國立交通大學

電子研究所

碩 士 論 文

二乘二矩陣排列Kinect v2的廣角虛擬視點影像合成

Wide Angle Virtual View Synthesis Using Two-by-Two Matrix Kinect V2

研 究 生：張桓睿

指導教授：杭學鳴 教授

中 華 民 國 一〇六 年 一 月

二乘二矩陣排列Kinect v2的廣角虛擬視點影像合成

Wide Angle Virtual View Synthesis Using Two-by-Two Matrix Kinect V2

研 究 生：張桓睿　　Student：Huan-Rui Chang

指導教授：杭學鳴 教授　　Advisor：Prof. Hsueh-Ming Hang

國 立 交 通 大 學

電子研究所

碩 士 論 文

A Thesis
Submitted to
Institute of Electronics
College of Electrical and Computer Engineering
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Master of Science
in
Electronics Engineering

January 2017
Hsinchu, Taiwan, Republic of China

中 華 民 國 一〇六 年 一 月

二乘二矩陣排列Kinect v2的廣角虛擬視點影像合成

研究生:張桓睿　　指導教授:杭學鳴 教授

國立交通大學

電子研究所

摘要

虛擬視影像合成是使用多台相機在同一時間點所拍攝到的彩色影像與深度圖來進行虛擬視點合成。傳統的虛擬視點合成系統一般使用兩台(以上)的相機排在同一水平位置上來拍攝場景，並且合成出相機基線上的虛擬視點影像。然而，在許多應用上我們可能想合成的虛擬視點影像是位於相機基線以外的部分，這種合成架構我們稱之為廣角虛擬視點合成。廣角虛擬視點合成需要解決合成結果有許多小的破洞以及大的遮蔽區。在本論文中，我們採用二乘二矩陣排列Kinect v2架構來設計我們的廣角虛擬視點合成系統。

在本論文中，我們的目標是實現一個可以運用在實際情況的廣角虛擬視點合成系統。主要有兩個貢獻，第一個貢獻是我們解決了四台Kinect v2之間無法完全同步拍攝的問題。由於Kinect v2並沒有提供可控制的外部同步訊號，因此四台Kinect v2彼此拍攝的時間點並不完全一致。我們使用時間校準軟體來解決四台Kinect v2彼此不同步拍攝的問題。本論文的第二個貢獻是我們提出了多視點圖融合演算法，這個方法可以濾除因為原始拍攝到的彩色影像與深度圖沒有對的很精準所造成的錯誤映射像素，同時也可有效改善了合成的影像品質。我們所提出的多視點圖融合演算法分成三個部分。第一個部分是針對不在深度邊緣周圍的像素使用主虛擬視點的彩色及深度像素來合成。第二個部

分會解決紋理與深度未對準以及虛擬視點的紋理之間未對準所造成的雜訊。第三個部分會從四個映射的虛擬視點中選取最佳匹配的像素來合成剩餘的虛擬視點像素。最後，我們觀察虛擬視點相機在不同傾斜或縮放情況下的合成結果，本論文所提出的多視點圖融合演算法在實際情況下可以有好的主觀合成影像品質。

Wide-Angle Virtual View Synthesis Using Two-by-Two Matrix Kinect V2

Student: Huan-Rui Chang Advisor: Prof. Hsueh-Ming Hang

Institute of Electronics

National Chiao Tung University

Abstract

Virtual view synthesis technique uses the color images and depth maps captured by multiple cameras to synthesize a virtual view image. The conventional view synthesis system typically arranges two or more cameras along a baseline line to capture and to synthesize the virtual view lies on the baseline. However, in many applications we may want to synthesize a virtual view beyond the baseline, which is called wide-angle view synthesis. Wide-angle view synthesis needs to solve the problems of many small cracks and large disocclusion regions in the synthesized virtual view. In this thesis, we adopt a two-by–two array of Kinect v2 cameras to design our wide-angle view synthesis system.

In this thesis, our target is practical real world case wide-angle view synthesis system. There are two key contributions in this thesis. The first is that we solve the synchronization problem among multiple Kinects in capturing images. Without external sync signal, the captured scenes of four Kinects are not exactly at the same time instance. We implement a clock adjustment system to solve this problem based on PC clock synchronization software. The second contribution is that we propose a multi-view blending algorithm that can remove

the wrong pixels due to texture-depth misalignment and texture-texture misalignment and provide a clear improvement on the synthesized image quality. The proposed multi-view blending algorithm consists of three parts. The first part picks up the dominate reference view to synthesize the corresponding virtual image pixels. The second part filters out the noises due to texture-depth misalignment and texture-texture misalignment. The third part chooses the best matched color pixels from the four warped virtual views to synthesize the remaining holes. At the end, we examine the quality of synthesized views for various camera tilt and zoom-in/out cases. The proposed multi-view blending algorithm achieves good subjective synthesized image quality in these real world cases.

誌謝

在這兩年的碩士研究生涯中，我非常榮幸的能在 杭學鳴教授的指導下完成我的碩士學位。非常感謝老師對我的指導，老師不僅在研究上指導我，也時常關心我們、與我們分享他的人生經驗，老師謙虛與正面的態度更是令人敬佩。老師不僅給予我在時間上的安排有很大的自由，更支持我的人生規劃，能在杭老師的指導下完成碩士學位是我的榮幸，桓睿對杭老師兩年的指導非常感謝。

感謝錢鈞亮博士、徐震濤博士時常與我們討論研究內容，讓我們的研究得以更加順利，同時也與我們分享許多經驗、幫實驗室做了許多事務。感謝游鈞元學長、李嵩聲學長對我的照顧、提攜。感謝 Jan(韋瑞安)、Sandy(余宏善)、David(許大為)、Patrisia(溫夏夢)、Amanda(王愛梅)、冠諭、敬昆、司頻、建成、哲瑋、治戎、黃暄、姿瑾、韶儀、玉芬、映瑄、科諭、逸文、家維、繁偉、余祥…等，感謝學長姐們對我在研究上的建議、生活以及課業上的照顧、找工作的建議與提攜，對我幫助很大。感謝實驗室夥伴們:士傑、維倫、詩晴、文廣、泰穎、辰泰、治維、宗彬、聖曄、佳容、奕翔、聖楓、宇倫、Mykola，與我討論課業、研究，有你們讓實驗室增添了許多歡笑。感謝學弟妹們:韶儀、東廷、品融，在我口試時候的幫忙。

能順利獲得碩士學位，最感謝我的父母、家人們讓我無憂無慮的完成碩士論文的研究，更給予我最大的支持、鼓勵，如果沒有你們的支持，我無法如此順利的取得碩士學位。我將此碩士學位獻給我最敬愛的家人!

在交大兩年期間受到許多人的幫助，若有遺漏您，在此一併感謝。

張桓睿

謹致於 交通大學

西元二０一七年一月

Content

摘要 i

Abstract iii

誌謝 v

Content vi

List of Figures x

List of Table xix

1. Introduction 1

1.1. Background 1

1.2. Motivations and Contributions 3

1.3. Organizations of Thesis 4

2. Camera Model and Calibration Algorithms 5

2.1. Pinhole Camera Model 5

2.2. Camera Calibration 7

2.2.1. Camera Intrinsic Parameters 8

2.2.2. Camera Extrinsic Parameters 9

2.2.3. Lens Distortion 9

2.3. Image Rectification 10

2.3.1. Stereo Image Rectification 12

2.3.2. Multi-camera Image Rectification 13

2.4. Disparity and Depth Estimation 17

2.5. Color Calibration 19

3. View Synthesis Basics 20

3.1. 3D Warping 20

3.1.1. Forward Depth Warping (FDW) 20

3.1.2. Backward Depth Warping (BDW) 23

3.1.3. Texture Warping 24

3.2. Virtual View Merging 25

3.3. Hole Filling 27

3.4. Depth Camera 28

4. Depth Map Refinement for Kinect V2 31

4.1. Background Construction 31

4.2. Alignment Between Color Image and Depth Map 33

4.3. Occlusion Region Detection and Filling 35

4.4. Detection and Refinement for Unreliable Depth Pixels and Reflection Regions 36

5. 3D Warping Before Wide Angle View Merging 41

5.1. Disocclusion Hole Detection 41

5.2. Fast Backward Depth Warping 42

5.3. Fast Backward Texture Warping 42

6. Proposed Wide Angle View Synthesis Algorithm 44

6.1. Synchronizing all the four RGB-D cameras 46

6.2. Multi-view Blending 47

7. Experimental Results and Discussions 59

7.1. First Test Sequence 60

7.1.1. Camera Calibration 63

7.1.2. Multi-camera Image Rectification 66

7.1.3. Depth Map Refinement for Kinect v2 Before 3D Warping 68

7.1.4. 3D Warping Before Wide Angle View Merging 69

7.1.5. Synchronizing All Four RGB-D Cameras 71

7.1.6. Multi-view Blending 74

7.1.6.1. Without Post-warped Depth Edge Dilation 75

7.1.6.2. With Post-warped Depth Edge Dilation 76

7.1.6.3. Results of Multi-view Blending Algorithm Part 1 77

7.1.6.4. Results of Multi-view Blending Algorithm Part 2 78

7.1.6.5. Results of Multi-view Blending Algorithm Part 3 79

7.1.6.6. Case 1: Tilt Down 2 Degrees, and Zoom In 100mm 80

7.1.6.7. Case 2: Tilt Down 5 Degrees, and Zoom In 100mm 83

7.1.6.8. Case 3: Zoom In 200mm 87

7.1.6.9. Case 4: Zoom In 400mm 91

7.1.6.10. Case 5: Zoom Out 400mm 95

7.2. Second Test Sequence 100

7.2.1. Multi-camera Image Rectification 103

7.2.2. Depth Map Refinement for Kinect v2 Before 3D Warping 104

7.2.3. 3D Warping Before Wide Angle View Merging 105

7.2.4. Multi-view Blending 106

7.2.4.1. Case 1:Title Down 5 Degrees, and Zoom In 100mm 106

7.2.4.2. Case 2: Zoom In 400mm 112

7.2.4.3. Case 3: Zoom Out 400mm 117

8. Conclusions 122

References 123

List of Figures

Figure 1-1 Architecture of 3D video system [1]. 2

Figure 1-2 Illustration of virtual view synthesis. 2

Figure 2-1 Pinhole camera model [2] 6

Figure 2-2 Transformation between world coordinate and camera coordinate [2]. 7

Figure 2-3 (a) Epipolar geometry. (b) Epipolar line and baseline. (c) Epipole [2]. 11

Figure 2-4 Rectified epipolar geometry [13]. 12

Figure 2-5 Flow chart of multi-camera rectification [16]. 14

Figure 2-6 Architecture of matrix multi-camera arrangement [16]. 15

Figure 2-7 (a) Original un-rectified images. (b) Rectified images [16]. 17

Figure 2-8 Rectified image [17]. 18

Figure 2-9 Histogram matching. 19

Figure 3-1 Flow chart of view synthesis. 21

Figure 3-2 3D projection between reference view and virtual view. 21

Figure 3-3 Problems of forward depth warping. 23

Figure 3-4 Backward depth warping. 24

Figure 3-5 Bicubic interpolation. 25

Figure 3-6 (a) Kinect™ for Xbox 360® [38]. (b) Kinect™ for Xbox One® [40]. 28

Figure 3-7 Kinect™ for Xbox One® teardown [41]. 29

Figure 3-8 Background-to-foreground occlusion problems [42]. 29

Figure 4-1 Flow chart of the overall depth enhancement algorithm for Kinect v2 [43]. 31

Figure 4-2 Flow chart of background construction [43]. 32

Figure 4-3 Flow chart of alignment between color image and depth map [43]. 34

Figure 4-4 Illustration of occlusion region [43]. 36

Figure 4-5 Flow chart of detecting unreliable depth pixels [43]. 37

Figure 4-6 Flow chart of processing the unreliable depth pixels and holes [43]..... 38

Figure 4-7 Flow chart of refining the reflection regions [43]..... 39

Figure 5-1 Flow chart of 3D warping before wide angle view merging [29]. 41

Figure 5-2 (a) reference color image (b) reference background color image (c) the pixels of same values in the reference color and reference background color image (d) virtual depth map (e) virtual background depth map (f) the pixels of the same values in virtual depth and virtual background depth map (g) pixels in virtual view whose color is the same as virtual background and filled by virtual background (h) the remaining black holes are filled by the bicubic interpolation [29]..... 43

Figure 6-1 Flowchart of the proposed wide angle view synthesis system. 45

Figure 6-2 The four color images captured from four Kinect v2 cameras without solving asynchronization problem. 46

Figure 6-3 Four color images captured by four Kinect v2 cameras at the same time. 47

Figure 6-4 Illustration of misalignment between reference color and depth cause wrongly warped pixels. 48

Figure 6-5 Illustration of the wrongly warped pixels due to texture-depth misalignment. 48

Figure 6-6 Illustration of misalignment between four warped virtual views. 49

Figure 6-7 Illustration of the blending result with the effect texture-texture misalignment. ... 50

Figure 6-8 Flow chart of extracting *MarkedPoint*. 51

Figure 6-9 Flow chart of the proposed multi-view blending algorithm. 51

Figure 7-1 Flowchart of the proposed wide angle view synthesis system. 59

Figure 7-2 Setup of four Kinect v2 sensors. 60

Figure 7-3 1st frame of the sequences captured from (a) camera 1 (b) camera 2 (c) camera 3 (d) camera 4. 61

Figure 7-4 2st frame of the sequences captured from (a) camera 1 (b) camera 2 (c) camera 3 (d)

camera 4. 61

Figure 7-5 3st frame of the sequences captured from (a) camera 1 (b) camera 2 (c) camera 3 (d) camera 4. 62

Figure 7-6 4st frame of the sequences captured from (a) camera 1 (b) camera 2 (c) camera 3 (d) camera 4. 62

Figure 7-7 Flowchart of camera calibration. 64

Figure 7-8 Calibration patterns. 65

Figure 7-9 Extract grid corners for computing camera intrinsic parameter. 65

Figure 7-10 Extract grid corners for computing camera extrinsic parameter. 65

Figure 7-11 Un-rectified color image of four reference views. 67

Figure 7-12 Rectified color image of the four reference views. 68

Figure 7-13 Raw depth map. 69

Figure 7-14 Refined depth map. 69

Figure 7-15 Original reference views of the four cameras. 70

Figure 7-16 Warped views of the four reference views with virtual camera tilt down 5 degrees and zoom in 100mm. 70

Figure 7-17 Geometric relationship of the virtual camera and the four reference cameras. 71

Figure 7-18 Four color images captured from the four Kinect v2 cameras with asynchronization problem. 72

Figure 7-19 Synthesized image using average blending and maximum capturing time skew is 0.12 seconds. 72

Figure 7-20 Synthesized image using average blending and maximum capturing time skew is 0.81 seconds. 73

Figure 7-21 Four color images captured by four Kinect v2 cameras at the same time instance. 73

Figure 7-22 Synthesized image using average blending and maximum capturing time skew is

0.00 second. 74

Figure 7-23 Synthesized image without post-warp depth edge dilation, but use multi-view blending algorithm (a) part 1 (b) part 1 and part 2 (c) part 1, 2, and 3. 75

Figure 7-24 Synthesized image using post-warp depth edge dilation and multi-view blending algorithm (a) part 1 (b) part 1 and part 2 (c) part 1, 2, and 3. 76

Figure 7-25 Blending result of multi-view blending algorithm part 1. 77

Figure 7-26 Iteratively blending results of multi-view blending algorithm part 2 in (a) 4th iteration (b) 8th iteration (c) 13th iteration. 78

Figure 7-27 Iteratively blending results of multi-view blending algorithm part 3 in (a) 6th iteration (b) 12th iteration (c) 18th iteration. 79

Figure 7-28 Geometric relationship between the four reference cameras and the synthesized virtual camera. 80

Figure 7-29 2nd frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera tilt down 2 degrees and zoom in 100mm. 81

Figure 7-30 3rd frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera tilt down 2 degrees and zoom in 100mm. 82

Figure 7-31 Geometric relationship between the four reference cameras and the synthesized virtual camera. 83

Figure 7-32 1st frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera tilt down 5 degrees and zoom in 100mm. 84

Figure 7-33 2nd frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera tilt down 5 degrees and zoom in 100mm. 85
Figure 7-34 3rd frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera tilt down 5 degrees and zoom in 100mm. 86
Figure 7-35 Geometric relationship between the four reference cameras and the synthesized virtual camera. 87
Figure 7-36 1st frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera zoom in 200mm. 88
Figure 7-37 2nd frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera zoom in 200mm. 89
Figure 7-38 3rd frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera zoom in 200mm. 90
Figure 7-39 Geometric relationship between the four reference cameras and the synthesized virtual camera. 91
Figure 7-40 1st frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c)

the proposed multi-view blending algorithm with four cameras. The virtual camera zoom in 400mm. 92

Figure 7-41 2nd frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera zoom in 400mm. 93

Figure 7-42 3rd frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera zoom in 400mm. 94

Figure 7-43 geometric relationship between the four reference cameras and the synthesized virtual camera. 95

Figure 7-44 1st frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera zoom out 400mm. 96

Figure 7-45 2nd frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera zoom out 400mm. 97

Figure 7-46 3rd frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera zoom out 400mm. 98

Figure 7-47 4th frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c)

the proposed multi-view blending algorithm with four cameras. The virtual camera zoom out 400mm. 99

Figure 7-48 20st frame of the sequences captured from (a) camera 1 (b) camera 2 (c) camera 3 (d) camera 4. 100

Figure 7-49 21st frame of the sequences captured from (a) camera 1 (b) camera 2 (c) camera 3 (d) camera 4. 101

Figure 7-50 22st frame of the sequences captured from (a) camera 1 (b) camera 2 (c) camera 3 (d) camera 4. 101

Figure 7-51 23st frame of the sequences captured from (a) camera 1 (b) camera 2 (c) camera 3 (d) camera 4. 102

Figure 7-52 Un-rectified color image of the four reference views. 103

Figure 7-53 Rectified color image of the four reference views. 103

Figure 7-54 Raw depth map. 104

Figure 7-55 Refined depth map. 104

Figure 7-56 Geometric relationship between the four reference cameras and the synthesized virtual camera. 105

Figure 7-57 Original reference views of the four cameras. 105

Figure 7-58 Warped views of the four reference views with virtual camera tilt down 5 degrees and zoom in 100mm. 106

Figure 7-59 Geometric relationship between the four reference cameras and the synthesized virtual camera. 107

Figure 7-60 20th frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera tilt down 5 degrees and zoom in 100mm. 108

Figure 7-61 21th frame of final blending results by using (a) adaptive blending with upper

two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera tilt down 5 degrees and zoom in 100mm. 109

Figure 7-62 22th frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera tilt down 5 degrees and zoom in 100mm. 110

Figure 7-63 23th frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera tilt down 5 degrees and zoom in 100mm. 111

Figure 7-64 Geometric relationship between the four reference cameras and the synthesized virtual camera. 112

Figure 7-65 20th frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera zoom in 400mm. 113

Figure 7-66 21th frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera zoom in 400mm. 114

Figure 7-67 22th frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera zoom in 400mm. 115

Figure 7-68 23th frame of final blending results by using (a) adaptive blending with upper

two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera zoom in 400mm. 116

Figure 7-69 Geometric relationship between the four reference cameras and the synthesized virtual camera. 117

Figure 7-70 20th frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera zoom out 400mm. 118

Figure 7-71 21th frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera zoom out 400mm. 119

Figure 7-72 22th frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera zoom out 400mm. 120

Figure 7-73 23th frame of final blending results by using (a) adaptive blending with upper two cameras [29] (b) the proposed multi-view blending algorithm with upper two cameras (c) the proposed multi-view blending algorithm with four cameras. The virtual camera zoom out 400mm. 121

List of Table

Table 3-1 Comparison of commercial depth cameras. 28

1. Introduction

1.1. Background

Recently, with the development of science and technology, multimedia has increasing more applications in our daily life. For example, 3DTV such as auto-stereoscopic TV, stereoscopic TV, multi-view TV, free-viewpoint television (FTV) can provide a 3D impression to the users. The auto-stereoscopic television can display 3D contents without wearing 3D-glasses. Free-viewpoint television (FTV) can display any viewpoint specified by the users.

It is hard to transmit all the view information to the user due to the bandwidth limitations. Therefore, the depth image based rendering (DIBR) technique is often used to generate the new views. DIBR is a popular view synthesis method adopted by the ITU/MPEG standard. By using the DIBR technique and 3D video coding, the transmitter only transmits the coded color and depth information of the reference views to the receiver. When the receiver receives this information, it applies the decoding procedures to recover the original color and depth information. And then the view synthesis algorithm synthesizes the virtual viewpoint pictures, which is specified by the user. Figure 1-1 gives an illustration of the system.

Virtual view image can be synthesized by using the reference color and depth images captured from different viewpoints, as shown in Figure 1-2. With the help of view synthesis algorithm, we can generate arbitrary viewpoints. Furthermore, it is impossible to capture all possible viewpoints by RGB-D cameras. Therefore, view synthesis plays an important role in the 3D content generation.

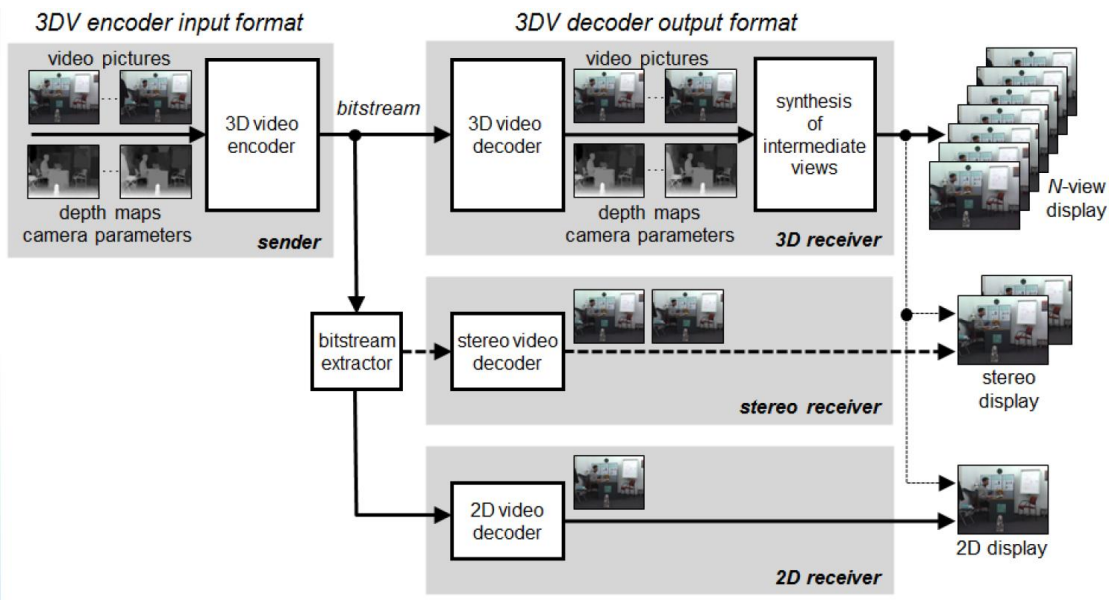


Figure 1-1 Architecture of 3D video system [1].

Figure 1-2 Illustration of virtual view synthesis.

1.2. Motivations and Contributions

There are many papers studying view synthesis based on 1-D array of cameras along the baseline. However, many applications want to synthesize the virtual views beyond the baseline like virtual camera zoom in/out or tilt up/down, which may be called wide angle view synthesis. Most of the conventional view synthesis uses the given reference cameras arranging along 1-D baseline. This type of systems cannot do wide angle view synthesis well. Therefore, in this thesis we use four Kinect v2 sensors arranging in two-by-two matrix manner for image capturing and view synthesis. A 2-D array of cameras can cover more view information than the traditional 1-D camera array.

There are two main contributions in this thesis. First, we solve the synchronization problem between four Kinect v2 sensors. Second, we propose a multi-view blending algorithm that is consisted of three parts. The first part of the proposed multi-view blending algorithm renders the pixels that are not near the depth edge region. The second part renders the pixels near the depth edge regions. It iteratively generates the virtual view by choosing the best matched color pixel from the four warped virtual views. The third part is similar to the second part in algorithm. It chooses the best matched color pixels from the warped four virtual views to render the synthesized virtual view for the holes. To evaluate effectiveness of the proposed multi-view blending algorithm, we capture two different sequences. The first sequence is a toy bear moving cross the table top; it consists of 7 frames at 1fps. The second sequence is one toy bear rotating and the other bear stationary on the table; it consists of 49 frames at 1fps. Experimental results show that we can reduce the noise and achieve a good subjective synthesized image quality by using the proposed multi-view blending algorithm.

1.3. Organizations of Thesis

In this thesis, we first introduce the camera model and camera calibration algorithms in Section 2. The basic view synthesis schemes are described in Section 3. Section 4 describes the depth map refinement algorithm adopted in this thesis. Section 5 presents the preprocessing process before the wide angle view merging. We introduce the proposed wide angle view synthesis algorithm in Section 6. Experimental results and discussions are shown in Section 7. Finally, we state the conclusions in Section 8.

2. Camera Model and Calibration Algorithms

In this chapter, we will introduce the basic concepts of camera model and camera calibration. In computer vision, it is important to know the camera parameters. View synthesis will use camera parameters to synthesize the virtual views. The camera parameters can be obtained by the camera calibration procedure. After that we can do image rectification to make the epipolar lines parallel. Image rectification cannot only rectify the epipolar lines, but also make sure the rectified camera parameters more precise. The more precise parameters we have, the better synthesized image quality we can achieve. The depth information plays an important role in a computer vision system. The depth information can be calculated from two aligned images or it can be captured by an active RGB-D camera such as Kinect V2 or SR4000. For more details, please see the sections below.

2.1. Pinhole Camera Model

The pinhole camera model [2] is a simplified camera model. Most of the cameras have multiple lenses. For the analysis purpose, we use pinhole camera model to design our basic algorithm. Figure 2-1 illustrates the pinhole camera model. There are two kinds of camera parameters. The first one is the camera intrinsic parameter, which consists of focal length, principal point and skew factor. Another set of camera parameter is the camera extrinsic parameter. It represents the transformation between the coordinates of camera and 3D world. The camera extrinsic parameter consists of rotation matrix and translation vector. The image plane is on the plane with z=f. Assume $P(X_{world}, Y_{world}, Z_{world})$ is a 3D world point. It is mapped to the coordinates of image plane by (1) using similar triangles.

$$p=(u,v)=\left(\frac{fX_{world}}{Z_{world}}, \frac{fY_{world}}{Z_{world}}\right), \quad (1)$$

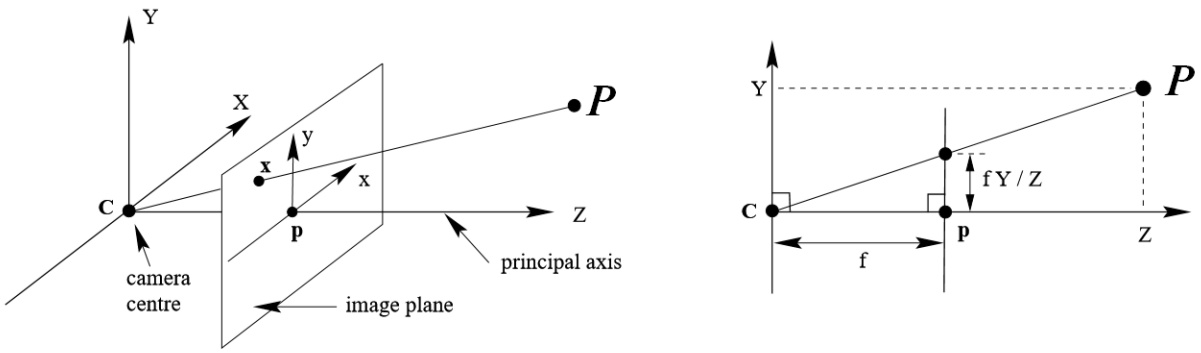


Figure 2-1 Pinhole camera model [2]

where the *f* is focal length. Taking the principal point (u_0,v_0) and skew factor into account, we can obtain (2).

$$p = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{z} K P_{cam} = \frac{1}{z} K[I\ 0]P$$

$$= \begin{bmatrix} f_x & skew & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} X_{world} \\ Y_{world} \\ Z_{world} \\ 1 \end{bmatrix}, \quad (2)$$

with

$$u = u_0 + f\frac{X_{world}}{Z_{world}} \quad (3)$$

and

$$v = v_0 + f\frac{Y_{world}}{Z_{world}}, \quad (4)$$

where the *P* represents a 3D world position, and *p* represents the coordinate of *P* on the image plane. [*I* 0] in (2) is used to normalize the coordinates to z=1 plane. f_x and f_y are the focal length of x-axis and y-axis, respectively. Focal length is used to map the image to the image plane of z=f. u_0 and v_0 are principal points in x-axis and y-axis, respectively. The skew factor is due to the lens skew. It represents the degree skewness of the two image axes. If $skew \neq 0$, then a straight line has a little tilt. For most cases, the skew factor is close to zero.

Next we looked the camera extrinsic parameters. The transformation between the 3D world coordinate and the camera coordinate can be represented by a rotation matrix and a translation vector, as shown in Figure 2-2 and (5).

$$p_{cam} = \begin{bmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \end{bmatrix} = R * P + t = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \begin{bmatrix} X_{world} \\ Y_{world} \\ Z_{world} \end{bmatrix} + \begin{bmatrix} t_{11} \\ t_{21} \\ t_{31} \end{bmatrix}. \quad (5)$$

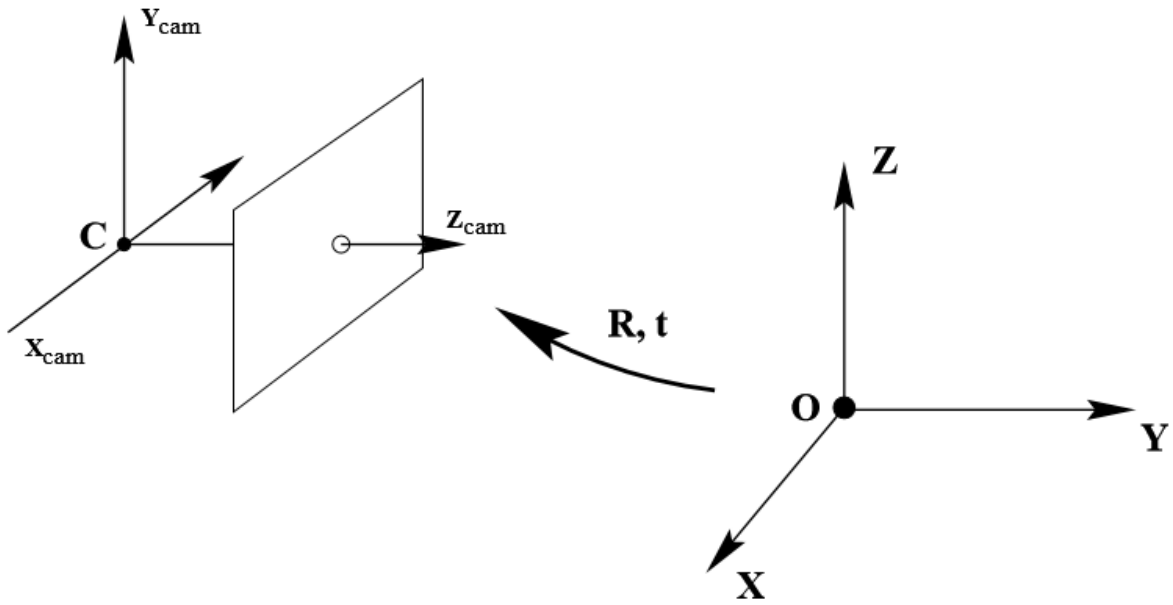


Figure 2-2 Transformation between world coordinate and camera coordinate [2].

Points O and C are the origins of world coordinate and camera coordinate, respectively. Combining the camera intrinsic and extrinsic parameter, we obtain (6).

$$p = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{z} K P_{cam} = \frac{1}{z} K[I\ 0][R \mid t\ ]P = \frac{1}{z} K[R \mid t\ ]P = \frac{1}{z} MP$$

$$= \begin{bmatrix} f_x & skew & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_{11} \\ R_{21} & R_{22} & R_{23} & t_{21} \\ R_{31} & R_{32} & R_{33} & t_{31} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_{world} \\ Y_{world} \\ Z_{world} \\ 1 \end{bmatrix}, \quad (6)$$

where $M = K[R \mid t\ ]$ is projection matrix.

2.2. Camera Calibration

The camera calibration algorithm estimates the camera intrinsic and extrinsic parameters. These parameters define the transformation relationship between a 2D image and the 3D world space. There are many papers they have studied the camera calibration methods [3-11].

References [3, 4] are two well-known papers on camera calibration.

2.2.1. Camera Intrinsic Parameters

To compute the projection matrix $M = K[R \mid t]$ mentioned in Section 2.1, we first use the closed-form formula to estimate the intermediate results, and then use the maximum likelihood estimation to do the nonlinear optimization [3]. The following derivations are provided by [3].

Define

$$\mathrm{B} = K^{-T}K^{-1} \equiv \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix}. \tag{7}$$

Let b= $[B_{11}, B_{12}, B_{22}, B_{13}, B_{23}, B_{33}]$, because B is symmetric. Let $m_i = [m_{i1},\ m_{i2},\ m_{i3}]^T$ be the ith column vector of projection matrix *M*, then we can derive that $m_i^T B m_i = v_{ij}^T b$. Assume the number of images used to evaluate the camera parameters is *n*. Then, we can solve $V_{2n*6}b = 0$. The solution of $V_{2n*6}b = 0$ is the eigenvector of $V^T V$ associated with the smallest eigenvalue. Once we obtain *b*, we can estimate the camera intrinsic parameter by (8) ~ (13).

$$v_o = (B_{12}B_{13} - B_{11}B_{23})/(B_{11}B_{22} - B_{12}^2), \tag{8}$$

$$\lambda = B_{33} - [B_{13}^2 + v_o(B_{12}B_{13} - B_{11}B_{23})]/B_{11}, \tag{9}$$

$$f_x = \sqrt{\lambda/B_{11}}, \tag{10}$$

$$f_y = \sqrt{\lambda B_{11}/(B_{11}B_{22} - B_{12}^2)}, \tag{11}$$

$$skew = -B_{12}f_x^2 f_y / \lambda, \tag{12}$$

$$u_o = \frac{\gamma v_o}{\beta} - B_{13}f_x^2 / \lambda. \tag{13}$$

2.2.2. Camera Extrinsic Parameters

Now we already have the camera intrinsic parameter *K*. We can calculate the camera extrinsic parameters R=[r_1, r_2, r_3] and *t* by

$$r_1 = \lambda K^{-1} m_1, \tag{14}$$

$$r_2 = \lambda K^{-1} m_2, \tag{15}$$

$$r_3 = r_1 \times r_2, \tag{16}$$

$$t = \lambda K^{-1} m_3. \tag{17}$$

After finishing the above closed-form evaluation, we will carry out the maximum likelihood estimation for nonlinear optimization by (18).

$$\sum_{i=1}^{n} \sum_{j=1}^{m} \left\| p_{ij} - \hat{p}(K, R_i, t_i, P_j) \right\|^2, \tag{18}$$

where *n* is the number of calibration patterns, and *m* is the extracted feature points from the calibration patterns. p_{ij} is the coordinate in image. $\hat{p}(K, R_i, t_i, P_j)$ represents the coordinate in image projected from a 3D world point P_j. Levenberg-Marquardt Algorithm will be used to calculate (18). For more details about computing camera extrinsic parameters, please refer to [3].

2.2.3. Lens Distortion

The lens distortion can also be solved by the camera calibration procedure. Assume (u_{ideal}, v_{ideal}) and (u,v) are the non-distorted and distorted image coordinates, respectively. The relationship between (u_{ideal}, v_{ideal}) and (u,v) can linked by (19) and (20).

$$u_{ideal} = u + (u - u_o) * [k_1(x^2 + y^2)^2 + k_2(x^2 + y^2)^2], \tag{19}$$

$$v_{ideal} = v + (v - v_o) * [k_1(x^2 + y^2)^2 + k_2(x^2 + y^2)^2], \tag{20}$$

where k_1 and k_2 are the radial distortion coefficients. Taking into account the radial

distortion, the maximum likelihood estimation can be modified to become (21). For more details about radial distortion, please refer to [3].

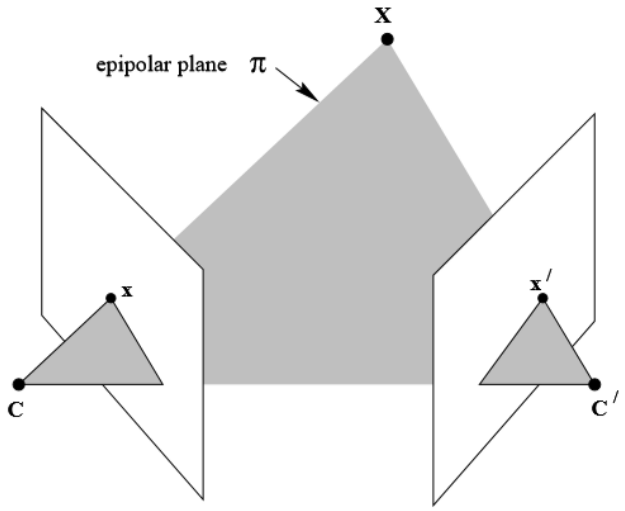
$$\sum_{i=1}^{n}\sum_{j=1}^{m}\left\|p_{ij}-\hat{p}(K,R_i,t_i,P_j,k_1,k_2)\right\|^2. \tag{21}$$

In our research, we use the camera calibration toolbox [12] to calculate the camera intrinsic and extrinsic parameters. This toolbox was developed based on [4, 5]. The complete camera calibration flow chart in our work will be described in Section 7.

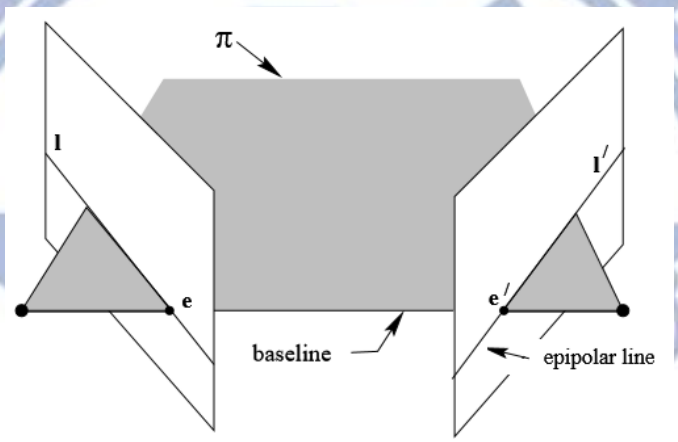
2.3. Image Rectification

The image rectification procedure [2, 13-16] adjusts the epipolar lines between two views to make them aligned. Moreover, image rectification can also reduce the computational complexity of 3D warping and disparity estimation.

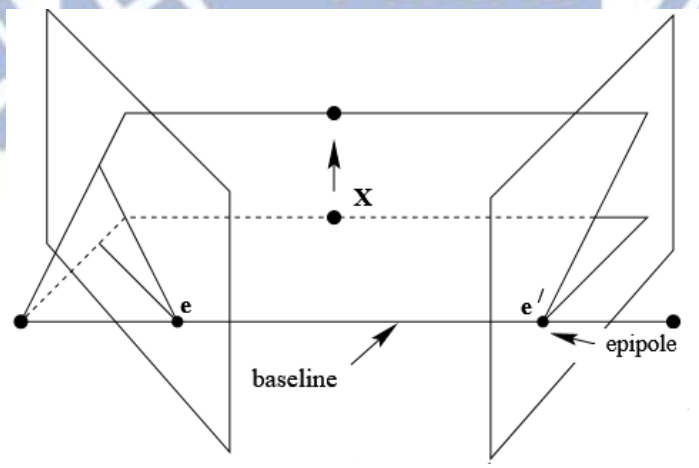
Assume C and C' are the optical centers of the left camera and the right camera, respectively. X is a point in 3D world space. x and x' are the projected points on the left image and right image, respectively. There exist an epipolar plane π, as shown in Figure 2-3(a). The epipolar plane π intersects with the left image plane and right image plane with epipolar line l and l', and the distance between C and C' is called baseline, as illustrated in Figure 2-3(b). If there exist many different 3D world points, the obtained epipolar line may be different. All the epipolar lines are connected to the epipole, as shown in Figure 2-3(c). After finishing image rectification, the epipolar lines will be aligned, as illustrated in Figure 2-4. For more details about image rectification, please refer to [2].



(a)



(b)



(c)

Figure 2-3 (a) Epipolar geometry. **(b)** Epipolar line and baseline. **(c)** Epipole [2].

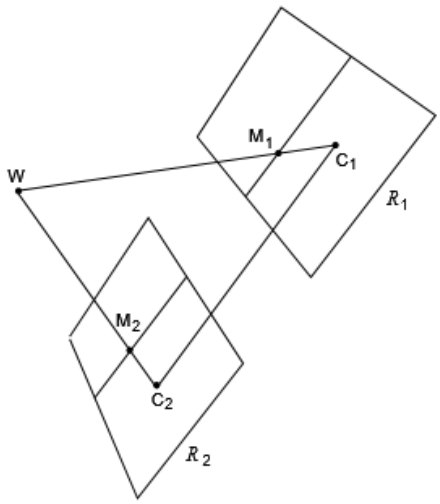


Figure 2-4 Rectified epipolar geometry [13].

2.3.1. Stereo Image Rectification

To rectify the stereo images, we need to rotate the axes of X, Y, Z [13]. Assume the original perspective projection matrix (PPM) and the rotation vector are denoted by (22) and (23). The rectified projection matrix and the rotation vector are represented by (24) and (25).

$$PPM_{i,old} = K_{i,old}[R_{i,old}|t_{i,old}] \ , \ i=1,\ 2, \tag{22}$$

$$R_{old} = \begin{bmatrix} X_{old}^T \\ Y_{old}^T \\ Z_{old}^T \end{bmatrix}, \tag{23}$$

$$PPM_{i,new} = K_{new}[R_{i,new}|t_{i,new}] \ , \ i=1,\ 2, \tag{24}$$

$$R_{new} = \begin{bmatrix} X_{new}^T \\ Y_{new}^T \\ Z_{new}^T \end{bmatrix}. \tag{25}$$

In order to achieve horizontally aligned epipolar lines, we need to make the rectified X axis parallel to the baseline. Eq. (26) is the calculation of the optical centers. c_1 and c_2 are the optical centers of left camera and right camera, respectively. The rectified X axis will be parallel to the baseline by using (27). The rectified Y and Z axes can be computed by (28) and (29).

$$c_i = -\mathrm{inv}(R_i) * t_i \ , \ i=1,\ 2, \tag{26}$$

$$X_{new} = \frac{c_2 - c_1}{\|c_2 - c_1\|}, \tag{27}$$

$$Y_{new} = Z_{old3}^T \times X_{new}, \tag{28}$$

$$Z_{new} = X_{new} \times X_{new}. \tag{29}$$

The rectified translation vectors of the left and right cameras can be estimated from (30) and (31).

$$t_{1,new} = -R_{new} * c_1, \tag{30}$$

$$t_{2,new} = -R_{new} * c_2. \tag{31}$$

The rectified camera intrinsic parameter is the average of left and right camera intrinsic parameters, as listed in (32).

$$K_{new} = (K_{1,old} + K_{2,old})/2. \tag{32}$$

Once we have obtained the rectified camera parameters, we can calculate the image rectifying transformation matrix (IRTM) by (33).

$$IRTM_i = PPM_{i,new} * PPM_{i,old}^{-1} \ , \ i=1, \ 2. \tag{33}$$

We can generate the rectified left and right images by applying (33) to the un-rectified left and right images. For more details about stereo image rectification, please refer to [13].

2.3.2. Multi-camera Image Rectification

It is hard and inefficient to rectify multi-camera arranged in 2-D matrix form by using the algorithm in [13]. For instance, if we use the strategy [13] to rectify four cameras in two-by-two matrix arrangement, we iteratively choose two of the four cameras and execute the image rectification procedures many times. Therefore, some scholars have developed a multi-view image rectification algorithm which is suitable and robust for rectifying multi-camera in matrix arrangement [16].

Figure 2-5 shows the flow chart of the multi-camera image rectification. In the image rectification procedures it uses the camera parameters obtained from the camera calibration

phase. First, it estimates the ideal camera parameters. Second, it calculates the image rectifying transform matrixes (IRTM). Third, it applies the IRTM to the un-rectified images to achieve multi-view image rectification.

Figure 2-6 shows the arrangement of multi-camera, where M is a point in world coordinate system, and m_{ij} is the projected point to the image plane of the camera in the ith row and j^{th} column.

There are four ideal camera parameters to be decided in this algorithm. First, we evaluate the ideal rotation vector by minimizing (38). The ideal rotation matrix can be obtained by (39).

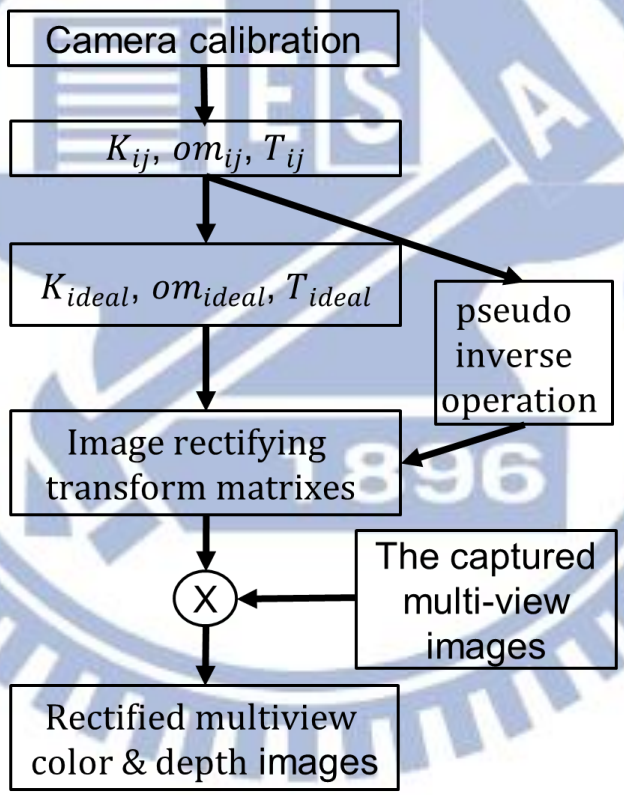


Figure 2-5 Flow chart of multi-camera rectification [16].

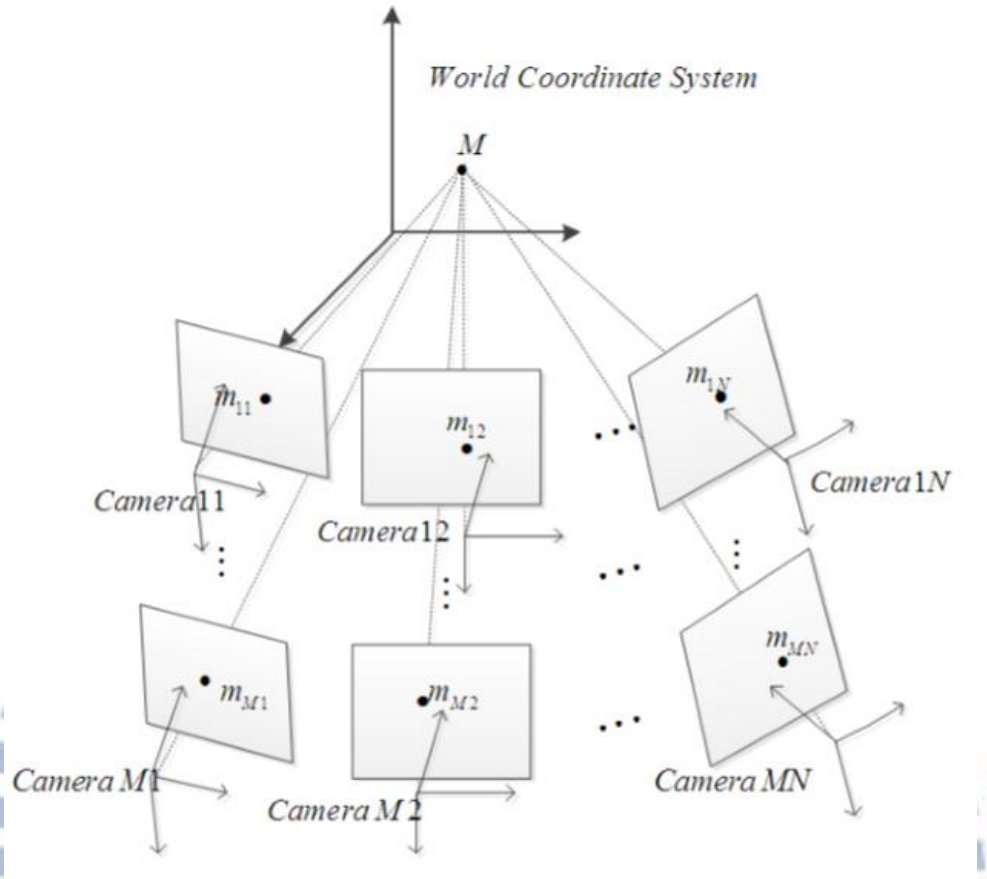


Figure 2-6 Architecture of matrix multi-camera arrangement [16].

$$om_{ij} = \left(\overrightarrow{x_{ij}}, \overrightarrow{y_{ij}}, \overrightarrow{z_{ij}}\right) = \text{Rodrigues transform}(R_{ij}), i = 1, 2, \ldots M, \tag{34}$$

$$\min(\overrightarrow{x_{11}} \ldots \overrightarrow{x_{ij}} \ldots \overrightarrow{x_{MN}}) \leq \vec{x} \in a \leq \max(\overrightarrow{x_{11}} \ldots \overrightarrow{x_{ij}} \ldots \overrightarrow{x_{MN}}), \tag{35}$$

$$\min(\overrightarrow{y_{11}} \ldots \overrightarrow{y_{ij}} \ldots \overrightarrow{y_{MN}}) \leq \vec{y} \in b \leq \max(\overrightarrow{y_{11}} \ldots \overrightarrow{y_{ij}} \ldots \overrightarrow{y_{MN}}), \tag{36}$$

$$\min(\overrightarrow{z_{11}} \ldots \overrightarrow{z_{ij}} \ldots \overrightarrow{z_{MN}}) \leq \vec{z} \in c \leq \max(\overrightarrow{z_{11}} \ldots \overrightarrow{z_{ij}} \ldots \overrightarrow{z_{MN}}), \tag{37}$$

$$om_{ideal} = (\overrightarrow{x_{ideal}}, \overrightarrow{y_{ideal}}, \overrightarrow{z_{ideal}})$$

$$= \arg\min \sum_{i=1}^{M} \sum_{j=1}^{N} \sqrt{(\vec{x} - \overrightarrow{x_{ij}})^2 + (\vec{y} - \overrightarrow{y_{ij}})^2 + (\vec{z} - \overrightarrow{z_{ij}})^2}, \tag{38}$$

$$R_{ideal} = \text{Rodrigues transform}(om_{ideal}). \tag{39}$$

The optical center of each camera can estimated from (40). In order to determine the ideal translation vector, we first shift the optical center of each camera to the position of C_{11} by using (41) and estimate the ideal optical center of C_{11} by using (45). Eq. (41) also takes into account the real world distance between adjacent camera in each row $\vec{l}=l*\overrightarrow{x_{ideal}}$ and column

$\vec{d}=d*\overrightarrow{y_{ideal}}$ (unit: the same as translation vector). After we have the ideal optical center $C_{11,ideal}$, it then shifts the position of each optical center by using (46) and the real distances l and d. The ideal translation vector can be evaluated based on (47).

$$C_{ij} = (x_{ij}, y_{ij}, z_{ij}) = -inv(R_{ij}) * t_{ij} \ , (i = 1,2 \dots M \ , j = 1,2 \dots N \ ), \tag{40}$$

$$C'_{ij} = (x'_{ij}, y'_{ij}, z'_{ij}) = C_{ij} - (i-1)\vec{d} - (j-1)\vec{l} \ , (i = 1,2 \dots M, j = 1,2 \dots N \ ), \tag{41}$$

$$\min(\ x'_{11} \dots x'_{ij} \dots x'_{MN}\ ) \le x \in a' \le \max(\ x'_{11} \dots x'_{ij} \dots x'_{MN}\ ), \tag{42}$$

$$\min(\ y'_{11} \dots y'_{ij} \dots y'_{MN}\ ) \le y \in b' \le \max(\ y'_{11} \dots y'_{ij} \dots y'_{MN}\ ), \tag{43}$$

$$\min(\ z'_{11} \dots z'_{ij} \dots z'_{MN}\ ) \le z \in c' \le \max(\ z'_{11} \dots z'_{ij} \dots z'_{MN}\ ), \tag{44}$$

$$C_{11,ideal} = (\ x_{ideal}\ , y_{ideal}, z_{ideal}\ )$$

$$= \arg\min \sum_{i=1}^{M} \sum_{j=1}^{N} \sqrt{(x - x'_{ij})^2 + (y - y'_{ij})^2 + (z - z'_{ij})^2}, \tag{45}$$

$$C_{ij,ideal} = C_{11,ideal} + (i-1)\vec{d} + (j-1)\vec{l} \ , (i = 1,2 \dots M, j = 1,2 \dots N \ ), \tag{46}$$

$$T_{ij,ideal} = -inv(R_{ideal}) * t_{ij,ideal} \ , (i = 1,2 \dots M \ , j = 1,2 \dots N \ ). \tag{47}$$

So far, we have calculated the ideal rotation matrix R_{ideal} and the ideal translation vector $T_{ij,ideal}$ for each camera. The ideal camera intrinsic parameters K_{ideal} in this thesis are the average of the four camera intrinsic parameters K_{ij}, as shown in (48), because we use identical cameras.

$$K_{new} = (K_{11} + K_{12} + K_{21} + K_{22})/4. \tag{48}$$

Eqs.(49) and (50) are the un-rectified and ideal projection matrix (PPM) of each camera, respectively. We can compute the image rectifying transformation matrix (IRTM) of each camera by (51).

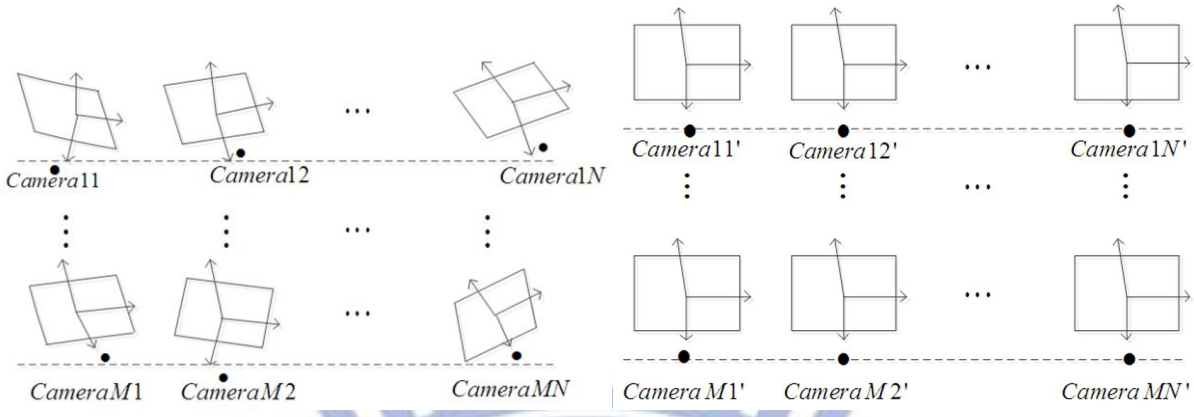
$$PPM_{ij} = K_{ij}[R_{ij}|t_{ij}] \ , (i = 1,2 \dots M \ , j = 1,2 \dots N \ ), \tag{49}$$

$$PPM_{ij,ideal} = K_{ideal}[R_{ideal}|t_{ij,ideal}] \ , (i = 1,2 \dots M \ , j = 1,2 \dots N \ ), \tag{50}$$

$$IRTM_{ij} = PPM_{ij,ideal} * pseduo_inverse(PPM_{ij}) \ , (i = 1,2 \ , j = 1,2 \ ). \tag{51}$$

Applying the image rectifying transformation matrix (IRTM) to the corresponding un-rectified images, we obtain the rectified images, as shown in Figure 2-7(a) and 2-7(b). For

more details about matrix camera rectification, please refer to [16].



(a) (b)

Figure 2-7 (a) Original un-rectified images. **(b)** Rectified images [16].

2.4. Disparity and Depth Estimation

After we complete the image rectification processes, all the epipolar lines between two views are aligned to each other. We can compute the position of the same object point projected into the rectified two views. Theoretically, the position difference is only along the horizontal direction. The difference of horizontal pixel is called disparity, as illustrated in Figure 2-8. If an object is close to the camera, its disparity is large. An far-away object has a samller disparity.

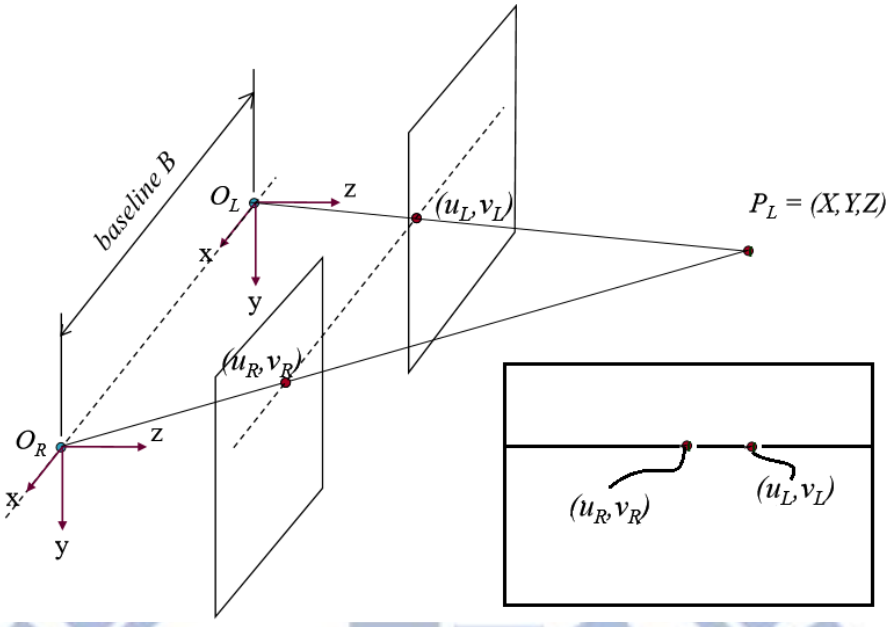


Figure 2-8 Rectified image [17].

From Figure 2-8, based on similar triangular we have

$$(u_L, v_L) = (f\frac{X}{Z}, f\frac{Y}{Z}), \tag{52}$$

$$(u_R, v_R) = \left(f\frac{X-B}{Z}, f\frac{Y}{Z}\right), \tag{53}$$

$$Disparity = d = u_L - u_R = f\frac{B}{Z}. \tag{54}$$

The depth value can then be converted from the corresponding disparity by (55).

$$Z = f\frac{B}{d}, \tag{55}$$

where f is focal length, B is the baseline distance between two cameras, d is disparity, and Z is depth value. The focal length is obtained from the camera intrinsic parameter set. The baseline is the translation between the two cameras; it is derived from the camera extrinsic parameter.

The depth information is essential in many applications in computer vision, 3D modeling, virtual reality, and virtual view synthesis. Once we finish the image rectification, we can estimate the disparity value of each pixel. Then, we can transform the disparity map into the depth map by (55).

2.5. Color Calibration

Color calibration [18-21] corrects the illumination difference between two images. Reference [21] proposed a depth based histogram matching algorithm which can reduce the brightness difference of two images. A simple method of color calibration is histogram matching [18] based on histogram equalization. First, we compute the probability density functions (PDF) and cumulative density functions (CDF) of both the source image color pixels and target image color pixels. The CDF is used as the transfer functions T_1 and T_2 for histogram equalization. We construct $T_2^{-1}(T_1(r))$ as the transfer function to adjust the colors of source and target images, as illustrated in Figure 2-9.

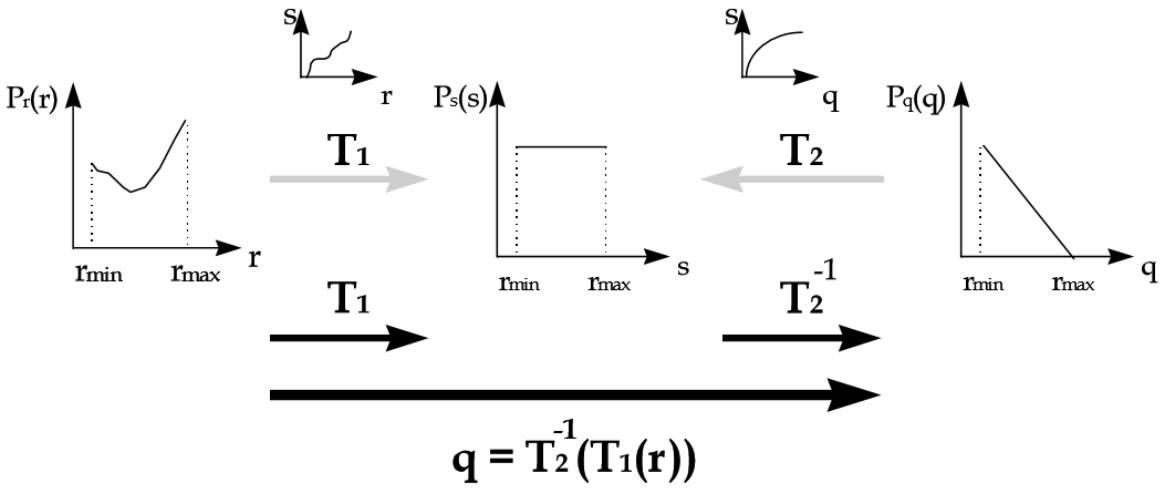


Figure 2-9 Histogram matching.

3. View Synthesis Basics

In this chapter, we will introduce the basic concepts of the known forward depth warping, backward depth warping, texture warping, virtual view merging, and hole filling. We also discuss the features and problems of the active RGB-D camera used in our experiments.

3.1. 3D Warping

3D warping uses the color and depth information to warp a view into another view point by the camera parameters. For virtual view synthesis, we can warp the reference color and depth to the virtual view point, which we want to synthesize. Sections 3.1.1, 3.1.2, and 3.1.3 describe three difference warping methods.

3.1.1. Forward Depth Warping (FDW)

Figure 3-1 shows the flow chart of the virtual view synthesis procesdure. The virtual view image will be synthesized by warping the two reference views to the virtual view, then we merge these two views and fill the holes to produce the final synthesized virtual view image. We derive the forward depth warping [22] formula in below. Eqs. (56) and (57) represent the perspective projection between the 3D world and the two image planes, as illustrated by Figure 3-2.

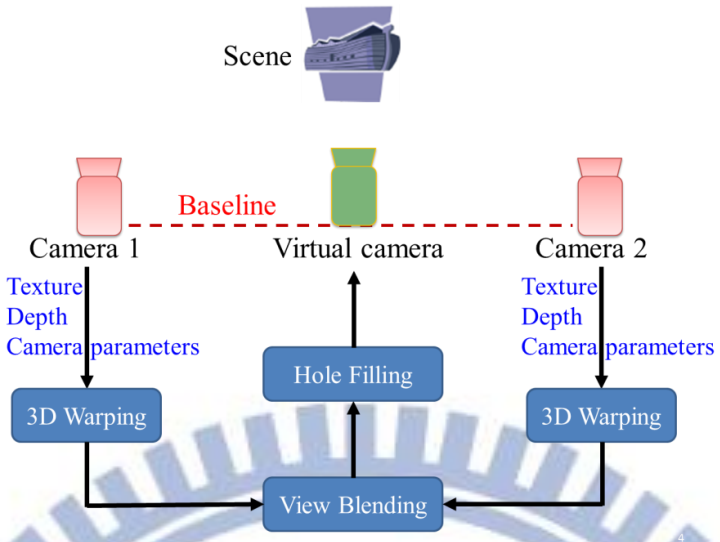


Figure 3-1 Flow chart of view synthesis.

$$z_r p_r = K_r(R_r \mathrm{P} + t_r) \tag{56}$$

$$z_v p_v = K_v(R_v \mathrm{P} + t_v) \tag{57}$$

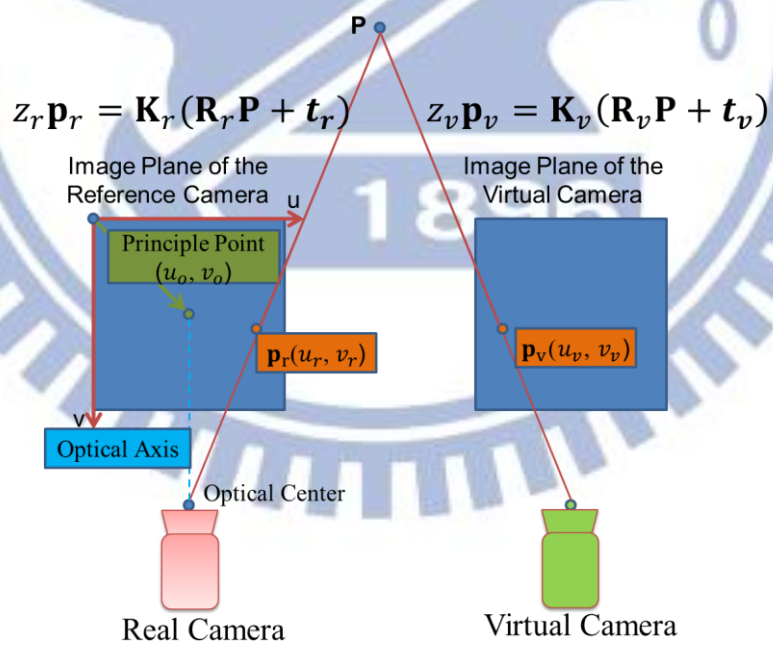


Figure 3-2 3D projection between reference view and virtual view.

Note that P is a point in the 3D world space. And p_r and p_v are the projected points in the reference image plane and virtual image plane, respectively. z_r and z_v indicate the depth

values of the reference image and the virtual image projected from the 3D world point *P*. The *R* and *t* are the rotation matrix and translation vector, which convert the world coordinate system to the camera coordinate system, as said in Section 2.1. *K* represents camera intrinsic parameters, which maps the camera coordinates to the image coordinate. Eq. (56) can be rewritten into (58), which projects the reference image into the 3D world space,

$$\mathrm{P} = R_r^{-1}(z_r K_r^{-1} p_r - t_r) \ . \tag{58}$$

We can obtain the general FDW equation by substituting (58) into (57) for projecting the 3D world space into the virtual image coordinates.

$$p_v = \frac{1}{z_v} K_v (R_v R_r^{-1}(z_r K_r^{-1} p_r - t_r) + t_v) \tag{59}$$

Rearrange (59), we can derive out (60).

$$z_v \begin{bmatrix} u_v \\ v_v \\ 1 \end{bmatrix} = z_r A \begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} + b, \tag{60}$$

where

$$A = (K_v R_v)(K_r R_r)^{-1},\ b = (K_v R_v)(c_r - c_v),\ c_r = -{R_r}^T t_r,\ c_v = -{R_v}^T t_v,$$

$$R_v = R_r * R_x(\alpha) * R_y(\beta) * R_z(\gamma),$$

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}, R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}, R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The rotation matrices $R_x(\alpha)$, $R_y(\beta)$, and $R_z(\gamma)$ are rotated along x, y, and z axis, respectively. The unit of α, β, and γ is degree, and then u_v and v_v may be floating numbers. We will do rounding in this situation. If there are more than one pixel warped to the same position, we will choose the foreground one. According to the z-buffer [22] method, the foreground object blocks the background object.

The numbers on the cell pixel in Figure 3-3 are disparity values. The disparity value means how many pixels will move after 3D warping. There are two problems suffered from FDW, as shown in Figure 3-3. The first one is the small cracks. The disparity value of an inclined object is gradually increasing or decreasing, it will cause small cracks after 3D

warping, as shown in Figure 3-3. Another problem of FDW is background appear in the foreground. This problem is due to the discontinuity of depth map such as the border of an object. These two problems can be reduced by using the backward depth warping.

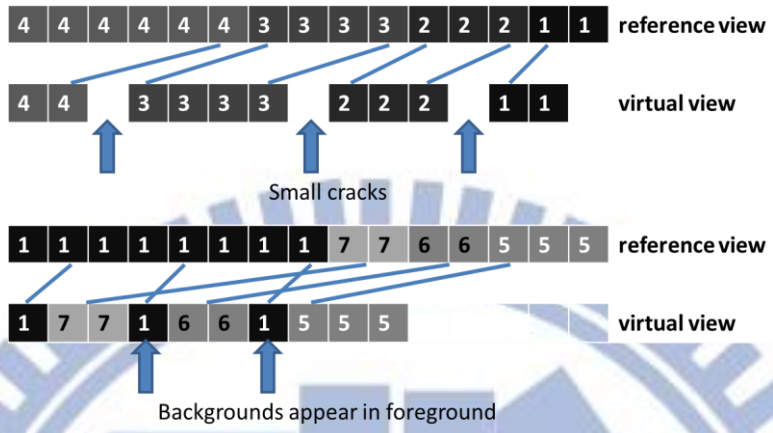


Figure 3-3 Problems of forward depth warping.

3.1.2. Backward Depth Warping (BDW)

The small cracks and background appearing in the foreground are the two problems cannot be solved by FDW. Therefore, some researchers proposed the backward depth warping techniques [23-25]. First, we rearrange (60) so it becomes (61),

$$z'\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = A^{-1}(z_v \begin{bmatrix} u_v \\ v_v \\ 1 \end{bmatrix} - b). \tag{61}$$

In the backward depth warping equation, the red letters in (61) are known value. We need to estimate the floating coordinates in the reference image u', v', and the depth value in the reference depth z', as illustrated in Figure 3-4.

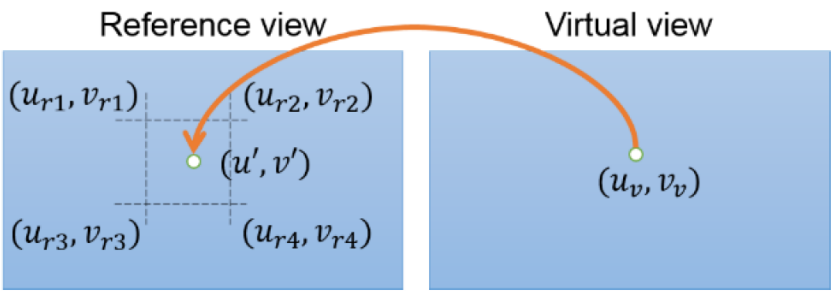


Figure 3-4 Backward depth warping.

It is hard to assure that the calculated depth value is right or not. Thus, [24] finds the best depth value by minimizing the error function (62).

$$E = \min_{\substack{u_r = floor(u')\ or\ ceil(u'),\\ v_r = floor(v')\ or\ ceil(v')}} (u_r - u')^2 + (v_r - v')^2 + (D(u_r, v_r) - D')^2, \quad (62)$$

where D' represents the disparity of z'. To find the best disparity value, we calculate the error function by trying all possible disparity values from 255 to 0. The disparity associated with the first local minimum of E is the best disparity. We use (55) to convert the disparity value to its associated depth value.

3.1.3. Texture Warping

There are two kinds of texture warping. The first method is forward texture warping (FTW), which warps the reference texture to virtual view using the coordinates of FDW. However, FDW suffers from small cracks and background appears in foreground problems, as mentioned in Section 3.1.1. Therefore, [24] proposed a backward texture warping (BTW) by using the bicubic interpolation method. The bicubic convolution kernel is listed in (63).

$$W(x) = \begin{cases} (a+2)|x|^3 - (a+3)|x|^2 + 1 & for\ |x| \le 1 \\ a|x|^3 - 5a|x|^2 + 8a|x| - 4a & for\ 1 < |x| < 2 \\ 0 & otherwise \end{cases}, \quad (63)$$

where a can set to be -0.5 or -0.75. If a = -0.5, the filtering formula become (64).

$$image_{(u',v')}(x) = p(t_x\,, x-1, x, x+1, x+2, y\,, image)$$

$$= \frac{1}{2}\begin{bmatrix}1 & t_x & {t_x}^2 & {t_x}^3\end{bmatrix}\begin{bmatrix}0 & 2 & 0 & 0\\ -1 & 0 & 1 & 0\\ 2 & -5 & 4 & -1\\ -1 & -3 & -1 & 1\end{bmatrix}\begin{bmatrix}image(x-1,y)\\ image(x,y)\\ image(x+1,y)\\ image(x+2,y)\end{bmatrix}, \tag{64}$$

where u' and v' are the floating coordinate obtained from (61), and $0 \le t_x = u' - x \le 1$, as shown in Figure 3-5. For interpolating images, we implement (63) along x-axis and y-axis, as listed in (65) ~ (69).

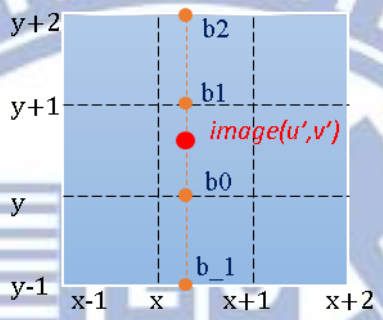


Figure 3-5 Bicubic interpolation.

$$image_{(u',v')}(x,y) = p\big(t_y\,, b_{-1}, b_0, b_1, b_2, image\big), \tag{65}$$

where

$$b_{-1} = p(t_x\,, x-1, x, x+1, x+2, y-1\,, image), \tag{66}$$

$$b_0 = p(t_x\,, x-1, x, x+1, x+2, y\,, image), \tag{67}$$

$$b_1 = p(t_x\,, x-1, x, x+1, x+2, y+1\,, image), \tag{68}$$

and

$$b_2 = p(t_x\,, x-1, x, x+1, x+2, y+2\,, image)\ , \tag{69}$$

where $image_{(u',v')}(x,y)$ is the result of bicubic interpolation.

3.2. Virtual View Merging

At this step, we use the virtual view warped from all the reference views to generate the

synthesized virtual images. The reason why we need to do view merging is some pixels in virtual view cannot be seen in one of the reference views, but it can be seen in the other reference view, thus we can use another reference view to synthesize it.

There are several papers discussing the view merging method [22, 25-28]. The simplest method is average merging; that is the synthesized virtual color image is the average of all virtual views. However, it is sensitive to noise. A better method is linear weight merging [25-27], which considers the camera baseline distance.

Average merging method and linear weight merging method will suffer from the blurring effect if the warped views have misalignment between them. Therefore, [28] proposed to choose one warped view as the dominant view for synthesizing virtual view. Then, the remaining holes can be refined by the other warped views. This method can reduce the blurring effect in the synthesized virtual view. The dominant merging method proposed in [28] still has poor synthesized image quality on the object edges when the virtual views are misaligned. The z-buffer [22] method can be used when there exist two different depth values at the same position.

In order to achieve better view merging quality, [29] proposed an adaptive blending technique. The pseudo codes are listed in **Algorithm 1**.

Algorithm 1: Adaptive blending algorithm.

---

if $|D_L(u,v) - D_R\ (u,v)| < th1\ and\ |I_L\ (u,v) - I_R(u,v)|^2 < th2$

$I_v(u,v) = (1-\alpha) * I_L(\ (u,v) + \alpha * I_R(u,v)$

$D_v(u,v) = (1-\alpha) * D_L(u,v) + \alpha * D_r(u,v)$

else

if $D_L\ (u,v)\ is\ more\ reliable$

$I_v(u,v) = I_L(u,v);\ \ D_v(u,v) = D_L(u,v)$

else if $D_R(u,v)\ is\ more\ reliable$

$I_v(u,v) = I_R(u,v);\ \ D_v(u,v) = D_R(u,v)$

```
        else
            if  D_L  (u,v)>D_R(u,v)
                I_v(u,v)= I_L(u,v);  D_v(u,v)= D_L(u,v)
            else
                I_v(u,v)= I_R(u,v);  D_v(u,v)= D_R(u,v)
        end
    end
end
```

The subscript *L*, *R*, and *v* in Algorithm 1 represent left reference view, right reference view, and virtual view, individually. *I* and *D* denote the color value and disparity value, respectively. **Algorithm 1** will first check whether the color and depth value of the two warped views are similar or not. If their difference less than a specified threshold *th1* and *th2*, then execute the linear weight merging, as described in Section 3.2. If the color or depth value of the two warped views are not close, then it uses the reliable one to synthesize that pixel. If the depth pixels of both warped views are not reliable, it uses the z-buffer [22] method to choose the one with larger disparity value.

3.3. Hole Filling

There are some holes in virtual view that cannot be found in the reference views. Therefore, after view merging, those holes still exist. There are a few papers use the inpainting techniques [30-32] or directional image inpainting [44] to fill the holes. Also, the directional hole filling methods [27, 33] are also proposed to deal with holes. Furthermore, some papers use the inter-frame information [35-37] and intra-frame information [36] of a video sequence to fill the holes. Recently, deep learning becomes more and more popular used

in many areas, a deep learning based hole filling scheme [34] was just proposed.

3.4. Depth Camera

For a more accurate way to produce depth maps, we use the depth camera to capture the depth maps. Although the captured depth map still has some occlusion regions, generally, its quality is better than the matching-based depth estimation method. The available commercial RGB-D camera such as Microsoft Kinect v1 [38] and Kinect v2 [40] can capture color and depth image/video. SR4000 [39] is another commercial depth camera using the time of flight depth sensor. We can roughly classify the depth camera into two types, as listed in Table 3-1. The first type is structured light camera. One example of the famous structured light cameras is Kinect v1, which is developed for Microsoft Xbox 360, as shown in Figure 3-6(a). The color camera is in 640*480 resolutions and the depth camera is in 320*240 resolution.

Table 3-1 Comparison of commercial depth cameras.

	Kinect v1 [38]	SR4000 [39]	Kinect v2 [40]
Color Resolution	640*480	N/A	1920*1080
Depth Resolution	320*240	176*144	512*424
Frame Rate	30 fps	10 ~ 30 fps	30 fps
Sensor Type	Structured Light	Time of Flight	Time of Flight

(a)

(b)

Figure 3-6 (a) Kinect™ for Xbox 360® [38]. **(b)** Kinect™ for Xbox One® [40].

The second type of depth camera is Time-of-Flight (ToF) camera [39]. Kinect v2 is one of the well-known ToF cameras, as shown in Figure 3-6(b). The color camera of Kinect v2 is in 1920*1080 resolution @30 fps and the depth camera is in 512*424 resolution @30 fps (16 bits). Kinect v2 can detect the depth range from 0.5m to 8m.

Figure 3-7 shows the inside of the Kinect v2 device. The positions of IR emitter and IR receiver are different. Hence, some background regions are blocked by the foreground object. The depth of the blocked background region called occlusion regions, as shown in Figure 3-8 red region.

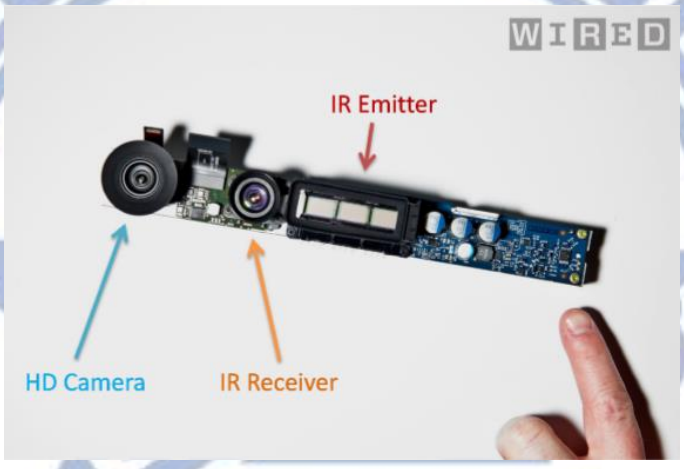


Figure 3-7 Kinect™ for Xbox One® teardown [41].

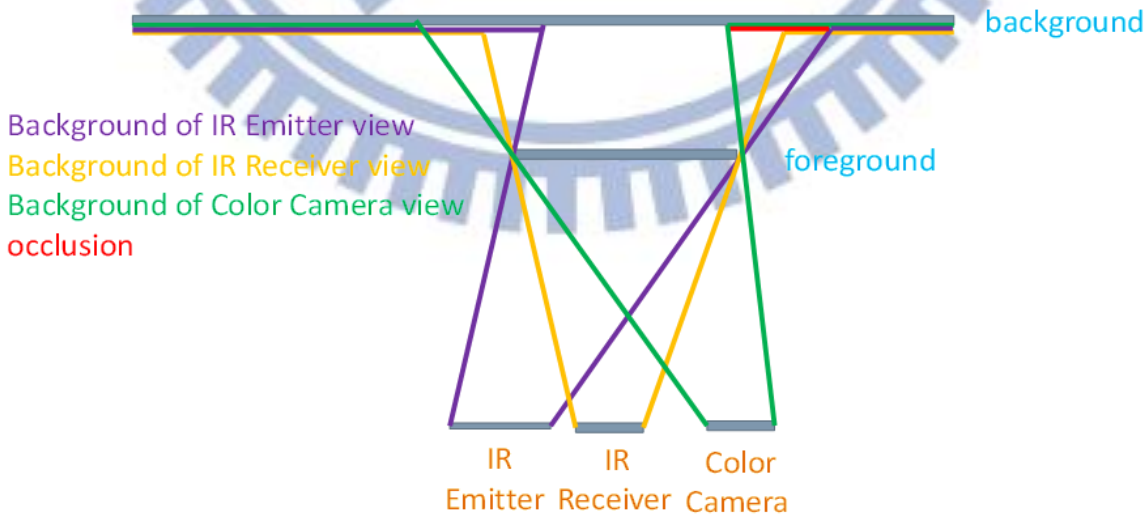


Figure 3-8 Background-to-foreground occlusion problems [42].

A few papers have proposed the depth map refinement algorithm to remove the holes and occlusion regions of the depth map. The occlusion regions and holes often indicate that they are unknown depth values at those pixels. For view synthesis or 3D modeling application, we cannot use a pixel with unknown depth value. Hence, depth map refinement is an important process for active depth camera. We will introduce the depth map refinement algorithm adopted by this thesis in Section 4.

Microsoft provides a SDK for Kinect v2. It provides some application software of depth map. Because the image sizes of depth and color image captured from Kinect v2 are different, we use the function, MapColorFrameToDepthSpace, to convert and map the depth map to match the color image.

There are three problems occurring in the captured a depth map by Kinect v2. The first problem is the resolution of depth map is much lower than that of the color image due to the hardware limitations. This may cause incorrect object edges in the depth map. The second problem is the Kinect depth value on a reflection surface is often incorrect. The third problem is the temporal and spatial mismatch problem between the color and depth cameras. The above problems can be solved by the depth map refinement algorithm [43]. We will discuss it in Section 4.

4. Depth Map Refinement for Kinect V2

The quality and resolution of depth map captured by Kinect v2 are noisy and poor. The resolution of depth map is 512*424. The noisy depth map will affect the synthesized image quality in view synthesis applications. Therefore, we adopt the depth map enhancement algorithm proposed by [43] to refine the depth map captured by Kinect v2.

Figure 4-1 Flow chart of the overall depth enhancement algorithm for Kinect v2 [43].

The overall flow chart of the algorithm proposed by [43] is shown in Figure 4-1. First, it inputs all the color and depth frames of a sequence. Then, it constructs the background and solves the asynchronization problem between the color and depth frames. After synchronizing the color and depth frame, it detects and fills the occlusion regions, unreliable depth pixels, holes, and reflection regions. Finally, bilateral filter is applied to the depth map for smoothing purpose. Each block in Figure 4-1 will be described briefly in the following sub-sections.

4.1. Background Construction

Background pixels typically have the minimum depth value or maximum disparity. Hence, the background depth map can be produced by applying a time-wise minimum filter to the depth frames of a sequence. However, some depth values may be wrong due to the

reflection surface or other light interference. In order to remove the wrong depth values, reference [43] keeps two possible background depth candidates of each pixel, and update one of the possible depth candidates when an input frame is accessed. The two possible background candidates make the final background more robust against the depth outlier or depth noise. The flow chart of background construction process is shown in Figure 4-2.

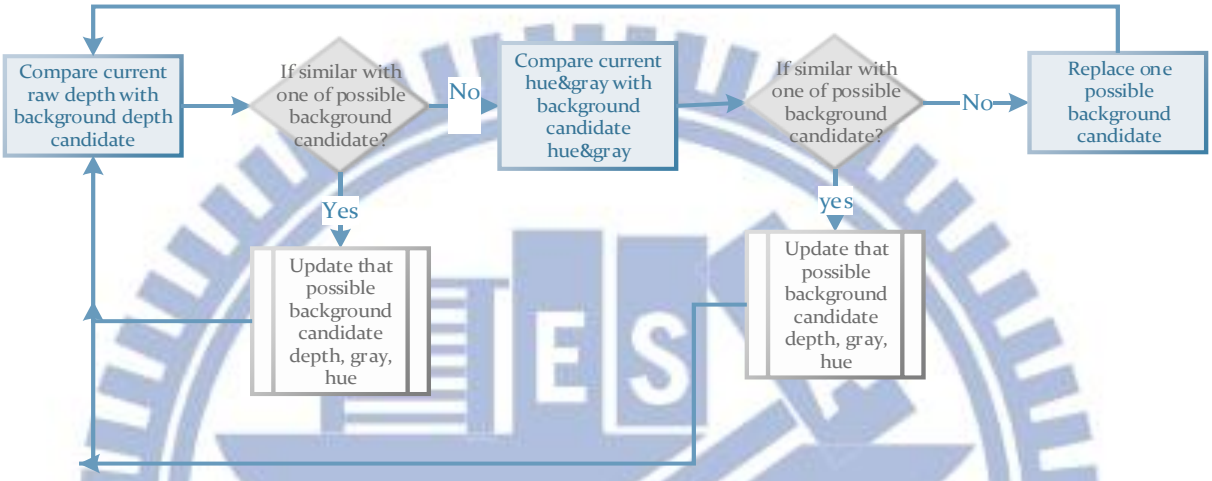


Figure 4-2 Flow chart of background construction [43].

The first step of background construction is applying the time-wise minimum filter to the depth maps of a sequence. And then it executes every steps in the flow chart as shown in Figure 4-2. If the difference of the current raw depth value and the background candidates is less than 5, update the depth, hue, and gray value of the background candidates. If the current depth is not similar to the background candidates, then check the similarity of the hue and gray values between the current pixel and the background candidates. If the hue and gray are similar to one of the background candidates, then update the depth candidate by (70) and (71).

$$Updated_bkgd_depth = \frac{frame_n - bkgd_{weight}}{frame_n} \times bkgd_d + \frac{bkgd_{weight}}{frame_n} \times current_d, \quad (70)$$

with

$$bkgd_{weight} = \frac{255 - current_d}{70}, \quad (71)$$

where *Updated_bkgd_depth* is the updated depth value of the background candidate, $frame_n$ is

the number of frames that have been processed in this background candidate, *bkgd_d* is the depth value of the background candidate, *current_d* is the depth value of the current pixel, and $bkgd_{weight}$ is the weight of the current depth pixel. Finally, it chooses the depth candidate that has the larger depth value is chosen to be the final background depth value at each pixel.

4.2. Alignment Between Color Image and Depth Map

The captured color images and depth maps may have a bit of misalignment in the moving object. This misalignment problem is due to Kinect v2 capturing the color image first and then the depth map. Although the capturing timing of color image and depth map are very close, it still has misalignment in the moving objects. This misalignment will degrade the quality of the processed depth map; particularly, it will make the view synthesized pictures blurred.

Figure 4-3 shows the overall alignment processes. First, it uses the current depth map and the last depth map to find the moving object. But the depth value of a moving object is often similar; hence, it uses the current color image and the previous color image to make the moving object detection more reliable. Second, it uses a three-step search algorithm [45] to find the motion vector based on color image and then use the motion vector to back-warp and generate the stitched depth map based on the previous refined depth frame. Third, it extracts the initial moving foreground of the stitched depth frame by comparing the depth maps of the current frame and the previous frame. Afterward, it removes the wrongly detected moving foreground. So far we have extracted the moving foreground of the current depth frame and the stitched depth frame. If the overlap ratio of the moving foreground of the current depth frame and the stitched depth frame estimated from (72) is larger than a threshold, we can assume the detected moving foreground is correct, and then we align these two objects.

$$\frac{Number\ of\ overlapping\ pixel}{Number\ of\ retrieved\ pixel(raw\ depth)} > threshold. \tag{72}$$

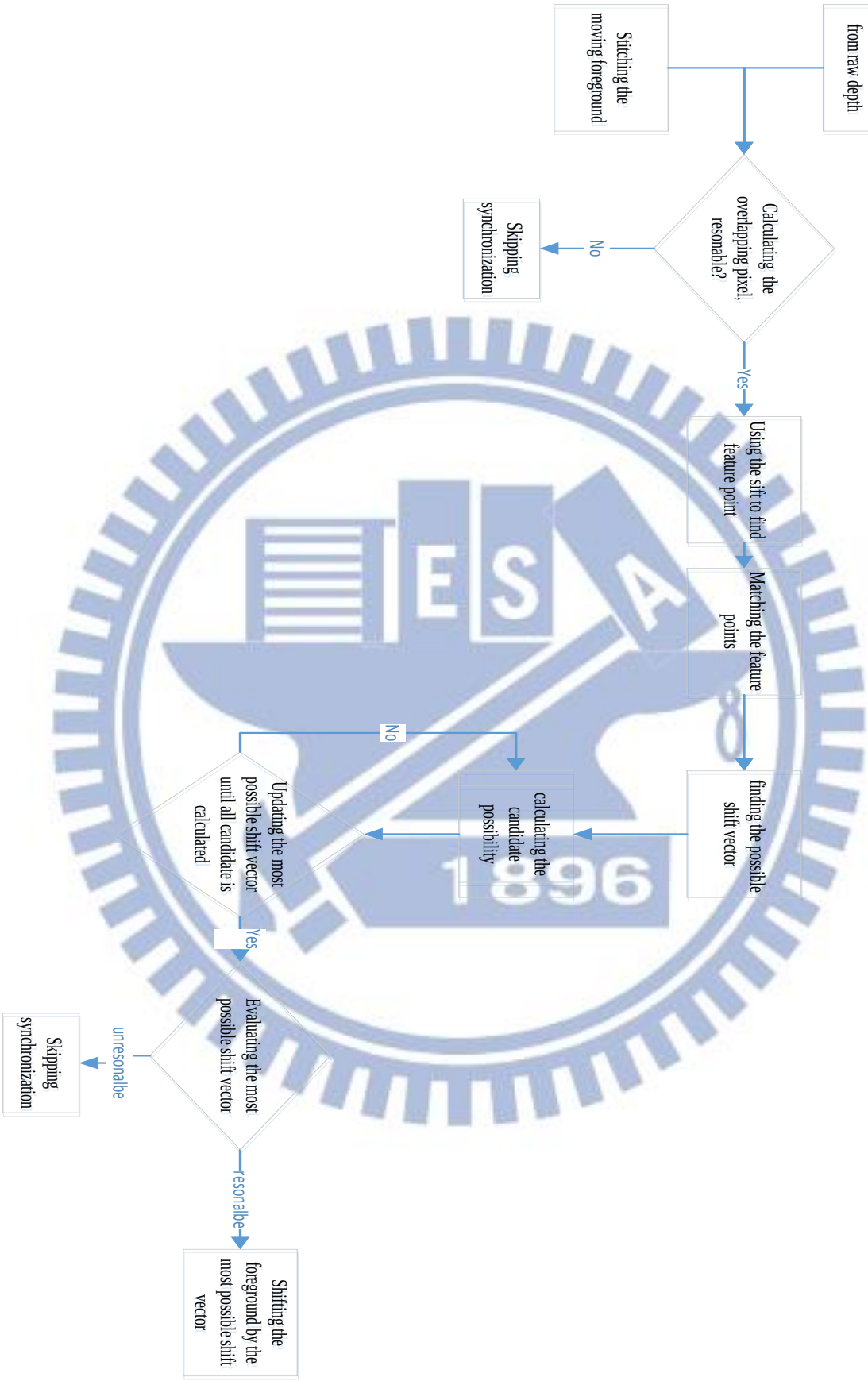


Figure 4-3 Flow chart of alignment between color image and depth map [43].

For aligning the moving foreground of the current depth map with the stitched depth map, we first find the SIFT features of both frames. To find the possible matched feature points, we perform inner product on the 128-D vectors of two feature points and then calculate the shift vector in the horizontal and vertical directions. We use the estimated shift vector to shift the current depth map to the position of the stitched depth map, and compute the overlapped pixel number. The final shift vector is associated with the largest overlapped pixel number. At the end, we use this shift vector to shift the moving regions of the current depth map.

4.3. Occlusion Region Detection and Filling

Occlusion region appears when mapping the images of two different position cameras to the same view point. As we known, the color camera and the depth camera of Kinect v2 are in different positions. Thus, when we map the depth and color image to the same view point, the occlusion region appears on the warped depth map. Reference [46] proposed an algorithm to estimate and fill the occlusion regions.

Figure 4-4 illustrates the occlusion region. Points D and E are the focal point of color camera and depth sensor. The region between A and B is a disoccluded region whose width is A_width. d1 and d2 are the depth of foreground and background, individually. Dis_camera is the distance between the focal points of the depth sensor and the color camera. Focal_length represents the focal length of color camera. Occlusion_width denotes the width of disocclusion region on color image. Dis_camera and Focal_length are fixed values for Kinect v2 device. Eq. (73) shows how to estimate the occlusion width based on the depth value of foreground and background. Because the occlusion regions belong to background, we fill up the occlusion region by using the constructed background depth map. For more details of detecting and filling occlusion regions please refer to [46].

$$Dis_camera \times Focal_length = \frac{Occlusion_width \times \mathrm{d2} \times \mathrm{d1}}{d2 - d1}. \tag{73}$$

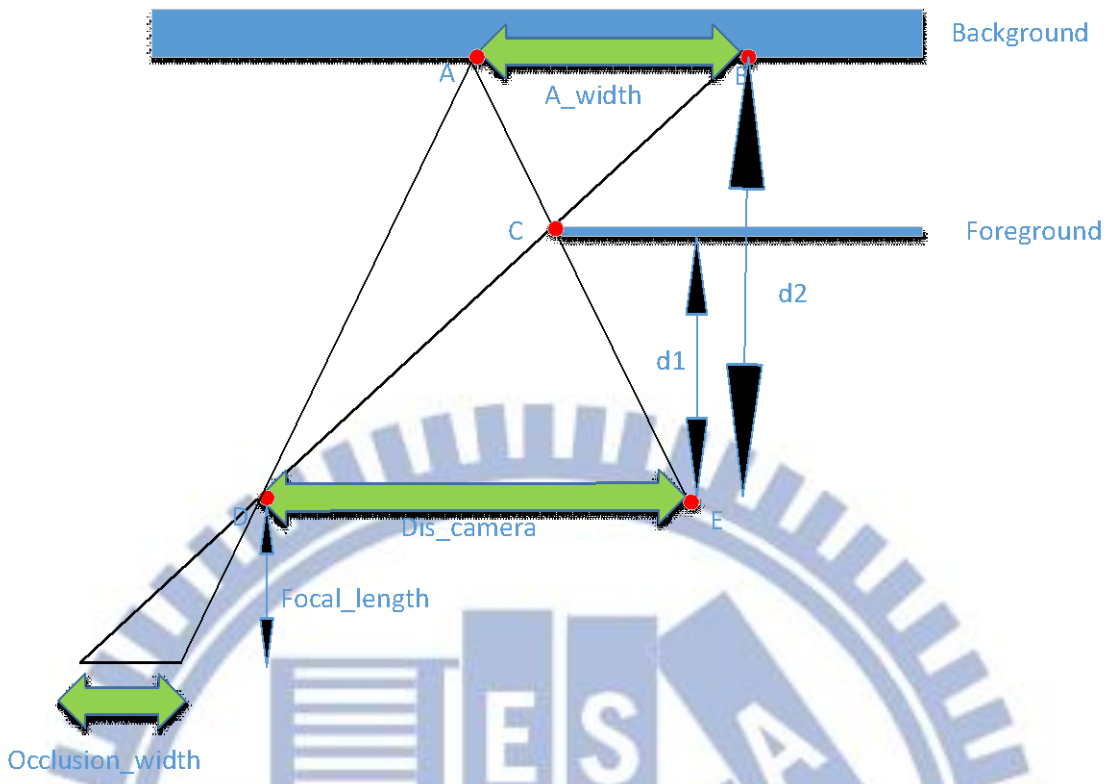


Figure 4-4 Illustration of occlusion region [43].

4.4. Detection and Refinement for Unreliable Depth Pixels and Reflection Regions

Unreliable depth pixels such as flicker depth noise or pixels near object boundaries should be detected and refined. For detecting and filling the unreliable depth pixels, we use the gray and hue of the color image to find the dominant color of each depth group in a block. And then we identify the unreliable depth pixels in that group based on the dominant color of each depth group.

Figure 4-5 shows the flow chart of detecting unreliable depth pixels. First, it partitions the depth map into 50*50 blocks and checks whether any of gradient in the block is larger than a threshold. If so, it executes the detection process. Second, divide and compute the average value of the hue and gray of each depth groups. Third, determine the reliable depth pixels of each group.

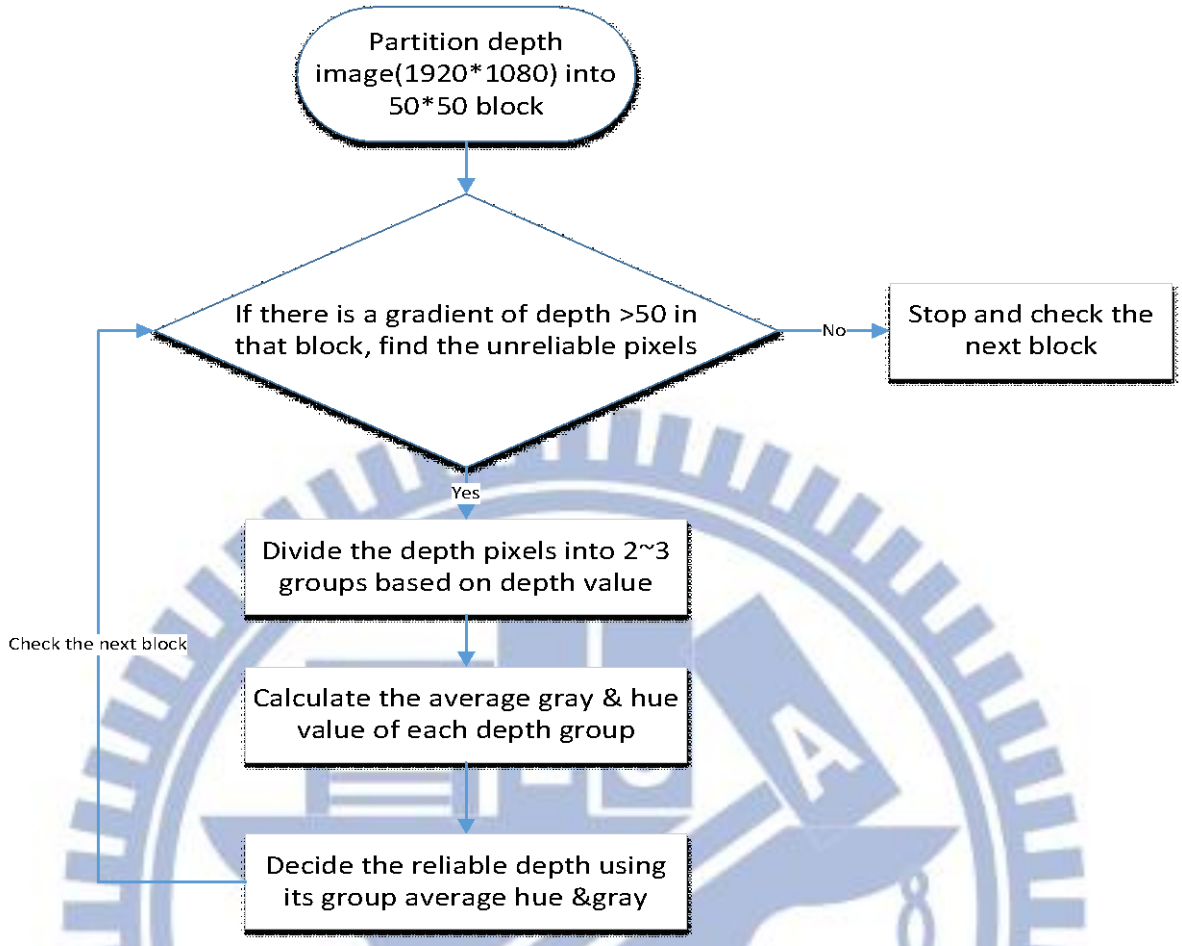


Figure 4-5 Flow chart of detecting unreliable depth pixels [43].

For detecting the unreliable depth pixels of foreground depth groups, if hue or gray intensity is similar to the average value of that block, then this depth pixel is regarded as an unreliable depth pixel. Furthermore, if hue or gray intensity is similar to the background, then it is considered to be unreliable depth pixel.

For detecting the unreliable depth pixels of background depth groups, if hue or gray intensity is similar to the average hue or gray intensity of the foreground depth group, then it is marked as an unreliable depth pixel. Moreover, if the hue or gray intensity is similar to the average hue or gray intensity of the third depth group, but either hue or gray is not similar to the average hue or gray intensity of the background depth group, then it is an unreliable depth pixel also.

For detecting the unreliable depth pixels of the third depth group, if the hue or gray

intensity is not similar to the average value of hue or gray intensity of the third group, then it is tagged as an unreliable depth pixel.

Because Kinect v2 use the ToF depth sensor, some reflection surfaces, such as mirror, reflect the IR light emitted from the ToF depth emitter. Hence, the receiver capture the wrong light and then produce defected depth maps. Reflection region often induces sparse depth holes. Hence, [43] proposed methods to detect and fix the reflection region.

For detecting the reflection region, we first identify the sparse depth holes, and then use those sparse holes as seeds. If any depth gradient is larger than a threshold in the window of size 50*50 centered at the seed, then the sparse depth expands. To decide the reflection pixels, we check whether the gray intensity of each pixel is similar to the average gray intensity of the window or not. If so, then label that pixel as a reflection pixel.

The refinement flow chart for unreliable depth pixels and holes are shown in Figure 4-6. First, it sets a square window to include 300 non-hole depth pixels. Second, calculate the weight by (74) ~ (78). Third, remove the weight less than 70% of the average weight in that window. Fourth, if the variance of depth is less than the threshold, refine the depth by the majority surviving depth candidates.

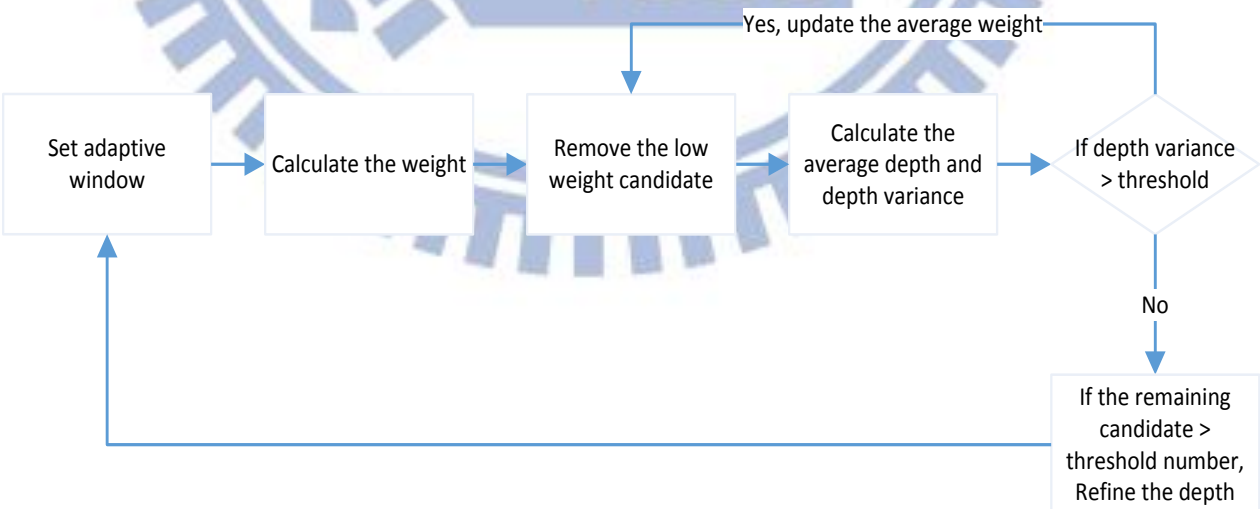


Figure 4-6 Flow chart of processing the unreliable depth pixels and holes [43].

$$\text{Weight_location} = e^{-\frac{x_dis^2+y_dis^2}{100}}, \tag{74}$$

$$\text{Weight_color} = e^{-\left(\frac{diff_gray^2}{50}+\frac{diff_hue^2}{50}+\frac{diff_sat^2}{400}\right)}, \tag{75}$$

$$\text{Weight_reliable} = \begin{cases} 0.1\,, & depth \in unreliable \\ 1\,, & otherwise \end{cases}, \tag{76}$$

$$\text{Weight_reflection} = \begin{cases} 0.3\,, & depth \in reflector \\ 1\,, & otherwise \end{cases}, \tag{77}$$

$$\text{Weight} = \text{Weight_location} \times \text{Weight_color} \times \text{Weight_reliable} \times \text{Weight_reflection}. \tag{78}$$

After processing the unreliable depth pixels and holes, we deal with the remaining reflection regions. The flow chart of refining the reflection regions is shown in Figure 4-7. If the gray intensity of a reflection pixel is similar to the gray intensity of background, then we use the background depth to refine the reflection pixel. If the gray intensity is not close to background, but similar to the gray intensity of the previous frame, then the reflection pixel will filled by the refined depth value of the previous frame. If the above two cases all failed, then we run the same depth refinement algorithm as shown in Figure 4-6 with different weights obtained from (79) ~ (81). After finishing all the above refinement processes, we apply bilateral depth filtering to smooth the depth map. For more details about depth refinement based on RGB-D images, please refer to [43].

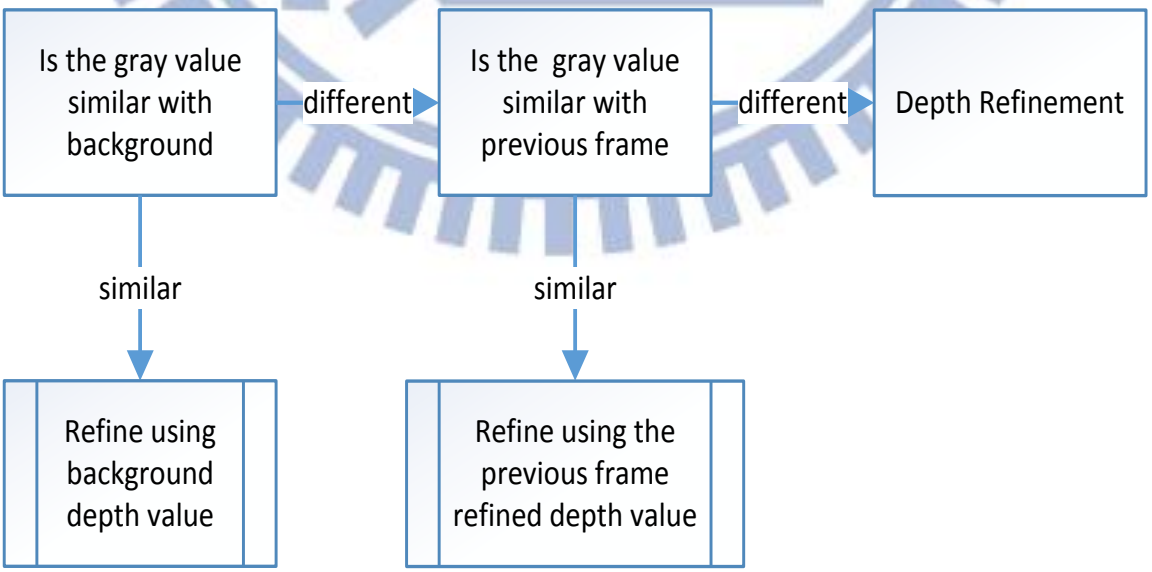


Figure 4-7 Flow chart of refining the reflection regions [43].

$$\text{Weight_reliable} = \begin{cases} 0.8\,, & depth \in unreliable \\ 1\,, & otherwise \end{cases} \tag{79}$$

$$\text{Weight_background} = \begin{cases} 0.01\,, & depth \in background \\ 1\,, & otherwise \end{cases} \tag{80}$$

$$\text{Weight} = \text{Weight_location} \times \text{Weight_color} \times \text{Weight_reliable} \times \text{Weight_background} \tag{81}$$

5. 3D Warping Before Wide Angle View Merging

In the section, we introduce the 3D warping process before our proposed wide angle view merging algorithm. The flow chart of the entire 3D warping [29] is shown in Figure 5-1. The procedure includes forward depth warping (FDW), disocclusion hole detection, fast backward depth warping, background construction, and fast backward texture warping.

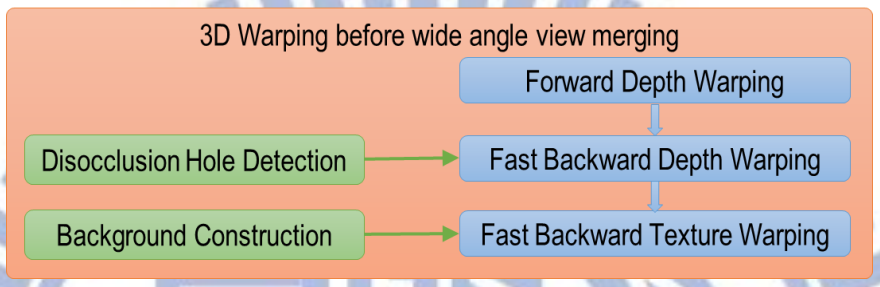


Figure 5-1 Flow chart of 3D warping before wide angle view merging [29].

First, we use FDW to warp the reference color image and depth map to the virtual view point as mentioned in Section 3.1.1. As we know, FDW often produces small cracks, which can be removed when it is replaced by the backward depth warping. For reducing complexity, we do not use the (fast) backward depth warping to remove disocclusion holes. After finishing the fast backward depth warping, we will do an opening-by-reconstruction operation to refine the virtual depth map. Finally, we construct the background [24] and produce the virtual views warped from every reference view. We will describe more details in the following sub-sections. For more details about this 3D warping process, please refer to [29].

5.1. Disocclusion Hole Detection

A depth pixel in virtual view cannot be seen in the reference views are called disocclusion hole, which cannot be solved by the backward depth warping. Hence, to reduce the calculations for backward depth warping, [29] proposed a method to detect the

disocclusion holes and it skips the disocclusion holes in the proposed fast backward depth warping. There are three steps for detecting disocclusion holes. First, mark a 3-pixel-width area along the depth object edges in the reference view. Second, warp the marked area to the virtual view. Third, if the number of the marked pixels virtual view near a hole is larger than a threshold, then specify the hole as a disocclusion hole.

5.2. Fast Backward Depth Warping

As we described in Section 3.1.2, the BDW algorithm needs to minimize the error function (62) by trying the disparity value from 255 to 0. Hence the computational complexity of BDW is 256 times of that of FDW. In order to reduce the computing time of BDW, [29] proposed a fast backward depth warping algorithm. It consists of two techniques. The first technique is to reduce the number of pixels needed to do BDW. There are two reasons for doing BDW. The first reason is that a hole located on the virtual view and it is not a disocclusion hole. The second reason is that the depth value changes after filtered by a median filter. The second technique to speed up the operating time of BDW is to reduce the computation complexity (of each pixel) needed in doing BDW. The idea is to choose the maximum disparity value of a pixel's neighbor in a 5*5 window centering at that pixel as an initial value. Then, we minimize the error function (62) by plugging testing values smaller than the initial value. By using the second technique, we do not need to start the minimizing process from the fixed peak disparity value. This method can efficiently saves a lot of computational complexity.

5.3. Fast Backward Texture Warping

The BTW algorithm stated in Section 3.1.3 needs the bicubic interpolation operation. The bicubic interpolation has a large amount of computational complexity. Hence, in order to

save the computing time of BTW, reference [29] reduces the number of pixels needed to execute the bicubic interpolation. It consists of three step. The first step marks the pixels, whose reference color is the same as the associated background color. The second step marks the pixels, whose virtual depth value is the same as the associated background virtual depth. The third step applies the BTW and bicubic interpolation to the pixels which are not marked in the first step or the second step. The remaining pixels will be filled by the background virtual color.

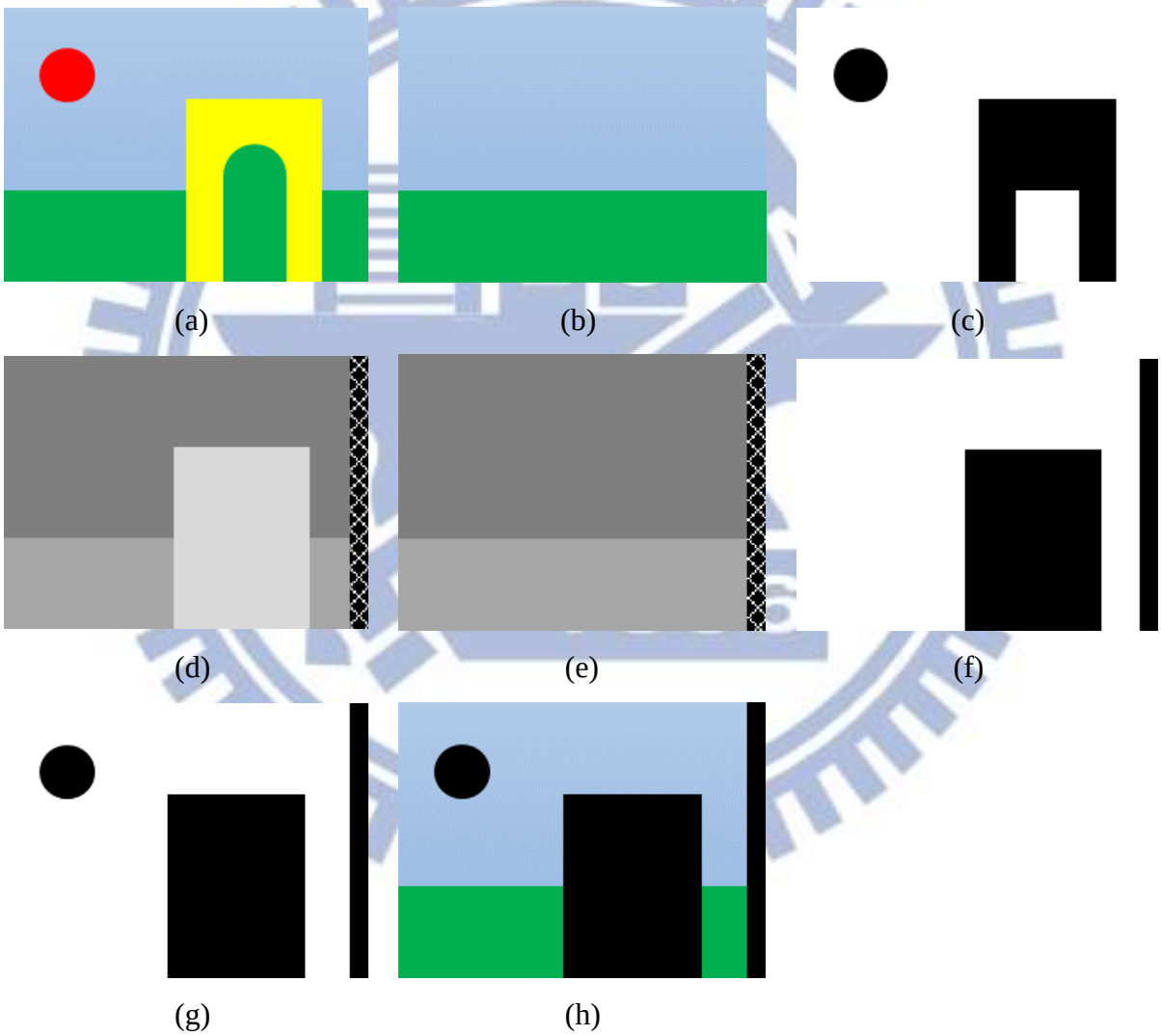


Figure 5-2 (a) reference color image **(b)** reference background color image **(c)** the pixels of same values in the reference color and reference background color image **(d)** virtual depth map **(e)** virtual background depth map **(f)** the pixels of the same values in virtual depth and virtual background depth map **(g)** pixels in virtual view whose color is the same as virtual background and filled by virtual background **(h)** the remaining black holes are filled by the bicubic interpolation [29].

6. Proposed Wide Angle View Synthesis Algorithm

In this thesis, we propose a multi-view blending algorithm to merge the warped virtual views and to produce the final synthesized image/video of the virtual view. The proposed multi-view blending algorithm is robust against to the misalignment between virtual views. It also filters out the wrong pixels due to misalignment between color image and depth map. The entire view synthesis system integrates our multi-view blending algorithm with the depth refinement algorithm and 3D warping algorithm, which were described in Section 4 and Section 5, respectively.

Figure 6-1 shows the flowchart of our wide angle view synthesis system. We use four Kinect v2 devices arranging in a two-by-two pattern in our experiments. Due to the hardware limitations, each Kinect v2 sensor is connect to a computer. We first execute the camera calibration [12] and multi-camera image rectification [16] to obtain the rectified camera parameters and the rectified color images and depth maps, as described in Section 2.2 and Section 2.3.2, respectively. After that we run the depth refinement procedures [43] and 3D warping [29] as stated in Section 4 and 5. Thus, we have the four warped virtual color images and depth maps. Then we can perform our multi-view blending algorithm to produce the final synthesized image/video of the virtual view.

For the video case, the above synthesis method suffers from the synchronization problem between different Kinect v2 devices. For a moving object, if the four Kinect v2 devices cannot record at the same time, the captured images of four cameras are not exactly at the same location because some cameras capture early while some cameras capture later. Therefore, we need to synchronize the four Kinect v2 devices recording at exactly the same time. Section 6.1 describes how we deal with the synchronization problem. Section 6.2 introduces the proposed multi-view blending algorithm in this thesis.

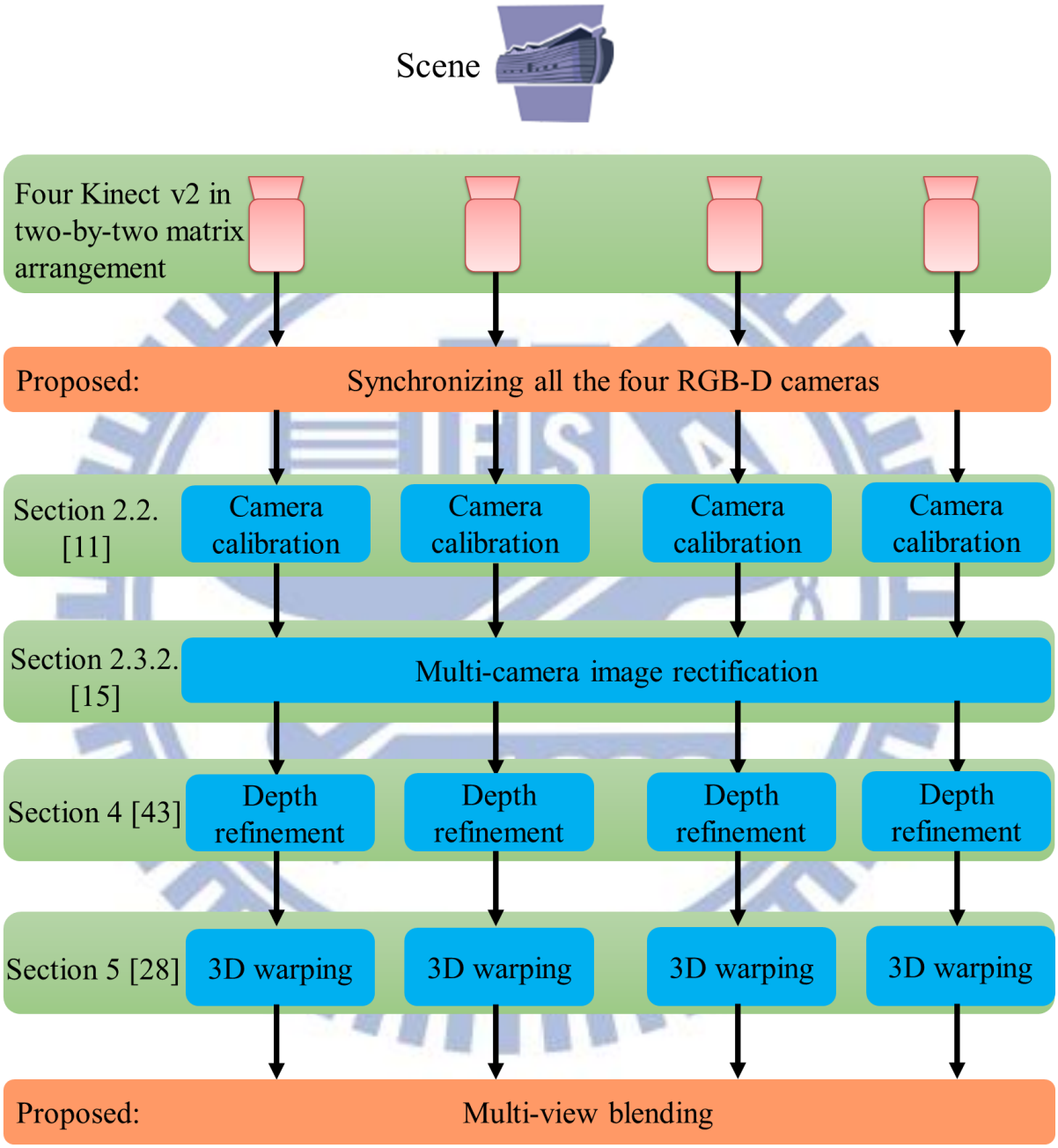


Figure 6-1 Flowchart of the proposed wide angle view synthesis system.

6.1. Synchronizing all the four RGB-D cameras

The asynchronization problem between four Kinect v2 cameras leads to that the four recorded views are not captured exactly the same time. This problem is particularly serious when there are moving objects. If we do not solve this asynchronization problem, the warped virtual views have an obvious shift near the boundaries of the moving objects. Figure 6-2 shows the four color images captured from four Kinect v2 cameras with asynchronization problem.

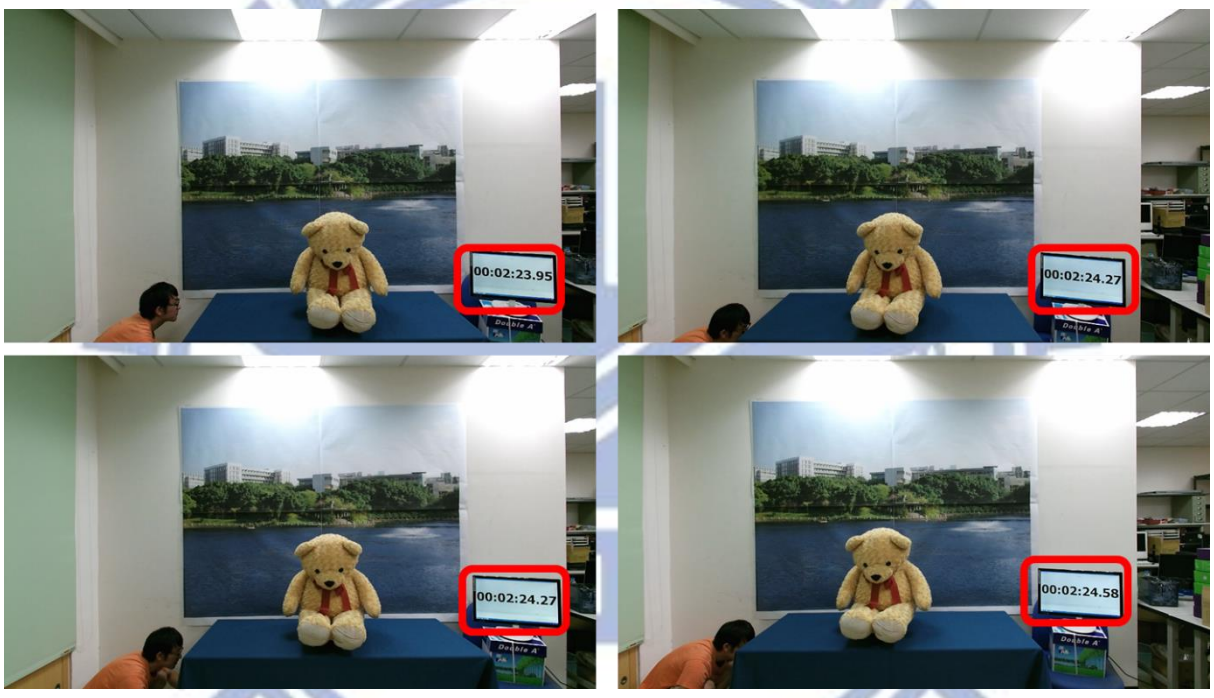


Figure 6-2 The four color images captured from four Kinect v2 cameras without solving asynchronization problem.

Reference [48] proposed method that enables four Kinect v2 sensors to communicate via TCP/IP and Ethernet network. However, it is hard to implement this real synchronization mechanism. Therefore, this method is not suitable in our case. Hence, to handle the asynchronization problem, we use a system time calibration software, NTPClock.exe [47], developed by National Time and Frequency Standard Laboratory. It checks with https://time.is/ to make sure the system time of the four computers are synchronized. The

NTPClock.exe software calibrates the system time clock of the computer. It can achieve 0.00s time skew with the national standard time. After we have calibrated the system time of four computers, we control four Kinect v2 cameras to record at the same system time and capture the color images and depth maps at each 0ms. Hence we can successfully synchronize all the four cameras recording at the same time, as shown in Figure 6-3. However, the frame rate is only 1 fps because we only trigger the camera to capture images at each 0ms. If we trigger the camera at 0ms and 0.5s, the computer cannot handle such huge amounts of data in 0.5s. Hence, we need to skip many frames, and this makes the synchronization problem worse. At the cost that the frame rate is only 1 fps, we can solve the asynchronization problem of four Kinect v2 devices. The speed bottleneck is the file write speed. The recording frame rate can be increased by replacing the current hard disk by solid-state disk.

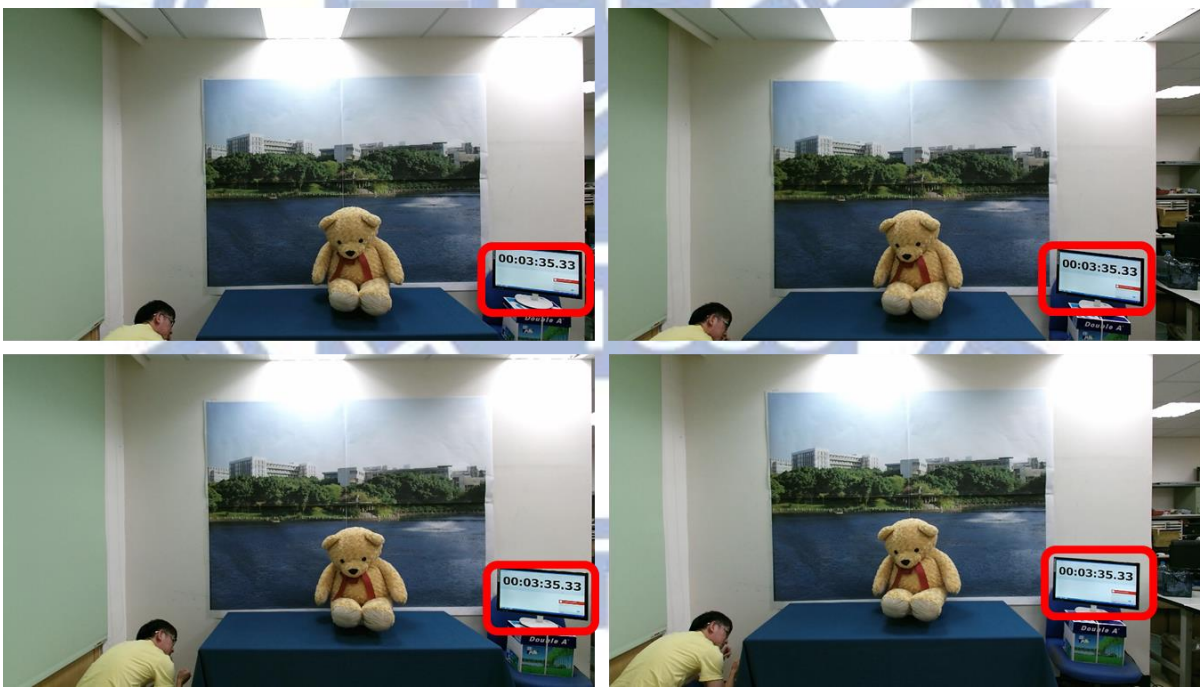


Figure 6-3 Four color images captured by four Kinect v2 cameras at the same time.

6.2. Multi-view Blending

In this section, we introduce the proposed multi-view blending algorithm. It consists of three parts. Before describing the proposed multi-view blending algorithm, we first analyze

the problem of the four warped virtual views we need to handle. The first problem is that the misalignment between reference color and depth picture causes the wrongly warped pixels after 3D warping, as shown in Figure 6-4. We call this problem as texture-depth misalignment. The sparse foreground pixels near the holes are wrongly warped pixels, as illustrated by Figure 6-5. This makes the blending result worse. We handle these wrongly warped pixels in our multi-view blending algorithm part 2 by choosing the correct color pixels from the other warped virtual views to be the synthesized color pixel.

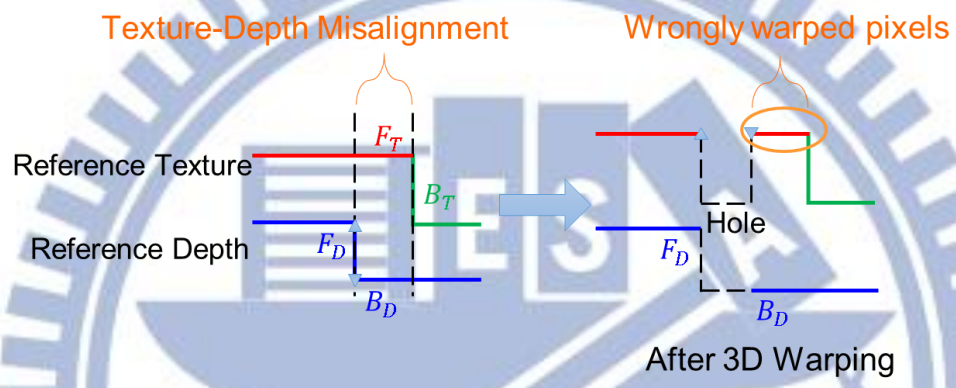


Figure 6-4 Illustration of misalignment between reference color and depth cause wrongly warped pixels.

Figure 6-5 Illustration of the wrongly warped pixels due to texture-depth misalignment.

The second problem of four warped virtual views is that the misalignment between four warped virtual colors, as shown in Figure 6-6. We call this problem as texture-texture misalignment. Most of the pixels of the four warped virtual views have texture-texture misalignment because it is hard to have perfectly accurate camera parameters or exactly correct depth maps. This problem produces the ghost effects in the blended pictures and blur not only on the object edge, but also in the depth smooth regions, as shown in Figure 6-7. To solve this texture-texture misalignment problem, we synthesize the pixels in the depth smooth region in our multi-view blending algorithm part 1 by choosing the dominant virtual color pixel to be the synthesized color pixel, and we synthesize the pixels near depth edges in our multi-view blending algorithm part 2 and part 3. The mechanism of choosing the dominant virtual view will be described later.

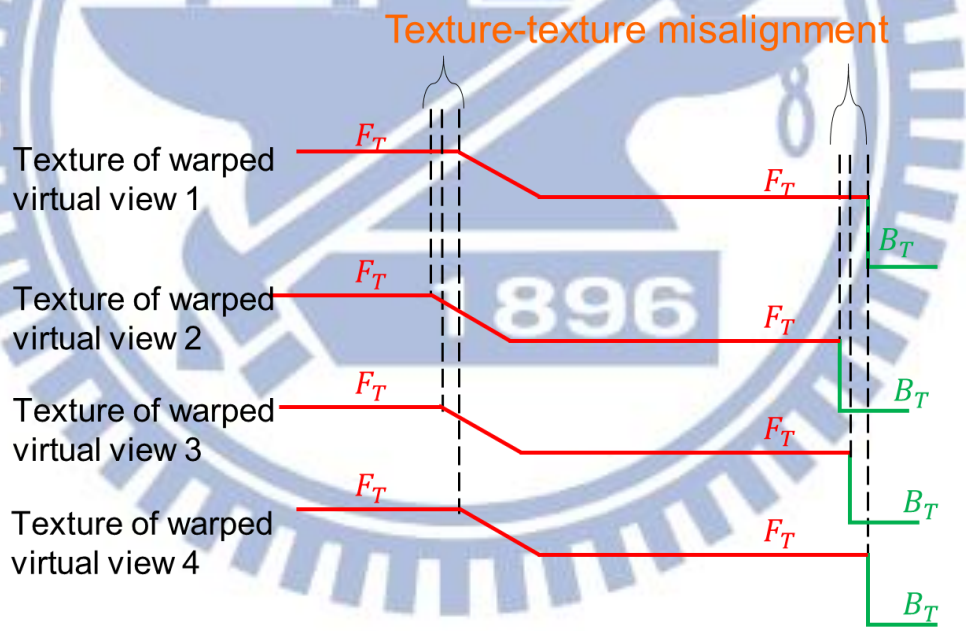


Figure 6-6 Illustration of misalignment between four warped virtual views.

Figure 6-7 Illustration of the blending result with the effect texture-texture misalignment.

We have observed that the depth value of object boundaries is often unreliable due to texture-depth misalignment and texture-texture misalignment. The depth value in the smooth region is often more accurate. Hence, we will first extract the depth edge from the original reference depth map, dilate the depth edge 10-pixel-width (pre-warp dilation), warp the dilated depth edge to the virtual view position, and dilate 3-pixel-width (post-warp dilation). Finally, we mark those regions as *MarkedPoint*. Figure 6-8 shows the flow chart of extracting *MarkedPoint*. The reason why we do pre-warp dilation and post-warp dilation is because the warped edge cannot cover the depth edge of virtual view well, we thus do post-warp dilation to cover the depth edge in virtual view.

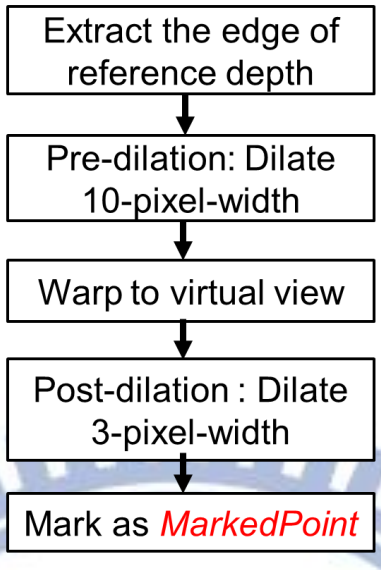


Figure 6-8 Flow chart of extracting *MarkedPoint*.

The proposed multi-view blending algorithm renders the virtual color and depth based on the notion of the dominant virtual view. The dominant virtual view is chosen according to the position of virtual camera. It chooses the warped virtual view whose camera position is closest to the virtual camera. The remaining three warped virtual views are denotes as *rem1*, *rem2*, and *rem3*.

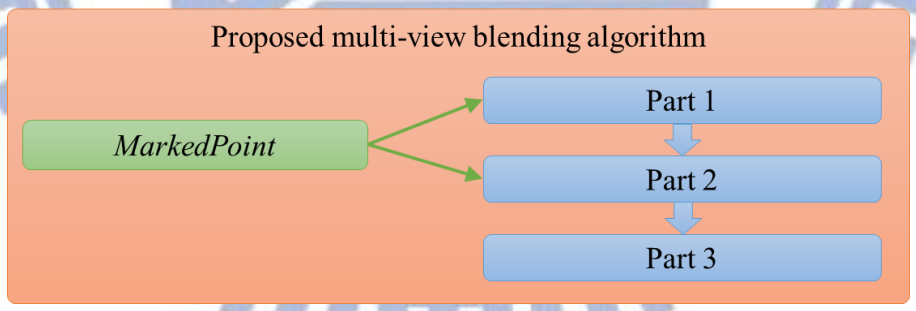


Figure 6-9 Flow chart of the proposed multi-view blending algorithm.

Figure 6-9 shows the flow chart of the proposed multi-view blending algorithm. The first part of the proposed multi-view blending algorithm is called in **Multi-view Blending Algorithm Part 1**. It renders the virtual view with the dominant virtual view if the depth of dominant virtual view $D_{dom}(u,v)\ exist$ and $MarkedPoint(u,v)\ is\ false$. This is because we have observed that the depth value of smooth regions are generally correct, we can use it

to synthesize the virtual pixels. This part keeps the synthesized quality clearly and solves the texture-texture misalignment problem in the depth smooth regions.

The second part of the proposed multi-view blending algorithm is **Multi-view Blending Algorithm Part 2**. It handles the virtual depth pixels marked with $MarkedPoint$ in this phase. We first check the similarity between the depth of dominant virtual view and its associated background depth. If they are close, then we will check the depth similarity of the remaining three warped virtual depth pixels. We record its color if its associated depth is close to its background depth. Afterward, we evaluate the similarity between each recorded color, and choose the most similar colors below a threshold to compute their variance and mean. If the variance less than a threshold (means the chose most similar colors are the same color), and the mean is close to the color value of dominant virtual color (means the chose most similar colors are similar to the dominant virtual color), we render the virtual color and depth with the dominant warped virtual view. If the mean is not close to the color of dominant virtual color (means the chose most similar colors are not similar to the dominant virtual color and either the dominant virtual color or the chose most similar colors are the wrongly warped pixels), we will choose one of the warped virtual color whose color is closest to the valid 8-neighbor color values of the synthesized virtual color image. If a pixel has an invalid depth value, it is not a valid 8-neighbor.

The purpose of **Multi-view Blending Algorithm Part 2** is to prevent the false rendering result due to the misalignment of the four warped virtual views (texture-texture misalignment). Second, it also can suppress the wrong rendering result due to the color and depth of the reference view are not match well (texture-depth misalignment). As a result, it can retain good synthesized quality in the non-depth edge regions and remove the wrong pixels in depth edge regions. By using four RGB-D cameras, we have enough information to decide which pixel in the warped virtual views is the wrongly warped pixel and we remove it away by choosing the correct virtual color pixel from the other warped virtual views to be the synthesized pixel.

The third part of the proposed multi-view blending algorithm is **Multi-view Blending Algorithm Part 3**. So far, the remaining depth holes of the rendered virtual view belong to three cases. First, the depth of dominant virtual view does not exist. Second, *num_element* in part 2 is less than 2. Third, *variance* in part 2 is larger than 200. Hence, we fill in the pixel using one of the warped virtual color, whose color is closest to the valid 8-neighbor color values of the synthesized color image. This is an iterative procedure. In each iteration a certain number pixels are filled using the above procedure. The valid pixel number is thus increased. Therefore, more pixels may be fixed in the next iteration.

Multi-view Blending Algorithm Part 1: Render the virtual pixels that are not near depth edge. (For solving texture-texture misalignment.)

---

for v=1:image_height

for u=1:image_width

  if $D_{dom}(u,v)\ exist\ and\ MarkedPoint(u,v)\ is\ false$

    $I_{virtual} = I_{dom}(u,v)$

    $D_{virtual} = D_{dom}(u,v)$

  end

end

end

---

where *I* and *D* indicate the color value and disparity value, respectively. The subscript *virtual* means virtual view which we want to synthesize. The subscript *dom* indicates dominant virtual view. The subscript *rem1*, *rem2*, and *rem3* are the warped virtual view except for the dominant virtual view, as described before. The algorithms below use the same notations. If the subscript contains the *bk* word, it means the associated background.

Multi-view Blending Algorithm Part 2: Render the virtual pixels near depth edge. (For solving texture-depth misalignment and texture-texture misalignment.)

hole_num =hole in $D_{virtual}$ with much more than one valid depth existing in 8-neighbor

while hole_num $\neq$ 0

for v=1:image_height

for u=1:image_width

if $MarkedPoint(u,v)$ *is true and* $\left|D_{dom}(u,v) - D_{bk_dom}(u,v)\right| \leq bk_d_th$

$I_{temp} = [\, I_{temp} \,;\, I_{dom}(u,v) \,]$

if $\left|D_{rem1}(u,v) - D_{bk_rem1}(u,v)\right| \leq bk_d_th$

$I_{temp} = [\, I_{temp} \,;\, I_{rem1}(u,v) \,]$

end

if $\left|D_{rem2}(u,v) - D_{bk_rem2}(u,v)\right| \leq bk_d_th$

$I_{temp} = [\, I_{temp} \,;\, I_{rem2}(u,v) \,]$

end

if $\left|D_{rem3}(u,v) - D_{bk_rem3}(u,v)\right| \leq bk_d_th$

$I_{temp} = [\, I_{temp} \,;\, I_{rem3}(u,v) \,]$

end

num_elements = element number of I_{temp}

if *num_elements* $\geq$ 2

for *source_ele*=1 : *num_elements* -1

for *target_ele*= *source_ele* : *num_elements*

if $\left|I_{temp}(source_ele) - I_{temp}(target_ele)\right| \leq c_th$

similar_count($source_ele$)= *similar_count*($source_ele$) + 1

similar_count($target_ele$)= *similar_count*($target_ele$) + 1

end

end

end

variance = variance(elements of I_{temp} have the largest *similar_count*)

mean = mean(elements of I_{temp} have the largest *similar_count*)

if *variance* <= *variance_threshold*

if $|mean - I_{dom}(u,v)| \leq$ *mean_difference_threshold*

$I_{virtual}(u,v) = I_{dom}(u,v)$

$D_{virtual}(u,v) = D_{dom}(u,v)$

else

for y=-1:1

for x=-1:1

if $D_{virtual}(x,y)\ exist$

error1=error1 + $(D_{virtual}(x,y) - I_{dom}(u,v))^2$

error2=error2 + $(D_{virtual}(x,y) - I_{rem1}(u,v))^2$

error3=error3 + $\left(D_{virtual}(x,y) - I_{rem2}(u,v)\right)^2$

error4=error4 + $(D_{virtual}(x,y) - I_{rem3}(u,v))^2$

end

if min(error1, error2, error3, error4) = error1

$I_{virtual}(u,v) = I_{dom}(u,v)$

$D_{virtual}(u,v) = D_{dom}(u,v)$

end

if min(error1, error2, error3, error4) = error2

$I_{virtual}(u,v) = I_{rem1}(u,v)$

$D_{virtual}(u,v) = D_{rem1}(u,v)$

end

if min(error1, error2, error3, error4) = error3

$I_{virtual}(u,v) = I_{rem2}(u,v)$

$D_{virtual}(u,v) = D_{rem2}(u,v)$

end

if min(error1, error2, error3, error4) = error4

$I_{virtual}(u,v) = I_{rem3}(u,v)$

$D_{virtual}(u,v) = D_{rem3}(u,v)$

end

end %% End of if for x=-1:1

end %% End of if for y=-1:1

end %% End of if $|mean - I_{dom}(u,v)| \leq$ *mena_difference_threshold*

end %% End of if *variance* $\leq$ *variance_threshold*

end %% End of if *num_element* ≥ 2

end %% End of *MarkedPoint*(u,v) is true and $\left|D_{dom}(u,v) - D_{bk_dom}(u,v)\right| \leq bk_d_th$

end %% End of for u=1:image_width

end %% End of for v=1:image_height

end %% End of while

Multi-view Blending Algorithm Part 3: Render the remaining virtual pixels by choosing the correct color pixel from the other warped virtual views. (For solving texture-texture misalignment)

hole_num =hole in $D_{virtual}$ with much more than one depth existing on 8-neighbor

while hole_num $\neq$ 0

for y=-1:1

for x=-1:1

if $D_{virtual}(x,y)\ exist$

error1=error1 + $(D_{virtual}(x,y) - I_{dom}(u,v))^2$

error2=error2 + $(D_{virtual}(x,y) - I_{rem1}(u,v))^2$

error3=error3 + $\left(D_{virtual}(x,y) - I_{rem2}(u,v)\right)^2$

error4=error4 + $(D_{virtual}(x,y) - I_{rem3}(u,v))^2$

end

if min(error1, error2, error3, error4) = error1

$I_{virtual}(u,v) = I_{dom}(u,v)$

$D_{virtual}(u,v) = D_{dom}(u,v)$

end

if min(error1, error2, error3, error4) = error2

$I_{virtual}(u,v) = I_{rem1}(u,v)$

$D_{virtual}(u,v) = D_{rem1}(u,v)$

end

if min(error1, error2, error3, error4) = error3

$I_{virtual}(u,v) = I_{rem2}(u,v)$

$D_{virtual}(u,v) = D_{rem2}(u,v)$

end

```
    if min(error1, error2, error3, error4) = error4
        I_virtual(u, v) = I_rem3(u, v)
        D_virtual(u, v) = D_rem3(u, v)
    end
end
end
end
```

The parameter values are decided empirically. In this study, we use the following values: *bk_d_th*=5, *c_th*=20, *variance_threshold*=200, and *mean_difference_threshold*=7. We did rather extensive experiments to choose these parameters. If the *bk_d_th* is much higher than 5, it may accept the wrong pixels in doing **multi-view blending algorithm part 2**. If it is much lower than 5, there will be only few of the pixels synthesized in **multi-view blending algorithm part 2**. This makes the synthesized results from **multi-view blending algorithm part 3** worse because we cannot provide enough pixel near the object edge. If the *c_th* is much higher than 20, it may mark two wrong color pixels as the similar color. If it is much lower than 20, two similar color pixels may not be marked as similar color. If the *variance_threshold* is much higher than 200, the wrong color pixel may choose as the synthesized pixels in the later stage of **multi-view blending algorithm part 2**. If *variance_threshold* is much lower than 200, this tight condition makes the synthesized results worse. If the *mean_difference_threshold* is much higher than 7, the wrong color pixel may choose to be the synthesized pixel. If it is much lower than 7, the correct color pixel may not chose to be the synthesized pixel.

7. Experimental Results and Discussions

In this section, we show the results of the two test sequences in our experiments. Figure 7-1 shows again our view synthesis system. We also show the experimental results in each block of Figure 7-1.

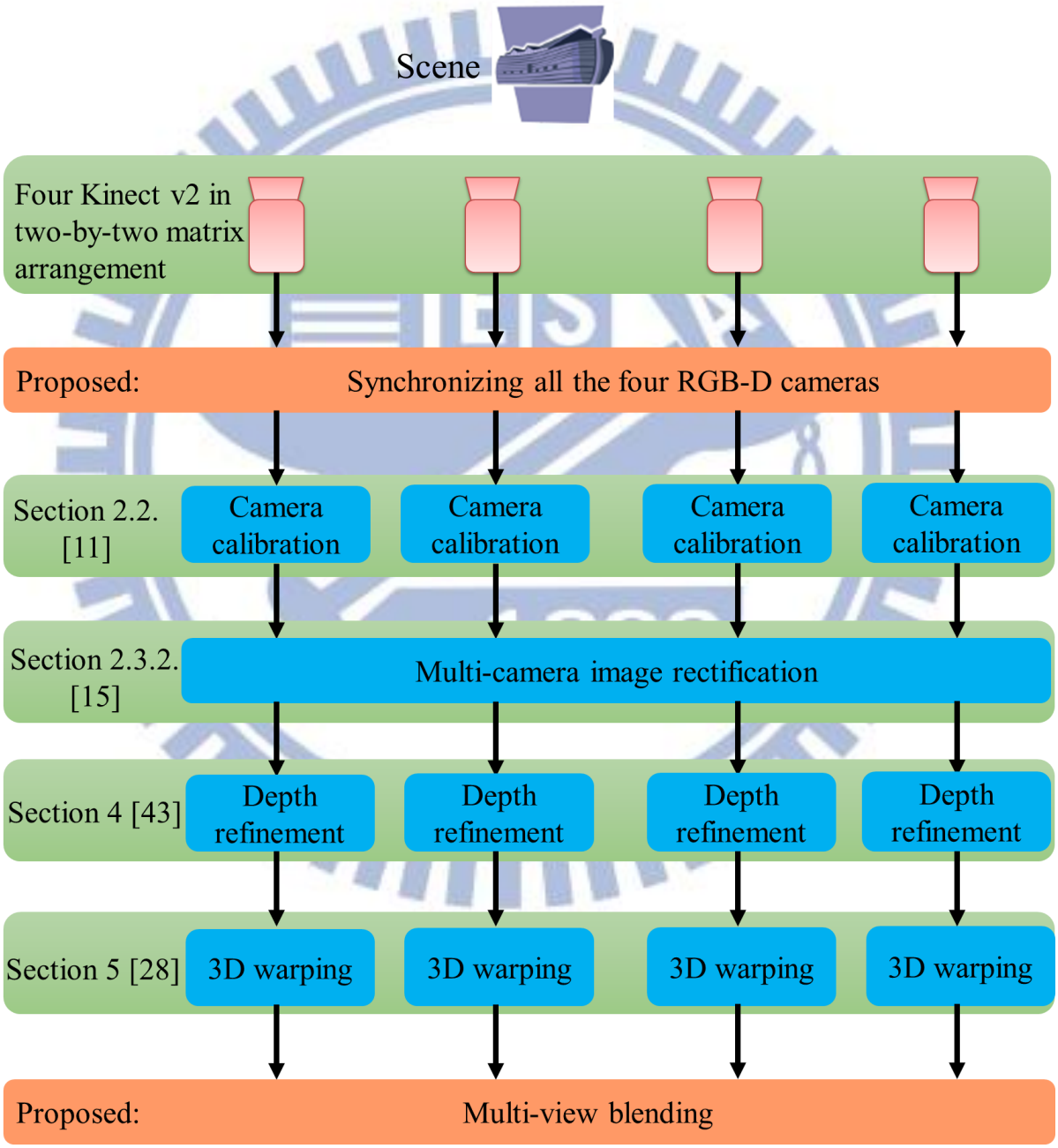


Figure 7-1 Flowchart of the proposed wide angle view synthesis system.

The two test sequences were captured in our Lab. The first test sequence is toy bear moving. It consists of 7 frames. The second test sequence is one toy bear rotating and the other bear is stationary. It consists of 49 frames. We will show the synthesized virtual view video with various virtual camera positions. The virtual camera has translation in x-axis or y-axis, rotation along x-axis (virtual camera tilt down), and zoom in/out along z-axis. Figure 7-2 shows the setup of the four Kinect v2 sensors in two-by-two arrangement.

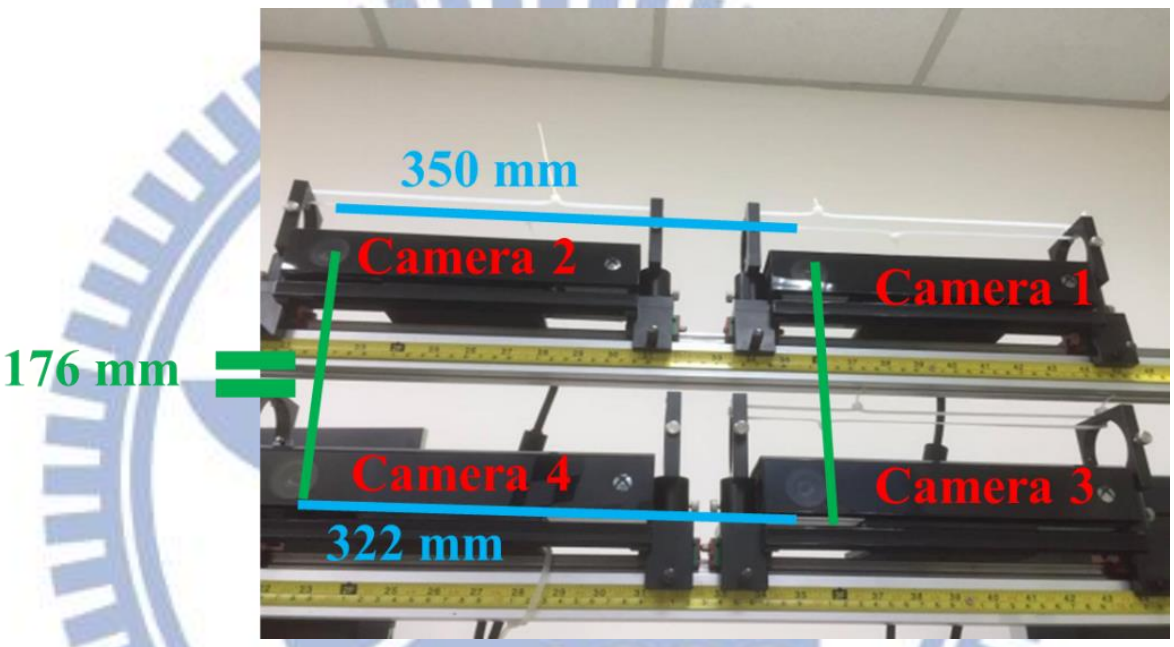


Figure 7-2 Setup of four Kinect v2 sensors.

7.1. First Test Sequence

The first test sequence is toy bear moving. It has seven frames. We show four frames of this sequence captured by four Kinect v2 sensors in Figure 7-3, Figure 7-4, Figure 7-5, and Figure 7-6.

(a)

(b)

(c)

(d)

Figure 7-3 1st frame of the sequences captured from **(a)** camera 1 **(b)** camera 2 **(c)** camera 3 **(d)** camera 4.

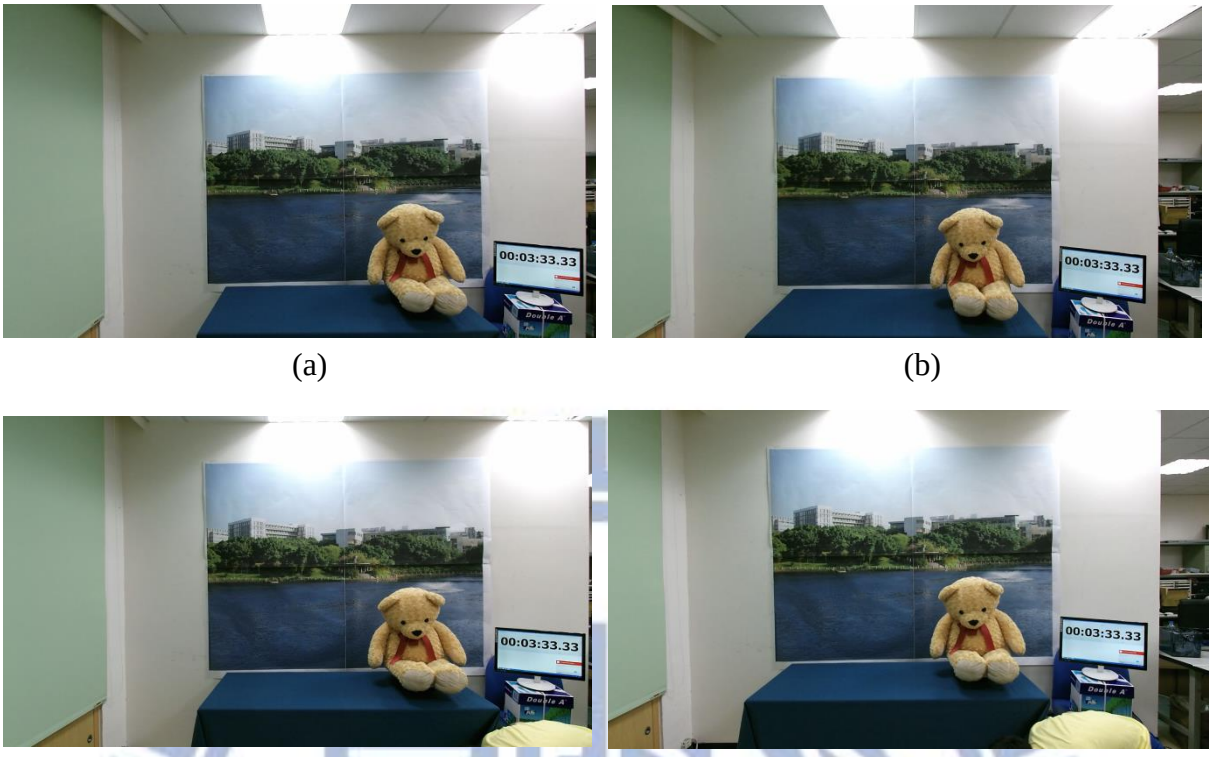
(a)

(b)

(c)

(d)

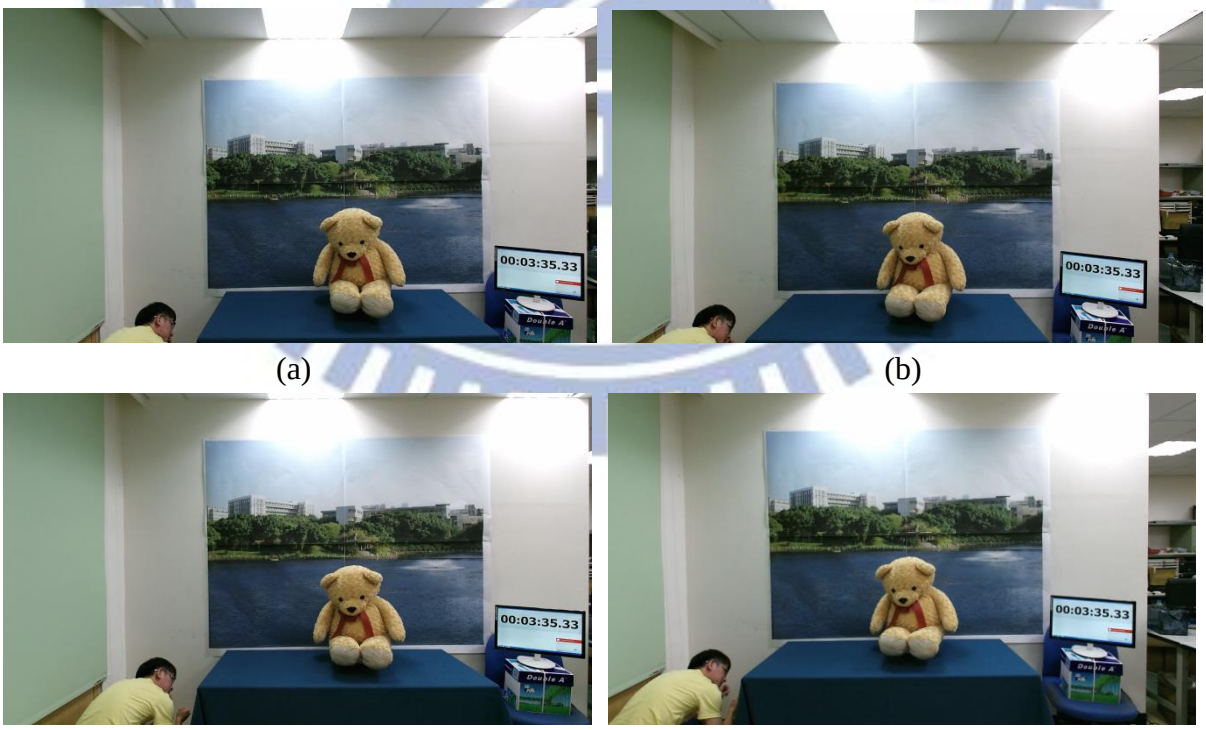
Figure 7-4 2st frame of the sequences captured from **(a)** camera 1 **(b)** camera 2 **(c)** camera 3 **(d)** camera 4.



(a) (b)

(c) (d)

Figure 7-5 3st frame of the sequences captured from **(a)** camera 1 **(b)** camera 2 **(c)** camera 3 **(d)** camera 4.



(a) (b)

(c) (d)

Figure 7-6 4st frame of the sequences captured from **(a)** camera 1 **(b)** camera 2 **(c)** camera 3 **(d)** camera 4.

7.1.1. Camera Calibration

In this sub-section, we briefly show the procedure of camera calibration. Figure 7-7 is the calibration flowchart adopted in our experiments. We use the camera calibration toolbox in Matlab [12] to do camera calibration. We need to capture 10 ~ 20 calibration patterns by using the checkerboard, as shown in Figure 7-8. First, keyin calib_gui in the MATLAB command window and click on the standard mode. Then click on the extract grid corners, as shown in Figure 7-9. After we finish the above process, we can do the remaining calibration procedures to obtain the camera intrinsic parameters, as described in (82) ~ (85). For estimating the camera extrinsic parameters, we capture another checkerboard pattern and extract the grid corners to calculate the camera extrinsic parameters, as shown in Figure 7-10. We conduct this camera calibration procedure to obtain the camera intrinsic and extrinsic parameters of these four Kinect v2 sensors.

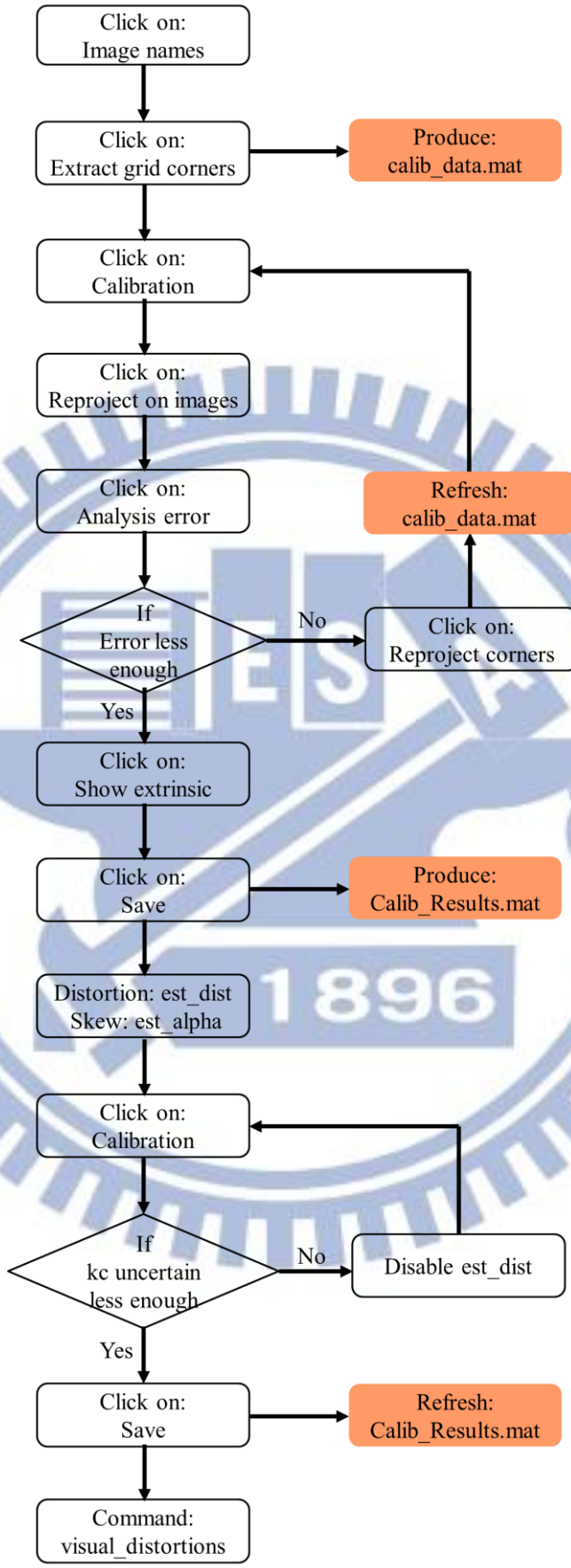


Figure 7-7 Flowchart of camera calibration.

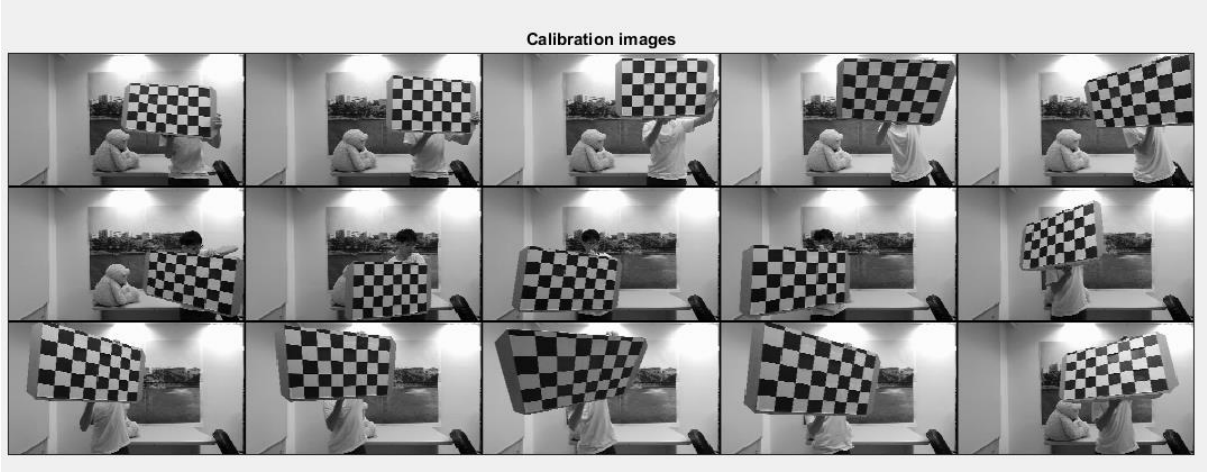


Figure 7-8 Calibration patterns.

Figure 7-9 Extract grid corners for computing camera intrinsic parameter.

Figure 7-10 Extract grid corners for computing camera extrinsic parameter.

$$K_1 = \begin{bmatrix} 1051.8 & 0 & 966.2 \\ 0 & 1048.3 & 546.9 \\ 0 & 0 & 1 \end{bmatrix}, [R_1|t_1] = \begin{bmatrix} 0.9997 & -0.0082 & 0.0210 & 108.7 \\ -0.0073 & -0.9992 & -0.0396 & 747.9 \\ 0.0213 & 0.0394 & -0.9990 & 1988 \end{bmatrix}, \quad (82)$$

$$K_2 = \begin{bmatrix} 1053.9 & 0 & 974 \\ 0 & 1051.0 & 556 \\ 0 & 0 & 1 \end{bmatrix}, [R_2|t_2] = \begin{bmatrix} 0.9999 & 0.0028 & 0.0108 & -211.9 \\ 0.0033 & -0.9993 & -0.0382 & 745.1 \\ 0.0107 & 0.0383 & -0.9992 & 1978.2 \end{bmatrix}, \quad (83)$$

$$K_3 = \begin{bmatrix} 1065.7 & 0 & 957 \\ 0 & 1062.4 & 559 \\ 0 & 0 & 1 \end{bmatrix}, [R_3|t_3] = \begin{bmatrix} 0.9999 & -0.0056 & 0.0086 & 139.9 \\ -0.0054 & -0.9998 & -0.0204 & 549.4 \\ 0.0087 & 0.0204 & -0.9998 & 2006.5 \end{bmatrix}, \quad (84)$$

$$K_4 = \begin{bmatrix} 1061.1 & 0 & 970 \\ 0 & 1056.3 & 544 \\ 0 & 0 & 1 \end{bmatrix}, [R_4|t_4] = \begin{bmatrix} 0.9999 & -0.0075 & 0.0155 & -212.6 \\ -0.0074 & -0.9999 & -0.0104 & 525.6 \\ 0.0156 & 0.0103 & -0.9998 & 2016.9 \end{bmatrix}. \quad (85)$$

7.1.2. Multi-camera Image Rectification

After the camera calibration is completed, we use the camera intrinsic and extrinsic parameters, (82) ~ (85), to compute the ideal camera parameters and image rectifying transformation matrix (IRTM), as shown in (86) ~ (92). The IRTM will be used to rectify the captured images to make their epipolar lines aligned horizontally. Figure 7-11 and Figure 7-12 show the un-rectified images and rectified images of four cameras. In Figure 7-11, we can see that the epipolar lines are not exactly horizontal. After rectifying the images, the epipolar lines become parallel to the horizontal line, as shown in Figure 7-12.

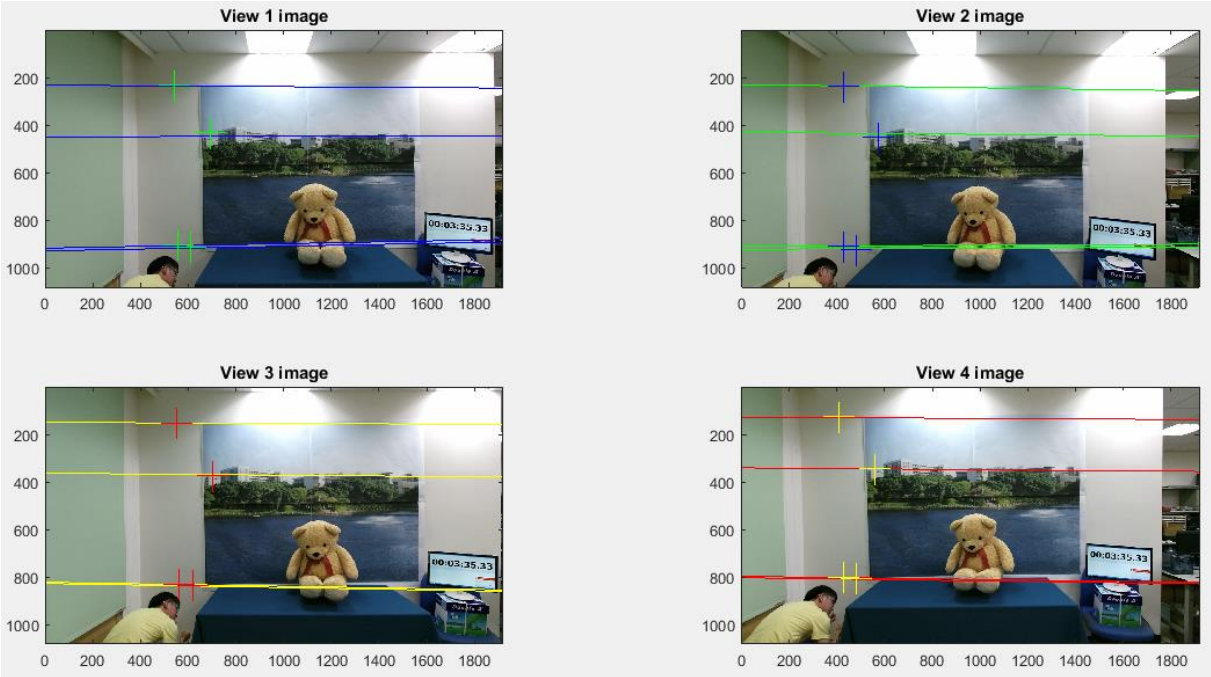


Figure 7-11 Un-rectified color image of four reference views.

$$K_{1,ideal} = K_{2,ideal} = K_{3,ideal} = K_{4,ideal} = \begin{bmatrix} 1058.1 & 0 & 967.1 \\ 0 & 1054.5 & 551.7 \\ 0 & 0 & 1 \end{bmatrix}, \tag{86}$$

$$R_{1,ideal} = R_{2,ideal} = R_{3,ideal} = R_{4,ideal} = \begin{bmatrix} 0.9999 & -0.0052 & 0.0121 \\ -0.0050 & -0.9997 & -0.0228 \\ 0.0122 & 0.0228 & -0.9997 \end{bmatrix}, \tag{87}$$

$$t_{1,ideal} = \begin{bmatrix} 135.7 \\ 726.0 \\ 2000.8 \end{bmatrix}, \; t_{2,ideal} = \begin{bmatrix} -214.3 \\ 727.8 \\ 1996.5 \end{bmatrix}, t_{3,ideal} = \begin{bmatrix} 120.8 \\ 550.2 \\ 2004.7 \end{bmatrix}, t_{4,ideal} = \begin{bmatrix} -201.2 \\ 551.8 \\ 2000.7 \end{bmatrix}, \tag{88}$$

$$IRTM_{cam1} = \begin{bmatrix} 0.9980 & 0.0148 & 9.7122 \\ -0.0017 & 1.0165 & -14.8449 \\ 0 & 0 & 0.9998 \end{bmatrix}, \tag{89}$$

$$IRTM_{cam2} = \begin{bmatrix} 1.0052 & 0.0226 & -26.0115 \\ -0.0080 & 1.0139 & -13.7923 \\ 0 & 0 & 0.9932 \end{bmatrix}, \tag{90}$$

$$IRTM_{cam3} = \begin{bmatrix} 0.9956 & -0.0043 & 6.5294 \\ 0.0021 & 0.9907 & -4.1709 \\ 0 & 0 & 0.9978 \end{bmatrix}, \tag{91}$$

$$IRTM_{cam4} = \begin{bmatrix} 0.9942 & -0.0141 & 11.5978 \\ 0.0006 & 0.9912 & 22.9354 \\ 0 & 0 & 1.0050 \end{bmatrix}. \tag{92}$$

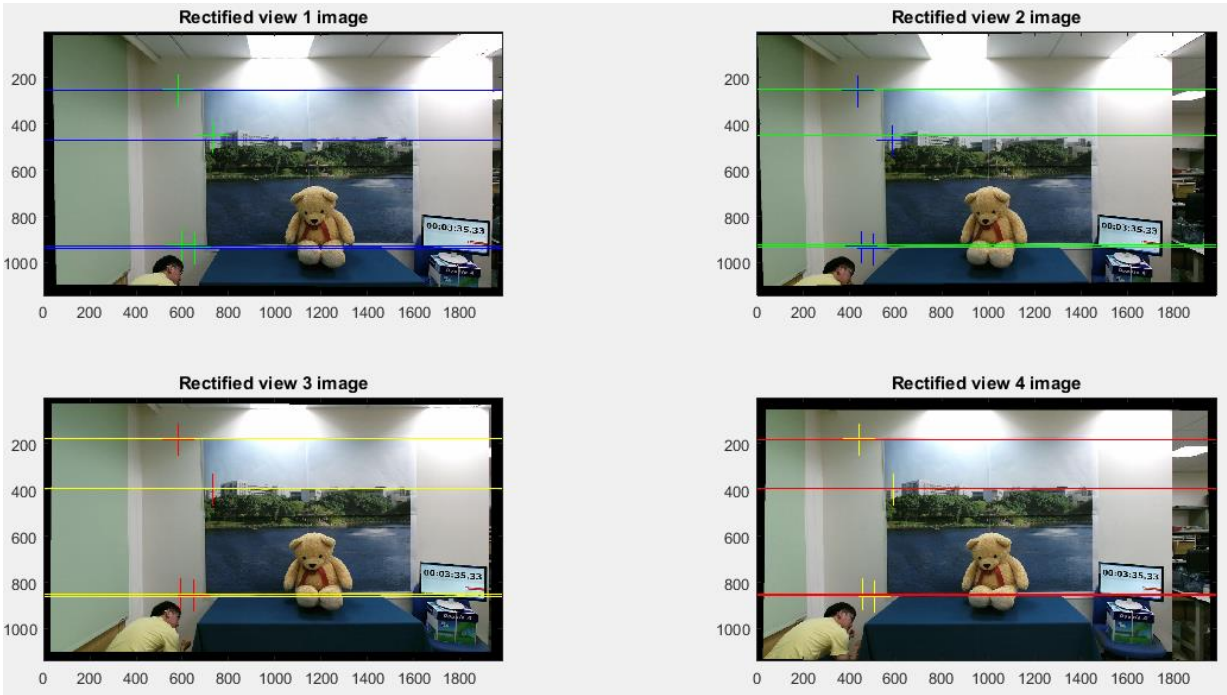


Figure 7-12 Rectified color image of the four reference views.

7.1.3. Depth Map Refinement for Kinect v2 Before 3D Warping

The captured raw depth maps are noisy. Hence, we refine the depth maps by using the method proposed in [43]. Figure 7-13 and Figure 7-14 show the raw depth map and refined depth map, respectively. The original depth map is in grayscale. However, it is too dark to see. Hence, for more clear reading, we draw it in pseudo color format.

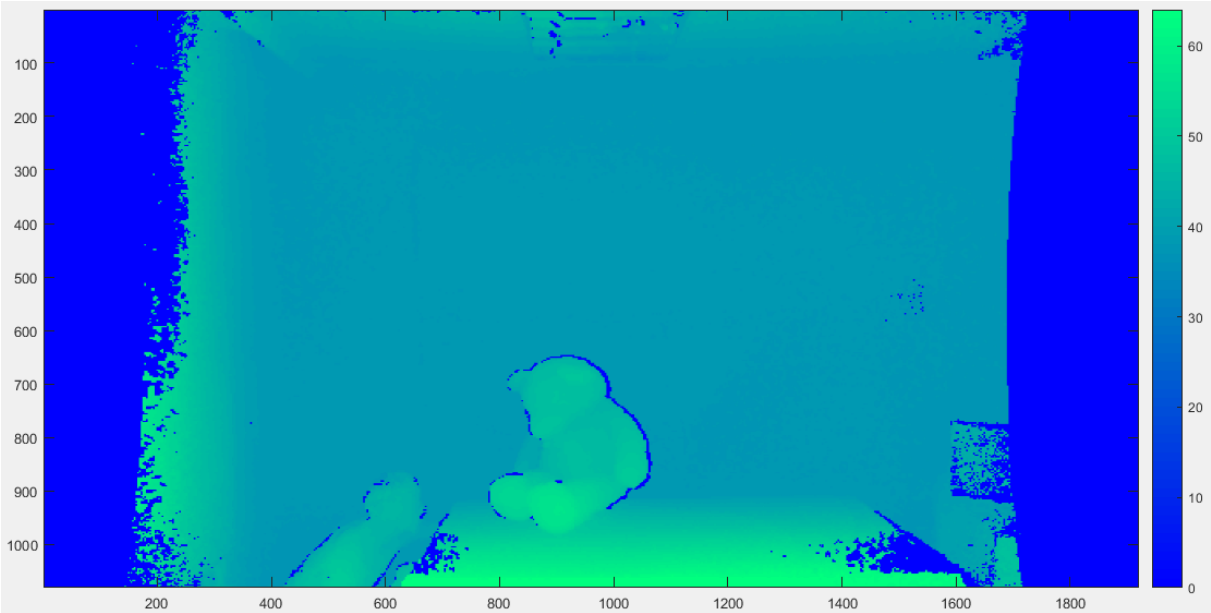


Figure 7-13 Raw depth map.

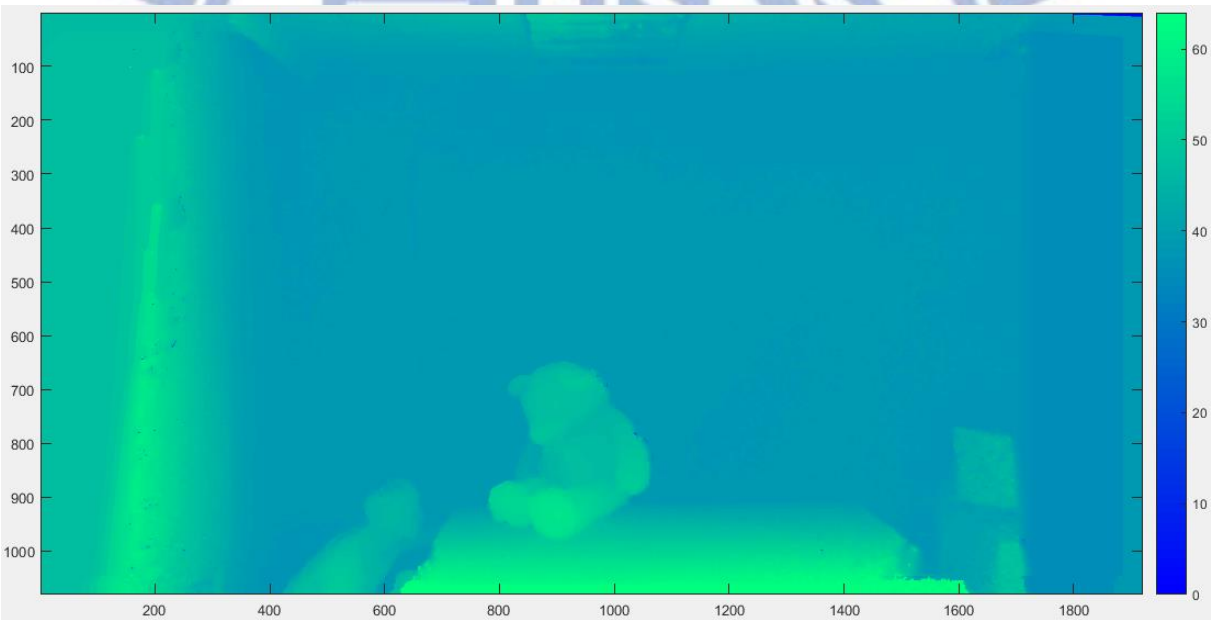


Figure 7-14 Refined depth map.

7.1.4. 3D Warping Before Wide Angle View Merging

After refining the raw depth maps, we apply 3D warping algorithm proposed in [29] to warp the four reference views into the virtual viewpoint. Figure 7-15 and figure 7-16 show the original four reference views and four warped virtual views with virtual camera tilt down 5 degrees and zoom in 100mm. Figure 7-17 shows the geometric relationship between the

virtual camera and the four reference cameras.

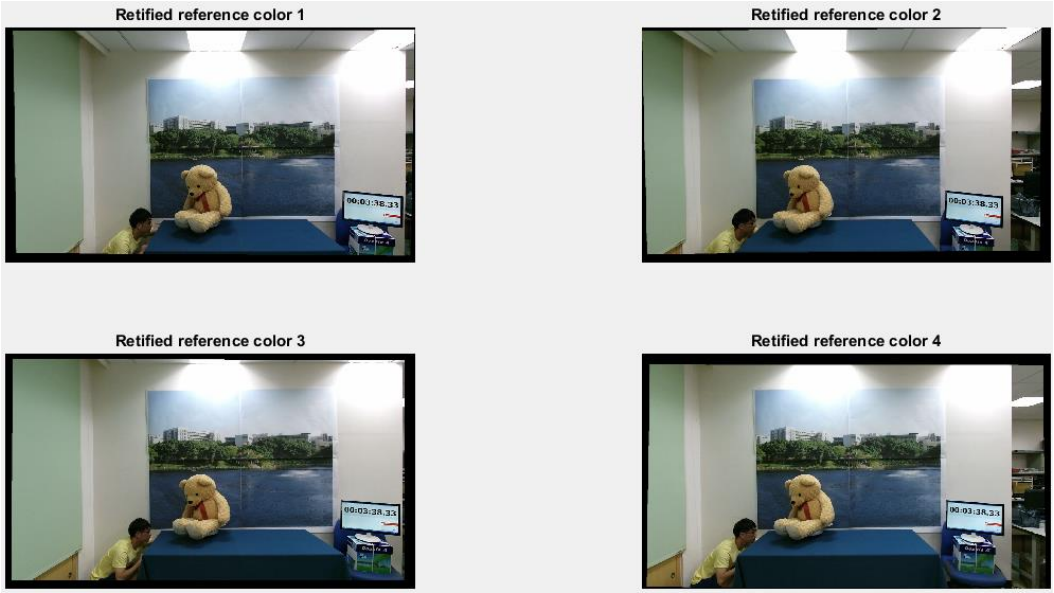


Figure 7-15 Original reference views of the four cameras.

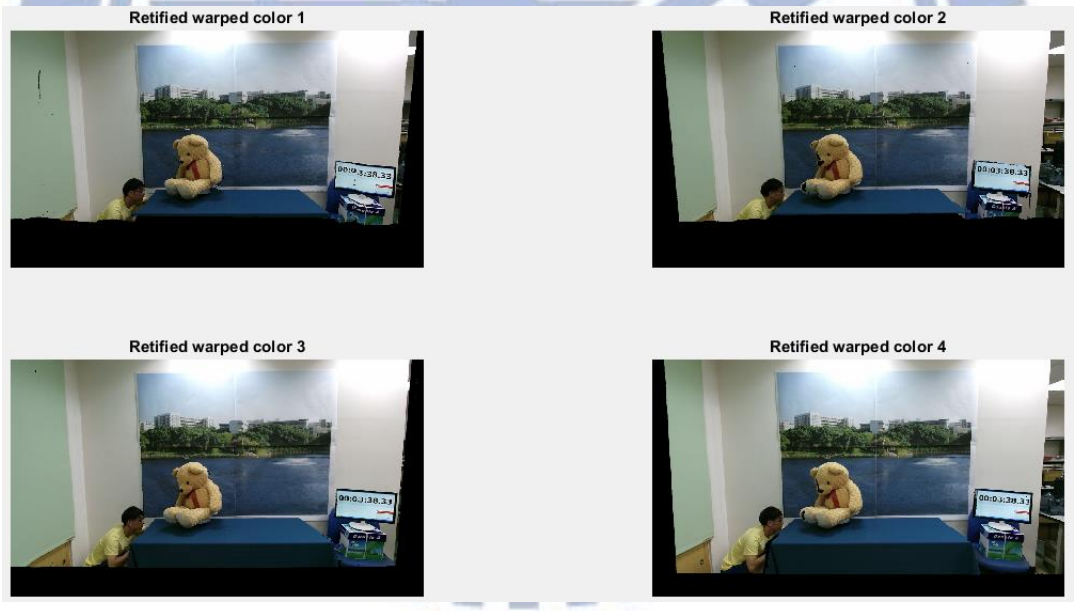


Figure 7-16 Warped views of the four reference views with virtual camera tilt down 5 degrees and zoom in 100mm.

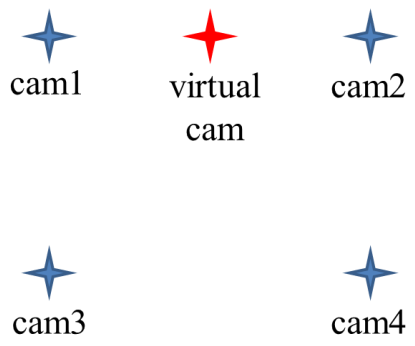


Figure 7-17 Geometric relationship of the virtual camera and the four reference cameras.

7.1.5. Synchronizing All Four RGB-D Cameras

In this section, we show the effect of synchronization among four cameras. A larger capturing time skew between four cameras causes more serious picture problem. Figure 7-18 shows the asynchronization of the four cameras. To understand the effect of asynchronization, we show three cases using the average blending. Figure 7-19 shows the synthesized virtual image with the maximum capturing time skew of four cameras is 0.12 seconds. The misalignment is obvious. When the capturing time skew of four cameras grows to 0.81 seconds, the synthesized image suffers more serious misalignment, as shown in Figure 7-20. By using the proposed synchronization method in this thesis in Section 6.1, we can achieve 0.00 second capturing time skew, and the synthesized image aligns well in the moving object, as illustrated in Figure 7-21 and Figure 7-22.

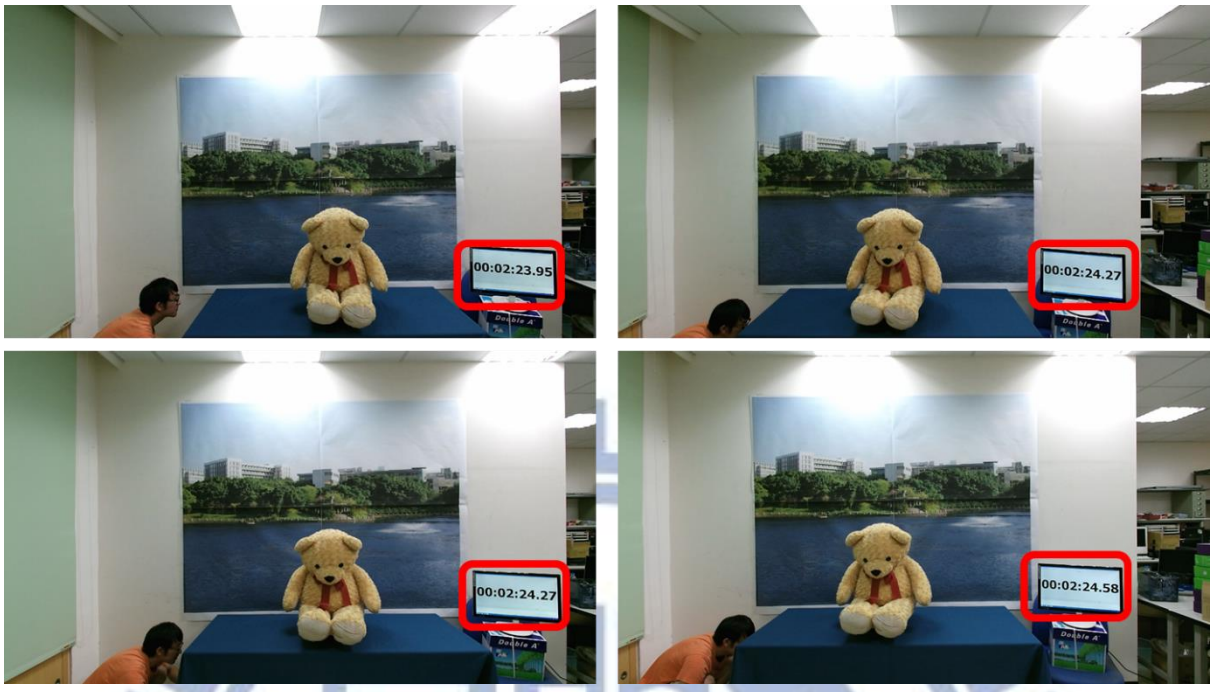


Figure 7-18 Four color images captured from the four Kinect v2 cameras with asynchronization problem.

Figure 7-19 Synthesized image using average blending and maximum capturing time skew is 0.12 seconds.

Figure 7-20 Synthesized image using average blending and maximum capturing time skew is 0.81 seconds.

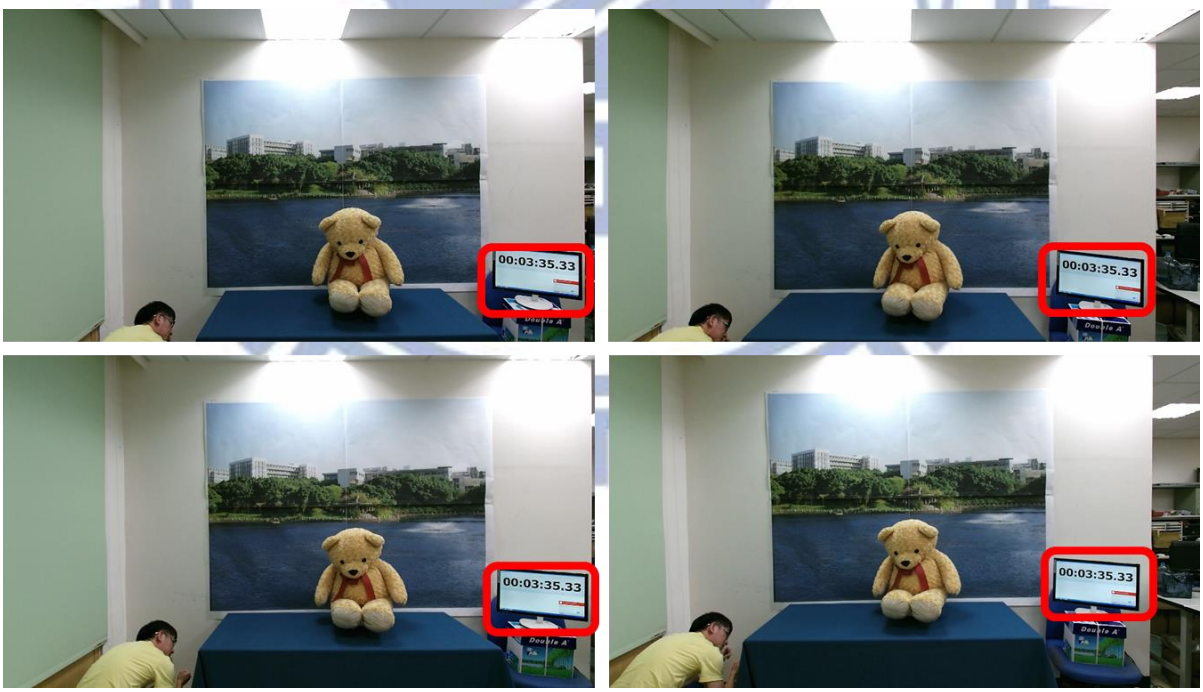


Figure 7-21 Four color images captured by four Kinect v2 cameras at the same time instance.

Figure 7-22 Synthesized image using average blending and maximum capturing time skew is 0.00 second.

7.1.6. Multi-view Blending

In this section, we first show the effects without/with post-warped depth edge dilation, as shown in Figure 7-23 and Figure 7-24. Because the position of color camera and depth sensor in Kinect v2 are different, the left and right picture borders of an image do not have either color or depth information. Thus, we clip the two borders. Then, we show the synthesized results using the proposed multi-view blending algorithm part 1, 2 and part 3. Third, we show the comparison of the blending results between the adaptive blending technique and the proposed multi-view blending algorithm. Finally, we show the five different experimental results at different virtual camera positions. The computing time of the proposed multi-view blending is about one minute per frame with Intel Core i7-4790 when operating at 3.8 GHz.

7.1.6.1. Without Post-warped Depth Edge Dilation

Figure 7-23(a) shows that the sparse wrong color pixels appear near the depth holes. It will cause poor blending result in the multi-view blending algorithm part 2 and 3.

Figure 7-23 Synthesized image without post-warp depth edge dilation, but use multi-view blending algorithm **(a)** part 1 **(b)** part 1 and part 2 **(c)** part 1, 2, and 3.

7.1.6.2. With Post-warped Depth Edge Dilation

As we can see, by using the post-warp depth edge dilation, we can reduce the wrong color pixels appear in result of **multi-view blending algorithm part 1**. Reducing these wrong color pixels can improve the synthesized picture quality in **multi-view blending algorithm part 2 and 3**, as shown in Figure 7-24(c).

(a) (b)

(c)

Figure 7-24 Synthesized image using post-warp depth edge dilation and multi-view blending algorithm **(a)** part 1 **(b)** part 1 and part 2 **(c)** part 1, 2, and 3.

7.1.6.3. Results of Multi-view Blending Algorithm Part 1

In this sub-section, we show the blending results of multi-view blending part 1. It uses the dominant virtual color and depth pixels to be the synthesized results for the pixel that is not near depth edge and the dominant virtual view exist at that pixel. This method removes the texture-texture misalignment in the synthesized result, as shown in Figure 7-25.

Figure 7-25 Blending result of multi-view blending algorithm part 1.

7.1.6.4. Results of Multi-view Blending Algorithm Part 2

In this sub-section, we show the blending results of multi-view blending part 2. It will iteratively generates pixels in the depth holes. Figure 7-26 illustrates the 4th, 8th, and 13th iteration blending results of multi-view blending algorithm part 2. It fills the holes by using the color pixels from four warped virtual views or from dominant virtual view.

Figure 7-26 Iteratively blending results of multi-view blending algorithm part 2 in **(a)** 4th iteration **(b)** 8th iteration **(c)** 13th iteration.

7.1.6.5. Results of Multi-view Blending Algorithm Part 3

In this sub-section, we show the blending results of the multi-view blending algorithm part 3. It fills the holes by using the color pixels from the four warped virtual views. Figure 7-27 illustrates the 6th, 12th , and 18th iteration of blending results produce by multi-view blending algorithm part 3.

Figure 7-27 Iteratively blending results of multi-view blending algorithm part 3 in **(a)** 6th iteration **(b)** 12th iteration **(c)** 18th iteration.

7.1.6.6. Case 1: Tilt Down 2 Degrees, and Zoom In 100mm

Figure 7-28 shows the geometric relationship between the four reference cameras and the synthesized virtual camera. Figure 7-29 and Figure 7-30 show the final blending result by using the adaptive blending technique in [29], and the proposed multi-view blending algorithm when the input camera number varies from two to four. The synthesized results using the adaptive blending technique [29] cannot handle the wrongly warped pixels due to texture-depth misalignment, as shown in Figure 7-29(a) and Figure 7-30(a). Using the proposed multi-view blending algorithm with the input camera same as adaptive blending, we can successfully remove the wrongly warped pixels near the object edge, as shown in Figure 7-29(b) and Figure 7-30(b). However, there are some wrongly synthesized pixel near the object edge. The blue pixels under the toy bear's right hand are the wrongly synthesized pixels. This is because this region is disocclusion region. Using the proposed multi-view blending algorithm with four input cameras, we not only efficiently remove the texture-depth misalignment problem, but also render the occlusion regions well, as shown in Figure 7-29(c) and Figure 7-30(c).

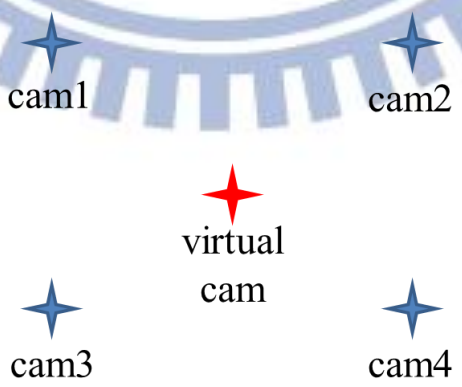


Figure 7-28 Geometric relationship between the four reference cameras and the synthesized virtual camera.

Figure 7-29 2nd frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera tilt down 2 degrees and zoom in 100mm.

Figure 7-30 3rd frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera tilt down 2 degrees and zoom in 100mm.

7.1.6.7. Case 2: Tilt Down 5 Degrees, and Zoom In 100mm

In this case, we test more difficult synthesis condition that the virtual camera has a tilt-down of 5 degrees and a zoom-in of 100mm. The geometric relationship between the four reference cameras and the synthesized virtual camera is shown in Figure 7-31. As we can see, the synthesized result of the adaptive blending technique in [29] cannot eliminate the wrongly warped pixels due to texture-depth misalignment. The synthesized result in the toy bear's eyes corrupted by the toy bear's skin due to texture-texture misalignment, as shown in Figure 7-32(a). By using the proposed multi-view blending algorithm with four input cameras, we achieve better subjective synthesized picture quality by removing the wrongly warped pixels due to texture-depth misalignment and choosing the best matched color pixels from the four warped virtual views, as shown in Figure 7-32(c), Figure 7-33(c), and Figure 7-34(c).

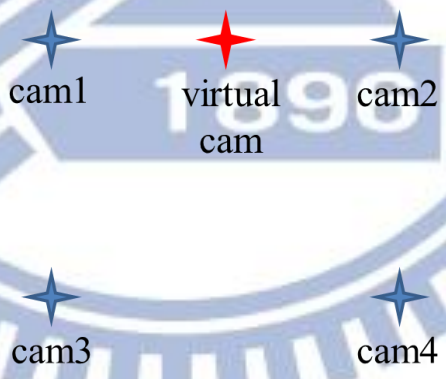


Figure 7-31 Geometric relationship between the four reference cameras and the synthesized virtual camera.

(a) (b)

(c)

Figure 7-32 1st frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera tilt down 5 degrees and zoom in 100mm.

(a) (b)

(c)

Figure 7-33 2nd frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera tilt down 5 degrees and zoom in 100mm.

(a) (b)

(c)

Figure 7-34 3rd frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera tilt down 5 degrees and zoom in 100mm.

7.1.6.8. Case 3: Zoom In 200mm

In this case, the virtual camera zooms in 200mm. Figure 7-35 shows the geometric relationship between the four reference cameras and the synthesized virtual camera. As our analysis in the previous section indicate that the blending result of the previous adaptive blending technique cannot reduce the wrongly warped pixels due to texture-depth misalignment. It also cannot robustly remove the texture-texture misalignment, so the synthesized quality in the toy bear's eyes and tie is poor subjectively, as shown in Figure 7-36(a), Figure 7-37(a), and Figure 7-38(a). By using the proposed multi-view blending algorithm, we can reduce the artifacts of texture-depth misalignment and texture-texture misalignment, as shown in Figure 7-36(c), Figure 7-37(c), and Figure 7-38(c). However, the blending results on the edge of the table have some fragment pixels in Figure 7-36(c). These wrongly rendered fragment pixels can be improved by considering the texture edge in the future.

Figure 7-35 Geometric relationship between the four reference cameras and the synthesized virtual camera.

(a)

(b)

(c)

Figure 7-36 1st frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera zoom in 200mm.

(a)

(b)

(c)

Figure 7-37 2nd frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera zoom in 200mm.

Figure 7-38 3rd frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera zoom in 200mm.

7.1.6.9. Case 4: Zoom In 400mm

In this experiment, we increase the amount of camera zoom-in to 400mm. Figure 7-39 shows the geometric relationship between the four reference cameras and the synthesized virtual camera. Compared with the blending results of Section 7.1.6.8, the synthesized results of the adaptive blending technique in [29] become worse, as shown in Figure 7-40(a), Figure 7-41(a), and Figure 7-42(a). The virtual camera with zoom in 400mm has large disocclusion regions. Hence, the synthesized results are much worse than the zoom in 200mm case does. Our multi-view blending algorithm retains most of the synthesized pixels correctly, even in this extreme synthesis condition, as shown in Figure 7-40(c), 7-41(c), and Figure 7-42(c).

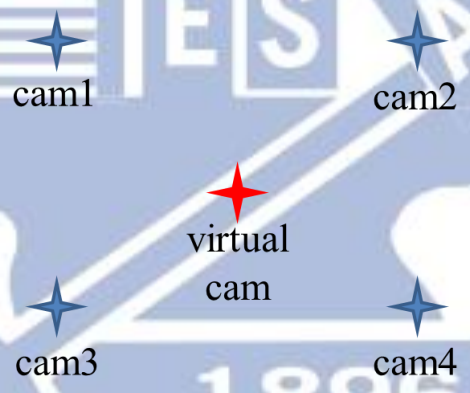


Figure 7-39 Geometric relationship between the four reference cameras and the synthesized virtual camera.

Figure 7-40 1st frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera zoom in 400mm.

Figure 7-41 2nd frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera zoom in 400mm.

Figure 7-42 3rd frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera zoom in 400mm.

7.1.6.10. Case 5: Zoom Out 400mm

The last experiment of the first test sequence synthesizes the virtual view with virtual camera zooming out 400mm. The geometric relationship between the four reference cameras and the synthesized virtual camera is shown in Figure 7-43. Compared with the adaptive blending technique in [29], our multi-view blending algorithm removes the wrongly warped pixels near the boundaries of the toy bear and maintains the sharpness of depth smooth region and object edges, as shown in Figure 7-44, Figure 7-45, Figure 7-46, and Figure 7-47. The synthesized results near the table edge may be further improved by using the texture object edge to be the drainage divide when choosing the best matched color pixel in our multi-view blending part 3.

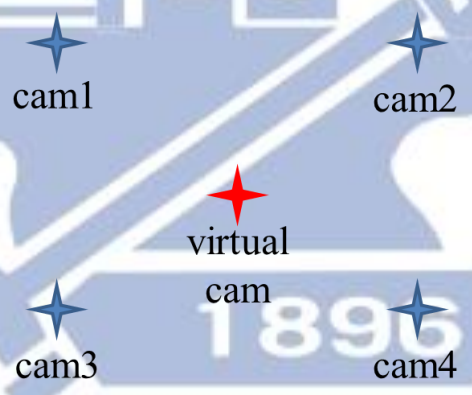


Figure 7-43 geometric relationship between the four reference cameras and the synthesized virtual camera.

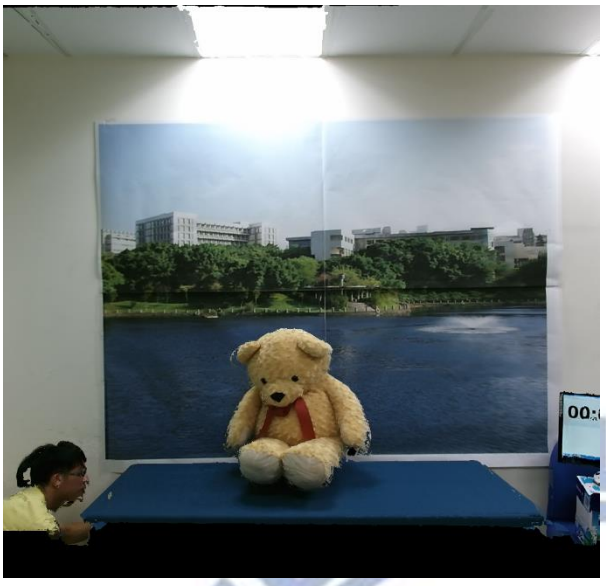
Figure 7-44 1st frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera zoom out 400mm.

(a) (b)

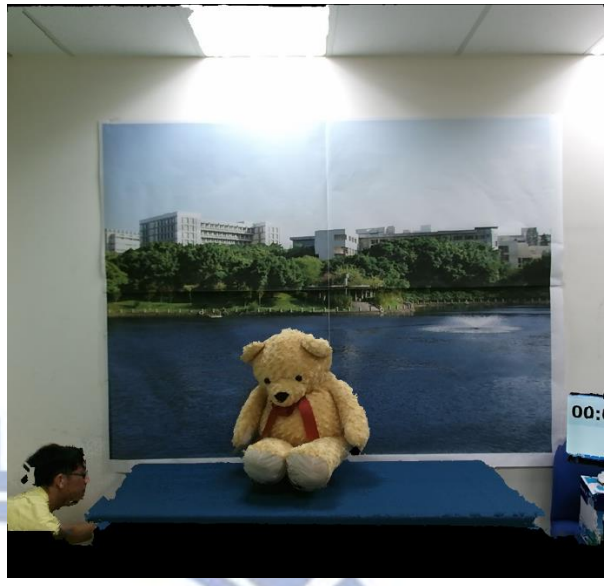
(c)

Figure 7-45 2nd frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera zoom out 400mm.

Figure 7-46 3rd frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera zoom out 400mm.



(a)



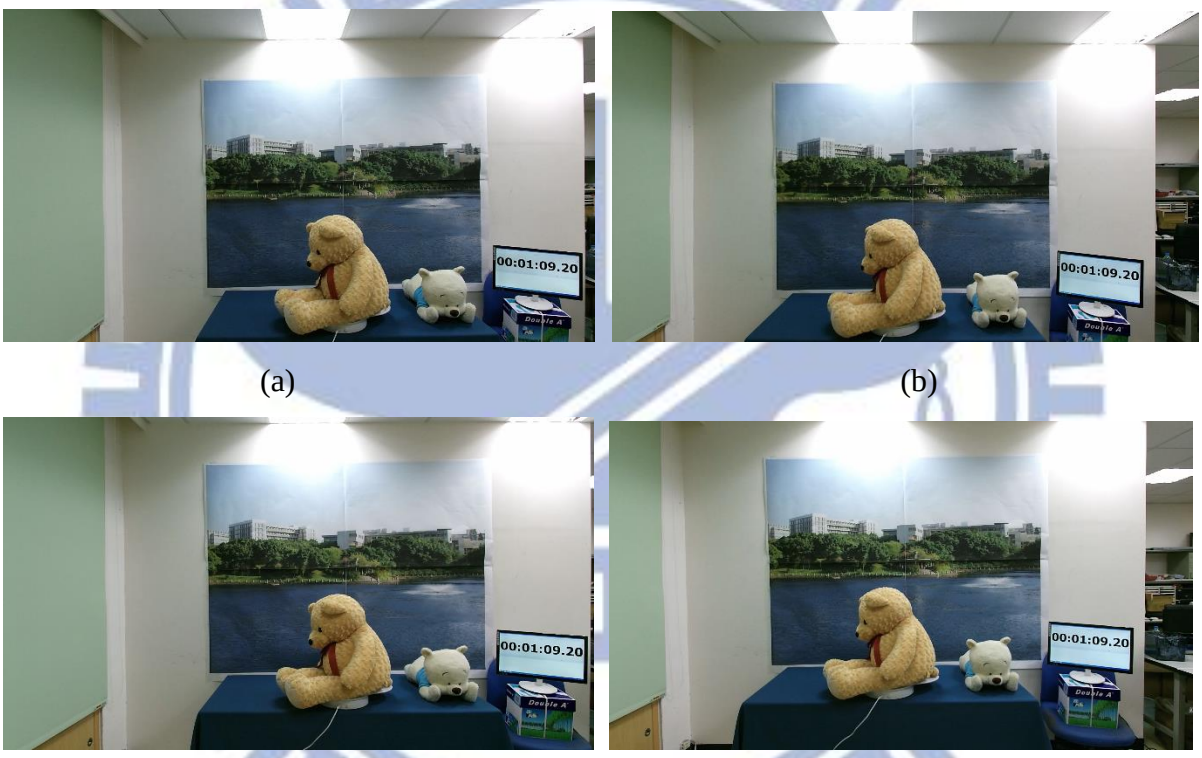
(b)

(c)

Figure 7-47 4th frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera zoom out 400mm.

7.2. Second Test Sequence

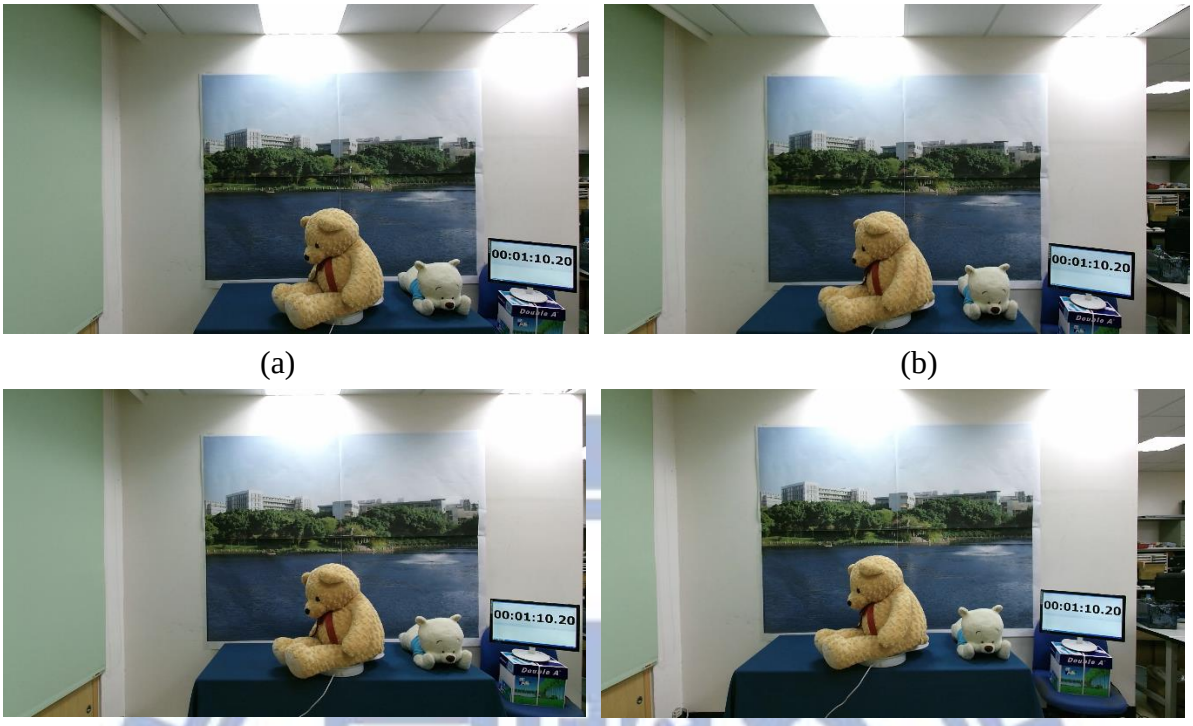
The second test sequence is an brown toy bear rotating and a white toy bear which is stationary on the table without moving. It consists of 49 frames. We record this test sequence after the first test sequence, therefore the camera intrinsic and extrinsic parameters in this test sequence is the same as the first test sequence. We list 20th ~ 23th frame of the sequence captured by the associated four Kinect v2 sensors in Figure 7-48, Figure 7-49, Figure 7-50, and Figure 7-51. The brown toy bear is the only moving object in this test sequence.



(a) (b)

(c) (d)

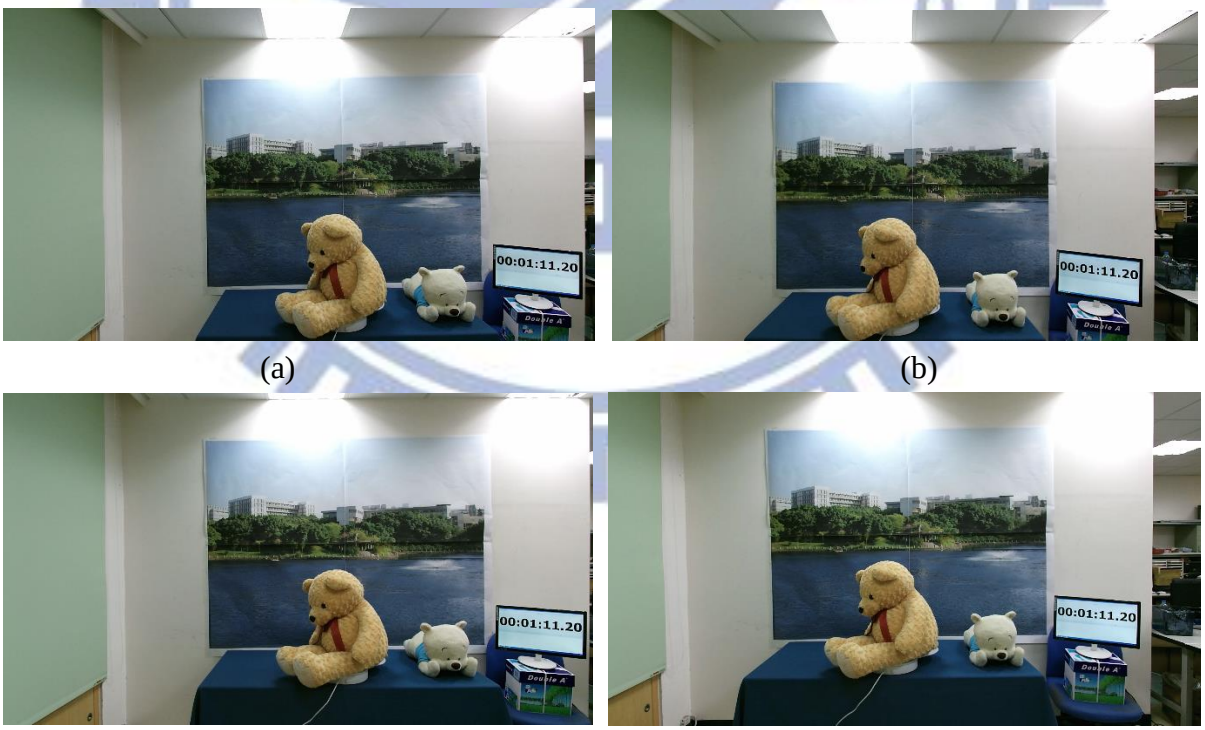
Figure 7-48 20st frame of the sequences captured from **(a)** camera 1 **(b)** camera 2 **(c)** camera 3 **(d)** camera 4.



(a) (b)

(c) (d)

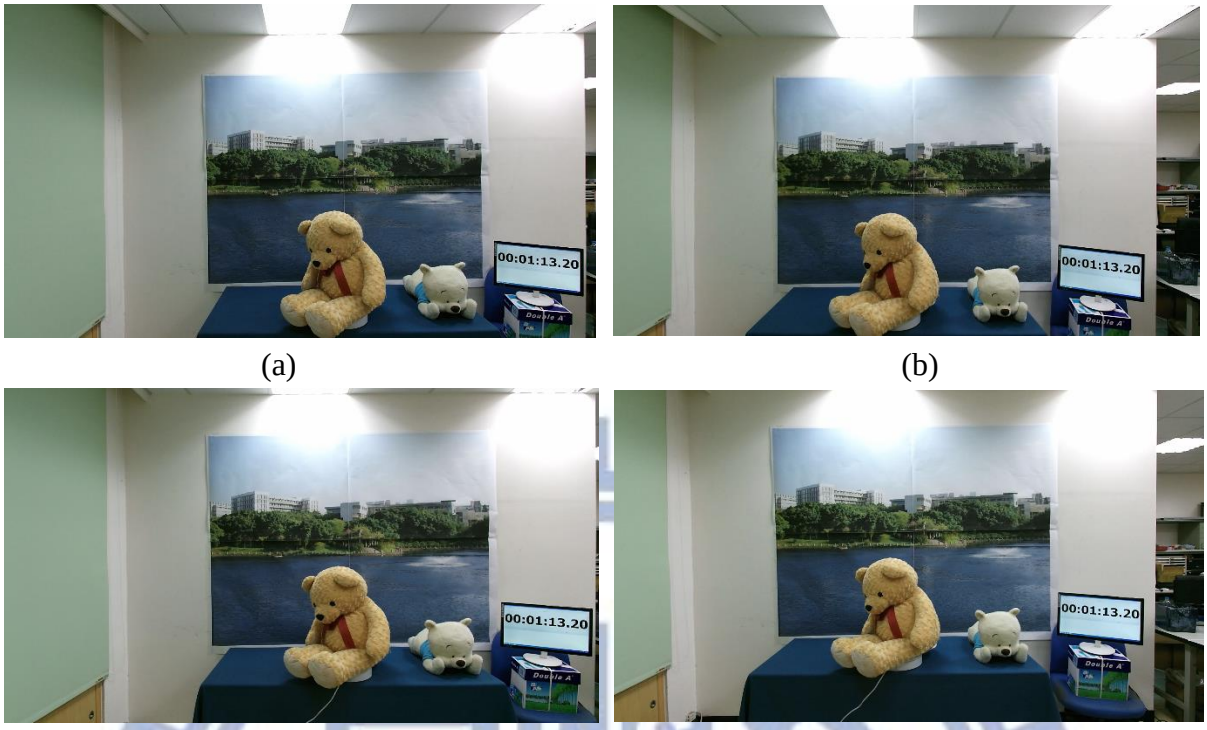
Figure 7-49 21st frame of the sequences captured from **(a)** camera 1 **(b)** camera 2 **(c)** camera 3 **(d)** camera 4.



(a) (b)

(c) (d)

Figure 7-50 22st frame of the sequences captured from **(a)** camera 1 **(b)** camera 2 **(c)** camera 3 **(d)** camera 4.



(a) (b)

(c) (d)

Figure 7-51 23st frame of the sequences captured from **(a)** camera 1 **(b)** camera 2 **(c)** camera 3 **(d)** camera 4.

7.2.1. Multi-camera Image Rectification

We use the same IRTM as listed in (89) ~ (92) to rectify the four sequences captured from the four Kinect v2 sensors. Figure 7-52 and Figure 7-53 show the un-rectified and rectified color image of the four reference views.

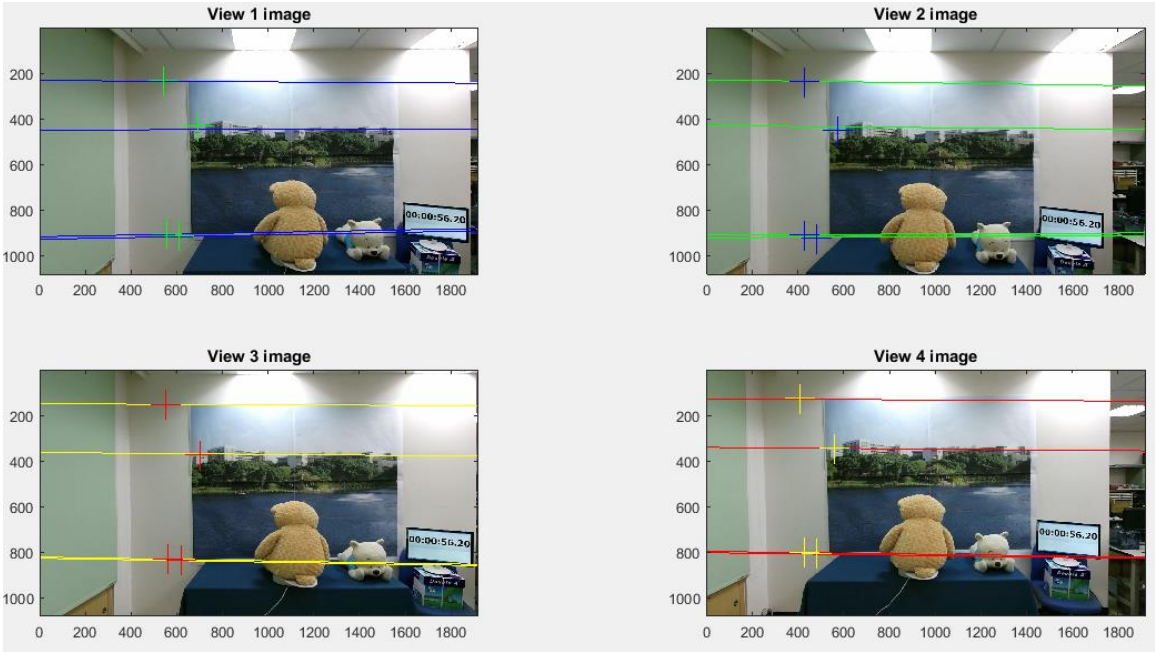


Figure 7-52 Un-rectified color image of the four reference views.

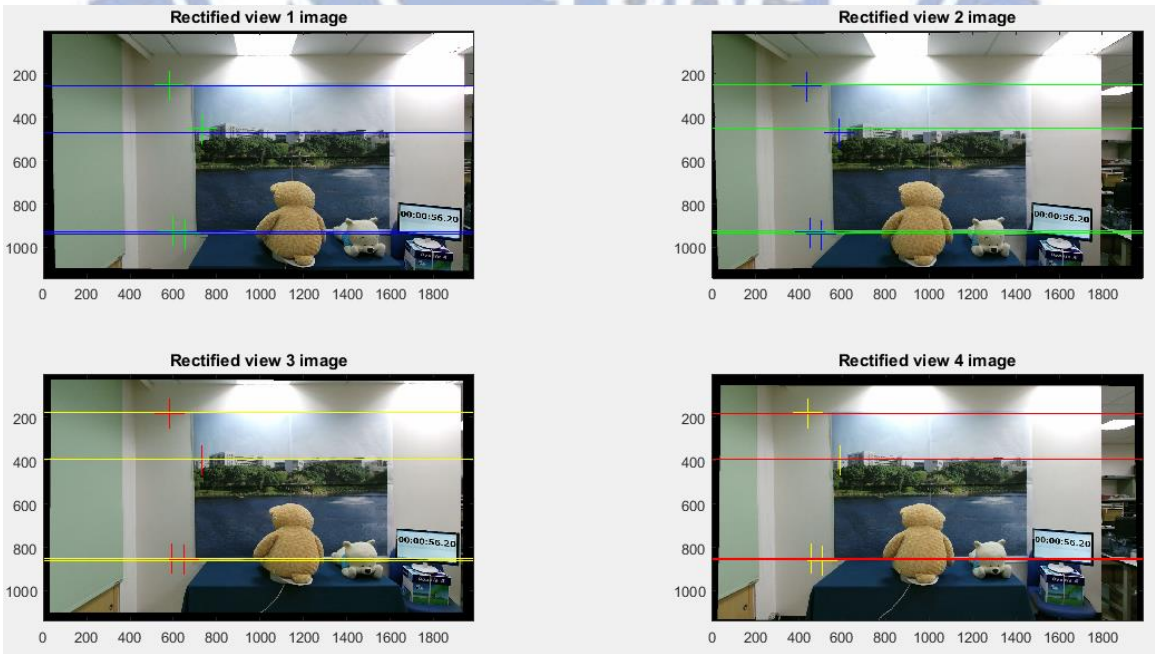


Figure 7-53 Rectified color image of the four reference views.

7.2.2. Depth Map Refinement for Kinect v2 Before 3D Warping

Figure 7-54 and Figure 7-55 show the raw depth map and the refined depth map of 20th frame. The original depth map is in grayscale. However, it is too dark to see. Hence, for more clear reading, we draw it in pseudo color format.

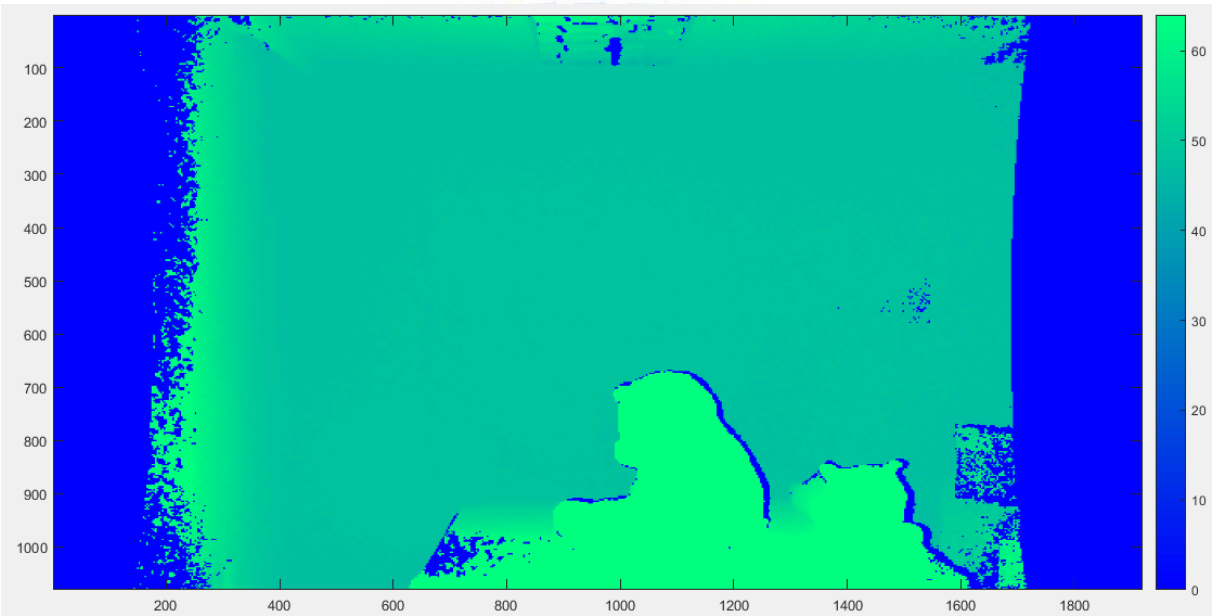


Figure 7-54 Raw depth map.

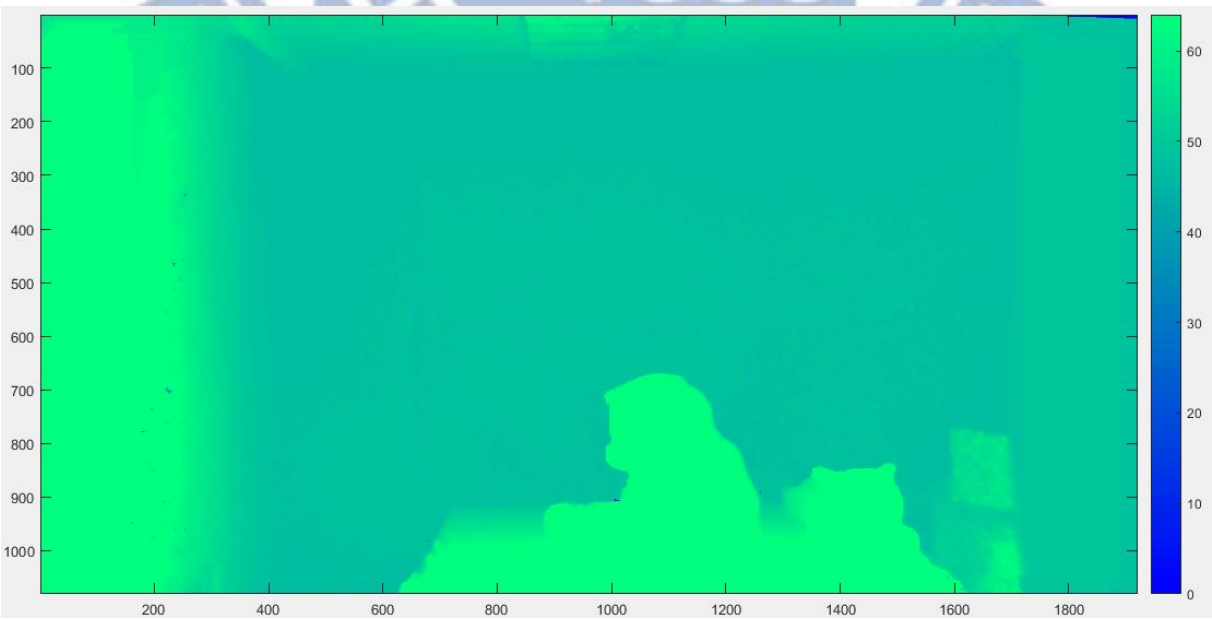


Figure 7-55 Refined depth map.

7.2.3. 3D Warping Before Wide Angle View Merging

Figure 7-56 shows geometric relationship between the four reference cameras and the synthesized virtual camera. Figure 7-57 and Figure 7-58 show the original four reference views and the four warped virtual views by applying 3D warping proposed in [29].

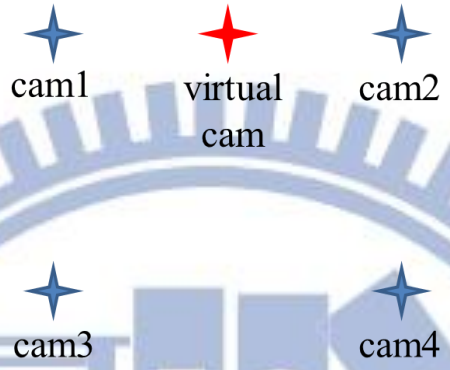


Figure 7-56 Geometric relationship between the four reference cameras and the synthesized virtual camera.

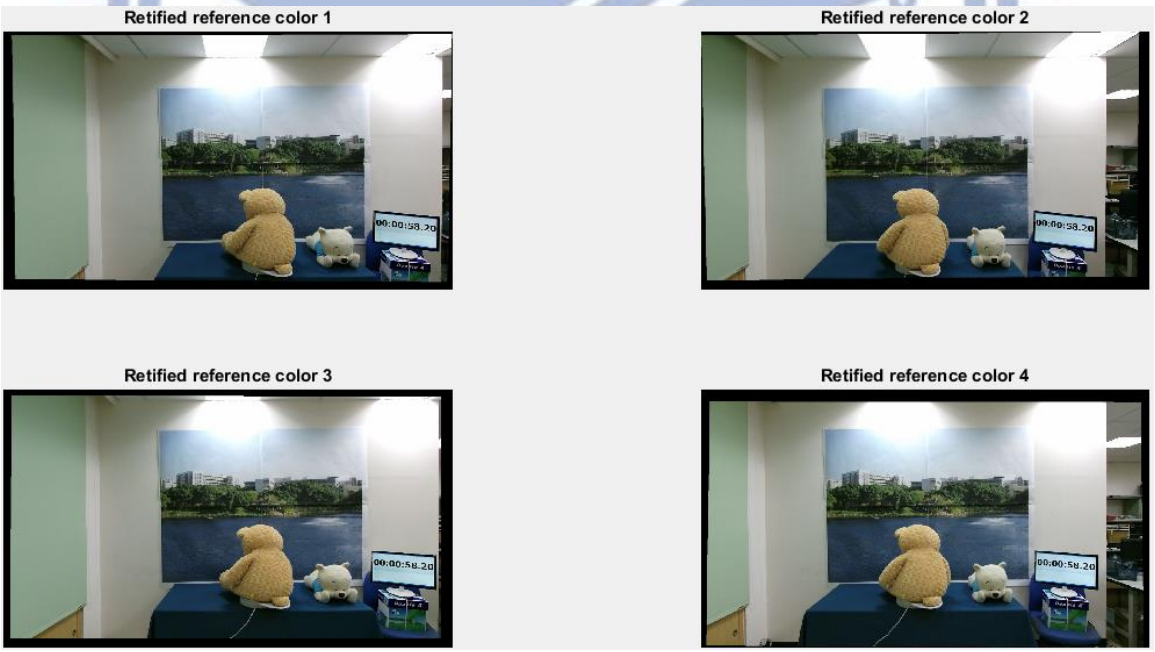


Figure 7-57 Original reference views of the four cameras.

Figure 7-58 Warped views of the four reference views with virtual camera tilt down 5 degrees and zoom in 100mm.

7.2.4. Multi-view Blending

In bear rotation test sequence, we do three different experiments with different virtual camera position. We only show 20th ~ 23th frames of the synthesized results.

7.2.4.1. Case 1:Title Down 5 Degrees, and Zoom In 100mm

The first experiment in this test sequence is that the virtual camera tilts down 5 degrees and zooms in 100mm. Figure 7-59 illustrates the geometric relationship between the four reference cameras and the synthesized virtual camera. Compared with the previous adaptive blending technique in [29], the synthesized result of our multi-view blending algorithm with two input cameras can eliminate the wrongly warped pixels due to texture-depth misalignment, as shown in Figure 7-60(a)(b), Figure 7-61(a)(b), Figure 7-62(a)(b), and Figure 7-63(a)(b). Moreover, as we described in Section 6.2, the texture-texture misalignment causes

the synthesized results having ghost effect near the depth edge, as shown in Figure 6-7. By using the proposed multi-view blending method with four input cameras, we not only remove the artifact of texture-depth misalignment and texture-texture misalignment, but also fills in large occlusion regions at the bottom of the synthesized images, as shown in Figure 7-60(c), Figure 7-61(c), Figure 7-62(c), and Figure 7-63(c).

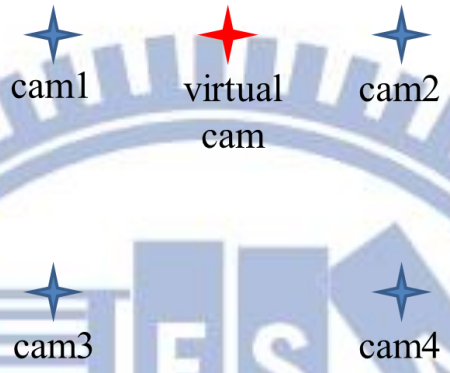


Figure 7-59 Geometric relationship between the four reference cameras and the synthesized virtual camera.

(a) (b)

(c)

Figure 7-60 20th frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera tilt down 5 degrees and zoom in 100mm.

(a) (b)

(c)

Figure 7-61 21th frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera tilt down 5 degrees and zoom in 100mm.

Figure 7-62 22th frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera tilt down 5 degrees and zoom in 100mm.

(a) (b)

(c)

Figure 7-63 23th frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera tilt down 5 degrees and zoom in 100mm.

7.2.4.2. Case 2: Zoom In 400mm

In this time we synthesize the virtual view with a zoom in of 400mm. Figure 7-64 shows the geometric relationship between the four reference cameras and the synthesized virtual camera. The wrongly warped pixel in the synthesized results of the adaptive blending technique in [29] is obvious, as shown in Figure 7-65(a), Figure 7-66(a), Figure 7-67(a), and Figure 7-68(a). The proposed multi-view blending algorithm in this thesis can successfully reduce the wrongly warped pixels near the depth edge, as shown in Figure 7-65(c), Figure 7-66(c), Figure 7-67(c), and Figure 7-68(c).

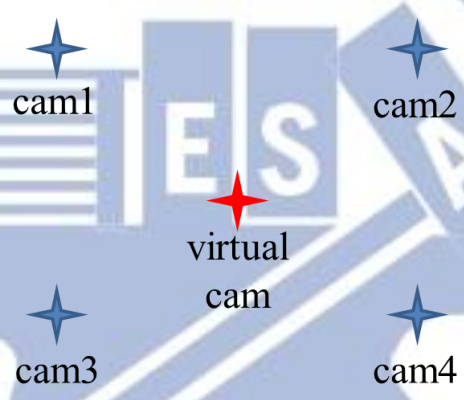


Figure 7-64 Geometric relationship between the four reference cameras and the synthesized virtual camera.

Figure 7-65 20th frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera zoom in 400mm.

(a)

(b)

(c)

Figure 7-66 21th frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera zoom in 400mm.

Figure 7-67 22th frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera zoom in 400mm.

Figure 7-68 23th frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera zoom in 400mm.

7.2.4.3. Case 3: Zoom Out 400mm

This is the last experiment of the second test sequence. We synthesize the virtual view with a zoom out of 400mm. The geometric relationship between the four reference cameras and the synthesized virtual camera is shown in Figure 7-69. The effect of texture-texture misalignment is not significant in the zooming out case. Compared with the adaptive blending method [29], the proposed multi-view blending algorithm can almost remove the wrongly warped pixels due to texture-depth misalignment. Furthermore, we keeps the synthesized quality well in the bottom disocclusion regions, as shown in Figure 7-70(c), Figure 7-71(c), Figure 7-72(c), and Figure 7-73(c).

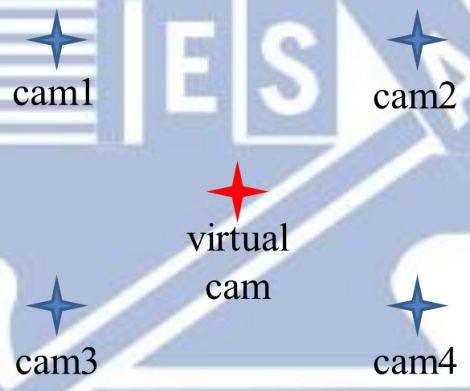


Figure 7-69 Geometric relationship between the four reference cameras and the synthesized virtual camera.

(a) (b)

(c)

Figure 7-70 20th frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera zoom out 400mm.

(a) (b)

(c)

Figure 7-71 21th frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera zoom out 400mm.

Figure 7-72 22th frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera zoom out 400mm.

Figure 7-73 23th frame of final blending results by using **(a)** adaptive blending with upper two cameras [29] **(b)** the proposed multi-view blending algorithm with upper two cameras **(c)** the proposed multi-view blending algorithm with four cameras. The virtual camera zoom out 400mm.

8. Conclusions

In this thesis, we have some challenges. The first challenge is how to synchronize the four Kinect v2 sensors in capturing pictures to zeros time skew. Second, we need to develop a multi-view blending algorithm which can produce a good synthesized image quality when the warped four virtual views are not perfect. This is the practical situation we meet in capturing the pictures using the commercial camera/depth devices.

We use the NTPClock.exe software [47] to calibrate the system time of the four computers connected to four Kinect v2 sensors. Solving the synchronization problem among the four Kinect v2 sensors improves the synthesized image quality. The main contribution in this thesis is that we propose a multi-view blending algorithm. It consists of three parts. The first part renders the pixels are not in the neighborhood marked depth edge. The second part checks the depth similarity between the virtual depth and the associated background depth. It also checks the color similarity of the virtual color pixels candidates. If the variance and mean of the most possible virtual color pixels satisfy certain criteria, then we will fill the pixel by the dominant virtual view; otherwise, we will iteratively render the pixel by choosing the color pixel of the four virtual views, which is closest to the color pixels of its 8-neighbors. The third part of the proposed multi-view blending algorithm iteratively renders the pixel by choosing the color pixel which is the most closest to the color pixels of its 8-neighbors.

In the practical cases where the video and depth maps are captured by the consuming electronical devices. The depth map contains noise, holes and occlusion regions. Compared to the adaptive blending method in [29], the proposed multi-view blending algorithm achieves better subjective synthesized image quality in these real cases.

References

[1] C. Zhu , Y. Zhao , L. Yu and M. Tanimoto, 3D-TVSystem With Depth-Image-Based Rendering: Architectures, Techniques and Challenges, 2013 :Springer

[2] R. Hartley and A. Zisserman, Multiple View Geometry in Computer Vision, Second Edition, Cambridge University Press, March 2004.

[3] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330-1334, 2000.

[4] Z. Zhang, "Flexible Camera Calibration By Viewing a Plane From Unknown Orientations," *Proc. ICCV*, pp. 666-673, 1999.

[5] J. Heikkila, and O.Silvén "A Four step Camera calibration Procedure with Implicit Image Correction," in *Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 97)*, San Juan, Puerto Rico, pp. 1106-1112, 1997.

[6] D. H. C, et al, "Joint Depth and Color Camera Calibration with Distortion Correction," *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 34, no.10, pp. 2058-2064, Oct 2012.

[7] E. Auvinet, et al, "Multiple Depth Cameras Calibration and Body Volume Reconstruction for Gait Analysis," *The 11th International Conference on Information Sciences, Signal Processing and Their Applications: Main Tracks*, pp. 478-483, 2012.

[8] B. Li, et al, "A Multiple-Camera System Calibration Toolbox Using A Feature Descriptor-Based Calibration Pattern," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1301-1307, November, 2013.

[9] D. H. C, et al, "Accurate and Practical Calibration of a Depth and Color Camera Pair," *Proceedings of the 14th international conference on Computer analysis of images and patterns* (*CAIP'11) - Volume Part II*, pp. 437-445, 2011.

[10] E. Levine, et all, “High-precision Multi-View Camera Calibration Using a Rotating Stage,” *IEEE International Conference on Image Processing (ICIP)*, pp. 1175-1179, September 2016.

[11] Q. Yao, et al, “Robust Moving Camera Calibration for Synthesizing Free Viewpoint Soccer Video,” *IEEE International Conference on Image Processing (ICIP)*, pp. 1185-1189, September 2016.

[12] Camera Calibration Toolbox for Matlab. Available software at http://www.vision.caltech.edu/bouguetj/calib_doc/

[13] A. Fusiello, E. Trucco, A. Verri, “A compact algorithm for rectification of stereo pairs,” *Machine Vision and Applications*, 12, pp. 16–22, 2000. Received: 25, Feb. 1999, Accepted: 2 March 2000.

[14] Z. Yang, et al, “A Rectification Algorithm for Un-calibrated Multi-view, Images Based on SIT Features,” *IEEE International Conference on Audio Language and Image Processing (ICALIP)*, pp. 143-147, November. 2010.

[15] J. Yang, et al, “Multiview Image Rectification Algorithm for Parallel Camera Arrays,” *Journal of Electronic Imaging*, 2014.

[16] J. Yang, F. Guo, H. Wang, and Z. Ding, "A multi-view image rectification algorithm for matrix camera arrangement", *Artificial Intelligence Research*, Vol. 3, No. 1, pp18~29, November 2013.

[17] Picture source Prof. D. Lowe, Computer Vision, UBC, CA.

[18] R. C. Gonzalez and R. E. Woods, Digital Image Processing, 3rd Edition, Prentice Hall, 2008.

[19] F. M. Porikli, “Inter-camera color calibration by correlation model function,” *IEEE International Conference on Image Processing*, pp. 133–136, 2003.

[20] J. R. Silva, T. T. Santos, and C. H. Morimoto, “Virtual Reality in Brazil: Automatic camera control in virtual environments augmented using multiple sparse videos,” [*Compuetr and Graphics*, vol. 35, iss.2, April 2011.

[21] K. J. Oh, et al, "Virtual view synthesis method and self−evaluation metrics for free viewpoint television and 3D video," *International Journal of Imaging Systems and Technology*, vol. 20, pp. 378-390, 2010.

[22] D. Tian, P.-L. Lai, P. Lopez, and C. Gomila, "View synthesis techniques for 3D video," Applications of digital image processing XXXII, *proceedings of the SPIE*, vol. 7443, pp. 74430T-74430T, 2009.

[23] D.-H. Li, H.-M. Hang, and Y.-L. Liu, "Virtual view synthesis using backward depth warping algorithm," in *Picture Coding Symposium* (*PCS*), pp. 205-208, 2013.

[24] Y.-L. Liu, Wide-Angle Virtual View Synthesis with Depth-Based Background Modeling, MS thesis, National Chiao Tung University, July, 2014.

[25] K. Müller, et al, “View synthesis for advanced 3D video systems,” *EURASIP Journal on Image and Video Processing*, vol 2008, Article ID 438148, 2008.

[26] Y. Mori, et al, “View generation with 3D warping using depth information for FTV,” *Sig Processing: Image Commun*, 24(1–2):65–72, 2009.

[27] S. Zinger, et al, “Free-viewpoint depth image based rendering,” *J Vis Commun and Image Represent*, 21:533–541, 2010.

[28] M. Doman, M. Gotfryd, and K. Wegner, “View synthesis for multiview video transmission,” *International conference on image processing, computer vision, and pattern recognition*, pp 13–16, Las Vegas, USA, July 2009.

[29] T.-C. Lee, Virtual View Synthesis Quality Refinement, MS thesis, National Chiao Tung University, July, 2016.

[30] M. Bertalmio, et al, “Image inpainting,” *Proceedings of ACM conference on computer*

graphics (SIGGRAPH), pp 417–424, New Orleans, LU, July 2000.

[31] A. Criminisi, P. Perez, and K. Toyama, “Region filling and object removal by exemplar-based inpainting,” *IEEE Trans Image Process*, 13(9):1200–1212, 2004.

[32] A. Telea, “An image inpainting technique based on the fast marching method,” *J Graph Tools*, 9(1):25–36, 2004.

[33] K. Oh, S. Yea, and Y. Ho, “Hole-filling method using depth based in-painting for view synthesis in free viewpoint television (FTV) and 3D video,” *Proceedings of the picture coding symposium (PCS)*, Chicago, pp 233–36, 2009.

[34] “Deep learning based image view synthesis for free view 3D video,” *IS&T International symposium on Electronic Imaging (EI)*, Feb 2016.

[35] P. Ndjiki-Nya, et al., "Depth image-based rendering with advanced texture synthesis for 3-D video," *IEEE Transactions on Multimedia*, vol. 13, pp. 453-465, 2011.

[36] S. S. Yoon, et al, "Inter-view consistent hole filling in view extrapolation for multi-view image generation," *IEEE International Conference on Image Processing (ICIP)*, pp. 2883-2887, 2014.

[37] M. Schmeing and X. Jiang, "Depth image based rendering: a faithful approach for the disocclusion problem," *Proceedings of 3DTV conference*, pp. 1-4, 2010.

[38] Kinect V1, https://zh.wikipedia.org/wiki/Kinect

[39] Time-of-flight camera, http://en.wikipedia.org/wiki/Time-of-flight_camera

[40] Kinect V2, https://en.wikipedia.org/wiki/Kinect_for_Xbox_One

[41] Kinect v2 teardown, http://nuigroup.com/forums/viewthread/14981/#72710

[42] S.-P. Weng, Depth Map Enhancement based on Its Associated High-Resolution RGB Video, MS thesis, National Chiao Tung University, July, 2015.

[43] K.-Y. Lin, Depth Map Enhancement on RGB-D Video Captured by Kinect v2, MS thesis, National Chiao Tung Unversity, July, 2017.

[44] I. Daribo, and B. Pesquet-Popescu, “Depth-aided image inpainting for novel view

synthesis," *Proceedings of IEEE international workshop on multimedia signal processing (MMSP)*, 2010.

[45] Li. Renxiang, et al, "A New Three-Step Search Algorithm for Block Motion Estimation", *IEEE Trans. Circuits And Systems For Video Technology*, vol 4., no. 4, pp. 438-442, August 1994.

[46] C.-L. Chien, A. Wang, S.-P. Weng, and H.-M. Hang, "A Method of Disocclusion Detection and Filling for the Depth Map Generated by Kinect Sensor," *2015 Conf. on Computer Vision, Graphics, and Image Processing*, Yilan, Taiwan, Aug. 17-19, 2015.

[47] NTP clock software.Available at http://www.stdtime.gov.tw/chinese/EXE/NTPClock.exe

[48] M. Kowalski, et al, "Livescan3D: A Fast and Inexpensive 3D Data Acquisition System for Multiple Kinect v2 Sensors," 2015 *International Conference on 3D Vision*, vol. 00, no. , pp. 318-325, 2015.