Abstract

There is considerable evidence to show that interaction between human and computer has received much attention from both academia and user (Zabulis, et al., 2009). The fundamental idea of the project is to make use of hand gesture to achieve a practical and interactive functionality between user and computer. Precisely, a system which enables users to segment reconstructed objects by making use of their gestures is proposed in this thesis. The reason why this project was undertaken is divided into two factors:

- The importance and practical applicability of the project idea
- The innovation and added value of the technical methods

Apparently, the methods related to this thesis contain four components: he Kinect, object reconstruction method based on Point Cloud Library, gesture tracking and object segmentation. The scope of this project can be defined as human-computer interaction and graphics. Although depth extraction using a stereo system is not a novel topic and concept, my project focuses on the combination of gesture tracking algorithm and object segmentation in a 3D depth reconstruction scene.

More importantly, the major contributions and achievements of this project are summarised as follows:

- The features and functionality of Kinect device have been grasped and applied in this project. More importantly, the thesis proposed and implemented the Kinect calibration process.
- Object reconstruction algorithm based on Point Cloud Library has been proposed and implemented in this project. Moreover, the performance of the reconstruction process has been investigated and analyzed.
- A novel gesture tracking and recognition algorithm: simultaneous 3D gesture was proposed in this paper. In addition, the feasibility of this method was verified by proposing the evaluation process.
- The object segmentation component which combines the object reconstruction algorithm (based on PCL) and simultaneous 3D gestures was developed as the primary functionality of my software. The paper has investigated the performance of the segmentation algorithm.
- The final contribution is the thesis has verified the validity of the entire software by conducting experimental analysis and investigating its performance and functionality. The evaluation process was proposed by making comparison with related conventional algorithms.

Contents

I. INTRODUCTION

1.1 Background and Motivation

Multiple lines of research have converged on a critical role for hand gestures in human-computer interactions. According to Zabulis, et al. (2009), hand gesture techniques might be the most natural and intuitive way for humans to communicate with machines. The naturalness and intuitiveness of gesture tracking and recognition have been employed in various areas and have produced a series of applications. Making use of gesture tracking to segment objects has emerged as a novel subject in information technology.

In terms of the new approach to reconstruct 3D objects in real-time scenes, KinectFusion algorithm plays a crucial role in this process (Izadi, et al., 2011). The Kinect, a depth camera which can get depth maps and colour images simultaneously, is the essential device of the algorithm. Specifically, recognized gestures can undertake object segmentation in the 3D depth environment created by the KinectFusion.

It should be mentioned that the motivation of this project can be separated into three aspects. First and foremost, my supervisor of this thesis Professor Mike Fraser, who leads the Bristol Interaction and Graphics Group, provided me with this brilliant project idea. Moreover, he proposed me to research the possibility and solutions of the project. Secondly, the practical applicability and the potential added value of this project are attractive. Last but not least, from my own perspective, the combination of gesture, reconstruction, segmentation and Kinect is dramatic.

1.2 Aims and Objectives

After providing the brief background and motivation, the aims and objectives of my project should be specified and presented. In general, this project involved in software development type and investigatory type, focuses on coming up with new techniques: combining gesture tracking method with object segmentation algorithm in a 3D reconstruction scene. More exactly, the general aim of the thesis can be separated into two elements: software aim and investigatory aim.

- The software aim can be described as: design and develop a system which provides user with the 3D objects reconstruction and enables user to interact with the objects and segment objects by utilizing hand gesture.

- The investigatory aim is verifying the feasibility and validity of the new techniques by conducting experimental analysis, investigating and evaluating the performance of the novel method.

Referring to my project, the fundamental idea of the project is to make use of hand gesture to achieve a practical and interactive functionality between user and computer. More precisely, my project can be described as follows: by combining with segmentation algorithm, hand gesture is employed to conduct the segmentation of the 3D models reconstructed by KinectFusion algorithm (based on Point Cloud Library, the open source version).

It should be clear that the issues and techniques of this project can be classified as three types: object reconstruction, gesture tracking and object segmentation. Specifically, after introducing the three types of techniques and algorithms, the detailed objectives of this project are presented as follows:

- The first objective is to specify the features and investigating the strategy of KinectFusion (based on Point Cloud Library, the open source version).
- The next objective is to prove KinectFusion (based on PCL) is an efficient and suitable reconstruction algorithm, by make comparison with other conventional methods.
- Thirdly, to propose an original gesture tracking algorithm: simultaneous 3D gestures. This method is the crucial technique utilised in my project to track and recognise gestures.
- Subsequently, to develop the segmentation method which combines the object reconstruction method (based on PCL) and simultaneous 3D gestures together is the fourth objective.
- Finally, to propose and specify the measurable elements and evaluation processes of the three techniques to estimate the performance of the software.

1.3 Outline and Contents

In this section, the structure and contents of the report is specified and presented. Besides the introduction chapter, this paper still contains 5 main chapters. The remaining chapters are organized as follows: chapter II provides the detailed background and related techniques of the entire project. The series of algorithms, methods and techniques contain the Kinect frameworks, different object reconstruction and segmentation methods, KinectFusion algorithm and gesture tracking. In addition, the evaluation methods are specified. More specifically, at the end of each subsection, a

brief summary is presented to specify the exact relation between the each method and this project.

Chapter III illustrates the overall solution and workflow of the project. The hardware requirement and programming environment configuration are also specified in this chapter. Moreover, the procedure of the software design and development are presented in chapter III. The experimental results of the software are presented and the evaluation processes of the project are conducted in the chapter IV. Subsequently, the advantages of algorithms, the overall performance of the system, the added values and the achievements of the project objectives are analyzed and discussed in the same chapter. Finally, a conclusion which includes the future work and improvements are presented.

II. BACKGROUND & RELATED TECHNIQUES

In this chapter, the background of my project is introduced. More importantly, the related techniques, methods and algorithms involved in object reconstruction, gesture tracking, object segmentation and PCL are also presented in this chapter. In addition, the relationships between these techniques and my project will be specified and interpreted. By considering the software designing and developing scenario, techniques related to the project are presented.

2.1 Kinect

2.1.1 Introduction

Microsoft Kinect is a motion-sensing input device developed by Microsoft, and it interacts with the XBOX360 as a new game controller (Figure 2.1.1) (Frati and Prattichizzo, 2011). The name of this device: Kinect was announced officially on 13th June, 2010 during the Electronic Entertainment Expo 2010 by Microsoft. It must be recognized that increasing attention has been paid on the Kinect related projects and techniques (Smisek, et al., 2011).

Figure 2.1.1 Microsoft Kinect

Source: Frati and Prattichizzo, 2011.

Microsoft has noticed this circumstance, and has released the Kinect Software Development Kit for Windows on 16th June, 2011. Obviously, Kinect is not just an innovative game controller; it can be used on computers and other devices to make use of the sensing camera, accelerometer and microphones (ibid). In the next part of this subsection, the hardware of Kinect is described and the unique technical features are presented.

2.1.2 Hardware and Technical Features

First of all, the arrangement of the three major sensing components of Kinect is introduced. The sensing hardware of the Kinect device includes an infrared projector, colour image camera and an infrared camera (Figure 2.1.2)

(Leyvand, et al., 2011). Obviously, the depth sensor is made up of two components: the infrared projector and the infrared camera, the latter being a monochrome complementary metal-oxide semiconductor (CMOS) sensor. Kinect is able to simultaneously recognise up to six people and analyse with a feature extraction of 20 joints per person (ibid).

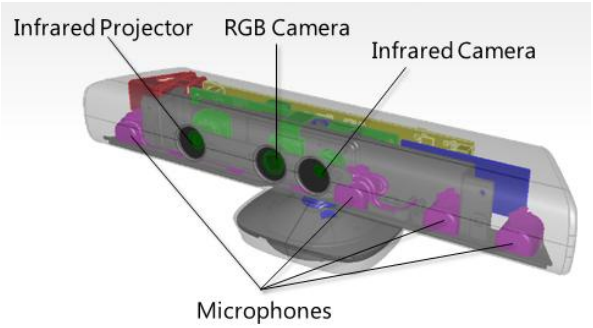

Figure 2.1.2. Kinect Sensing Components Arrangement

Source: Smisek, et al., 2011.

Secondly, the technical characters of the Kinect device are illustrated. As mentioned earlier, the Kinect contains a regular RGB camera and an active-sensing depth camera. Both of the two cameras transmit images with resolution of 640*480 pixels, 32-bit colour at 30 frames per second (Leyvand, et al., 2011). More specifically, from the figure 2.1.3, it can be implied that Kinect is definitely not a single piece device. Besides the three sensing components, quite many components and techniques are integrated together to provide the device with well adaptable functionality, such as the accelerometer, microphones and rotating motor.

Figure 2.1.3 Completely Disassembled Kinect Device

Source: Leyvand, et al., 2011.

2.1.3 Working Process

After introducing the hardware and techniques features, the brief working process of Kinect will be illustrated with diagrams in this subsection. As mentioned in the last subsection, the depth sensor of Kinect device contains a depth image CMOS sensor and an infrared laser projector. It is acknowledged from the Frati and Prattichizzo (2011) that Kinect can be described as a "3D infrared structured light scanner". More specifically, the working process of Kinect device is described is Figure 2.1.4 (Leyvand, et al., 2011).

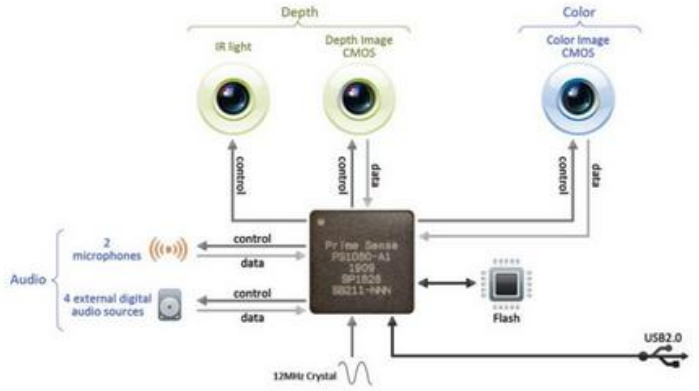

Figure 2.1.4. Kinect Functioning Process Diagram (PrimeSense)

Source: Leyvand, et al., 2011

According to the reference platforms of PrimeSense, the process that how a Kinect work can be illustrated as follows:

- To begin with, the infrared structure light is released from the infrared transmitter. It should be mentioned that the information of a speckle light patterns is encoded within the structure light.
- Secondly, from Figure 2.1.5, it can be implied that when the infrared camera catches the returned signal, the sensor will verify the encoded pattern information and associate it with a reference pattern of a known depth.
- Subsequently, by making use of encoded information, Kinect describes the detected objects and people in the detection field after altering and calculating their depth data.
- Finally, it should be clear that the working processes of the Kinect depth sensing component described above are the mandatory processes. In other words, the RGB camera and the microphones are optional based on different applications.

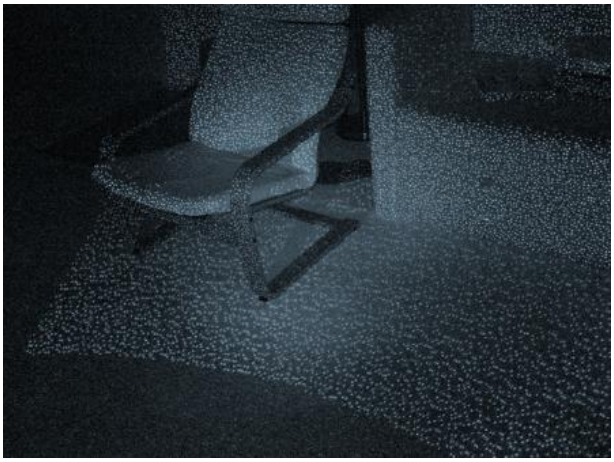

Figure 2.1.5. Kinect Infrared Light Pattern For Depth Detection

Source: (Frati and Prattichizzo, 2011).

2.1.4 Calibration and Geometrical Analysis

Before moving on to the calibration strategy, the reasons why the Kinect device is supposed to be calibrated should be interpreted. Frati and Prattichizzo (2011) highlight the necessity of the calibration; the reasons can be specified as follows:

- Initially, the intrinsic and extrinsic parameters of the Kinect camera are factory default settings without calibration.

More exactly, the intrinsic and extrinsic parameters of the device are crucial factors, when a position of a pixel needs to be calculated and translated from the camera's frame to global frame. However, without calibration the position of the special pixel can' be calculated properly. In other words, the calibration process has great influence on locate the position of pixels.

- Subsequently, besides the influence of the pixels position, the calibration can affect the combination between depth map and colour images of a particular object.

That is to say, without calibration, both the infrared camera and colour camera provide us with useless depth and RGB information. Especially, during the reconstruction process of my project, the depth map of a specific object can be connected with a set of known RGB data after calibration. Hence, the calibration of the Kinect device is crucial for my project.

In the light of the calibration strategy of Smisek, et al. (2011), the Kinect sensor can be modeled as a multi-view system which consists of RGB colour camera, infrared camera and infrared projector. As can been seen from Figure 2.1.6, the geometrical model is presented.

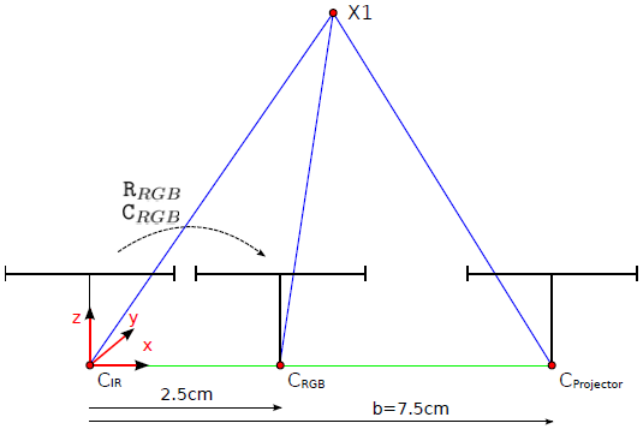

Figure 2.1.6. Kinect Geometrical Model

Source: Smisek, et al., 2011.

More exactly, the colour camera and infrared camera project a 3D point X1 into an image point $[u, v]^T$, the distortion parameters $k = [k_1, k_2, \ldots, k_5]$, camera calibration matrix K rotation R and camera centre C (ibid). The equations which demonstrate their relationships are presented as follows.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} s \\ t \\ 1 \end{bmatrix} \tag{2.1}$$

The equation 2.2 illustrates the radial distortion and the tangential distortion of the geometrical model. The actual depth is z and inverse depth is d.

$$\begin{bmatrix} s \\ t \\ 1 \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4 + k_5 r^6) \begin{bmatrix} p \\ q \\ 0 \end{bmatrix} + \begin{bmatrix} 2k_3 pq + k_4 (r^2 + 2p^2) \\ 2k_4 pq + k_3 (r^2 + 2q^2) \\ 1 \end{bmatrix} \tag{2.2}$$

$$r^2 = p^2 + q^2, \qquad \begin{bmatrix} pz \\ qz \\ z \end{bmatrix} = R(X - C) \tag{2.3}$$

The depth camera of Kinect is associated to the geometry of the infrared camera. It returns the inverse depth d which is provided above. X represents the 3D coordinates of a 3D point. a_1 and a_2 are defined as parameters of the Kinect geometrical model. Meanwhile, from equation 2.2 and 2.3, the R_{IR} and C_{IR} can be calculated, $R_{IR} = I$ and $C_{IR} = 0$. Hence, the equation which associates the Kinect coordinate system with the infrared camera is presented as follows.

$$X_{IR} = \frac{1}{a_2 d + a_1} dis^{-1}(K_{IR}^{-1} \begin{bmatrix} x + u_0 \\ y + v_0 \\ 1 \end{bmatrix}, k_{IR}) \qquad (2.4)$$

From the equation 2.4, the dis represents the radial and the tangential distortion which are given from 2.3. X_{IR} represents the depth images. If K_{RGB} is defined as the calibration matrix; R_{RGB} and C_{RGB} are defined as the rotation matrix and center of the colour camera. The equation 2.4 can be projected to the colour (RGB) images as:

$$u_{RGB} = K_{RGB} dis(R_{RGB}(X_{IR} - C_{RGB}), k_{RGB}) \qquad (2.5)$$

Apparently, the Kinect camera can be calibrated by displaying the same calibration target to the infrared camera and colour camera. After providing the matrix equations of the infrared (IR) camera calibration and colour (RGB) camera calibration, the process of Kinect calibration can be briefly accomplished. In addition, the projection equation between colour images and infrared images is also specified which might be used to verify the validity of the calibration. It should be clear that the detailed experimental calibration process will be demonstrated in the Method Chapter (Chapter III).

2.1.5 Kinect Frameworks

In this subsection several popular Kinect frameworks, such as Kinect for Windows SDK, OpenNI and NITE, are introduced briefly.

To begin with, the official driver for Kinect device: Kinect for Windows Software Development Kit (SDK) provided by Microsoft Research is used as a beginning programming toolkit for developers and researchers (Figure 2.1.7). More exactly, The Kinect for Windows SDK contains the drivers for utilizing the Kinect sensor (the driver of IR camera, the driver of RGB camera and the motor driver) on Windows-based computers.

Figure 2.1.7. Kinect for Windows SDK

Source: Microsoft Kinect for Windows Official Website, 2012.

In addition, the SDK still contains the sensor interfaces which include some pieces of source code samples (Arici, 2012). In the light of the words from Arici (ibid), there are three major functions of the Kinect for Windows SDK:

- First, this driver provides developer with the raw image and audio data captured by the Kinect device from the infrared camera, the colour camera and the microphones.
- The next feature is the SDK enables developers and researchers with the functionality of skeleton tracking.
- The third application for the SDK is advanced audio captured function, which means that this driver can provide user with the "sophisticated acoustic noise suppression, echo cancellation and identified beam formation" (ibid).

Secondly, another popular framework for Kinect device is OpenNI which is short for Open Natural Interaction (Figure 2.1.8). According to the definition from the OpenNI official website, OpenNI can be defined as an "industry-led, not-for-profit" organization which is focusing on verifying and improving the capability and functionality of natural interaction devices and applications. On the other hand, OpenNI also can be defined as a "multi-language and cross-platform" framework which is based on human senses such as hearing and vision (Arici, 2012).

Figure 2.1.8. OpenNI Logo

Source: OpenNI Official Website, 2012.

Basically, the functionality and features of OpenNI framework can be divided into three elements: hand gesture, body tracking and voice recognition. In terms of the hand tracking element, OpenNI provides user with the function of pre-defining hand gesture for recognized, interacting with machines and controlling devices by utilizing user's bare hands (Rogge, et al., 2012). As mentioned previously, OpenNI supplies developers with a bunch of APIs to be implemented by the sensor devices and middleware components. More exactly, referring to the Figure 2.1.9, OpenNI driver architecture for Kinect is divided into three components:

- The top component describes the software containing the OpenNI applications.

- The middle component demonstrates the OpenNI APIs supplying interfaces, methods and solutions with both the sensor devices and middleware components.
- The bottom component presents the hardware devices that are utilized to capture colour images, depth map and observing the environment (ibid).

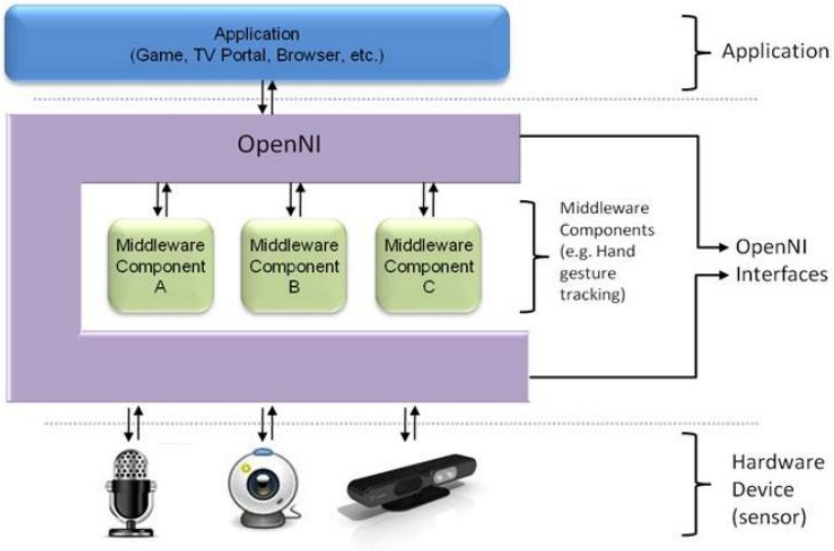

Figure 2.1.9. OpenNI Architecture for Kinect Schema

Source: OpenNI Official User Guide, 2012.

The third Kinect framework presented in the last part of this subsection is Natural Interaction Technology for End-user (NITE). NITE is the middleware supplied by the PrimeSense Company. Generally, NITE is primarily employed to represent the environment as 3D reconstruction using depth map captured by Kinect Sensor (Arici, 2012). In addition, the NITE framework contains the functionality of recognizing hand gesture, which tracks hands in the foreground and extract them from the background. According to the PrimeSense NITE Programmer's Guide, NITE cooperates with OpenNI over applications based on OpenNI.

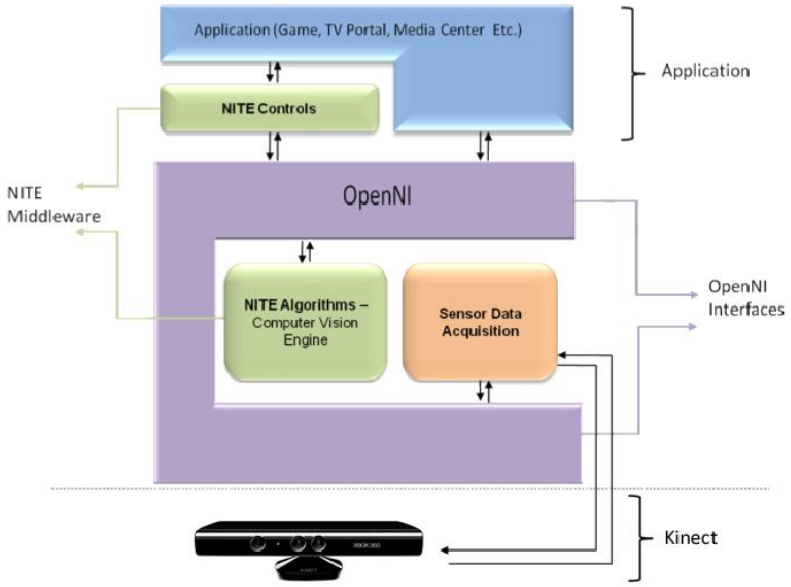

Figure 2.1.10. NITE Architecture Overview

Source: Prime Sensor NITE Programmer's Guide, 2012.

Referring to the Figure 2.1.10, the overview architecture of NITE is illustrated. The structure of NITE framework comprises four diverse components (Arici, 2012).

- The first one is the application component which is based on the NITE control algorithms and methods.
- Secondly, as can be seen from the diagram, the next one is NITE control component. This component enables applications to track and recognize gesture by supplying the NITE user-interface.
- Subsequently, the third component is NITE algorithms, which are employed to reconstruct objects, detect body motion and track hand gesture, by capturing and transforming the depth maps.
- The last component at the bottom of the diagram is NITE infrastructure. It is applied to provide drives to the sensing devices, collect user information and transfer depth maps to other components (ibid).

2.1.6 Concise Summary

In this subsection, the general information of Kinect which is the primary sensing device of my project is provided. First, the hardware arrangement and technical features of Kinect device is introduced. Afterwards, the working procedure of Kinect has been illustrated. Furthermore, the reasons why Kinect should be calibrated before making use of it have been explained, the strategy of calibration has been specified as well. Subsequently, several popular Kinect frameworks are demonstrated.

As mentioned previously, Kinect is not just a novel game controller. It can be potentially applied to robotics, computer vision, human-computer interaction and etc. It is the reason why Kinect is used for my project as the primary input device. On the other hand, by making comparison of the functionality and features between OpenNI framework and Kinect for Windows SDK, OpenNI is employed in my project as the Kinect driver. The reason can be divided into three aspects:

- First and foremost, OpenNI framework provides interfaces and methods of hand tracking and hand gesture recognition, which is one of the most crucial elements of this project.
- Secondly, the OpenNI framework can synchronize the depth maps with the colour images automatically (Rogge, et al., 2012).
- Last but not least, besides it supports multiple sensing devices, OpenNI consumes less hardware resources than Kinect for Windows SDK during the object reconstruction process.

Hence, the OpenNI framework has been employed in this project based on its functionality and advantages. It should be clear that the calibration process will be illustrated in detail in the Method chapter.

2.2 Object Reconstruction

2.2.1 Introduction

Three-dimensional models are demanded and employed in many fields such as animation, game industries, film productions, medicine, virtual reality and etc. In terms of the 3D reconstruction, it can be defined as a process which extracts the 3D geometrical data from stereo images and creates realistic 3D models (Steinbach, et al., 2000) (Figure2.2.1).

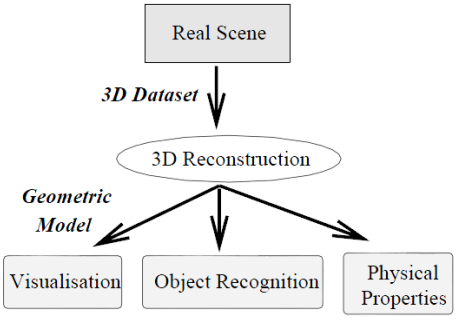

Figure 2.2.1. Conventional Stereo Reconstruction Process

Source: Faugeras, 2001.

Obviously, three-dimensional object reconstruction techniques play a crucial role in this project. Hence, in this subsection different reconstruction methods are presented: both the conventional stereo reconstruction algorithms and KinectFusion technique employed in my project. By making comparison with the traditional reconstruction algorithms, the advantages of KinectFusion algorithm are described. In other words, the reason why KinectFusion is applied in this project will be specified. Additionally, in the last part of this subsection, the evaluation approach will be presented, as well as the detailed relationship between KinectFusion and this project.

2.2.2 Surface-based Method

As can been seen from Figure 2.2.2, the first typical and traditional object reconstruction approach is conducted based on the images of object captured from different view directions using calibrated cameras. Duckworth and Roberts (2011) highlight that reconstruction algorithm is generally carried out by means of feature extraction from multiple views.

From the figure (Figure 2.2.2), it can be implied that the reconstruction process can be divided into three steps (ibid).

- First, images of objects are captured from different view directions.
- Secondly, the data of images are transmitted to reconstruction module.
- Finally, the reconstruction algorithm is used to build the 3D model.

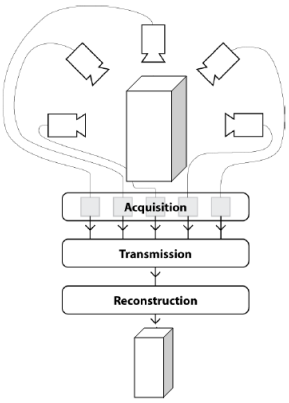

Figure 2.2.2. Typical Surface-based Reconstruction Process

Source: Duckworth and Roberts, 2011.

There are many depth extraction systems based on the first kind of approaches employed in computer vision. However, due to the "sophisticated nature of topological connectivity" of the structure of an object, it is apparent that rebuilding the entire model of an object by just utilizing all the images captured from different view is an impossible task (Walck and Drouin, 2009).

On the other hand, this kind of algorithm can be only applied to rebuild models of static objects efficiently. In other words, in terms of the cost and efficiency of the modeling system, the surface-based algorithm can't be applied to model dynamic objects.

2.2.3 Volume-based Method

Another kind of approach to reconstruct 3D models is computing consistent volume models directly in 3D space. By comparing with the first method, volume-based algorithm does not require a great number of images captured by calibrated cameras, which decrease the reconstruction time. It is acknowledged from Walck and Drouin that embedding the object in a bounded area is the specific feature of the volume-based method (ibid). The bounding area is divided into regular small grids called voxels. The voxel is determined according to its consistency to the object.

Referring to the Figure 2.2.3, the general process of the volume-based method can be described as follows:

- First, images of object are inputted into the system.
- Secondly, the object is divided into a number of voxels.
- Subsequently, the images are normalized according to different algorithm, and the geometry information of the voxels is collected.
- Finally, the 3D model is acquired referring to the relative position between voxels and the geometry information of the voxels from the original images.

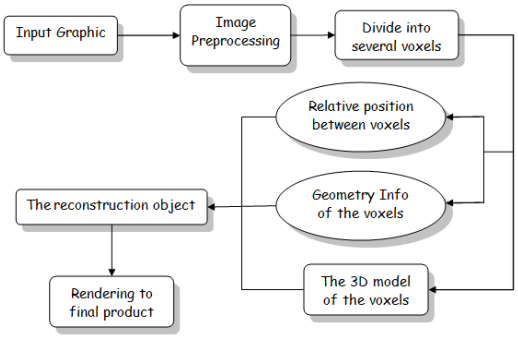

Figure 2.2.3. Traditional Volume-based Reconstruction Process

Source: Sun, et al., 2008.

In comparison with the first surface-based method, the volume-based approach is good at modeling dynamic objects. However, the reconstructing speed of the volume-based method when processing dense graphs is slow. More importantly, the result of the reconstruction related to the algorithm of determining the shape and density of the voxels (Walck and Drouin, 2009).

2.2.4 KinectFusion

In the last subsection, several conventional 3D object reconstruction algorithms have been presented and interpreted. Unfortunately, these algorithms can't be employed to build reconstructions for objects in my project. It should be recognized that the reconstruction algorithm which can satisfy the requirement of my project must have the ability to create high-quality and geometrically accurate 3D model without consuming too much time. Hence, the effective object reconstruction algorithm KinectFusion (Izadi, et al., 2011), which is utilized in my project is presented and illustrate in this part.

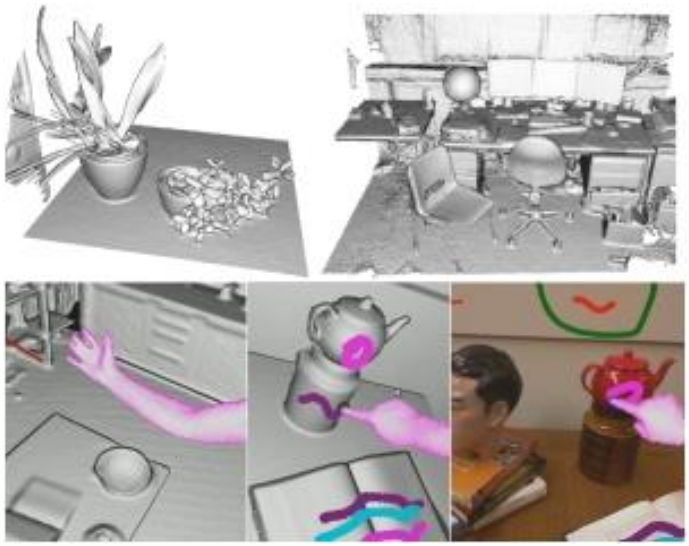

Figure 2.2.4. KinectFusion Functionality Overview

Source: Microsoft Research 3D Surface Reconstruction Website, 2012.

Previously, the functionality, hardware arrangement and calibration strategy of Kinect have been described. According to Izadi, et al. (2011), a novel interactive reconstruction system named KinectFusion is constructed, by making use of the specific feature that Kinect uses a structured light technique to generate real-time depth images containing discrete rage measurements of the physical scene (Figure2.14). Izadi, et al. (ibid) highlight that their system extracts depth data form a Kinect camera and creates a high-quality, geometrically accurate 3D reconstruction.

Basically, according to the requirement of my project, the KinectFusion pipeline is divided into four components, as can be seen from Figure 2.2.5.

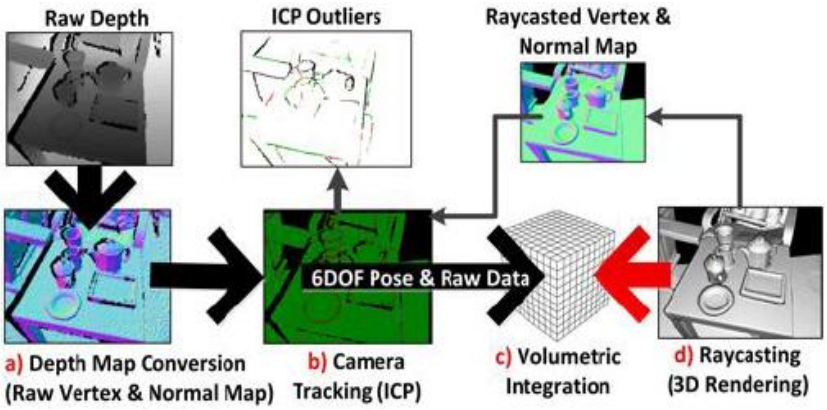

Figure 2.2.5. KinectFusion Components and Pipeline

Source: Izadi, et al., 2011.

(1) Depth Image Conversion

The first stage of the KinectFusion pipeline is depth image conversion. The depth map is converted from image coordinates into 3D points and normals in the coordinate space of the Kinect camera. The detailed process can be described as follow. At time t, pixel u = (x, y) in depth map $D_t(u)$. The calibration matrix for Kinect infrared camera is defined as K_{IR}, according to equation 2.4. Hence, the vertex map V_t can be computed in equation 2.6.

$$V_t = D_t(u)K_{IR}^{-1}[u,1] \qquad (2.6)$$

On the other hand, the normal vectors N_t for each vertex can be computed by making use of the method of neighboring projected points. The equation used to compute N_t is demonstrated in 2.7

$$N_t(u) = (V_t(x+1,y) - V_t(x,y)) * (V_t(x,y+1) - V_t(x,y)) \qquad (2.7)$$

Afterwards, a transform matrix T_t, rotation matrix R_t and translation vector W_t are defined. Hence, the vertexes map V_t and the normal vectors N_t are converted into global coordinates as follows.

$$V_t^g = T_t V_t(u) \quad N_t^g = R_t N_t(u) \qquad (2.8)$$

(2) Iterative Closest Point

Secondly, the next step of the pipeline is tracking the Kinect camera by making use of the Iterative Closest Point (ICP) algorithm. In this stage, a six degree of freedom (DOF) transform is computed to align the oriented points. Moreover, the global position of the Kinect is defined in this stage. The output of each ICP loop is a single transformation matrix S defined as the sum of

squared distance between each point in current frame and its corresponding point in the previous frame, the equation of matrix can be seen in 2.9.

$$\sum_{\substack{u \\ D_t(u)>0}} ((SV_t(u) - V_{t-1}^g(u)) * N_{t-1}^g(u))^2 \qquad (2.9)$$

Izadi, et al. (2011) emphasize that one of the most significant contribution of the KinectFusion is that Iterative Closest Point is conducted in all measurements provided in each Kinect depth image. This kind of camera tracking plays a primary role in model reconstruction and user interaction.

(3) Representation and Integration

The KinectFusion algorithm employs a volumetric surface representation base on volume-based algorithm (Walck and Drouin, 2009). In this stage, a 3D volume of voxels grid is defined. Afterwards, global 3D vertices are integrated into the voxels grid. Importantly, each voxel has a parameter which indicates the average distance between itself and the assumed position of a physical surface. More exactly, the relationship between global coordinates and voxel coordinates are described at the end of this stage.

(4) Rendering

The last step of the pipeline is rendering by employing a GPU-based ray casting method. The volume of the reconstruction is raycasted to extract views of the implicit surface. The raycasted view of the volume is equal to an artificial depth map rather than the current live depth map. The artificial map is less noisy which can be used to be rendering to user.

2.2.5 Comparison and Evaluation

(1)Comparison and Advantages

By making comparison with the conventional surface reconstruction methods, this paper presents the advantages of KinectFusion. Generally, the advantages of the algorithm can be separated into three factors. These factors which help KinectFusion algorithm excel in the area of 3D object reconstruction are displayed as follows:

- High quality 3D object reconstruction.

Obviously, the first factor is high quality and geometrically accurate 3D model. One of the objectives in my project is capture detailed and dense 3D models of the real scene (Figure 2.2.6). KinectFusion algorithm is able to produce more accurately approximate real geometry.

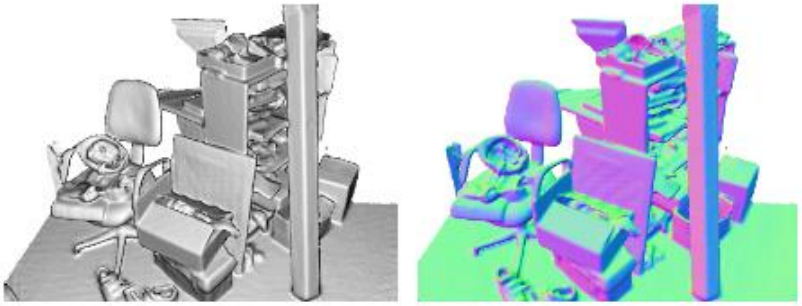

Figure 2.2.6. Object Reconstruction by KinectFusion

Source: Izadi, et al., 2011.

- Interact Immediately.

The second factor is that KinectFusion can achieve real-time interactive rates for both camera tracking and 3D reconstruction. Referring to the Figure 2.2.7, this specific feature can be used to achieve interaction (such as segmentation) between user and the 3D models created by the KinectFusion system.

Figure 2.2.7. Fast and Direct Object Interaction

Source: Newcombe, et al., 2011.

- Reconstruction Scale.

KinectFusion is able to produce the entire room reconstruction and interaction, rather than focusing on small physical objects (Figure 2.2.8). It should be clear that, although this project focuses on models within a small desktop area, due to the limitation of the hardware configuration, this specific feature of KinectFusion algorithm can be applied in the future work.

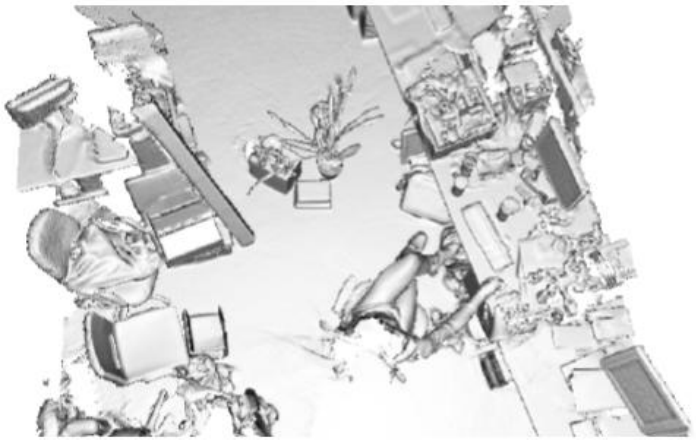

Figure 2.2.8. Large Scale Reconstruction

Source: Izadi, et al., 2011.

(2)Evaluation Process

Secondly, the criteria of evaluation to object reconstructions are specified. It is acknowledged from Steinbach, et al. (2000) that the quality of reconstructed 3D models can be evaluated based on three main criteria:

- Reconstruction distortion
- Reconstruction rate
- Reconstruction time

The function of reconstruction distortion is to determine the difference between a reconstructed model and the corresponding real-world 3D object. On the other hand, the reconstruction rate is used to estimate how solid the representation of a reconstruction model is. The reconstruction time is used to determine the reconstruction speed.

2.2.6 Brief Summary

This subsection introduces different object reconstruction techniques: conventional stereo reconstruction methods and KinectFusion. By making comparison between KinectFusion and other reconstruction algorithms, the capability and advantages of this technique are illustrated and interpreted. Previously, it has been mentioned that the KinectFusion algorithm is used as the object reconstruction algorithm for the project. Hence, the reason why it is employed in my project is explained in detail.

Furthermore, the precise relationship between my project and the KinectFusion need to be clarified. Although the unique features of KinectFusion make it excel than other reconstruction methods, it is not an open source algorithm. It should be clear that this project just takes

advantage of the strategy of KinectFusion algorithm without making use of any pieces of its source code. In terms of the implemental process, Point Cloud Library (PCL) has been employed to supply libraries and functions for my project. In other words, the object reconstruction experiment was conducted based on the PCL and the KinectFusion strategy. Another crucial element illustrated in this subsection is the evaluation parameters of the reconstruction, which supplies approaches to analyze and assess the algorithm in the following chapters.

2.3 Gesture Tracking

2.3.1 Components and Challenges

(1) Hand Gesture and Components

Zabulis, et al. state that most of the hand gesture interactive system can be separated into three components (Figure 2.3.1): object detection, gesture tracking and gesture recognition (2009).

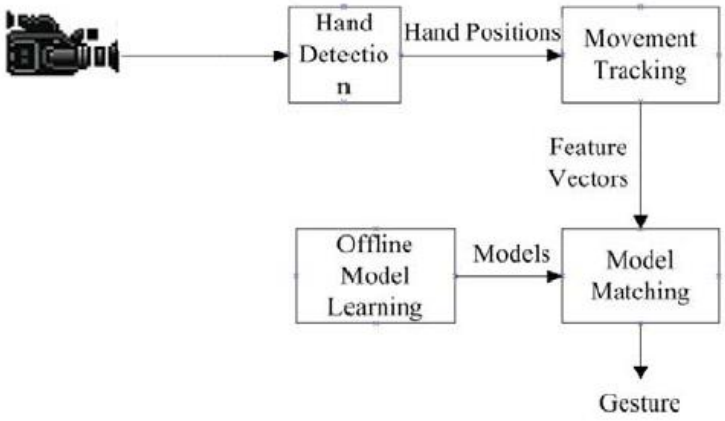

Figure 2.3.1. Traditional Gesture Tracking System

Source: Zhu and Pun, 2010.

- Object Detection.

In terms of the object detection, it involves detecting target objects in the scene. Basically, most of the detected methods are conducted based on the knowledge of the specific features of objects. The features of hands can be the colour of skin and the shape of hands (Nguyen, et al., 2010).

- Gesture Tracking.

Gesture tracking can be described as a process of observing and tracking the movements of objects. It should be recognized that the tracking process does

not just follow the motion of objects but also provide user with information of certain parameters model at a certain movement (Noonan, et al., 2011).

- Gesture Recognition.

Gesture recognition plays a significant role in the hand gesture interactive process. In the stage of the process, information about hand features is extracted depending on specific requirement such as the shape of the hand, the shape of fingertip and hand posture.

(2) Challenges and Drawbacks

Besides these internal problems, difficulties emerge from external factors such as the complexity searching for optimal motion estimation in a high dimensional configuration space and representing high data volume motion restrict their opportunities to be utilized. The drawbacks can be divided into two aspects.

- First, these vision-based hand tracking systems can't be employed to cooperate with a geometrically accurate, real-time and robust 3D reconstruction system used in my project to produce the 3D models.
- Subsequently, these algorithms are not able to work if there are cluttered backgrounds. More exactly, even the alteration of lighting condition will has strong influence on these systems.

Referring to the drawbacks and disadvantages of the traditional hand tracking algorithms, obviously, they can't satisfy the requirement of my project. Hence, a novel 3D gesture algorithm is demanded to accomplish the hand tracking and recognition tasks for my project.

2.3.2 Simultaneous 3D gestures

Considering all the mentioned challenges and drawback in precious works and implementations, it is not a sample endeavour to create a 3D gesture tracking system. Evidence to support this comes from both the complexity of capturing human movements in 3D and the difficulties associated with recognizing gestures from human motion data (Zhu and Pun, 2010).

In this paper, a novel practical and effective technique for gesture tracking and recognition is proposed (Bigdelou, et al., 2012). The specific features, functionality as well as the main contributions of the simultaneous 3D gestures algorithm compared to conventional hand gesture systems are demonstrated as follows.

- First and foremost, gestures can be tracked and recognized for both categorical control and spatiotemporal control (Chavez and Kyaw, 2011).

In other words, this method can not only provide user with approaches to recognize and track gestures but also supply feedback and interaction to gestures data.

- Subsequently, natural gestures are able to be tracked and recognized in the system.

Basically, user can make use of their natural behaviors to interact with the system without pre-defining them. More precisely, natural gestures can be used to segment reconstruction models in this project.

It should be mentioned that there are two primary parameters of the simultaneous 3D gestures system. The first one is gesture state used to provide the system with the information that whether a gesture is recognized and tracked or not. The second parameter is gesture position applied in the system to activate, update and control the segmentation process(Chavez and Kyaw, 2011).

Referring to the Figure 2.3.2, besides the depth map and colour image capture, the algorithm can be divided into three components: feature extraction, gesture learning and gesture tracking.

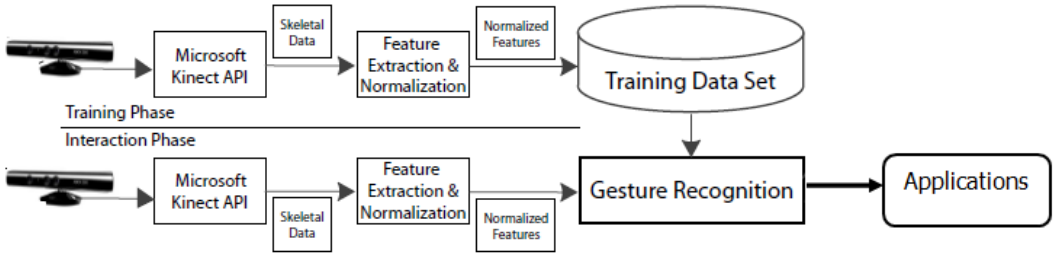

Figure 2.3.2. Simultaneous 3D Gestures Algorithm

Source: Bigdelou, et al., 2012.

(1) Feature Capture

The first component in the procedure is feature extraction. The input sensor of the system is Kinect camera. Leyvand, et al. demonstrated that a typical human body reconstruction produced by Kinect is made up with a number of bones connected through joints (2011). Moreover, each joint has a unique identifier. Due to this specific structure of reconstruction model, the joints are the major feature extraction objects. The maximum number of joint is 20. At each time t, the equation of representation of each can be defined as follow:

$$J_i^t = [x, y, z]^T \qquad i \in \{1, \ldots, 20\} \qquad (2.11)$$

The extracting feature vectors V_t are defined as the distance between each joint and the center of the human body. More specifically, the spine point is represented by J^t_{spine} is an estimation for the centre of the skeletal data. Thus, the distance-based feature extracting vector at time t is defined in equation 2.22.

$$V_t = \left[dis_1^t, \ldots, dis_{20}^t \right] \qquad dis_i^t = \| j_i^t - j_{spine}^t \| \qquad (2.12)$$

This feature representation is independent of the Kinect camera and user location and orientation. Additionally, each joint contains a confidence value C^t_i, which indicates the confidence of the tracking system about the joint location. In order to improve the success rate of the recognition approach, feature normalization is conducted by modifying the skeletal data according to user's status.

(2) Gesture Leaning

The second step of the process is gesture learning which is conducted in the train phase. In this process, the simultaneous 3D gestures method learns prior models of gesture from sample pose feature data. The known gesture for the training data is defined as KG = {kg_1, ... , kg_n}. Moreover, a one-dimensional intermediate representation X = {x_1, ... , x_n}. Therefore, the feature extracting vectors which are defined above can be assigned with the value of the known gesture. In other words, for the stage of gesture leaning, the V can be redefined as:

$$V^{kg} = \{ v \in V \mid kg_i = kg \} \qquad (2.13)$$

According to equation 2.13, the resulting dimensional datasets X^{kg} is normalized. Afterwards, for every index i, the training dataset which is made up with a feature vector V_i, the known gesture label kg_i and the one-dimensional representation is specified.

(3) Gesture Tracking and Recognizing

The last factor of the algorithm is gesture tracking and recognizing approach, which is the essential component of the procedure. Apparently, the objective of process is to determine new gestures. After the training process, the feature vectors V, at any time t are presented. In order to identify new performed gestures by taking advantages of the feature vectors, an equation which projects arbitrary feature vectors to the dimensionality-reduced representation X is proposed as follow:

$$\hat{x}_i = f(V_i) = \sum_{i=1}^{n} \frac{w_i(V_i)}{\sum_{j=1}^{n} w_j(V_i)} * x_i \qquad (2.14)$$

In equation 2.14, the estimate $\hat{x}_t$ is calculated from the projection $x_i \in X$ of the feature vectors $V_i \in V$ in the training stage. Moreover, the weight parameters are defined by a Gaussian kernel $w_i(V_t)$ with a width derived from the training features V. By making use of the $w_i(V_t)$, the current gesture index $\hat{kg}_t$ can be determined:

$$\hat{kg}_t = kg_{a_t} \qquad a_t = \arg\max w_i(V_t) \qquad (2.15)$$

Consequently, the current gesture index has been determined, according to the equation above. In the equation, a_t identifies the feature vector in the train data which is most similar to V_t, and assigned largest weight if it is similar to V_t. The reason why a_t is used is to make sure the gestures which are determined by the system are accurate.

2.3.4 Evaluation Parameters

There are two factors involved in the evaluation method: measurements and explanation. Measurements are used to provide evidence to prove the algorithm is more efficient and suitable than other approaches to be employed in my project to accomplish certain tasks. On the other hand, explanation is used to interpret the results of the evaluation.

There are two measurable elements for the simultaneous 3D gestures algorithm: recognition rate and recognition quality.

- The recognition rate is determined at each time t by the value $\hat{kg}_t \in \{1,...,N\}$ and generated by the gesture recognition component.

In order to evaluate the performance of this task, the accurate gestures recognition time should be measured. More precisely, the implementation of this measurement should be designed as follow: first, a set of training data will be inputted into the system; secondly, the testing step will be conducted, according to the known gestures. Thirdly, the time of the recognized gestures can be calculated.

- The second measurable factor is recognition quality given by the number $\hat{x}_t \in [0,1]$.

This value $\hat{x}_t$ is employed to translate arbitrary gesture movements of user to parameters of the system (ibid). In order to evaluate the recognition quality, the correlation between the value $\hat{x}_t$ and an ideal value obtained

from the ground true data X^{kg} . The maximum correlation 1 is achieved if the change in $\hat{X}_t$ is aligned in time and match with the ground truth exactly. Referring to Figure 2.3.3, the gesture has been recognized accurately. Moreover, if there are 5 gestures captured by Kinect during the experiment, the system can only recognize 4 of them. Thus, the recognition rate of gesture is 80%.

Figure 2.3.3. Accurate Recognized Gestures

Source: Zabulis, et al., 2009.

2.3.5 Summary

Simultaneous 3D gestures algorithm, an effective and practical gesture tracking and recognition method is proposed in this subsection. The features and functionality of this technique differentiate itself from the conventional hand tracking algorithm. Evidence to support this comes from the evaluation process where two factors are specified and employed to measure the technique. More exactly, a detailed evaluation will be proposed in evaluation chapter to measure the ability of the simultaneous 3D gestures method.

2.4 Object Segmentation

2.4.1 Segmentation Algorithm

In comparison with the traditional object segmentation methods which are good at segmenting interesting objects from colour images, a segmentation technique based on Iterative Closest Point (ICP) and clustering algorithm is proposed to segment objects from depth images captured by Kinect camera. In the previous section, the Iterative Closest Point method is used to track the Kinect camera in KinectFusion.

Sometimes, a user wants to segment a specific small object from the entire scene. In order to achieve this function, the entire scene is reconstructed first, and then user makes use of gestures to determine which objects can be segmented. As can be seen from Figure 2.4.1, the first image indicates that the scene is reconstructed; the second image indicates that the hand gesture determines which objects will be segmented. Finally, the last image indicates that the object is segmented successfully. It should be clear that when the user's gesture is getting close to the particular object, the object will be conducted segmentation process. It is not necessary for the gesture to touch the object, once the gesture is gesture is getting closer to this object than others, this particular object can be segmented.

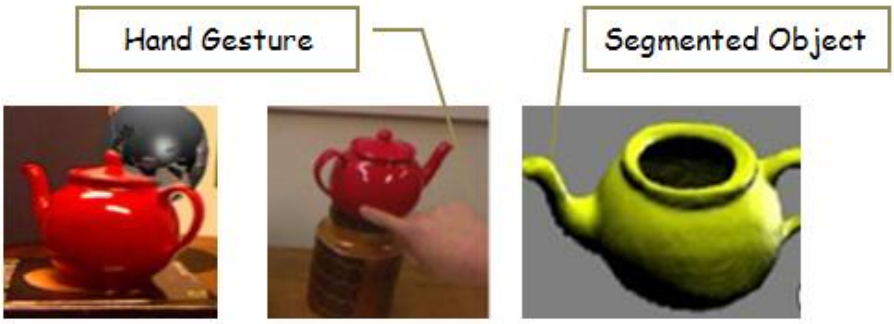

Figure 2.4.1. Object Segmentation Using Gesture

Source: Izadi, et al. 2011.

The segmentation process based on KinectFusion algorithm can be divided into four steps, which are illustrated as follows:

- To begin with, a moving object entering the scene which contains oriented points with significant disparity to the already reconstructed surfaces, after the initial scan (Figure 2.4.2).

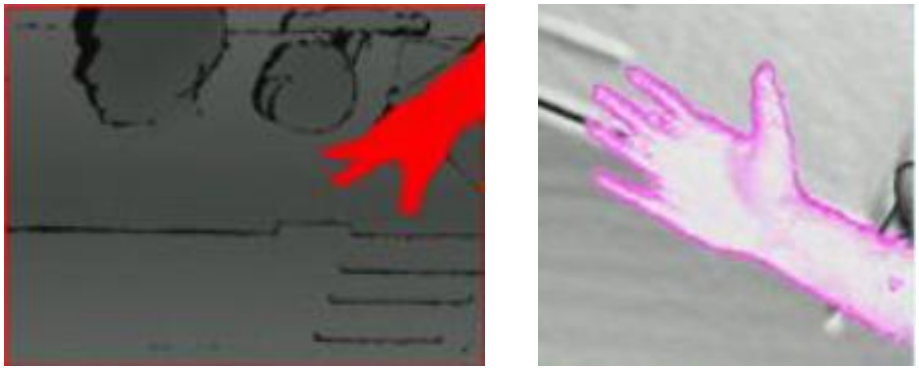

Figure 2.4.2. Hand Gesture and Reconstructed Scene

Source: Newcombe, et al., 2011.

- Secondly, the moving object is defined as the part of foreground scene, and assigned with special ICP projective data. Moreover, the motion of the gesture is detected when the gesture is getting close to an object.
- Subsequently, the input depth map is masked for background reconstruction (Figure 2.4.3). Afterwards, a depth-aware connected component analysis is performed on the depth map to cluster large connected patches and remove smaller ones due to sensor noise.

Figure 2.4.1 Segment Object Using Clustering Algorithm

Source: Izadi, et al. 2011.

- Finally, the object is subtracted from the background reconstruction according to ICP algorithm and clustering algorithm.

2.4.2 Evaluation and Brief Summary

(1) Evaluation Elements

One of the most popular methods to evaluate object segmentation is taking full advantages of the ground truth segmentation. In other words, the evaluation is conducted by comparing the segmentation results to the ground truth segmentation.

The measurable elements of the object segmentation are proposed based on method introduced by Abramov, et al. (2012). There are two elements involved in the evaluation process: segmentation covering metric and time consuming.

- Segmentation Covering Metric. The segmentation covering metric is defined as:

$$C(S' \to S) = \frac{1}{N} \sum_{R \in S} |R| * \max_{R' \in S'} O(R, R') \qquad (2.16)$$

In equation 2.16, S' is the ground truth segmentation which is a manual annotation of the results showing how human perceive the segmentation process. And S is the segmentation results conducted by the algorithm. Moreover, Nis the total number of pixel in the depth image, and |R| is the

number of pixel in region R, and 0(R, R') is the overlap between regions R and R'. The covering metric is used to evaluate the accuracy of the segmentation.

- Time Consuming. Time consuming parameter is used to calculate the total processing time (Silberman and Fergus, 2011).

The processes which have influence on the time performance include recognizing the gesture and segmenting the objects.

(2) Summary

In the subsection, the general information of the segmentation algorithm employed in this project is illustrated. In the method chapter, a more specific segmented scheme based on this method will be proposed. More importantly, during the experiment process different objects are assigned with different colour, including user's gesture. At the end of the segmentation process, the segmented object will change its colour to the same as the gesture's colour, which indicates that the object has been segmented successfully.

2.5 Background Summary

These techniques presented in second chapter are mainly involved in four types: Kinect related techniques, object reconstruction methods, hand tracking algorithms and object segmentation techniques. Referring to the relation between these techniques and my project, it can be divided into several individual aspects.

- First of all, Kinect severed as the primary input device for this project, the calibration scheme has been provided in this chapter, as well as the importance of the calibration. Moreover, OpenNI is employed as the Kinect driver, after making comparison with other drivers.
- Subsequently, the strategy of KinectFusion algorithm is applied to produce the object reconstruction. In terms of the implemental process, Point Cloud Library (PCL) has been employed to supply libraries and functions for my project.
- Thirdly, an original gesture tracking algorithm: simultaneous 3D gestures is proposed in this chapter. Object segmentation based on Iterative Closest Point (ICP) method and clustering algorithm is undertaken as well.
- Another crucial element illustrated in this chapter is the measurable evaluation parameters and processes of the reconstruction, hand tracking

and segmentation which supplies approaches to analyze and estimate the performance the algorithms in the following chapters.

The following method chapter will describe the overall solution and workflow of the project. Afterwards, the hardware requirement and programming environment configuration are also specified in this chapter. Moreover, the procedure of the software design and development are presented in THE method chapter.

III. METHODS

The method chapter is used to introduce the details of proposed solutions to the project. The designing and developing processes of the software are also presented in this chapter. Additionally, some technical and scientific problems encountered during the software development are specified and explained in this chapter. More precisely, this chapter is divided into four sections: the first section is applied to introduce the overall solution and general workflow of the software development; the second section presents the hardware requirement and configuration; subsequently, the next section describes the programming environment configuration; the final section of this chapter is applied to illustrate the software designing and developing procedure.

3.1 Overall Solution and Workflow

Referring to my project, the fundamental idea of the project is to make use of hand gesture to achieve a practical and interactive functionality between user and computer. Specifically, in order to achieve this functionality, the gesture tracking algorithm will be combined and implemented with some other algorithm such as point cloud library and object segmentation algorithm. Consequently, my project can be described as follows: by combining with segmentation algorithm, hand gesture is employed to conduct the segmentation of the 3D models which are reconstructed by KinectFusion (based on point cloud library).

In the light of the specific idea and objectives of the project, the overall solution can be described as follows.

(1) To begin with, after reviewing literatures related to techniques of object reconstruction, segmentation and gesture tracking, the requirement of the hardware is specified, as well as the software configuration.

(2) Secondly, as the prerequisite of my project, object reconstruction method is specified. According to last chapter, the reconstruction algorithm is making use of the open source point cloud library based on the strategy of KinectFusion.

(3) After implementing the reconstruction method, the novel simultaneous 3D gestures algorithm is proposed. Basically, there are two steps of this algorithm: the gesture leaning process and the gesture tracking and recognizing process.

(4) Finally, object segmentation step is proposed based on clustering technique and Iterative Closest Point (ICP) algorithm.

More precisely, the overall workflow is depicted in Figure 3.1.1. From the figure it can be implied that the details of every steps of the proposed solution are illustrated. Additionally, the major techniques employed in each stage are presented, as well as the brief advantages. According to this diagram of the overall workflow of my project, the following subsections will be described.

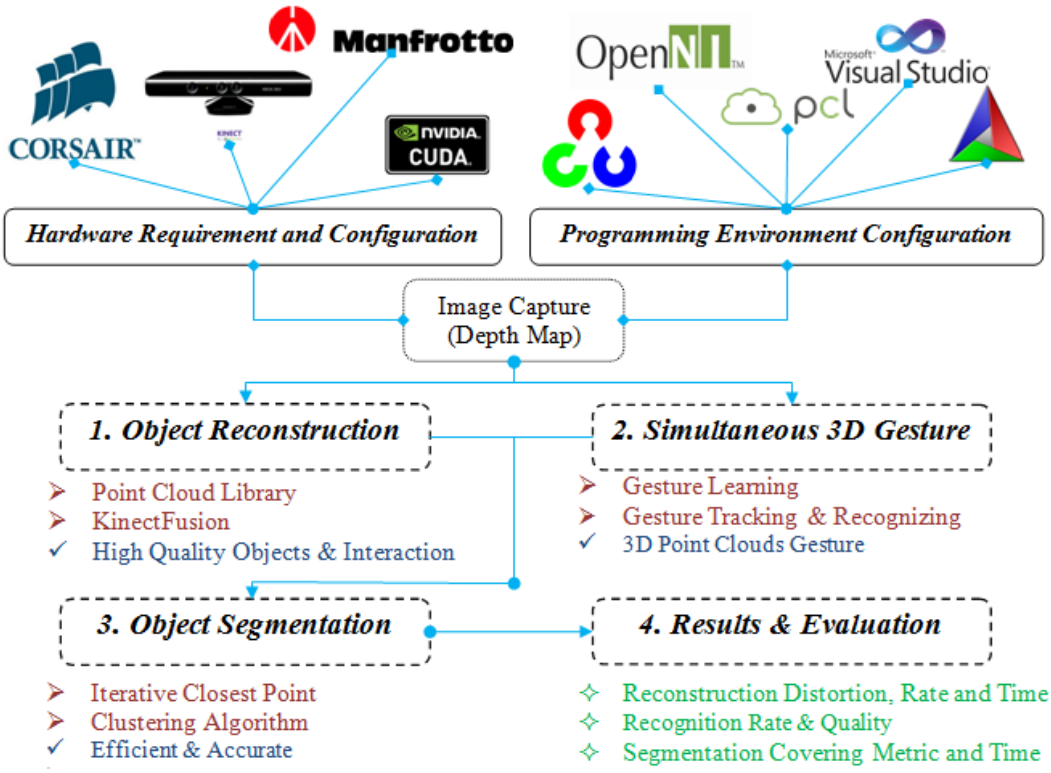

Figure 3.1.1. Proposed Solution and Overall Workflow

Source: Zhengfei Yin, 2011.

3.2 Hardware Requirement and Configuration

3.2.1 Kinect with Power Supply

According to the diagram of the overall workflow, the most significant hardware requirement of this project is the Kinect camera, which served as the primary input device. Basically, Kinect uses a structured light technique to generate real-time depth images containing discrete rage measurements of the physical scene (Arici, 2012). The Kinect section in the second chapter has introduced the technical features and the work process of this advanced sensing device. Furthermore, the calibration strategy and the geometrical analysis has been described.

In order to take full advantage of the Kinect camera, there is another significant issue should be taken into account. It is the power requirements of the camera. According to Arici, the operating Kinect requires 12-volt direct current (2012). It consumes 12 watts of power per hour during the operation. On the other hand, the power of the USB port of most computers (5 volts and 2.5 watts) can meet the requirement of the Kinect. In other words, the Kinect needs an additional transformer type power supply, can be seen from Figure 3.2.1. Moreover, from the graph it can be implied that the Kinect power supply has a specific USB adapter which can be applied to connect with the Kinect, as well as a computer.

Figure 3.2.1. Kinect Power Supply

Source: Arici, 2012.

3.2.2 Graphics Card and CUDA

Apparently, object reconstruction method is implemented based on Kinect Fusion, by taking advantage of the Point Cloud Library. It should be clear that the both the models reconstruction and object segmentation processes are based on GPU pipeline. That is to say, this project has strict requirement of the graphics card of the computer. More exactly, according to the PCL official website, the minimum compute capability of a graphics card should be more than 2.0, if the KinectFusion algorithm wanted to be implemented.

Besides the compute capability, the graphics card should be able to support CUDA. According to NVIDIA, CUDA is a parallel computing platform and programming model which enables dramatic increases in computing performance by harnessing the power of the graphics processing unit. After discussing the compute capability and CUDA-enabled ability of the graphics card with my supervisor Professor Mike Fraser, the Geforce GTX 480 is used in my project as the graphics to conduct the models reconstruction. As can be seen from the Figure 3.2.2, the compute capability of this graphics card is 2.0, and it is CUDA-enabled. After implementing and investigating the performance of it, GTX 480 is advanced enough to be applied in this project.

Figure 3.2.2. GeForce GTX 480

Source: Geforce Official Website, 2012.

3.2.3 Power Supply

Referring to the overall workflow diagram, a more powerful power supply is demanded to provide the computer with power. As can be seen from the Figure 3.2.3, Corsair AX850 power supply is utilized for the implementation. As mentioned in the last subsection, a new graphics card is employed in the implementation, and the minimum system power requirement is 600 watts. By making use of this Corsair AX850 power supply, it can provide the system with 850 watts, which is able to fulfill the power requirement completely.

Figure 3.2.3. Corsair AX850 Power Supply

Source: Corsair Official Website, 2012.

3.2.4 Kinect Tripod

The last component of the hardware configuration is a tripod which is used to fix the Kinect and adjust the orientation and distance between objects and Kinect. As can be seen from the Figure 3.2.4, there are three graphs: the first one is the tripod itself. It should be mentioned that there is no tripod which is designed specifically to the Kinect. Hence, a tripod used for Digital Single Lens Reflex is used and modified to connect and fix the Kinect, as can be seen from the second graph.

On the other hand, there are two major steps of the gesture tracking process. The first step is gesture learning and the second one is gesture tracking and recognizing. During the first step, the orientation of the Kinect is set up horizontally as shown in the second graph. Because a user is supposed to stand in front of the Kinect and predefine gestures. In other words, the Kinect will be used to capture the predefined gesture based on the entire skeleton of the user. However, in terms of the gesture tracking and recognizing process, the direction of Kinect is shown in the third graph. In this step, gesture is utilized to conduct the object segmentation. Hence, Kinect fixed in that direction can capture user's gesture and objects.

Figure 3.2.4. Tripod and Kinect

Source: Zhengfei Yin, 2012.

3.3 Programming Environment Configuration

3.3.1 OpenNI and NITE

The last chapter has illustrated functionality and features of the OpenNI framework. By comparing with other framework such as Kinect for Widows SDK, the OpenNI is employed in my project served as the Kinect driver. NITE cooperates with OpenNI over most applications. Hence, NITE is also necessary to this project. In this subsection, the process that how to install Kinect on Windows based on OpenNI and NITE is presented.

According to Rogge, et al. (2012), the installing process can be separated into four steps. Before the installing process, all of the previous drivers in the computer must be uninstalled.

- The first step is downloading and installing SensorKinect, which is the Kinect dirver presented by PrimeSense company, OpenNI binaries and NITE binaries from the website.
- The next step is connecting the Kinect device with the USB port of the computer. Obviously, the power supply of the Kinect should be plugged in to the plug seat.
- Thirdly, after a few minutes, the driver software should be detected and applied. The Kinect can be found in the device manager in the machine which indicates that it has been installed successfully (Figure 3.3.1).

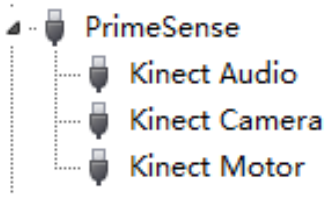

Figure 3.3.1. Kinect Driver

Source: Zhengfei Yin, 2012.

- Finally, in order to prove the OpenNI and NITE are also installed successfully, their existing demo applications can be tested (Figure 3.3.2).

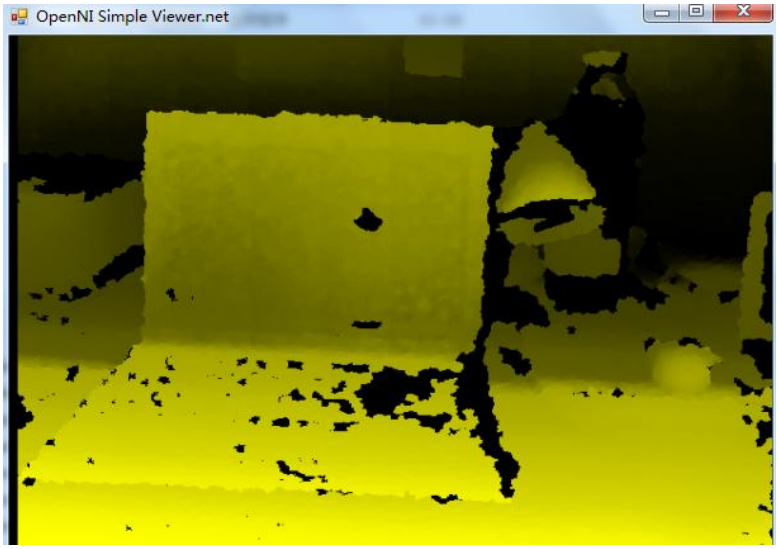

Figure 3.3.2. OpenNI Simple Viewer Demo Application

Source: Zhengfei Yin, 2012.

Currently, Kinect device can be used to capture colour images and depth maps for my project. According to the overall workflow, the programming platform utilized in this project is Windows Visual Studio. Hence, several configurations are required to transfer the depth maps from the Kinect to

Visual Studio where the depth data can be managed and manipulated to acquire specific achievement.

3.3.2 Point Cloud Library

Before configuring the Point Cloud Library source code, there are two pieces of important software (OpenCV and CMake) should be installed and configured. OpenCV (Open Source Computer Vision) is a library of programming functions for real time computer vision (Noonan, et al., 2012). There are more than 2000 functions dedicated to various computer vision problems, such as camera calibration, image segmentation, gesture tracking and etc. OpenCV employed in this project to help calibrate the Kinect, track and recognize hand gesture. The second software employed in this project is CMake, a cross-platform, open-source build system. CMake is used to control the software compilation process using simple platform and compiler independent configuration files (Faugeras, 2001).

The Point Cloud Library (Figure 3.3.3) is an open source library for the point cloud processing (Rusu and Cousins, 2011). It is free for research use. In the light of the words from Rusu and Cousins, PCL has been complied successfully on several operating systems (2011). In terms of the data acquisition, PCL provides the "OpenNI wrappers API", which enables many popular data capturing devices such as the Microsoft Kinect (ibid). Furthermore, PCL is integrated with the Visualization Toolkit (VTK) to achieve the visualization library. In this paper, the library is used to visualize the point data for the reconstructed surfaces.

Figure 3.3.3. The Point Cloud Library Logo

Source: Rusu and Cousins, 2011.

As indicated previously, this project just takes advantage of the strategy of KinectFusion algorithm. In terms of the implemental process, PCL has been employed to supply libraries and functions for my project, since the PCL provides all required functionality from the data acquisition method to the object reconstruction. More specifically, compiling process of the PCL contains three steps:

- To begin with, the prebuilt dependencies should be downloaded and installed to the default locations, which will make configuring more convenient. The prebuilt dependencies include Boost, Eigen, FLANN, VTK and Qt.
- Subsequently, the next step is downloading the PCL source code by making use SVN trunk.
- The final step is configuring the PCL by making use the CMake-gui (Figure 3.3.4). Windows Visual Studio 2010 is used as the generator during this configuring and generating process. Afterwards, the PCL can be build and installed using Visual Studio.

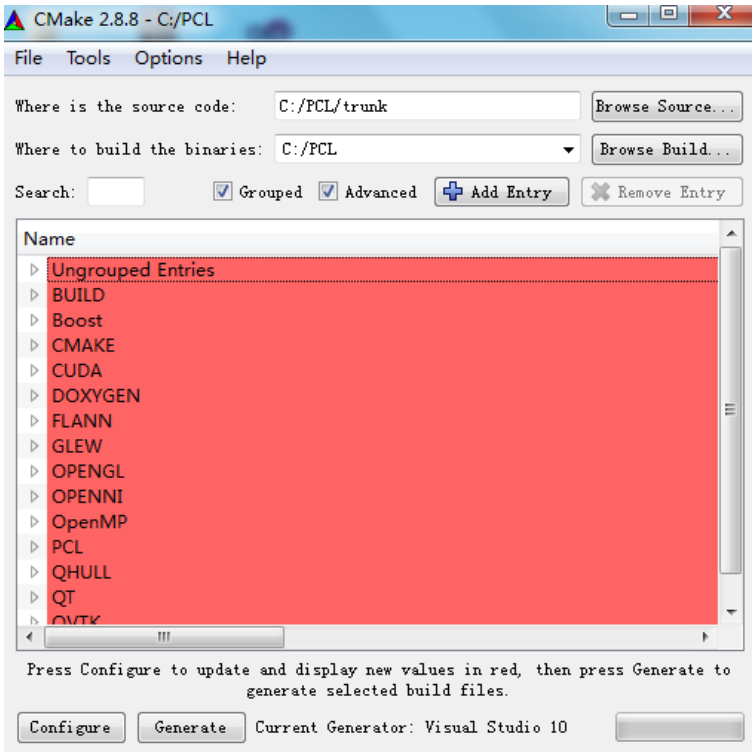

Figure 3.3.4. Configure and Generate PCL Source Code Successfully

Source: Zhengfei Yin, 2012.

3.4 Software Design and Development

3.4.1 Calibrating the Kinect

In the last sections, the hardware requirements has been described and provided. Afterwards, programming environment configuration has been investigated and built. Hence, the software is ready to be developed step by step based on the overall design workflow.

The first step of the development process is calibrating the Kinect device. Since the calibration procedure has been proposed in the second chapter, the implementation of it can be conducted more straightforward.

(1) Calibrating Target

To begin with, a black and white chessboard image is used as the calibrating target to calibrate the Kinect. After printing it out and taping it a piece of cardboard, the square size is measured (about 24mm), which will be used as one important calibrating parameter.

(2) Calibrating and Recording

By making use RGB-Demo which is a piece of popular calibrating open source software, the calibrating process is conducted (Figure 3.4.1). From the graph, it demonstrates the position and distance between the chessboard and the Kinect device, the corners of the target should always appear in the image.

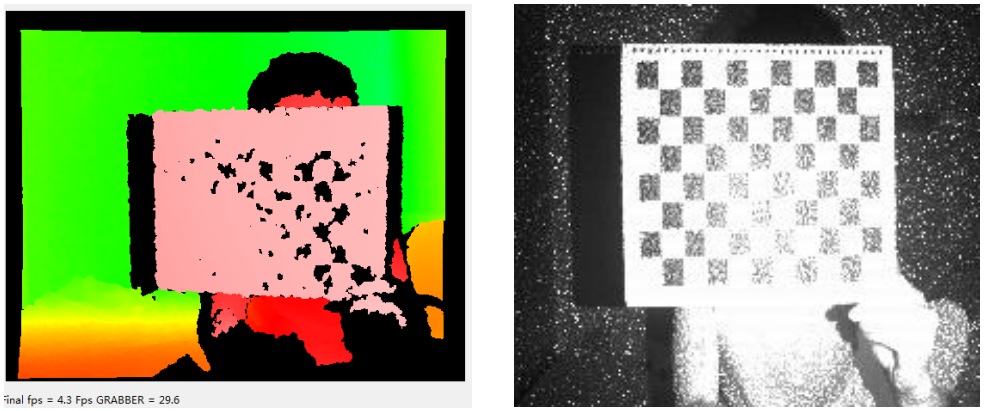

Figure 3.4.1. Chessboard and Calibrating Position

Source: Zhengfei Yin, 2012.

More importantly, the RGB-Demo software can detect and record the distance between chessboard and Kinect device. After moving the target around, the software will generate a file which contains calibration information (Figure 3.4.2). The rgb_intrinsics and depth_intrinsics are the intrinsic parameters of Kinect. According to Smisek, et al. (2011), Kinect is calibrated using pinhole model. Namely, 3D points are projected onto the image plane with perspective transformation. By calculating and analyzing the data, the results are used to determine whether the calibration is finished and successful.

```
rgb_intrinsics: !!opencv-matrix
   rows: 3
   cols: 3
   dt: d
   data: [ 5.1849264445794347e+02, 0., 3.3438790034141709e+02, 0.,
       5.1589335524184094e+02, 2.5364152041171963e+02, 0., 0., 1. ]
rgb_distortion: !!opencv-matrix
   rows: 1
   cols: 5
   dt: d
   data: [ 2.4542340694293793e-01, -8.4327732173133640e-01,
       -1.8970692121976125e-03, 5.5458456701874270e-03,
       9.7254412755435449e-01 ]
depth_intrinsics: !!opencv-matrix
   rows: 3
   cols: 3
   dt: d
   data: [ 5.8089378818378600e+02, 0., 3.1345158291347678e+02, 0.,
       5.7926607093646408e+02, 2.4811989404941977e+02, 0., 0., 1. ]
depth_distortion: !!opencv-matrix
   rows: 1
   cols: 5
   dt: d
   data: [ -2.3987660910278472e-01, 1.5996260959757911e+00,
       -8.4261854767272721e-04, 1.1084546789468565e-03,
       -4.1018226565578777e+00 ]
```

Figure 3.4.2. Recording Calibration Information

Source: Zhengfei Yin, 2012.

3.4.2 Object Reconstruction Based on PCL

After calibrating the Kinect camera, the software can acquire meaningful colour images and depth maps. The second step of the software development is making use of PCL to reconstruction objects.

(1) Creating Point Cloud from Depth Map

A point cloud can be defined as a set of 3D points which have not connection with each other. The reason why it is called cloud is that the points are separated from each other and there is little connectivity between them (Rusu and Cousins, 2011). Basically, a point cloud just contains the position information, but each point can include colour information and normal orientation.

A pixel in the depth images will be transformed into 3D point without considering the connection between itself and other pixels. Hence, it should be recognized that point clouds are the natural method to represent depth images captured from Kinect. As can be seen from Figure 3.4.3, the 3D point with coordinates is defined.

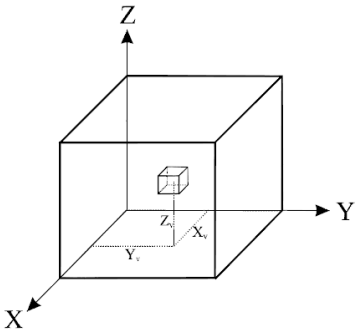

Figure 3.4.3. A 3D Point with Coordinates (X_V, Y_V, Z_V)

Source: Walck and Drouin, 2009.

The depth data captured by the Kinect camera is 2D intensity image, which means that some specific methods are required to conduct the transformation. According to Levoy and Rusinkiewicz (2001), the transformation from depth pixel coordinates to 3D point coordinates can be specified in the equation 3.1. S_v represents the metric size of a pixel, X, Y, Z are the 3D point coordinates, $X_V^{'}$, $Y_V^{'}$, $Z_V^{'}$ are the pixel coordinates and X_v^0, Y_v^0, Z_v^0 are the depth pixel in the 3D coordinates. Moreover, Z is the projection angle about the 3D point coordinate.

$$\begin{aligned} X &= s_v \bullet \cos\alpha \bullet X_V^{'} - s_v \bullet \sin\alpha \bullet Y_V^{'} + X_v^0 \\ Y &= s_v \bullet \sin\alpha \bullet X_V^{'} + s_v \bullet \cos\alpha \bullet Y_V^{'} + Y_v^0 \\ Z &= s_v \bullet Z_V^{'} + Z_v^0 \end{aligned} \qquad (3.1)$$

The 3D point coordinates has been transformed into depth data coordinates. By the end of this step, Kinect depth data has been constructed to 3D point cloud.

(2) Normal Estimation

The second step of the object reconstruction is estimating the surface normal. Walck and Drouin highlight that a surface is represented by a set of vertices and their connectivity between each other (2009). More exactly, every vertex contains a position, normal direction and colour information. In this project, the normal direction of each 3D point cannot be measured by the Kinect device. However, by making use of the Iterative Closest Point (ICP) method which has been specified in chapter II, the normal of each 3D point can be estimated (Figure 3.4.4).

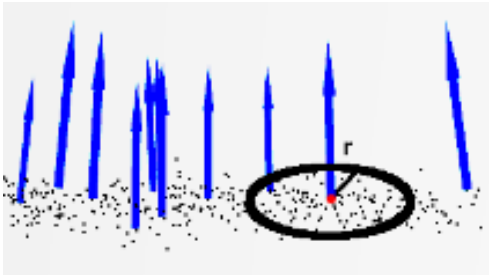

Figure 3.4.4. 3D Point with Normal

Source: Duckworth and Roberts, 2011.

(3) Extracting Triangulation Mesh from Point Cloud

The last step of the object segmentation is extract triangulation mesh for the point clouds. The primary challenge of this step is to determine which vertices should be connected with each other. Fortunately, the PCL provides the project with related functions to determine vertices which need to be combined with others. Moreover, the ICP method is also employed in this step to calculate the distance between two points and estimate the scale of a point cloud. More specifically, by taking advantage of the ICP and PCL functions, the nearest neighbours of point clouds are combined are extracted to mesh. As can be seen from Figure 3.4.5, after rendering the triangulation mesh with ray casting technique, the software can get the object reconstructed.

Figure 3.4.5. Rendered Objects and Depth Map

Source: Zhengfei Yin, 2012.

3.4.3 Simultaneous 3D Gesture

As indicated previously, the third stage of the software development is tracking and recognizing the hand gesture. In this step, a novel hand gesture algorithm: simultaneous 3D gesture is proposed.

(1) Image Capture and Background Elimination.

The first step is capturing an image and storing it as the gray scale image. As mentioned earlier, Kinect is able to acquire depth maps by making use of the infrared camera. By making use this feature, the shape of hand can be captured. After capturing the image, a modified threshold filter is employed to eliminate the background. Due to the different depth information between foreground (the hand gesture) and the background, the background can be eliminated by altering gray level threshold. As can be seen from Figure 3.4.6, the background can be subtracted by applying a threshold filter.

Figure 3.4.6. Depth Image and Background Elimination

Source: Zabulis, et al., 2009.

(2) Hand Extraction

The next step of the simultaneous 3D gesture is extracting the gesture. Gesture extraction can be defined as a process that captures hand features and transfers hand gesture into data. The reason why this factor is significant is that well-defining hand gesture can help system filter unwanted data and only capture relevant movements. The detailed equation of hand extraction has been presented in the second chapter; by making use of the method, the hand can be extracted shown in Figure 3.4.7.

Figure 3.4.7. Gesture Extraction

Source: Zabulis, et al., 2009.

(3) Gesture Leaning and Tracking

In this process, the simultaneous 3D gestures method learns prior models of gesture from sample pose feature data. More precisely, in the step, a user is supposed to provide the software with natural gestures. After the gestures are recognized and tracked, these gestures can be used to conduct the segmentation process. The detailed equations and algorithm have been specified in the second chapter. Referring to Figure 3.4.8, the gesture can be tracked and recognized by the end of this stage.

Figure 3.4.8. Gesture Tracking and Recognition

Source: Zhengfei Yin, 2012.

Obviously, the object reconstruction is represented by 3D point cloud. In order to interact with the reconstructed model, the tracked and recognized gesture should be also modelled to 3D point cloud (Figure 3.4.9). More exactly, the software will model the gesture during the tracking and recognition process, after the gesture learning step. In other words, only the recognized gestures which are able to be used during the segmentation process will be represented by 3D point cloud. This operation makes sure that the recognized gestures can be employed in the 3D point cloud reconstruction environment.

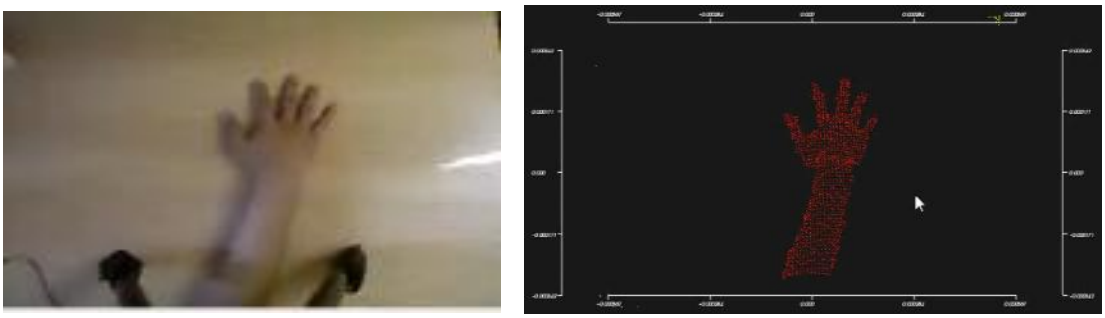

Figure 3.4.9. RGB Image and 3D Point Cloud Gesture

Source: Zhengfei Yin, 2012.

3.4.4 Object Segmentation

This subsection is used to demonstrate the object segmentation functionality. More precisely, the segmentation can be described as an application which combines functions of the object reconstruction component and the gesture tracking algorithm together. The developing process is illustrated as follows.

(1) Organizing and Clustering Point Clouds

As indicated in previous subsections, depth map captured by the Kinect has been transformed into 3D point cloud. More importantly, the surface normal has been estimated at the end of the model reconstruction step. Based on the estimated normal of the surface, points belonging to one object are organized and connected. Referring to Figure 3.4.10, the boundary of one object has been detected. Afterwards, point clusters of object will be generated.

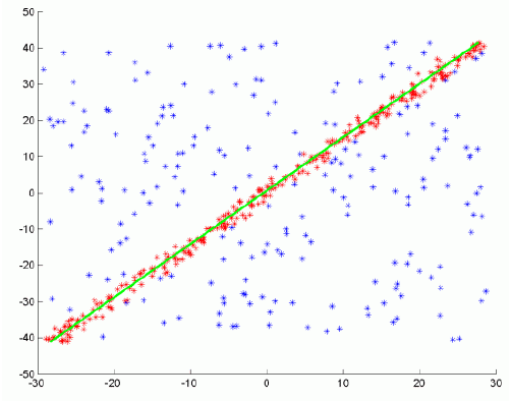

Figure 3.4.10. The Point Cloud Library Logo

Source: Rusu and Cousins, 2011.

(2) Integrating Gesture with Segmentation Method

Each cluster will be given different colour, which means that user can find the clear boundaries of the segmented regions. Apparently, the position of object clusters will not change. Hence, the moving object with disparity oriented points which could be detected by the Kinect is defined as user's gesture. Moreover, the gesture is specified as the part of foreground scene, and assigned with special ICP projective data. Additionally, the motion of the gesture will be detected when the gesture is getting close to an object. The graph (Figrue 3.4.11) shows that the 3D point gesture is ready to interact with the reconstructed objects.

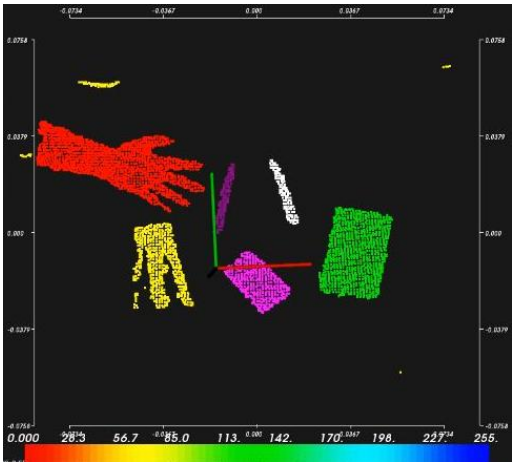

Figure 3.4.11. 3D Point Cloud Gesture and Reconstructed Objects

Source: Zhengfei Yin, 2012.

By the end of the design and develop stage, the essential functionality of this software has been projected and programmed. Precisely, objects can be reconstructed, various natural gesture can be detected and tracking. Importantly, gestures are represented as 3D point cloud, which can be utilized to conduct the object segmentation process. The experimental results of the software will be presented in the next chapter. Inevitably, the evaluation process of the result will be also proposed to investigate the performance and functionality of this software.

IV EXPERIMENTAL RESULTS & EVALUATION

This chapter demonstrates the experimental results of the software, the evaluation processes of the chapter are also conducted in the same chapter. Apparently, the evaluation methods have been proposed in second chapter. Hence, the results of the project are estimated and analyzed based on the measurable evaluation parameters. Subsequently, the advantages of the algorithms, the performance of the system and the achievements of the objectives are investigated and illustrated in the discussion section.

There are several specific features of this software should be mentioned at the beginning of this chapter.

- First, due to the limitation of the hardware, several functions of the software have been adjusted to achieve the real time interaction.

For example, the reconstructed objects will not be rendered by making use of ray casting technique. On the hand, the objects are represented by 3D point clouds to make sure the segmentation process is real time interaction between user and the objects.

- Secondly, objects used to be reconstructed are placed on a table which will be eliminated during the reconstruction process (Figure 4.1). Similarly, the process was proposed to reduce the burden of CPU of my machine and make sure the real time interaction.

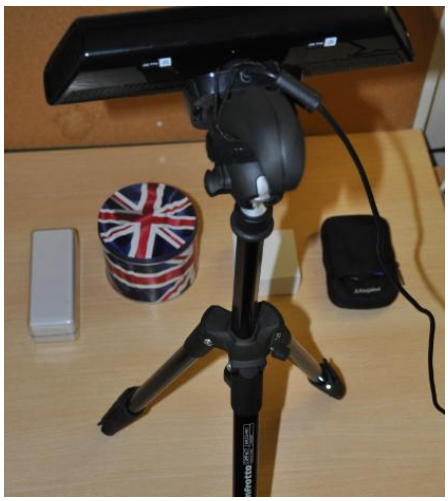

Figure 4.1. Objects Placed on a Table

Source: Zhengfei Yin, 2012.

- Thirdly, as interpreted in the last chapter, recognized gestures are represented as 3D point clouds. General, the 3D point cloud gestures are assigned with special colour (red in most situations), which can be found out as the gestures.

4.1 Results and Evaluation

According to the structure of the entire software, the experimental results contain three components. Experimental results of object reconstruction, hand tracking and object segmentation will be presented respectively. Moreover, the evaluation parameters have been specified for each component.

4.1.1 Object Reconstruction

(1) Results

There is no denying the fact that the reconstruction process is the fundamental component of the software. Hence, the results of this algorithm are presented at first. Basically, there are three stages of the experimental results: the 3D point cloud cluster, the triangulation mesh and ray casting rendered model.

As can be seen from Figure 4.1.1, four models are captured by the Kinect from the top view are reconstructed as 3D point cloud clusters with different colours.

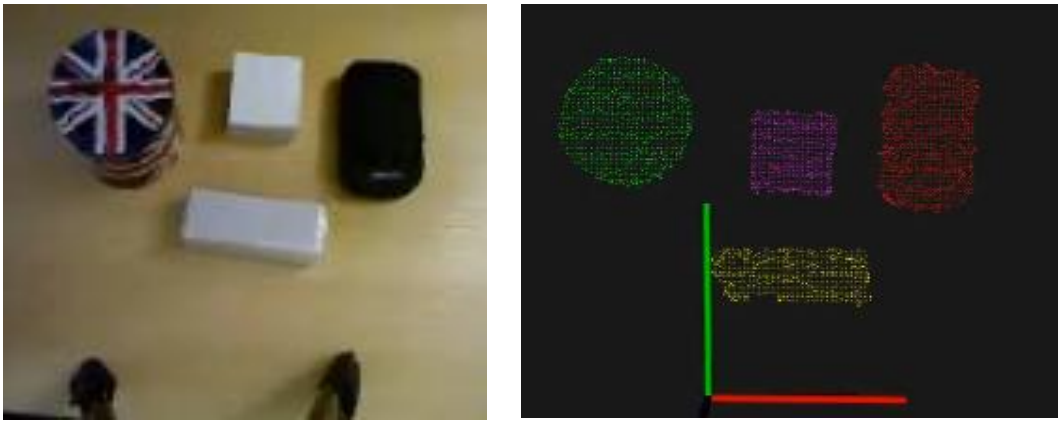

Figure 4.1.1. RGB Image and 3D Point Cloud Cluster Reconstruction

Source: Zhengfei Yin, 2012.

The second stage is acquiring triangulation mesh from the point cloud (Figure 4.1.2). From the graph it can be indicated that the mesh has been achieved successfully. More importantly, the mesh can be manipulated by making use of the MeshLab software.

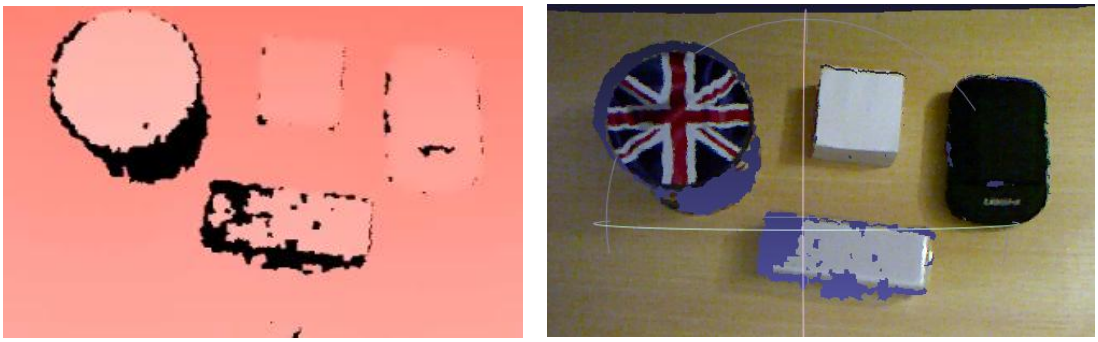

Figure 4.1.2. Depth Map and Triangulation Mesh Reconstruction

Source: Zhengfei Yin, 2012.

Thirdly, the last stage of the reconstruction is rendering the point clouds by making use of ray casting method (Figure 4.1.3). More precisely, in order to reduce the burden of the hardware and acquire the real-time interaction, the 3D point cloud clustering reconstruction is employed in the experimental process.

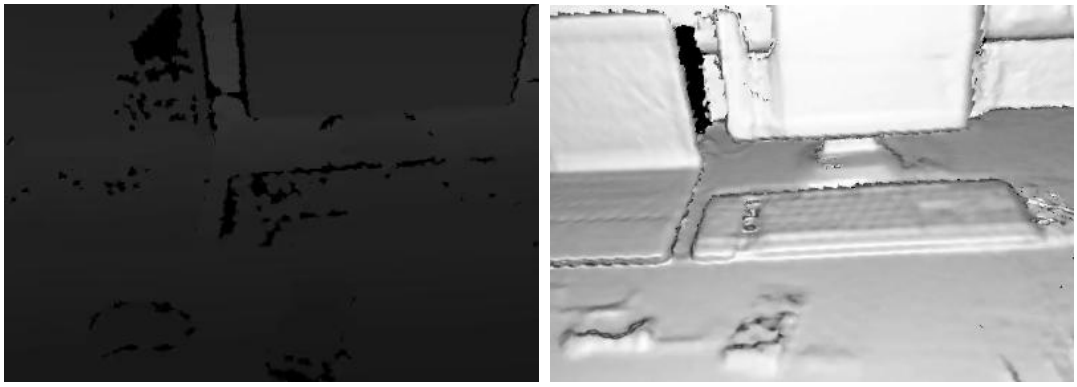

Figure 4.1.3. Depth Map and Ray Casting Objects Reconstruction

Source: Zhengfei Yin, 2012.

(2) Evaluation

As indicated previously, there are three measurable evaluation parameters for the reconstruction process: reconstruction distortion, rate and time. In order to evaluate the performance of this reconstruction method, a conventional surface-based reconstruction algorithm (proposed by Duckworth and Roberts, 2011) is used to make comparison with it.

First of all, the comparison of reconstruction distortion between the two algorithms is conducted, by utilizing the equation provided by Steinbach, et al. (2000). Referring to the Figure 4.1.4, it is clear that the reconstruction distortion of the method employed in this project is less the traditional reconstruction methods.

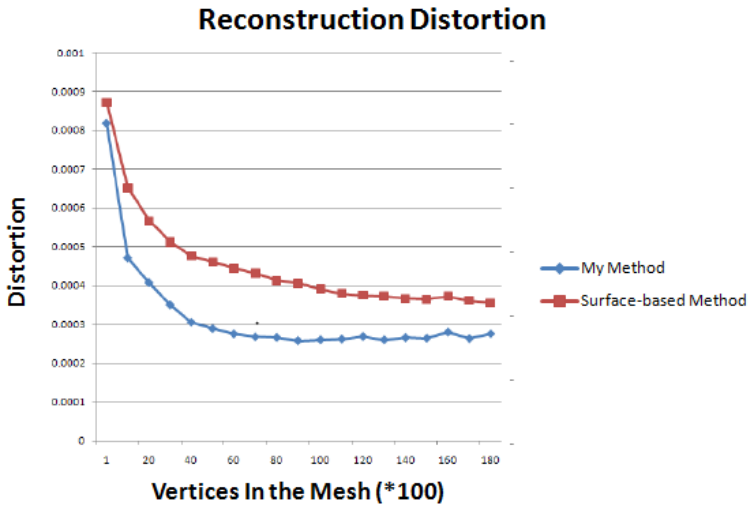

Figure 4.1.4. Reconstruction Distortion Comparison

Source: Zhengfei Yin, 2012.

On the other hand, the reconstruction rate of my algorithm is better than the conventional surface-based algorithm. In other words, mesh reconstructed by my algorithm will be smoother than others (Figure 4.1.5). More importantly, it will take less time to reconstruct a model by making use of this algorithm than the surface-based method. Evidence to support this comes from both the normal estimation and depth map capturing processes.

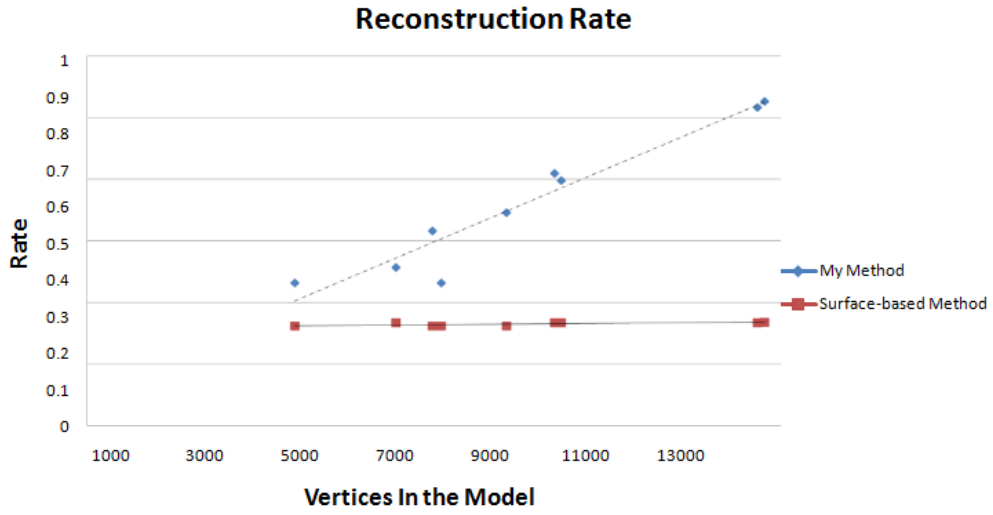

Figure 4.1.5. Reconstruction Rate Comparison

Source: Zhengfei Yin, 2012.

4.1.2 Gesture Tracking

(1) Outcomes

The results of the gesture tracking step can be divided into two parts: the results of the gesture leaning process and the results of the gesture tracking process. During the gesture leaning process, depth maps are captured by the Kinect, hand gestures are extracted from the background. As can be seen from Figure 4.1.6, various natural gestures can be detected and extracted.

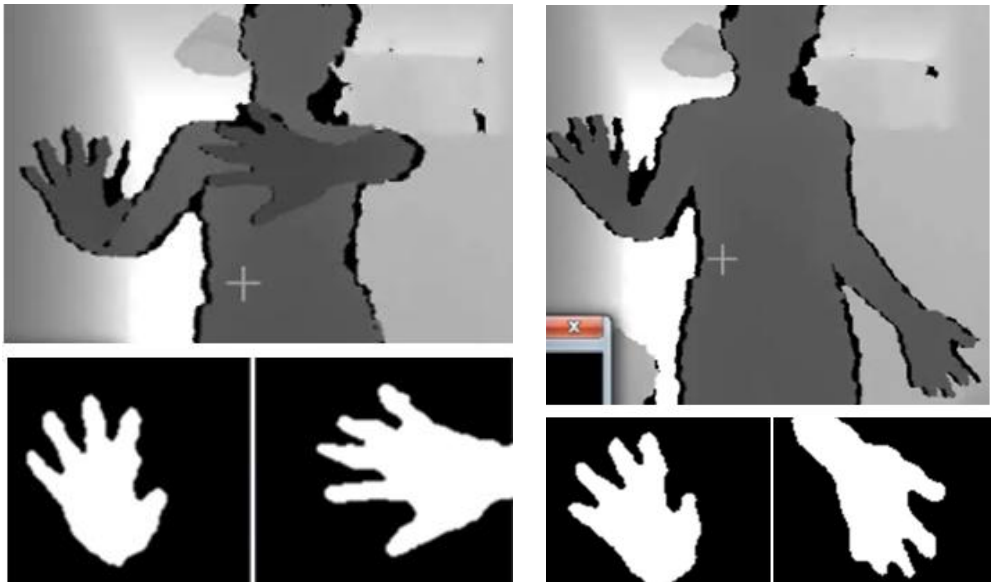

Figure 4.1.6. Depth Maps and Gesture Recognition

Source: Zhengfei Yin, 2012.

More significantly, during the gesture tracking and recognizing process, the recognized gestures will be represented as 3D point clouds. By taking advantage of the 3D point cloud, recognized gestures can be used to interact with the reconstructed models. From the Figure 4.1.7, recognized gestures are represented as 3D point cloud and are able to conduct the object segmentation.

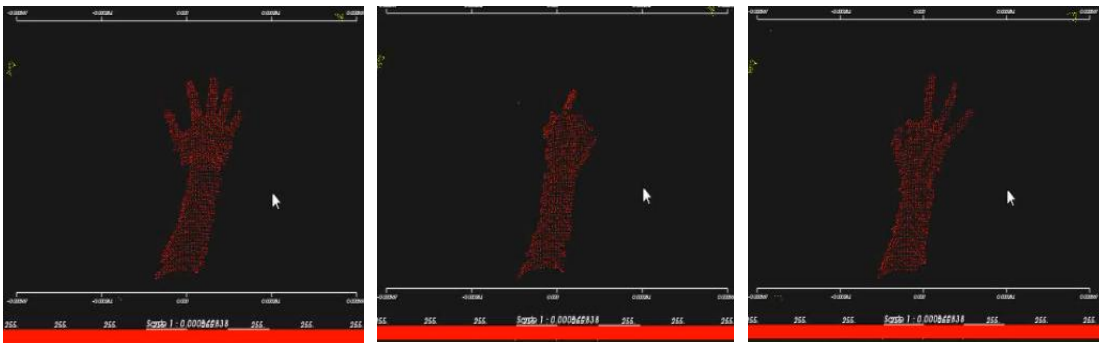

Figure 4.1.7. 3D Point Cloud Recognized Gestures

Source: Zhengfei Yin, 2012.

(2) Evaluation

Specifically, there are two measurable evaluation factors. The first one is recognition rate which refers to the gestures tracking and recognition time. A lot of tests have been made to implement the tracking and recognition speed. By making comparison with traditional vision-based hand tracking algorithm (Holz and Wilson, 2011), the speed of my method is 2-3 times faster than the vision-based algorithm which makes used of webcam to capture data.

Moreover, the second evaluation factor is recognition quality which refers to the accuracy of the gesture tracking and recognition. Table 4.1.1 shows the best recognition quality is 94.13% during the gesture learning process. In terms of the gesture tracking and recognition step the quality is 86.97%.

Tabel 4.1.1

The Recognition Quality of Gesture learning and Tracking

Source: Zhengfei Yin, 2012

Processes	Number of Testing Gestures	Number of Correct Gestures	Number of Fail Gestures	Recognition Quality
Gesture Learning	460	433	27	94.13%
Gesture Tracking & Recognition	376	327	49	86.97%

4.1.3 Model Segmentation

(1) Results

Basically, the object segmentation process can be defined as an application which combines the object reconstruction algorithm and gesture tracking algorithm. More exactly, before segmenting the objects, objects should reconstructed first. At the same time, gestures used to conduct the segmentation should have been recognized and tracked. From the Figure 4.1.8, it can be indicated that four models have been reconstructed as 3D point cloud clusters and assigned with different colour. More importantly, the gesture has been detected and tracked.

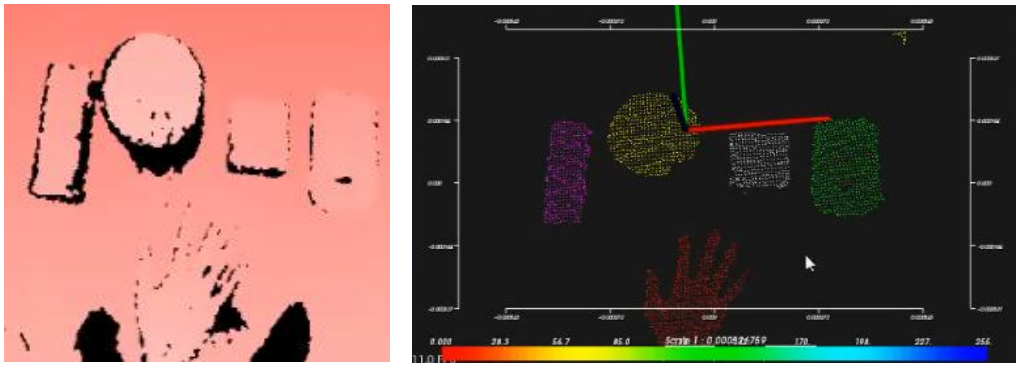

Figure 4.1.8. Segmentation Preparation

Source: Zhengfei Yin, 2012.

Afterwards, the 3D point cloud gesture can be employed to conduct the segmentation. Referring to Figure 4.1.9, when the distance between the gesture and a model is the shortest, namely, the gesture is closest to this object, this object can be segmented. Moreover, the colour of the model will be altered to the same colour as the gesture (Basically, the colour of the tracked gesture is red).

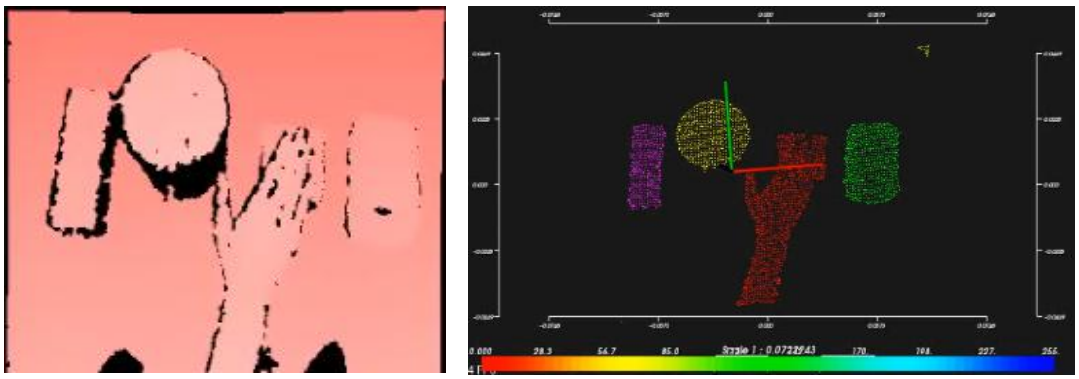

Figure 4.1.9. Segmentation Results

Source: Zhengfei Yin, 2012.

(2) Assessments

According to the evaluation process of the segmentation step provided in the second chapter, there are two primary elements involved in assessments: segmentation covering metric and time consuming. By making use of the equation of the covering metric, the covering metric calculated in Figure 4.1.10. As can be seen from the figure, with the increase of the 3D point cloud, the covering metric is raising. In other words, this algorithm can acquire high segmentation accuracy. More exactly, the average segmentation covering value is 0.8746. On the other hand, another assessment is segmentation time consuming. As the entire system provides user with real-time segmentation experience, the segmentation can be achieved immediately.

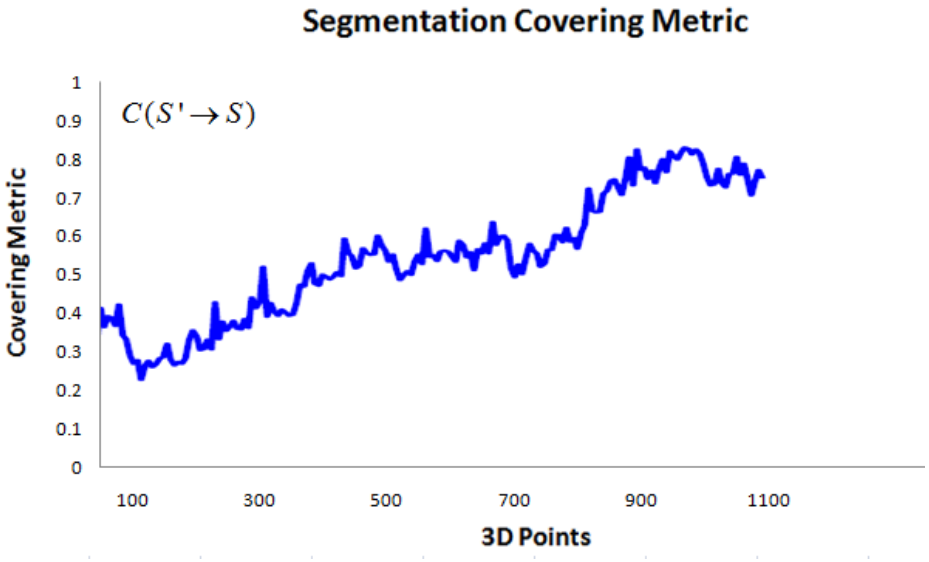

Figure 4.1.10. Segmentation Covering Metric

Source: Zhengfei Yin, 2012.

4.2 Analysis and Discussion

4.2.1 Performance and Advantages

This subsection is used to analyze the performance and the advantages of the entire software system based on the evaluation processes. Basically, performance and advantages can be divided into five components:

(1) First of all, high quality and geometrically accurate 3D models can be reconstructed by this software. More importantly, the software provides user with three different types of reconstruction: 3D point cloud cluster, triangulation mesh and ray casting rendered object.

(2) Secondly, the software provides user with a real-time interaction system. In other words, user can interact with the reconstructed objects immediately.

(3) Thirdly, natural gestures are used to interact with the system. User can just make use their hand to interact with the software without any additional devices or equipments.

(4) The next advantage is high gesture tracking rate and accurate gesture recognition. Various gestures can be tracking and recognized to conduct the segmentation process.

(5) The last one is that the software supply user with an accurate and efficient object segmentation algorithm.

4.2.2 Contributions and Achievements

After explaining the performance and advantages of the entire software, the contributions and achievements of this project are summarised in this subsection. Additionally, it should be mentioned that the major added value and innovation of the project is combining gesture tracking algorithm with object segmentation method to interact with reconstructed models.

(1) To begin with, more than 100 literatures related to Kinect, KinectFusion, object reconstruction, object recognition, object segmentation, gesture tracking, gesture detection, gesture recognition and Point Cloud Library have been reviewed. Most of them are contributed in my project.

(2) In terms of the hardware configuration, the features and functionality of Kinect device have been grasped and applied in this project. More importantly, the thesis proposed and implemented the Kinect calibration process.

(3) On the other hand, a special and novel approach was utilized to fix the Kinect with a tripod. Additionally, the technical features and functions of the graphics card (GTX 480) and power supply (Corsair AX850) has been grasped as well.

(4) Referring to the objectives of my project, the strategy and functionality of the KinectFusion algorithm have been explored and applied in my project. Object reconstruction algorithm based on Point Cloud Library has been proposed and implemented in this project. Moreover, the performance of the reconstruction process has been investigated and analyzed.

(5) Subsequently, a novel gesture tracking and recognition algorithm: simultaneous 3D gesture was proposed in this paper. Obviously, this method is proved as the most significant and fundamental technique in my thesis. In addition, the feasibility of this method was verified by proposing the evaluation process.

(6) Afterwards, the object segmentation component which combines the object reconstruction algorithm (based on PCL) and simultaneous 3D gestures was developed as the primary functionality of my software. The paper has investigated the performance of the segmentation algorithm.

(7) The final contribution is the thesis has verified the validity of the entire software by conducting experimental analysis and investigating its performance and functionality. The evaluation process was proposed by making comparison with related conventional algorithms.

V. CONCLUSION & FUTURE WORK

To conclude, a system which is able to reconstruct objects and enable user to segment the reconstructed models by making use of their gestures has been proposed in this thesis. Obviously, there are four technical components involved in this thesis: the Kinect, object reconstruction method based on Point Cloud Library, gesture tracking and object segmentation. The Kinect device has been employed in my project to capture depth maps and colour images for the software. Based on the strategy of KinectFusion, object reconstruction algorithm has been implemented with Point Cloud Library to reconstruct objects. Moreover, a novel gesture tracking and recognition method has been proposed in this thesis. After the gesture learning process, recognized gestures can be utilized to interact with the 3D reconstructed models. Precisely, the interaction has been specified as segmentation. Object reconstruction, gesture tracking and object segmentation are three fundamental elements of this project. In order to investigate the performance of the software, each element has been evaluated by designing and conducting specific process. Consequently, the performance of the novel techniques was demonstrated by comparing with conventional algorithms, and the feasibility of this software was verified by analyzing the experimental results.

Although the performance of the software is satisfactory, the functionality can be enhanced and the efficiency of the software can be improved. The possible improvements and future work are described as follow:

(1) Making Use of Multiple Kinects

In this project, only one single Kinect was utilized to capture data. The field view and depth reach of a Kinect is limited. Hence, by making use multiple Kinects, the efficiency of the system will be improved.

(2) Applying Models Rendered by Ray Casting

Although the project is able to produce ray casting models, it is too hardware consuming to interacted with in real-time scenario. By modifying the hardware configuration (especially the graphics card), this functionality can be achieved.

(3) Multiple Users and Various Operations

Another possible improvement of the software is enabling several users to interact with the reconstructed models at the same time. In addition, besides the segmentation, different kinds of operations (scale and rotate the models) can be integrated into the software.

VI. BIBLIOGRAPHY

Abramov, A., Pauwels, K., Papon, J., Worgotter, F. and Dellen, B. (2012). Depth-supported Real-time Video Segmentation with the Kinect. Applications of Computer Vision, 2012 IEEE Workshop on. (On Pages: 457-464). ISSN: 1550-5790.

Aggarwal, J.K., Lu, X. and Chen, C. (2011). Human Detection Using Depth Information by Kinect. Computer Vision and Pattern Recognition Workshops. (On Pages: 15–22). ISSN: 2160-7508.

Arici, T. (2012). Introduction to programming with Kinect. Signal Processing and Communications Applications Conference (SIU), 20th 2012. (On pages: 1). ISBN: 978-1-4673-0055-1. Istanbul Sehir University, Istanbul, Turkey.

Bigdelou, A., Benz, T., Schwarz, L. and Navab, N. (2012). Simultaneous Categorical and Spatio-temporal 3D Gestures Using Kinect. 3D User Interfaces, 2012 IEEE Symposium on. (On Pages: 53–60). ISBN: 978-1-4673-1204-2.

Chavez, M. and Kyaw, A. (2011). A Gesture-based Interface and Active Cinema. Affective Computing and Intelligent Interaction, volume 6975 of Lecture Notes in Computer Science. (On Pages: 309–310).

Dal Mutto, C., Zanuttigh, P. and Cortelazzo, G. (2012). Fusion of Geometry and Color Information for Scene Segmentation. IEEE Journal of Selected Topics in Signal Processing. ISSN: 1932-4553.

Duckworth, T. and Roberts, D.J. (2011). Camera Image Synchronisation in Multiple Camera Real-Time 3D Reconstruction of Moving Humans. Distributed Simulation and Real Time Applications, (On Pages: 138–144). ISBN: 978-1-4577-1643-0.

Faugeras, 0. (2001). Three-Dimensional Computer Vision: A Geometric Viewpoint. The MIT Press, London, 2001.

Flores, F.C. and de Alencar Lotufo, R. (2008). Benchmark for Quantitative Evaluation of Assisted Object Segmentation Methods to Image Sequences. Computer Graphics and Image Processing. (On Pages: 95-102). ISSN: 1530-1834.

Frati, V. and Prattichizzo, D. (2011). Using Kinect for Hand Tracking and Rendering in Wearable Haptics. World Haptics Conference, 2011 IEEE (On Pages: 317–321).

Holz, C. and Wilson, A.D. (2011). Data Miming: Inferring Spatial Object Descriptions from Human Gesture. In Proceedings of the 2011 annual conference on Human factors in computing systems. ACM, New York, USA.

Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A. and Fitzgibbon, A. (2011). KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera. ACM Symposium on User Interface Software and Technology, October 2011.

Levoy, M. and Rusinkiewicz, S. (2001). Efficient Variants of the ICP Algorithm. 3D Digital Imaging and Modeling, Int. Conf. on, 2001.

Leyvand, T., Meekhof, C., Wei, Y., Sun, J. and Guo, B. (2011). Kinect Identity: Technology and Experience. Computing & Processing. (On Pages: 94–96). ISSN : 0018-9162.

Mutto, C.D., Zanuttigh, P. and Cortelazzo, G.M. (2012). Fusion of Geometry and Color Information for Scene Segmentation. Department of Information Engineering, University of Padova, Italy. April 2012. ISSN : 1932-4553.

Newcombe, R. Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A. and Fitzgibbon, A. (2011). KinectFusion: Real-Time Dense Surface Mapping and Tracking. In IEEE ISMAR, IEEE, October 2011.

Nguyen, Q.M., Nam, V., Tran, Q., Thang, B.D. and Tien, B.D. (2010). An Efficient Human-Computer Interaction Framework Using Skin Color Tracking and Gesture Recognition. Computing and Communication Technologies, Research, Innovation, and Vision for the Future, 2010 IEEE RIVF International Conference on. (On Pages: 1-6). ISBN: 978-1-4244-8074-6.

Noonan, P.J., Cootes, T.F., Hallett, W.A. and Hinz, R. (2011). The Design and Initial Calibration of an Optical Tracking System Using the Microsoft Kinect. Nuclear Science Symposium and Medical Imaging Conference. (On Pages: 3614–3617). ISBN: 978-1-4673-0118-3.

Ren, Z., Meng, J. and Yuan, J. (2011). Depth Camera Based Hand Gesture Recognition and its Applications in Human-Computer-Interaction. Information, Communications and Signal Processing. International Conference on. (On Pages: 1-5). ISBN: 978-1-4577-0029-3.

Rogge, S., Amtsfeld, P., Hentschel, C., Bestle, D. and Meyer, M. (2012). Using gestures to interactively modify turbine blades in a Virtual Environment. Emerging Signal Processing Applications (ESPA), 2012 IEEE International Conference on. (On Pages: 155-158). ISBN: 978-1-4673-0899-1. Media Technol., Brandenburg Univ. of Technol., Cottbus, Germany.

Rusu, R.B. and Cousins, S. (2011). 3D is here: Point Cloud Library (PCL). Robotics and Automation (ICRA), 2011 IEEE International Conference on. (On Pages: 1-4). ISBN: 978-1-61284-386-5. Willow Garage, Menlo Park, CA, USA.

Silberman, N. and Fergus, R. (2011). Indoor Scene Segmentation Using a Structured Light Sensor. Computer Vision Workshops, 2011 IEEE International Conference on. (On Pages: 601-608). ISBN: 978-1-4673-0062-9.

Smisek, J., Jancosek, M. and Pajdla, T. (2011). 3D with Kinect. Computer Vision Workshops, 2011 IEEE. (On Pages: 1154–1160). ISBN: 978-1-4673-0062-9.

Steinbach, E., Girod, B., Eisert, P. and Betz, A. (2000). 3-D Reconstruction of Real-world Objects Using Extended Voxels. Image Processing. International Conference. ISBN: 0-7803-6297-7.

Sun, G., Xu, L., Chen, D., Li, G. and Wang, J. (2008). A system Scheme of 3D Object Reconstruction from Single 2D Graphics Based on Neural Networks. Industrial Informatics, 2008. (On Pages: 1081–1085). ISSN : 1935-4576.

Tsai, T.H. and Lin, C.Y. (2007). Visual Hand Gesture Segmentation Using Signer Model for Real-Time Human-Computer Interaction Application. Signal Processing Systems, 2007 IEEE Workshop on. (On Pages: 567-572).

Walck, G. and Drouin, M. (2009). Progressive 3D Reconstruction of Unknown Objects Using One Eye-in-hand Camera. Robotics and Biomimetics, IEEE International Conference. (On Page: 971). ISBN: 978-1-4244-4774-9.

Wallenberg, M., Felsberg, M., Forssen, P. and Dellen, B. (2011). Channel Coding for Joint Colour and Depth Segmentation. Proceedings of Pattern Recognition 33rd DAGM Symposium, vol. 6835 of Lecture Notes in Computer Science. (On Pages: 306–315).

Zabulis, X., Baltzakis, H. and Argyros, A.A. (2009). Vision-based Hand Gesture Recognition for Human-Computer Interaction. Lawrence Erlbaum Associates, Inc. Series on "Human factors and ergonomics". (On Pages: 34.1 – 34.30). ISBN: 978-0-8058-6280-5.

Zhu, H.M. and Pun, C.M. (2010). Movement Tracking in Real-Time Hand Gesture Recognition. Computer and Information Science, 2010 IEEE/ACIS 9th International Conference on. (On Pages: 240-245). ISBN: 978-1-4244-8198-9.

VII. APPENDIX

(1) Object Reconstruction and Segmentation Functions

```
#include <iostream>
#include <ntk/ntk.h>
#include <ntk/mesh/mesh_generator.h>
#include <ntk/utils/debug.h>
#include <ntk/camera/openni_grabber.h>
#include <ntk/gesture/body_event.h>
#include <ntk/mesh/pcl_utils.h>
#include
<ntk/geometry/relative_pose_estimator_icp.h>
#include <pcl/ModelCoefficients.h>
#include <pcl/point_types.h>
#include <pcl/io/pcd_io.h>
#include <pcl/filters/extract_indices.h>
#include <pcl/filters/voxel_grid.h>
#include <pcl/features/normal_3d.h>
#include <pcl/kdtree/kdtree.h>
#include <pcl/sample_consensus/method_types.h>
#include <pcl/sample_consensus/model_types.h>
#include <pcl/segmentation/sac_segmentation.h>
#include <pcl/segmentation/extract_clusters.h>
#include <pcl/visualization/cloud_viewer.h>
#include <pcl/point_types.h>
#include <pcl/filters/passthrough.h>
#include <QApplication>
#include <QDir>
#include <QMutex>

using namespace ntk;
using namespace std;
namespace opt
{
ntk::arg<bool> high_resolution("--highres", "High
resolution color image.", 0);
ntk::arg<int> kinect_id("--kinect-id", "Kinect id", 0);
}
int main (int argc, char** argv)
{
  arg_base::set_help_option("-h");
  arg_parse(argc, argv);
  ntk::ntk_debug_level = 1;
  QApplication app (argc, argv);

QDir::setCurrent(QApplication::applicationDirPath(
));

  OpenniDriver ni_driver;
  OpenniGrabber grabber(ni_driver,
opt::kinect_id());
        int i=0;
rgbdImageToPointCloud(*cloud, image);
 std::cout << "PointCloud before filtering has: " <<
cloud->points.size () << " data points." << std::endl;
//*
 // Create the filtering object
 int i=0, nr_points = (int) cloud_filtered-
>points.size ();
 while (cloud_filtered->points.size () > 0.3 *
nr_points)
 {
  seg.setInputCloud (cloud_filtered);
  seg.segment (*inliers, *coefficients);
  if (inliers->indices.size () == 0)
  {
   std::cout << "Could not estimate a planar model
for the given dataset." << std::endl;
   break;
  }
  // Extract the planar inliers from the input cloud
```

```
    pcl::ExtractIndices<pcl::PointXYZ> extract;
    extract.setInputCloud (cloud_filtered);
    extract.setIndices (inliers);
    extract.setNegative (false);

    // Write the planar inliers to disk
    extract.filter (*cloud_plane);
    std::cout << "PointCloud representing the planar component: " << cloud_plane->points.size () << " data points." << std::endl;

    // Remove the planar inliers, extract the rest
    extract.setNegative (true);
    extract.filter (*cloud_f);
    cloud_filtered = cloud_f;
  }

  // Creating the KdTree object for the search method of the extraction
  pcl::search::KdTree<pcl::PointXYZ>::Ptr tree (new pcl::search::KdTree<pcl::PointXYZ>);
  tree->setInputCloud (cloud_filtered);

  std::vector<pcl::PointIndices> cluster_indices;
  pcl::EuclideanClusterExtraction<pcl::PointXYZ> ec;
  ec.setClusterTolerance (0.02); // 2cm
  ec.setMinClusterSize (100);
  ec.setMaxClusterSize (25000);
  ec.setSearchMethod (tree);
  ec.setInputCloud (cloud_filtered);
  ec.extract (cluster_indices);

  std::cout << "PointCloud clustering has: " << cluster_indices.size() << " data points." << std::endl;

            pcl::PointCloud<pcl::PointXYZRGB> colorCloud;

            //copy original points to colorCloud
            colorCloud.width = cloud_filtered->width;
            colorCloud.height = cloud_filtered->height;
            colorCloud.points.resize(colorCloud.width*colorCloud.height);

            for(int i=0;i<cloud_filtered->points.size();i++)
            {
                        colorCloud.points[i].x = cloud_filtered->points[i].x;
                        colorCloud.points[i].y = cloud_filtered->points[i].y;
                        colorCloud.points[i].z = cloud_filtered->points[i].z;
                        colorCloud.points[i].r = 255;
                        colorCloud.points[i].g = 255;
                        colorCloud.points[i].b = 0;
            }

            viewer.showCloud (colorCloud.makeShared());
            char c = cvWaitKey(1);
        if( c == 27 ) break;

            }

  grabber.stop();

  return (0);
}
```

(2) Hand Tracking and Recognition Function

```
#include <iostream>
#include "SkeletonSensor.h"
#include <opencv/highgui.h>
#include <opencv/cv.h>

using namespace cv;
using namespace std;

SkeletonSensor* sensor;

const unsigned int XRES = 640;
const unsigned int YRES = 480;
const float DEPTH_SCALE_FACTOR =
255./4096.;
const unsigned int BIN_THRESH_OFFSET = 5;
const unsigned int ROI_OFFSET = 70;
const unsigned int MEDIAN_BLUR_K = 5;
const double GRASPING_THRESH = 0.9;

bool handApproachingDisplayPerimeter(float x,
float y)
{
   return (x > (XRES - ROI_OFFSET)) || (x <
(ROI_OFFSET)) ||
         (y > (YRES - ROI_OFFSET)) || (y <
(ROI_OFFSET));
}

struct ConvexityDefect
{
   Point start;
   Point end;
   Point depth_point;
   float depth;
};

void findConvexityDefects(vector<Point>&
contour, vector<int>& hull,

      // calculate convexity defects
      storage = cvCreateMemStorage(0);
      defects = cvConvexityDefects(contourPoints,
&hullMat, storage);
      defectArray =
(CvConvexityDefect*)malloc(sizeof(CvConvexit
yDefect)*defects->total);
      cvCvtSeqToArray(defects, defectArray,
CV_WHOLE_SEQ);
      for(int i = 0; i<defects->total; i++){
         ConvexityDefect def;
         def.start      = Point(defectArray[i].start-
>x, defectArray[i].start->y);
         def.end        = Point(defectArray[i].end-
>x, defectArray[i].end->y);
         def.depth_point =
Point(defectArray[i].depth_point->x,
defectArray[i].depth_point->y);
         def.depth      = defectArray[i].depth;
         convexDefects.push_back(def);
      }
   cvReleaseMemStorage(&contourStr);
   cvReleaseMemStorage(&strDefects);
   cvReleaseMemStorage(&storage);

   }
}

int main(int argc, char** argv)
{
   sensor = new SkeletonSensor();
   sensor->initialize();
   sensor->setPointModeToProjective();
```

```
    Mat depthRaw(YRES, XRES, CV_16UC1);
    Mat depthShow(YRES, XRES, CV_8UC1);
    Mat handDebug;
    vector<Mat> debugFrames;
    Rect roi;
    roi.width  = ROI_OFFSET*2;
    roi.height = ROI_OFFSET*2;

    namedWindow("depthFrame",
CV_WINDOW_AUTOSIZE);
    namedWindow("leftHand",
CV_WINDOW_AUTOSIZE);
    namedWindow("rightHand",
CV_WINDOW_AUTOSIZE);

    int key = 0;
    while(key != 27 && key != 'q')
    {
        sensor->waitForDeviceUpdateOnUser();

        memcpy(depthRaw.data, sensor-
>getDepthData(), XRES*YRES*2);
        depthRaw.convertTo(depthShow, CV_8U,
DEPTH_SCALE_FACTOR);

        for(int handI = 0; handI < 2; handI++)
        {
            int handDepth = 0;
            if(sensor->getNumTrackedUsers() > 0)
            {
                Skeleton skel =  sensor-
>getSkeleton(sensor->getUID(0));
                SkeletonPoint hand;

                if( handI == 0)
                    hand = skel.leftHand;
                else
                    hand = skel.rightHand;
                if(hand.confidence == 1.0)
                {
                    handDepth = hand.z *
(DEPTH_SCALE_FACTOR);

        imshow("depthFrame", depthShow);

        if(debugFrames.size() >= 2 )
        {
            resize(debugFrames[0], debugFrames[0],
Size(), 3, 3);
            resize(debugFrames[1], debugFrames[1],
Size(), 3, 3);
            imshow("leftHandFrame",
debugFrames[0]);
            imshow("rightHandFrame",
debugFrames[1]);
            debugFrames.clear();
        }

        key = waitKey(10);

    }

    delete sensor;

    return 0;
}
```