Towards high-quality 3D telepresence with commodity RGBD camera

Zhao, Mengyao

2018

Zhao, M. (2018). Towards high-quality 3D telepresence with commodity RGBD camera.
Doctoral thesis, Nanyang Technological University, Singapore.

<http://hdl.handle.net/10356/73161>

<https://doi.org/10.32657/10356/73161>

NANYANG TECHNOLOGICAL UNIVERSITY

Towards High-quality 3D Telepresence with Commodity RGBD Camera

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Doctor of Philosophy

by

Zhao Mengyao

2016

Abstract

3D telepresence aims at providing remote participants to have the perception of being present at the same physical space, which cannot be achieved by any 2D teleconference system. The success of 3D telepresence will greatly enhance communications, allowing much better user experience, which could stimulate many applications including teleconference, telesurgery, remote education, etc. Despite years of study, 3D telepresence research still faces many challenges such as high system cost, hard to achieve real-time performance with consumer-level hardware and high computation requirement, costly to obtain depth data, hard to extracting 3D people in real-time with high quality and difficult for 3D scene replacement and composition.

The emerging of consumer-grade range cameras, such as Microsoft Kinect, which provides convenient and low-cost acquisition of 3D depth in real-time, accelerate many multimedia applications. In this thesis, we make a few attempts, aim at improving the quality of 3D telepresence with commodity RGBD camera. First, considering that the raw depth data of commodity depth camera is highly noisy and error-prone, we carefully study the error patterns of Kinect and propose a multi-scale direction-aware filtering method to combat Kinect noise. We have also implemented the proposed method in CUDA to achieve real-time performance. Experimental results show that our method outperforms the popular bilater filter.

Second, we consider the problem of real-time extracting dynamic foreground person from RGB-D video, which is a common task in 3D telepresence. Existing methods are hard to ensure real time, high quality and temporal coherence at the same time. We propose a foreground extraction framework which nicely integrates many existing techniques including background subtraction, depth hole filing and 3D matting. We also take advantage of various CUDA strategies and spatial data structures to improve the speed. Experimental results show that, compared with state-of-the-art methods, our proposed method can extract stable foreground objects with

higher visual quality as well as better temporal coherence, while still achieving real-time performance.

Third, we further consider another challenging problem in 3D telepresence, i.e. given a RGBD video, we want to replace the local 3D background scene by a target 3D scene. There are a lot of issues such as the mismatch between the local scene and the target scene, the range of motion in different scenes, the collision problem, etc. We propose a novel scene replacement system that consists of multi-stages of processing including foreground extraction, scene adjustment, scene analysis, scene suggestion, scene matching, and scene rendering. We also develop our system entirely on the GPU by parallelizing most of the computation with CUDA strategies, by which we can achieve not only good visual quality scene replacement but also real-time performance.

Acknowledgments

I would like to express my gratitude to all those who gave me the possibility to complete this report.

My most sincere thanks go to my advisor Prof. Chi-Wing Fu and Prof. Jianfei Cai. I thank them for introducing me to the wonders and frustrations of scientific research. I thank them for his guidance, encouragement and support during the development of this work. The supervision and support that they gave truly help the progression and smoothness. I have been extremely lucky to have two supervisors who cared so much about my work, and who responded to my questions and queries so promptly.

I also would like to express my very great appreciation to Prof. Cham Tat-Jen for their constructive suggestions during our weekly meetings.

I thank my colleague Fuwen Tan for his help in our team-working. I also wish to thank all my friends of the BeingThere Centre at Institute of Multimedia Innovation who supported me a lot by providing valuable feedback in many fruitful discussions. This report would not be possible in this form without the support and collaboration of several friends, in particular Mr. Li Bingbing, Mr. Ren Jianfeng, Mr. Xu Di, Mr. Chen Chongyu, Mr. Deng Teng, Dr. Cdric Fleury, Mr. Lai Chi-Fu William, Mr. Guo Yu.

I would like to thank my parents and husband for their trust and encouragement.

This work, which is carried out at BeingThere Centre, is supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office.

Publications

Published:

Mengyao Zhao; Fuwen Tan; Chi-Wing Fu; Chi-Keung Tang; Jianfei Cai; Tat Jen Cham, "High-quality Kinect depth filtering for real-time 3D telepresence," *Multimedia and Expo (ICME), 2013 IEEE International Conference on*, pp.1-6, 15-19 July 2013

M. Zhao, C. W. Fu, J. Cai and T. J. Cham, "Real-Time and Temporal-Coherent Foreground Extraction With Commodity RGBD Camera," in *IEEE Journal of Selected Topics in Signal Processing*, vol. 9, no. 3, pp. 449-461, April 2015.

In Preparation:

Mengyao Zhao, Jianfei Cai, Chi-Wing Fu, Tat Jen Cham. Automatic 3D Scene Replacement in real-time for 3D Telepresence. 2016.

Contents

Abstract	i
Acknowledgments	iii
Publications	iv
List of Figures	ix
List of Tables	xiii
1 Introduction	1
1.1 Research Motivation	1
1.1.1 3D Telepresence	1
1.1.2 Directions of 3D Telepresence	2
1.1.3 Challenges	4
1.2 Research Objective	5
1.3 Report Organization	6
2 Literature Review	7
2.1 History of Telepresence	7
2.2 Emerging 3D Input Devices	10
2.3 Kinect Depth Denoising and Filtering	12
2.3.1 Telepresence Applications with Kinect	12
2.3.2 Kinect Depth Filtering	12
2.3.3 Depth Inpainting	15
2.3.4 Scale-space and Multi-scale Analysis	16
2.4 Foreground Extraction Methods	18
2.4.1 Interactive Foreground Extraction	18

2.4.2	Automatic Foreground Extraction	20
2.4.3	Real-time Foreground Extraction	21
2.4.4	Real-time Foreground Extraction with RGBD videos	22
2.5	Summary	23
3	High-quality Kinect Depth Filtering for Real-time 3D Telepresence	24
3.1	Introduction	24
3.2	Our Approach	26
3.2.1	Kinect Raw Depth Data	26
3.2.2	Multi-scale Filtering	27
3.2.3	Direction-aware Filtering	29
3.3	Processing Pipeline	29
3.4	Algorithm	30
3.4.1	Multi-scale Analysis	30
3.4.2	Direction-aware Analysis	30
3.4.3	Data Filtering	31
3.4.4	CUDA Implementation on GPU	32
3.5	Experiments and Results	32
3.5.1	Quantitative Comparison	32
3.5.2	Visual Comparison	33
3.5.3	Performance Evaluation	34
3.6	Summary	35
4	Real-time and Temporal-coherent Foreground Extraction with Commodity RGBD Camera	38
4.1	Introduction	39
4.2	Overview	40
4.2.1	Background Modeling	41
4.2.2	Data Preprocessing	41
4.2.3	Trimap Generation	42
4.2.4	Temporal Matting	42
4.3	Preprocessing	43

4.3.1	Shadow Detection	43
4.3.2	Adaptive Temporal Hole-Filling	45
4.4	Automatic Trimap Generation	46
4.4.1	Background Subtraction	46
4.4.2	Adaptive Mask Generation	46
4.4.3	Morphological Operation	48
4.5	Temporal Matting	49
4.5.1	Closed-form Matting	49
4.5.2	Our Approach: Construct the Laplacian matrix	50
4.5.3	Our Approach: Solving for the Alpha Matte	53
4.6	Experiments and Results	53
4.6.1	Implementation Details	53
4.6.2	Foreground Extraction Results	54
4.6.3	Experiment: Time Performance	54
4.6.4	Experiment: Compare with other methods	58
4.6.5	Experiment: Adaptive mask generation method	59
4.6.6	Experiment: Robustness and Stability	59
4.7	Summary	59
5	Automatic 3D Scene Replacement for 3D Telepresence	61
5.1	Introduction	62
5.2	Related Work	63
5.3	Overview	66
5.3.1	Foreground Extraction	66
5.3.2	Scene Adjustment	67
5.3.3	Scene Analysis	67
5.3.4	Scene Suggestion	68
5.3.5	Scene Matching	68
5.3.6	Scene Rendering	68
5.3.7	Offline Analysis	69
5.4	Our Approach: Scene Adjustment	69

5.5	Our Approach: Scene Analysis	72
5.6	Our Approach: Scene Suggestion	75
5.7	Our Approach: Scene Matching	77
5.8	Our Approach: Scene Rendering	78
5.9	Results	79
5.9.1	Implementation Details	79
5.9.2	Scene Replacement Results	80
5.9.3	Time Performance	80
5.9.4	Experiments	82
5.10	Summary	82
6	Conclusion and Future Work	84
6.1	Conclusion	84
6.2	Future Work	85
6.2.1	Multi-view 3D Foreground Extraction	86
6.2.2	Interactive Background Replacement	86
	Bibliography	89

List of Figures

1.1	Illustration of 3D telepresence where remote collaborators can communicate with each other as if they are co-located at the same place (adapted from [86]).	2
1.2	Various 3D telepresence systems (adapted from [86]).	3
2.1	The office of future [58].	8
2.2	The blue-c system [23].	9
2.3	Principle of time-of-flight camera. [86].	10
2.4	Left image is time-of-flight camera: D-IMager [53]; right image is 3D laser scanner [86].	10
2.5	Left image is Microsoft Kinect [44]; right image is PrimeSense Carmine 3D sensor [57].	11
2.6	Gaussian filter and bilateral filter. Left image is gaussian filter; right image is bilateral filter [86].	13
2.7	Comparison of gaussian filter and bilateral filter. Left image is input image; middle image is gaussian filtered result; right image is bilateral filtered result [79].	14
2.8	Left image is illustration of spatial and temporal depth characteristics. Top: Plot of the neighboring depth difference; the horizontal axis represents depth value. Bottom: depth variation with the time; the horizontal axis represents frame number.[19]; right image is scheme of NL-means strategies [4].	15
2.9	Left image is measured lateral noise (top) and axial noise (bottom) for Kinect raw depth; right image is linear and quadratic fits to the lateral and axial noise components [52].	16
2.10	Scale space representation at different scales (adapted from [89]).	17
2.11	Segmentation with user interaction (adapted from [62]).	19

2.12	Comparison of some matting and segmentation tools [62].	20
3.1	Color-coded Kinect depth data on a flat wall.	26
3.2	1D quantized depth with ground truth in green.	27
3.3	First row: two different regions in the same raw depth data: large patches (left) and a small feature (right). Second row: applying small (left) and large (right) bilateral filtering windows. Third row: our multi-scale filtering method avoids producing bumpy patches and can better preserve small surface details. Green lines show the ground truth while red pixels are the filtered result.	28
3.4	Our data processing pipeline.	29
3.5	Depth profile on the walls shown on 1st column of Fig. 3.7. (a), (b), and (c) in this figure correspond to raw data, bilateral filtering and our result, respectively.	33
3.6	Histograms of depth deviations between point clouds and the fitted surface (ground truth). The window size used in bilateral filtering is 11x11.	36
3.7	1st and 4th rows: raw data. 2nd and 5th rows: bilateral-filtered results. 3rd and 6th rows: our results. 4th, 5th, and 6th rows are zoom-in view of 1st, 2nd, and 3rd rows respectively. Red lines shown in the 1st column are the intersection lines used in depth profiling (see Fig. 3.5). Size of neighborhood used in bilateral filtering is 11x11.	37
4.1	Overview of our foreground extraction approach, which consists of four stages of GPU computation: 1) background modeling; 2) data preprocessing; 3) trimap generation; and 4) temporal matting.	41
4.2	Shadow NMD region.	44
4.3	Temporal hole-filling. Left: an input raw depth map, and right: a hole-filled depth map.	44
4.4	Adaptive mask generation: blue boxes highlight smooth and correct boundary, while red boxes highlight rough and erroneous boundary.	47
4.5	Morphological operations (erosion and dilation) to generate the trimap.	48
4.6	3D spatiotemporal neighbors (right) for temporal matting: the red dot denotes the unknown pixel.	51

4.7	Foreground extraction results. The first column shows a snapshot of each RGBD video before foreground extraction; subsequent images in each row show snapshots of the foreground extracted from the corresponding video. . . .	55
4.8	Comparing our method with background subtraction (with RGBD) [30], standard closed-form matting [35], and FreeCam [34] (from left to right). From top to bottom, the 1st and 3rd rows examine the boundary of foreground objects extracted from two different RGBD videos; the 2nd and 4th rows show the related zoom-in views. The last three rows focus on comparing the temporal coherency of foreground object boundary across consecutive video frames. . . .	56
4.9	Experiment on adaptive mask generation. From left to right: the 1st column shows the input color and depth masks; the 2nd column shows the binary masks produced by different methods (simple intersection, simple union, and our method); the 3rd and 4th columns show the trimap and foreground extraction results, respectively, produced from the related binary masks, where our adaptive mask generation method can always produce better quality results. . .	57
5.1	Overview of Offline part in our GPU-based scene replacement method: 1)scene adjustment; 2) scene analysis.	66
5.2	Overview of Online part in our GPU-based scene replacement method: 1) foreground extraction; 2) scene adjustment; 3) scene analysis; 4) scene suggestion; 5) scene matching and 6) rendering.	67
5.3	Our Floor Detection Results: a)input rgb image; b)detected planes; c)dominant planes.	70
5.4	Our Floor Adjustment Result: left: floors before adjustment; right: floors after adjustment.	72
5.5	Transformed Scenes.	73
5.6	User Scenarios: left: standing; middle: sitting; right: meeting.	73
5.7	Human Skeletal Tracking: left: standing; right: sitting (adapted from [86]). . .	74
5.8	Walkable area. Left: front view; middle: top view; right: top view with labels. .	75
5.9	Determination of walkable area using polygons: left: blue polygon represents the outline of the floor plane while red polygon represents the outline of 2D area occupied by a furniture; right: the calculated final walkable area.	75

5.10	The log-polar coordinates we use for shape context (adapted from [2]).	76
5.11	Descriptors are similar for homologous (corresponding) points and dissimilar for non-homologous points (adapted from [2]).	77
5.12	Walkable Area Matching: blue polygon represents the walkable area of <i>target scene</i> while red polygon represents the walkable area of <i>local scene</i> . Point O denotes the center of the scene; point P_i denotes a position in the <i>local scene</i> while point Q_i denotes the corresponding position of point P_i in the <i>target scene</i>	79
5.13	Problem of Partial Occlusion: a)human can be rendered in the middle of a furniture which cause partial occlusion problem; b)human can only rendered near furniture avoiding the partial occlusion problem.	80
5.14	Our Scene Replacement Results.	81

List of Tables

3.1	Comparison between CPU and GPU performance.	35
3.2	Comparison between bilateral filter and our method.	35
4.1	Time Performance of our CUDA-based pipeline.	54
5.1	Time Performance of our CUDA-based pipeline.	81

Chapter 1

Introduction

1.1 Research Motivation

3D telepresence is a next-generation multimedia application, allowing remote collaboration through a natural and immersive environment supported with a set of real-time 3D graphics. By this, participants can have a perception of being co-located, or give the appearance of being present, with others who are remote (see Fig. 1.1 and Fig. 1.2). Additionally, users may be given the ability, via telerobotics, to affect the remote location other than their true location. Applications of 3D telepresence include remote education and surgery, as well as 3D teleconferencing.

1.1.1 3D Telepresence

3D teleconference deploys greater technical sophistication and fidelity of multiple senses of human, e.g. sight, sound or action, than in traditional videoconferencing. Therefore, someone experiencing telepresence would be able to behave, and receive stimuli, as if part of a meeting at the remote site. For example, some recent 3D telepresence systems have brought great features for interaction between users and the environment. Participants could move through the scene of remote partners by freely and interactively changing their viewpoints. Some systems enable users to interact with virtual objects as if they were real. In other systems users are allowed to project their documents onto any surface. All these powerful features result in

Figure 1.1: Illustration of 3D telepresence where remote collaborators can communicate with each other as if they are co-located at the same place (adapted from [86]).

interactive participation of group activities that would bring benefits to a wide range of users. 3D telepresence can place people at the center of the collaboration, empowering them to work together in new ways to exchange idea, accelerate innovation, and do more with less. And it is more than just academic or business meetings; it could also be in-person experience for all people. Imagine that individuals from every aspect of life use it in new and innovative ways, from remote learning to telemedicine, from corporate training to everything that requires people joining together to collaborate and gain access to places that were previously out of reach. Given the aforementioned advantages of 3D telepresence, we should make efforts to improve all aspects of it since it is still relatively young compared to other established scientific fields.

1.1.2 Directions of 3D Telepresence

Since 3D telepresence is full of potential, many researchers are working on it, focusing on different directions. Some major research directions include:

3D Hardware. Hardware plays one of the most important roles in 3D telepresence. Lots of

Figure 1.2: Various 3D telepresence systems (adapted from [86]).

input devices have been developed to connect the user and the telepresence system. For example, RGBD cameras and optical tracking glasses could pass the users' appearance, position and point of view to the telepresence system. Other wearable tracking devices with touch points and activator pads, e.g. data gloves and tracking suits, allow people to control the input using motion. Output devices, such as stereoscopic display, are capable of conveying a depth perception to the viewer.

3D Modeling and Reconstruction. 3D modeling refers to the technique of developing a mathematical representation of any three-dimensional surface of object. It is often used to construct the virtual environment, or the virtual object users are interacting with. 3D reconstruction is the process of capturing and recovering the shape and the appearance of real objects. Many 3D modeling and reconstruction techniques are developed to construct high-quality model of scenes and objects.

Multi-view Telepresence. Multi-view telepresence means the scene can be rendered from a viewpoint different from those acquired from the physical cameras. The goal of multi-view telepresence is to allow the viewer to move through the scene by freely and interactively chang-

ing his/her viewpoint. Such telepresence systems are usually equipped with multiple cameras and rely on novel view synthesis from the acquired video.

User Interaction. User interaction is another interesting topic in 3D telepresence. There are various interaction ways that have been discussed in recent years. For example, user/system interaction allows users to communicate with the system interactively with gesture, i.e. adjusting viewpoint using head movement, or use any surface as display. User/object interaction often involves virtual objects which users are able to play with. This technique may rely on collision detection. User/user interaction mostly refers to interaction between remote collaborators in 3D teleconferencing, focusing on gaze correction, and so on.

Foreground Extraction. In many 3D telepresence applications, participants in different locations can be extracted and put in a same novel environment to produce the perception of co-locating. Therefore, many foreground extraction techniques are proposed to generate high-quality extraction of foreground. We will discuss these techniques in Section 2.4.

Data Processing in 3D Telepresence. A 3D telepresence system often has a large amount of data coming from different devices, including critical and redundant data. Therefore, efficient processing of these data is necessary and important. One conventional direction in 3D telepresence aims to find the best way to process these data, e.g. denoising and filtering the noisy input data. We survey some related work on denoising and filtering, please refer to Section 2.3.

Others. There are also other popular research directions focusing on improvement of different aspects of 3D telepresence. 3D Rendering is mainly about improving the rendering quality of 3D telepresence using modern 3D graphics techniques. Data compression and transmission targets at efficient transmission to improve rendering frame rates in 3D telepresence.

1.1.3 Challenges

Real-time Performance. Real-time performance is critical to the user experience in 3D telepresence. Either 3D teleconferencing or other 3D telepresence applications (telemedicine/tele-robotics/remote education) requires smooth communication between users with minimal delay and interruption. However, due to the limitation of hardware and sometimes the high computational cost, real-time performance is difficult to achieve.

High-quality Data. Input data of 3D telepresence is often noisy and error-prone due to hardware instability, especially for the data from commodity RGBD cameras. Although various conventional and novel methods have been applied to denoise or filter the data, high-quality result can only be obtained in rare cases.

High-quality Foreground Extraction. Foreground extraction is an important direction in 3D telepresence. It can be used to separate the foreground person with the background scene. Current foreground extraction techniques can achieve high-quality result, however, only under some circumstance with many constraints. Therefore, high-quality foreground extraction methods that can guarantee real-time performance are needed.

Foreground with Novel Background. While merging extracted foreground persons with novel background scenes, new problems emerge. First, foreground and background might be obtained from very different situation, for example, under different lighting conditions and from different input devices. How to merge the foreground and background to make them look more natural remains a problem. Then, another problem is the interaction between the foreground and the novel background, or the interaction between different foreground. The occlusion and the collision have to be considered to enable natural experience.

1.2 Research Objective

The major goal of this research is to find out qualified methods to solve these problems. In details, my research objectives are:

- To explore the nature of depth data from commodity RGBD camera, find its intrinsic problem and develop an efficient method to improve the quality of such data to further enable high-quality 3D telepresence;
- To devise an efficient method that can support automatic and real-time foreground extraction for arbitrary background scene with commodity RGBD camera;
- To implement a novel CUDA-based temporal-coherent approach for real-time foreground object extraction from RGBD videos.

1.3 Report Organization

The remainder of this report is organized as follows: Chapter 2 reviews the history of 3D telepresence, and surveys related work in emerging 3D input devices, data filtering, as well as foreground extraction techniques that can possibly be used. Chapter 3 presents novel filtering methods tailored for filtering raw 3D depth data acquired from commodity RGBD camera. Chapter 4 describes our proposed foreground extraction method to perform real-time extraction of foreground by taking full advantage of various novel techniques and commodity RGBD camera. Chapter 5 describes our proposed 3D scene replacement method to perform real-time composition of foreground and novel background using commodity RGBD camera. Chapter 6 draws the conclusion and discusses future work.

Chapter 2

Literature Review

Since my research aims to explore new methods to achieve high-quality 3D telepresence experience, this chapter starts by first reviewing the history of telepresence. After that, emerging 3D input devices are reviewed because we mainly focus on 3D telepresence system with commodity RGBD camera. Then, to explore new filtering methods tailored for raw depth data, related work of various denoising and filtering methods are discussed. Lastly, we survey foreground extraction techniques that could possibly be referenced and adopted to provide and support the high-quality foreground extraction for real-time 3D telepresence.

2.1 History of Telepresence

There is a rich history of work on telepresence. The development of the idea of telepresence is attributed to Robert A. Heinlein for his science fiction short story *Waldo* [24], where he first proposed a primitive telepresence master-slave manipulative system. The term *telepresence* was then coined by American cognitive scientist Marvin Minsky in 1980, who outlined his vision that focused on giving a remote participant a feeling of actually being present at a different location [46].

One earliest and important research project in telepresence, the Ontario Telepresence Project (OTP), began in 1990. The target of this project is to design and field trial advanced media space system in a variety workplaces in order to gain insights into key sociological and engineering issues.

Figure 2.1: The office of future [58].

The pioneering works, the office of the future [58] and the blue-c system [23], are milestone projects toward the vision of telepresence. The office of the future [58] introduced a novel semi-immersive display in an office-like environment, one that combines acquisition and display (Fig. 2.1). The basic idea is to use real-time computer vision techniques to dynamically extract information of the visible surfaces in the office, e.g. walls, furniture, objects and even humans, and then to project images on the surfaces. In this case, one could either designate every-day physical surfaces as *spatially immersive display* surfaces, or transmit the dynamic image-based models over a network for display at a remote site. The blue-c system [23] is a immersive projection and 3D video acquisition environment to support virtual design and collaboration (Fig. 2.2). It combines simultaneous acquisition of multiple live video streams with advanced

Figure 2.2: The blue-c system [23].

3D projection technology, creating the perception of total immersion. While the system is primarily intended for high-end collaborative spatially immersive display and for telepresence, the setup is highly scalable, allowing users to adjust the number of projectors and cameras.

There are more and more research projects targeting at telepresence application in recent years. For example, Wu et al. [90] in their work presented a multi-layer framework and a new dissemination protocol to support multi-stream/multi-site collaboration. Jones et al. [28] presented a 3D teleconferencing system able to transmit the face of a remote participant in 3D to an audience gathered around a 3D display, maintaining accurate cues of gaze, attention and eye contact. Yang et al. [94] employed multiple correlated 3D video streams to provide a comprehensive representation of the physical scene in each environment.

In 2011, Maimone and Fuchs demonstrated a telepresence system able to capture a fully dynamic 3D scene the size of a cubicle while allowing a remote user to explore around the scene from any viewpoint [43]. The system preserves eye gaze and does not require any wearing devices. Then in 2012, they further enhanced their system by presenting solutions to several issues: resolving interference between multiple RGBD cameras, data merging and color matching [42]. And they also presented a real-time 3D capture system that offers improved image quality and significantly reduced temporal noise, and is capable of capturing entire room-sized scene simultaneously [40].

Besides these progress, Kuster et al. introduced the FreeCam system based on novel view synthesis that could provide live free-viewpoint video at interactive rates using digital cameras

Figure 2.3: Principle of time-of-flight camera. [86].

Figure 2.4: Left image is time-of-flight camera: D-IMager [53]; right image is 3D laser scanner [86].

and commodity RGBD cameras [34]. The viewer is allowed to move through the scene by freely and interactively change his viewpoint.

2.2 Emerging 3D Input Devices

In recent years, many emerging 3D input devices have been used in various applications, such as movie industry, 3D gaming and 3D telepresence. These devices are capable of analyzing the real-world object or environment to collect data on its shape and appearance, or even position and movement. Then the collected data can be used as input to construct digital, 3D models.

Figure 2.5: Left image is Microsoft Kinect [44]; right image is PrimeSense Carmine 3D sensor [57].

They each have strength and weakness making them suitable for different situations.

Among them, the triangulation 3D laser scanners use light to probe the environment and then find the location of the objects through the triangulation technique (Fig. 2.4). The advantage of triangulation range scanners is that their accuracy is relatively high, which is on the order of tens of micrometers. The disadvantage of triangulation range scanners is that have a limited range of meters.

Time-of-flight camera (ToF camera) (Fig. 2.4: left), the range imaging camera that resolves distance based on the known speed of light, measures the time-of-flight of a light signal between the camera and the object. The lateral resolution of time-of-flight cameras is generally low compared to standard 2D video cameras, with most commercially available devices at 320x240 pixels or less [66, 5]. The characteristics of Time-of-flight scanners are exactly the opposite of triangulation-based laser scanner. They are capable of operating over very long distance, on the order of kilometers, but the accuracy of the distance measurement is relatively low, on the order of millimeters. Example of time-of-flight cameras include SwissRanger 4500 [25] and D-IMager [53] (Fig. 2.4: left).

Another kind of 3D scanner is the structured-light scanner that projects a pattern of light on the object and measures the three-dimensional shape of the object by looking at the deformation of the pattern on the object. The advantage of structured-light 3D scanners is speed and precision.

They are exponentially more precise than laser triangulation and some existing systems are capable of scanning moving objects in real-time.

One popular structured-light scanner that is gaining more and more attention is the Kinect device [44] (Fig. 2.5: left), which was originally designed for Microsoft Xbox 360 game system. Using Kinect, 3D depth information can be acquired in real-time, and by using currently available APIs such as the OpenNI and OpenKinect, we can obtain high-level information such as 3D skeleton and point cloud data. This recent innovation has spawned many interesting mid-air interaction applications. Similar to Kinect, PrimeSense Carmine 3D sensor [57] is also based on structured-light principle (Fig. 2.5: right).

2.3 Kinect Depth Denoising and Filtering

2.3.1 Telepresence Applications with Kinect

Among the various researches that use RGBD camera, KinectFusion by Newcombe et al. [51] is a state-of-the-art method, which offers efficient high-quality room-sized 3D reconstruction. Since this method employs Kinect as a handheld 3D scanner to acquire depth, it can capture different views of the same object to improve the reconstruction quality. However, this approach is not feasible for our targeted application - *3D telepresence*, because the hardware sensor in our application has to be mounted and fixed in the physical space rather than freely movable.

More recently, Maimone and Fuchs [43, 41, 42] proposed an enhanced 3D telepresence system with multiple Kinects to improve the 3D visual experience. Kuster et al. [34] developed a hybrid camera system for telepresence and 3D video-conferencing using multiple high-quality digital cameras.

2.3.2 Kinect Depth Filtering

Traditional image denoising methods such as bilateral filtering [79] can be applied to filter 3D depth data. It smoothes images while preserving edges, by means of nonlinear combination of nearby image values, see Fig. 2.6 and Fig. 2.7. The method is non-iterative, local and simple.

Figure 2.6: Gaussian filter and bilateral filter. Left image is gaussian filter; right image is bilateral filter [86].

Another traditional method that can be employed is the NL-means algorithm [4]. It uses the non local means (NL-means) algorithm, which is based on a non-local averaging of all pixels in the image. The NL-means compares the grey level similarity in the geometrical configuration in a whole neighborhood rather than only in a single point, see Fig. 2.8: right for reference. This fact allows a more robust comparison than normal neighborhood filters, therefore producing better filtering results. However, it is not a good option with Kinect because Kinect has special disparity-like properties.

Feature-preserving denoising methods [64] have also been developed to process the range data captured by Kinect. This work presents a similarity-based neighborhood filtering technique for static and dynamic range data from 3D cameras. The basic idea is the non-local similarity measure that determines the resemblance of two points by not only utilizing their local properties (e.g. position and normal) but also comparing the region of the surface surrounding the vertices instead.

Based on general image denoising approach, Maimone and Fuchs [43] developed a modified two-pass median filter for hole-filling of Kinect depth data. In this first pass, an expanded filtering window is used to fill larger holes with no smoothing applied. In the second pass, a smaller window is used to fill any remaining small holes and smooth the depth image. Their method can ensure that edges in the depth map are preserved precisely and no holes are ignored.

Fu et al. [19] proposed a bilateral filtering framework that integrates both spatial and temporal information in filtering Kinect depth data. The method exploit both the intra-frame spatial

Figure 2.7: Comparison of gaussian filter and bilateral filter. Left image is input image; middle image is gaussian filtered result; right image is bilateral filtered result [79].

correlation and the inter-frame temporal correlation to fill the depth hole and suppress the depth noise (see Fig. 2.8: left). Besides this, a divisive normalization approach is used to assist the noise filtering process.

Nguyen and Izadi [52] derived an empirical noise model for Kinect sensor, and applied it to improve the filtering quality in KinectFusion [51]. In the model, both lateral and axial noise distributions are measured, as a function of both distance and angle of the Kinect to an observed surface. And they found that lateral noise increases linearly with distance while axial noise increases quadratically, see Fig. 2.9 for details.

Other than local neighborhood filtering methods, Reisner-Kollmann et al. [60] proposed a view-dependent locally optimal projection method to specifically addressing point-sample data from depth cameras. The method is adaptive to different noise scale to keep details in well-defined areas, but meanwhile smooth the areas with large noise values. Liu et al. [37] introduced a guided inpainting and filtering method for Kinect depth maps. In order to enhance the depth maps, they propose an inpainting algorithm based on extended fast marching method, which incorporates an aligned color image as the guidance. Then an edge-preserving guided

Figure 2.8: Left image is illustration of spatial and temporal depth characteristics. Top: Plot of the neighboring depth difference; the horizontal axis represents depth value. Bottom: depth variation with the time; the horizontal axis represents frame number.[19]; right image is scheme of NL-means strategies [4].

filter is further applied for noise reduction. However, the above two methods require huge computational load, making them unsuitable for real-time depth filtering.

To fill the holes in the depth map, Richard et al. [61] introduced a multi-resolution fill-in technique inspired by a push/pull approach [22] applied to joint-bilateral upsampling [32].

In contrast, in our first project we propose a real-time GPU-based multi-scale filtering method for filtering Kinect depth data. Compared to local methods, our method produces higher-quality 3D reconstruction results, in particular, on large depth-quantized surface patches. Compared to global methods, which could produce higher-quality results by sacrificing the computational performance, my method is local, and so, can run in parallel with real-time performance.

2.3.3 Depth Inpainting

Depth inpainting is the process of reconstructing lost or deteriorated parts of depth maps. Depth map inpainting can generate visually plausible structures for the missing areas. A popular inpainting algorithm is the Fast Marching Method (FMM) by Alexander Telea [77].

Recently, in Petrescu's work [56] they propose a lightweight algorithm based on raymarching that applies the classic raymarching algorithm used in rendering for inpainting the depth maps

Figure 2.9: Left image is measured lateral noise (top) and axial noise (bottom) for Kinect raw depth; right image is linear and quadratic fits to the lateral and axial noise components [52].

from Kinect. This method allows for fast filtering, implemented in parallel on the GPU, even on extremely bad samples and unfavorable filtering scenarios.

Also, Xue [92] proposes a low gradient regularization method in which they reduce the penalty for small gradients while penalizing the nonzero gradients to allow for gradual depth changes.

2.3.4 Scale-space and Multi-scale Analysis

Scale-space theory, or multi-scale analysis, is a popular tool in computer vision and image processing. It dated back to the early work of Witkin [89] who suggested that real-world objects have structures of different scales, thereby motivating the development of computational methods to perform image denoising. Many of later work are based on this scale-space theory.

Faghih et al. [16] proposed to combine spatial and scale-space technique in a probabilistic manner so that edge images can be derived in which the edges are well localized and at the same time noise effects are successfully suppressed. Tai et al. [73] applied multi-scale tensor voting for simultaneous image denoising and compression. In their process, tensor voting are performed at multiple automatically selected scales at all input pixel groups to infer the feature grouping attributes such as region, curve and junction. Liu et al. [39] provided a quantitative definition for the scale of edges, and based on this an automatic scale selection algorithm was proposed.

Figure 2.10: Scale space representation at different scales (adapted from [89]).

Multi-scale analysis is often combined with anisotropic diffusion for image filtering [55]. Schemunders et al. [65] proposed an anisotropic diffusion algorithm based on a multi-scale fundamental form. The edge information of a color image is first assessed by the multi-scale form from which a local anisotropy measure is derived, and used into a gaussian kernel. Feng et al. [18] presents an image local orientation estimation method, which is based on a combination of the principle component analysis (PCA) and multi-scale decomposition. Here the multi-scale framework helps in noise suppression and balancing localization and accuracy.

we approach the Kinect data filtering problem with this multi-scale concept, but consider also

the depth quantization orientation (affected by the relative orientation between Kinect viewing direction and surfaces normal, see Sections 3.2.1 and 3.2.3 for details), which inspires me to devise a novel direction-aware filtering mechanism to further suppress the depth error.

2.4 Foreground Extraction Methods

Traditionally, foreground extraction is a binary segmentation, or hard segmentation, problem. Its goal is to extract the foreground object in an image/video by classifying each image pixel as foreground or background. A number of different approaches have been developed to deal with this problem, e.g., intelligent scissor [48], active contour [31, 52, 6, 8, 7], graph cut and GrabCut methods [3, 62], etc.

Rather than binary labeling, alpha matting improves the quality of foreground extraction by allowing foreground pixels around object boundary to take nonzero transparent values. Hence, we can extract foreground objects with semi-transparency, and avoid extracting colors originated from the background. This is also known as soft foreground extraction, or soft segmentation. Several approaches have been developed to handle this problem, e.g. Bayesian matting [13], closed-form matting [35], KNN matting [10], etc. Refer to [83] for a comprehensive survey on image and video matting.

In our second project, we take the alpha matting approach to extract foreground objects, and employ the closed-form matting model because it has been proved as one of the most effective matting methods [83]. In particular, we devise a novel closed-form matting formulation with temporal coherency for extracting foreground objects from RGBD videos, and I parallelize the computation with CUDA to achieve real-time performance.

2.4.1 Interactive Foreground Extraction

To achieve high-quality foreground extraction, user interaction is often employed to guide the segmentation, e.g., [26, 3, 62, 48, 94, 97, 54] for images and [82, 81, 38, 54, 96] for videos. In detail, users can interactively mark up and annotate the foreground/background regions on

Figure 2.11: Segmentation with user interaction (adapted from [62]).

an image by drawing strokes. Then, they can initiate the segmentation, and further refine it by drawing additional strokes. By this, we can achieve very accurate foreground extraction.

In interactive graph cuts [3], users first mark certain pixels as object or background to provide hard constraints, then graph cuts are used to find the global optimal segmentation of the N-dimensional image. The method can obtain high-quality result that gives the best balance between boundary and region properties. GraphCut [62] developed an iterative version of the interactive graph cuts. It also incorporates a robust algorithm for border matting to estimate simultaneously the alpha matte around an object boundary and the colors of foreground pixels.

Interactive video cut [82] demonstrated an interactive system for efficiently extracting foreground objects from a video. The method is extended from previous min-cut based image segmentation techniques. Meanwhile, a hierarchical mean-shift preprocess to minimize the computation. The authors of Progressive cut [81] proposed to explicitly model the user's intention into a graph cut framework for the object cutout. Not only that the location of the strokes are used to indicate the region of interest, but also the colors of the user drew strokes indicates the kind of changes he expects, and the relative position between the stroke and the previous result indicates the segmentation error.

Figure 2.12: Comparison of some matting and segmentation tools [62].

However, since manual inputs are required, these methods are not suitable for real-time applications, such as 3D telepresence, which is my target application.

2.4.2 Automatic Foreground Extraction

Some recent approaches have been proposed to develop automatic techniques for foreground extraction with videos.

Apostoloff et al. [1] proposed to incorporate learnt priors into Bayesian video matting to allow automatic layer extraction. Chen et al. [11] demonstrated an automatic foreground/background labeling algorithm for web images. In details, the proposed global cost function is combine with prior knowledge, then fuzzy matting components are computed and hierarchically clustered. Tang et al. [74] also presents an automatic foreground extraction algorithm. The method start from the coarsely extraction of shape of foreground object by a saliency detection algorithm. Then the object shape is refined by weighted kernel density estimation and graph cut algorithm. The method can automatically extract the foreground with high accuracy even for the video with dynamic background. In the same year, Tian et al. [78] proposed a 3D spatio-temporal graph cuts for video segmentation. By combining multiple cues at each super pixel, the method can extracts meaningful moving objects.

However, It still remains challenging to achieve both result accuracy and method robustness since full automation is required. For example, Wang et al. [85] proposed TofCut, an effective bi-layer segmentation algorithm, combining color and depth cues in a unified probabilistic

fusion framework and an adaption weighting scheme is employed to control these two intelligently. And Fan et al. [17] proposed a model adaption framework based on the combination of matting and tracking. In their framework, coarse tracking results are used to provide sufficient scribbles for matting, and then the obtained matting results are used to update the tracking model. Both these works pointed out that their methods would easily accumulate and propagate errors when the foreground and background contain similar colors. Another method by Narayana et al. [49] also suffers from a similar problem.

In contrast, my method incorporates only the closest neighboring frames and the most similar neighboring pixels to compute alpha matte, thus can achieve stable and high-quality foreground extraction even when it runs for a long period of time. Moreover, my method is robust against different background environments and human users, see Section 4.6 for my experiment results.

2.4.3 Real-time Foreground Extraction

Other than results accuracy and method automation, another major challenge in video-based foreground extraction is to attain real-time performance. Some proposed methods were focusing on this issue.

For example, Sun et al. [72] proposed an adaptive background contrast attenuation method to preserve the contrast along the foreground/background boundary while achieving real-time performance. The method is based on the observation that the contrast in the background is dissimilar to the contrast across foreground/background boundaries in most cases. Jung et al. [29] proposed a background subtraction method for gray-scale videos. The background image is modeled using robust statistical descriptors, and a noise estimate is obtained. Li et al. [36] presented a framework for real-time segmentation of foreground moving objects with static background. The basic idea is the combination of background modeling and quadrant-map-based segmentation framework.

Another method by Gong et al. [21] developed a GPU-based method with the Z-kill feature to limit the computation to the unknown regions for optimizing the foreground extraction performance. The algorithm is based on some Poisson equations derived for handling multi-channel color, as well as the depth information.

However, one common critical problem with the above methods is temporal coherency. Although certain attempts have been developed for improving the temporal coherency, e.g., sampling the pixels across temporal domain ([72]), the visual results are still not satisfactory.

2.4.4 Real-time Foreground Extraction with RGBD videos

With the emergence of depth cameras, several methods were proposed for extracting foreground objects from RGBD videos with real-time consideration. Wang et al. [85, 84] developed the TofCut system, which adaptively fuses and models the color and depth data, and then extracts the foreground object in real-time. Cho et al. [12] employed the depth data to initiate the binary segmentation, and proposed to achieve temporal coherency by employing a weighted average method in the data post-processing. Kuster et al. [34] developed the FreeCam software with cameras and Kinects for a 3D telepresence application. They segmented the foreground object from the background by using a weighted smoothness term to align the foreground segmentation boundary with the dominant edges in the color and depth images.

Tomori et al. [80] constructed a motorized pan/tilt Kinect system for the acquisition of color and depth images. They applied GrabCut for segmentation, transformed contours into the polar coordinates and combined them. This method significantly improved segmentation near the floor as well as in partially overlapping objects.

Since the depth data from RGBD cameras is usually noisy and error-prone, especially around the foreground object boundaries, maintaining temporal coherency in the foreground extraction is highly challenging. Hence, severe visual flickering could easily be found in the results of existing works. To address this problem, I develop a new closed-form formulation with temporal coherence to extract foreground objects from an input RGBD video, and balance between the accuracy and real-time performance with the help of the GPU. Compared to the existing works above, my method can achieve high-quality results with better temporal coherence while being fully automatic and real-time, see Section 4.6 for the related comparison results.

2.5 Summary

This chapter summarizes the research work related to this research project. I first review the history of telepresence and introduce the latest emerging 3D input devices. Then, I summarize related works in denoising and filtering methods for depth data. Lastly, I survey four categories of foreground extraction methods that can possibly be used to extract dynamic foreground in Telepresence. After this chapter, I will detail the first two projects in Chapter 3 and Chapter 4.

Chapter 3

High-quality Kinect Depth Filtering for Real-time 3D Telepresence

3D telepresence is a next-generation multimedia application, offering remote users an immersive and natural video-conferencing environment with real-time 3D graphics. Kinect sensor, a consumer-grade range camera, facilitates the implementation of some recent 3D telepresence systems. However, conventional data filtering methods are insufficient to handle Kinect depth error because such error is quantized rather than just randomly-distributed. Hence, one could often observe large irregularly-shaped patches of pixels that receive the same depth values from Kinect. To enhance visual quality in 3D telepresence, in this chapter we propose a novel depth data filtering method for Kinect by means of multi-scale and direction-aware support windows. In addition, we develop a GPU-based CUDA implementation that can perform real-time depth filtering. Results from the experiments show that my method can reconstruct hole-free surfaces that are smoother and less bumpy compared to existing methods like bilateral filtering.

3.1 Introduction

3D telepresence is a next-generation multimedia application, allowing remote collaboration through a natural and immersive video-conferencing environment supported with real-time remote 3D graphics. By this, participants can have a perception of being co-located with others who are remote. The pioneering works, the office of the future [58] and the blue-c system [23], are milestone projects toward this vision.

The launch of consumer-grade range cameras, such as Microsofts Kinect [44], enables convenient and low-cost acquisition of 3D depth in real-time, thus not only facilitating numerous applications, such as 3D gaming, but also accelerating the progress in 3D telepresence research. One recent example is a system by Maimone and Fuchs [43].

However, when employing Kinect in 3D telepresence applications, we have to consider the following vital issues:

- First, raw depth data from Kinect is noise-prone and unstable. We need proper filtering to improve the visual quality of 3D data in the 3D telepresence applications;
- Existing data filtering methods on Kinect depth data mainly address uniformly-distributed random noise, but ignore depth quantization problem in the data;
- In 3D telepresence applications, e.g., teleconferencing, the hardware depth sensors have to be mounted on a fixed location in the physical space, rather than being freely movable as in KinectFusion [51];
- The filtering process should run in real-time, so we cannot afford tedious computation with global data optimization, e.g., solving with Poisson equations.

Considering these requirements, we propose a multi-scale direction-aware filtering method, aiming at making use of Kinect in 3D telepresence applications. Our method not only can effectively filter Kinect depth data, in particular to resolve the depth quantization issue, but also can run in real-time. Our specific contributions are as follows:

- (i) The analysis of quantization errors in raw Kinect data and the resulting holes between irregularly-shaped surface patches in the 3D reconstruction;
- (ii) A new filtering technique that smoothes large surface areas while preserving small-scaled surface details; in particular, it can effectively reduce quantization error and filter holes between surface patches;
- (iii) We also devise an efficient CUDA implementation on the GPU; it can perform our filtering method in real-time, thus facilitating 3D telepresence applications.

Figure 3.1: Color-coded Kinect depth data on a flat wall.

3.2 Our Approach

3.2.1 Kinect Raw Depth Data

Kinect sensor is a low-cost depth acquisition device that comes with various kinds of errors, e.g., system error, random noise, and depth quantization. To better understand them, we start with a simple preliminary experiment. First, we use a Kinect sensor to capture a flat white wall with uniform color and illumination at various distances: from 0.5m to 5m, which is the operation range of Kinect. Fig. 3.1 shows color-coded visualizations of two raw depth maps from Kinect; each colored strip in the visualization shows a group of pixels that have the same depth value. When the orientation of Kinect changes with respect to the walls normal, e.g., from a frontal view as in Fig. 3.1(a) to a slanted view as in Fig. 3.1(b), the orientation of the strips changes relatively (see the black arrows in Fig. 3.1).

In detail, these stripes are caused by Kinects low precision, or quantization, in acquiring the depth data. Such precision decreases with increases in distances from Kinect: The point spacing along the Kinects optical axis can be as large as 7cm up to a maximum of 5m. Hence, we often observe large irregularly-shaped patches/strips (see again Fig. 3.1) in the acquired depth data, where the strip width and orientation could change according to the angle between the surfaces normal and the Kinects viewing direction.

Figure 3.2: 1D quantized depth with ground truth in green.

3.2.2 Multi-scale Filtering

Fig. 3.2 depicts the problem of quantization with a 2D example: the vertical axis is depth while the horizontal axis is the screen. The green line shows the actual 1D surface. After quantizing it into three depth levels (grey), the surface continuity is lost, and we see only three groups of pixels with the same depth similar to the stripes shown in Figure 1.

Traditional filtering methods, in particular local methods, aim at high computational performance. They often use a fixed-size support window to filter all the pixels. The larger the window size, the smoother the surface, but the larger the computational demand and the less the amount of small details that can be preserved. Since image features usually occur at multiple different scales as suggested by the scale-space theory, the optimal scale to analyze different pixels can be very different subject to the local image context. Hence, we employ the multi-scale method to detect an optimal scale for each pixel in the depth map. Note that multi-scale analysis is particularly important for Kinect data since we need to overcome the depth quantization problem.

Fig. 3.3 illustrates depth filtering of two different regions (see 1st row: left and right) on the same 1D depth map that is quantized (grey pixels). For the region on the left, it has three

Figure 3.3: First row: two different regions in the same raw depth data: large patches (left) and a small feature (right). Second row: applying small (left) and large (right) bilateral filtering windows. Third row: our multi-scale filtering method avoids producing bumpy patches and can better preserve small surface details. Green lines show the ground truth while red pixels are the filtered result.

large quantized patches. Here, if we use a small three-pixel-sized filtering window (see 2nd row (left)), the gaps between quantized depth levels are smoothed a bit but it creates certain unnatural bumpiness (red pixels) since the ground truth is actually a straight line (green). This suggests why surfaces reconstructed from Kinect are usually bumpy (see Section 3.5). These large patches require a larger filtering window. However, for the region shown on the right, it has a small feature; if we use a large window, the detail will be undesirably filtered out (see 2nd row (right)). Our method determines appropriate filtering window sizes for different regions adaptively (see 3rd row), and thus can produce higher-quality depth filtering results, see Section 3.5. In particular, we suggest that for quantized strips (see Fig. 3.1) the filter window size should be comparable to the strip width; otherwise, undesirable bumpiness would be resulted.

Figure 3.4: Our data processing pipeline.

3.2.3 Direction-aware Filtering

As shown in Fig. 3.1, the orientation of depth quantization (strips) is usually not aligned with screen axes because it depends on the angle between the object surface and Kinect viewing direction. Hence, different surface regions could receive different quantization orientations. Since filtering is fundamentally weighed averaging, the choice and weights of neighbor pixels used in the filtering can greatly affect the filtering quality. Therefore, we propose a direction-aware filtering method (see Section 3.4), attempting to improve the filtering quality by adjusting the pixel weight contributions based on the orientation of local depth gradient.

3.3 Processing Pipeline

Our data processing pipeline has three stages (see Fig. 3.4):

- (i) **Multi-scale Analysis.** We take a Kinect depth data as input, compute the optimal filter window size (scale) for each pixel based on the statistic analysis around the pixel neighborhood, and output an optimal scale map.
- (ii) **Direction-aware Analysis.** In this stage, we estimate the local predominant orientation using the structure tensor for each pixel at the optimal scale (from the previous step) and output an eigenvector map.
- (iii) **Data Filtering.** The two maps are then employed to reconstruct and hole-fill the raw depth map.

3.4 Algorithm

Given a raw depth map from Kinect, say D , we first compute the *optimal scale*, i.e., the filtering window size, of each pixel by [73], and infer the pixels *local predominant orientation* by analyzing the depth gradient. Then, we use these information to construct the weights in the filtering.

3.4.1 Multi-scale Analysis

- (i) We define a set of discrete scales, i.e., candidate filter window sizes. For each scale, we compute the local depth (Gaussian) distribution of every pixel, say (μ_i, Σ_i) , where μ_i and Σ_i are the mean and covariance matrix of the depth distribution at scale i .
- (ii) We compute the conformity of each pixel (per scale) by checking how well its depth value is modeled by (μ_i, Σ_i) . Here, we use a fixed conformity threshold, say t_{con} ; if the conformity is smaller than t_{con} for all scales, the pixel is labeled as *noise*; otherwise, it is labeled as a *salient* pixel.
- (iii) For each salient pixel, we determine the optimal scale as the largest filter size that gives a $|\Sigma_i|$ that is smaller than a fixed covariance threshold, say t_{cov} . If no such scale exists, we label the pixel as *boundary*; else we label it as *region*. The results are aggregated over all pixels and output as an optimal scale map, say S .

3.4.2 Direction-aware Analysis

Using the optimal scale, we compute for each pixel a structure tensor, which is a matrix representation; its major eigenvector indicates the direction of the largest depth gradient in the local pixel neighborhood. In detail, we have the following steps:

- (i) For each pixel, we perform the *Difference of Gaussians* method to calculate the partial derivative, or gradient, $\nabla I = (I_x, I_y)$, over its optimal window I , where I_x and I_y are the local partial derivatives along x and y of the image domain, respectively.

- (ii) Then, we construct the structure tensor matrix

$$M = \begin{bmatrix} (I_x)^2 & I_x I_y \\ I_x I_y & (I_y)^2 \end{bmatrix}, \quad (\text{Eq. 3.1})$$

and apply eigen-decomposition to M to calculate eigenvectors e_1, e_2 .

- (iii) For each pixel, e_1 is the unit vector that indicates the maximum gradient while e_2 is the local tangent. The output of this step is an eigenvector map E .

3.4.3 Data Filtering

In this stage, we construct the weights by using S and E , and perform the filtering with the following formulation:

- (i) For each pixel, say p , we compute its depth similarity, say f_d , against its neighbor pixel q by

$$f_d(p, q) = e^{-\frac{1}{2} \left(\frac{|D(q) - D(p)|}{\sigma_d} \right)^2} \quad (\text{Eq. 3.2})$$

where σ_d is a threshold, which is 9 in practice.

- (ii) Instead of rotating the filter windows to align them with the local depth gradient, we devise an efficient method for direction-aware filtering by giving higher weight to pixels along the eigenvector directions. We define function f_e to measure the directional closeness between pixel p and its neighbor pixel q :

$$f_e(p, q) = e^{-\frac{1}{2} \left(\frac{\min(v \cdot e_1, v \cdot e_2)}{\sigma_e} \right)^2} \quad (\text{Eq. 3.3})$$

where v is the unit vector from p to q , e_1, e_2 are the eigenvectors of p (from E), and σ_e is a threshold, which is 30 in practice.

- (iii) Lastly, we filter depth map D by

$$D'(p) = \frac{1}{W(p)} \sum_{q \in \Omega(p)} D(q) f_d(p, q) f_e(p, q), \quad (\text{Eq. 3.4})$$

where $W(p)$ is the total sum of weights $\sum f_d f_e$, and $\Omega(p)$ is the local filter window (from S).

3.4.4 CUDA Implementation on GPU

Since our filtering method is local, we can develop GPU-based parallel computation methods with CUDA to enable real-time depth filtering. In detail, we assign one CUDA thread to process each pixel in the raw depth data, and group 16×16 threads into one CUDA block. We allocate shared memory per block to store the depth data local to each pixel block (rather than using global memory), so that threads within the same block can access the depth data more efficiently. Since the resolution of Kinect depth data is 640×480 , there are altogether 1200 blocks. Moreover, we divide our method into three CUDA kernels since these three kernel programs are interdependent, and have to be executed sequentially one after the other:

- (i) **Initialization.** First, we copy the raw depth map from host memory to GPU's global memory, and distribute local depth data to the shared memory of each block;
- (ii) **Multi-scale kernel.** Then, we execute the 1st kernel over all threads to compute the optimal scale map; this map is stored in the GPU's global memory;
- (iii) **Direction-aware filtering kernel.** Next, we compute the difference of Gaussian (DoG) using the constant GPU memory (which is cached), and generate the eigenvector map by forming the structure tensor and performing the eigen-analysis [27];
- (iv) **Filtering kernel.** Lastly, we apply the 3rd kernel to perform per-pixel filtering and hole-filling.

3.5 Experiments and Results

Implementation Our test system runs on 64-bit Windows and employs the OpenNI APIs along with a Kinect driver.

3.5.1 Quantitative Comparison

To quantitatively evaluate our method, we perform a surface fitting experiment. First, we use a Kinect sensor to capture depth maps of a white flat wall and a white cylindrical column

Figure 3.5: Depth profile on the walls shown on 1st column of Fig. 3.7. (a), (b), and (c) in this figure correspond to raw data, bilateral filtering and our result, respectively.

(diameter 0.8m and height 2.5m), each at two different distances: 1m and 1.5m. For each of the four captured depth maps, we collect all the 3D raw data points and fit a plane or cylinder over them respectively. Hence, we can consider the fitted plane or cylinder as the estimated ground truth for quantitative comparison, and project each raw data point onto its related fitted surface to compute the deviation (in millimeter) as the acquisition error (per pixel).

Fig. 3.6 (left column) shows the histograms of such error for the four raw depth maps. More specifically, the first two rows correspond to the cases of the wall at 1m and 1.5m while the last two rows are for the cases of the cylinder, respectively. In each histogram, the horizontal axis denotes the amount of error in millimeter while the vertical axis presents the related pixel counts. Furthermore, we perform bilateral filtering and our filtering method on the four raw depth maps, and compute also the deviation error of the filtered results against the estimated ground truth, i.e., the fitted planes/cylinders, see the middle and right columns in Fig. 3.6 for the results. From the histograms, we can see that errors of our filtered results are consistently smaller than those of the bilateral filtering and the raw data. The MSE results also confirm the superiority of our method since it produces much smaller MSE values.

3.5.2 Visual Comparison

To visually assess the quality of a depth map (raw or filtered), we connect the data points into a 3D mesh, and render it with Phong shading. Fig. 3.7 presents the visual comparison results

with four different data sets, each in one column. From top to bottom, the first three rows in the figure correspond to raw depth data, bilateral filtered results (related code is extracted from the PCL (Point Cloud Library) implementation of KinectFusion [51]), and results from our method, while the next three rows present corresponding zoom-in views of the results.

Due to depth quantization, we can see from the 1st row of the figure that raw data typically produce large amount of bumpy surfaces, e.g., the sofa (1st column) and the plane (2nd column). Bilateral filtering can smooth out certain amount of bumpiness (see 2nd row), but since it uses only a small filter window (11x11) over the entire data, quantization effect still affects the filtered results. Our approach produces the best result by adaptively picking an appropriate filter size for different regions in the depth data. Hence, we can produce smoother surfaces (3rd row) as compared to bilateral filtering, and at the same time, can also better preserve small-scale details, see and compare the results on the Human and Cloth data sets in 3rd and 4th columns.

To further evaluate the filtering quality of the wall surface shown in the 1st column of Fig. 3.7, we do a depth profiling by intersecting the wall with a horizontal plane. See the red lines in the related sub-figures. The resulting intersection lines help indicate the surface smoothness (see Fig. 3.5). Comparing the results, it is clear that our method can produce the smoothest surface with the least bumpiness.

3.5.3 Performance Evaluation

We implemented two versions of our method: a CPU version with C++ and a GPU version with CUDA 5.0, and employed a desktop computer with Intel Six Core 3.2GHz CPU and an NVIDIA GeForce GTX 580 1.5GB graphics card. Table 3.1 presents the time performance statistics and shows that our GPU implementation achieves real-time performance. Table 3.2 compares the performance of bilateral filter and our method on the four different data sets shown in Fig. 3.7. It shows that our filtered results have higher quality as compared to that from bilateral filtering. Since our method allows real-time filtering, it can be used as a pre-processing component in 3D telepresence applications.

Table 3.1: Comparison between CPU and GPU performance.

Stage	CPU (millisec.)	GPU (millisec.)
Multi-scale Analysis	26968	3.3209
Direction-aware Analysis	16825	0.5780
Data Filtering	4121	5.1491

Table 3.2: Comparison between bilateral filter and our method.

Data Set	Performance on GPU (millisec.)	
	Bilateral Filter	Our Method
sofa	2.9961	9.0480
ball and box	3.0492	9.7417
human	3.0338	10.1719
cloth	3.0634	8.6158

3.6 Summary

This chapter presents a novel filtering method tailored for filtering raw 3D depth data acquired from Kinect-like RGBD camera. Since Kinect depth data is heavily quantized, I propose a multi-scale direction-aware filtering method, which is capable of effectively addressing the depth quantization problem with Kinect depth data. My experimental results show that surfaces reconstructed from our method are closer to the estimated ground truths and are also of higher visual quality as compared to the common filtering method, i.e., bilateral filtering. Moreover, I develop and implement my method with CUDA, and show that it can efficiently run in parallel on the GPU to filter Kinect depth data in real-time. Hence, it is possible to incorporate it to support our targeted 3D telepresence application, which is our future work.

There is an obvious limitation of our current work, which is the application to depth data captured from those depth sensors who don't deploy the structure-light camera, such as the time-of-flight cameras. Future experiment on the depth data from time-of-flight cameras can be one extension of this work.

Figure 3.6: Histograms of depth deviations between point clouds and the fitted surface (ground truth). The window size used in bilateral filtering is 11x11.

Figure 3.7: 1st and 4th rows: raw data. 2nd and 5th rows: bilateral-filtered results. 3rd and 6th rows: our results. 4th, 5th, and 6th rows are zoom-in view of 1st, 2nd, and 3rd rows respectively. Red lines shown in the 1st column are the intersection lines used in depth profiling (see Fig. 3.5). Size of neighborhood used in bilateral filtering is 11×11 .

Chapter 4

Real-time and Temporal-coherent Foreground Extraction with Commodity RGBD Camera

Foreground extraction from video stream is an important component in many multimedia applications. By exploiting commodity RGBD cameras, we could further extract dynamic foreground objects with 3D information in real-time, thereby enabling new forms of multimedia applications such as 3D telepresence. However, one critical problem with existing methods for real-time foreground extraction is temporal coherency. They could exhibit severe flickering results for foreground objects such as human motion, thus affecting the visual quality as well as the image object analysis in the multimedia applications.

In this chapter we present a new GPU-based real-time foreground extraction method with several novel techniques. First, we detect shadow and fill missing depth data accordingly in RGBD video, and then adaptively combine color and depth masks to form a trimap. After that, we formulate a novel closed-form matting model to improve the temporal coherence in foreground extraction while achieving real-time performance. Particularly, we propagate RGBD data across temporal domain to improve the visual coherence in the foreground object extraction, and take advantage of various CUDA strategies and spatial data structures to improve the speed. Experiments with a number of users on different scenarios show that, compared with state-of-the-art methods, my method can extract stabler foreground objects with higher visual quality as well as better temporal coherence, while still achieving real-time performance (experimentally, 30.3 frames per second on average).

4.1 Introduction

Foreground extraction has been an important component in many multimedia applications. Its goal is to segment out the foreground object from a given background image or video. By this, we can focus the data processing on the extracted foreground region, and reduce the computational cost. Moreover, foreground extraction can also benefit many multimedia applications, for example, visual recognition of objects, image/video editing, and compression.

Recent advances in parallel computing technologies such as GPU and CUDA make it possible to develop complex methods that run in real-time. Through this, we could significantly enrich the capability of multimedia applications. Particularly, by exploiting commodity RGBD cameras such as Microsoft's Kinect, we could extract color and depth of the foreground objects from RGBD videos, and enable new forms of multimedia applications such as 3D telepresence, where the receiver sees the sender as more than a front-view image but a 3D entity. This research focuses on RGBD videos, and our goal is to develop a real-time foreground extraction method to extract the color and depth of the foreground object from an RGBD video with high temporal coherence and quality.

To support real-time foreground extraction, one well-known approach is chroma key, which is also commonly known as the blue screen or green screen method [47, 67]. By capturing the actor (foreground) in front of a blue/green background, we can easily segment out the foreground by examining the hue of each pixel. It is an effective, simple, and stable solution, but assumes a background with a fixed color, so it cannot deal with general background environments. Moreover, it also requires proper lighting and camera exposure, and the foreground colors have to be significantly different from the background.

Several real-time foreground extraction methods have been developed to overcome the limitations of chroma key. Kaewtrakulpong and Bowden [30] developed a background subtraction method to detect dynamic objects by modeling and updating the background over time. Levin et al. [35] proposed a closed-form matting model to improve the quality of foreground extraction, and later, Xiao et al. [91] accelerated the method with real-time performance using graphics hardware. More recently, Kuster et al. [34] developed a 3D telepresence system with RGBD videos, where they segmented out the foreground actor from the background by using

a weighted smoothness term to align the segmentation boundary with dominant edges in the image frame. However, one common critical problem with the existing methods is *temporal coherency*. They often produce severe flickering results for the foreground objects such as human motion, thus affecting the visual quality as well as the data analysis.

In this chapter, I propose a novel temporal-coherent approach for real-time foreground object extraction from RGBD videos. My work is designed to take an RGBD video stream from a commodity depth camera such as Microsoft’s Kinect and PrimeSense 3D sensor, and extract a high-quality temporal coherent foreground object with motion and depth from an RGBD video. Compared to the existing methods for foreground extraction, my method is fully automatic without manual markup; and more importantly, we achieve real-time performance by developing our method on the parallel computation platform of CUDA and GPU. Moreover, the foreground extraction results are of high temporal coherency. Particularly, I reduce the temporal flickering artifacts around foreground object boundary compared to state-of-the-art real-time foreground extraction methods.

In summary, this work has the following contributions:

- (i) First, we present a new method to temporally fill the missing data in the RGBD video based on shadow detection, and adaptively combine the color and depth masks of the foreground object to form a better trimap for subsequent temporal matting;
- (ii) Second, we develop a novel closed-form matting formulation with temporal coherence, and formulate an efficient CUDA-based pipeline that can maintain temporal coherence in the foreground object extraction as well as attaining real-time performance; and
- (iii) Lastly, we develop our method entirely on the GPU by parallelizing most of the computation with CUDA strategies; by this, we can achieve not only high-quality foreground extraction but also real-time performance.

4.2 Overview

The input to our method is an RGBD video stream of color and depth, capturing from a commodity depth camera. In our implementation, we have tried our method with both Kinect [44]

Figure 4.1: Overview of our foreground extraction approach, which consists of four stages of GPU computation: 1) background modeling; 2) data preprocessing; 3) trimap generation; and 4) temporal matting.

and PrimeSense 3D sensor [57]. While streaming RGBD data into the GPU, our method performs the following stages of computation with CUDA to achieve real-time foreground extraction (see Fig. 4.1 for a running example):

4.2.1 Background Modeling

Our method assumes a background-only RGBD video stream at the first few seconds when the system starts. By this, we can construct a background model individually for the color and depth channels of the RGBD video (see Fig. 4.1, stage 1). Here we employ the improved adaptive Gaussian mixture model [30] because its learning process is fast and accurate, and its computation can also be parallelized with the GPU.

4.2.2 Data Preprocessing

After background modeling, we start the main computation pipeline, which is a real-time process with CUDA (see stages 2 to 4 in Fig. 4.1). Among them, we first need a preprocessing stage (stage 2) because the depth images from commodity RGBD cameras are usually noisy

and incomplete with no-measured depth pixels (NMD). General hole-filling methods may not work properly for these NMD pixels. Hence, we adopt the shadow-detection method [95, 76] to classify the NMD pixels on the GPU, and then temporally and adaptively fill these pixels with more appropriate depth values according to their NMD types (shadow or noise). The output from this stage is a refined depth map without NMD (zero) values (see Section 4.3 for details).

4.2.3 Trimap Generation

Next, we prepare a trimap for temporal matting with the following steps: First, we apply the results from stage 1 to subtract the background from the current frame, and obtain two initial binary masks: one for color and one for depth. Note that we cannot naively combine these two masks because it will lead to problematic extraction results (see our experiment in Section 4.6). Then, based on the observation that depth masks usually give better foreground shape (but noisy object boundary) while color masks usually give smoother object silhouette (but easily corrupted by shadow and illumination variation) (see also Fig. 4.1, stage 3), we propose a novel CUDA-based scheme to adaptively combine the color and depth masks into a more reliable binary mask (see Section 4.4 for details). Lastly, we further apply morphological operations to generate the trimap from the combined mask.

4.2.4 Temporal Matting

After generating the trimap, we develop a CUDA-based temporal matting method to compute the alpha matte (see Fig. 4.1, stage 4). In particular, we construct a 3D volume of color values by combining the current video frame with its neighboring frames across time. Then, we iteratively partition the 3D volume into smaller blocks, so that we can attain real-time performance later by working with the blocks in parallel on the GPU. After that, it comes to our key contribution at this stage, i.e., we formulate the closed-form temporal matting model for achieving temporal coherency across frames, as well as for maintaining spatial consistency and smoothness in the current frame. One strategy we proposed to balance quality and performance is to approximate the nearest neighbor search by examining only similar pixels rather than all

pixel neighbors. Lastly, we construct a matting Laplacian per block, solve linear systems in parallel on the GPU, and obtain the overall alpha matte to produce the foreground extraction (see Section 4.5 for details).

4.3 Preprocessing

To employ depth images for later stages in our pipeline, we need a preprocessing stage to clean up the raw input depth data. Here we have two concerns:

- First, preprocessing depth images is challenging mainly due to the no-measured depth (NMD) pixels, which often require extensive processing. According to Yu et al. [95], NMD pixels can be classified as noise (due to out-of-range or reflective surface) or shadow (due to foreground-object occlusion). Thus, we follow this classification, and fill depth values based on the NMD types.
- The second concern is performance. Sophisticated image processing methods, e.g., those by optimization, are not suitable here since we aim for real-time performance and have to reserve computation time for later stages in the pipeline. In addition, a very high-quality depth image might not be necessary for generating the trimap and later the alpha matte.

Taking into account these concerns, we develop a CUDA-based method to automatically fill the NMD pixels in parallel with the following two sub-stages on the GPU:

4.3.1 Shadow Detection

We adapt the method by Yu et al. [95] to a CUDA-based version as follows:

- (i) First, along each horizontal scanline in the image space, we identify NMD pixels whose left neighbor in the scanline is a non-NMD pixel. These pixels are called *object-edge* pixels (see also Fig. 4.2). To find these object-edge pixels, we deploy one CUDA kernel to independently check each pixel in parallel, and the result of this step is a list of object-edge pixels.

Figure 4.2: Shadow NMD region.

- (ii) Then, we use another CUDA kernel to operate on each object-edge pixel in parallel. Starting from the object-edge pixel, the kernel searches along the scanline from left to right for a consecutive list of NMD pixels until it reaches a non-NMD pixel. Such a consecutive list of NMD pixels is called an *NMD region*.
- (iii) Lastly, we continue with the same CUDA kernel (per object-edge pixel) to compare a theoretical offset from a mathematical model against the number of NMD pixels in the NMD region. If the difference is less than a certain threshold, which is 7 in practice, we classify the pixels in the NMD region as shadow, or noise, vice versa.

Figure 4.3: Temporal hole-filling. Left: an input raw depth map, and right: a hole-filled depth map.

4.3.2 Adaptive Temporal Hole-Filling

After shadow detection, our next task is to fill the depth value of each NMD pixel according to its type: shadow or noise. To achieve high performance, we continue to work inside the last CUDA kernel in the shadow detection process since it allows us to operate with each NMD region in parallel (note: per object-edge pixel is equivalent to per NMD region). Moreover, for fast data access, the GPU memory stores not only the current raw depth image but also the two temporally-neighboring depth images from the previous and next frames.

- Case 1: shadow NMD region. Since shadow pixels are caused by foreground-object occlusion, they are in fact background elements. Hence, we estimate their depth values by drawing depth values from nearby non-NMD background pixels. To do so, we denote q_0 as the rightmost pixel in a given NMD shadow region, and q_1, q_2 , etc. as the non-NMD pixels on its right hand side in sequential order (see Fig. 4.2). Then, we average the depth values of q_1 to q_N (while excluding the maximum and minimum depth values among them, i.e., z_{max} and z_{min}):

$$z(q_0) = \frac{1}{N-2} \left[\sum_{1 \leq i \leq N} z(q_i) - z_{max} - z_{min} \right],$$

where $z(q_i)$ denotes q_i 's depth value, and N is a parameter chosen to be 8. After computing $z(q_0)$ for the current time frame, we further obtain $z(q_0)$ for the previous and next time frames, and apply standard Gaussian filtering to average these three depth values (temporally) to produce the final depth value. Then, this final depth value is assigned to pixels in the NMD region.

- Case 2: noise NMD region. In the second case, we estimate the depth value of a noise pixel by using a joint bilateral filter that considers all surrounding non-NMD pixels over the spatio-temporal domain. The Gaussian distance weights are computed using three-dimensional Euclidean distances, and the Gaussian range weights are computed over the RGB space. Since a noise NMD region may have more than one noise pixels, the CUDA kernel here may need to loop over each noise pixels.

Note that to accelerate the CUDA computation with Gaussian and joint bilateral filters, we precompute the Gaussian distance weights, and store them as a matrix in the GPU constant

memory, which has the fastest access speed compared to global memory and shared memory since it is cached. Fig. 4.3 shows our hole-filled result. Note that the hole-filling result may not be perfect, but it provides a completed depth image without zero-valued NMD pixels; these zero-valued NMD pixels may lead to computation errors in the later stages.

4.4 Automatic Trimap Generation

Trimap is a 2D grayscale image used for guiding the determination of the alpha matte. In detail, there are three kinds of pixels in a trimap: white, black, and gray used for indicating *foreground*, *background*, and *unknown*, respectively. In the second stage of our CUDA-based pipeline, our goal is to automatically generate a trimap. To do so, we have the following three processes:

4.4.1 Background Subtraction

Given the color and depth images from the depth sensor, and also the background model (see Section 4.2), we devise a CUDA kernel to perform the background subtraction method [30] on each pixel in parallel. The results are two independent binary masks for the color and depth images of each video frame, where foreground and background pixels are labeled by 1 and 0, respectively.

4.4.2 Adaptive Mask Generation

As discussed earlier in Section 4.2, color and depth masks have their own strengths and weaknesses. Depth masks usually possess better foreground shape but more noisy boundary, while color masks usually possess smoother boundary but can be easily corrupted by shadow and illumination changes (see Fig. 4.4).

Hence, we propose an adaptive approach to combine the two masks, aiming to retaining their advantages in the combined mask. Our key observation here is:

Figure 4.4: Adaptive mask generation: blue boxes highlight smooth and correct boundary, while red boxes highlight rough and erroneous boundary.

The closer a pixel to the depth mask boundary, the more reliable its color mask is, while the farther a pixel to the depth mask boundary, the more reliable the depth mask is.

To put this observation into practice with performance concern, we develop two sequential CUDA kernels to operate on each pixel in parallel. Note that we cannot combine the two CUDA kernels into one because we can only start the second kernel after completing the first one over all the pixels.

- (i) The first kernel operates on each pixel in parallel, aiming to identifying foreground pixels that are on boundary. The kernel starts by extracting the depth mask values in the local 5×1 neighborhood centering on the current pixel; then, it packs the five 0/1 values into a depth mask vector, and computes the dot product between this vector and $[1, 1, 6, -1, -1]$. If the result is 4 or 8, we label the pixel as boundary. Note that 4 and 8 can only be produced by depth mask vectors of $[B, B, F, F, F]$ and $[F, F, F, B, B]$, respectively, where F and B refers to foreground and background, respectively. Hence,

we can efficiently compute one dot product in CUDA to determine both boundary cases. After horizontal detection, we repeat the above process three times: one for vertical and two for diagonal. In the end, this kernel outputs a 2D boundary map after completing the execution over all pixels.

- (ii) The second kernel also operates on each pixel in parallel. At each pixel, say p , this kernel first looks for the boundary pixel closest to p within a local 7×7 pixel neighborhood centering at p . If the distance from p to the closet boundary pixel is smaller than a threshold, which is 5 in practice, we use the color mask of p as its value on the combined mask. Otherwise, we use p 's depth mask value. Fig. 4.4(right) shows the combined mask.

4.4.3 Morphological Operation

After combining the color and depth masks into a single binary mask, we next refine it by morphological operations to generate the trimap. Here we develop three CUDA kernels to operate on pixels in parallel:

- (i) The first two CUDA kernels take the binary mask from the adaptive mask generation as input, and employ a diamond-shaped 5×5 structuring element (stored in GPU constant

Figure 4.5: Morphological operations (erosion and dilation) to generate the trimap.

memory for fast access) to perform erosion and dilation (for two iterations) to generate an eroded binary mask and a dilated binary mask (see Fig. 4.5(middle)).

- (ii) The third CUDA kernel takes the resultant eroded and dilated binary masks as inputs, and examines them pixel by pixel in parallel: if both masks label a pixel as foreground (or background), the output trimap also labels it as foreground (or background); otherwise it is labeled as unknown (see Fig. 4.5).

To improve the data access performance on the GPU, for each kernel, we assign one CUDA thread to process each pixel in parallel, and group 16x16 threads into a CUDA block. Moreover, we allocate shared memory per CUDA block to store the binary-mask input data that is local to each pixel block rather than using global memory, so that threads within the same block can efficiently access neighborhood pixel data. Since the image resolution is 640×480, we have altogether 1200 CUDA blocks.

4.5 Temporal Matting

4.5.1 Closed-form Matting

Before presenting our temporal matting method, which is the last stage in the GPU pipeline, we first briefly describe below the formulation of the closed-form matting model [35].

In alpha matting, a pixel's color, say I_i , is considered to be a linear combination of foreground and background colors:

$$I_i = \alpha_i F_i + (1 - \alpha_i) B_i, \quad (\text{Eq. 4.1})$$

where α_i is the pixel's foreground opacity, and F_i and B_i are the pixel's foreground and background colors, respectively. By assuming that both F_i and B_i are locally smooth over a small window w , we can rewrite the matting equation above, devise the matting Laplacian $\mathbf{L}$, and then solve for α_i by minimizing $\mathbf{L}$ against certain input constraints:

$$\alpha^T \mathbf{L} \alpha = \sum_{k=1}^N \bar{\alpha}_k^T \bar{G}_k^T \bar{G}_k \bar{\alpha}_k, \quad (\text{Eq. 4.2})$$

where α is the alpha matte; N is the number of pixels in the image; $\bar{\alpha}_k$ is a vector of alpha values of pixels in local window w centered at pixel k ; and $\bar{G}_k$ encodes affinities among pixels in w , and each element in $\bar{G}_k$ represents the correlation between a pair of pixels. See [35] for detail.

4.5.2 Our Approach: Construct the Laplacian matrix

Our CUDA-based method for computing the alpha matte consists of two major steps: 1) constructing the matting Laplacian from spatiotemporal pixel neighbors (this subsection) and 2) efficiently solving for the alpha matte on the GPU (next subsection). To enhance temporal coherency while achieving real-time performance, we construct the Laplacian matrix with the following considerations:

1) Trimap partitioning. To achieve real-time performance, it is not feasible to solve the matting equation over the entire image space. The reason here is that since there can be over several thousands of unknown pixels in a trimap, we will end up creating a huge linear system, which is not possible to be solved in real-time. Hence, we first partition the trimap (from the previous stage) into smaller image blocks, and then independently solve the related linear systems in parallel.

To perform the trimap partitioning, we design a CUDA kernel with the following operations. First, we divide the 640×480 image space into fixed-size blocks of size 32×24 , i.e., 400 blocks in total. Then, we use the CUDA kernel to operate on each block in parallel: count the total number of unknown pixels in each block; if the number of unknowns is larger than 100, subdivide the block into four 16×12 sub-blocks. After that, we repeat the process by applying the same CUDA kernel to operate on the divided 16×12 blocks (in parallel) until all partitioned blocks contain less than 101 unknown pixels. During this adaptive partitioning, we ignore image blocks with no unknown pixels, and record the image blocks with unknowns in a list, say S .

2) Spatiotemporal pixel neighbors. In the closed-form matting model, affinities among pixels are defined with 2D neighbors in a small local window w of size 3×3 over the spatial

Figure 4.6: 3D spatiotemporal neighbors (right) for temporal matting: the red dot denotes the unknown pixel.

domain. To achieve temporal coherency, we need to consider 3D spatiotemporal pixel neighbors to encode affinities among pixels (see Fig. 4.6), where the 3D neighbors of a pixel p are the neighboring pixels that are most similar to p within a local spatiotemporal window centered at p .

To efficiently determine the 3D neighbors of each unknown pixel in the trimap, we propose the following parallel method in CUDA. First, we pack the current video frame together with the previous and next video frames into a 3D volume of RGB values, and locate in it the related sub-volume for each image block in S , e.g., $32 \times 24 \times 3$ or $16 \times 12 \times 3$. Then, we use a CUDA kernel to operate (in parallel) on each unknown pixel together with its volume block. In detail, the CUDA kernel first constructs a feature vector by collecting all pixel RGB values in a local spatial 5×5 neighborhood centering at the unknown pixel. Then, the CUDA kernel searches for pixels in the same volume block with similar feature vectors; note that to improve the performance, we use the parallelized k-nearest neighbor (KNN) searching algorithm [20] to find 8 similar neighbors within the volume block for each unknown pixel. Note also that during this process, the similar neighbors can come from the current frame, as well as from other time frames (see again Fig. 4.6).

3) Closed-form Temporal Matting. To support temporal coherency, we extend the closed-

form matting model as follows:

First, for each unknown pixel, we consider a set of nine pixel neighbors (the unknown pixel itself together with its 8 similar pixel neighbors), and encode 3D distances (in the related volume block) between every pair of pixel neighbors, say i and j , into a 9×9 kernel matrix:

$$\mathbf{M}_K(i, j) = \exp\left(-\frac{\text{dist}_{3D}(i, j)^2}{2\sigma^2}\right), \quad (\text{Eq. 4.3})$$

where $\text{dist}_{3D}(i, j)$ is the Euclidean distance between i and j in the related volume block, and σ is $\sqrt{2}$. Later, $\mathbf{M}_K$ will be used to set weights on elements in the Laplacian matrix, so that we can emphasize proximity among pixel neighbors.

Second, for each image block in S , we construct the matting Laplacian $\mathbf{L}_{3D}$, which contains the affinities among all pixels in its related volume block, i.e., across the temporal domain:

$$\alpha^T \mathbf{L}_{3D} \alpha = \sum_{t=-1}^1 \sum_{k=1}^N \bar{\alpha}_{k_{3D}}^T \bar{G}_{k_{3D}}^T \bar{G}_{k_{3D}} \bar{\alpha}_{k_{3D}}, \quad (\text{Eq. 4.4})$$

where t is the time frame; N is the total number of pixels in the image block, so $\mathbf{L}_{3D}$ is $3N \times 3N$; and following [35], $\bar{\alpha}_{k_{3D}}$ and $\bar{G}_{k_{3D}}$ encode the alpha values of pixels and affinities among pixels, respectively, but in the 3D neighborhood. From this, we can further obtain α by solving:

$$(\mathbf{L}_{3D} + \lambda \mathbf{D}) \alpha = \lambda \beta, \quad (\text{Eq. 4.5})$$

where λ is a constant (set to be 100); $\mathbf{D}$ is a diagonal matrix whose elements are 1 for known pixels and 0 for unknown pixels in the trimap; and β is a row vector whose elements are 1 for foreground pixels in trimap and 0, otherwise.

Third, to prepare $\mathbf{L}_{3D}$ for solving for α , we progressively construct sub-matrices of $\mathbf{L}_{3D}$ related to the unknown pixels in the image block by the following five CUDA kernels; these CUDA kernels operate on unknown pixels in parallel:

- (i) The first kernel averages the RGB values of the nine pixel neighbors of the given unknown pixel and computes the mean vector, while the second kernel computes the related covariance matrix;

- (ii) Then, the third kernel computes the 9×9 affinity matrix using the mean vector and the covariance matrix (see [35] for detail about affinity matrix), while the fourth kernel computes $\mathbf{M}_K$ of the nine pixel neighbors; and
- (iii) Lastly, the fifth kernel performs an element-wise multiplication between $\mathbf{M}_K$ and the affinity matrix, and constructs the related sub-matrix in $\mathbf{L}_{3D}$.

4.5.3 Our Approach: Solving for the Alpha Matte

In general, the number of unknown pixels in an image block is significantly smaller than that of the known pixels, so $\mathbf{L}_{3D}$, which is mathematically a $3N \times 3N$ matrix, usually contains a lot of zeros, and zero sub-matrices (for known pixels). To improve the computational efficiency with $\mathbf{L}_{3D}$, we employ a sparse matrix data structure called the coordinate list, which is a list of (row, column, value) tuples. By this, we can later solve the linear system efficiently on the GPU with the CULA Sparse library [14].

In detail, rather than storing $\mathbf{L}_{3D}$ as a full matrix, after we compute a 9×9 sub-matrix of $\mathbf{L}_{3D}$ in the last CUDA kernel of the previous subsection, the CUDA kernel further goes through the sub-matrix and constructs the coordinate list tuples for data values in the sub-matrix. By this, we can efficiently construct the coordinate list without explicitly creating the entire $3N \times 3N$ matrix for $\mathbf{L}_{3D}$. After that, we further construct row vector β and diagonal matrix $\mathbf{D}$ (also as a coordinate list) (see Eq. 4.5) with two other CUDA kernels, and then apply the CULA Sparse iterative solver to solve for the alpha value of pixels in the related image block.

4.6 Experiments and Results

4.6.1 Implementation Details

We implement and run our CUDA-based foreground extraction method on a 64-bit desktop computer with Intel Six Core 3.2GHz CPU and NVIDIA GeForce GTX 580 1.5GB graphics card. For software, we employ the CUDA 5.0 library to develop the GPU code, and the CULA Sparse S5 library [14], which is a GPU-accelerated library, to develop the iterative solver for

Table 4.1: Time Performance of our CUDA-based pipeline.

	Stage	Average Time (milli.sec/frame)	FPS (frame/sec)
One-time	Background Modeling	2.61	383.14
Real-time	Preprocessing	0.33	3030.30
	Trimap Generation	2.21	452.49
	Temporal Matting	30.49	32.80
	Total (Real-time)	33.03	30.27

the sparse matrix system described in Section 4.5. In addition, we use the OpenNI 2.0 API along with the Kinect driver since it is mostly compatible with GPU parallelization, and it was programmed with CUDA 5.0. For RGBD cameras, our method can work with both Microsoft Kinect 1 [44] and PrimeSense 3D Sensor (Carmin 1.09) [57]. Our resolution setting for the Kinect sensor is 640×480 for both RGB and depth at a frame rate of 30 Hz, while the PrimeSense 3D sensor also has the same resolution for the RGBD video, but operates with a shorter capturing range of 0.35 meters.

4.6.2 Foreground Extraction Results

Fig. 4.7 shows the foreground extraction results produced from our method on seven different RGBD videos. These RGBD videos are captured with different participants against different background scenes, which are not simple single-color backgrounds used in green/blue screen methods.

Please see the supplementary video at [here](#) for these results in detail. It can be seen that our method in general achieves more visually pleasant results under different complex background and different foreground users with different motion.

4.6.3 Experiment: Time Performance

To evaluate the time performance of our foreground extraction method, we test it with seven different RGBD videos, each with 500 frames and a resolution of 640×480 . Table 4.1 shows

Figure 4.7: Foreground extraction results. The first column shows a snapshot of each RGBD video before foreground extraction; subsequent images in each row show snapshots of the foreground extracted from the corresponding video.

Figure 4.8: Comparing our method with background subtraction (with RGBD) [30], standard closed-form matting [35], and FreeCam [34] (from left to right). From top to bottom, the 1st and 3rd rows examine the boundary of foreground objects extracted from two different RGBD videos; the 2nd and 4th rows show the related zoom-in views. The last three rows focus on comparing the temporal coherency of foreground object boundary across consecutive video frames.

Figure 4.9: Experiment on adaptive mask generation. From left to right: the 1st column shows the input color and depth masks; the 2nd column shows the binary masks produced by different methods (simple intersection, simple union, and our method); the 3rd and 4th columns show the trimap and foreground extraction results, respectively, produced from the related binary masks, where our adaptive mask generation method can always produce better quality results.

the results: the first four rows list the average time taken per frame and the corresponding frame rate for the four stages in the pipeline, while the last row shows the overall performance for the online portion (stages 2 to 4) of the pipeline.

Note that the time needed to transfer video frame data from main memory to GPU and also to transfer the foreground extraction result from GPU back to main memory are included in the timing statistics of stages 2 and 4, respectively. Hence, we can see from the results here that real-time performance can be achieved by our method. In addition, we would also like to highlight that our method has consistent and stable performance over all the RGBD videos.

4.6.4 Experiment: Compare with other methods

To assess the effectiveness of our method, we compare it with three different state-of-the-art methods: background subtraction (with RGBD) [30], standard closed-form matting [35], and FreeCam [34]. Fig. 4.8 presents snapshots of the comparison results (see supplementary video for details). In this comparison, we would like to highlight two aspects concerning the foreground extraction quality: 1) boundary of the extracted foreground object and 2) temporal coherency.

In the first four rows of the figure, we employ two different RGBD videos, and compare the boundary of the extracted foreground objects. We can see from the results, in particular the zoom-in views in 2nd and 4th rows, that the background subtraction method can only produce very coarse foreground shapes while the standard closed-form matting and FreeCam methods both produce inaccurate or jerky boundaries. In particular, FreeCam may produce false-positive foregrounds, e.g., green pixels near the up-pointing thumb in the 2nd row. In contrast, our method generally extracts foreground objects with smoother and more accurate boundaries. In the last three rows of the figure, we explore and compare the temporal coherency of our results with the other three methods. Zoom-in views of three consecutive video frames are shown here from 5th to 7th rows. From the results, we can see that all the other three methods produce inconsistent boundary over time, thus resulting in serious flickering artifacts when we play the results as videos. In contrast, our method greatly alleviates the problem, and extracts foreground objects from RGBD videos with smooth boundaries that are more consistent over time.

Since we can only present static images here, we encourage readers to see the supplementary video for video versions of the above comparisons.

4.6.5 Experiment: Adaptive mask generation method

We perform an experiment to examine the effectiveness of our adaptive mask generation method by comparing it against simple but efficient methods that combine color and depth masks, i.e., pixel-wise intersection or union. Fig. 4.9 shows the comparison results using two different RGBD videos.

Given the inputs, i.e., color and depth masks (1st column), we can check in this experiment how binary masks (2nd column) affect the quality of trimap (3rd column) and foreground extraction results (4th column). From the results shown in the figure, we can see that though our mask generation method is relatively more complex than pixel-wise intersection and union, it helps generate higher quality trimaps that could lead to more accurate foreground extraction results. Moreover, our adaptive mask generation method, as a part of stage 3 in the GPU pipeline, can reach a very high speed of several hundred frames per seconds (see again Table 4.1).

4.6.6 Experiment: Robustness and Stability

Lastly, to evaluate the robustness and stability of our method, we set up an experiment to run our method in two different environments over a long period of two hours: one in a living room and the other in a research staff office. Here people can keep passing by the depth camera over the testing period. The result we obtained was that even after running for two hours, our system still achieved the same time performance as reported in Table 4.1, and it could still produce plausible foreground-extraction results like those in Fig. 4.7.

4.7 Summary

This chapter has presented a novel CUDA-based temporal-coherent approach for real-time foreground object extraction from RGBD videos. The main challenge here is to seek for high-quality foreground extraction, particularly on temporal coherency, while still achieving real-time performance.

To address the challenge, we have developed our foreground extraction method by carefully devising the CUDA-based parallel computation. In particular, given an RGBD video stream, our

method starts by first detecting the shadows in each depth image and filling the missing NMD pixels accordingly in the depth images. Then, we adaptively combine the color and depth masks to construct a more reliable and effective trimap while maximizing the strengths of the color and depth masks. By further devising a closed-form temporal matting model with spatiotemporal pixel neighbors, we can produce temporal-coherent foreground mattes that could effectively reduce the amount of flickering artifacts. Compared to the existing methods, our method is fully automatic without any manual markup, and it can achieve real-time performance through CUDA-based parallel computation. Lastly, we have also performed a number of experiments to evaluate our method, particularly comparisons against several state-of-the-art methods such as FreeCam. By these experiments, we can see that our method is capable of extracting more stable foreground objects while achieving real-time performance.

Since our current pipeline is based on input depth data from single depth sensor, acquiring higher quality of foreground extraction through deploying multiple cameras are not explored in this work. It is natural to design a multi-sensor system to achieve better results. Hence, future study on temporal matting based on multiple trimaps (produced from multiple input depthmaps) can be one extension of current work.

Chapter 5

Automatic 3D Scene Replacement for 3D Telepresence

Background replacement is an important component in many multimedia applications. By exploiting commodity RGBD cameras, we could further extract dynamic foreground objects with 3D information and replace the background of it in real-time, thereby enabling new forms of multimedia applications such as 3D telepresence. However, one critical problem with existing methods for real-time background replacement is sense of reality. They could exhibit severe implausible and unconvincing results for foreground and background composition, thus affecting the visual quality as well as the user experience in the multimedia applications.

In this chapter we present a new GPU-based real-time 3D background scene replacement method with several novel techniques. First, we employ a high-quality and temporal-coherent algorithm to segment foreground human from background scene. After that, we analyze both the foreground human and the background scene based on different criteria to build a description of the whole scene. Then, we search from a RGBD scene database for the best candidate scene that can be composited with the segmented human naturally and realistically. Survey with a number of people on different datasets show that, compared with simple direct 2D background composition, our method can generate more realistic composition results, while still achieving real-time performance.

5.1 Introduction

In this chapter, we aim to allow a user to pretend to be in another place (not the local scene he/she is physically located) during the real-time 3D telepresence. The following problem statement summarizes our goal:

Given a user (*actor*) who is being continuously captured by a depth sensor during a 3D telepresence, we aim to dynamically replace the scene that he is currently physically in (*local scene*) by some other similar scene (*target scene*), so that he/she can pretend to be in some other place in the eyes of the remote user (*viewer*) .

- *actor*: the local participant who wants to pretend to be in some other scenes;
- *viewer*: the remote participant who communicates with the *actor* through the 3D telepresence system;
- *local scene*: the physical environment where the *actor* is located;
- *target scene*: the target environment where the *actor* wants to pretend to be in;

Here, we will use the above four terms (*actor*, *viewer*, *local scene*, and *target scene*) consistently throughout this chapter.

The motivation behind this work is that users may need to pretend to be in another place during telepresence in many scenarios. For example, a user is about to be interviewed through telepresence system at home. In order to protect his/her privacy and appear more professional to make a good impression to interviewer, replacing the home-like background with an office-like background could be necessary. Also, we need to consider that users may have various behaviors in different situations, such as walking, sitting or presenting.

To achieve such realistic scene replacement is technically not easy. First, in telepresence, user's effective *range of motion* in *local scene* and *target scene* are usually different. In particular, while exploring the *target scene*, the user often has to stop due to the presence of a wall in the *local scene* even though the corresponding space in the *target scene* is empty, or vice versa. This issue is described as *locomotion problem*, and presents a severe limitation to user's ability to interact with *target scene* upon *scene replacement*. Existing works mostly focus on 2D

scene replacement, such as chroma key [47], therefore producing low-quality result. Some other work proposed to solve the locomotion problem but did not give flexible and complete solution that can be applied in a telepresence system. Second, during the *scene replacement*, we need to consider different user-scene interactions when user is walking, sitting or presenting. And this issue has not been thoroughly discussed yet in any previous work.

In this chapter, we assume a simple and convenient setup with only a desktop computer together with a single consumer-grade RGBD camera such as Microsoft Kinect [44], rather than other heavier devices such as head-mounted display and multiple camera's setup that requires calibration. We propose a novel system that consists of a scene database to achieve high-quality scene replacement for real-time 3D telepresence. Compared to the existing methods for scene replacement, my method is fully automatic without user interaction; and more importantly, we achieve real-time performance by developing our method on the parallel computation platform of CUDA and GPU. Moreover, our approach is adaptive to different user scenarios.

In summary, this work has the following contributions:

- (i) First, we propose a novel real-time scene replacement system, and formulate an efficient CUDA-based pipeline that can automatically replace local scene with a proper target scene as well as attaining real-time performance;
- (ii) Second, we present a new scene database with analyzed datasets for the scene replacement system to choose target scene from; and
- (iii) Lastly, we develop our method entirely on the GPU by parallelizing most of the computation with CUDA strategies; by this, we can achieve not only good visual quality scene replacement but also real-time performance.

5.2 Related Work

3D Video Background Composition. Date et al. [15] introduced a highly realistic 3D display system that displays a person in a remote location as a life-size stereoscopic image against background scenery, which corresponds to the observer's viewing position to well reproduce

the fidelity of existence. Zhong et al. [99] focused on the challenging task of generating a video with a new background, such that the new background motion appears compatible with the original one. One common limitation of above works is their lack of attention on the natural and consistent composition of the scene space (geometry-wise). In this case, it is difficult to achieve realistic and deceptive composition result. Another limitation is that most of such works require user interaction to generate realistic result, which makes it hard to be applied to a real-time telepresence system.

Redirected Walking. Redirected walking is the technique that manipulates how we map the user's physical movement to his avatar's position in the virtual environments. This is accomplished by applying a variety of mapping functions (e.g., rotational, translational, or curvature) commonly employed in virtual environments to enable the avatar to "copy" the user's action.

The literature on redirected walking is fairly decent, but often returns to environment-dependent methods. The first successful application of redirected walking appears in 2001 by Razzaque et al. [59], who modeled a virtual hallway much longer than the physical laboratory. The results demonstrated that redirected walking is possible without alerting the user while enabling them to explore virtual spaces much larger than their corresponding physical space without any special hardware. However, the study also had a major limitation, that the users in the study were not allowed to explore the virtual space at will. Instead, they were given an assigned path to follow and were not permitted to deviate from that path. This severely limits the usefulness of the technique, since in most virtual environment scenarios, the user should have some degree of autonomy to explore the space of their own volition.

Then, following successive related studies have been proposed towards more advanced and flexible redirected walking, such as [33, 87, 68, 69]. In 2004, Kuhl [33] conducted a virtual reality study in which subjects' rate of rotation was manipulated. In 2006, Williams et al. [87] explored the effects of applying a fixed translational gain in virtual environments on subjects' perceptual accuracy. This study further reinforced the potential utility of using perceptual manipulations to enable people to explore large virtual environments without constantly needing to manually reorient. Neth et al. [50] explored the implementation of a dynamic curvature mapping algorithm that permitted autonomous exploration by the users and significantly increased how far users could walk before reaching a physical boundary. Sanz et al. [63] investigated

obstacle avoidance behavior during real walking in a large immersive projection setup. They analyzed the walking behavior of users when avoiding real and virtual static obstacles.

One of other important techniques related to redirected walking is manual reorientation, also known as resetting. When the user is about to collide with a physical obstacle in the real world, the program informs the user to stop and reorientation occurs. Williams et al. [88] investigated three different reset algorithms, freeze-back up, freeze-turn, and turn.

Change Blindness. Change blindness algorithm aims to steer users away from physical world obstacles while exploring large virtual environments by exploiting the fact that humans are often poor at noticing changes to their environment when they are not directly observing it. For example, if someone enters a room, and the door behind the person changes position, and when the person turns around, he fails to notice that the door is in a new location. Change blindness is a closely related technique which could potentially be combined with redirected walking to further reduce the number of manual reorientations.

Suma et al. [70] exploited change blindness to allow the user to walk through an immersive virtual environment that is much larger than the available physical workspace. This approach relies on subtle manipulations to the geometry of a dynamic environment model to redirect the user's walking path without becoming noticeable. Study by Suma et al. [71] also proved as an effective proof-of-concept of the change blindness technique. However, like many redirected walking studies, it tightly restricted users' actions, forcing them to enter and exit rooms in a strict sequence. These techniques are powerful, but inflexible. Additionally, change blindness seems difficult to generalize.

Others. Many other fields of techniques are also related to create a realistic composition of 3D scenes in telepresence, such as scene labeling technique and color transfer. In 2014, Chen et al. [9] presented a novel solution to automatic semantic modeling of indoor scenes from a sparse set of low-quality RGB-D images. Such data presents challenges due to noise, low resolution, occlusion and missing depth information.

In 2014, Yamada et al. [93] proposed a novel color transfer method based on spatial structure. This work considers an immersive telepresence system, in which distant users can feel as if

Figure 5.1: Overview of Offline part in our GPU-based scene replacement method: 1)scene adjustment; 2) scene analysis.

they are present at a place other than their true location. The proposed method can offer users an experience that allows them to feel as if they are in the same room as the other participant of telepresence by changing the colors of the remote room to match the colors of the local room.

5.3 Overview

The input to our method is an RGBD video stream of color and depth, capturing from a commodity depth camera. In our implementation, we have tried our method with both Kinect [44] and PrimeSense 3D sensor [57]. Our pipeline can be divided into two parts: the offline part (see Fig. 5.1) and the online part (see Fig. 5.2). The offline part runs as a preprocessing step before the real-time scene replacement, and the purpose of it is to generate a scene database for the system to choose target scene from in the next part. The online part is the main body of our approach, it can automatically segment *actor* from *local scene* and replace it with *target scene* for *viewer* to see. While streaming RGBD data into the GPU, our method performs the abovementioned following stages of computation with CUDA to achieve real-time scene replacement (see Fig. 5.2):

5.3.1 Foreground Extraction

Our method assumes a background-only RGBD video stream at the first few seconds when the system starts. By this, we can construct a background model individually for the color and depth channels of the RGBD video. Then, we employ the high-quality real-time foreground

Figure 5.2: Overview of Online part in our GPU-based scene replacement method: 1) foreground extraction; 2) scene adjustment; 3) scene analysis; 4) scene suggestion; 5) scene matching and 6) rendering.

extraction method we described in Chapter 4 [98] because its result is of high-quality and temporal coherent, and its computation can be parallelized with the GPU (see stage 1 in Fig. 5.2). The output from this stage is an alpha mask that can separate *actor* from *local scene*.

5.3.2 Scene Adjustment

After foreground extraction, we start the main computation pipeline, which is a real-time process with CUDA (see stages 2 to 6 in Fig. 5.2). Among them, we first need a scene adjustment stage (see Fig. 5.2, stage 2) because we need to prepare each scene dataset for better alignment in later stage. Considering that *local scene* and *target scene* are captured with different camera poses, it is better to pre-transform the scene coordinate system based on major planes in the scene. Hence, in this stage, we first adopt a floor detection algorithm to identify major planes in the scene. Then, we slightly adjust some of the planes detected, and transform the current scene coordinate system based on them. The output from this stage is a newly transformed *actor*, a new transformed *local scene* or *target scene*, whose center of floor is the origin of the coordinate system, together with a list of plane objects that describe major planes in *local scene* (see Section 5.4 for details).

5.3.3 Scene Analysis

Next, we analyze the *actor* and the *local scene* with the following steps (see Fig. 5.2, stage 3): First, for the *actor*, we employ standard human skeletal tracking method to identify initial user scenario as *standing*, *sitting* or *meeting*. Then, for the current scene, based on some simple

assumptions, our system automatically searches for supporting planes and furniture volume. Lastly, we further apply polygon fitting and boolean operations to determine the *walkable area* in *local scene* (see Section 5.5 for details).

5.3.4 Scene Suggestion

After analysis of *actor* and *local scene*, we need to search from our scene database and suggest the best candidate target scene that mostly suits *actor* and *local scene* (see Fig. 5.2, stage 4). First of all, we select all candidate scenes that share the same user scenario. Then, we deploy a shape matching algorithm between *walkable area* of *local scene* and *walkable area* of all abovementioned candidate scenes. Upon completion of such matching, we can order all the candidate scenes from “most suited” to “least suited” based on shape context similarity. Lastly, the system adopts the “most suited” candidate scenes as *target scene* for later stages (see Section 5.6 for details).

5.3.5 Scene Matching

Since we already have both *local scene* and *target scene*, we start to match them with each other at this stage (see Fig. 5.2, stage 5). First, we match from *target scene* to *local scene* based on user scenario. Particularly, we first align the main supporting planes in two scenes in case that *actor* is in *meeting* scenario. If *actor* is in *sitting* or *standing* scenario, we align *walkable area* in two scenes first. Next, it comes to our main contribution in this stage: for each position in *walkable area* of *local scene*, we determine its corresponding matching position in *target scene*. Hence, for every movement of *actor* in *local scene*, we can obtain a corresponding movement in *target scene*. This method aims to maximize *actor* physical space flexibly and can be easily parallelly implemented to attain real-time performance with CUDA (see Section 5.7 for details).

5.3.6 Scene Rendering

At last, we place the *actor* in *target scene* based on user scenarios (see Fig. 5.2, stage 6). Then, we need to deploy color transfer method to match colors between these two different scenes.

What's more, we apply rendering technique to avoid the artifact that *actor* is caught in the middle of furnitures. Lastly, the system will automatically choose a perspective to render the composited scene for *viewer* to view (see Section 5.8 for details).

5.3.7 Offline Analysis

The offline analysis part consists of two stages: scene adjustment and scene analysis. The algorithms we apply in these two stages are the same as those in the online part, so we do not go into details here, please refer to Section 5.4 and Section 5.5 for details. But different with the online part, the input of the offline part are multiple single RGBD images of different scenes, instead of a stream of RGBD video. Hence in this part, the system only processes background scenes rather than foreground user. The output of this part is a database of different scenes (eg. office, living room, meeting room, etc.), each of them is with a list of its planes and a label that identifies its user scenario (e.g., *standing*, *sitting*, or *meeting*). All scenes in this database are considered as candidate target scenes at the beginning of the online part.

5.4 Our Approach: Scene Adjustment

To replace the *local scene* with *target scene* while retaining sense of reality, the dominant planes of these two scenes must be aligned, especially for the floor plane (in *standing* scenario) or the major supporting plane (in *meeting* scenario). Considering that *local scene* and *target scene* are captured with different camera poses, and cameras may be mounted at different heights, therefore, we first adopt a floor detection algorithm to identify major planes in the scene. Then, we slightly adjust some of the planes detected, because they can be drifted from their true positions and orientations due to RGBD camera's low resolution.

1) Floor Detection. In this stage, we first adopt the floor detection technique described in Taylor's work [75] to find all planes in the scene. By this method, we can not only determine the position of floor, but also other dominant planes, such as walls. In case of the existence of floor, we adopt it as the *major plane*; otherwise, we adopt the largest plane which is parallel to the ground as the *major plane*. Fig. 5.3 shows examples of our floor detection results.

Figure 5.3: Our Floor Detection Results: a)input rgb image; b)detected planes; c)dominant planes.

2) Plane Adjustment. As we mentioned above, some of the detected planes can be drifted from their true positions and orientations due to RGBD camera's low resolution, see Fig. 5.4 left for example. Hence at this step, we aim to find those planes whose angle between them are close to architectural angle, then adjust them.

In details, once planes are detected, their positions and orientations may be used to define plane objects (e.g., a collection of information about a given plane). Additional information, such as 2D visual features, using SURF and SIFT for example, boundaries, edges, corners, adjacent planes, location and visual appearance of observed points, or other data may be recorded as part of the plane object. This additional information may be determined by projecting 3D scene data and other associated spatial data that is within a particular distance threshold of the plane onto the plane along the dimension of the normal vector to the plane. If multiple plane objects are close to a common architectural angle from each other in orientation (e.g., multiples of 45 degrees such as 0, 45, 90, or 180 degrees), their orientations may be altered slightly in order to get them to match up with the common architectural angle. Methods such as RANSAC may be used to group plane objects with similar normal vectors. These groups may be used to bring the plane objects in the group into alignment with one another.

Furthermore, energy minimization and other optimization methods may be used to alter orientations of many planes or groups of planes at once. The function to be optimized may include penalty terms for changes in the orientations or normal vectors of plane objects or the positions

of points comprising the plane objects, as well as terms based on the angle between orientations or normals of pairs of plane objects. For example, these latter terms may be smaller if the angle between two plane object normals is close to a multiple of 45 degrees such as 0, 45, 90, or 180 degrees, and these terms may be regularized so that only small angular adjustments are preferred. Examples of specific terms in the function may include the L1 or L2 norms, or squared L2 norm, of the angles or sines of the angles between the normal of a plane object before and after alteration, or of the vector difference between the normalized normal vectors before and after alteration, and the regularized L1 or L2 norms, or squared L2 norm, of the differences or sines of the differences between the angles between pairs of two different planes and the preferred angles that are multiples of 45 degrees such as 0, 45, 90, and 180 degrees. An example of the former type of term is $|v - w|^2$ where v is the unit normal vector of the plane before alteration and w is the unit normal vector of the plane after alteration. Another example is $\sqrt{(|\sin(\theta)|^2)}$, where θ is the angle between normal vectors v and w . An example of the latter type of term is $|\sin(4\theta)|$, where θ is the angle between the normals of the two plane objects. The latter term may be capped so that planes that are significantly far from an architectural angle such as a multiple of 45 degrees are not impacted. An example of such a term is $\min(|\sin(4\theta)|, 0.1)$.

Techniques for solving such an optimization problem may include, depending on the exact function chosen, quadratic programming, convex optimization, gradient descent, Leven-berg-Marquardt, simulated annealing, Metropolis-Hastings, combinations of these, or closed-form. The result of such an optimization is a new choice of normal direction for each plane object. The optimization may also be set up to choose a rigid transform of each plane object, and also take into account considerations such as minimizing movement of points in the planes, and movement relative to other planes, boundaries, lines, and other considerations. If two planes (P_1, P_2) with normals (N_1, N_2) respectively) which are close to 90 degrees from each other, in the formula below, vector v_i represents the original normal vector of plane P_i , and w_i represents the proposed new normal vector of plane P_i . θ represents the angle between v_1 and v_2 in the diagram. The v_i terms are simultaneously chosen in an attempt to minimize the sum over all terms in the energy function, including terms not shown in the diagram. Examples of such terms are:

$$\min(|\sin(4\theta)|, 0.1) + |v_1 - w_1|^2 + |v_2 - w_2|^2.$$

Figure 5.4: Our Floor Adjustment Result: left: floors before adjustment; right: floors after adjustment.

Fig. 5.4 right show and example of adjusted planes using algorithms described before. Notice in Fig. 5.4 left, planes detected are clearly not perpendicular to each other; but in Fig. 5.5 right, planes are perpendicular to each other after our adjustment.

3) Coordinate System Transformation. After the above adjustment, next we transform the current coordinate system of the scene to a new coordinate system of which the origin is the center of the *major plane*. The Y axis of the new coordinate system is the normal of the *major plane*, while the X and Z axes are two mutually perpendicular vector on the *major plane*. Fig. 5.5 shows the example of some transformed scene.

5.5 Our Approach: Scene Analysis

The scene analysis stages can be divided into several steps, which we will describe one by one.

1) User Scenario Determination. In this step, we divide user scenarios into different cases by *actor* initial pose, see below and Fig. 5.6 for example.

- *standing*: for online part: *actor* is standing and floor exists as *major plane*; for offline part: floor exists as *major plane*;
- *sitting*: for online part: *actor* is sitting at least 0.5 meters away from camera and floor

Figure 5.5: Transformed Scenes.

Figure 5.6: User Scenarios: left: standing; middle: sitting; right: meeting.

exists as *major plane*; for offline part: floor exists as *major plane* and at least one supporting plane sittable exists;

- *meeting*: for online part: *actor* is sitting not further than 0.5 meters away from camera or a supporting plane exists as *major plane*; for offline part: a supporting plane exists as *major plane*;

For the online part, first, we employ standard human skeletal tracking method described by Microsoft [45] to identify user's initial pose. This method is designed to track people who are seated on a chair or couch, or whose lower body is not entirely visible to the sensor. The default tracking mode, in contrast, is optimized to recognize and track people who are standing and fully visible to the sensor (see Fig. 5.7). After the human pose detection, together with the information of planes detected from last stage, we can assign each scene a label (e.g., *standing*, *sitting*, *meeting*) based on the criteria we mentioned above. What need to emphasized here is

Figure 5.7: Human Skeletal Tracking: left: standing; right: sitting (adapted from [86]).

that, the user scenario labels determined here are not fixed, they could change based on *actor* later movement. Therefore the label here is only for scene suggestion.

2) Walkable Area Determination. For the online part, sometimes in the physical space of the *local scene*, there are areas which human could not step on although they can be seen as floor. Such areas include the underlying floor which is occluded by furniture, for example, the floor under a chair or sofa, see Fig. 5.8 for example. Thus in this step, we find such area and exclude them from the truly walkable area. To do this, we first project 3D points of any supporting planes above the floor plane onto the floor plane. By this, we can obtain the unwalkable area due to each supporting plane. Then, we further apply polygon fitting on both the floor plane and the found unwalkable areas to determine the outlines of them. Now we have one polygon describing the outline of the floor plane, and some other polygons indicating the outlines of unwalkable areas. Lastly, we apply boolean operations on these polygons to determine the *walkable area* in the scene. Fig. 5.9 illustrates how to calculate the walkable area using polygons, given the floor plane and the occupied area of a furniture.

Figure 5.8: Walkable area. Left: front view; middle: top view; right: top view with labels.

Figure 5.9: Determination of walkable area using polygons: left: blue polygon represents the outline of the floor plane while red polygon represents the outline of 2D area occupied by a furniture; right: the calculated final walkable area.

5.6 Our Approach: Scene Suggestion

After analysis of *actor* and *local scene*, we need to search from our scene database and suggest the best candidate target scene that mostly suits *actor* and *local scene*.

1) Filtering by User Scenario. First of all, we select all candidate scenes that share the same

Figure 5.10: The log-polar coordinates we use for shape context (adapted from [2]).

user scenario with the *local scene*.

2) Shape Matching. Then, we deploy a shape matching algorithm between *walkable area* of *local scene* and *walkable area* of each of abovementioned candidate scenes. Here we use *shape context* as descriptor for finding correspondences between pointsets [2]. The basic idea behind *shape context* is illustrated in Fig. 5.10 and Fig. 5.11. Given a set of points from an image (e.g. extracted from a set of detected edge elements), the shape context captures the relative distribution of points in the plane relative to each point on the shape. Specifically, we compute a histogram using log-polar coordinates, as shown in Fig. 5.10. Thus we get descriptors that are similar for homologous (corresponding) points and dissimilar for non-homologous points, as illustrated in Fig. 5.11, where the bin counts in the histogram are indicated by the grayshade (black=large, white=small).

By using the *shape context* as attributes for a weighted bipartite matching problem, we can compute the similarity between each pair of *local scene* and *target scene*. Upon completion of such matching, we can order all the candidate scenes from “most suited” to “least suited” based on shape context similarity. Lastly, the system adopts the “most suited” candidate scenes as *target scene* for later stages.

Figure 5.11: Descriptors are similar for homologous (corresponding) points and dissimilar for non-homologous points (adapted from [2]).

5.7 Our Approach: Scene Matching

Since we already have both *local scene* and *target scene*, we start to match them with each other at this stage.

1) Major Plane Alignment. First, we match from *target scene* to *local scene* by major plane. In case of *meeting* scenario, the *major plane* in two scenes are the largest supporting planes which are parallel to the ground plane. In case of *standing* and *sitting* scenario, the *major planes* in the two scenes are the *walkable area*. In this step, we first align the center of *major planes* in two scenes with each other. Then we rotate the *major plane* of the *target scene* to minimize the distance between it and the *local scene*.

1) Walkable Area Matching. Next, it comes to our main contribution in this stage: for the *standing* and *sitting* scenario, for each position in *walkable area* of *local scene*, we determine its corresponding matching position in *target scene*. Hence, for every movement of *actor* in *local scene*, we can obtain a corresponding movement in *target scene*. To do this, we consider the *walkable areas* as polygons, and overlap a grid with both polygons, as shown in Fig. 5.12. For all position points in a grid cell, they are all denoted by the central position of the cell. By

doing this, we can simplify the matching problem. Then for each position in the *local scene*, our target is to find its corresponding position in the *target scene*. In Fig. 5.12, point O is the center of the *walkable area*; point P_i is one position in the *local scene*, while point Q_i is the corresponding position of point P_i in the *target scene*. For each point P_i , we connect it with the center point O and obtain two intersection point A_i and B_i , with the *local scene* and the *target scene* respectively. If $|OA_i| \geq |OB_i|$, which means *local scene* has longer distance in this direction than *target scene*, then $Q_i = P_i$. If $|OA_i| < |OB_i|$, which means *target scene* has longer distance in this direction than *local scene*, then $\frac{|OP_i|}{|OQ_i|} = \frac{|OA_i|}{|OB_i|}$. In this case, when *actor* moves from P_1 to P_2 in the *local scene*, his/her corresponding moving path in the *target scene* is from Q_1 to Q_2 . This method aims to maximize *actor* physical space flexibly and can be easily parallelly implemented to attain real-time performance with CUDA.

5.8 Our Approach: Scene Rendering

At last, we place the *actor* in *target scene*. In case of the *standing* scenario, we place the *actor* on the center of the major plane. In case of the *sitting* scenario, we place the *actor* on the supporting furniture. In case of the *meeting* scenario, we place the *actor* besides the major plane.

Then, we use simple color transfer method to match colors of *target scene* to *local scene*. After this, we still have one more problem, which is the partial occlusion problem. The partial occlusion problem refers to the artifact that if no additional constraints exist, *actor* can freely walk through any furnitures, which is not possible in real life (see Fig. 5.13 left for example). To solve this problem, we first extract the bottom up volume of each supporting plane (e.g., chair, desk, sofa, etc.), these volume are considered as furniture volumes. Then we use rendering technique to render every furniture volume separately, so that when *actor* come close to a furniture volume, he can only be placed around this furniture, instead of in it. Lastly, the system will automatically choose a perspective that can perceive the face of *actor* to render the composited scene for *viewer* to view.

Figure 5.12: Walkable Area Matching: blue polygon represents the walkable area of *target scene* while red polygon represents the walkable area of *local scene*. Point O denotes the center of the scene; point P_i denotes a position in the *local scene* while point Q_i denotes the corresponding position of point P_i in the *target scene*.

5.9 Results

5.9.1 Implementation Details

We implement and run our CUDA-based foreground extraction method on a 64-bit desktop computer with Intel Six Core 3.2GHz CPU and NVIDIA GeForce GTX 580 1.5GB graphics card. For software, we employ the CUDA 5.0 library to develop the GPU code, and the CULA Sparse S5 library [14], which is a GPU-accelerated library. In addition, we use the OpenNI 2.0 API along with the Kinect driver since it is mostly compatible with GPU parallelization, and it was programmed with CUDA 5.0. For RGBD cameras, our method can work with both

Figure 5.13: Problem of Partial Occlusion: a)human can be rendered in the middle of a furniture which cause partial occlusion problem; b)human can only rendered near furniture avoiding the partial occlusion problem.

Microsoft Kinect 1 [44] and PrimeSense 3D Sensor (Carminie 1.09) [57]. Our resolution setting for the Kinect sensor is 640×480 for both RGB and depth at a frame rate of 30 Hz, while the PrimeSense 3D sensor also has the same resolution for the RGBD video, but operates with a shorter capturing range of 0.35 meters.

5.9.2 Scene Replacement Results

Fig. 5.14 shows the scene replacement results produced from our method on different RGBD videos. These RGBD videos are captured with different participants against different background scenes, which are not simple single-color backgrounds used in green/blue screen methods.

5.9.3 Time Performance

To evaluate the time performance of our method, we test it with four different RGBD videos, each with 250 frames and a resolution of 640×480 . Table 5.1 shows the results: the first eight rows list the average time taken per frame and the corresponding frame rate for each stage in the pipeline, while the last row shows the overall performance for the online portion of the pipeline.

Figure 5.14: Our Scene Replacement Results.

Table 5.1: Time Performance of our CUDA-based pipeline.

	Stage	Average Time (milli.sec/frames)	FPS (frame/sec)
One-time	Scene Adjustment	4.89	204.50
	Scene Analysis	5.77	173.31
Real-time	Foreground Extraction	32.29	30.97
	Scene Adjustment	5.12	195.31
	Scene Analysis	6.60	151.51
	Scene Suggestion	2.88	347.22
	Scene Matching	10.31	96.99
	Scene Rendering	0.99	1010.10
	Total (Real-time)	58.19	17.19

Note that the time needed to transfer video frame data from main memory to GPU and also to transfer the foreground extraction result from GPU back to main memory are included in the timing statistics, respectively. Hence, we can see from the results here that near real-time performance can be achieved by our method. In addition, we would also like to highlight that our method has consistent and stable performance over all the RGBD videos.

5.9.4 Experiments

Since there is no defined metrics to quantitatively evaluate the quality of 3D scene replacement results, we conduct a simple survey to obtain people’s subjective opinions towards our 3D scene replacement results. In this survey, we invite 15 people (among which 10 are women, 5 are men) and show them 8 videos. Each video shows our 3D scene replacement result compared to direct 2D chroma key background replacement result for same dataset. Then we ask them to score, both for our result and the direct 2D replacement result, for each dataset. Score is from 10 to 1, in which 10 represents “as realistic as in real-life”, and 1 represents “unrealistic”. The higher the score is, the more realistic the result is. Then we average the scores from all testers and from all datasets to get an overall score for both our results and the direct replacement result. In this survey, our method scores 80.1 and the direct 2D background replacement method scores 72.9. It means that at least to some extent, our 3D scene replacement has advantage over the traditional 2D method.

5.10 Summary

This chapter has presented a novel CUDA-based approach for automatic real-time 3D scene replacement from RGBD videos. The main challenge here is to seek for realistic composition of foreground human and background scene, while still achieving real-time performance.

To address the challenge, we have developed our 3D scene replacement method by carefully devising the CUDA-based parallel computation. In particular, given an RGBD video stream, our method starts by first identifying the user scenario according to the initial user pose. Then, we adaptively analyze both the user and the background scene to obtain descriptions for them. By this, we can produce matching positions for local scene from suggested target scene. This enables realistic user movement in the composited scene. Compared to the existing methods, our method is fully automatic without any manual markup and more realistic, and it can achieve real-time performance through CUDA-based parallel computation. Lastly, we have also performed an experiment to evaluate our method, particularly comparisons against traditional 2D background replacement method. By this experiment, we can see that our method is capable of compositing more realistic results while achieving real-time performance.

However, this work is still limited by many aspects. The most important one is that the time performance is expected to achieve real-time (30 fps). Optimization on the GPU implementation can be our future work.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

So far in this report, we have presented my research projects towards high-quality 3D telepresence with commodity RGBD camera. First, we explained the importance of high-quality 3D telepresence, and discussed about existing problems in modern 3D telepresence system. Also, we proposed a set of possible solutions, which are described below:

High Quality Depth Filtering Method Tailored for Kinect-like RGBD Camera. In detail, we established the experiment of raw depth data captured from Kinect. The observation from the experiment shows that there are often large irregularly-shaped patches of pixels that receive the same depth value from Kinect. Against this quantization problem we designed a novel filtering method tailored for Kinect-like RGBD camera. The basic idea is multi-scale analysis and direction-aware filtering. In addition, we developed and implemented our method with CUDA, and show that it can efficiently run in parallel on the GPU to filter Kinect depth data in real-time. Moreover, through the comparison between the filtering results from our method and the state-of-the-art technique bilateral filter, we can see that our method is efficient to support our targeted 3D telepresence application.

Real-time and Temporal-coherent Foreground Extraction with Commodity RGBD Camera. In addition, we proposed a novel foreground extraction method with commodity RGBD camera to support real-time 3D telepresence. First, we presented a new method to temporally fill the missing data in the RGBD video with shadow detection, and a new method to combine

the color and depth masks of the foreground object to form a better trimap for our temporal matting; Then, we developed a novel closed-form matting formulation with temporal coherence, and formulate an efficient optimization model that can maintain temporal coherence in the foreground object extraction as well as attain real-time performance; And at last, we developed our method entirely on the GPU by parallelizing most of the computation with CUDA strategies; by this, we can achieve high-quality foreground extraction with real-time performance. Our experiment results show that our approach, compared to several existing state-of-the-art techniques, e.g. FreeCam, can stably extract higher quality foreground objects with higher temporal consistency, and is completely robust against different background scenes and foreground humans. Chapter 4 describes our proposed foreground extraction method to performing real-time extraction of foreground by taking full advantage of various novel techniques and commodity RGBD camera.

Automatic 3D Scene Replacement in real-time for 3D Telepresence. What's more, we have presented a novel CUDA-based approach for automatic real-time 3D scene replacement from RGBD videos. The main challenge here is to seek for realistic composition of foreground human and background scene, while still achieving real-time performance. To address the challenge, we have developed our 3D scene replacement method by carefully devising the CUDA-based parallel computation. This enables realistic user movement in the composited scene. Compared to the existing methods, our method is fully automatic without any manual markup and more realistic, and it can achieve real-time performance through CUDA-based parallel computation. Lastly, we have also performed an experiment to evaluate our method, particularly comparisons against traditional 2D background replacement method. By these experiment, we can see that our method is capable of compositing more realistic results while achieving real-time performance.

6.2 Future Work

After the first three goals of our research projects, our future work will focus on adapting current work to a multi-view 3D telepresence system, and enhancing it by upgrading it to an interactive background replacement telepresence system.

6.2.1 Multi-view 3D Foreground Extraction

A multi-view 3D telepresence system allows a user to explore the 3D space more freely and flexibly, thus enabling more immersive 3D telepresence experience. Therefore it is natural to introduce and incorporate the multi-view feature into our current foreground extraction telepresence system. This makes it possible for one user to view more aspects of his/her remote collaborators.

However, this is a challenging problem. First, real-time performance is still the number one priority. But the multi-view feature, which means multiple foreground extraction at the same time, obviously brings more computational cost and leads to lower performance. Therefore, an optimized version of current method is required.

Second, another problem is how to make the full advantage of the multi-view information. The naive way is to calculate the alpha map from multiple view separately, and then adaptively combine them. But it means the alpha value for the overlapping part will be repeatedly calculated, which makes it less efficient. Therefore, The cost function of the existing optimization could be revised, so that all the multi-view alpha maps could be attained at the same time.

In summary, we plan to devise the current foreground extraction method to adapt to multi-view data. The method should meet below requirements:

- **Real-time Performance.** As we emphasized, real-time performance is essential to 3D telepresence, So the proposed method must be efficient, and can be parallelized on GPU.
- **Adaption to Multi-view Data.** The optimization should include the correlation between pixels that could be projected to same 3D vertex in different views. The result for those pixels should be coherent.

6.2.2 Interactive Background Replacement

6.2.2.1 Modeling Foreground/Background Interaction

Although the 3D replacement of background seems easy and natural by just combining the 3D model of foreground person and background scene together, it is harder than it looks.

The first problem is the color contrast consistency. Since the source of foreground person and background scene might be captured under different lighting conditions or using devices with different parameters, the color contrast between them might be not consistent. This problem can be solved by multiple ways, e.g. histogram equalization and image recoloring. But considering the 3D rendering of the whole scene, more advanced methods, e.g. relighting, are needed.

The second problem is the scale problem. The foreground person and the background scene is captured from different point of view. So when placing the foreground in the scene, it is necessary to measure the physical height of the foreground as well as the size of the objects in the scene to match the scale. And then the foreground is placed at the optimal distance from the point of view of the scene.

The third problem is the ground detection problem. After we confirm the proper scale for the foreground, the vertical position of it is another factor to be considered. A ground detection algorithm is to be performed so that the bottom of the foreground is touching the ground.

Another problem is the occlusion and collision between foreground and objects in the background.

In this project, we plan to build a framework that is capable of identifying, modeling and resolving the above problems.

6.2.2.2 Interactive Background Replacement 3D Telepresence

Besides the 3D representation and the multi-view, the interactive background replacement 3D telepresence has additional two important functions: 1) the participant can replace the physical background by novel virtual environment; 2) the participant can freely interact with the virtual object in the virtual environment.

In this work we first need a smart integration of the previous works to have a complicated and well-functioned 3D telepresence system. we also would like to utilize some interaction techniques as a minor contribution, for example, using gesture to switch the virtual environment or adjusting the lighting of the rendering.

6.2.2.3 Benchmark of Foreground/Background Interaction

Although there are many state-of-the-art 3D replacement techniques, it is quite hard to compare with each other quantitatively due to the missing of a standard evaluation method. we would like to design a benchmark that could be used to evaluate the 3D replacement. The benchmark consists of multiple extracted high-quality foreground and virtual background environment with different layout. Each pair of foreground and background is corresponding to a data set that indicates a list of properties modeled by foreground/background interaction framework as follows:

- **Optimal Color Contrast.** The color contrast distribution of the foreground and the background that makes the most natural look of the whole scene;
- **Optimal Distance.** The optimal distance between the foreground and the viewpoint to make the proper scale;
- **Optimal Position.** The optimal vertical position of the foreground;
- **Collision.** The list that indicates the position of the collision that should be detected over time.

Through this benchmark it should be easier to compare different 3D replacement techniques.

References

- [1] N. Apostoloff and A. Fitzgibbon. Bayesian video matting using learnt image priors. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 407–414, 2004.
- [2] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(4):509–522, 2002.
- [3] Y.Y. Boykov and M.P. Jolly. Interactive graph cuts for optimal boundary region segmentation of objects in n-d images. In *Proceedings of IEEE International Conference on Computer Vision*, volume 1, pages 105–112, 2001.
- [4] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. A non-local algorithm for image denoising. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 60–65, 2005.
- [5] Canesta. Cobra, 2010. <http://www.youtube.com/watch?v5PVx1NbUZQ>.
- [6] V. Caselles, R. Kimmel, and G. Sapiro. Geodesic active contours. In *Proceedings of International Conference on Computer Vision*, pages 694–699, 1995.
- [7] T. F. Chan and L. A. Vese. Active contours without edges. *Trans. Img. Proc.*, 10(2):266–277, 2001.
- [8] Tony F. Chan, B. Yezrielev Sandberg, and Luminita A. Vese. Active contours without edges for vector-valued images. *Journal of Visual Communication and Image Representation*, 11:130–141, 2000.

- [9] Kang Chen, Yu-Kun Lai, Yu-Xin Wu, Ralph Martin, and Shi-Min Hu. Automatic semantic modeling of indoor scenes from low-quality rgb-d data using contextual information. *ACM Transactions on Graphics (TOG)*, 33(6):1–12, 2014.
- [10] Qifeng Chen, Dingzeyu Li, and Chi-Keung Tang. Knn matting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(9):2175–2188, 2013.
- [11] Zhuoyuan Chen, Lifeng Sun, and Shiqiang Yang. Auto-cut for web images. In *Proceedings of the 17th ACM International Conference on Multimedia*, pages 529–532, 2009.
- [12] J.-H. Cho, T. Yamasaki, K. Aizawa, and K.H. Lee. Depth video camera based temporal alpha matting for natural 3d scene generation. In *3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video*, pages 1–4, 2011.
- [13] Yung-Yu Chuang, B. Curless, D.H. Salesin, and R. Szeliski. A bayesian approach to digital matting. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 264–271, 2001.
- [14] CULA. Sparse library, 2013. <http://www.culatools.com/sparse/>.
- [15] M. Date, H. Takada, S. Ozawa, S. Mieda, and A. Kojima. Highly realistic 3d display system for space composition telecommunication. In *Proceedings of IEEE Industry Applications Society Annual Meeting*, pages 1–6, 2013.
- [16] F. Faghih and M. Smith. Combining spatial and scale-space techniques for edge detection to provide a spatially adaptive wavelet-based noise filtering algorithm. *IEEE Transactions on Image Processing*, 11(9):1062–1071, 2002.
- [17] Jialue Fan, Xiaohui Shen, and Ying Wu. Scribble tracker: A matting-based approach for robust tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(8):1633–1644, 2012.
- [18] XiaoGuang Feng and P. Milanfar. Multiscale principal components analysis for image local orientation estimation. In *Conference Record of Asilomar Conference on Signals, Systems and Computers*, volume 1, pages 478–482, 2002.

REFERENCES

- [19] Jingjing Fu, Shiqi Wang, Yan Lu, Shipeng Li, and Wenjun Zeng. Kinect-like depth denoising. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 512–515, 2012.
- [20] V. Garcia, E. Debreuve, and M. Barlaud. Fast k nearest neighbor search using GPU. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–6, 2008.
- [21] Minglun Gong, Liang Wang, Ruigang Yang, and Yee-Hong Yang. Real-time video matting using multichannel poisson equations. In *Proceedings of Graphics Interface*, pages 89–96, 2010.
- [22] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In *Proceedings of Annual Conference on Computer Graphics and Interactive Techniques*, pages 43–54, 1996.
- [23] M. Gross, S. Würmlin, M. Naef, E. Lamboray, C. Spagno, A. Kunz, E. Koller-Meier, T. Svoboda, L. Van Gool, S. Lang, K. Strehlke, A. W. Moere, and O. Staadt. Blue-c: A spatially immersive display and 3D video portal for telepresence. *ACM Transactions on Graphics*, 22(3):819–827, 2003.
- [24] R. A. Heinlein. Active contours without edges. *Astounding Magazine*, 1942.
- [25] MESA Imaging. Swissranger 4500. <http://www.mesa-imaging.ch/swissranger4500.php>.
- [26] Adobe Systems Incorp. Adobe photoshop user guide, 2002.
- [27] Bernd Jahne. *Spatio-Temporal Image Processing: Theory and Scientific Applications*. 1993.
- [28] A. Jones, M. Lang, G. Fyffe, X. Yu, J. Busch, I. McDowall, M. Bolas, and P. Debevec. Achieving eye contact in a one-to-many 3D video teleconferencing system. *ACM Transactions on Graphics*, 28(3):64:1–64:8, 2009.
- [29] C.R. Jung. Efficient background subtraction and shadow removal for monochromatic video sequences. *IEEE Transactions on Multimedia*, 11(3):571–577, 2009.

- [30] P. Kaewtrakulpong and R. Bowden. An improved adaptive background mixture model for realtime tracking with shadow detection. In *Proceedings of European Workshop on Advanced Video Based Surveillance Systems*, 2001.
- [31] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.
- [32] Johannes Kopf, Michael F. Cohen, Dani Lischinski, and Matt Uyttendaele. Joint bilateral upsampling. *ACM Transactions on Graphics*, 26(3), 2007.
- [33] S. Kuhl. Recalibration of rotational locomotion in immersive virtual environments. In *Proceedings of ACM SIGGRAPH Symposium on Applied Perception in Graphics and Visualization (APGV)*, pages 23–26, 2004.
- [34] C. Kuster, T. Popa, C. Zach, C. Gotsman, and M. Gross. FreeCam: A hybrid camera system for interactive free-viewpoint video. In *Proceedings of Vision, Modeling, and Visualization*, 2011.
- [35] Anat Levin, Dani Lischinski, and Yair Weiss. A closed form solution to natural image matting. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 61–68, 2006.
- [36] Zongmin Li, Liangliang Zhong, and Yujie Liu. Efficient foreground layer extraction in video. In *Proceedings of Pacific Rim Conference on Advances in Multimedia Information Processing*, pages 319–329, 2010.
- [37] Junyi Liu, Xiaojin Gong, and Jilin Liu. Guided inpainting and filtering for kinect depth maps. In *International Conference on Pattern Recognition*, pages 2055–2058, 2012.
- [38] Ming Liu, Shifeng Chen, and Jianzhuang Liu. Precise object cutout from images. In *Proceedings of ACM International Conference on Multimedia*, pages 623–626, 2008.
- [39] Xian-Ming Liu, Changhu Wang, Hongxun Yao, and Lei Zhang. The scale of edges. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 462–469, 2012.
- [40] A. Maimone and H. Fuchs. Real-time volumetric 3d capture of room-sized scenes for telepresence. In *3DTV-Conference: The True Vision - Capture, Transmission and Display of 3D Video*, pages 1–4, 2012.

REFERENCES

- [41] A. Maimone and H. Fuchs. Reducing interference between multiple structured light depth sensors using motion. In *IEEE Virtual Reality Short Papers and Posters*, pages 51–54, 2012.
- [42] Andrew Maimone, Jonathan Bidwell, Kun Peng, and Henry Fuchs. Augmented reality: Enhanced personal autostereoscopic telepresence system using commodity depth cameras. *Comput. Graph.*, 36(7):791–807, 2012.
- [43] Andrew Maimone and Henry Fuchs. Encumbrance-free telepresence system with real-time 3d capture and display using commodity depth cameras. In *Proceedings of IEEE International Symposium on Mixed and Augmented Reality*, pages 137–146, 2011.
- [44] Microsoft. Kinect, 2010. <http://www.xbox.com/Kinect>.
- [45] Microsoft. Kinect skeletal tracking, 2012. <https://msdn.microsoft.com/en-us/library/jj131025.aspx>.
- [46] M. Minsky. Telepresence. *Omni*, 1980.
- [47] Yasushi Mishima. Soft edge chroma-key generation based upon hexoctahedral color space, 1994. U.S. Patent 5355174.
- [48] Eric N. Mortensen and William A. Barrett. Intelligent scissors for image composition. In *Proceedings of Annual Conference on Computer Graphics and Interactive Techniques*, pages 191–198, 1995.
- [49] M. Narayana, A. Hanson, and E. Learned-Miller. Background modeling using adaptive pixelwise kernel variances in a hybrid feature space. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2104–2111, 2012.
- [50] C. T. Neth, J. L. Souman, D. Engel, U. Kloos, H. H. Bulthoff, and B. J. Mohler. Velocity-dependent dynamic curvature gain for redirected walking. *IEEE Transactions on Visualization and Computer Graphics*, 18(7):1041–1052, 2012.
- [51] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Proceedings of IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, 2011.

- [52] C.V. Nguyen, S. Izadi, and D. Lovell. Modeling kinect sensor noise for improved 3d reconstruction and tracking. In *International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission*, pages 524–530, 2012.
- [53] Panasonic. D-imager. <http://pewa.panasonic.com/components/built-in-sensors/3d-image-sensors/d-imager/>.
- [54] Bo Peng, Lei Zhang, and Jian Yang. Iterated graph cuts for image segmentation. In *Proceedings of the 9th Asian Conference on Computer Vision*, pages 677–686, 2010.
- [55] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639, 1990.
- [56] L. Petrescu, A. Morar, F. Moldoveanu, and A. Moldoveanu. Kinect depth inpainting in real time. In *International Conference on Telecommunications and Signal Processing (TSP)*, pages 697–700, 2016.
- [57] PrimeSense. Carmine 3D sensor, 2013. <http://www.primesense.com/solutions/3d-sensor>.
- [58] R. Raskar, G. Welch, M. Cutts, A. Lake, L. Stesin, and H. Fuchs. The office of the future: A unified approach to image-based modeling and spatially immersive displays. In *Proceedings of Annual Conference on Computer Graphics and Interactive Techniques*, pages 179–188, 1998.
- [59] Sharif Razzaque, Zachariah Kohn, and Mary C. Whitton. Redirected walking. In *Proceedings of Eurographics*, volume 9, pages 105–106, 2001.
- [60] I. Reisner-Kollmann and S. Maierhofer. Consolidation of multiple depth maps. In *IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 1120–1126, 2011.
- [61] Christian Richardt, Carsten Stoll, Neil A. Dodgson, Hans-Peter Seidel, and Christian Theobalt. Coherent spatiotemporal filtering, upsampling and rendering of rgbz videos. *Comp. Graph. Forum*, 31(21):247–256, 2012.
- [62] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. ”grabcut”: interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics*, 23(3):309–314, 2004.

REFERENCES

- [63] Fernando Argelaguet Sanz, Anne-Hlne Olivier, Gerd Bruder, Julien Pettr, and Anatole Lcuyer. Virtual proxemics: Locomotion in the presence of obstacles in large immersive projection environments. In *Proceedings of IEEE Virtual Reality Conference(VR)*, 2015.
- [64] Oliver Schall, Alexander Belyaev, and Hans-Peter Seidel. Feature-preserving non-local denoising of static and time-varying range data. In *Proceedings of ACM Symposium on Solid and Physical Modeling*, pages 217–222, 2007.
- [65] P. Scheunders and J. Sijbers. Multiscale anisotropic filtering of color images. In *Proceedings of International Conference on Image Processing*, volume 3, pages 170–173, 2001.
- [66] S. Schuon, C. Theobalt, J. Davis, and S. Thrun. High-quality scanning using time-of-flight depth superresolution. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–7, 2008.
- [67] Alvy Ray Smith and James F. Blinn. Blue screen matting. In *Proceedings of Annual Conference on Computer Graphics and Interactive Techniques*, pages 259–268, 1996.
- [68] F. Steinicke, G. Bruder, J. Jerald, H. Frenz, and M. Lappe. Analyses of human sensitivity to redirected walking. In *Proceedings of ACM Symposium on Virtual Reality Software and Technology*, pages 149–156, 2008.
- [69] F. Steinicke, G. Bruder, J. Jerald, H. Frenz, and M. Lappe. Estimation of detection thresholds for redirected walking techniques. *IEEE Transactions on Visualization and Computer Graphics*, 16(1):17–27, 2010.
- [70] E. A. Suma, S. Clark, S. L. Finkelstein, and Z. Wartell. Exploiting change blindness to expand walkable space in a virtual environment. In *Proceedings of IEEE Virtual Reality Conference (VR)*, pages 305–306, 2010.
- [71] E. A. Suma, S. Clark, D. Krum, S. Finkelstein, M. Bolas, and Z. Warte. Leveraging change blindness for redirection in virtual environments. In *Proceedings of IEEE Virtual Reality Conference (VR)*, pages 159–166, 2011.
- [72] Jian Sun, Weiwei Zhang, Xiaoou Tang, and Heung-Yeung Shum. Background cut. In *Proceedings of European conference on Computer Vision*, volume 2, pages 628–641, 2006.

- [73] Yu-Wing Tai, Wai-Shun Tong, and Chi-Keung Tang. Simultaneous image denoising and compression by multiscale 2d tensor voting. In *International Conference on Pattern Recognition*, volume 3, pages 818–821, 2006.
- [74] Zhen Tang, Zhenjiang Miao, Yanli Wan, and Jia Li. Automatic foreground extraction for images and videos. In *IEEE International Conference on Image Processing*, pages 2993–2996, 2010.
- [75] C. J. Taylor and A. Cowley. Parsing indoor scenes using rgb-d imagery. *Robotics: Science and System*, 8:401–408, 2013.
- [76] T.Deng, H.Li, J.Cai, T.-J.Cham, and H.Fuchs. Kinect shadow detection and classification. In *Proceedings of ICCV Workshop: Big Data in 3D Computer Vision*, 2013.
- [77] Alexandru Telea. An image inpainting technique based on the fast marching method. *Journal of Graphics Tools*, 9(1):25–36, 2004.
- [78] Zhiqiang Tian, Jianru Xue, Nanning Zheng, Xuguang Lan, and Ce Li. 3d spatio-temporal graph cuts for video objects segmentation. In *IEEE International Conference on Image Processing*, pages 2393–2396, 2011.
- [79] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *International Conference on Computer Vision*, pages 839–846, 1998.
- [80] Z. Tomori, R. Gargalik, and I. Hrmo. Active segmentation in 3d using kinect sensor. In *WSCG*, 2012.
- [81] Chao Wang, Qiong Yang, Mo Chen, Xiaou Tang, and Zhongfu Ye. Progressive cut. In *Proceedings of Annual ACM International Conference on Multimedia*, pages 251–260, 2006.
- [82] Jue Wang, Pravin Bhat, R. Alex Colburn, Maneesh Agrawala, and Michael F. Cohen. Interactive video cutout. *ACM Transactions on Graphics*, 24(3):585–594, 2005.
- [83] Jue Wang and Michael F. Cohen. Image and video matting: a survey. *Found. Trends. Comput. Graph. Vis.*, 3(2):97–175, 2007.

REFERENCES

- [84] Liang Wang, Minglun Gong, Chenxi Zhang, Ruigang Yang, Cha Zhang, and Yee-Hong Yang. Automatic real-time video matting using time-of-flight camera and multichannel poisson equations. *Int. J. Comput. Vision*, 97(1):104–121, 2012.
- [85] Liang Wang, Chenxi Zhang, Ruigang Yang, and Cha Zhang. Tofcut: Towards robust real-time foreground extraction using a time-of-flight camera. In *3DPVT*, 2010.
- [86] Web. Image materials.
- [87] B. Williams, G. Narasimham, T. P. McNamara, T. H. Carr, J. J. Rieser, and B. Bodenheimer. Updating orientation in large virtual environments using scaled translational gain. In *Proceedings of Symposium on Applied Perception in Graphics and Visualization*, pages 21–28, 2006.
- [88] B. Williams, G. Narasimham, B. Rump, T. P. McNamara, T. H. Carr, J. Rieser, and B. Bodenheimer. Exploring large virtual environments with an hmd when physical space is limited. In *Proceedings of Symposium on Applied Perception in Graphics and Visualization*, pages 41–48, 2007.
- [89] Andrew P. Witkin. Scale-space filtering: A new approach to multi-scale description. In *IEEE International Conference on ICASSP Acoustics, Speech, and Signal Processing*, volume 9, pages 150–153, 1984.
- [90] W. Wu, Z. Yang, I. Gupta, and K. Nahrstedt. Towards multi-site collaboration in 3D tele-immersive environments. In *International Conference on Distributed Computing Systems*, pages 647–654, 2008.
- [91] Chunxia Xiao, Meng Liu, Donglin Xiao, Zhao Dong, and Kwan-Liu Ma. Fast closed-form matting using a hierarchical data structure. *IEEE Transactions on Circuits and Systems for Video Technology*, 24(1):49–62, 2014.
- [92] H. Xue, S. Zhang, and D. Cai. Depth image inpainting: Improving low rank matrix completion with low gradient regularization. *IEEE Transactions on Image Processing*, 26(9):4311–4320, 2017.
- [93] Kentaro Yamada, Hiroshi Sankoh, and Sei Naito. Color transfer based on spatial structure for telepresence. In *Proceedings of the ACM International Conference on Multimedia (MM)*, pages 1217–1220, 2014.

- [94] Z. Yang, W. Wu, K. Nahrstedt, G. Kurillo, and R. Bajcsy. Enabling multi-party 3d tele-immersive environments with viewcast. *ACM Trans. Multimedia Comput. Commun. Appl.*, 6(2):7:1–7:30, 2010.
- [95] Yu Yu, Yonghong Song, Yuanlin Zhang, and Shu Wen. A shadow repair approach for kinect depth maps. In *Proceedings of the 11th Asian conference on Computer Vision*, volume 4, pages 615–626, 2012.
- [96] Guofeng Zhang, Jiaya Jia, Wei Hua, and Hujun Bao. Robust bilayer segmentation and motion/depth estimation with a handheld camera. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(3):603–617, 2011.
- [97] Juyong Zhang, Jianmin Zheng, and Jianfei Cai. A diffusion approach to seeded image segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2125–2132, 2010.
- [98] M. Zhao, C. W. Fu, J. Cai, and T. J. Cham. Real-time and temporal-coherent foreground extraction with commodity rgb-d camera. *IEEE Journal of Selected Topics in Signal Processing*, 9(3):449–461, 2015.
- [99] Fan Zhong, Song Yang, Xueying Qin, Dani Lischinski, Daniel Cohen-Or, and Baoquan Chen. Slippage-free background replacement for hand-held video. *ACM Transactions on Graphics (TOG)*, 33(6):199:1–199:11, 2014.