

ABSTRACT OF THESIS

RGBD 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 (RGBD) sensors. Without calibration, a RGBD 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^W, Y^W, Z^W) for every single pixel could be generated in real-time. The per-pixel calibration method could be applied universally on any RGBD 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 25, 2016

RGBD 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 25, 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Unpublished theses submitted for the Master's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with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

RGBD 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

Copyright[©] Sen Li 2016

TABLE OF CONTENTS

Table of Contents	iii
Chapter 1 Per-Pixel Calibration and 3D Reconstruction on GPU	1
1.1 $RGBD$ to $X^CY^CZ^CRGB$	1
1.2 Rail Calibration System	6
1.3 Data Collection	9
1.4 Data Process and LUT Generation	20
1.5 Alignment of RGB Pixel to Depth Pixel	23
List of Symbols	25
Bibliography	26

Chapter 1 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.

1.1 RGBD to $X^C Y^C Z^C$ RGB

As described in chapter ??, 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 ??, 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 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 D s from KinectV2 are always positive whereas Z^C 's should be always negative, we will

add a negative sign in the un-scaling step to recover the Z^C in metric on GPU:

$$Z^C_{m,n} = -\beta D_{m,n}, \quad (1.1)$$

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 1.1 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. 1.2. The horizontal range value $|X^C_{\text{Max}}|$ depends on $|Z^C_0|$ and the horizontal FoV, given by:

$$|X^C_{\text{Max}}| = |Z^C_0| \cdot \tan(\text{horFov}/2) \quad (1.2)$$

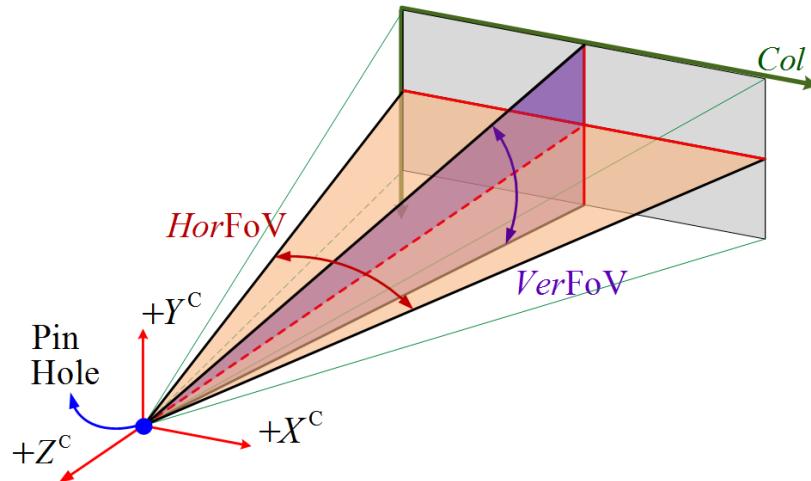


Figure 1.1: Field of View in Pinhole-Camera Model

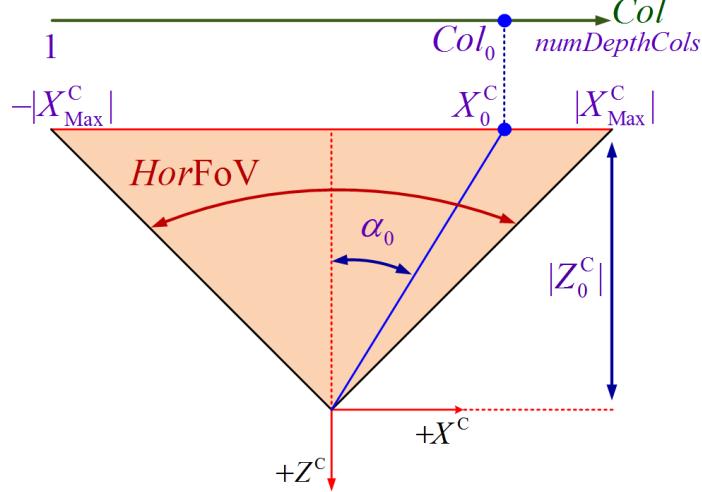


Figure 1.2: map C to X^C via horizontal FoV

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 expressed based on its own horizontal view angle α_0 :

$$|X^C_0| = |Z^C_0| \cdot \tan(\alpha_0). \quad (1.3)$$

To combine eqn. (1.2) and eqn. (1.4), we get

$$\frac{X^C_0}{|X^C_{\text{Max}}|} = \frac{\tan(\alpha_0)}{\tan(horFov/2)} = \frac{Col_0}{\text{numDepthCols}/2} - 1, \quad (1.4)$$

which shows how to get the per-pixel X^C_0 from $|X^C_{\text{Max}}|$ based on its column address, while $|X^C_{\text{Max}}|$ depends on the depth sensor's horizontal field of view. It is intuitively better to change eqn. (1.4) a little by substituting $|X^C_{\text{Max}}|$ with eqn. (1.2) such that we can get eqn. (1.5), a proportional per-pixel mapping based on the column addresses from Z^C to X^C .

$$X^C_{m,n} = \tan(horFov/2) \cdot \left(\frac{n}{\text{numDepthCols}/2} - 1 \right) \cdot |Z^C_{m,n}|, \quad (1.5)$$

where $[m, n]$ is the discrete space R and C 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

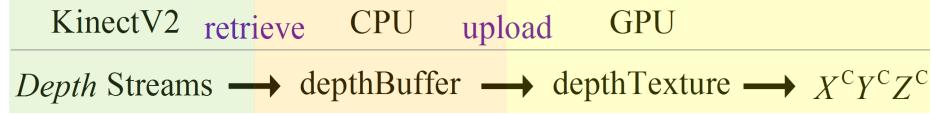


Figure 1.3: Diagram for Camera Space 3D Reconstruction without calibration

vertical FoV and row addresses:

$$Y^C m, n = \tan(\text{verFov}/2) \cdot \left(\frac{m}{\text{numDepthRows}/2} - 1 \right) \cdot |Z^C m, n|. \quad (1.6)$$

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. (1.5) and eqn. (1.6) could be expressed as

$$\begin{aligned} X^C m, n &= a[m, n] \cdot |Z^C m, n| \\ Y^C m, n &= b[m, n] \cdot |Z^C m, n| \end{aligned} \quad (1.7)$$

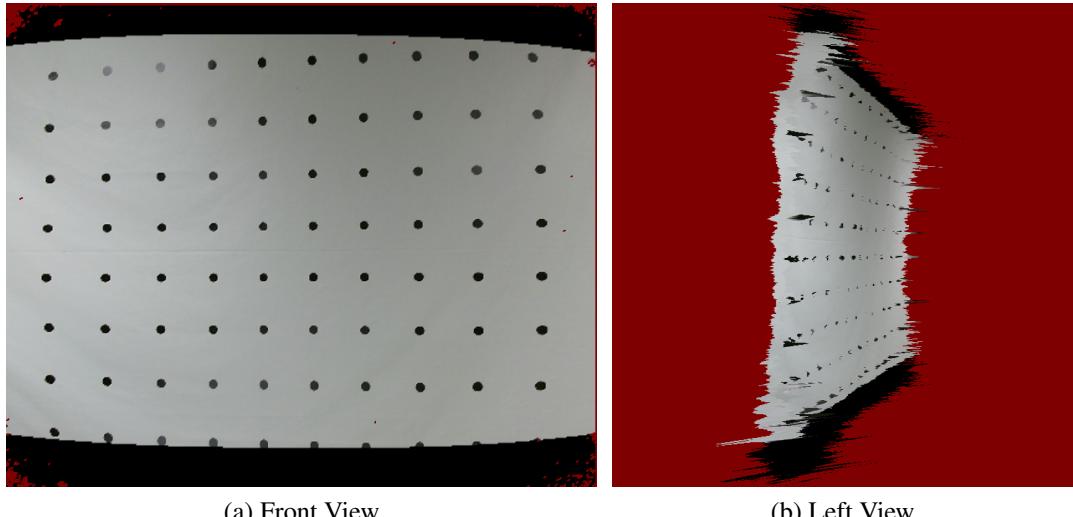
where

$$\begin{aligned} a[m, n] &= \tan(\text{horFov}/2) \cdot \left(\frac{n}{\text{numDepthCols}/2} - 1 \right) \\ b[m, n] &= \tan(\text{verFov}/2) \cdot \left(\frac{m}{\text{numDepthRows}/2} - 1 \right). \end{aligned} \quad (1.8)$$

Now that we have the per-pixel mapping from Z^C to $X^C Y^C$, it is time to draw the camera space 3D image on GPU. Figure 1.3 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 $X^C Y^C Z^C$ will be generated during its fragment-shader processing from depth texture based on eqn. (1.7). The fragment shader is programmed as below.

```
uniform sampler2D qt_depthTexture;
uniform sampler2D qt_spherTexture;

layout(location = 0, index = 0)out vec4 qt_fragColor;
```



(a) Front View

(b) Left View

Figure 1.4: Colored Camera Space 3D Reconstruction

```

void main()
{
    ivec2 textureCoordinate=ivec2( gl_FragCoord.x, gl_FragCoord.y);

    float z = 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. (1.8). Note that we add three negative signs in front of $X^CY^CZ^C$ respectively to account for the pinhole-imaging. The whole parallel image processing above shows a natural 3D reconstruction on GPU. Figure 1.4 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^CY^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.*, $Z_{[row,col]}^C - Depth_{[row,col]} \neq E_{\text{Constant}}$, 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.

1.2 Rail Calibration System

Talking about camera calibration, the pinhole camera matrix M will come up in most people's mind. As discussed in Chapter ??, the pinhole-camera matrix M consists of an intrinsic matrix K and an extrinsic matrix $[R_{3*3}, T_{3*1}]$. The camera space 3D reconstruction method we discussed in section 1.1 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 [1] did a good job on structured light 3D scanner parallel calibration on GPU, and derived the per-pixel beam equation (1.9) directly from a pinhole camera matrix M , which offers the possibility of natural 3D reconstruction on GPU similar to eqn. (1.7).

$$\begin{aligned} X^W m, n &= a[m, n] Z^W m, n + b[m, n] \\ Y^W m, n &= c[m, n] Z^W m, n + d[m, n] \end{aligned} , \quad (1.9)$$

where $[m, n]$ is the discrete space R and C 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. (1.7), 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

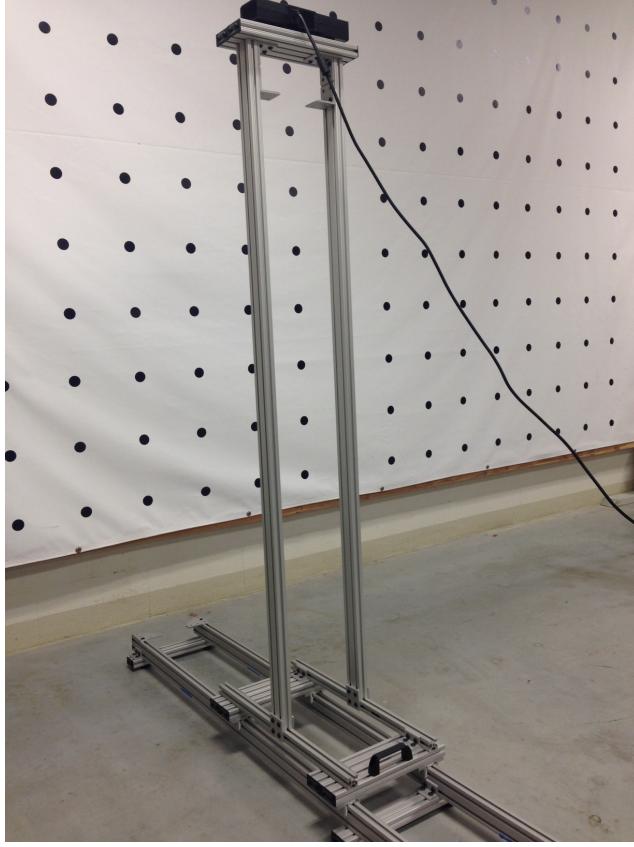


Figure 1.5: KinectV2 Calibration System

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. (1.9).

In this section, we will find a best-fit calibration system for a KinectV2 camera's natural calibration and reconstruction, which is able to 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. (1.9). 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. 1.5 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 RGBD camera waiting

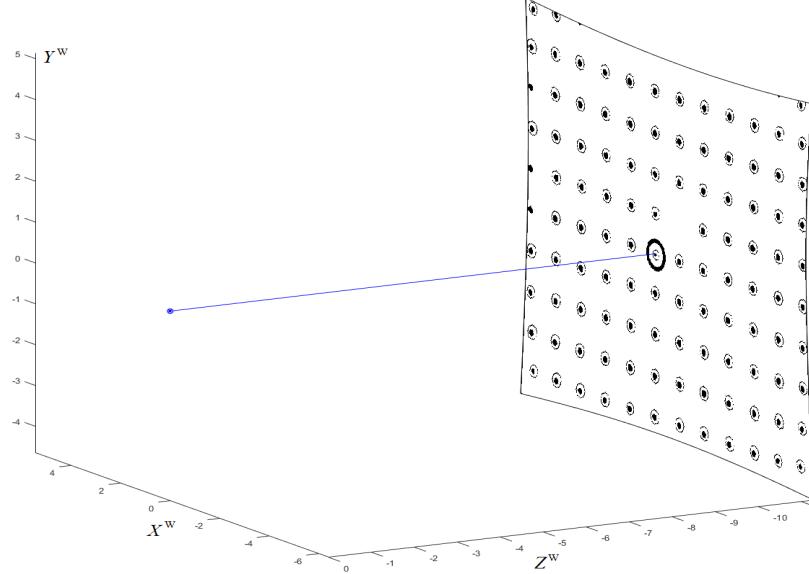


Figure 1.6: NearIR $X^W Y^W Z^W$ 3D Reconstruction

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 RGBD 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 1.6 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*.

1.3 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 pattern, 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, $Z^W = -Z(\text{mm}) / 228(\text{mm})$, 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: $X^W Y^W Z^W RGBT$ 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 streams saved on GPU during processing is single-floating, with a range from 0 (black) to 1 (white).

Gray-scaling is done to unify processing steps of both RGB and NearIR streams. For NearIR stream, its data contains only color gray and data will be saved on GPU as single-floating automatically. Whereas for RGB stream, 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

$$\text{Intensity_gray} = 0.21\text{Red} + 0.72\text{Green} + 0.07\text{Blue}, \quad (1.10)$$

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 stream image is always very dark, as shown in Fig. 1.7a (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

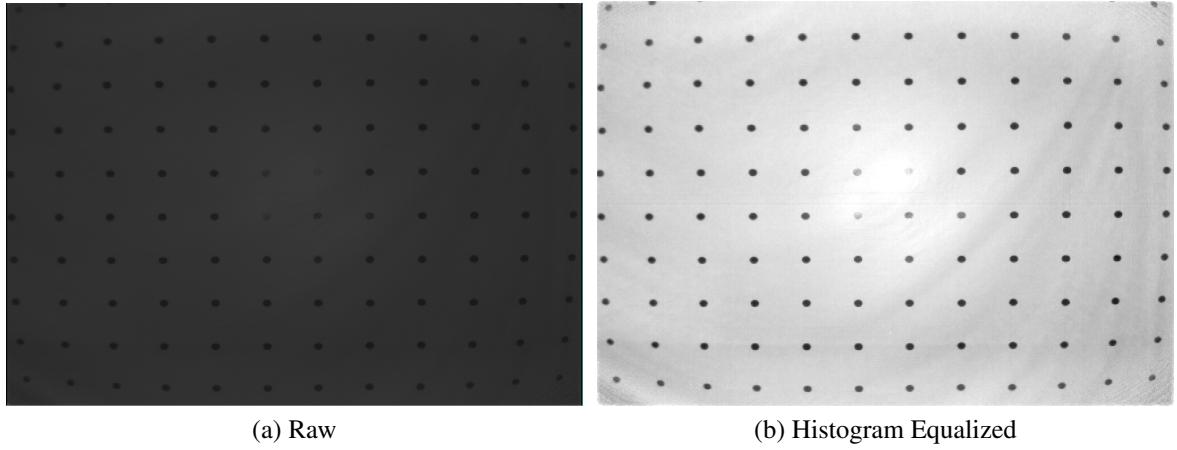


Figure 1.7: NearIR Streams before / after Histogram Equalization

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 expressed based on its gray level n

$$\text{Intensity} = n/N * (1.0f - 0.0f) + 0.0f = n/N, \quad (1.11)$$

where n and N are integers and $1 \leq n \leq N$. 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

$$CDF(n) = \frac{\text{sum}}{N_{\text{Num of Total Pixels}}}, \quad (1.12)$$

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

$$\begin{aligned} \text{floor} &= n_{\text{floor}}/N \\ \text{ceiling} &= n_{\text{ceiling}}/N \end{aligned} \quad (1.13)$$

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

$$\text{Intensity}_{\text{new}} = \frac{\text{Intensity}_{\text{original}} - \text{floor}}{\text{ceiling} - \text{floor}}, \quad (1.14)$$

and the image will get a better contrast, as compared in Fig. 1.7.

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

$$\text{Intensity}_{\text{binarized}} = \begin{cases} 1, & I_{\text{textured}} - I_{\text{background}} - C_{\text{offset}} > 0 \\ 0, & \text{else} \end{cases}, \quad (1.15)$$

where I is short for Intensity of every single pixel, and C_{offset} is a small constant that could be adjusted depending on various thresholding situations. In this project, C_{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. 1.8,

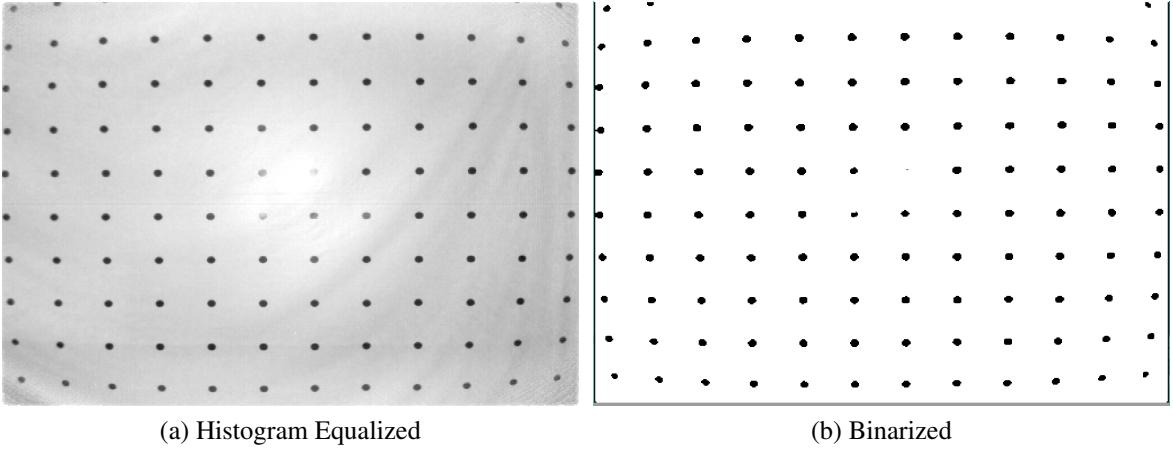


Figure 1.8: NearIR Streams before / after Adaptive Thresholding

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 $A \sim H$ 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 $A \sim H$ 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.

$$I_{\text{Path Marked}} = I_{\text{Original}} * \frac{(128I_A + 64I_B + 32I_C + 16I_D + 8I_E + 4I_F + 2I_G + I_H)}{255} \quad (1.16)$$

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 pixels and the following pixels waiting to be visited. On every 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 $A\sim H$ that has been discussed above and their corresponding weight given by equation 1.16, 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.

```

if ( $I_O \& 0x80 == 1$ ), then, marker A is valid ( go Up )
if ( $I_O \& 0x40 == 1$ ), then, marker B is valid ( go Left)
if ( $I_O \& 0x20 == 1$ ), then, marker C is valid ( go Right )
if ( $I_O \& 0x10 == 1$ ), then, marker D is valid ( go Down )

```

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. 1.9b, 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^WY^W generation. The world coordinates are based on the uniform grid. Taking the side of unit-square (distance between two adjacent dots)

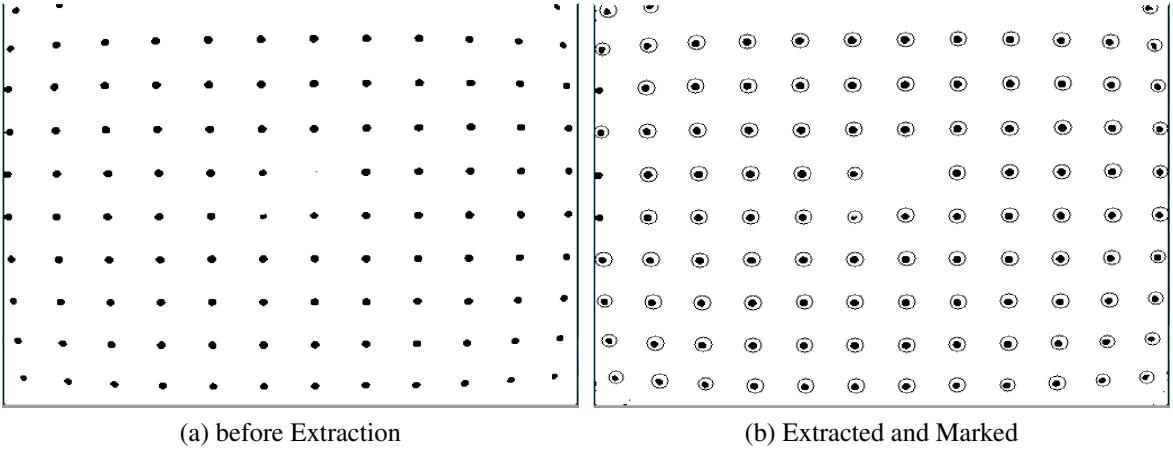


Figure 1.9: Valid Dot-Clusters Extracted in NearIR

as “Unit One” in the world coordinates and one dot as the origin of plan X^WY^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

$$\begin{bmatrix} zX^W \\ zY^W \\ z \end{bmatrix} = A \begin{bmatrix} C \\ R \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} C \\ R \\ 1 \end{bmatrix}, \quad (1.17)$$

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 point to be at the dot cluster we want:

$$A_{\text{refined}} = T \cdot A = \begin{bmatrix} 1 & 0 & -X_{\text{Zero_A}} \\ 0 & 1 & -Y_{\text{Zero_A}} \\ 0 & 0 & 1 \end{bmatrix} \cdot A , \quad (1.18)$$

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

$$\begin{bmatrix} zX_{\text{Zero_A}} \\ zY_{\text{Zero_A}} \\ z \end{bmatrix} = A \cdot \begin{bmatrix} C_{\text{center}} \\ R_{\text{center}} \\ 1 \end{bmatrix} . \quad (1.19)$$

where $(C_{\text{center}}, R_{\text{center}})$ denotes the center point of FoV in image space. Eventually, the refined transformation matrix A_{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. 1.10b.

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 $X^W/Y^W/Z^W$ are mapped to undistorted

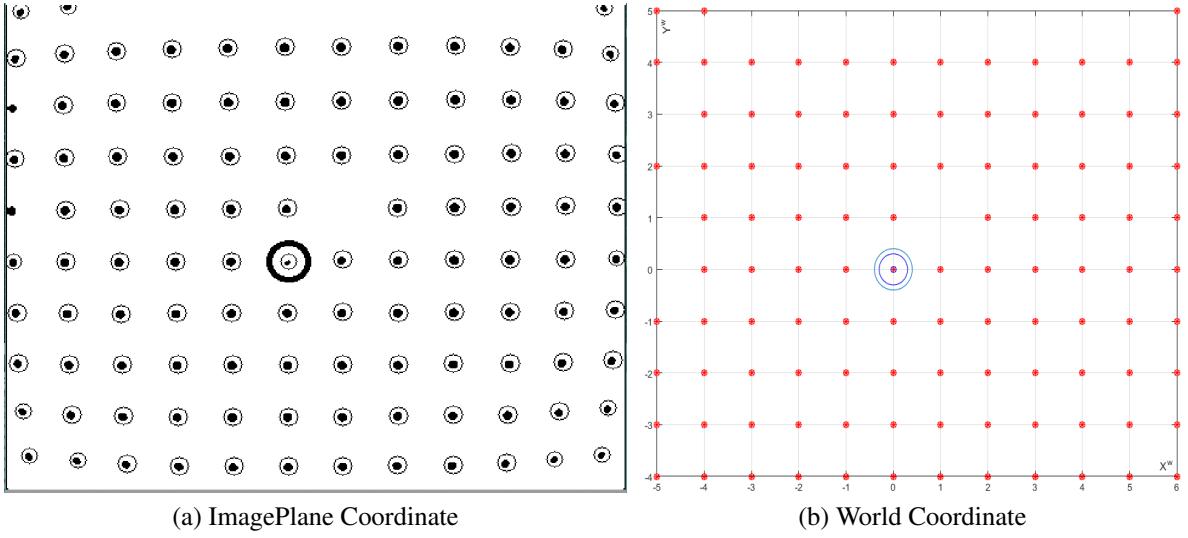


Figure 1.10: Coordinates-Pairs: (R, C) s and (X^W, Y^W) s

(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. (??), 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^W, Y^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. (1.17) (perspective correction) as 1st order polynomial mapping is not able to handle lens distortions. The second order polynomial mapping has $2 \times 6 = 12$ parameters, written as

$$\begin{aligned} X^W &= a_{11}C^2 + a_{12}CR + a_{13}R^2 + a_{14}C + a_{15}R + a_{16} \\ Y^W &= a_{21}C^2 + a_{22}CR + a_{23}R^2 + a_{24}C + a_{25}R + a_{26}, \end{aligned} \quad (1.20)$$

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

$$\begin{aligned} X^W = & a_{11}C^4 + a_{12}C^3R + a_{13}C^2R^2 + a_{14}CR^3 + a_{15}R^4 + a_{16}C^3 + a_{17}C^2R \dots \\ & + a_{18}CR^2 + a_{19}R^3 + a_{110}C^2 + a_{111}CR + a_{112}R^2 + a_{113}C + a_{114}R + a_{115} \end{aligned} \quad (1.21)$$

$$\begin{aligned} Y^W = & a_{21}C^4 + a_{22}C^3R + a_{23}C^2R^2 + a_{24}CR^3 + a_{25}R^4 + a_{26}C^3 + a_{27}C^2R \dots \\ & + a_{28}CR^2 + a_{29}R^3 + a_{210}C^2 + a_{211}CR + a_{212}R^2 + a_{213}C + a_{214}R + a_{215}. \end{aligned}$$

To prototype equation 1.20 and 1.21 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.

1.4 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 1.2, we know that our whole idea of a calibrated natural 3D reconstruction on GPU is to make use of per-pixel eqn. (1.9), which requires undistorted $X^WY^WZ^W$ data and a per-pixel mapping from D to Z^W . With enough $X^WY^WZ^WID$ 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 be 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 1.11 shows the polynomial fitting result in Matlab “Curve Fitting Tool” toolbox, with 32 points of DZ^W values (at pixel $C=256$ and $R=212$) 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

$$Z^W[m, n] = e[m, n]D[m, n] + f[m, n], \quad (1.22)$$

where $[m, n]$ denotes the image address R and C of a pixel, e/f are the corresponding linear coefficients that help map from D to Z^W . With eqn. (1.22) supporting the per-pixel Z^W values and eqn. (1.9) help generating the per-pixel X^WY^W values, we found all of the six

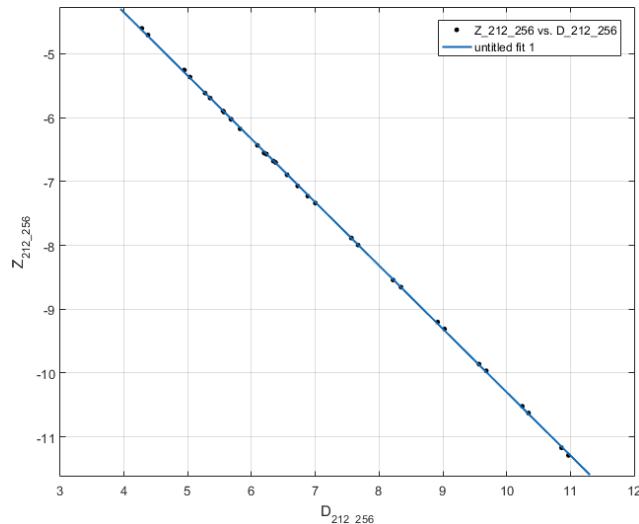


Figure 1.11: Polynomial Fitting between D and Z^W

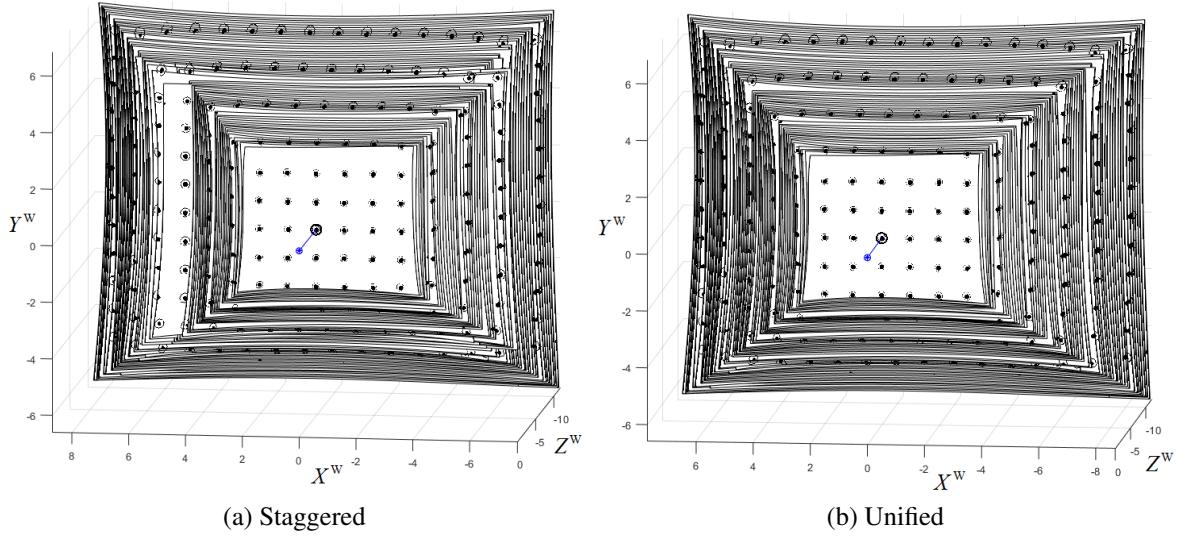


Figure 1.12: World Space Unification of Collected Data

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 1.12a 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. 1.12b, we are all ready to determine the per-pixel mapping parameters $a/b/c/d/e/f$ and generate the LUT.

1.5 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 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 streams and Kinect NearIR streams, a Matlab prototype of RGB pixels alignment is shown in Fig. 1.13a, 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 1.13 shows the screen-shot of live video after calibration with the RGB pixels aligned by a pinhole camera model M .

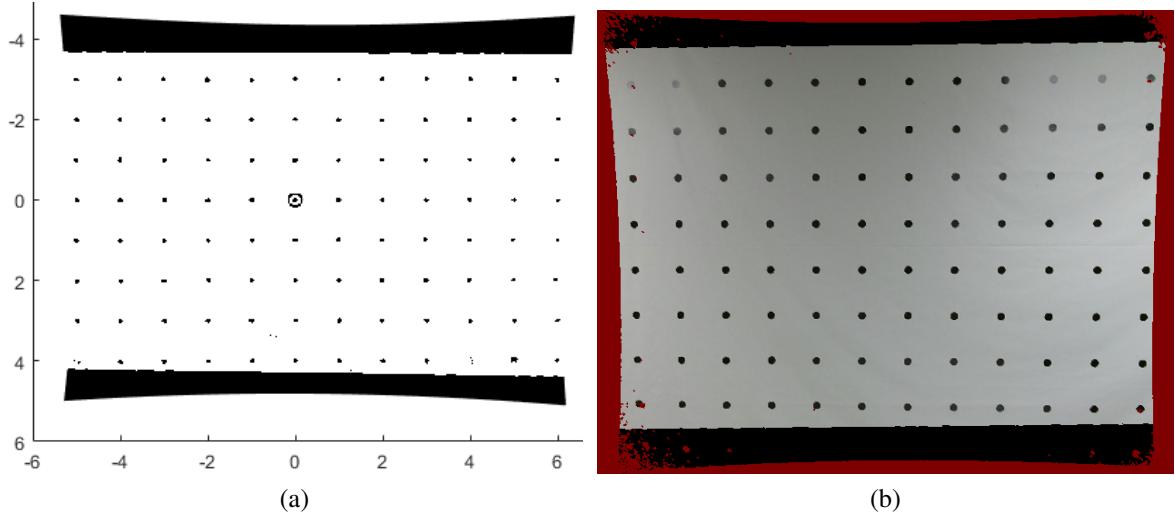


Figure 1.13: Alignment of RGB Texture onto NearIR Image

Copyright[©] Sen Li, 2016.

List of Symbols

3D Three-Dimensional. i, iii, vi, 1, 4–8, 20–23

X^C Camera Space X -axis of Cartesian coordinate. 1–5

Y^C Camera Space Y -axis of Cartesian coordinate. 1–5

Z^C Camera Space Z -axis of Cartesian coordinate. 1–6

C Column coordinate in two-dimensional image space. 3, 6, 21

DIP Digital Image Processing. 10, 11

D Depth value. i, 1, 2, 7, 9, 14–16, 21

FoV Field of View. i, 2–4, 6, 18, 22

GPU Graphics Processing Unit. i, iii, vi, 1, 2, 4–7, 11, 14, 15, 21

KinectV2 The second generation of Kinect RGBD camera, features ToF sensor. 1, 2, 4, 7, 10

LUT Look-Up Table. iii, 20–23

NearIR Near infrared, electromagnetic spectrum from about 700 nm to 2500 nm.. 8, 10–12, 14, 17, 20, 23

RGB-D Red-Green-Blue and Depth. 25

RGBD Same with RGB-D. i, iii, vi, 1, 7, 8, 25

R Row coordinate in two-dimensional image space. 3, 6, 21

SLAM Simultaneous Localization and Mapping. 1

ToF Time-of-Flight. 25

X^W World Space X -axis of Cartesian coordinate. i, 6–8, 10, 16–20, 22

Y^W World Space Y -axis of Cartesian coordinate. i, 6–8, 10, 16–20, 22

Z^W World Space Z -axis of Cartesian coordinate. i, 6–10, 18, 21, 22

Bibliography

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