ABSTRACT OF THESIS

RGB-D Cameras' Per-Pixel Calibration and Natural 3D Reconstruction on GPU

Ever since the Kinect brought low-cost depth cameras into consumer market, great interest has been invigorated into Red-Green-Blue-Depth (RGB-D) sensors. Without calibration, a RGB-D camera's horizontal and vertical field of view (FoV) could help generate 3D reconstruction in camera space naturally on graphics processing unit (GPU), which however is badly deformed by the lens distortions and imperfect depth resolution (*depth distortion*). The camera's calibration based on a pinhole-camera model and a high-order distortion removal model requires a lot of calculations in the fragment shader. In order to get rid of both the lens distortion and the *depth distortion* while still be able to do simple calculations in the GPU fragment shader, a novel per-pixel calibration method with look-up table based 3D reconstruction in real-time is proposed, using a rail calibration system. This rail calibration system offers possibilities of collecting infinite calibrating points of dense distributions that can cover all pixels in a sensor, such that not only lens distortions, but *depth distortion* can also be handled by a per-pixel D to Z^W mapping. Instead of utilizing the traditional pinhole camera model, two polynomial mapping models are employed. One is a two-dimensional high-order polynomial mapping from R/C to X^W/Y^W respectively, which handles lens distortions; and the other one is a per-pixel linear mapping from D to Z^W, which can handle *depth distortion*. With only six parameters and three linear equations in the fragment shader, the undistorted 3D world coordinates $(X^{\mathrm{W}}, Y^{\mathrm{W}}, Z^{\mathrm{W}})$ for every single pixel could be generated in real-time. The per-pixel calibration method could be applied universally on any RGB-D cameras. With the alignment of RGB values using a pinhole camera matrix, it could even work on a combination of a random Depth sensor and a random RGB sensor.

KEYWORDS: 3D Reconstruction; Camera Calibration; RGBD; Image Processing

Author's signature: Sen Li

Date: September 1, 2016

RGB-D Cameras' Per-Pixel Calibration and Natural 3D Reconstruction on GPU

By
Sen Li

Director of Dissertation: Dr. Daniel L. Lau

Director of Graduate Studies: Dr. Caicheng Lu

Date: September 1, 2016

RULES FOR THE USE OF THESES

Unpublished theses submitted for the Master's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of parts may be published only with the permission of the author, and with the usual scholarly acknowledgments.

Extensive copying or publication of the thesis in whole or in part also requires the consent of the Dean of the Graduate School of the University of Kentucky.

A library that borrows this thesis for use by its patrons is expected to secure the signature of each user.

Name	Date

THESIS

Sen Li

The Graduate School
University of Kentucky
2016

RGB-D Cameras' Per-Pixel Calibration and Natural 3D Reconstruction on GPU

THESIS

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science in the
College of Engineering
at the University of Kentucky

By
Sen Li
Lexington, Kentucky

Director: Dr. Daniel L. Lau, Professor of Electrical Engineering
Lexington, Kentucky

2016

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1 Introduction

1.1 RGBD Cameras

A Red-Green-Blue-Depth (RGB-D) camera is a sensing system that captures RGB images along with per-pixel depth information. Usually it is simply a combination of a RGB sensor and a depth sensor with an alignment algorithm. Ever since the KinectV1 brought low-cost depth cameras to the consumer market, simple 2D scenes are not able to meet the social demands any more. With the fast development of multimedia technologies, three-dimensional scene display has become a hotspot in the display field. As an extension of classic 2D video, the 3D dynamic display technology can provide a more comprehensive immersive feeling to users than a 2D video. Gesture recognition, 3D modeling, 3D printing, augmented reality, virtual reality, *etc*., a lot of ongoing researches and applications on depth cameras are famous now, cooperated with Human Computer Interaction (HCI) technologies.

The PrimeSense's technology had been originally applied to gaming, with user interfaces based on gesture recognition instead of using a controller (also called Natural User Interface, NUI [8]). It was best known for licensing the hardware design and chip used in Microsoft's first generation of Kinect motion-sensing system for the Xbox 360 in 2010 [9]. The PrimeSense sensor projects an infrared speckle pattern, which will then be captured by an infrared camera in the sensor. A special microchip is employed to compare the captured speckle pattern part-by-part to reference patterns stored in the device, which were captured previously at known depths. The final per-pixel depth will be estimated based on which reference patterns the captured pattern matches best [10]. Other than the first generation of Kinect camera, Asus Xtion PRO sensor, another consumer NUI application product, has also applied the PrimeSense's technology [11].

As a competitor [12] of PrimeSense Structured Light technology, time-of-flight technology had been applied into PMD[Vision] CamCube cameras and 3DV's ZCam cameras. Based on known speed of light, Time-of-Flight (ToF) camera resolves distance by measuring the "time cost" of a special light signal traveling between the camera and target for every single point. The "time cost" variable that ToF camera measures is the phase shift between the illumination and reflection, which will be translated to distance [13]. To detect the phase shifts, a light source is pulsed or modulated by a continuous wave, typically a sinusoid or square wave. The ToF camera illumination is typically from a LED or a solid-state laser operating in the near-infrared range invisible to human eyes. Fabrizio *et al*.

[14] compared the time-of-flight (PMD[Vision] CamCube) camera and PrimeSense (first generation Kinect) camera in 2011. He showed that the time-of-flight technology is more accurate and claimed that the time-of-flight technology will not only be extended to support colours and higher frame sizes, but also rapidly drop in price. In 2010, it was announced that Microsoft would acquire Canesta for an undisclosed amount [15]. And in 2013, Microsoft released the Xbox One, whose NUI sensor KinectV2 features a wide-angle Canesta ToF camera.

Unlike the PrimeSense's speckle pattern or KinectV2's ToF, Intel RealSense camera utilizes stereo vision [16]. Its sensor actually has three cameras: two IR cameras (left and right), and one RGB camera. Additionally, RealSense camera also has an IR laser projector to help the stereo vision recognize depth at unstructured surfaces. Compared with KinectV2 camera, RealSense camera is more like a desktop usage to capture faces or even finger gestures, whereas the KinectV2 could do better to capture the full body actions with all joints [17]. The effective distances of KinectV2 and RealSense hardwares are different. The KnectV2 is optimized to 0.5m 4.5m, while RealSense are designed for 0.2m 1.2m depends on different devices.

1.2 Human Computer Interface

Gesture recognition is one of the hottest sustained research activities in the area of HCI [2]. It has a wide area of application including human machine interaction, sign language, immersive game technology *etc*. Being a significant part in non-verbal communication, hand gestures are playing vital role in our daily life. Hand Gesture recognition system provides us an innovative, natural, user friendly way of interaction with the computer. By keeping in mind the similarities of human hand shape with four fingers and one thumb, Meenakshi [18] presents a real time system for hand gesture recognition on the basis of detection of some meaningful shape based features like orientation, center of mass (centroid), status of fingers, thumb in terms of raised or folded fingers of hand and their respective location in image. Since gestures based on hand and finger movements can be robustly understood by computers by using a special 3D IR camera, users are allowed to play games and interact with computer applications in natural and immersive ways that improve the user experience.

Kam *et al.* [19] developed a real-time gesture-driven human computer interface using the KinectV1 camera and achieved close to 100% practical recognition rates. After Kam, a Kinect-based calling gesture recognition scenario is proposed by Xinshuang *et al.* [1] for taking order service of an elderly care robot. Its proposed scenarios are designed mainly for

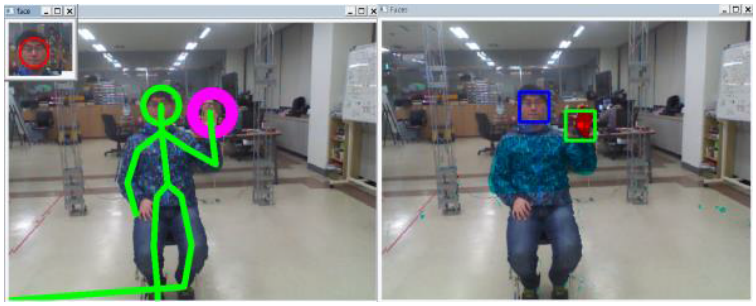


Figure 1.1: Calling Gesture Recognition Using Kinect [1]

helping non expert users like elderly to call service robot for their service request. In order to facilitate elderly service, natural calling gestures are designed to interact with the robot. Figure 1.1 shows the evaluation of gesture recognition when sitting on chair. Individual people is segmented out from 3D point cloud acquired by Microsoft Kinect, skeleton is generated for each segment, face detection is applied to identify whether the segment is human or not, and specific natural calling gestures are designed based on skeleton joints. Dan *et al*. [2] proposed another smart and real-time depth camera based on a new depth generation principle. A monotonic increasing and decreasing function is used to control the frequency and duty-cycle of the NIR illumination pulses. The adjusted light pulses reflect off of the object of interest and are captured as a series of images. A reconfigurable hardware architecture calculates the depth-map of the visible face of the object in real-time from a number of images. The final depth map is then used for gesture detection, tracking and recognition. Figure 1.2 shows an example extraction of hand skeleton. In 2013, Jaehong *et al*. [20] develop and implement a Kinect-based 3D gesture recognition system for interactive manipulation of 3D objects in educational visualization softwares.

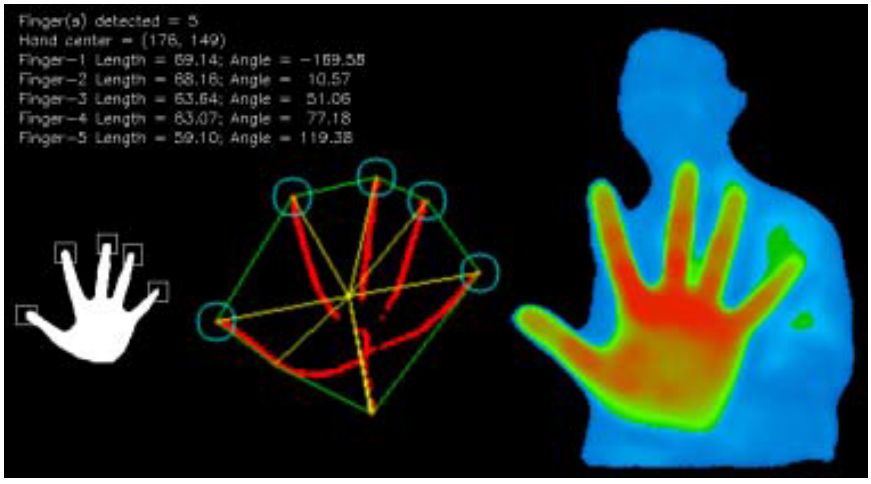


Figure 1.2: Finger Detection using Depth Data [2]

RGB-D cameras own great credits in mobile robotics building dense 3D maps of indoor environments. Such maps have applications in robot navigation, manipulation, semantic mapping, and telepresence. Peter *et al.* [21] present a detailed RGB-D mapping system that utilizes a joint optimization algorithm combining visual features and shape-based alignment. Building on best practices in Simultaneous Localization And Mapping (SLAM) and computer graphics makes it possible to build and visualize accurate and extremely rich 3D maps with RGB-D cameras. Visual and depth information are also combined for view-based loop closure detection, followed by pose optimization to achieve globally consistent maps. SLAM is the process of generating a model of the environment around a robot or sensor, while simultaneously estimating the location of the robot or sensor relative to the environment. SLAM has been performed in many ways, which can be categorized generally by their focus on localization or environment mapping [22]. SLAM systems focused on localizing the sensor accurately, relative to the immediate environment, make use of sparse sensor data to locate the sensor. Using range sensors such as scanning laser range-finders [23], LiDAR and SONAR [24], many robot applications use SLAM systems only to compute the distance from the sensor to the environment. SLAM systems focused on mapping use dense sensor output to create a high-fidelity 3D map of the environment, while using those data to also compute relative location of the sensor [25, 26]. Many modern SLAM algorithms combine both approaches, usually by extracting sparse features from the sensor and using these for efficiently computing the location of the sensor. This position is then used to construct a map from dense sensor data.

With a consumer RGB-D camera providing both color images and dense depth maps at full video frame rate, there appears a novel approach to SLAM that combines the scale information of 3D depth sensing with the strengths of visual features to create dense 3D environment representations, which is called RGB-D SLAM. Felix *et al.* [27] gives an open source approach to visual SLAM from RGB-D sensors, which extracts visual keypoints from the color images and uses the depth images to localize them in 3D. Maohai *et al.* [3] builds an efficient SLAM system using three RGBD sensors. As shown in Fig. 1.3, one Kinect looking up toward the ceiling can track the robot's trajectory through visual odometry method, which provide more accurate motion estimation compared to wheel motion measurement without being disturbed under wheel slippage. And the other two contiguous horizontal Kinects can provide wide range scans, which ensure more robust scan matching in the RBPF-SLAM framework. Also using RGB-D sensor for SLAM, Kathia *et al.* [28] presents a constraint bundle adjustment which allows to easily combine depth and visual data in cost function entirely expressed in pixel. In order to enhance the instantaneity of SLAM for indoor mobile robot, Guanxi *et al.* [4] proposed a RGBD SLAM method based

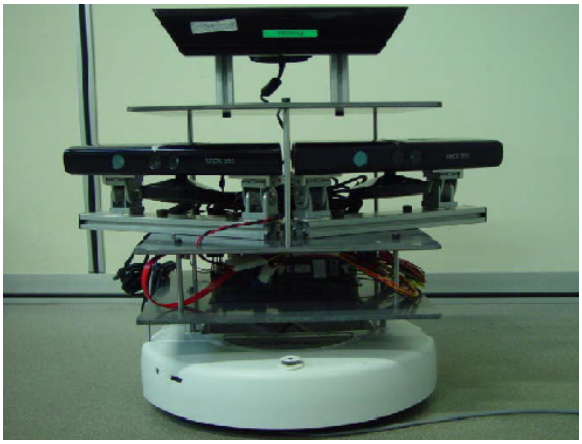
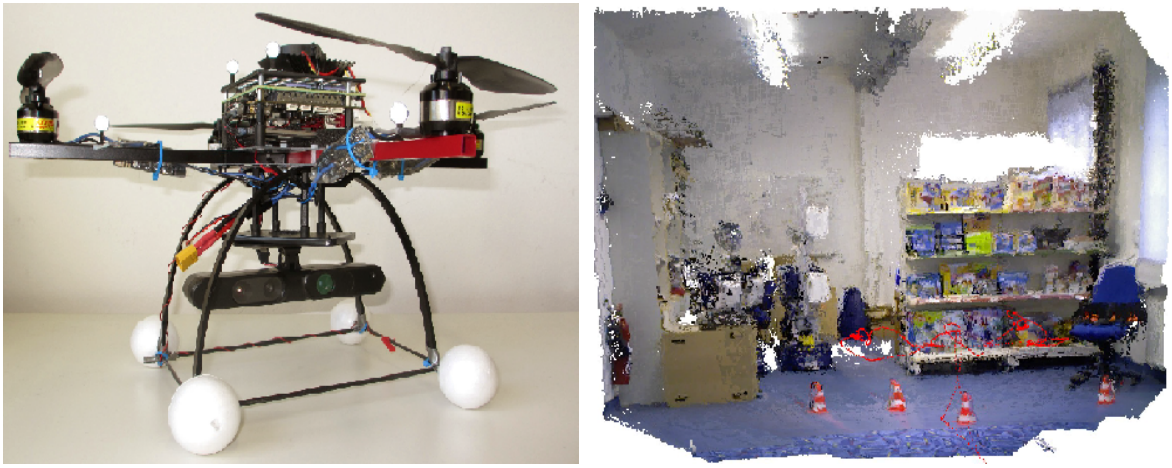


Figure 1.3: SLAM system with Only RGBD Cameras [3]

on Kinect camera, which combined Oriented FAST and Rotated BRIEF (ORB) algorithm with Progressive Sample Consensus (PROSAC) algorithm to execute feature extracting and matching. ORB algorithm which has better property than many other feature descriptors was used for extracting feature. At the same time, ICP algorithm was adopted for coarse registration of the point clouds, and PROSAC algorithm which is superior than RANSAC in outlier removal was employed to eliminate incorrect matching. To make the result more accurate, pose-graph optimization was achieved based on General Graph Optimization (g2o) framework. Figure 1.4 shows the 3D volumetric map of the lab, which can be directly used to navigate robots.

RGB-D camera is also famous in application of doing visual odometry on autonomous flight of a micro air vehicle (MAV), helping acquire 3D models of the environment and es-

Figure 1.4: 3D Map of RGBD-SLAM with ORB and PROSAC [4]



(a) RGBD UAV (b) Reconstruction and Estimated Trajectory

Figure 1.5: RGBD-SLAM for Autonomous MAVs [5]

timate the camera pose with respect to the environment model. Visual odometry generally has unbounded global drift while estimating local motion. To bound estimation error, it can be integrated with SLAM algorithms, which employ loop closing techniques to detect when a vehicle revisits a previous location. A computationally inexpensive RGBD-SLAM solution tailored to the application on autonomous MAVs is discussed by Sebastian and Andreas [5], which enables our MAV to fly in an unknown environment and create a map of its surroundings completely autonomously, with all computations running on its on-board computer. Figure 1.5a shows the MAC with an RGB-D sensor (the first generation of Kinect) mounted. And Fig. 1.5b shows the reconstruction based on the full point clouds, with the estimated trajectory shown in red dots.

RGB-D sensors can be used on a much smaller scale than SLAM to create more detailed, volumetric reconstructions of objects and smaller environments, which opens a new world to the fast 3D printing. 3D printing is an additive technology in which 3D objects are created using layering techniques of different materials, such as plastic, metal, *etc*. It has been around for decades, but only recently is available and famous among the general public. The first 3D printing technology developed in the 1980's was stereolithography (SLA) [29]. This technique uses an ultraviolet (UV) curable polymer resin and an UV laser to build each layer one by one. Since then other 3D printing technologies have been introduced. Nowadays, some companies like iMaterialise or Shapeways offer 3D printing services where you can simply upload your CAD model on-line, choose a material and in a few weeks your 3D printed object will be delivered to your address. This procedure is quite straight-forward when you got your CAD model. However, 3D shape design tends

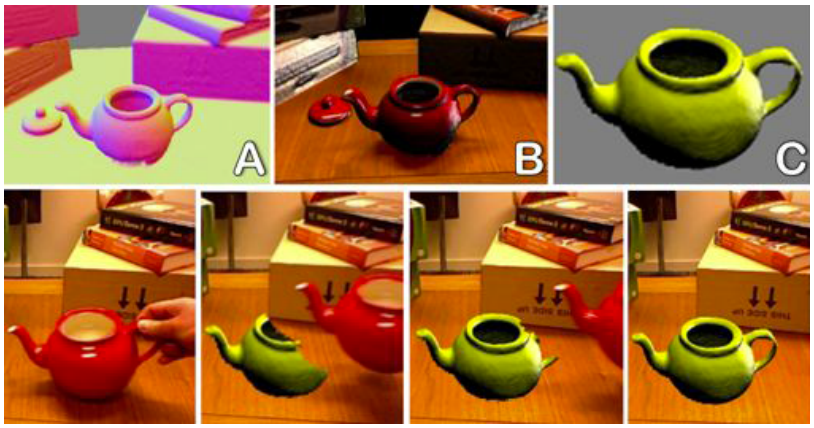


Figure 1.6: Object Segmentation in KinectFusion [6]

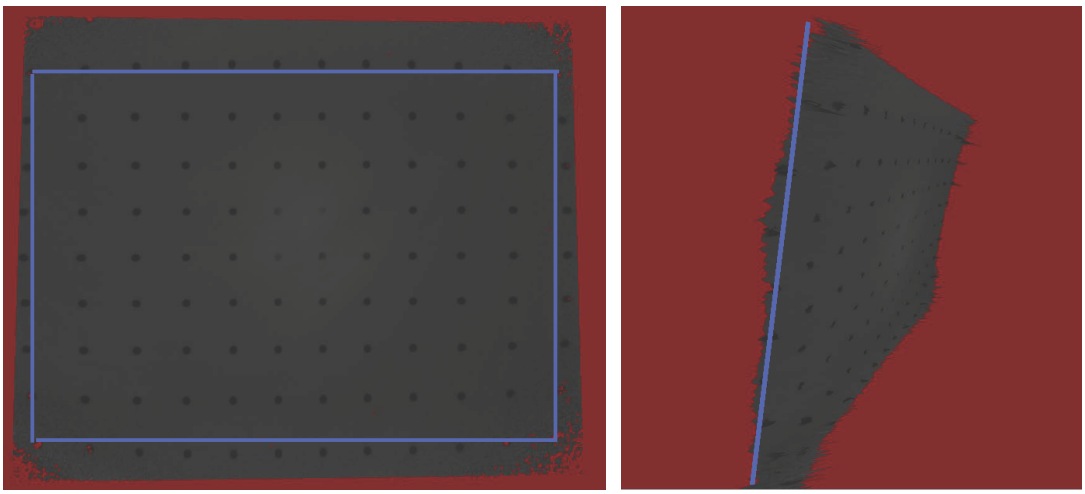
to be a long and tedious process, with the design of a detailed 3D part usually requiring multiple revisions. Fabricating physical prototypes using low cost 3D fabrication technologies at intermediate stages of the design process is now a common practice, which helps the designer discover errors, and to incrementally refine the design [30]. Most often, implementing the required changes directly in the computer model, within the 3D modeling software, is more difficult and time consuming than modifying the physical model directly using hand cutting, caving and sculpting tools, power tools, or machine tools. When one of the two models is modified, the changes need to be transferred to the other model, a process we refer to as synchronization.

KinectFusion, a framework that allows a user to create a detailed 3D reconstruction of an object or a small environment in real-time using Microsoft Kinect sensor, has garnered a lot of attention in the reconstruction and modeling field. It enables a user holding and moving a standard Kinect camera to rapidly create detailed 3D reconstructions of an indoor scene [6]. Not only an entire scene, a specific smaller physical object could also be cleanly segmented from the background model simply by moving the object directly. Figure 1.6 shows how the interested object (a teapot) is accurately segmented from the background by physically removed. The sub-figure (A) shows surface normals, and sub-figure (B) is the texture mapped model. Nadia *et al*. [31] proposed and introduced a from-Sense-to-Print system that can automatically generate ready-to-print 3D CAD models of objects or humans from 3D reconstructions using the low-cost Kinect sensor. Further, Ammar and Gabriel [30] addresses the problem of synchronizing the computer model to changes made in the physical model by 3D scanning the modified physical model, automatically detecting the changes, and updating the computer model. A new method is proposed that allows the designer to move fluidly from the physical model (for example his 3D printed object, or his carved object) to the computer model. In the proposed process the physical modification

applied by the designer to the physical model are detected by 3D scanning the physical model and comparing the scan to the computer model. Then the changes are reflected in the computer model. The designer can apply further changes either to the computer model or to the physical model. Changes made to the computer model can be synchronized to the physical model by 3D printing a new physical model.

1.3 RGB-D Cameras' Calibration and 3D Reconstruction on GPU

As discussed above, applications like RGBD-SLAM and KinectFusion apply 3D reconstruction techniques using an RGBD camera, in which an RGB sensor offers color values and a depth sensor measures objects' distances (D or Z^C). RGBD cameras, e.g. KinectV2, offer the horizontal and vertical field of view (FoV)s of the sensors based on a pinhole camera model, from which a proportional per-pixel beam equation (from Z^C to X^C/Y^C) could be derived. It is not hard to do 3D reconstruction in camera space naturally on GPU with the help of the proportional per-pixel beam equations, however, the 3D reconstructed image in that case will be deformed a lot by distortions. Figure 1.7 shows the KinectV2 NearIR 3D reconstruction in camera space, when observing a canvas hung on a flat wall printed with uniform grid ground-dots pattern. In the front view, a blue rectangle is drawn based on four corner dot-clusters, which reflects lens distortions (the uniformed distribution of the captured dot-clusters). While in the side view, a blue straight line added on the side of 3D reconstruction, which shows the unflatness of captured "flat wall". The deformation in the side view is probably caused by the various resolutions of depth sensor on per-pixel



(a) Front View (b) Side View

Figure 1.7: KinectV2 NearIR 3D Reconstruction in Camera Space

basis, which we will call as *depth distortion* in this thesis. In order to get undistorted 3D images, camera calibration is necessary before a camera being employed.

Camera calibration usually use calibration objects, which could be assigned world space coordinates ($X^W/Y^W/Z^W$) to help remove distortions. For decades, much work on camera calibration has been done, starting from the photogrammetry community [32, 33], to computer vision ([34, 35, 36] to cite a few). And the combination of a pinhole-camera matrix with a distortion removal vector (which contains five high-order polynomial parameters) are widely known as important tools in camera calibration. However, there needs to be a lot calculations in the GPU fragment shader based on those parameters from both pinhole-camera matrix and distortion removal vector. And we would like to find a simple method with fewer calculations when generating the 3D coordinates. Similar with the per-pixel *proportional* beam equations in camera space reconstruction on GPU, Kai [37] derived more common *linear* beam equations (from X^W to Y^W/Z^W) directly from the pinhole-camera matrix, on the basis of per-pixel. That *linear* beam equations make it possible to show world space 3D reconstruction naturally on GPU, but it did not contains infos about lens distortion correction.

Our goal is to draw undistorted 3D reconstruction on GPU with the fewest calculations. Inspired by Kai, we will build up a rail calibration system to support the per-pixel D to Z^W mapping, such that Kai's per-pixel X^W to Y^W/Z^W linear mappings could be applied during the 3D reconstruction on GPU. We will call this new method *per-pixel calibration*. As shown in Fig. 1.8, the camera observing the uniform grid dots pattern is mounted on a rail, which is perpendicular to pattern on the wall. A laser distance measurer will be used to supply accurate per-frame Z^W, so that the per-pixel D to Z^W mapping could handle *depth distortion*. As long as the undistorted dense X^W/Y^W could be acquired, we will be able determine the parameters of per-pixel *linear* beam equations.

Undistorted world space coordinates $X^W/Y^W/Z^W$ with D together will be collected and saved onto local drives, during which lens distortions will be removed. Instead of using the combination of pinhole-camera matrix and distortion removal vector, we will determine a best-fit two–dimensional high-order polynomial mapping that can directly map from *Row* and $Column$ in image space to X^W and Y^W, and the high-order polynomial mapping will handle the lens distortions. After the data collection, we will determine a best-fit mapping model between per-pixel D and Z^W, and then process the collected data, and finally generate per-pixel mapping parameters, which can make up a look-up table that will help draw undistorted 3D reconstruction on GPU in real-time.

In Chapter 2, a pinhole-camera-model based calibration method is discussed in detail, including the lens distortions analysis and its removal. Chapter 3 will introduce how to

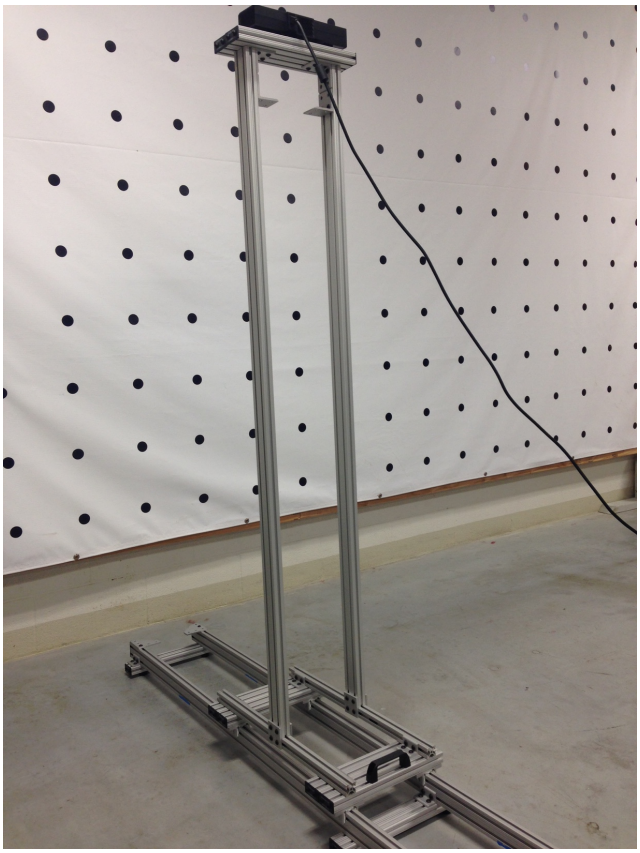


Figure 1.8: KinectV2 Calibration System

draw the camera space 3D reconstruction on GPU, introduce a rail calibration system's set-up, and then talk about the proposed per-pixel calibration method and simple 3D reconstruction on GPU in detail. Chapter 4 will explain how the two polynomial mapping models in the proposed calibration method are determined, and then show the calibrated results about how well the lens distortions and *depth distortion* are corrected. Chapter 5 will conclude this thesis and talk about the future work of RGB-D cameras calibration.

Chapter 2 Traditional RGB-D Cameras Calibration

A pinhole-camera model can be used to describe an image sensor's field of view. When applying the pinhole camera model in world space, it explains the mapping relationship from world space to camera space, and then to image space. A 3×4 pinhole-camera matrix expresses the mappings mathematically. It consists of an intrinsic matrix mapping from the 3D camera space to 2D image space, and an extrinsic matrix map from 3D world space to 3D camera space. Traditionally, lens distortions correction is after, and separated from the pinhole-camera model calibration. In this chapter, we will introduce the camera calibration methods based on the pinhole-camera model in detail, and then discuss how to remove the lens distortions in traditional methods.

2.1 Pinhole Camera

A pinhole camera is a simple optical imaging device in the shape of a closed box or chamber. A pinhole camera is completely dark on all the other sides of the box including the side where the pin-hole is created. Figure 2.1 shows an inspection of a pinhole camera. In its front is a pin-hole that help create an image of the outside space on the back side of the box. When the shutter is opened, the light shines through the pin-hole and imprint an image onto a sensor (or photographic paper, or film) placed at the back side of the box. In order to analyze parameters like focal distance, field of view, etc., pinhole camera has its own three dimensional space (noted as X^c, Y^c, and Z^c). Note that, according to Cartesian Coordinates "right hand" principle, the camera is looking down the negative of Z^c-axis, given X^cY^c directions as shown in the figure. Its focal length of the pinhole camera is the

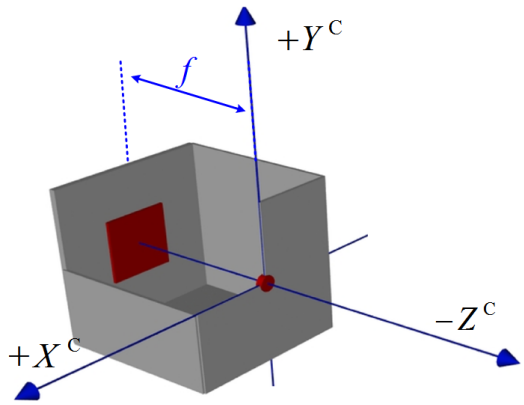


Figure 2.1: The Pinhole Camera Inspection

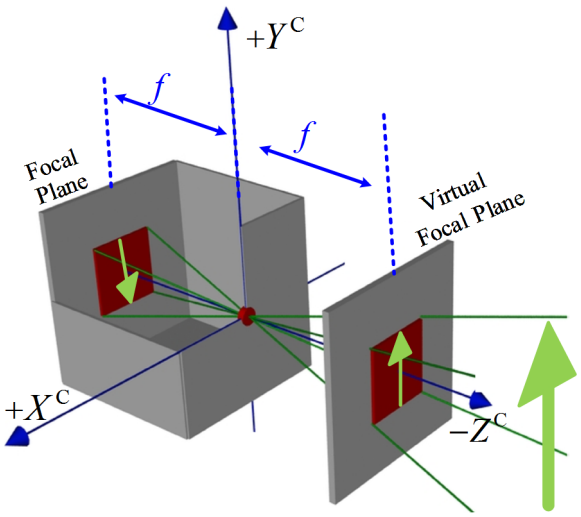


Figure 2.2: Virtual Focal Plane of a Pinhole Camera

distance on the Z^c-axis, between the pinhole at the front of the camera and the paper or film at the back of the camera.

Pinhole cameras are characterized by the fact that they do not have a lens. It rely on the fact that light travels in straight lines, which is a principle called the rectilinear theory of light. This makes the image appear upside down in the camera, as shown in Fig. 2.2. Tracing the corners of the camera sensor through the pin hole, those dark green lines show the limits of the field of view in 3D coordinate space. The back side plane of the pinhole camera, which is behind the origin at a positive Z^C-axis and also where our sensor sits, is also called the focal plane. It is not intuitive, nor convenient for mathematical analysis that the images on the focal plane are always upside down. So a virtual focal plane is defined in front of the pinhole on the negative Z^C-axis, which is equal distant from the focal point (pin hole) as the actual focal plane is behind. Notice that the limits of the field of view intersect with the virtual focal plane at the four corners of the up-right image just as they disseminate from the four corners of the sensor at the real focal plane.

With the virtual focal plane, the camera body with the real focal plane could be removed. And the rest parts in front of the the camera body, the focal point and the virtual focal plane together, form the most common pinhole camera model. In order to employ this model to analyze arbitrary 3D object points inside the camera's field of view in 2D image space, the prior step is to define the relationship between points in 3D camera space and the 2D image space (*row* and *column*). As shown in Fig. 2.3, The focal point is right at the origin of the camera 3D space coordinates, from where to the sensor is the vertical distance of f, the focal distance. The 2D image coordinates are in dark green, and its origin is sitting at the up-left corner of the sensor. Only the a virtual sensor (in color red) is visible on the

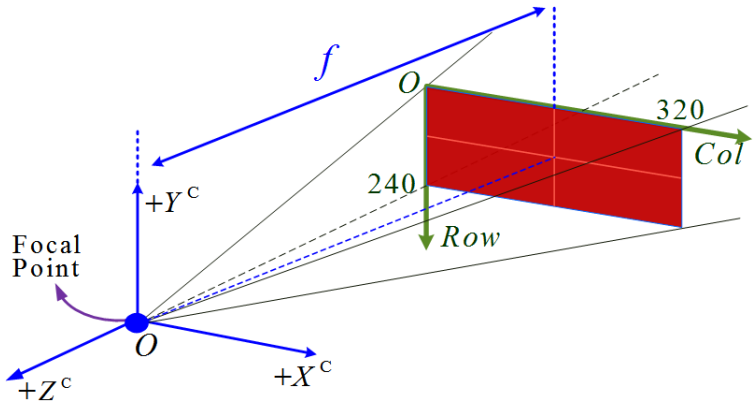


Figure 2.3: Common Pinhole Camera Model

virtual focal plane, whose size in 2D image space is noted as 240 by 320 (using the size of PrimeSense camera). As long as both of the camera 3D space coordinates and image 2D space coordinates are defined, the next job is to build a mapping between them. Note that, the range of image should be either ([0:239], [0:319]) or ([1:240], [1:320]).

Select a random object point P^C in the camera space located at camera 3D coordinates (X_C, Y_C, Z_C). A line passing both of the point P^C and the focal point intersects with the virtual focal plane at P^I, with its image 2D coordinates (R, C). To determine the mapping function, we can start a the proportional relationship. As shown in Fig. 2.4, the center point in the image coordinates, which is usually called *principle point*, could be determine by column of half-width and row of half-height. Concretely, the principle point (R_h, C_h) is either (119.5, 159.5) if range is ([0:239], [0:319]), or (120, 320) if range is ([1:240],

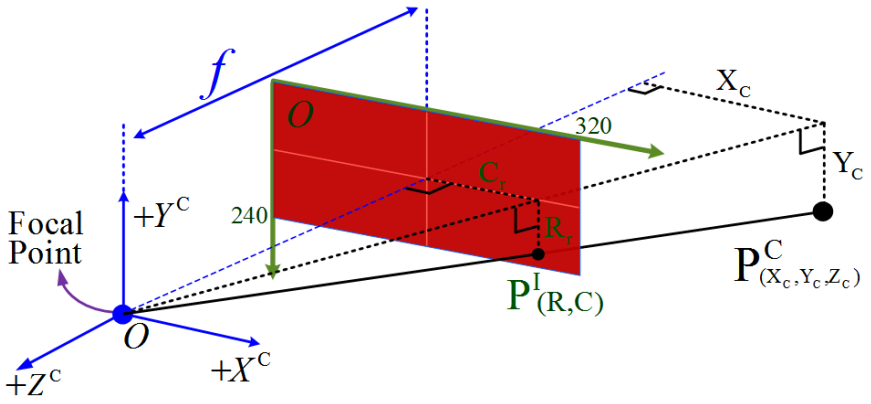


Figure 2.4: Mapping from Camera Space to Image Space

[1:320]). So, we could get the relative row and column distance of R_r and C_r by:

$$\begin{aligned} R_r &= R - R_h \\ C_r &= C - C_h \quad . \end{aligned} \tag{2.1}$$

Based on by triangulation, it is straight forward to tell the proportional relationship between f/Z_C and C_r/X_C, R_r/Y_C. Thus we get

$$\begin{bmatrix} C_r \\ R_r \end{bmatrix} = f \begin{bmatrix} X_C/Z_C \\ Y_C/Z_C \end{bmatrix}. \tag{2.2}$$

And by changing the relative distance R_rC_r back to the 2D image coordinates (R, C), then eqn. (2.2) will be written as

$$\begin{bmatrix} C \\ R \end{bmatrix} = f \begin{bmatrix} X_C/Z_C \\ Y_C/Z_C \end{bmatrix} + \begin{bmatrix} C_h \\ R_h \end{bmatrix}. \tag{2.3}$$

If written in homogeneous coordinates, we will get eqn. (2.4):

$$Z_C \begin{bmatrix} C \\ R \\ 1 \end{bmatrix} = \begin{bmatrix} fX_C \\ fY_C \\ Z_C \end{bmatrix} + \begin{bmatrix} Z_CC_h \\ Z_CR_h \\ 0 \end{bmatrix} = \begin{bmatrix} f & 0 & C_h \\ 0 & f & R_h \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_C \\ Y_C \\ Z_C \end{bmatrix}. \tag{2.4}$$

Till Now, we haven't consider the units translation between the camera 3D space the image 2D space. The random object point P^C's mapping point P^I (R, C) on the image space is expressed in millimeters (or inches). Since it is necessary to express the image space coordinates (R, C) in pixels, we need to find out the resolution of the sensor in pixels/millimeter. Considering that, the pixels are not necessarily be square-shaped, we assume they are rectangle-shaped with resolution α_c and α_r pixels/millimeter in the *Col* and *Row* direction respectively. Therefore, to express P^I in pixels, its C and R coordinates should be multiplied by α_c and α_r respectively, to get:

$$\begin{bmatrix} Z_CC \\ Z_CR \\ Z_C \end{bmatrix} = \begin{bmatrix} f\alpha_cX_C \\ f\alpha_rY_C \\ Z_C \end{bmatrix} + \begin{bmatrix} Z_C\alpha_cC_h \\ Z_C\alpha_rR_h \\ 0 \end{bmatrix} = \begin{bmatrix} \alpha_cf & 0 & \alpha_cC_h \\ 0 & \alpha_rf & \alpha_rR_h \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_C \\ Y_C \\ Z_C \end{bmatrix} = KP^C. \tag{2.5}$$

Note that K only depends on the intrinsic camera parameters like its focal length, resolution in pixels, and sensor's width and height. Thus, the mapping matrix K is also called a cam-

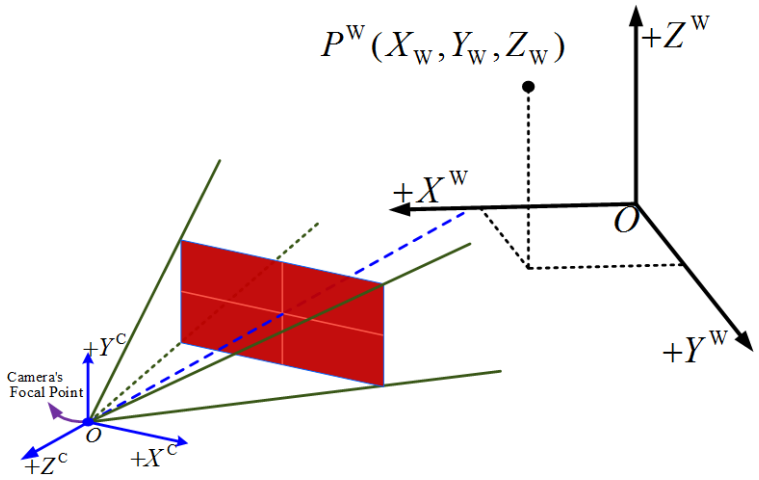


Figure 2.5: Pinhole Camera in World Space

era's intrinsic matrix. Considering that the pixels might be parallelogram-shaped instead of rigid rectangle-shaped (when the image coordinate axis *Row* and *Col* are not orthogonal to each other), usually K has a skew parameter s, given by

$$K = \begin{bmatrix} f_c & s & t_c \\ 0 & f_r & t_r \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.6}$$

where $f_c = \alpha_c f$ and $f_r = \alpha_r f$ are the focal length in pixels on the *Col* and *Row* directions respectively, $t_c = \alpha_c \mathrm{R_h}$ and $t_r = \alpha_r \mathrm{R_h}$ are the translation parameters that help move the origin of image coordinate to the principle point.

Now we have K, which helps map between camera 3D space and image 2D space. But we are still not able to employ it yet. The camera 3D space is with respect to the camera sensor only. Neither can we directly tell the camera 3D coordinates of an object point, nor can we assign it. All we can do is to use the camera space as an intermediate space between the image coordinates and world coordinates, which we could assign by ourselves.

Figure 2.5 shows a pinhole camera observing an arbitrary object point P in the world space. We assign the world coordinates so that the object point has world space coordinates $P^W(X_w, Y_w, Z_w)$. Although the world space and camera space are two different spaces, we could easily transform between each other through rotation and translation, as long as both of the spaces are using rigid Cartesian Coordinates. With a standard rotation matrix R_{3*3}

and a translation matrix T_{3*1}

$$R_{3*3} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, T_{3*1} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}, \tag{2.7}$$

we can get the transformation matrix $[R_{3*3}\ T_{3*3}]$ from the world space to camera space:

$$\begin{bmatrix} X^C \\ Y^C \\ Z^C \end{bmatrix} = R \begin{bmatrix} X^W \\ Y^W \\ Z^W \end{bmatrix} + T = \begin{bmatrix} R_{3*3} & T_{3*1} \end{bmatrix} \begin{bmatrix} X^W \\ Y^W \\ Z^W \\ 1 \end{bmatrix}. \tag{2.8}$$

The parameters that help map from world space to camera space depend on how we assign the world coordinates. Since none of them are from the camera even though they are belongs to an important part of camera calibration, usually the matrix $[R_{3*3}\ T_{3*3}]$ is called extrinsic camera matrix. With both of the extrinsic camera matrix (help map from world space to camera space) and the intrinsic camera matrix (help map from camera space to image space), we are now able to build the connection between the world space coordinates, which could be assigned by ourselves, and the image space *Row* and *Column*, which are the streams we retrieved from the camera.

To combine the intrinsic camera matrix and extrinsic camera matrix (combine eqn. (2.5) and eqn.(2.8)), we get

$$Z^C \begin{bmatrix} C \\ R \\ 1 \end{bmatrix} = K \begin{bmatrix} X^C \\ Y^C \\ Z^C \end{bmatrix} = K \begin{bmatrix} R_{3*3} & T_{3*1} \end{bmatrix} \begin{bmatrix} X^w \\ Y^w \\ Z^w \\ 1 \end{bmatrix} = M \begin{bmatrix} X^w \\ Y^w \\ Z^w \\ 1 \end{bmatrix}, \tag{2.9}$$

where:

$$M = K \begin{bmatrix} R_{3*3} & T_{3*1} \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix}. \tag{2.10}$$

Note that, although Z^C values can be retrieved from the depth sensor streams, they will be employed during the calculation of M, because they will be expressed by the third row parameters in matrix M. Z^C will only be used in the step of 3D reconstruction after the pinhole camera matrix M is determined, as will be discussed in details in Section 2.2. Thus, Z^C in eqn. (2.9) is commonly substituted as an intermediate parameter k. We did

not change Z^C for the consistency of derivations. To inspect the pinhole camera matrix M, it is composed of rotation/translation matrix for 3D space transforming and intrinsic perspective matrix for handling both of perspective view mapping and shape-skewing, all of which belong to linear processing. In other words, this 3×4 transformation matrix is specially for handling perspective view, or perspective distortion. The pinhole camera model is based on the homogeneous coordinates, which means its matrix M is also limited by linear processing.

2.2 3D Camera Calibration

The calibration of a 3D camera aims to be able to generate the world coordinates (X^W, Y^W, Z^W) and corresponding *RGB* values for every single pixel, given the depth steams and RGB streams retrieved from the 3D camera. From Section 2.1, we know that the pinhole camera matrix M (eqn. (2.10)) could help map from the world space to image space; however not able to directly transform image space data to world space coordinates. In order to determine $X^W/Y^W/Z^W$ (based on eqn.(2.5) and eqn.(2.8)), both of the intrinsic camera matrix and extrinsic camera matrix are needed, both of which are intermediate parameters and practically can only be determined through matrix M. Thus, the first job for 3D camera calibration is to solve the pinhole camera matrix M.

To solve the pinhole camera matrix, we can use least squares fit with known 3D points (X^W, Y^W, Z^W) and their corresponding image points (R,C). With one point, based on eqn. (2.10) and 2.9, we can get two equations.

$$\begin{aligned} m_{11}X^W + m_{12}Y^W + m_{13}Z^W + m_{14} - m_{31}X^W C - m_{32}Y^W C - m_{33}Z^W C - m_{34}C = 0 \\ m_{21}X^W + m_{22}Y^W + m_{23}Z^W + m_{24} - m_{31}X^W R - m_{32}Y^W R - m_{33}Z^W R - m_{34}R = 0 \end{aligned} \tag{2.11}$$

There are totally 12 unknowns to solve, thus we need at least six points to solve the 3×4 pinhole camera matrix M. Using n-points least squares to solve the best fit, we can build a

$2n$ equations matrix, given by eqn. (2.12).

$$\begin{bmatrix} X_1^W & Y_1^W & Z_1^W & 1 & 0 & 0 & 0 & 0 & -X_1^W C_1 & -Y_1^W C_1 & -Z_1^W C_1 & -C_1 \\ 0 & 0 & 0 & 0 & X_1^W & Y_1^W & Z_1^W & 1 & -X_1^W R_1 & -Y_1^W R_1 & -Z_1^W R_1 & -R_1 \\ X_2^W & Y_2^W & Z_2^W & 1 & 0 & 0 & 0 & 0 & -X_2^W C_2 & -Y_2^W C_2 & -Z_2^W C_2 & -C_2 \\ 0 & 0 & 0 & 0 & X_2^W & Y_2^W & Z_2^W & 1 & -X_2^W R_2 & -Y_2^W R_2 & -Z_2^W R_2 & -R_2 \\ & & & & & & & \vdots & & & & \\ X_n^W & Y_n^W & Z_2^n & 1 & 0 & 0 & 0 & 0 & -X_n^W C_n & -Y_n^W C_n & -Z_n^W C_n & -C_n \\ 0 & 0 & 0 & 0 & X_n^W & Y_n^W & Z_n^W & 1 & -X_n^W R_n & -Y_n^W R_n & -Z_n^W R_n & -R_n \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \tag{2.12}$$

Considering that this matrix is build on homogeneous system, there is no unique solution. There can always be a total-zeros solution. To make the solution unique, we select $m_{34} = 1$, so that the homogeneous eqn. (2.12) could be changed into an inhomogeneous format like $AX = B$, where the known matrix A is a $2n \times 11$ matrix and known matrix B is a $2n$ vector:

$$\begin{bmatrix} X_1^W & Y_1^W & Z_1^W & 1 & 0 & 0 & 0 & 0 & -X_1^W C_1 & -Y_1^W C_1 & -Z_1^W C_1 \\ 0 & 0 & 0 & 0 & X_1^W & Y_1^W & Z_1^W & 1 & -X_1^W R_1 & -Y_1^W R_1 & -Z_1^W R_1 \\ X_2^W & Y_2^W & Z_2^W & 1 & 0 & 0 & 0 & 0 & -X_2^W C_2 & -Y_2^W C_2 & -Z_2^W C_2 \\ 0 & 0 & 0 & 0 & X_2^W & Y_2^W & Z_2^W & 1 & -X_2^W R_2 & -Y_2^W R_2 & -Z_2^W R_2 \\ & & & & & & & \vdots & & & \\ X_n^W & Y_n^W & Z_2^n & 1 & 0 & 0 & 0 & 0 & -X_n^W C_n & -Y_n^W C_n & -Z_n^W C_n \\ 0 & 0 & 0 & 0 & X_n^W & Y_n^W & Z_n^W & 1 & -X_n^W R_n & -Y_n^W R_n & -Z_n^W R_n \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \end{bmatrix} = \begin{bmatrix} C_1 \\ R_1 \\ C_2 \\ R_2 \\ \vdots \\ C_n \\ R_n \end{bmatrix}. \tag{2.13}$$

Using pseudo inverse, eqn. (2.13) can be solved by $X = (A^T A)^{-1} A^T B$, where X is an 11-elements vector and $X(1) \sim X(11)$ correspond to $m_{11} \sim m_{33}$. And the 3×4 pinhole camera

matrix (eqn. (2.10)) will be solved as:

$$M = \begin{bmatrix} X(1) & X(2) & X(3) & X(4) \\ X(5) & X(6) & X(7) & X(8) \\ X(9) & X(10) & X(11) & 1 \end{bmatrix} \tag{2.14}$$

After we get the perspective projection matrix M, the next step is to recover the intrinsic and extrinsic camera matrix K and $[R_{3*3}, T_{3*1}]$, with which we could generate the world coordinates $X^W/Y^W/Z^W$.

Starting from the decomposition of eqn. (2.10) step by step:

$$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} & \\ m_{21} & m_{22} & m_{23} & O_{3*1} \\ m_{31} & m_{32} & m_{33} & \end{bmatrix} + \begin{bmatrix} & & & m_{14} \\ & O_{3*3} & & m_{24} \\ & & & m_{34} \end{bmatrix}, \tag{2.15}$$

$$K[R_{3*3}\ T_{3*1}] = \begin{bmatrix} KR_{3*3} & O_{3*1} \end{bmatrix} + \begin{bmatrix} O_{3*3} & KT_{3*1} \end{bmatrix} \tag{2.16}$$

and

$$M_{3*3} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} = KR_{3*3} \tag{2.17}$$

where O denotes zero matrices with their sizes noted by subscripts. From eqn. (2.7), we know that R_{3*3} is a standard rotation matrix, which has its property of orthogonal. Also from eqn. (2.6), we know that K is an upper triangular matrix. Thus, all of the above fit in the prerequisites of RQ decomposition, which is a technique that could help us decompose the M_{3*3} into the upper triangular intrinsic matrix K and rotation matrix R_{3*3}. After we got R_{3*3}, the translation matrix T_{3*1} could be determined with eqn. (2.16).

Now we find the way to determine both of the intrinsic camera matrix and the extrinsic camera matrix. With depth streams measuring Z^C, we are able to transform the 2D image data retrieved from the camera into 3D camera space point cloud by eqn. (2.5), and then generate the world space point cloud by eqn. (2.8). The basic pinhole camera model calculation is widely used in various camera calibration techniques. Based on different calibration systems, Zhengyou [36] classified those calibration techniques into four categories: unknown scene points in the environment (self-calibration), 1D objects (wand with dots), 2D objects (planar patterns undergoing unknown motions) and 3D apparatus (two or three planes orthogonal to each other).

Self-calibration technique do not use any calibration object, and can be considered as zero-dimension approach because only image point correspondences are required. Just by moving a camera in a static scene, the rigidity of the scene provides in general two constraints [38] on the cameras' internal parameters from one camera displacement by using image information alone. Therefore, if images are taken by the same camera with fixed internal parameters, correspondences between three images are sufficient to recover both the internal and external parameters which allow us to reconstruct 3-D structure up to a similarity [39, 40].

One-Dimension points-line calibration employs one dimension objects composed of a set of collinear points. With much lower cost than two dimensional or even three dimensional calibration system, using one dimension objects in camera calibration is not only a theoretical aspect, but is also very important in practice especially when multi-cameras are involved in the environment. To calibrate the relative geometry between multiple cameras, it is necessary for all involving cameras to simultaneously observe a number of points. It is hardly possible to achieve this with 3D or 2D calibration apparatus if one camera is mounted in the front of a room while another in the back. This is not a problem for 1D objects. Xiangjian [41] shows how to estimate the internal and external parameters using one dimensional pattern in the camera calibration. And Zijian [42] employed one dimensional objects as virtual environments in practical multiple cameras calibration.

Two and Three dimensional objects calibration systems usually give better calibrations. P. F. Sturm *et al*. [43] presented a general algorithm for plane-based calibration that can deal with arbitrary numbers of views that observe a planar pattern shown at different orientations, so that almost anyone can make such a calibration pattern by him/her-self, and the setup is very easy. Both of Matlab and OpenCV have applied this two dimension plane calibration method in their applications. Zhengdong [44] compared this two dimension plane camera calibration method and self-calibration method. Hamid [45] applied this method into practical calibration and employed the calibrated camera into camera pose estimation and distance estimation application.

In three-dimensional object calibration technique, camera calibration is performed by observing a calibration object whose geometry in 3D space is known for very good precision. Calibration can be done very efficiently [35]. The calibration objects usually consist of two or three planes orthogonal to each other. Paul [46] applied the three dimension object calibration in his PHD project. Mattia [7] wrote a detailed tutorial from building the 3D object (Fig. 2.6) for calibration, to scanning using the calibrated camera. Figure 2.7a shows how six points are selected for calibration and Fig. 2.7b shows the 3D reconstruction after calibration.

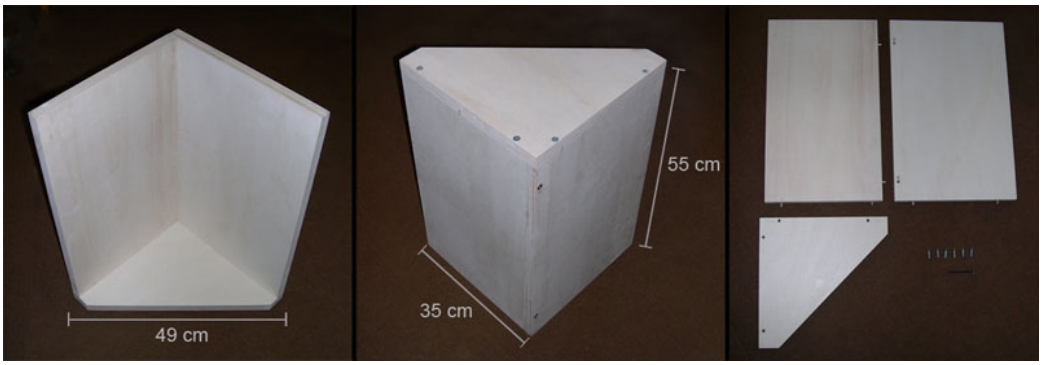
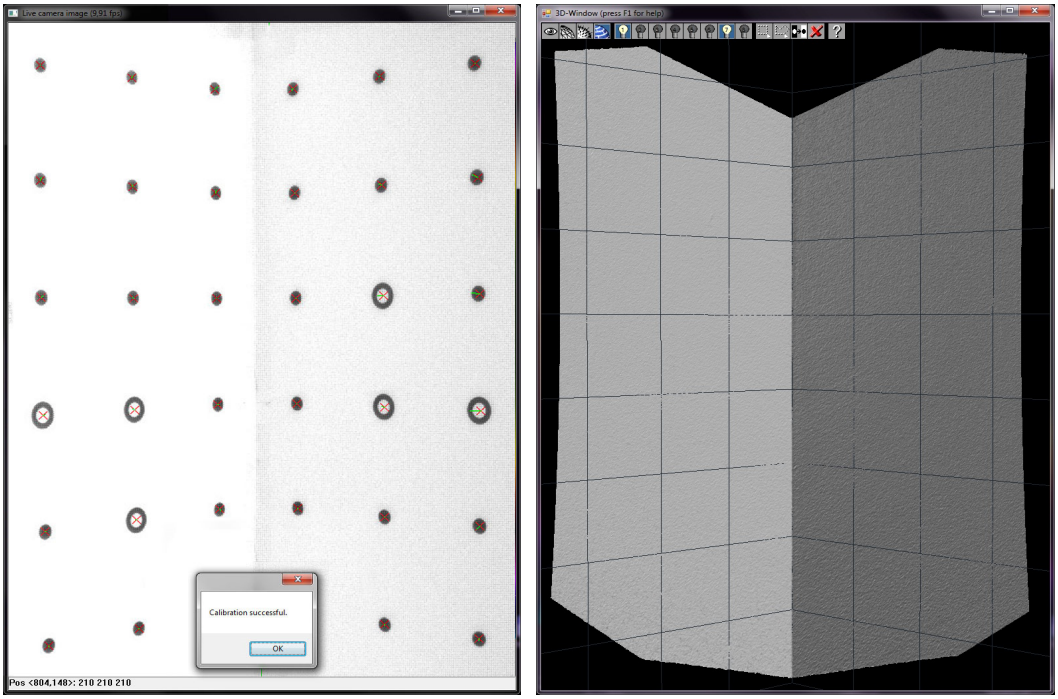


Figure 2.6: Building 3D Calibration Object [7]



(a) Six Points to Calibrate (b) Reconstruction After Calibration

Figure 2.7: Three Dimension Object Camera Calibration [7]

Zhengyou Zhang, a Chinese professor of computer science, IEEE and ACM Fellow and a specialist in computer vision and graphics, has deep studies on camera calibration from one-dimension calibration to tree-dimension calibration [47, 48, 36]. The accuracy of calibration from 1D to 3D is getting better, but the calibration system setting-up needs more and more work and cost as well. One dimension object is suitable for calibrating multiple cameras at once. Two dimension planer pattern approaches seems to be a good compromise, with good accuracy and simple setup. Also using the three dimension method for calibration, Kai [37] derived the per-pixel beam equation, the linear relationship that

could map to X^W/Y^W from Z^W as eqn. (2.18) shows, directly from pinhole camera matrix M. That is to say, we could easily look up X^W/Y^W after calibration once found the way to get Z^W.

$$\begin{aligned} X^W[row, col] &= a[row, col]Z^W[row, col] + d[row, col] \\ Y^W[row, col] &= c[row, col]Z^W[row, col] + d[row, col] \end{aligned} \tag{2.18}$$

where $a/b/c/d$ are per-pixel coefficients for the linear beam equations, and the subscripts $[row, col]$ are corresponding pixel address in image space.

2.3 Lens Distortion

All above in Chapter 2 are talking about the ideal pinhole camera, without lenses. Whereas in practice, as a result of several types of imperfections in the design and assembly of lenses composing the camera optical system, there are always lens distortions for a camera, and the expressions in eqn. (2.2) are not valid any more. Lens distortion could be classified into two groups [49] : radial distortion and tangential distortion. Imperfect lens shape causes light rays bending more near the edges of a lens than they do at its optical center. Barrel distortions happen commonly on wide angle lenses, where the field of view of the lens is much wider than the size of the image sensor [50]. Improper lens assembly will lead to tangential distortion, which occurs when the lens and the image plane are not parallel. Fig. 2.8 shows how radial distortion d_r and tangential distortion d_t affect the object point position in the image. Note that both of radial distortion and tangential distortion are with

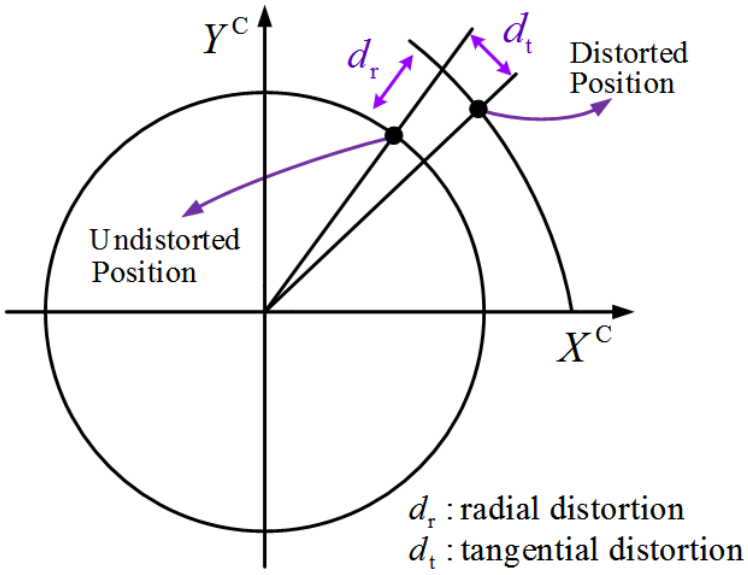


Figure 2.8: Radial and Tangential Distortion Affection In Image Space

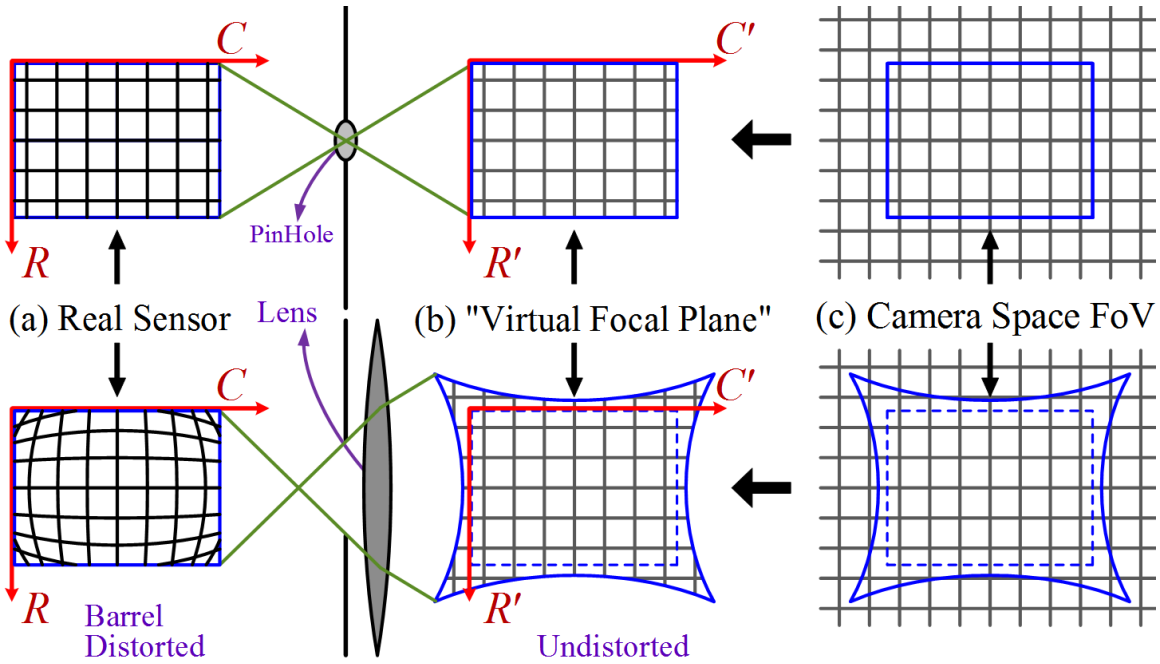


Figure 2.9: From Camera Space to Image Space with Lens Distortions

respect to image space row and column, and what we will take later is negative distortion instead of positive. Distortions are present because the field of view (FoV) in camera space has been affected by the lens. For most consumer RGB-D cameras with cheap lens, their distortions are usually barrel distortions (negative distortion) resulted by the enlarged field of view in the camera space, because the larger view was squeezed into the sensor. Figure 2.9 intuitively shows how the lens enlarged the field of view of in the camera space and then generates the barrel distortions.

There are (a)(b)(c) three parts shown in Fig. 2.9. Each part has the pinhole camera only on the top, in contrast to the camera-with-lens situation at the bottom. To understand how the barrel distortion happens, we should go through from part (c) to part (a). In part (c), the gray background uniform grid is the "object" our that the camera is going to observe, and the blue frames shows the FoV of the camera in the camera space. Due to the fact that, there will be worse and worse distortions as one pixel goes from the center to the edge, the enlarged FoV of a camera with lens in the camera space is in pincushion (or star) shape. With the enlarged FoV is mapped to the "Virtual Focal Plane", as defined in Fig. 2.2, the pincushion shape doesn't change because rays from the camera space have not gone through the lens yet. Note that we quoted the "Virtual Focal Plane" because the image on this virtual plane, when considering lens distortion, does not equal to the real focal plane (where the sensor is) any more. We can tell from part (b) that, even though the image space coordinates still are composed of *Col* and *Row*, their ranges have changed from positive

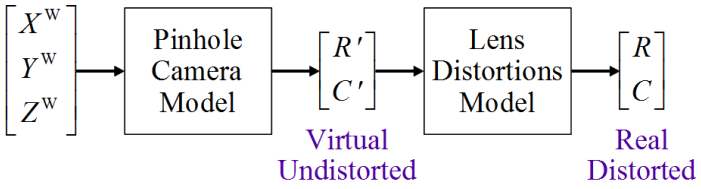


Figure 2.10: Traditional Camera Calibration Flow Chart

integers only to the whole real integers that include negative ones. But the sensor never changes, and so the image space in part (a) still has its range of positive integers. With rays going through the lens, the pincushion-shape FoV (the frame in blue) will be squeezed into a small rectangle, and thus we get the image in the real focal plane with its background grid showing a barrel distorted shape. With lens distortions counted, eqn. (2.2) now needs to be changed into

$$\begin{bmatrix} \mathrm{C'_r} \\ \mathrm{R'_r} \end{bmatrix} = f \begin{bmatrix} \mathrm{X_C}/\mathrm{Z_C} \\ \mathrm{Y_C}/\mathrm{Z_C} \end{bmatrix} \tag{2.19}$$

where $\mathrm{C'_r}$ and $\mathrm{R'_r}$ denote the relative pixel distance on the undistorted "Virtual Focal Plane", whose FoV is pincushion-shape and image coordinates' ranges include negative integers.

Duane [51] gave the lens distortion equation, and the undistorted *Col* and *Row* (C'/R' in our notation) can be expressed as power series in radial distance $r = \sqrt{C^2+R^2}$:

$$\begin{aligned} C' &= C(1+k_1r^2+k_2r^4+k_3r^6)+[p_1(r^2+2C^2)+2p_2CR] \\ R' &= R(1+k_1r^2+k_2r^4+k_3r^6)+[p_2(r^2+2R^2)+2p_1CR] \end{aligned} \tag{2.20}$$

where higher order parameters are omitted for being negligible; (C', R') denote the undistorted pixels in the "Virtual Focal Plane", (C, R) denote the distorted pixel in real sensor image, k_i's are coefficients of radial distortion, and p_j's are coefficients of tangential distortion. The five parameters $k_1/k_2/k_3/p_1/p_2$ are usually called distortion parameters. With the distortion parameters calculated, the distorted (C, R) could be undistorted into (C', R'), and then (C', R') could be used to generate the world space $X^W/Y^W/Z^W$ with intrinsic and extrinsic parameters.

Figure 2.10 shows the flow chart of the whole traditional camera calibration method based on the pinhole camera model. Considering the lens distortions, both of the pinhole camera model (matrix M) and the lens distortions model (five parameters for undistortion) need to be determined. The pinhole camera model can help map from the world space (X^W, Y^W, Z^W) to the undistorted image space (R', C'), which are on the "Virtual Focal Plane" as noted in Fig. 2.9. And the lens distortion model help remove the lens distor-

tions by mapping from (R', C') to (R, C). The pinhole camera model can be determined by eqn. (2.14), and the lens distortion model could be determined by eqn. (2.20).

Chapter 3 Per-Pixel Calibration and 3D Reconstruction on GPU

RGBD cameras are famous for its 3D reconstruction application. In this chapter, we will show how to reconstruct 3D image naturally on GPU in camera space based on raw data without calibration. Considering the deformation of raw data resulted from lens distortions and *depth distortion*, we propose a per-pixel calibration method based on a rail calibration system. Instead of using a pinhole-camera model, a two-dimensional high-order polynomial mapping model is determined and employed for lens distortions removal. Corresponding data collection procedures and look-up table based undistorted real-time world space 3D reconstruction on GPU are discussed in detail. Color values alignment is also discussed at last.

3.1 *RGBD* to $X^C Y^C Z^C RGB$

As described in chapter 1, there are applications like SLAM and KinectFusion that require D to be converted into $X^C Y^C Z^C$ coordinates on a per-pixel basis. From Chapter 2, we know that the depth sensor measures Z^C, and a pinhole camera model (more specifically the intrinsic matrix) in homogeneous coordinates offers the relationship between Z^C and X^C/Y^C respectively. In this section, we will introduce how to generate the camera space 3D coordinates without calibration, and draw a camera space 3D reconstruction on GPU using a KinectV2 camera.

The KinectV2 depth sensor measures Z^C in millimeter and supports its positive data in

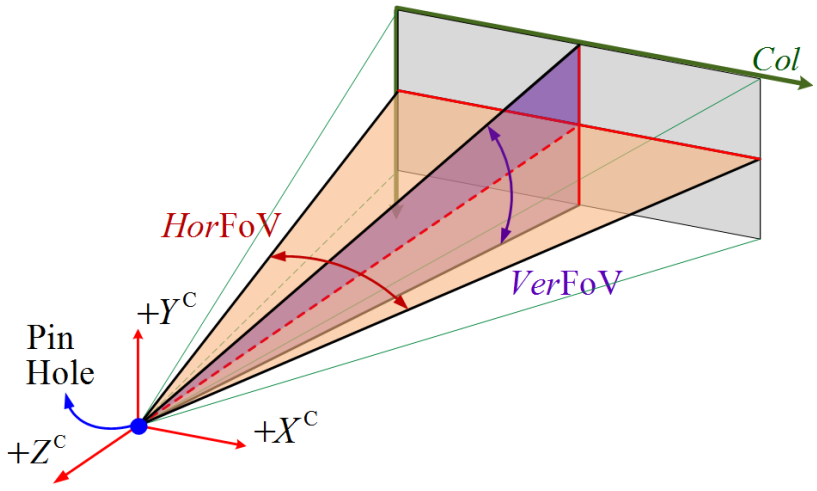


Figure 3.1: Field of View in Pinhole-Camera Model

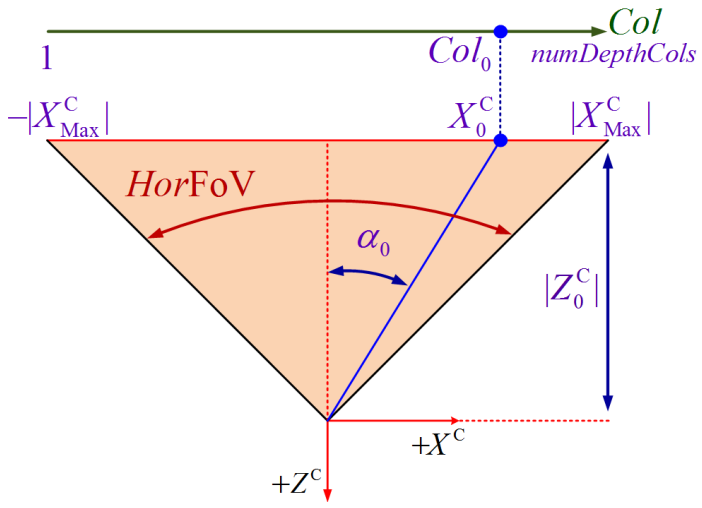


Figure 3.2: map Col to X^C via horizontal FoV

unsigned-short data-type. Those data will be automatically converted into single-floating type with its range from 0.0f to +1.0f when uploaded onto GPU. Considering that in practice Ds from KinectV2 are always positive whereas Z^Cs should be always negative, we will add a negative sign in the un-scaling step to recover the Z^C in metric on GPU:

$$Z^C[m,n] = -\beta D[m,n]\,, \tag{3.1}$$

where β constantly equals to 65535.0 (range of unsigned short in single-floating) for all pixels. Besides the depth stream, KinectV2 supports both of the horizontal and vertical field of view (FoV) that can help generate per-pixel X^C and Y^C, which share the same credits with the intrinsic parameters in a pinhole camera model. Figure 3.1 shows an intuitive view of the horizontal and vertical FoVs in a pinhole camera model, based on which we can derive the X^C and Y^C values given a random Z^C value on the per-pixels basis. Assuming the depth sensor, with its size numDepthRows by numDepthCols, is observing right perpendicularly to a wall where all pixels share the same $|Z^C_0|$ from the sensor to the wall. Talking about the horizontal FoV only, it is easy to get the range of the camera space FoV along X^C-axis from $-|X^C_{\text{Max}}|$ to $|X^C_{\text{Max}}|$, as shown in Fig. 3.2. The horizontal range value $|X^C_{\text{Max}}|$ depends on $|Z^C_0|$ and the horizontal FoV, given by:

$$|X^C_{\text{Max}}| = |Z^C_0| \cdot tan(horFov/2) \tag{3.2}$$

where the pixel observing on $X^C = |X^C_{\text{Max}}|$ has its column address of numDepthCols. Similarly, given a random pixel of column address Col_0, its horizontal view X^C_0 could be ex-

pressed based on its own horizontal view angle α_0:

$$|X_0^C| = |Z_0^C| \cdot tan(\alpha_0). \tag{3.3}$$

To combine eqn. (3.2) and eqn. (3.4), we get

$$\frac{X_0^C}{|X_{\text{Max}}^C|} = \frac{tan(\alpha_0)}{tan(horFov/2)} = \frac{Col_0}{numDepthCols} - 0.5\,, \tag{3.4}$$

which shows how to get the per-pixel X_0^C from $|X_{\text{Max}}^C|$ based on its column address, while $|X_{\text{Max}}^C|$ depends on the depth sensor's horizontal filed of view. It is intuitively better to change eqn. (3.4) a little by substituting $|X_{\text{Max}}^C|$ with eqn. (3.2) such that we can get eqn. (3.5), a proportional per-pixel mapping based on the column addresses from Z^C to X^C.

$$X^C[m,n] = tan(horFov/2) \cdot (\frac{n}{numDepthCols} - 0.5) \cdot |Z^C[m,n]|\,, \tag{3.5}$$

where $[m,n]$ is the discrete space *row* and *column* coordinate of each pixel in the depth sensor. Similarly, we can also get the proportional per-pixel mapping from Z^C to Y^C, based on the vertical FoV and row addresses:

$$Y^C[m,n] = tan(verFov/2) \cdot (\frac{m}{numDepthRows} - 0.5) \cdot |Z^C[m,n]|\,. \tag{3.6}$$

Note that, *horFov* and *verFov* are constant during image processing, such that the mapping functions from per-pixel Z^C to per-pixel X^C/Y^C totally depend on a pixel's address $[m,n]$. Therefore eqn. (3.5) and eqn. (3.6) could be expressed as

$$\begin{aligned} X^C[m,n] &= a[m,n] \cdot |Z^C[m,n]| \\ Y^C[m,n] &= b[m,n] \cdot |Z^C[m,n]| \end{aligned} \tag{3.7}$$

where

$$\begin{aligned} a[m,n] &= tan(horFov/2) \cdot (\frac{n}{numDepthCols} - 0.5) \\ b[m,n] &= tan(verFov/2) \cdot (\frac{m}{numDepthRows} - 0.5)\,. \end{aligned} \tag{3.8}$$

Now that we have the per-pixel mapping from Z^C to X^CY^C, it is time to draw the camera space 3D image on GPU. Figure 3.3 shows the streams flow diagram. We will retrieve *Depth* streams from the KinectV2 camera, save them into corresponding buffers on CPU and upload the the streams onto GPU as textures. Then the per-pixel's camera space 3D coordinates $X^CY^CZ^C$ will be generated during its fragment-shader processing from depth

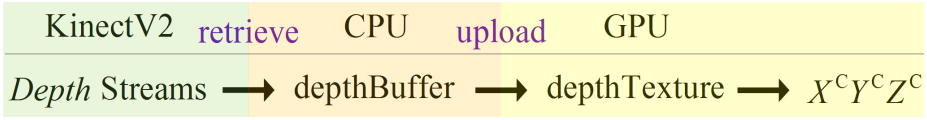


Figure 3.3: Diagram for Camera Space 3D Reconstruction without calibration

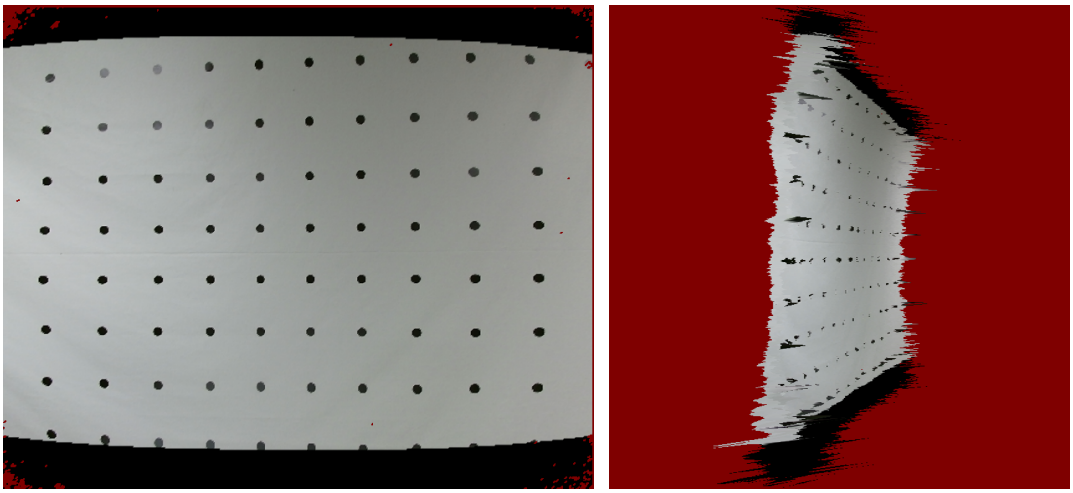
texture based on eqn. (3.7). The fragment shader is programmed as below.

```
uniform sampler2D qt_depthTexture;
uniform sampler2D qt_spherTexture;

layout(location  =0,index  =0)out  vec4 qt_fragColor;
void main()
{
    ivec2 textureCoordinate=ivec2( gl_FragCoord.x, gl_FragCoord.y);

    floatz = texelFetch(qt_depthTexture,textureCoordinate,0).r *65535.0;
    vec4 a = texelFetch(qt_spherTexture,textureCoordinate,0);

    qt_fragColor.x = -z*a.r;
    qt_fragColor.y = -z*a.g;
    qt_fragColor.z = -z;
}
```

The uniform *qt_spherTexture* is a Z^C to X^C/Y^C proportional mapping texture, which contains the per-pixel parameters a/b based on eqn. (3.8). Note that we add three negative signs in front of $X^C Y^C Z^C$ respectively to account for the pinhole-imaging. The whole parallel image processing above shows a natural 3D reconstruction on GPU. Figure 3.4 shows the view of camera space 3D reconstruction when observing uniform grid dots pattern on the flat wall. We can tell from the image that, this reconstruction is totally based on raw data, without calibration at all. Not only the $X^C Y^C$ plane is apparently deformed (lens distortions) in the front view, but the Z^C is also not as flat as it should be, which we will call it *depth distortion* since now. The *depth distortion* comes from the imperfect depth resolutions among pixels, *i.e.*, $Z^C_{[row,col]} - Depth_{[row,col]} \neq E_{\text{Constant}}$, which lead to bumps and hollows in the camera space reconstruction when observing a flat wall. Therefore, a per-pixel calibration is necessary for an undistorted 3D image.

3.2 Rail Calibration System

Talking about camera calibration, the pinhole camera matrix M will come up in most people's mind. As discussed in Chapter 2, the pinhole-camera matrix M consists of an intrinsic



(a) Front View (b) Left View

Figure 3.4: Colored Camera Space 3D Reconstruction

matrix K and an extrinsic matrix $[R_{3*3}, T_{3*1}]$. The camera space 3D reconstruction method we discussed in section 3.1 utilizes horizontal and vertical field of view (FoV)s, which works in the same way with the intrinsic matrix K's principle. However, a camera's calibration needs external help from world space objects, which means neither of the FoVs nor intrinsic matrix K alone is able to do calibration, and extrinsic parameters that can link to world space are necessary in calibration.

Kai [37] did a good job on structured light 3D scanner parallel calibration on GPU, and derived the per-pixel beam equation (3.9) directly from a pinhole camera matrix M, which offers the possibility of natural 3D reconstruction on GPU similar to eqn. (3.7).

$$\begin{aligned} X^W[m,n] &= a[m,n]Z^W[m,n] + b[m,n] \\ Y^W[m,n] &= c[m,n]Z^W[m,n] + d[m,n] \end{aligned}, \tag{3.9}$$

where $[m,n]$ is the discrete space *row* and *column* coordinate of each pixel in a M by N sensor. This per-pixel beam equation shows per-pixel linear mappings from Z^W to X^W/Y^W, and the per-pixel Z^W can be mapped from features of structured light. It is not specially proportional like eqn. (3.7), because it contains space translation infos from camera space to world space. Although it is in world space now and the intrinsic parameters are able to be determined, however, lens distortions are still not able to be handled. To make it ideal, we need to not only remove the lens distortions and *depth distortion*, but also realize the 3D reconstruction on GPU in a natural method similar to eqn. (3.9).

In this section, we will find a best-fit calibration system for a KinectV2 camera's natural

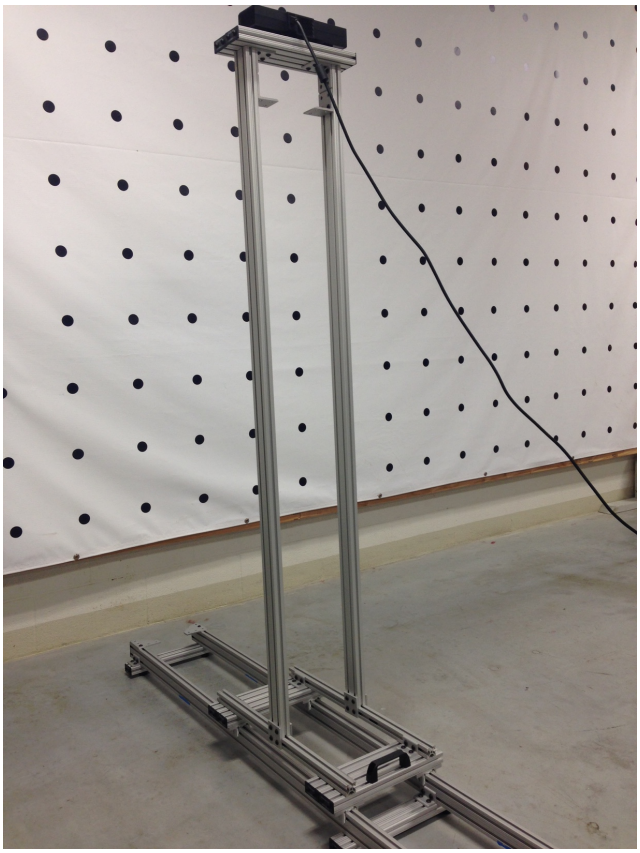


Figure 3.5: KinectV2 Calibration System

calibration and reconstruction, which is able the handle both of lens distortion and *depth distortion*. To easily show 3D reconstruction in a parallel way on the GPU, we would like our calibration system to be able to offer a per-pixel mapping from D to Z^W, which then could be used to map to X^W/Y^W using eqn. (3.9). In this way, the *depth distortion* could also be corrected during the per-pixel D to Z^W mapping. Of all different kinds of calibration systems, a camera on rail system with a planar pattern on wall is finally decided, which offers a moving plane with respect to the camera when the camera moves along the rail. As Fig. 3.5 shows, a canvas on which printed an uniform grid dots pattern is hung on the wall, and the rail is required to be perpendicular to the wall. The RGB-D camera waiting to calibrate is mounted on the slider. Note that, in this calibration system, the only unit that needs to be perpendicular to the wall is the rail, whereas the RGB-D camera has no need to require its observation orientation. Because the per-pixel calibration requires only accurate world space coordinates which will be decided by the rail and the wall, whereas the camera's space is not considered at all. We will assign the pattern plane as the $X^W Y^W$ plane in world space, and the rail to be along with (not exactly on) Z^W-axis. The world coordinate is static with the camera on the slider.

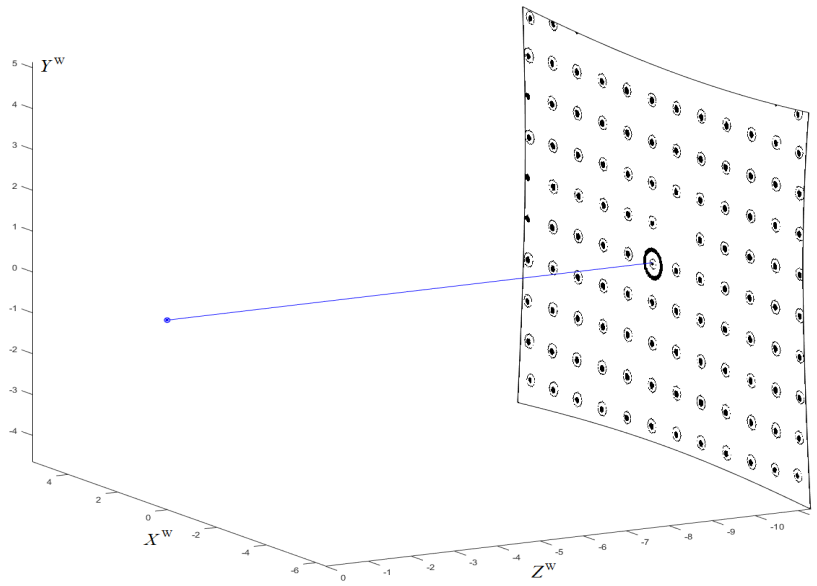


Figure 3.6: NearIR $X^W Y^W Z^W$ 3D Reconstruction

Figure 3.6 shows one frame of NearIR $X^W Y^W Z^W$ 3D reconstruction, which can help a lot explaining how the world space origin is assigned. Inside the figure, both of the origin and Z-axis are high-lighted in blue, and the origin of world space is on the left end of the blue line. On the pattern plane with dot-clusters marked in circles, we can see one dot-cluster is high-lighted inside a thick circle, and its center point is where $X^W/Y^W = 0$. The dot-cluster which will be sitting on the Z^W-axis is the one whose center point is closest to the center pixel of the sensor. All pixels in this frame share the exact same Z^W, which is also why we require the rail to be perpendicular to the pattern. And the value of Z^W is measured by a laser distance measurer that static with the camera. The final origin of the world space will be decided by both of the camera's observing orientation and the laser distance measurer's position. This kind of world coordinate assignment is totally for simplifying image processing during calibration. Practically, we do not even care where exactly the origin is, as long as the rail is perpendicular to the pattern and the distance measurer is static with the camera.

With this rail calibration system, infinite number of frames with infinite number of calibration points could be utilized for training a calibration model. Besides, as the slider moves along the rail, the amount and distribution of the grid dots captured by the camera will change, which means a dynamic pattern for calibration instead of static. With more and more dots walking into the camera's field of view under a certain rhythm as the slider moves further from the pattern plane, the dots (which will be extracted as calibration points) are able to cover all pixels of a sensor. What's more, a moving-plane system (multiple

frames calibration instead of one frame calibration) makes it possible to do dense D to Z^W mapping, which will handle *depth distortion*.

3.3 Data Collection

With the calibration system built up and world coordinate assigned, we are now ready to calibrate. Z^W values for all pixels of every frame will be supported from external laser distance measurer. To simplify potential calculation during image processing, we assign the world coordinate *Unit One* based on the uniform grid dots patter, to be same with the side of pattern's unit-square. Concretely, the distance between every two adjacent dots' centers in real-world is 228mm. Therefore, Z^W = -Z(mm) / 228(mm), where Z is the vertical distance to the pattern plane in reality measured by the laser distance measurer. Note that, Z^W values are always negative, based on the assignment of Cartesian world coordinate. The outline of calibration procedures is listed below.

1. Mount both of the camera and laser distance measurer onto the slider.
2. Move the slider to the nearest position to the pattern plane.
3. Record one frame of RGB data with one frame of NearIR data at this position.
    a) measure $|Z^W|$ using the laser distance measurer.
    b) grab RGB, NearIR and Depth streams from KinectV2 camera.
    c) extract center points (R/C) of dot-clusters from RGB and NearIR streams respectively.
    d) assign X^W/Y^W values to the extracted points, RGB and NearIR respectively.
    e) train and determine the best-fit high order polynomial model that map from *RC* to X^W/Y^W, RGB and NearIR respectively.
    f) generate dense X^W/Y^W for all pixels using the model, for NearIR and RGB streams respectively.
    g) save 2 frames of RGB and NearIR all pixels' data respectively: $X^W Y^W Z^W RGBD$ for RGB stream, and $X^W Y^W Z^W ID$ for NearIR stream, where the channel I in NearIR frame denotes *Intensity*.
4. Move the slider to the next position, and repeat step 3.

Concretely, we will record RGB and NearIR frames every 25mm at a time. During every time, $|Z^W|$ will be measured by the laser measurer and manually input into the shader

at the very beginning of streams recording. After RGB, NearIR and Depth streams have been retrieved from KinectV2, we will utilize digital image processing (DIP) techniques to extract the center points' image addresses (R/C) of dot-clusters, and then determine a high-order polynomial model to build a mapping from RC to X^W/Y^W for dense world coordinates generation.

DIP Techniques on (R, C) Extraction

A robust DIP process on the extraction the points' addresses (R, C)s determines the accuracy of calibration. In this project, the extraction steps consist of gray-scaling, histogram equalization, adaptive thresholding and a little trick on black pixels counting. OpenGL is selected as the GPU image processing language. The default data type of steams saved on GPU during processing is single-floating, with a range from 0 (balck) to 1 (white).

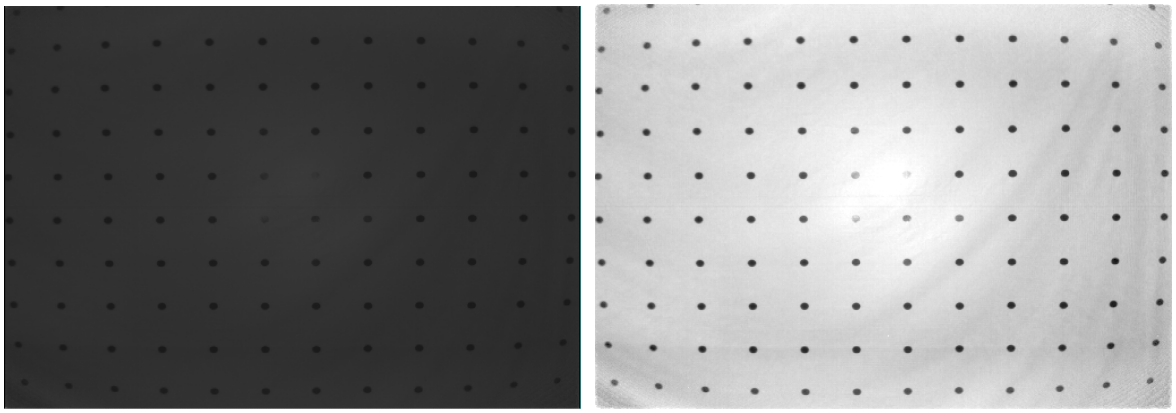
Gray-scaling is done to unify processing steps of both RGB and NearIR steams. For NearIR steam, its data contains only color gray and data will be saved on GPU as single-floating automatically. Whereas for RGB steam, a conversion from RGB to gray value is needed. Typically, there are three converting methods: lightness, average, and luminosity. The luminosity method is finally chosen as a human-friendly way for gray-scaling, give by

$$Intensity_{\_\text{gray}} = 0.21Red + 0.72Green + 0.07Blue\,, \tag{3.10}$$

which uses a weighted average value to account for human perception.

The pixel intensity values are automatically converted into single-floating type with its range from 0.0f to 1.0f, where "0.0f" means 100% black and "1.0f" means 100% white. In practice, NearIR steam image is always very dark, as shown in Fig. 3.7a (with their intensity values every close to zero). In order to enhance the contrast of NearIR image for a better binarizing, histogram equalization technique is utilized to maximize the range of valid pixel intensity distributions. Same process is also compatible on the RGB stream. Commonly, Probability Mass Function (PMF) and Cumulative Distributive Function (CDF) will be calculated to determine the minimum valid intensity value ($floor$) and maximum valid value ($ceiling$) for rescaling, whereas tricks could be used by taking advantage of the GPU drawing properties.

PMF means the frequency of every valid intensity value for all of the pixels in an image. The calculation of PMF on GPU will be very similar to points' drawing process, both of which are on a per-pixel basis. Dividing all of the pixels in terms of their intensity values into N levels, every pixel belongs to one level of them, which is called gray level. With a proper selection of N to make sure a good accuracy, the intensity value of a pixel could be



(a) Raw (b) Histogram Equalized

Figure 3.7: NearIR Streams before / after Histogram Equalization

expressed based on its gray level n

$$Intensity = n/N * (1.0f - 0.0f) + 0.0f = n/N, \tag{3.11}$$

where n and N are integers and $1 \leqslant n \leqslant N$. We will count PMF by drawing all pixels with "1" being their intensity (color) value, zero being y-axis all the time and their original intensity being their position on x-axis. Thus, PMF values at various gray levels could be drawn into a framebuffer object.

With PMF determined, CDF at gray leverl n could be calculated by

$$CDF(n) = \frac{sum}{N_{\text{\_Num of Total Pixels}}}, \tag{3.12}$$

where *sum* denotes the summation of PMF added up consecutively from gray leverl 1 till n, and N denotes how may pixels totally in a image. Then, the intensities $floor$ and $ceiling$ could be determined by

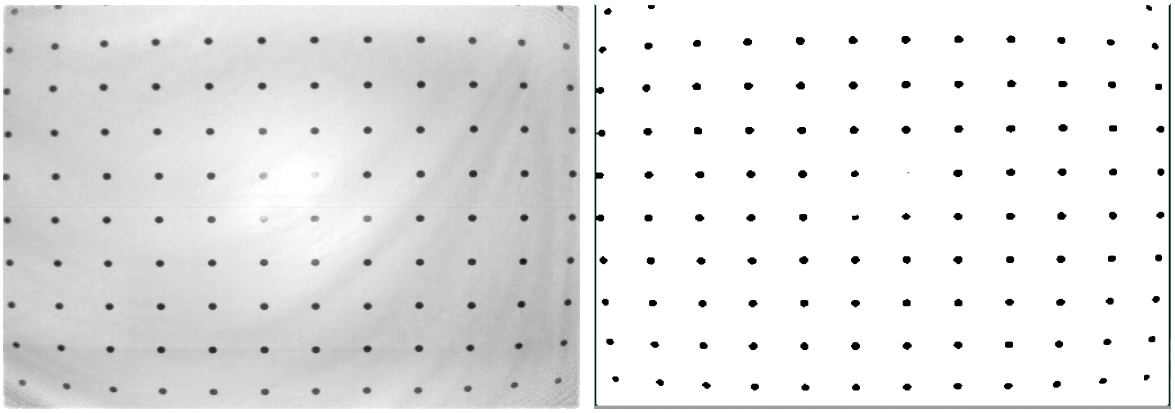
$$\begin{aligned} floor &= n_{\text{\_floor}}/N \\ ceiling &= n_{\text{\_ceiling}}/N \end{aligned} \tag{3.13}$$

through choosing appropriate CDFs, e.g., $CDF(n_{\text{\_floor}}) = 0.01$ and $CDF(n_{\text{\_ceiling}}) = 0.99$. Finally, a new intensity value of every single pixel in an image could be rescaled by

$$Intensity_{\text{\_new}} = \frac{Intensity_{\text{\_original}} - floor}{ceiling - floor}, \tag{3.14}$$

and the image will get a better contrast, as compared in Fig. 3.7.

Affected by radial dominated lens distortions, the intensity value tend to decrease as the



(a) Histogram Equalized (b) Binarized

Figure 3.8: NearIR Streams before / after Adaptive Thresholding

position of a pixel moves from the center of an image to the borders, in the case of observing a singular color view. Therefore, an adaptive thresholding process is needed, whereas using fixed thresholding will generate too much noise around borders. To segment the black dots from white background, we could simply subtract an image's background from an textured image, where the background comes from a blurring process of that image.There are three common types of blurring filters: mean filter, weighted average filter, and gaussian filter. Mean filter is selected for this background-aimed blurring process, because it has the smallest calculation and also a better effect of averaging than the others. After the blurred image containing background information is obtained, the binarizing (subtraction) process for every single pixel could be written as

$$Intensity_{\_\text{binarized}} = \begin{cases} 1, & I_{\_\text{textured}} - I_{\_\text{background}} - C_{\_\text{offset}} > 0 \\ 0, & else \end{cases}, \tag{3.15}$$

where I is short for $Intensity$ of every single pixel, and $C_{\_\text{offset}}$ is a small constant that could be adjusted depending on various thresholding situations. In this project, $C_{\_\text{offset}}$ is around 0.1.To sharpen the edge of the binarized image for a better "circle" shape detection, a median filter could be added as the last step of adaptive thresholding. As shown in Fig. 3.13, background is removed in the binarized image after adaptive thresholding.

After the adaptive thresholding, image data saved on GPU is now composed of circle-shaped "0"s within a background of "1"s. In order to locate the center of those "0"s circle, which is the center of captured round dot, it is necessary to know the edge of those circles. A trick is used to turn all of the edge data into markers that could lead a pathfinder to

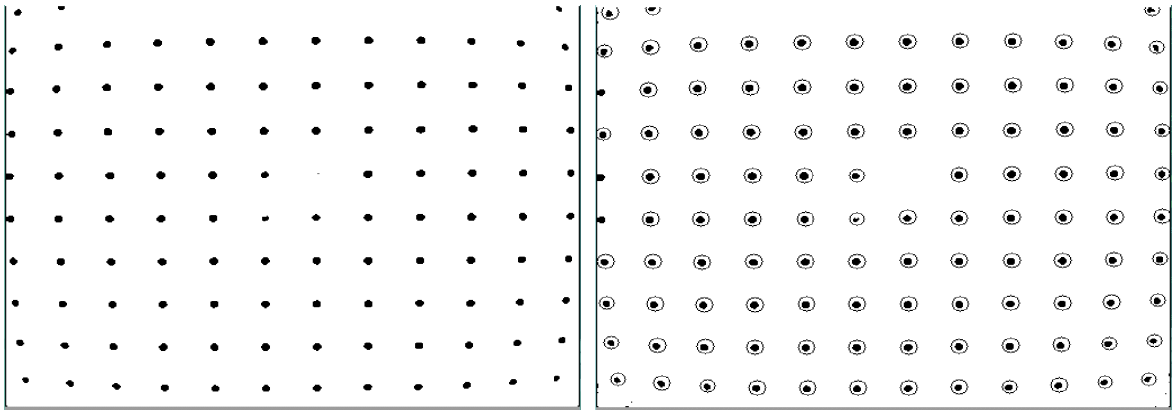
retrieve circle information.The idea that helps to mark edge data is to reassign pixels' values (intensity values) based on their surroundings. Using letter O to represent one single pixel in the center of a 3×3 pixels environment, and letters from A~H to represent surroundings, a mask of 9 cells for pixel value reassignment could be expressed as below.

E	A	F
B	O	C
G	D	H

To turn the surroundings A~H into marks, different weights will be assigned to them. Those markers with different weights have to be non-zero data, and should be counted as the edge-part of circles. Therefore, the first step is to inverse the binary image, generating an image that consists of circle-shaped "1"s distributed in a background of "0"s.After reversing, the next step is to assign weight to the surroundings. OpenGL offers convenient automatic data type conversion, which means the intensity values from "0" to "1" of single-floating data type save on GPU could be retrieved to CPU as unsigned-byte data type from "0" to "255". Considering a bitwise employment of markers, a binary calculation related weight assignment is used in the shader process for pixels. The intensity reassignment for every single pixel is expressed as the equation below.

$$I_{\text{\_Path Marked}} = I_{\text{\_Original}} * \frac{(128I_{\text{A}} + 64I_{\text{B}} + 32I_{\text{C}} + 16I_{\text{D}} + 8I_{\text{E}} + 4I_{\text{F}} + 2I_{\text{G}} + I_{\text{H}})}{255} \tag{3.16}$$

After this reassignment, the image is not binary any more. Every non-zero intensity value contains marked information of its surroundings, data at the edge of circles are now turned into fractions. In other words, the image data saved on GPU at the moment is composed of "0"s as background and "non-zero"s circles, which contains fractions at the edge and "1"s in the center.Now, it is time to discover dots through an inspection over the whole path-marked image, row by row and pixel by pixel. Considering that, a process of one single pixel in this step may affect the processes of the other pixels (which cannot be a parallel processing), it is necessary to do it on CPU. The single-floating image data will be retrieved from GPU to a buffer on CPU as unsigned-byte data, waiting for inspection. And correspondingly the new CPU image will have its "non-zero"s circles composed of fractions at the edge and "1"s in the center. Whenever a non-zero value is traced, a dot-circle is discovered and a singular-dot analysis could start. The first non-zero pixel will be called as an anchor, which means the beginning of a singular-dot analysis. During the singular-dot analysis beginning from the anchor, very connected valid (non-zero) pixel will be a stop, and a "stops-address" queue buffer is used to save addresses of both visited



(a) before Extraction (b) Extracted and Marked

Figure 3.9: Valid Dot-Clusters Extracted in NearIR

pixels and the following pixels waiting to be visited. On very visit of a pixel, there is a checking procedure to find out valid (non-zero) or not. Once valid, the following two steps are waiting to go. The first step is to sniff, looking for possible non-zero pixels around as the following stops. And the second step is to colonize this pixel, concretely, changing the non-zero intensity value to zero. Every non-zero pixel might be checked 1~4 times, but will be used to sniff for only once.

As for the sniffing step, base on the distribution table of *A*~*H* that has been discussed above and their corresponding weight given by equation 3.16, the markers *A*/*B*/*C*/*D* are valid (non-zero) as long as the intensity value of pixel *O* satisfies the following conditions shown as below.

$$
\begin{aligned}
&if\ (I_{\mathrm{O}}\ \&\ \mathrm{0x80} == 1), \quad then, \quad \text{marker } A \text{ is valid} \ \text{(go Up)} \\
&if\ (I_{\mathrm{O}}\ \&\ \mathrm{0x40} == 1), \quad then, \quad \text{marker } B \text{ is valid} \ \text{(go Left)} \\
&if\ (I_{\mathrm{O}}\ \&\ \mathrm{0x20} == 1), \quad then, \quad \text{marker } C \text{ is valid} \ \text{(go Right)} \\
&if\ (I_{\mathrm{O}}\ \&\ \mathrm{0x10} == 1), \quad then, \quad \text{marker } D \text{ is valid} \ \text{(go Down)}
\end{aligned}
$$

Once a valid marker is found, its address $(column, row)$ will be saved into the "stops-address" queue. One pixel's address might be saved for up to 4 times, but "colonizing" procedure will only happen once at the first time, so that the sniffing will stop once all of the connected valid pixels in a singular dot-cluster are colonized as zeros. In the second step "colonizing", I_{O} is changed to zero, variable *area* of this dot-cluster pluses one, and bounding data *RowMax* / *RoxMin* / *ColumnMax* / *ColumnMin* are also updated.Finally, the Round Dot Centers $(column, row)$ could be determined as the center of bounding boxes with their borders *RowMax* / *RoxMin* / *ColumnMax* / *ColumnMin*. After potential noises

being removed based on their corresponding *area* and *shape* (ratio of width and height), the data left are taken as valid dot-clusters. As shown in Fig. 3.9b, the centers of valid dot-clusters are marked within their corresponding homocentric circles.

(X^W, Y^W) Fitting based on Uniform Grid

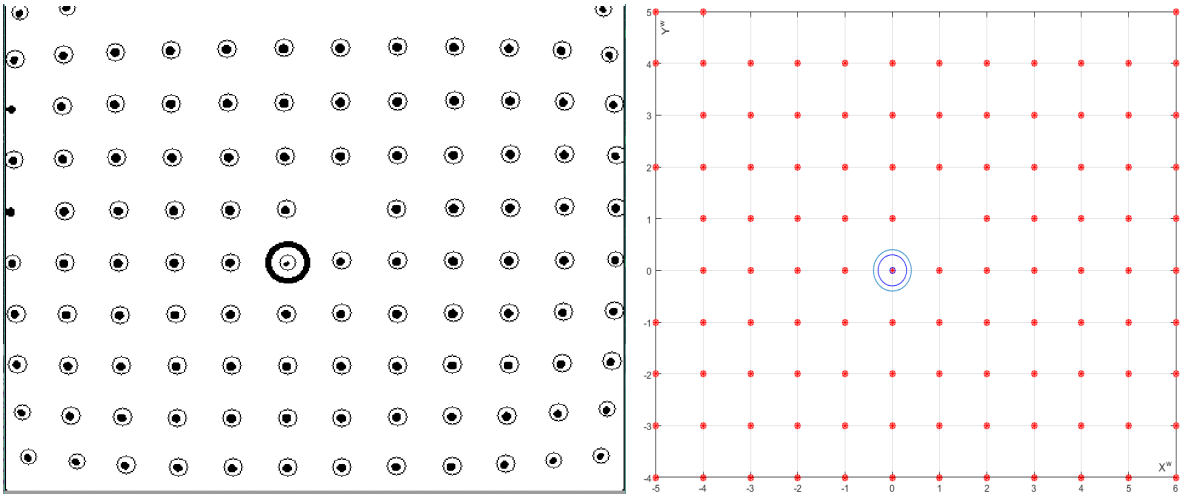
With a list of image space points (R,C)s extracted, the following is to assign those points with their corresponding world coordinates (X^W, Y^W)s, so that we can get coordinate-pairs to train a mapping model for dense $X^W Y^W$ generation. The world coordinates are based on the uniform grid. Taking the side of unit-square (distance between two adjacent dots) as "Unit One" in the world coordinates and one dot as the origin of plan $X^W Y^W$, all fitted X^W/Y^W values will be integers.

Ideally, a 3×3 perspective transformation matrix could help set a linear mapping between two different 2D coordinates, and 3 dot centers with known coordinates pair of (R,C) and (X^W, Y^W) are enough to determine the transformation matrix. Once four points with a squared-shape R/C distribution is found, a 3×3 perspective transformation matrix A could be determined by solving

$$\begin{bmatrix} zX^W \\ zY^W \\ z \end{bmatrix} = A \begin{bmatrix} C \\ R \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} C \\ R \\ 1 \end{bmatrix}, \tag{3.17}$$

where C and R are vectors consist of four squared-shape distributed points' addresses; X^W and Y^W are vectors consist of four points $(0,0)$, $(0,1)$, $(1,1)$, and $(1,0)$; z denotes the third axis in the homogenous system connecting two coordinates.

Deformed by lens distortions, the cluster centers in image plane are not uniformly distributed, and this 3×3 transformation matrix can only generate corresponding decimal X^W/Y^W values that are close integers. But in practice, the correct integer values X^W/Y^W could still be located through *Rounding*. Considering that a list of cluster centers' image coordinate (R,C)s will give many groups of four squared-shape distributed points, and each of them gives a different image coordinate distance will be mapped to the "Unit One" in world coordinates, we will traverse all of the possible groups of four squared-shape distributed points and pick out the group whose mapping matrix A generates the most points that are close to integers. In this way, we find the transformation matrix A can give the best "Unit One" distance in world coordinates. However, its generated point $X^W/Y^W = 0$ is usually not at the dot-cluster that is closest to the center of cameras' FoV. A translation matrix T could be used to refine the transformation matrix A and help to translate the origin



(a) ImagePlane Coordinate (b) World Coordinate

Figure 3.10: Coordinates-Pairs: (R,C)s and (X^W, Y^W)s

point to be at the dot cluster we want:

$$A_{\text{\_refined}} = T \cdot A = \begin{bmatrix} 1 & 0 & -X_{\text{\_Zero\_A}} \\ 0 & 1 & -Y_{\text{\_Zero\_A}} \\ 0 & 0 & 1 \end{bmatrix} \cdot A \quad , \tag{3.18}$$

where $(X_{\text{\_Zero\_A}}, Y_{\text{\_Zero\_A}})$ is the integer world space address rounded after the transformation from image space center point:

$$\begin{bmatrix} zX_{\text{\_Zero\_A}} \\ zY_{\text{\_Zero\_A}} \\ z \end{bmatrix} = A \cdot \begin{bmatrix} C_{\text{\_center}} \\ R_{\text{\_center}} \\ 1 \end{bmatrix} . \tag{3.19}$$

where $(C_{\text{\_center}}, R_{\text{\_center}})$ denotes the center point of FoV in image space. Eventually, the refined transformation matrix $A_{\text{\_refined}}$ can help assign world coordinates to the extracted image space coordinates, with the point $X^W/Y^W = 0$ at the dot-cluster nearest to the image center, as shown in Fig. 3.10b.

Dense X^W/Y^W Generation

The generation of undistorted dense $X^W Y^W$ needs to consider lens distortion removal. In the traditional calibration method, world space $X^{\mathrm{W}}/Y^{\mathrm{W}}/Z^{\mathrm{W}}$ are mapped to undistorted (R',C') by linear pinhole camera matrix M. And then the undistortion step from undistorted (R',C') to distorted (R,C) is done by eqn. (2.20), which uses a high order (higher than

2nd order) polynomial equation. Assuming that there is a high order mapping relationship directly from the distorted image space (R,C) to world space $(X^{\mathrm{W}}, Y^{\mathrm{W}})$, we will do different orders of two-dimensional polynomial prototypes in Matlab using its application of "Curve Fitting Toolbox", and then decide a best-fit mapping model with a high accuracy and a relative small number of parameters.

A two-dimensional polynomial model means surface mapping between two different spaces. Equation. (3.17) (perspective correction) as 1^{st} order polynomial mapping is not able to handle lens distortions. The second order polynomial mapping has $2 \times 6 = 12$ parameters, written as

$$\begin{aligned} X^W &= a_{11}C^2 + a_{12}CR + a_{13}R^2 + a_{14}C + a_{15}R + a_{16} \\ Y^W &= a_{21}C^2 + a_{22}CR + a_{23}R^2 + a_{24}C + a_{25}R + a_{26}\,, \end{aligned} \tag{3.20}$$

and similarly, the fourth order polynomial mapping has $2 \times 15 = 30$ parameters, given by

$$\begin{aligned} X^W &= a_{11}C^4 + a_{12}C^3R + a_{13}C^2R^2 + a_{14}CR^3 + a_{15}R^4 + a_{16}C^3 + a_{17}C^2R \quad ... \\ &\quad + a_{18}CR^2 + a_{19}R^3 + a_{110}C^2 + a_{111}CR + a_{112}R^2 + a_{113}C + a_{114}R + a_{115} \\ Y^W &= a_{21}C^4 + a_{22}C^3R + a_{23}C^2R^2 + a_{24}CR^3 + a_{25}R^4 + a_{26}C^3 + a_{27}C^2R \quad ... \\ &\quad + a_{28}CR^2 + a_{29}R^3 + a_{210}C^2 + a_{211}CR + a_{212}R^2 + a_{213}C + a_{214}R + a_{215}\,. \end{aligned} \tag{3.21}$$

To prototype equation 3.20 and 3.21 in Matlab, "Curve Fitting Toolbox" is used to obtain the 2×6 and 2×15 parameters, using 107 points' coordinates pairs of image space R/C and world coordinates X^W/Y^W. Based on the "Goodness of fit" of transformation parameters from Matlab, the Root-Mean-Square Error (RMSE) of (X^W, Y^W) is (0.06796, 0.05638) for the 2^{nd} order polynomial, and (0.02854, 0.02343) for the 4^{th} order polynomial. As the order of polynomial goes higher, the RMSE gets smaller, and the cost of parameters' calculation gets much higher as well. After the prototyping in Matlab, the different orders of polynomial models will be applied into real-time streams transformation to get live transformed world space reconstruction. Eventually, the 4^{th} order polynomial transformation model is selected as the distortion removal mapping model, which can give an intuitive undistorted world space image while costing relative less calculations. This mapping model will be trained by coordinate-pairs of the extracted points, and then be applied to image space addresses to generate dense X^W/Y^W for all pixels.

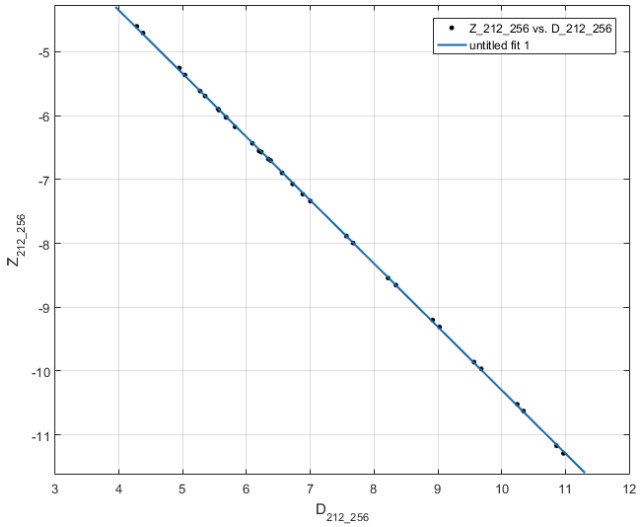


Figure 3.11: Polynomial Fitting between D and Z^W

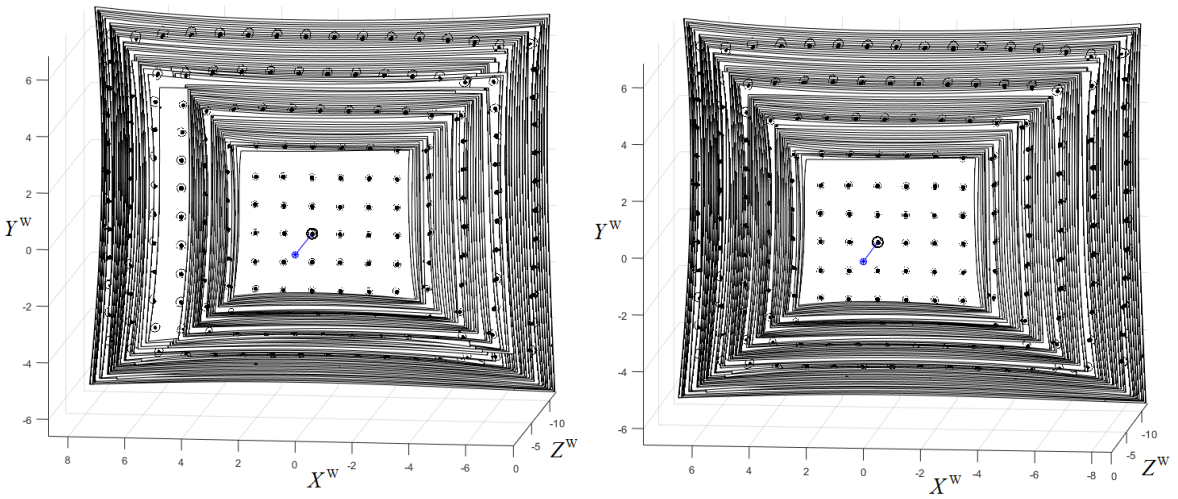
3.4 Data Process and LUT Generation

Till now, we have collected frames of $X^W Y^W Z^W RGBD$ data from RGB streams and $X^W Y^W Z^W ID$ from NearIR stream. The $X^W Y^W Z^W RGBD$ data from RGB streams can be used for color values' alignment after the 3D reconstruction based on depth stream. Since the NearIR stream has same pixel distribution with depth stream, we will process the $X^W Y^W Z^W ID$ data and generate per-pixel mapping parameters, which can be saved as LUT for undistorted 3D reconstruction. From section 3.2, we know that our whole idea of a calibrated natural 3D reconstruction on GPU is to make use of per-pixel eqn. (3.9), which requires undistorted $X^W Y^W Z^W$ data and a per-pixel mapping from D to Z^W. With enough $X^W Y^W Z^W ID$ data already been collected, we will determine a per-pixel D to Z^W mapping, and then generate LUT for calibrated 3D reconstruction.

Both of D and Z^{W} are continuous data, so that their function could written as a polynomial expression, based on Taylor series. We will determine a best-fit mapping model from D to Z^{W} using Matlab. Figure 3.11 shows the polynomial fitting result in Matlab "Curve Fitting Tool" toolbox, with 32 points of DZ^{W} values (at pixel *column*=256 and *row*=212) from 32 frames. It is apparent that Z^{W} is linear with D. Therefore, for every single pixel, Z^{W} could be mapped from D through

$$Z^W[m,n] = e[m,n]D[m,n] + f[m,n], \tag{3.22}$$

where $[m,n]$ denotes the image address *Row* and *Col* of a pixel, e/f are the corresponding linear coefficients that help map from D to Z^{W}. With eqn. (3.22) supporting the per-pixel



(a) Staggered (b) Unified

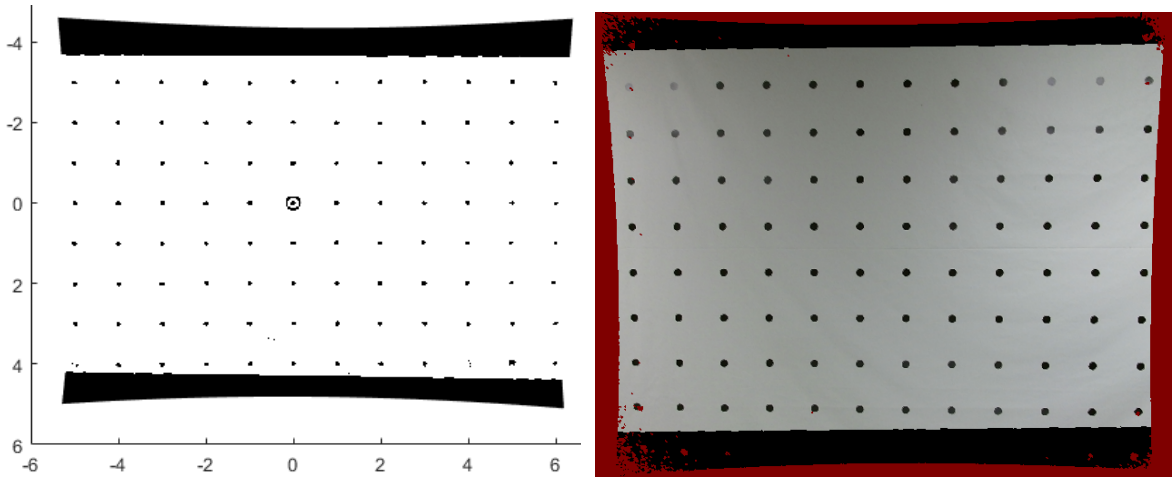
Figure 3.12: World Space Unification of Collected Data

Z^W values and eqn. (3.9) help generating the per-pixel $X^W Y^W$ values, we found all of the six per-pixel parameters $(a/b/c/d/e/f)$ we need. We are now ready to process the collected data off-line and generate a *numDepthRows* by *numDepthCols* by 6 LUT.

The collected data cannot be used directly for LUT generation, because X^W/Y^W may not be well aligned and pre-processes of data are needed. Our calibration system did not require the camera's observing orientation to be along with Z^W-axis, which results to the fact that, the collected frames might be staggered when the centered dot-cluster in camera's FoV moves. Figure 3.12a shows the raw collected data before world space unification, which has a staggered piles of frames. In order to make the data available for generating valid per-pixel 3D reconstruction parameters, we need to unify their world space by adding or subtracting a corresponding integer to all pixels in every staggered frame, making sure the point $X^W Y^W = 0$ is at the same dot-cluster for all frames. After the data unification of world space, as shown in Fig. 3.12b, we are all ready to determine the per-pixel mapping parameters $a/b/c/d/e/f$ and generate the LUT.

3.5 Alignment of RGB Pixel to Depth Pixel

Now an undistorted 3D reconstruction could be displayed with the help of the calibrated LUT. However, we have not figured out yet what the color of each pixel is. To generate a colored 3D reconstruction with a combination of a random depth sensor and a random RGB sensor, we need to align the RGB pixels to depth pixels. The intermediate between the depth sensor image space and RGB sensor image space is the world space. As long as



(a) (b)

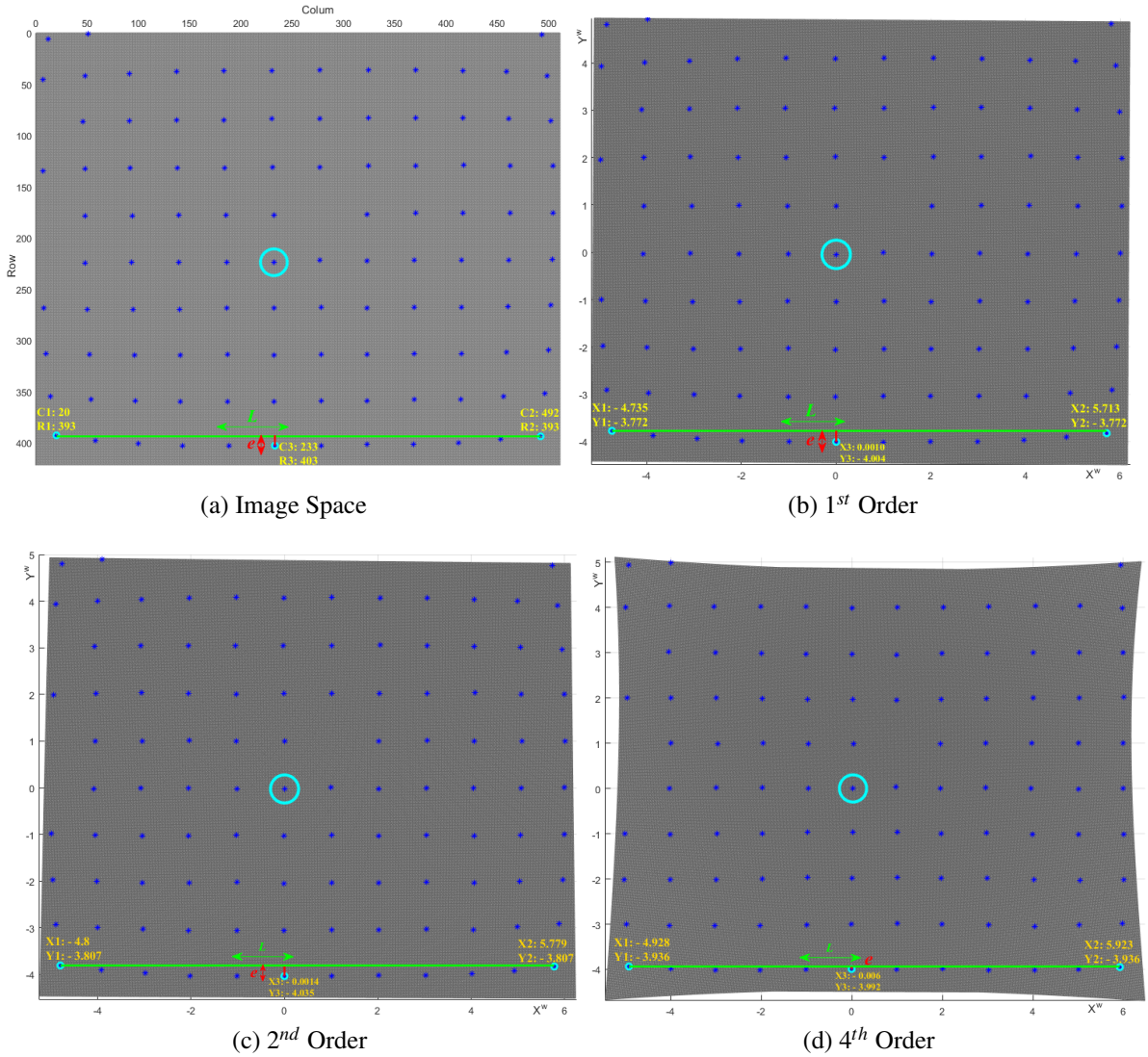
Figure 3.13: Alignment of RGB Texture onto NearIR Image

we figure out the mapping from world space to RGB sensor image space, the color of pixel with known $X^W Y^W Z^W$ could be looked up from the RGB image space. The pinhole camera matrix M is used to map from world space to RGB image space. Using the frame data from Kinect RGB steams and Kinect NearIR streams, a Matlab prototype of RGB pixels alignment is shown in Fig. 3.13a, where the RGB textured is mapped onto its corresponding NearIR image, who has same pixels with depth sensor. The total black area on the top edge and bottom edge is where the depth sensor's view goes beyond the RGB sensor's view. Figure 3.13b shows the screen-shot of live video after calibration with the RGB pixels aligned by a pinhole camera model M.

Chapter 4 Results of Calibration and 3D Reconstruction

4.1 Calibration Results

As discussed in section 3.3, the per-pixel calibration method requires a two-dimensional high-order polynomial model to remove lens distortions and generate estimated (X^W, Y^W)s from image space (R, C)s. In this section, we will show detailed comparisons among different order polynomial results of both Matlab prototypes and real-time live applications. Figure 4.1 shows the Matlab prototypes of the simulated original image, world space $X^W Y^W$



(a) Image Space

(b) 1^{st} Order

(c) 2^{nd} Order

(d) 4^{th} Order

Figure 4.1: $X^W Y^W$ Matlab Polynomial Prototype

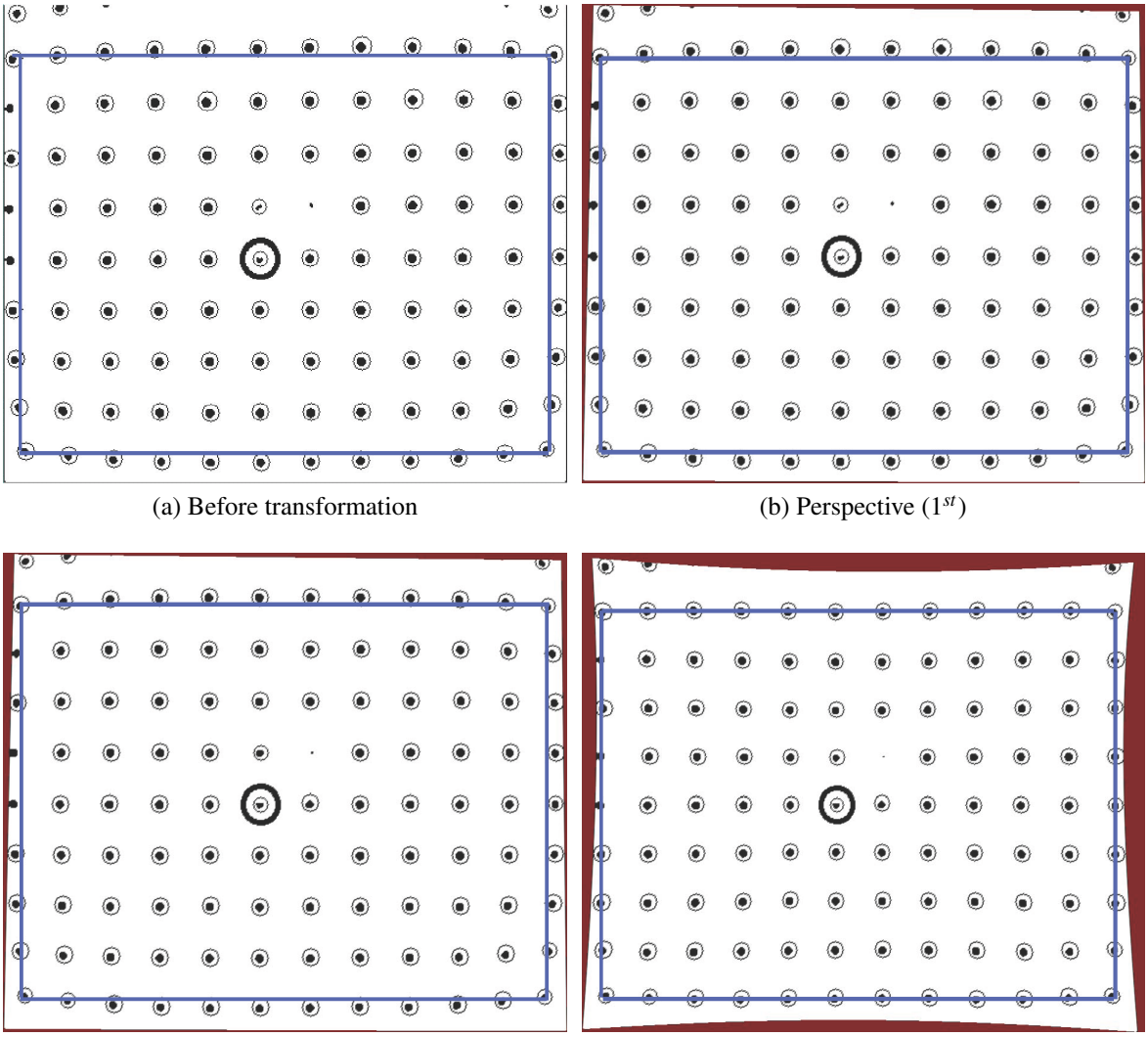
plane result after a two-dimensional 1^{st} order polynomial transformation, 2^{nd} order polynomial and 4^{th} order polynomial transformation.

With squared-shaped distributed points (C, R)s extracted from image streams, we can recover the original distorted image in Matlab, as shown in Fig.4.1a. Using a mathematical distortion (d) measurement [52]

$$d(\%) = e * 100/L, \tag{4.1}$$

we can get the original distortion $d_0 = (R3-R1)/(C2-C1) = (403-393)/(492-20) = 2.1\%$. Figure 4.1b shows estimated world space $X^W Y^W$ plane after a two-dimensional 1^{st} order polynomial transformation, whose distortion $d_1 = (Y1-Y3)/(X2-X1) = [-3.772-(-4.004)]/[5.713-(-4.735)] = 2.2\%$. As we may have expected, the distortion d_1 is not getting smaller at all. Figure 4.1c and Fig. 4.1d show the transformed world space $X^W Y^W$ plane images after the 2^{nd} order and 4^{th} order polynomial transformation respectively, from which we can get $d_2 = [-3.807-(-4.035)]/[5.779-(-4.8)] = 2.1\%$ and $d_4 = [-3.936-(-3.992)]/[5.923-(-4.928)] = 0.516\%$. It is straightforward to tell that, d_4 is much smaller than d_0 and Fig. 4.1d intuitively shows a satisfying undistorted image. From eqn. (3.20) and eqn. (3.21), we know that the second order polynomial mapping has $2 \times 6 = 12$ parameters, and the fourth order polynomial mapping has $2 \times 15 = 30$ parameters. The higher order polynomial we use, the better radial distortion we are able to correct. In the meantime, the distortion removal model will have more parameters to calculate, and need more calibrating points to train the model.

By applying those two-dimensional polynomial models into real-time streams transformation, we can get the transformed stream images. As shown in Fig. 4.2, the outlines of the transformed steam images are same with Matlab prototypes in Fig. 4.1. It is easy to tell that the 4^{th} order polynomial surface mapping is much better than the second order, and a higher order than 4^{th} should be more accurate. However, as the order of the polynomial mapping goes higher, the number of parameters also get larger and larger, which costs more calculations and requires more data (coordinate-pairs) for training the transformation model. Considering that a 5^{th} order polynomial mapping will have much more parameters ($2 \times 21 = 42$) to calculate while may not enhance much accuracy, we choose the 4^{th} order polynomial as the main mapping model to get $X^W Y^W$ values from RC. Limited by the static dot pattern, fewer and fewer dot-clusters could be observed by the camera as the camera getting closer to the dot pattern. Practically, 4^{th} order calibration is replaced by 2^{nd} order to guarantee a robust software when the observed dot-clusters are too few to train the transformation model.



(a) Before transformation

(b) Perspective (1^{st})

(c) 2^{nd} Order

(d) 4^{th} Order

Figure 4.2: NearIR Stream High Order Polynomial Transformation

The two-dimensional high-order polynomial mapping model will be applied in the first calibration step of $X^{\mathrm{W}}Y^{\mathrm{W}}Z^{\mathrm{W}} + D$ frames data collection. Figure 4.3 shows 63 frames of collected $X^{\mathrm{W}}Y^{\mathrm{W}}Z^{\mathrm{W}}$, which gives an pyramid shape of a camera sensor's undistorted world space field of view. For each single pixel, its field of view is a beam, which could be mathematically expressed as equation (3.9). Some sample beams are shown in Fig. 4.4, whose beam equation parameters $c/d/e/f$ are determined as the best-fit totally by the collected undistorted data.

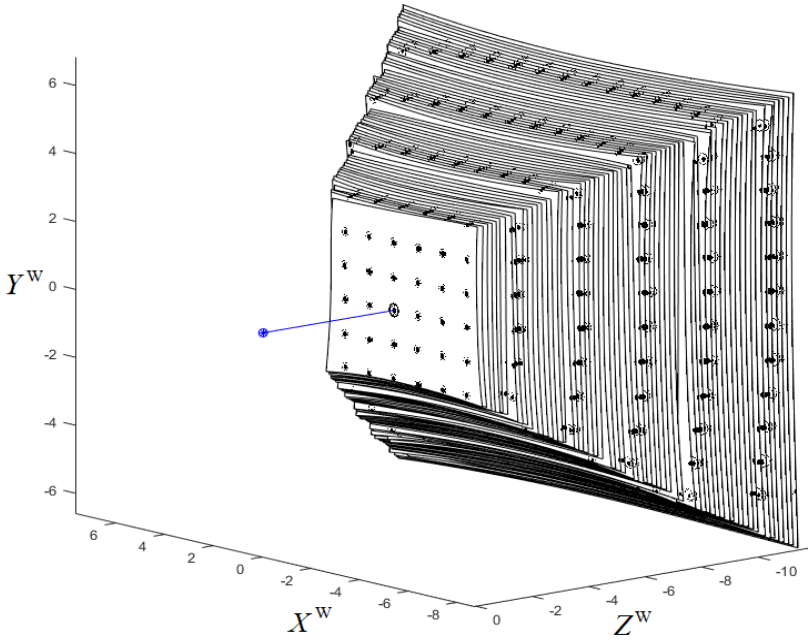


Figure 4.3: 63 Frames NearIR Calibrated 3D Reconstruction

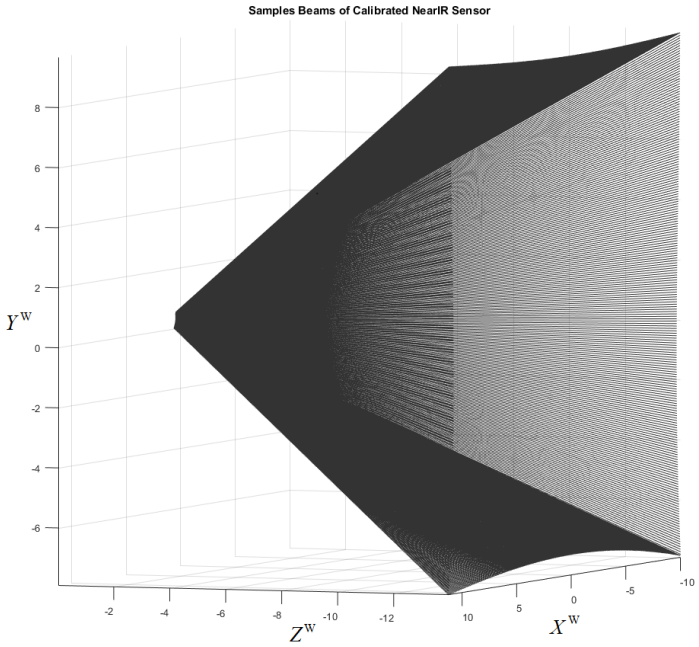
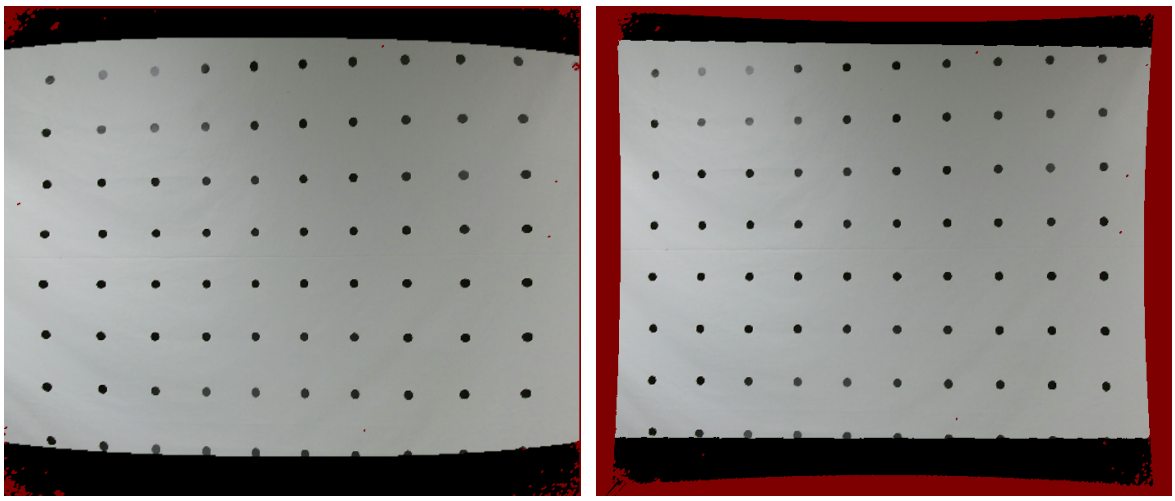


Figure 4.4: Sample Beams of Calibrated NearIR Field of View

4.2 Real-Time 3D Reconstruction on GPU

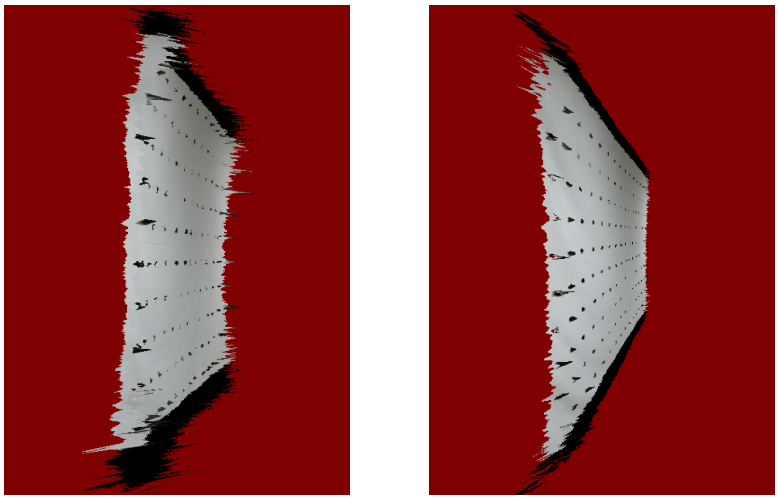
The 3D Reconstruction of undistorted $X^{\mathrm{W}}/Y^{\mathrm{W}}/Z^{\mathrm{W}}$ in real-time is the final aim of a 3D camera's calibration. In the traditional camera calibration method, which consist of one pinhole-camera matrix to generate raw world space 3D coordinates and another model for lens distortion removal, three big transformations are needed to generate the world space coordinates: from 2D distorted image space to 2D undistorted image space, then to 3D camera space, and finally to 3D world space. For every single pixel's processing, it needs 5 parameters from distortion removal model for the first step non-linear calculation, and then a 3×3 intrinsic matrix to get its camera space coordinates, and a 3×4 extrinsic matrix to finally acquire the world space coordinates. The 3D reconstruction after the traditional calibration requires a lot of calculations for every single pixel, and *depth distortion* is not corrected at all.

Using the proposed per-pixel calibration method, only three linear calculations with six parameters are needed to determine the world space coordinates for every single pixel. Two parameters e/f are utilized to generate world space Z^W, as expressed in eqn. (3.22). And the other four parameters $a/b/c/d$ are applied to get X^W/Y^W respectively based on eqn. (3.9). In this way, there is no need to calculate any non-linear equation for distortions removal, and the camera space is totally left aside. Combining two equations together, the undistorted 3D world coordinates $(X^{\mathrm{W}}, Y^{\mathrm{W}}, Z^{\mathrm{W}})$ for every single pixel could be looked up based on D from a *column*-by-*row*-by-6 look-up table. Figure 4.5 shows how lens distortions are moved, and Fig. 4.6 shows how the *depth distortion* is removed by per-pixel



(a) Raw (Distorted) (b) Calibrated

Figure 4.5: Lens-Distortions Removal by Per-Pixel Calibration Method



(a) Raw (b) Calibrated

Figure 4.6: Depth-Distortions Removal by Per-Pixel Calibration Method

D to Z^W mapping.

Chapter 5 Conclusion and Future Work

5.1 Conclusion

The depth sensor technologies opens a new epoch for 3D markets. Ever since Microsoft brought the low-cost depth camera Kinect into consumer market, RGB-D cameras are famous for their 3D reconstruction applications in research areas of HCI. Without calibration, the horizontal and vertical FoV could help generate 3D reconstruction in camera space naturally on GPU through proportional calculations in fragment shader, which however is badly deformed by the lens distortions and *depth distortion*. Traditionally, the camera calibration is done based on a pinhole-camera model and a high-order distortion removal model. But there will be a lot of calculations in the fragment shader to process parameters from both of pinhole-camera matrix and distortion removal vector. In order to get rid of both the lens distortion and the *depth distortion* while still be able to do simple calculations in the GPU fragment shader, a new solution is needed.

In this thesis, a novel per-pixel calibration method with look-up table based 3D reconstruction in real-time is introduced, using a rail calibration system. This rail calibration system offers possibilities of collecting infinite calibration points with dense distributions that can cover all pixels in a sensor. Not only lens distortions, but *depth distortion* can also be handled by a per-pixel D to Z^W mapping with data along Z^W-axis collected on the rail. Instead of utilizing the traditional pinhole camera model, two polynomial mapping models are employed in this calibration method. The first model is the two-dimensional 4^{th} order polynomial mapping from R/C to X^W/Y^W during the frames data collection, which takes care of the removal of lens distortions; and the second model is the linear mapping from D to Z^W, which can handle "depth distortion". The method consists of two big steps: $X^W Y^W Z^W + D$ data collection and mapping parameters determination. D is simply from depth streams. Z^W is from external based on the camera's position on the rail. And the undistorted (X^W, Y^W) are from the transformation of R/C by a 4^{th} order polynomial mapping model, during which lens-distortions could be removed. This method is claimed as "data-based" calibration method, because both of the two mapping models are determined and calculated by real streams data from the camera.

With the fewest calculations, the undistorted 3D world coordinates $(X^{\mathrm{W}}, Y^{\mathrm{W}}, Z^{\mathrm{W}})$ for every single pixel could be looked up in real-time based on D from a *column*-by-*row*-by-6 look-up table. Only three linear equations with six parameters need to be calculated in the fragment shader. Two out of six parameters a/b are to determine the per-pixel Z^W,

which is generated from per-pixel depth value D; and the other four parameters $c/d/d/f$ are to determine the per-pixel X^W/Y^W respectively, which are mapped by per-pixel linear beam equation from the per-pixel Z^W. Note that "real-time" here means being able to show an undistorted frame in 3D world space before the start of the second frame processing, and this LUT based 3D reconstruction method can realize real-time very well. The data-based per-pixel calibration method could be applied universally on any RGB-D cameras. With the alignment of RGB pixels using a pinhole camera matrix, it could even work on a combination of a random Depth sensor and a random RGB sensor.

5.2 Future Work

The rail calibration system and per-pixel calibration method with LUT-based real-time 3D reconstruction could be applied universally to all kinds of RGB-D cameras. With a more precise calibration system and corresponding DIP technologies, there could be a huge improvement space for calibration accuracy. Hardware improvement is sometimes more important than a software updating. Concretely the system in the lab now can only handle a working distance Z^W from 1.165m to 2.565m. Considering that the depth resolution deteriorates notably with depth, it might not be a simple linear relationship from D and Z^W, when the depth D value goes much further than the limit of the rail. In that case, the per-pixel D to Z^W mapping could be changed from singular linear to segmented mapping function when D gets larger than a certain value.

Not only the rail, the planar pattern could also be changed based on the resolution of the camera (*e.g.*, size of the dots and side distance of the unit square-shaped of the uniform grid could be larger when the camera has a supper high resolution). A two dimensional object is totally enough for calibrating KinectV2, because the NearIR stream have same pixels' distortions with the depth stream. However, if NearIR stream could not be used to extract points R/C in image space (*e.g.* a structured light based depth sensor) while the size (pixel numbers) of the RGB sensor does not share the same one with the depth sensor, we could make the planar pattern into a "three dimension mode": change the printed dots into real wholes such that the depth stream could detect a difference between the "dots" and "white background".

Instead of using a laser distance measurer to manually input Z^W into every frame at the very beginning of the frame data collection, we could add a tracking module onto the rail to active the frame collection by software. In this way, not only Z^W could be traced by the tracking module, but also the frame collection could be automatically done by the activation from the tracking module, recording one frame data after every certain distance

of movement has been detected.

Besides the hardware enhancement, softwares like DIP process can also be improved. The order of polynomial model that help map from image space R/C to world space X^W/Y^W could also be heightened for a better accuracy, as long as there will be enough calibrating points to offer the coordinates pairs of both image space and world space. Considering the possible lens-distortions of the RGB sensor, the high order polynomial mapping could also be applied into the color values' alignment from RGB sensor pixels to depth sensor pixels.

Bibliography

[1] Xinshuang Zhao, Ahmed M. Naguib, and Sukhan Lee. Kinect based calling gesture recognition for taking order service of elderly care robot. In *The 23rd IEEE International Symposium on Robot and Human Interactive Communication*, Aug 2014.

[2] Dan Ionescu, Viorel Suse, Cristian Gadea, and Bogdan Solomon. A Single Sensor NIR Depth Camera for Gesture Control. In *2014 IEEE International Instrumentation and Measurement Technology Conference (I2MTC) Proceedings*, May 2014.

[3] Maohai Li, Rui Lin, Han Wang, and Hui Xu. An efficient SLAM system only using RGBD sensors. In *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Dec 2013.

[4] Guanxi Xin, Xutang Zhang, Xi Wang, and Jin Song. A RGBD SLAM algorithm combining ORB with PROSAC for indoor mobile robot. In *International Conference on Computer Science and Network Technology (ICCSNT)*, Dec 2015.

[5] Sebastian A. Scherer and Andreas Zell. Efficient onbard RGBD-SLAM for autonomous MAVs. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov 2013.

[6] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, and Richard Newcombe. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, 2011.

[7] Mattia Mercante. Tutorial: Scanning Without Panels. In *DAVID-Wiki*, Jun 2014.

[8] Daniel Wigdor and Dennis Wixon. Brave NUI World: Designing Natural User Interfaces for Touch and Gesture. *1 edition*, Apr. 2011.

[9] Dean Takahashi. Beyond Kinect, PrimeSense wants to drive 3D sensing into more everyday consumer gear. *Venturebeat*, Jan. 2013.

[10] Krystof Litomisky. Consumer RGB-D Cameras and their Applications. University of California, Riverside. 2012.

[11] Csaba Kertesz. Physiotherapy Exercises Recognition Based on RGB-D Human Skeleton Models. *Modelling Symposium (EMS)*, Nov 2013.

[12] Manuel Gesto Diaz, Federico Tombari, Pablo Rodriguez-Gonzalvez, and Diego Gonzalez-Aguilera. Analysis and Evaluation Between the First and the Second Generation of RGB-D Sensors. *Sensors*, 15(11), Jul. 2015.

[13] Larry Li. Time-of-Flight Camera – An Introduction. *Texas Instruments Technical White Paper*, SLOA190B, 2012.

[14] Three Depth-Camera Technologies Compared. F. Pece and J. Kautz and T. Weyrich". *First BEAMING Workshop*, June 2011.

[15] Erica Ogg. Microsoft to acquire gesture control maker Canesta. *CNET*, Oct. 2010.

[16] Colleen C. Introducing the Intel RealSense R200 Camera (world facing). *Intel Developer Zone*, May 2015.

[17] Liberios Vokorokos, Juraj Mihalov, and Lubor Lescisin. Possibilities of depth cameras and ultra wide band sensor. *International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, Jan 2016.

[18] Meenakshi Panwar. Hand gesture recognition based on shape parameters. In *2012 International Conference on Computing, Communication and Applications*, Feb 2012.

[19] Kam Lai, Janusz Konrad, and Prakash Ishwar. A gesture-driven computer interface using Kinect. In *Image Analysis and Interpretation (SSIAI)*, Apr 2012.

[20] Jaehong Lee, Heon Gul, Hyungchan Kim, Jungmin Kim, Hyoungrae Kim, and Hakil Kim. Interactive manipulation of 3D objects using Kinect for visualization tools in education. In *Control, Automation and Systems (ICCAS)*, Oct 2013.

[21] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments. In *Experimental Robotics*, 2014.

[22] Jodie Wetherall, Matthew Taylor, and Darren Hurley-Smith. Investigation into the Effects of Transmission-channel Fidelity Loss in RGBD Sensor Data for SLAM. In *International Conference on Systems, Signals and Image Processing (IWSSIP)*, Sep 2015.

[23] Henrik Kretzschmar, Cyrill Stachniss, and Giorgio Grisetti. Efficient information-theoretic graph pruning for graph-based SLAM with laser range finders. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep 2011.

[24] Maurice F. Fallon, John Folkesson, Hunter McClelland, and John J. Leonard. Relocating Underwater Features Autonomously Using Sonar-Based SLAM. *IEEE Journal of Oceanic Engineering*, 38(3), Jul 2013.

[25] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, and Andrew J. Davison. KinectFusion: Real-time dense surface mapping and tracking. In *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, Oct 2011.

[26] Cheng-Kai Yang, Chen-Chien Hsu, and Yin-Tien Wang. Computationally efficient algorithm for simultaneous localization and mapping (SLAM). In *IEEE International Conference on Networking, Sensing and Control (ICNSC)*, Apr 2013.

[27] Felix Endres, Jurgen Hess, Nikolas Engelhard, Jurgen Sturm, Daniel Cremers, and Wolfram Burgard. An evaluation of the RGB-D SLAM system. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2012.

[28] Kathia Melbouci, Sylvie Naudet Collette, Vincent Gay-Bellile, Omar Ait-Aider, Mathieu Carrier, and Michel Dhome. Bundle adjustment revisited for SLAM with RGBD sensors. In *IAPR International Conference on Machine Vision Applications (MVA)*, May 2015.

[29] C. W. Hull. âĂIJApparatus for production of three-dimensional objects by stereolithography,âĂİ. *US Patent US4575330 A*, Mar 1986.

[30] Ammar Hattab and Gabriel Taubin. 3D Modeling by Scanning Physical Modifications. In *2015 28th SIBGRAPI Conference on Graphics, Patterns and Images*, Aug 2015.

[31] Nadia Figueroa, Haiwei Dong, and Abdulmotaleb El Saddik. From Sense to Print: Towards Automatic 3D Printing from 3D Sensing Devices. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*, Oct 2013.

[32] D. C. Brown. Close-range camera calibration. *Photogrammetric Engineering*, 37(8), 1971.

[33] W. Faig. Calibration of close-range photogrammetry systems: Mathematical formulation. *Photogrammetric Engineering and Remote Sensing*, 41(12), 1975.

[34] R. Tsai. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. *IEEE Journal on Robotics and Automation*, 3(4), Aug 1987.

[35] Olivier Faugeras. Three-Dimensional Computer Vision. *MIT Press*, Nov 1993.

[36] Zhengyou Zhang. Camera Calibration. (Chapter 2). *Emergin Topics in Computer Vision*, 2004.

[37] Kai Liu. *Real-time 3-D Reconstruction by Means of Structured Light Illumination*. PhD thesis, University of Kentucky, 2010.

[38] Stephen J. Maybank and Olivier D. Faugeras. A theory of self-calibration of a moving camera. *International Journal of Computer Vision*, 8(2), Aug 1992.

[39] Q.-T. Luong and O.D. Faugeras. Self-Calibration of a Moving Camera from Point Correspondences and Fundamental Matrices. *International Journal of Computer Vision*, 22(3), Mar 1997.

[40] R. I. Hartley. An algorithm for self calibration from several views. Jun 1994.

[41] Xiangjian He. Estimation of Internal and External Parameters for Camera Calibration Using 1D Pattern. Nov 2006.

[42] Zijian Zhao. Practical multi-camera calibration algorithm with 1D objects for virtual environments. Apr 2008.

[43] P. F. Sturm and S. J. Maybank. On plane-based camera calibration: A general algorithm, singularities, applications. Jun 1999.

[44] Zhengdong Zhang. Camera calibration with lens distortion from low-rank textures. Jun 2011.

[45] Hamid Bazargani. Camera calibration and pose estimation from planes. *IEEE Instrumentation & Measurement Magazine*, 18(6), Dec 2015.

[46] Paul Ernest Debevec. *Modeling and Rendering Architecture from Photographs*. PhD thesis, University of California at Berkeley, 1996.

[47] Zhengyou Zhang. Camera calibration with one-dimensional objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(7), Jul 2004.

[48] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11), Nov 2000.

[49] J. Weng, P. Cohen, and M. Herniou. Camera calibration with distortion models and accuracy evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(10), Oct 1992.

[50] Nasim Mansurov. What is Distortion. In *PhotographyLife*, Aug 2013.

[51] Duane C Brown. Decentering Distortion of Lenses. *Photogrammetric Engineering*, 32(3), May 1966.

[52] Z. Tang, R. Grompone von Gioi, P. Monasse, and J.M. Morel. High-precision Camera Distortion Measurements with a "Calibration Harp". *Computer Vision and Pattern Recognition*, 29(10), Dec 2012.