAR. The 3D registration method proposed can not only be used to match point clouds generated from images taken in different viewing directions, but also to improve an existing model by adding additional parts extracted from a different source. The 3D feature descriptors are defined based on the innovative usage of the color information instead of normal make the alignment possible in a noisy condition that surface normal cannot be estimated accurately. In addition, based on further research, the surface and even the texture of buildings can be generated from these oblique images captured from different directions.

5.3 Limitations

The method used in this thesis is especially designed to adapt to the Pictometry airborne imagery and has its limitations as follows:

1. Require enough geographical overlap for images

The imagery used in this project only covers the RIT campus area. Every image covers a similar area. The geographical overlap between images is larger than 60%. It makes sure that enough corresponding could be found between images and the extraction of 3D point clouds is successful. It also guarantees large geographical overlap between the point clouds generated.

2. Require enough geometrical overlap between point clouds

The extraction of point clouds relies on proper disparity between the input images. Since images provided by Pictometry are taken in different flights on different date, only the nadir and north image sets lead to available point clouds suitable for later registration. These point clouds both have a large portion of points describing the ground scene and the roof facets. So the geometrical overlap is also larger than 60%. Our 3D feature is extract based on local structure, so if the geometrical overlap (amount of overlapped facets) is not large enough, the registration may fail even the geographical overlap is high.

3. The scene of RIT campus area consists of simple structures

Every buildings covered in the point clouds have unique contour outlines. Side facets of a building are almost always surrounded by distinguishable environment. There is no large area of high frequent patterns existing. These conditions make the registration much easier to implement. But for a scene that covers lots of repeated structure or buildings, the method presented in this thesis may fail.

4. The scale/density difference between the original clouds is small.

The registration method can only optimize the scale difference in a small range. Since in most case, we have preliminary knowledge of the scale of clouds need registration, we transform the Pictometry point clouds to similar scale by hand to simplify the process. The density of the point clouds is also similar, which confirms that the neighbors of a certain point in both clouds can represent similar surface feature correctly.

5. Only suitable for rigid registration.

Since the registration is for buildings which are fixed on the ground, we assumed that there is no distortion between the point clouds. The registration is based on rigid transformation. No distortion is considered. So this method may fail if it is used for registration for point cloud including large moving object.

5.4 Future work

1. Set up ground truth for the campus model. Compare the ground truth with our result to testify the reliability of the method presented in this thesis.

2. Utilize additional information to optimize the 3D feature. Currently, only the color information is considered. For a more complicated model, additional normal information may improve the accuracy of the corresponding pair selection and produce a better registration result in the initial alignment.

3. So far, the parameters used for the processing are selected manually. Try to select the parameters according to the characteristics of the point clouds themselves automatically.

4. Optimize the coding and make the whole process more computational efficient.

5. Optimize the algorithms to overcome the limitations listed in 5.3.

6. Realize the one by one matching of a series of point clouds. Apply the current method to different scenes to testify its robustness and optimize the algorithm accordingly. Also, try to apply this method to point clouds from different sources.

7. Mesh surfaces from the result point cloud and add texture information to the point cloud to obtain the final 3D campus model.

Reference

Agarwal, S and Snavely, N. Building Rome in a Day. Computer Vision, IEEE 12th International Conference, 2009.

Alexa, M and Behr, J. Computing and Rendering Point set surface. IEEE Transaction on Visualization and Computer Graphics, Vol 9, No.1, Jan-Mar， 2003

Bignone, F and Henricsson, O. Automatic Extraction of Generic House Ruffs from High Resolution Aerial Imagery. 1996

DeWitt, B and Wolf, P. Elements of Photogrammetry (with Applications in GIS), McGraw-Hill Higher Education, 3rd edition, 2000

Eggert, D and Lorusso, A. Estimating 3-D Rigid Body Transformations: A Comparison of Four Major Algorithms. Machine Vision and Applications, Volume 9, Numbers 5-6, 272-290, 1997.

Fischler, M and Bolles, R. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM, vol. 24, no. 6, pp. 381–395, 1981.

Frueh, C and Sammon, R. Automated Texture Mapping of 3D City Models with Oblique Aerial Imagery. 3D Data Processing, Visualization and Transmission, 2004.

Furukawa, Y and Ponce, J. Accurate, Dense, and Robust Multi-View Stereopsis. Conference on Computer Vision and Pattern Recognition, 2007. http://grail.cs.washington.edu/software/pmvs/

Gerke, M and Kerle, N. Automatic Structural Seismic Damage Assessment With Airborne Oblique Pictometry® Imagery. Photogrammetric Engineering & Remote Sensing, vol.77, No.9, Sep. 2011, pp.885-898

Harris, C. and Stephens, M. A combined corner and edge detector. Alvey Vision Converence, 1988, pp. 147-152.

Hartley, R and Zisserman, A. Multiple View Geometry in Computer Vision. Cambride University Press, 2nd edition, 2004.

Hirschmuller, H. Stereo Processing by Semi-Global Matching and Mutual Information. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2007

Horn, B. Closed-form Solution of Absolute Orientation Using Orthonormal Matrices. J Opt Soc Am Ser A 5: 1127-1135.

Jian, B and Vemuri, B. A Robust Algorithm for Point Set Registration Using Mixture of Gaussians. IEEE ICCV'05

Jurisch, A and Mountain, D. Evaluating the Viability of Pictometry® Imagery for Creating Models of the Built Environment. Computer Science, 2008, Volume 5072/2008, 663-677

Lemmens, M and Lemmens, C. Pictometry: Potentials for Land Administration. Strategic Integration of Surveying Services 6th FIG Regional Conference 2007.

Lowe, D. Distinctive image features from scale invariant keypoints, International journal of computer vision, vol. 60, no. 2, pp. 91–110, 2004.

Lourakis, M and Argyros, A. The Design and Implementation of a Generic Sparse Bundle Adjustment Software Package Based on the Levenberg-Marquardt Algorithm. Institute of Computer Science, 4th edition, 2004.

Luong, Q and Faugeras, A. The Fundamental Maxtrix: Theory, Algorithms, and Stability Analysis. International Journal of Computer Vision, 17, 43-75, 1996.

Matas, J and Chum, O. Robust wide-baseline stereo from maximally stable extremal regions. Image and Vision Computing, 22(10):761–767, 2004.

Mikolajczyk, K and Schmid, C. Scale & Affine Invariant Interest Point Detectors. International Journal of Computer Vision 60(1), 63–86, 2004.

More, J. The Levenberg-Marquardt algorithm: Implementation and theory. Lecture Notes in Mathematics, 1978, Volume 630/1978, 105-116

Morel, J and Yu, G. Asift: A New Framework for Fully Affine Invariant Image Comparison. SIAM Journal on Imaging Sciences, 2009.

Nilosek, D and Salvaggio, C. Applying Computer Vision Techniques to Perform Semi-automated Analytical Photogrammetry, IEEE, 2010

Nilosek, D and Walli, K. AeroSynth: Aerial Scene Synthesis from images. 2009.

Notargiacomo, R and Zhuang, L. Research Study of 3D Model Applications, 2012

PCL API Documentation 1.7.0, http://docs.pointclouds.org/trunk/index.html.

Richard Notargiacomo, R and Zhuang L. Research Study of 3D Model Applications, 2012.

Rusu, R and Marton, Z. Persistent Point Feature Histograms for 3D Point Clouds. in Proceedings of the 10th International Conference on Intelligent Autonomous Systems 56, 2008.

Rusu, R and Marton, Z. Towards 3D Point Cloud based Object Maps for Household Environments. Robotics and Automation, 2009.

Rusu, R and Blodow, N. Fast Point Histograms for 3D registration. Robotics and Automation, 2009.

Sedaghat, A and Mokhtarzade, M. Uniform Robust Scale-Invariant Feature Matching for Optical Remote Sensing Images. IEEE Transactions on Geoscience and Remote Sensing, VOL. 49, NO. 11, 2011

Snavely, N, Seitz, S and Szeliski R. Bundler v0.4 User's Manual, 2008. http://phototour.cs.washington.edu/bundler/bundler-v0.4-manual.html.

Tang, P and Huber, D. Automatic Reconstruction of As-built Building Information Models from Laser-scanned Point Clouds: A Review of Related Techniques. Automation in Construction, volume 19, Issue 7, Nov. 2010.

Triggs, B and McLauch;an, P. Bundle Adjustment – Amodern Synthesis. Vision Algorithms: Theory and Practice. Lecture Notes in Computer ScienceVolume 1883, 2000, pp 298-372.

Tuytelaars, T and Gool, L. Matching Widely Separated Views Based on Affine Invariant Regions. International Journal of Computer Vision, 59(1):61–85, 2004.

Walli, K. Relating Multimodal Imagery Data in 3D, PhD dissertation, RIT, 2010.

Wang, Y and S. Schultz, S. Pictometry's Proprietary Airborne Digital Imaging System and Its Application in 3D City Modeling. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. Vol. XXXVII. Part B1. Beijing 2008.

Werner, T. Matching of Line Segments Across Multiple Views: Implementation Description (memo). 2002.

You, R and Lin B. A Quality Prediction Method for Building Model reconstruction Using LiDAR Data and Topographic Maps. IEEE Transactions on Geoscience and Remote Sensing. Vol. 49, No.9, Sep. 2011

Zhang, Z. Iterative Point Matching for Registration of Free-Form Curves and Surfaces. International Journal of Computer Vision, 13:2, 119-152, 1994.

Appendix

C++ Code (3D point clouds processing for Pictometry data)

```
#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <iterator>

#include <pcl/point_types.h>
#include <pcl/io/pcd_io.h>
#include <pcl/io/ply_io.h>
#include <pcl/visualization/cloud_viewer.h>
#include <boost/thread/thread.hpp>
#include "pcl/visualization/pcl_visualizer.h"

#include <pcl/features/normal_3d.h>
#include <pcl/filters/voxel_grid.h>
#include <pcl/filters/statistical_outlier_removal.h>
#include <pcl/filters/radius_outlier_removal.h>
#include <pcl/surface/mls.h>
#include <pcl/keypoints/sift_keypoint.h>
#include <pcl/features/fpfh.h>
#include <pcl/registration/ia_ransac.h>
#include <pcl/registration/icp.h>

int SimViewRGB (pcl::PointCloud<pcl::PointXYZRGB>::Ptr Cloud)
{
        pcl::visualization::CloudViewer viewer ("Simple Cloud Viewer");
        viewer.showCloud(Cloud);
        while (!viewer.wasStopped ())
        {
         }
        return 0;
}

int NormView (pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloudRGB,pcl::PointCloud<pcl::Normal>::Ptr
Normal)
{
        std::cerr<<"Show the normals...(Close the pop-out viewer to continue)"<<std::endl;
        boost::shared_ptr<pcl::visualization::PCLVisualizer> viewer (new pcl::visualization::PCLVisualizer
("Normal Viewer"));
        viewer->setBackgroundColor (0, 0, 0);
        pcl::visualization::PointCloudColorHandlerRGBField<pcl::PointXYZRGB> rgb(cloudRGB);
        viewer->addPointCloud<pcl::PointXYZRGB> (cloudRGB, rgb, "sample cloud");
        viewer->setPointCloudRenderingProperties (pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 3,
"sample cloud");
        viewer->addPointCloudNormals<pcl::PointXYZRGB, pcl::Normal> (cloudRGB, Normal, 15, 0.05,
"normals");
        while (!viewer->wasStopped())
        {
                viewer->spinOnce ();
        }
```

```
	return 0;
}

void KeyView (pcl::PointCloud<pcl::PointXYZRGB>::Ptr Cloud, pcl::PointCloud<pcl::PointXYZRGB>::Ptr
Key)
{
	int v1(0);
	boost::shared_ptr<pcl::visualization::PCLVisualizer> viewer (new pcl::visualization::PCLVisualizer
("Points with persistent features"));
	viewer->setBackgroundColor (0, 0, 0, v1);
	//pcl::visualization::PointCloudColorHandlerRGBField<pcl::PointXYZRGB> rgb(Cloud);
	pcl::visualization::PointCloudColorHandlerCustom<pcl::PointXYZRGB> single_color1(Key, 255, 255,
255);
	viewer->addPointCloud<pcl::PointXYZRGB>(Cloud,single_color1,"cloud",v1);
	viewer->setPointCloudRenderingProperties (pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 2,
"cloud");
	pcl::visualization::PointCloudColorHandlerCustom<pcl::PointXYZRGB> single_color2(Key, 255, 0,
0);
	viewer->addPointCloud<pcl::PointXYZRGB>(Key,single_color2,"key",0);
	viewer->setPointCloudRenderingProperties (pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 2,
"key");
	while (!viewer->wasStopped ())
	{
		viewer->spinOnce (100);
	 }
}

pcl::PointCloud<pcl::PointXYZRGB>::Ptr VoxFilterSave(pcl::PointCloud<pcl::PointXYZRGB>::Ptr Cloud,
std::string FilterFile)
{
	pcl::VoxelGrid<pcl::PointXYZRGB> vg;
	vg.setInputCloud (Cloud);
	vg.setLeafSize (0.0005f, 0.0005f, 0.0005f);
	pcl::PointCloud<pcl::PointXYZRGB>::Ptr Cloud_out (new pcl::PointCloud<pcl::PointXYZRGB>);
	vg.filter(*Cloud_out);
	std::cerr<<"Dowsampled to "<< Cloud_out->points.size()<<" points"<< std::endl;

	pcl::io::savePCDFile (FilterFile, *Cloud_out);
	std::cerr<<"Show the downsampled points...(Close the pop-out viewer to continue)"<<std::endl;
	SimViewRGB(Cloud_out);
	return(Cloud_out);
}

pcl::PointCloud<pcl::PointXYZRGB>::Ptr SorFilterSave (pcl::PointCloud<pcl::PointXYZRGB>::Ptr Cloud,
std::string FilterFile)
{
	pcl::StatisticalOutlierRemoval<pcl::PointXYZRGB> sor(true);
	sor.setInputCloud(Cloud);
	sor.setMeanK(50);//choose a suitable parameter manually//50-30
	sor.setStddevMulThresh(0.5);//choose a suitable parameter manually//0.5-0.8
	pcl::PointCloud<pcl::PointXYZRGB>::Ptr Cloud_out (new pcl::PointCloud<pcl::PointXYZRGB>);
	sor.filter(*Cloud_out);
	std::cerr << "Left " << Cloud_out->points.size () << " data points after SOR filter, removed
"<<sor.getRemovedIndices()->size()<< "points"<<std::endl;

	pcl::io::savePCDFile (FilterFile, *Cloud_out);
	std::cerr<<"Show the filtered points...(Close the pop-out viewer to continue)"<<std::endl;
	SimViewRGB(Cloud_out);
	return(Cloud_out);
```

```
}

pcl::PointCloud<pcl::PointXYZRGB>::Ptr RorFilterSave (pcl::PointCloud<pcl::PointXYZRGB>::Ptr Cloud,
std::string FilterFile)
{
	pcl::RadiusOutlierRemoval<pcl::PointXYZRGB> ror(true);
	ror.setInputCloud(Cloud);
	ror.setMinNeighborsInRadius(30);//choose a suitable parameter manually
	ror.setRadiusSearch(0.01);//choose a suitable parameter manually
	pcl::PointCloud<pcl::PointXYZRGB>::Ptr Cloud_out (new pcl::PointCloud<pcl::PointXYZRGB>);
	ror.filter(*Cloud_out);
	std::cerr << "Left " << Cloud_out->points.size () << " data points after ROR filter, removed
"<<ror.getRemovedIndices()->size()<< "points"<<std::endl;

	pcl::io::savePCDFile (FilterFile, *Cloud_out);
	std::cerr<<"Show the filtered points...(Close the pop-out viewer to continue)"<<std::endl;
	SimViewRGB(Cloud_out);
	return(Cloud_out);
}

pcl::PointCloud<pcl::PointXYZRGB>::Ptr MlsFilterSave(pcl::PointCloud<pcl::PointXYZRGB>::Ptr Cloud,
std::string FilterFile)
{
	pcl::PointCloud<pcl::PointXYZ>::Ptr cloudXYZ (new pcl::PointCloud<pcl::PointXYZ>);
	pcl::copyPointCloud(*Cloud,*cloudXYZ);
	pcl::search::KdTree<pcl::PointXYZ>::Ptr tree (new pcl::search::KdTree<pcl::PointXYZ>);
	pcl::PointCloud<pcl::PointNormal> mls_points;
	pcl::MovingLeastSquares<pcl::PointXYZ, pcl::PointNormal> mls;
	mls.setComputeNormals (false);
	mls.setInputCloud (cloudXYZ);
	mls.setPolynomialFit (true);
	mls.setSearchMethod (tree);
	mls.setSearchRadius (0.01);
	mls.process (mls_points);
	std::cerr<<"Complete surface smoothing! "<< std::endl;
	pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloudRGB (new pcl::PointCloud<pcl::PointXYZRGB>);
	pcl::copyPointCloud(*Cloud,*cloudRGB);
	pcl::copyPointCloud(mls_points,*cloudRGB);
	SimViewRGB(cloudRGB);
	pcl::io::savePCDFile (FilterFile, *cloudRGB);
	return(cloudRGB);
}

pcl::PointCloud<pcl::Normal>::Ptr getNormals( pcl::PointCloud<pcl::PointXYZRGB>::Ptr Cloud )
{
	pcl::NormalEstimation<pcl::PointXYZRGB, pcl::Normal> norm_est;
	norm_est.setInputCloud( Cloud );
	norm_est.setRadiusSearch(0.005);
	pcl::PointCloud<pcl::Normal>::Ptr Normals (new pcl::PointCloud<pcl::Normal>);
	norm_est.compute( *Normals );
	for (size_t i = 0; i < Normals->points.size (); ++i)
	{
		if (Normals->points[i].normal_z < 0)
		{
			Normals->points[i].normal_x = -Normals->points[i].normal_x;
			Normals->points[i].normal_y = -Normals->points[i].normal_y;
			Normals->points[i].normal_z = -Normals->points[i].normal_z;
		}
	}
```

```
        std::cerr<<"Normal estimated!"<<endl;
        return(Normals);
 }

pcl::PointCloud<pcl::PointXYZRGB>::Ptr scaleCloud (pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloudRGB,
float s)
{
        for (size_t i = 0; i < cloudRGB->points.size (); ++i)
        {
                cloudRGB->points[i].x = (cloudRGB->points[i].x)*s;
                cloudRGB->points[i].y = (cloudRGB->points[i].y)*s;
                cloudRGB->points[i].z = (cloudRGB->points[i].z)*s;
        }
        return(cloudRGB);
}

pcl::PointCloud<pcl::Normal>::Ptr RGB2Normal (pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloudRGB,
std::string filename)
{
        //can not use normal2RGB+RGB2normal to return the original normal, because RGB lost sign.
        pcl::PointCloud<pcl::Normal>::Ptr normal (new pcl::PointCloud<pcl::Normal>);
        pcl::copyPointCloud(*cloudRGB,*normal);
        for (size_t i = 0; i < cloudRGB->points.size (); ++i)
        {
                normal->points[i].normal_x = float(cloudRGB->points[i].r) /255.0;
                normal->points[i].normal_y = float(cloudRGB->points[i].g) /255.0;
                normal->points[i].normal_z = float(cloudRGB->points[i].b) /255.0;
        }
        pcl::io::savePCDFile (filename, *normal);
        return(normal);
}

pcl::PointCloud<pcl::PointWithScale>::Ptr getKeys (pcl::PointCloud<pcl::PointXYZRGB>::Ptr Cloud)
{
        float min_scale = 0.001; //0.001
        int nr_octaves = 4;
        int nr_scales_per_octave = 4;  //5
        float min_contrast = 5; //7 for RGB
        pcl::SIFTKeypoint<pcl::PointXYZRGB, pcl::PointWithScale> sift;
        pcl::PointCloud<pcl::PointWithScale>::Ptr sifts (new pcl::PointCloud<pcl::PointWithScale>);
        pcl::search::KdTree<pcl::PointXYZRGB>::Ptr tree(new
pcl::search::KdTree<pcl::PointXYZRGB> );//new API
        sift.setInputCloud(Cloud);
        sift.setSearchMethod (tree);
        sift.setScales(min_scale, nr_octaves, nr_scales_per_octave);
        sift.setMinimumContrast(min_contrast);
        sift.compute (*sifts);
        cerr <<"Computed "<<sifts->points.size ()<<" SIFT Keypoints"<<endl;
        return(sifts);
}

pcl::PointCloud<pcl::FPFHSignature33>::Ptr getSiftPers (pcl::PointCloud<pcl::PointXYZRGB>::Ptr keys,
pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloudRGB,pcl::PointCloud<pcl::Normal>::Ptr normals,std::string
file_pers, std::string file_ind, std::string file_cloud)
{
        pcl::FPFHEstimation<pcl::PointXYZRGB,pcl::Normal,pcl::FPFHSignature33>::Ptr FPFH (new
pcl::FPFHEstimation<pcl::PointXYZRGB,pcl::Normal, pcl::FPFHSignature33>());
        FPFH->setInputCloud(keys);
        FPFH->setInputNormals(normals);
        FPFH->setSearchSurface(cloudRGB);
        pcl::search::KdTree<pcl::PointXYZRGB>::Ptr tree (new pcl::search::KdTree<pcl::PointXYZRGB>);
        FPFH->setSearchMethod (tree);

        std::vector<float> scale; //or 12,15,18
        scale.push_back(0.015);
        scale.push_back(0.020);
        scale.push_back(0.025);
        pcl::PointCloud<pcl::FPFHSignature33>::Ptr mfps (new pcl::PointCloud<pcl::FPFHSignature33> ());
        std::vector< int > ind;

        pcl::MultiscaleFeaturePersistence< pcl::PointXYZRGB,pcl::FPFHSignature33 > MFP;
        MFP.setInputCloud(keys);
        MFP.setFeatureEstimator (FPFH);
        MFP.setScalesVector (scale);
        MFP.setAlpha(1.2);//miu +/- 3sigma
        std::cerr<<"start determine..."<<endl;
        MFP.findPersistentFeatures(*mfps,ind);

        int n_p = ind.size();
        std::cerr<<"Found "<<n_p<<" persistent features!"<<endl;

        pcl::io::savePCDFile (file_pers, *mfps);

        std::ofstream f_ind(file_ind);
        std::ostream_iterator<int> output_iterator(f_ind, "\n");
        std::copy(ind.begin(), ind.end(), output_iterator);

        pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloudRGBpers (new
pcl::PointCloud<pcl::PointXYZRGB>());
        cloudRGBpers->width    = n_p;
        cloudRGBpers->height   = 1;
        cloudRGBpers->is_dense = false;
        cloudRGBpers->points.resize (cloudRGBpers->width * cloudRGBpers->height);
        for (size_t i = 0;i<n_p; i++)
        {
                cloudRGBpers->points[i].x = keys->points[ind[i]].x;
                cloudRGBpers->points[i].y = keys->points[ind[i]].y;
                cloudRGBpers->points[i].z = keys->points[ind[i]].z;
        }
        pcl::io::savePCDFile (file_cloud, *cloudRGBpers);
        return(mfps);
}

pcl::PointCloud<pcl::FPFHSignature33>::Ptr getPers (pcl::PointCloud<pcl::PointXYZRGB>::Ptr
cloudRGB,pcl::PointCloud<pcl::Normal>::Ptr normals,std::string file_pers, std::string file_ind, std::string
file_cloud)
{
        pcl::FPFHEstimation<pcl::PointXYZRGB,pcl::Normal,pcl::FPFHSignature33>::Ptr FPFH (new
pcl::FPFHEstimation<pcl::PointXYZRGB,pcl::Normal, pcl::FPFHSignature33>());
        FPFH->setInputCloud(cloudRGB);
        FPFH->setInputNormals(normals);
        FPFH->setSearchSurface(cloudRGB);
        pcl::search::KdTree<pcl::PointXYZRGB>::Ptr tree (new pcl::search::KdTree<pcl::PointXYZRGB>);
        FPFH->setSearchMethod (tree);

        std::vector<float> scale; //or 12,15,18
        scale.push_back(0.010);
        scale.push_back(0.015);
```

```
    scale.push_back(0.020);
    pcl::PointCloud<pcl::FPFHSignature33>::Ptr mfps (new pcl::PointCloud<pcl::FPFHSignature33> ());
    std::vector< int > ind;

    pcl::MultiscaleFeaturePersistence< pcl::PointXYZRGB,pcl::FPFHSignature33 > MFP;
    MFP.setInputCloud(cloudRGB);
    MFP.setFeatureEstimator (FPFH);
    MFP.setScalesVector (scale);
    MFP.setAlpha(1.6);//miu +/- 3sigma
    std::cerr<<"start determine..."<<endl;
    MFP.findPersistentFeatures(*mfps,ind);

    int n_p = ind.size();
    std::cerr<<"Found "<<n_p<<" persistent features!"<<endl;

    pcl::io::savePCDFile (file_pers, *mfps);

    std::ofstream f_ind(file_ind);
    std::ostream_iterator<int> output_iterator(f_ind, "\n");
    std::copy(ind.begin(), ind.end(), output_iterator);

    pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloudRGBpers (new
pcl::PointCloud<pcl::PointXYZRGB>());
    cloudRGBpers->width    = n_p;
    cloudRGBpers->height   = 1;
    cloudRGBpers->is_dense = false;
    cloudRGBpers->points.resize (cloudRGBpers->width * cloudRGBpers->height);
    for (size_t i = 0;i<n_p; i++)
    {
        cloudRGBpers->points[i].x = cloudRGB->points[ind[i]].x;
        cloudRGBpers->points[i].y = cloudRGB->points[ind[i]].y;
        cloudRGBpers->points[i].z = cloudRGB->points[ind[i]].z;
    }
    pcl::io::savePCDFile (file_cloud, *cloudRGBpers);
    return(mfps);
}


int main (int argc, char** argv)
{

    pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud1RGB (new pcl::PointCloud<pcl::PointXYZRGB>);
    pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud2RGB (new pcl::PointCloud<pcl::PointXYZRGB>);
    pcl::io::loadPLYFile ( "Nadir.ply", *cloud1RGB); //Load original point clouds
    pcl::io::loadPLYFile ( "North.ply", *cloud2RGB);

    float s = 1.0/1.54;
    cloud2RGB = scaleCloud(cloud2RGB,s);

    cloud1RGB = VoxFilterSave(cloud1RGB,"vf1.pcd");//Down sampling
    cloud1RGB = SorFilterSave(cloud1RGB,"sf1.pcd");//SOR filter
    cloud1RGB = RorFilterSave(cloud1RGB,"rf1.pcd");//ROR filter
    cloud1RGB = MlsFilterSave(cloud1RGB,"mf1.pcd");//MLS smoother

    cloud2RGB = VoxFilterSave(cloud2RGB,"vf2.pcd");
    cloud2RGB = SorFilterSave(cloud2RGB,"sf2.pcd");
    cloud2RGB = RorFilterSave(cloud2RGB,"rf2.pcd");
    cloud2RGB = MlsFilterSave(cloud2RGB,"mf2.pcd");
```

```
        pcl::PointCloud<pcl::Normal>::Ptr norm_rgb1 (new pcl::PointCloud<pcl::Normal>);
        pcl::PointCloud<pcl::Normal>::Ptr norm_rgb2 (new pcl::PointCloud<pcl::Normal>);
        norm_rgb1 = RGB2Normal(cloud1RGB,"norm_rgb1.pcd");//Use color as normals
        norm_rgb2 = RGB2Normal(cloud2RGB,"norm_rgb2.pcd");

        pcl::PointCloud<pcl::FPFHSignature33>::Ptr feature1 (new pcl::PointCloud<pcl::FPFHSignature33>);
        pcl::PointCloud<pcl::FPFHSignature33>::Ptr feature2 (new pcl::PointCloud<pcl::FPFHSignature33>);
        //Unique points obtained  by persistence analysis for FPFH
        feature1 = getPers(cloud1RGB,norm_rgb1,"pers1.pcd","ind1.txt","cloudRGBpers1.pcd");
        feature2 = getPers(cloud2RGB,norm_rgb2,"pers2.pcd","ind2.txt","cloudRGBpers2.pcd");

        pcl::PointCloud<pcl::PointXYZRGB>::Ptr key1pts (new
pcl::PointCloud<pcl::PointXYZRGB>),key2pts (new pcl::PointCloud<pcl::PointXYZRGB>);
        pcl::io::loadPCDFile ("cloudRGBpers1.pcd", *key1pts);
        pcl::io::loadPCDFile ("cloudRGBpers2.pcd", *key2pts);
        KeyView(cloud1RGB,key1pts);
        KeyView(cloud2RGB,key2pts);

        std::cerr<<"Start matching..."<<endl;
        pcl::SampleConsensusInitialAlignment<pcl::PointXYZRGB,pcl::PointXYZRGB,pcl::FPFHSignature3
3> sacIA; //Initial alignment
        sacIA.setMaximumIterations(1000);
        sacIA.setMinSampleDistance(0.001);//small is better, like voxel
        sacIA.setMaxCorrespondenceDistance(1);
        sacIA.setNumberOfSamples(4);
        sacIA.setCorrespondenceRandomness(10);
        sacIA.setInputTarget(key1pts);
        sacIA.setTargetFeatures(feature1);
        sacIA.setInputCloud(key2pts);
        sacIA.setSourceFeatures(feature2);
        pcl::PointCloud<pcl::PointXYZRGB>::Ptr registration_output (new
pcl::PointCloud<pcl::PointXYZRGB>);
        sacIA.align(*registration_output);
        pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud2RGB_new (new
pcl::PointCloud<pcl::PointXYZRGB>);
        Eigen::Matrix4f transform = sacIA.getFinalTransformation();
        std::cerr<< "Cloud2(Input) is transformed by"<<std::endl<<transform<<endl;
        std::ofstream tran("trans_sac.txt");
        tran<<transform<<std::endl;
        transformPointCloud(*cloud2RGB,*cloud2RGB_new,transform);
        pcl::io::savePCDFile ("2_sac.pcd", *cloud2RGB_new);
        pcl::PointCloud<pcl::PointXYZRGB> final = *cloud2RGB_new;
        final += *cloud1RGB;
        pcl::io::savePCDFile ("sac_final.pcd", final);

        pcl::PointCloud<pcl::PointXYZRGB>::Ptr finalpoints (new pcl::PointCloud<pcl::PointXYZRGB>);
        pcl::io::loadPCDFile ("sac_final.pcd", *finalpoints);
        SimViewRGB(finalpoints);

        std::cerr<<"Refine matching..."<<endl;
pcl::IterativeClosestPoint<pcl::PointXYZRGB, pcl::PointXYZRGB> icp; //Final ICP
        icp.setInputCloud(cloud2RGB_new);
        icp.setInputTarget(cloud1RGB);
        icp.setMaximumIterations(50);
        icp.setMaxCorrespondenceDistance(0.2);
        icp.setRANSACOutlierRejectionThreshold(0.15);//0.1 works
        pcl::PointCloud<pcl::PointXYZRGB> icpFinal;
        icp.align(icpFinal);
        pcl::io::savePCDFile ("cloud2RGBnew_icp.pcd", icpFinal);
```

```
    icpFinal += *cloud1RGB;
    pcl::io::savePCDFile ("icpFinal.pcd", icpFinal);
    pcl::PointCloud<pcl::PointXYZRGB>::Ptr icpfinalpoints (new pcl::PointCloud<pcl::PointXYZRGB>);
    pcl::io::loadPCDFile ("icpFinal.pcd", *icpfinalpoints);
    std::cerr<<"Show the icp result:"<<endl;
    SimViewRGB(icpfinalpoints);
    std::cout << "has converged:" << icp.hasConverged() << " score: " <<
    icp.getFitnessScore() << std::endl;
    Eigen::Matrix4f tranm = icp.getFinalTransformation();
    std::cout << "Transform matrix:"<< std::endl << tranm << std::endl;
    std::ofstream trans("trans_icp.txt");
    trans<<tranm<<std::endl;

    Eigen::Matrix4f t_final = transform * tranm;
    std::cout << "Final Transform matrix:"<< std::endl << t_final << std::endl;
    std::ofstream transf("trans_final.txt");
    transf<<t_final<<std::endl;
    float s_final = sacIA.sacScale*icp.icpScale;
    cerr<<"Final Scale estimated is to be "<<s_final<<endl;
    std::ofstream scales("scales.txt");
    scales<<sacIA.sacScale<<endl<<icp.icpScale<<endl<<s_final<<endl;
    float error_scale = abs(s_final - inputscale)/inputscale;
    cerr<<"Scale error is "<<error_scale*100<<"%"<<endl;

     system("pause");
     return (0);
}
```