Efficient and Accurate Stereo Matching for Cloth Manipulation

Tian Xu

Submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy

School of Computing Science

College of Science and Engineering
University of Glasgow

December 2015

Abstract

Due to the recent development of robotic techniques, researching robots that can assist in everyday household tasks, especially robotic cloth manipulation has become popular in recent years. Stereo matching forms a crucial part of the robotic vision and aims to derive depth information from image pairs captured by the stereo cameras. Although stereo robotic vision is widely adopted for cloth manipulation robots in the research community, this remains a challenging research task. Robotic vision requires very accurate depth output in a relatively short timespan in order to successfully perform cloth manipulation in real-time.

In this thesis, we mainly aim to develop a robotic stereo matching based vision system that is both efficient and effective for the task of robotic cloth manipulation. *Effectiveness* refers to the accuracy of the depth map generated from the stereo matching algorithms for the robot to grasp the required details to achieve the given task on cloth materials while *efficiency* emphasizes the required time for the stereo matching to process the images.

With respect to *efficiency*, firstly, by exploring a variety of different hardware architectures such as multi-core CPU and graphic processors (GPU) to accelerate stereo matching, we demonstrate that the parallelised stereo-matching algorithm can be significantly accelerated, achieving $12\times$ and $176\times$ speed-ups respectively for multi-core CPU and GPU, compared with SISD (Single Instruction, Single Data) single-thread CPU.

In terms of *effectiveness*, due to the fact that there are no cloth based testbeds with depth map ground-truths for evaluating the accuracy of stereo matching performance in this context, we created five different testbeds to facilitate evaluation of stereo matching in the context of cloth manipulation. In addition, we adapted a guided filtering algorithm into a pyramidical stereo matching framework that works directly for unrectified images, and evaluate its accuracy utilizing the created cloth testbeds. We demonstrate that our proposed approach is not only efficient, but also accurate and suits well to the characteristics of the task of cloth manipulations. This also shows that rather than relying on image rectification, directly applying stereo matching to unrectified images is effective and efficient.

Finally, we further explore whether we can improve efficiency while maintaining reasonable accuracy for robotic cloth manipulations (i.e. trading off accuracy for efficiency). We use a foveated matching algorithm, inspired by biological vision systems, and found that it is effective in trading off accuracy for efficiency, achieving almost the same level of accuracy for both cloth grasping and flattening tasks with two to three fold acceleration. We also demonstrate that with the robot we can use machine learning techniques to predict the optimal foveation level in order to accomplish the robotic cloth manipulation tasks successfully and much more efficiently.

To summarize, in this thesis, we extensively study stereo matching, contributing to the long-term goal of developing effective ways for efficient whilst accurate robotic stereo matching for cloth manipulation.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my PhD advisor Dr. Paul Cockshott for the continuous support of my Ph.D study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Ph.D study. In addition, I would give a big thank to Dr. Paul Siebert for his continuous support during my PhD.

I would like to thank Mr. Marco Starace for giving me the chance for collaborating with Sumsung Research and Development Institute UK and thank Sumsung colleagues for a warm hospitality in London. Thanks for offering me the summer internship opportunities in their groups and leading me working on diverse exciting projects. A special thank to Dr. Tommaso Maestri for his mentoring and valuable research discussions during my internship.

I could not have survived living here without a great support from the Glasgow and ex-Glasgow teams. I thank my good friends Mr. Li Sun and Ms. Susanne Oehler for all their support during my PhD. I learned a lot from you guys. Thanks to Dr. Yuan Liu, Mrs. Maha Maabar, Ms. Samantha Mulholland and many others, as I cherish all the lunch time chat and evening fun with you guys during my PhD years. I also thank my fellow labmates in the Glasgow Computer Vision Group: Dr. Gerardo Aragon Camarasa, Dr. Youssef Gdura, Dr. Paul Keir and others, for the stimulating discussions we have had in the last few years.

Before joining Glasgow vision group, I tried my first steps in doing research with my mentor Prof. Yanwu Wang at Huazhong University of Science and Technology. I thank Prof. Yanwu Wang, in particular for encouraging me to pursue a PhD.

Finally, I would like to thank my family for their love and moral support, and especially my parents and Ke Zhou for inspiration and always being there when I needed.

Author's Declaration

This thesis presented a work that was carried at the University of Glasgow under the supervision of Dr. Paul Cockshott, School of Computing Science, during the period between October 2011 to December 2015. I declare that this thesis is entirely my own work and it has not been previously submitted for any other degree or qualification in any university.

Tian Xu

Dedication to my parents and Ke!

Table of Contents

List of Tables

List of Figures

Part I

Introduction and Background

Chapter 1

Introduction

Computer vision deals with acquiring, processing, analyzing, and understanding images. This field has been actively developed for many decades [Szeliski, 2010] and it aims to duplicate the abilities of human vision by electronically perceiving and understanding an image. Due to the fast development of various image processing algorithms, many real-world applications have benefited and been made possible, including object recognition (such as face [Zhao et al., 2003], gesture [Mitra and Acharya, 2007], fingerprint [Maltoni et al., 2009] or optical character [Mori et al., 1999]), 3D modeling [Henry et al., 2010], motion capture [Moeslund et al., 2006] and many more[1].

One of the most important computer vision algorithms is *stereo matching*, which aims to obtain a depth map from images [Kanade and Okutomi, 1994]. The attempts to solve this problem can be traced back to several decades ago [Barnard and Fischler, 1982]. Many problems of scene analysis arise due to inherent depth ambiguities in a monocular 2-D image. Humans easily perceive depth in 2-D images, but the process of doing so is not well understood. Stereopsis provides a direct way of inferring the 3-D range, and is heavily used by the human visual system. Stereo matching is the process of computing a three-dimensional reconstruction of the scene, given a set of images taken from different viewpoints. This task involves automatically finding matching pixels and ultimately a dense disparity or depth map from a pair of images under known camera configuration. An example of stereo matching system on a scene with paper cones is presented in Figure 1.1. This illustrates that for each surface point visible in both images, there is a ray in 3D space connecting the surface point (cone tips) with each camera's centre of projection.

The problem of stereo matching has been studied intensively over several decades, with many effective algorithms developed [Kanade and Okutomi, 1994][Sun et al., 2003][Yoon and Kweon, 2006]. Especially, the research on stereo matching advanced more rapidly as a result of publicly available performance evaluations such as the Middlebury library [Scharstein

[1] A list of such example applications can be found in http://www.cs.ubc.ca/~lowe/vision.html.

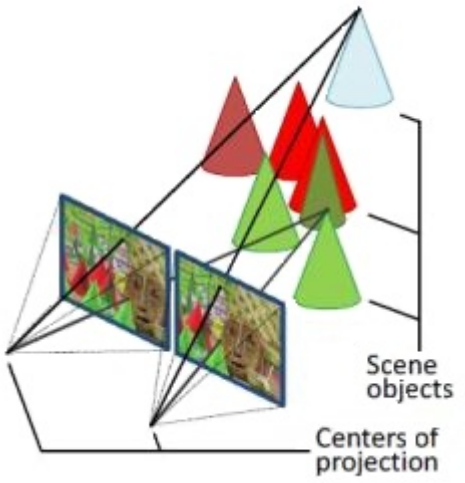


Figure 1.1: Stereo matching example: this illustrates an example of "cones" where a scene with paper cones is imaged with a stereo camera where the projection rays of two cone tips into both camera images are drawn.

and Szeliski, 2002], which allow researchers to compare new algorithms with the current state-of-the-art with little effort. Depth data derived from stereo has been used for applications in a variety of domain, such as passive navigation [Gennery, 1980], cartography [Kelly et al., 1977] and surveillance [Hengstler et al., 2007]. For example, given a large enough set of views of a particular object or facade, we can create accurate dense 3D surface models using stereo matching [Goesele et al., 2007]. Other applications of stereo matching include head and gaze tracking [Matsumoto and Zelinsky, 2000], as well as depth-based background replacement [Gordon et al., 1999].

Stereo matching has been also widely used for robotic vision systems to perceive the world. Widespread adoption of autonomous robots demands high robustness to environmental change and minimal reliance on application specific knowledge. Rich sensing modalities such as vision plays a central role in their success. One common class of robotic vision system is constructed by two cameras with depth information extracted by stereo matching algorithms. Stereo vision has been integrated with various mobile robots [Murray and Little, 2000] due to its various advantages: its reliability and effectiveness in extracting range information from the environment with low-cost hardware and near real-time implementation; its passive nature that does not interfere with other sensors; and its ease of integration with other tasks relying on obtained stereo depth information (such as object recognition and tracking).

More recently, researching robots that can assist in everyday household tasks is becoming popular. Robotic cloth manipulation (as our main research context) has become a trendy research area in recent years. For example, the UC Berkeley robotic group attempted to teach a robot to solve the laundry problem, e.g. folding towels and sorting socks [2]. Across

[2] http://www.eecs.berkeley.edu/~pabbeel/personal_robotics.html

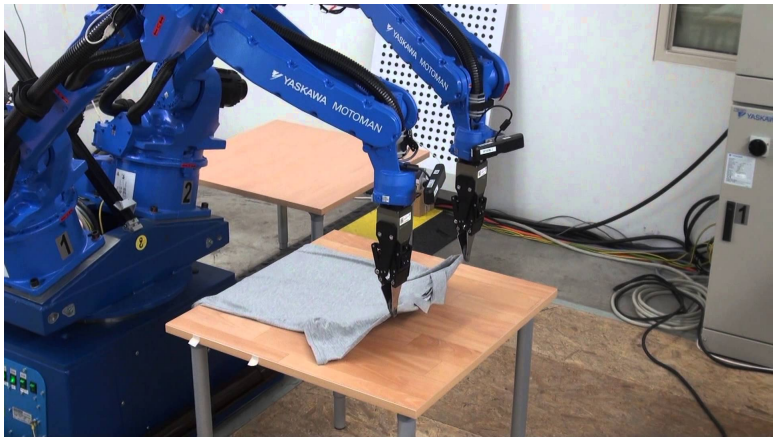


Figure 1.2: Example: robotic cloth manipulation system for grasping and folding cloth (CloPeMa project).

five universities in Europe, the CloPeMa project[3] (Clothes Perception and Manipulation) was launched in 2012 and the project aims to advance the state of the art in the autonomous perception and manipulation of all kinds of fabrics, textiles and garments. In the robotic cloth manipulation research community, various attempts have been made to enable a robot to *grasp* [Maitin-Shepard et al., 2010, Ramisa et al., 2012], *flatten* [Cusumano-Towner et al., 2011, Doumanoglou et al., 2014b, Sun et al., 2015], *fold* [Bersch et al., 2011, Van Den Berg et al., 2011] or *unfold* [Willimon et al., 2011a] clothes with wrinkles on them. An example of the robotic cloth grasping in the CloPeMa project is shown in the Figure 1.2. One common challenge of all those tasks is to develop a robotic vision system that is able to perceive the clothes with sufficient detail, to allow the robotic components to manipulate them (e.g. grasping the cloth wrinkles).

Although stereo robotic vision is widely adopted in the cloth manipulation context in the research community [Cusumano-Towner et al., 2011][Sun et al., 2015][Bersch et al., 2011], this remains a challenging research task. Robotic vision requires very *accurate* depth output in a relatively *short timespan* in order to successfully perform cloth manipulations in real-time. Despite the effectiveness of image stereo matching for various tasks, the usage of this algorithm in practice is still largely limited due to its *efficiency* issue, especially in real-time tasks. Generally, stereo matching is inherently computationally expensive, originating from the large amount of data required to be processed and the complexity of calculation at each pixel. Firstly, the image usually contains millions of pixels in order to capture sufficient details (e.g. images size used in CloPeMa project is 4928×3264 pixels [Aragon-Camarasa et al., 2013]). This means that the stereo matching algorithms need to cope with millions of pixels in a very short time. Secondly, since stereo matching aims at generating accurate disparity map, to deal with noise and avoid inaccurate estimation, a large amount of operations are required for every single pixel.

[3]http://http://www.clopema.eu/

On the other hand, the cloth manipulation robotic vision system is required to capture very *accurate* and detailed representation of the clothes for successfully accomplishing many manipulation tasks. For example, for the task of cloth flattening (i.e. stretching out cloth lying on a flat surface) [Sun et al., 2015], the robotic vision system needs to detect any wrinkles that are larger than 5mm in order to flatten all the cloth wrinkles. Therefore, how to develop accurate stereo matching that is suitable for the cloth materials (e.g. various continuous garment surfaces without drastic depth change) is still an open question.

1.1 Thesis Statement

The broad question that motivates the research in this thesis is: *Can we develop a robotic stereo matching based vision system that is both sufficiently efficient and accurate for the task of robotic cloth manipulation?* As we described earlier, stereo matching has been extensively investigated in the past [Scharstein and Szeliski, 2002] and such techniques have been widely exploited for robotic vision in various tasks such as autonomous driving [Geiger et al., 2012], robotic object manipulation [Kragic et al., 2005] and robotic towel folding [Maitin-Shepard et al., 2010]. However, in the context of robotic cloth manipulation, how to *effectively* and *efficiently* apply stereo matching for cloth materials is still an open question. Here, *effectiveness* refers to the accuracy of the depth map generated from the stereo matching algorithms for the robot to grasp the required details to accomplish the given task on cloth materials while *efficiency* emphasizes the required time for the stereo matching to process the images.

This thesis aims to extensively study stereo matching from both the above two aspects, contributing to the long-term goal of developing ways for efficient whilst accurate robotic stereo matching for cloth manipulation. This consists of accelerating the stereo matching algorithms using various hardware architectures; developing accurate stereo matching algorithms and an evaluation methodology that fits well with robotic cloth manipulation tasks, and optimally trading off accuracy for efficiency in order to successfully accomplish the task within much less time. Part of this work has contributed to the CloPeMa project and the stereo matching algorithms described in this thesis have been proved in practice to be very effective and efficient for a variety of different cloth manipulation tasks (such as cloth flattening [Sun et al., 2015], cloth classification [Sun et al., 2016]), and may be utilized to further improve other applications (such as cloth unfolding [Doumanoglou et al., 2014a]).

1.2 Research Outline and Questions

The general framework of this thesis is depicted in Figure 1.3. This general framework consists of two parts: stereo matching and robotic cloth manipulation. The former deals with

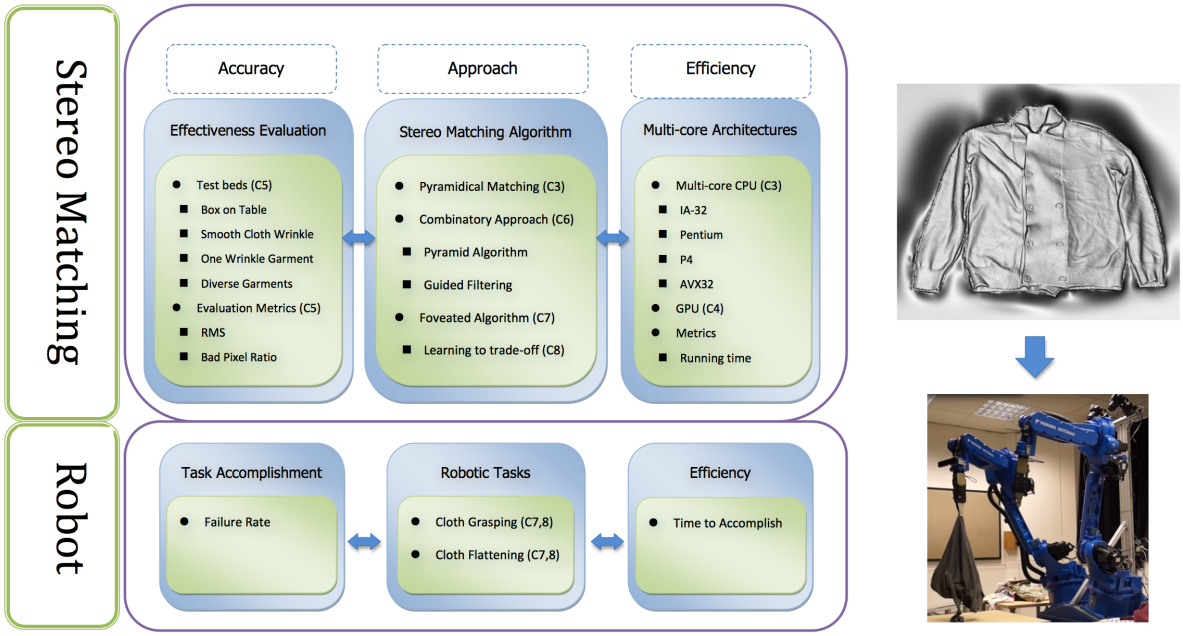


Figure 1.3: General Overview of Stereo Matching for Robotic Cloth Manipulations in this thesis (e.g. C3 means Chapter 3)

obtaining the depth map of the garment while the latter addresses how to utilize the stereo matching output for the robot to effectively manipulate the clothes. The main focus of the thesis is to mainly study how to obtain accurate stereo matching for cloth images efficiently, but as an application, we also investigate how the performance of stereo matching affect the robotic manipulation tasks, especially in terms of task accomplishment success rate and efficiency (time).

The research work in this thesis is mainly organized into three parts:

- **(Efficiency)** how to accelerate stereo matching algorithms using various hardware architectures;
- **(Accuracy)** how to evaluate stereo matching for cloth images and can we propose better stereo matching algorithms that well fit for cloth characteristics;
- **(Trade-off Accuracy for Efficiency)** how to trade-off stereo matching accuracy for efficiency for robotic cloth manipulation so that the robot can successfully accomplish the given task with least time.

We briefly summarize the findings of each of those three parts respectively. With respect to *efficiency*, firstly, by exploring a variety of different hardware architectures such as multi-core CPU and graphic processors (GPU) to accelerate stereo matching, we demonstrate that the parallelised stereo-matching algorithm can be significantly accelerated, achieving $12\times$ and $176\times$ speed-up respectively for multi-core CPU (64-core AMD Opteron 6366HE) and GPU

(Nvidia GeForce GTX 770), compared with same CPU but only use single-thread with SISD (Single Instruction, Single Data). This experiment demonstrates the average performance based on Glasgow Stereo Image Database of Garments [Aragon-Camarasa et al., 2013]. Vector Pascal [Cockshott, 2004][Cockshott, 2011] and CUDA C are the programming languages used for implementation for CPU and GPU respectively. Moreover, to analyze the origin of the speed-up and gain deeper understanding about the choice of the optimal hardware, the stereo matching algorithm is broken into key sub-tasks and the efficiency performance of those sub-components is tested for different hardware architectures.

In terms of *accuracy*, due to the fact that there are no cloth based testbeds with depth map ground-truths for evaluating the accuracy of stereo matching performance in this context, we create five different testbeds to facilitate evaluation of stereo matching in the context of cloth manipulation. In addition, we propose to adapt a guided filtering algorithm into the pyramidical stereo matching framework that works directly for unrectified images without image rectification, and evaluate its accuracy performance utilizing the created cloth testbeds. Note that image rectification is a transformation process that used to project stereo images onto a common image plane, so that the correspondence points have the same y-direction coordinates. This could essentially simplify the 2D stereo correspondence problem to 1D and various algorithms have been proposed to rectify images [Tsai, 1987][Zhang, 2000]. Nevertheless, we demonstrate that our proposed approach for unrectified images is not only efficient, but also accurate and suits well the characteristics of the task of cloth manipulation. This shows that rather than relying on image rectification, directly applying stereo matching to the unrectified images is also effective and efficient.

With respect to the *accuracy-efficiency trade off*, in addition to accelerating stereo matching algorithm using current multi-core hardware, we further explore whether we can improve the efficiency while maintaining reasonable accuracy for robotic cloth manipulation. We propose to use a foveated matching algorithm, an algorithm inspired by biological vision systems, and found that it is effective in trading off accuracy for efficiency for stereo matching. By using the robotic behavior from previous work for two common robotic manipulation tasks, i.e. cloth grasping and flattening, we demonstrate that using foveated matching with an optimal selection of its foveation level can achieve the same level of accuracy for completing both tasks with a speedup of a factor of two to three. Furthermore, we aim to evaluate this foveated stereo matching algorithm with the real robot and test its real robotic cloth manipulation performance. By extracting various very simple visual features from the low-resolution cloth images very efficiently, we demonstrate that we can use machine learning techniques to predict the optimal foveation level in order to accomplish the robotic cloth manipulation tasks successfully (above certain accuracy level) and most efficiently. We apply this learned foveated stereo matching algorithm on simulated robotic grasping task and demonstrate its effectiveness, with the acceleration of stereo matching by almost three times

for grasping and accomplish the tasks at over 90% accuracy.

Given the above overview, next we respectively describe in details each of the research questions.

1.2.1 On the Acceleration of Stereo Matching

Efficiency of using image stereo matching algorithm has not been reported for cloth manipulation [Willimon et al., 2011b][Maitin-Shepard et al., 2010] while to our knowledge, it remains an issue, especially in this real-time task. In addition, a more efficient cloth manipulation can result in a faster production line that leads to more productivity. Therefore, we aim to provide a more efficient solution to utilize image stereo matching, especially within the context of cloth manipulation.

With the development of computing technology, the performance of CPUs and GPUs has been rapidly increased. Advances in those different hardware architectures make it possible to perform computation intensive tasks in real time by parallelization. More and more algorithms that were not fit for real-time application are now feasible. The migration of the old algorithms onto the new architectures or the formation of new algorithms harnessing new architectures is attracting more attention. Because different architectures have different advantages, it requires extra effort to choose a suitable architecture and optimize the processing algorithm according to the specific architecture features. The aim of the work is to improve the efficiency of image matching approach under two specific parallel environments (i.e. multi-core CPU and GPU), to enable it to run in real-time.

Multi-core CPU Acceleration

In terms of CPU, recent trends in computing systems have shown that future increases in performance, will only be achieved through increases in system scale, i.e., using a larger number of components (e.g. cores in the context of CPU) and not by improvements in single-processor performance. As a consequence to this Multi-core technology, parallel computing is clearly becoming crucial. Parallel computation could dramatically increase the speed, efficiency and performance of computing systems. We aim to utilize multi-core CPUs to parallelize stereo matching and specifically answer the following research questions (RQs):

> **RQ 1**: How much speed-up can multi-core CPUs offer on accelerating stereo matching algorithms? Is there an acceleration plateau regardless of further increase of the number of CPU cores?
>
> **RQ 2**: How do different CPU system architectures (such as 486, Pentium, Pentium 4 and Sandybridge) affect the acceleration performance?

RQ 3: After decomposing the stereo matching algorithms into different sub-components, how do different sub-tasks perform differently in terms of acceleration? Can we further infer from the algorithmic perspective, which characteristics can benefit more from the multi-core CPU acceleration?

The experimental results show significant performance accelerations with multi-core CPUs giving $12\times$ speedup on average (using Vector Pascal programming language to implement on the Sandybridge architecture, i.e. 64-core AMD Opteron 6366HE), compared to SISD single-thread CPU performance. The acceleration plateau of the MSSM stereo matching algorithm is achieved at around 30 cores while further increasing the number of cores (even to 64 cores) do not further improve the efficiency. Comparing different architectures, it is not surprising that Sandybridge outperforms Pentium 4, Pentium and 486. In addition, we found that the optimized acceleration of one stereo matching sub-task "interpolation" has a greater impact on the overall speed-up than another sub-task "convolution". This is because "interpolation" is more suited for parallel execution, but "convolution" currently has a speed-up limitation due to its high demands on storage of intermediate results.

GPU Acceleration

A serious competitor for the multi-core CPU is represented by graphical processing units (GPUs), which are graphic cards used for scientific computing, especially for computer vision problems. Comparatively speaking, GPU is faster, cheaper, uses less power and it was designed specifically to process graphics, and that means processing streams of data. Graphics chips may simply be seen as massive multi-cores, where in high end versions thousands of units are running in parallel.

As stereo matching is an image processing algorithm and it is natural to assume that utilizing GPU can be a better fit in acceleration of stereo matching algorithm than multi-core CPU. In order to study whether this is the case, we conduct experiments to answer:

RQ 4: How many times speed up can GPUs achieve on accelerating stereo matching algorithms? How does it compare with multi-core CPUs?

RQ 5: How do different stereo matching sub-tasks perform differently in terms of acceleration on GPUs? Does the GPU acceleration trend vary from the one of multi-core CPUs?

We found that overall GPU (using CUDA C programming language to implement on the GeForce GTX 770) gives $176\times$ acceleration compared to single-thread CPU performance (using Vector Pascal programming language to implement on the 64-core AMD Opteron

6366HE), which is significantly better than multi-core CPUs ($12\times$ speed-ups). Processes of array calculations (such as polynomial maximization as one sub-task) can be best paralleled with a 731 times speed-up on GPU. We also found that when implementing algorithms on GPU, one should let GPU do parallel computing as much as possible, and try the best to keep intermediate variables on GPU only, because the data transfer within GPU is much faster than the data transfer to CPU.

1.2.2 Accurate Stereo Matching for Cloth Manipulation

After studying stereo matching from solely the *efficiency* perspective by accelerating the algorithm using various hardware architectures, we move our attention towards the *effectiveness* aspect, i.e. how can we improve the stereo matching algorithms that perform more accurately on cloth images? This requires proposing a novel stereo matching algorithm that suits the characteristics of cloth images where the surface is generally non-rigid and continuous, and there are also small depth changes (wrinkles on the cloth).

Before proposing a new stereo matching algorithm, we also need to evaluate the accuracy of the stereo matching performance within the context of our interest (cloth images). However, to our knowledge, no public testbeds are available with depth map ground-truths for evaluation purposes. Therefore, we also create a set of testbeds and adapt a set of evaluation metrics for us to thoroughly measure the cloth based stereo matching accuracy. We made those testbeds public available [4] and they can facilitate the cloth based stereo matching evaluation and development in the research community.

Cloth Manipulation Stereo Matching Evaluation

To evaluate the performance of stereo matching in the context of cloth manipulation, a ground-truth disparity is required for those images with garments within it. In order to thoroughly evaluate in this context, we need to construct a set of images with different cloth materials and different pose configurations for the thorough evaluation. In addition, in most circumstances, as the robot normally aims to deal with clothes in a chaotic state (e.g. for unfolding or grasping), we also make sure there are various wrinkle types on the clothes. We construct various testbeds and aim to answer:

> **RQ 6**: How to construct the proper datasets to evaluate stereo matching algorithms in the context of cloth manipulations? Can we also provide disparity ground-truths for both unrectified and rectified images?

[4] https://sites.google.com/site/skytianxu

RQ 7: What evaluation metrics should be used to evaluate the accuracy performance of cloth related stereo matching algorithms given the testbeds we developed?

We construct in total five datasets for evaluating stereo matching in the context of cloth manipulation (two simulated and three real-world), not only considering large depth changes, but also measuring micro depth changes such as cloth wrinkles. We also include both unrectified and rectified images, and their corresponding ground-truths in some of our created testbeds. The proposed evaluation methodology can be useful and the datasets can serve as standard testbeds for measuring stereo matching for cloth manipulation. We also review a set of evaluation metrics that are based on either disparity map or depth map ground-truths.

Guided Filtering based Pyramidcal Stereo Matching

Due to the requirement to obtain wrinkle details on the clothes (e.g. for flattening the clothes [Sun et al., 2015]), proposing stereo matching algorithms that can accurately extract wrinkles with small depth changes is important.

In addition, although the normal process for most current stereo matching is to first rectify image, and then apply existing stereo matching algorithms, nevertheless, this procedure could be expensive and a non-ideal stereo configuration usually produces inferior results. Therefore, we also would like to study whether it is possible to use a stereo matching algorithm that directly applies to unrectified images. To cope with the above challenges and remedy the rectification issue, we aim to answer:

RQ 8: Can we propose a new stereo matching algorithm that better suits to cloth images?

RQ 9: Can we effectively derive accurate depth maps by directly matching unrectified cloth images?

Therefore, we adopt a single stage stereo matching methodology that derives the disparity information directly from the unrectified images and adapt a state-of-the-art guided filtering algorithm [He et al., 2013] within a pyramid based stereo matching framework [Xu et al., 2014]. We found that this proposed algorithm is capable of processing high resolution images with low memory cost, is easy to parallelize, and accurately derives depth information for small cloth wrinkles. The proposed algorithm is better suited to cloth characteristics and outperforms other state-of-the-art stereo matching algorithms (such as guided filtering and pyramidical stereo matching algorithms). We also show that rather than relying on image rectification, directly applying stereo matching to the unrectified images is effective and efficient.

1.2.3 Trading off Cloth Stereo Matching Accuracy for Efficiency

So far, for cloth images, we have proposed a stereo matching algorithm that effectively derives accurate depth maps in a very short time span, benefitted from the acceleration from the multi-core CPU and GPU hardware architectures. Now the aim is to apply the stereo matching algorithms to various robotic cloth manipulation scenarios and study how the stereo matching performance translates into the robotic task accomplishment, in terms of both effectiveness and efficiency.

We demonstrate in RQ8-9 that our newly proposed stereo matching algorithm is able to derive detailed cloth wrinkle depth maps and we aim to have this stereo matching sufficiently accurate and efficient to suit all robotic manipulation tasks. However, when actually applying stereo matching for robotic vision, different robotic cloth manipulation tasks (e.g. grasping, flattening, etc.) may require different levels of matching accuracy to accomplish the task. In addition, the task accomplishment performance may also vary according to different cloth characteristics (such as materials or wrinkle size).

This provides an additional opportunity for us to dynamically accelerate the stereo matching algorithm given the robotic manipulation task. The idea is to dynamically tailor the matching performance and tradeoff accuracy for efficiency, as long as the matching accuracy is sufficient for the given robotic task. From the entire robotic system perspective, this can result in a more efficient system without dropping any manipulation performance.

Foveated Stereo Matching for Cloth Manipulation

Stereo vision is an important component of robotic cloth manipulation (i.e grasping and flattening), which required to be processed fast. To tailor the stereo matching to tradeoff accuracy for efficiency, motivated by biological systems, we exploit utilizing foveated stereo matching to accomplish this, rather than working with coarse resolution images which restricts the acquisition of detailed information. Specifically, we implement a parallel extension of Boyling's foveated pyramid matching algorithm [Boyling and Siebert, 2000] and aim to answer:

> **RQ 10**: Whether foveated matching can be used to accelerate the process while being sufficiently accurate, for two robot cloth manipulation tasks: flattening and grasping?
>
> **RQ 11**: How to determine when foveated matching is sufficient to accomplish the given robotic cloth grasping and flattening task?

We found that foveated matching is effective in trading off accuracy for efficiency for stereo matching, especially for those cloth wrinkles that are "smooth" (wrinkles of low height and

wide width). In addition, we compare foveated matching with the simple solution of just utilizing low resolution images, in terms of the accuracy versus efficiency trade off. Finally, by conducting simulation of robot behavior using previous work for both cloth grasping and flattening tasks, we demonstrate that using foveated matching can achieve the similar levels of accuracy with two to three times of acceleration.

Learning to Trade-off Accuracy for Efficiency in Robot Cloth Manipulation

Although the simulation provides preliminary insights on the effectiveness of this foveated stereo matching framework in achieving a better accuracy-efficiency tradeoff in robot manipulation tasks, however, there are several limitations to this simulation. Firstly, we assume that there is only one wrinkle on the cloth, which is a simple case we simulate for the simplicity of explanation. Secondly, we assume a certain stereo matching accuracy is required to achieve the robotic manipulation tasks. This might vary according to different cloth materials, wrinkle properties, etc. We aim to deal with more real cloth wrinkles with real robotic manipulations (various tasks) as our focus.

> **RQ 12**: Can we simulate more complex scenarios of garments with various wrinkle properties and can we simulate the task success for grasping and flattening under those scenarios?
>
> **RQ 13**: Can we learn to effectively predict which foveation level is required given the image features of the current configuration?
>
> **RQ 14**: Can we apply this learning framework to the foveation level and obtain better efficiency while keeping the same task accomplishment rates? If so, how much can we further accelerate the stereo matching performance?

By utilizing a more complex garment dataset with more wrinkle types, positions and sizes, we propose to derive the success labels given the quantified wrinkles for both robot cloth grasping and flattening tasks. Then we propose a machine learning based classification approach that is very effective in predicting the success and failure of a given robotic manipulation task, with an traditional classification metric AUC (area under the ROC curve) of 0.84 and 0.90 achieved respectively for grasping and flattening tasks. When applying this learned foveated stereo matching to the robotic manipulation tasks, we achieved that for grasping, with our learned classifier, over 90.0% of the accuracy achieved for the dynamic foveated stereo matching. Compared to the original stereo matching, we only spend 37.0% of the original time. This implies that we can accelerate almost three times for grasping while remaining over 90% of accuracy. On the other hand, for flattening, this task seems to be more difficult. To maintain the flattening accuracy over 90.0% for the dynamic foveated stereo

matching, we can only save around 8.7% of the execution time. We also verify this to the real-world CloPeMa robot on the towel grasping task.

1.3 Summary of Contributions

The work on studying stereo matching specifically to robotic cloth manipulation is a relatively new field of research. The main contributions of this thesis lies in our practical contribution of stereo matching acceleration, several cloth related stereo matching testbeds, one newly proposed stereo matching algorithm that is effective for clothes, and our theoretical contribution on the framework of trading off accuracy for efficiency using foveated stereo matching.

C1. We provide a comprehensive understanding of how to exploit both multi-core CPU and GPU to accelerate stereo matching algorithm. In particular, we employ various strategies present in the literature and design extensive experiments to investigate the parallel effect on different sub components of the stereo matching algorithm. Our results provide a solid framework in parallelizing image processing algorithms using different hardware architectures. *(Chapter 3 and Chapter 4)*

C2. We are the first to construct testbeds with ground truth for cloth based images in order to measure robot cloth manipulation stereo matching performance. We build several practical testbeds and obtain the disparity and depth map ground-truths through simulation. Those testbeds consist of both unrectified and rectified images while the created testbeds could be beneficial for the stereo matching and robotic vision research community. *(Chapter 5)*

C3. We propose a new and effective single stage stereo matching algorithm that derives the disparity directly from the unrectified images. By adapting a state-of-the-art guided filtering algorithm within a pyramid based stereo matching framework, we propose an effective methodology to match images with small and smooth depth changes on a continuous surface (such as cloth wrinkles). We also evaluate extensively on our created testbeds and prove that directly applying stereo matching through the unrectified images can be effective and efficient. *(Chapter 6)*

C4. We propose a general framework for trading off stereo matching accuracy for efficiency by utilizing foveated stereo matching for robot cloth manipulation tasks. Specifically, we propose a machine learning based methodology to extract various visual features from images and depth maps, and apply that for predicting the optimal trade-off for both accurate and efficient robot manipulations. By conducting both simulation and applying to real-world robots, we demonstrate that using foveated matching

can achieve the similar level of accuracy for completing two common robotic cloth manipulation tasks, cloth grasping and flattening, with several times of acceleration. *(Chapter 7 and Chapter 8)*

1.4 Origins of the Materials

Most of the material presented in this thesis has previously appeared in several peer-reviewed conference or journal papers published in the course of this PhD programme:

P.1 W. Paul Cockshott, Susanne Oehler, **Tian Xu**, J. Paul Siebert, and Gerardo Aragon-Camarasa. Parallel Stereo Vision Algorithm. Many-core Applications Research Community Symposium (MARC 2012), pages 45-50, Aachen, Germany, 2012.
(This publication is included in Chapter 3. My contribution is to provide preliminary experiment results for stereo matching algorithm on different parallelizable CPU hardware architectures, using Vector Pascal language.)

P.2 **Tian Xu**, W. Paul Cockshott, and Susanne Oehler. Acceleration of Stereo-Matching on Multi-core CPU and GPU. 16th IEEE International Conference on High Performance Computing and Communications (HPCC 2014), pages 108-115, Paris, France, 2014.
(This publication is included in Chapter 3 and 4. In this paper, I provide a comprehensive study of how to exploit both multi-core CPU and GPU to accelerate stereo matching algorithm. I also investigate the parallel effect on different sub components of the stereo matching algorithm. The results provide a solid framework in parallelizing image processing algorithms using different hardware architectures.)

P.3 W. Paul Cockshott, Susanne Oehler and **Tian Xu**. Developing a compiler for the XeonPhi (TR-2014-341). Technical Report. University of Glasgow, Glasgow, 2014.
(This publication is included in Chapter 3 and 4. My contribution is to implement two simple algorithms (image scaling and convolution) for multi-core CPU (including Xeon Phi) on Vector Pascal, and for GPU on CUDA C. I also provide several paragraphs to analyze the performance.)

P.4 Mozhgan Chimeh, W. Paul Cockshott, Susanne B. Oehler, Ashkan Tousimojarad and **Tian Xu**. Compiling Vector Pascal to the XeonPhi. Concurrency and Computation: Practice and Experience (Journal), pages 5060-5075, 2015.
(This publication is included in Chapter 3 and 4. This is an extension of [P.3] while my contribution is similar to [P.3]. I implement the image scaling and convolution algorithms for multi-core CPU (including Xeon Phi) on Vector Pascal, and for GPU on CUDA C. I also provide analysis on the performance.)

P.5 **Tian Xu** and W. Paul Cockshott, Guided Filtering Based Pyramidical Stereo Matching for Unrectified Images. International Conference of Image and Vision Computing New Zealand (IVCNZ 2015), Auckland, New Zealand, 2015. ***Best Paper Award***
(This publication is included in Chapter 5 and 6. In this paper, by adapting guided filtering algorithm into a pyramidical stereo matching framework, I propose a new single stage stereo matching algorithm that derives the disparity directly from the unrectified images. By evaluating extensively on three created testbeds that suits to the characteristics of the task of cloth manipulation, the algorithm has been proved to be effective and efficient for matching images with small and smooth depth changes on a continuous surface, such as cloth wrinkles.)

P.6 **Tian Xu** and W. Paul Cockshott, Evaluation of Foveated Stereo Matching for Robotic Cloth Manipulation. 11th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISAPP 2016), Rome, Italy, 2016.
(This publication is included in Chapter 5 and 7. The aim of this paper is to study the effect of foveated matching algorithm on robotic manipulation tasks. I first create a "garment with wrinkle" dataset that includes depth map ground-truth for garments. Then I evaluate the performance using this dataset and found that foveated matching is effective in trading off accuracy for efficiency for stereo matching. Also by assuming the robotic behavior of grasping and flattening, I demonstrate that using foveated matching can achieve the same level of accuracy for completing both tasks with several times of acceleration.)

P.7 **Tian Xu** and W. Paul Cockshott, Learning to Trade-off Accuracy for Efficiency for Robotic Cloth Manipulation (Under Submission).
(This publication is included in Chapter 7 and 8. I propose a machine learning based methodology to extract various visual features from images and depth maps, and apply that for predicting the optimal tradeoff for both accurate and efficient robot manipulations. By conducting both simulation and applying to real-world robots, we demonstrate that using foveated matching can achieve the similar level of accuracy for completing two common robotic cloth manipulation tasks, cloth grasping and flattening, with several times of acceleration.)

P.8 Li Sun, Gerardo Aragon-Camarasa, **Tian Xu**, W. Paul Cockshott and J. Paul Siebert. Clothes Perception and Manipulation Using A Stereo Vision System. IEEE Transactions of Robotics (Under Submission).
(This publication is included in Chapter 4. My contribution is to provide an introduction of parallel pyramidal stereo matching algorithm within the cloth perception and manipulation framework, and also GPU Parallel Strategies for this algorithm. I also

compare the performance of the GPU version of this key stereo matching component with multi-core CPU version.)

1.5 Organisation

The thesis is divided into five main parts. Part I presents motivations, research questions, technical background and the state-of-the-art. Part II-IV presents our methodology for efficient yet accurate stereo matching for cloth manipulations, including utilizing hardware architectures for acceleration (Part II), proposing cloth related stereo matching testbeds and algorithm (Part III), and learning to tradeoff robotic manipulation accuracy and efficiency (Part IV). Part V concludes and outlines the future work. The detailed organization of the thesis is given below.

Part I. Introduction and Background

Chapter 1 includes this introduction that mainly describes the motivation of this work. Research questions and methodology are also discussed in this chapter.

Chapter 2 presents prior work in this area. This consists of a brief overview of state-of-the-art stereo matching approaches, followed by the descriptions of different acceleration methodologies and robotic vision work for cloth manipulations.

Part II. On the Acceleration of Stereo Matching

Chapter 3 presents and measures various ways of accelerating stereo matching algorithms using multi-core CPU.

Chapter 4 describes and evaluates parallelizing stereo matching algorithms using GPU.

Part III. Accurate Stereo Matching for Cloth Manipulation

Chapter 5 describes our methodology to create five different stereo matching testbeds for cloth manipulations.

Chapter 6 presents and evaluates our proposed guided-filtering based pyramidical stereo matching algorithms on both accuracy and efficiency.

Part IV. Trading off Cloth Stereo Matching Accuracy for Efficiency

Chapter 7 describes and evaluates our methodology to utilize foveated stereo matching for trading off accuracy for efficiency for two robotic cloth manipulation tasks.

Chapter 8 presents the learning framework to dynamically adjust foveation level for optimal accuracy-efficiency tradeoff.

Part V. Conclusions and Discussion

Chapter 9 concludes the thesis by discussing the main findings, implications for each research question and presents the future work that could be carried out.

Readers familiar with stereo matching approaches, hardware acceleration or robotic vision can skip over Chapter 2.

Chapter 2

Prior Work and Background

In this chapter, we introduce the background and the most relevant prior work to this thesis. Therefore, we aim to cover the three main themes of this thesis (as indicated by the thesis title): stereo matching (Section 2.1), acceleration of efficiency (Section 2.2) and robotic cloth manipulations (Section 2.3). Within the entire literature review, we mainly focus on the approaches that are most relevant while briefly discussing the broad background. We refer the readers to other relevant books or surveys if more details are needed.

2.1 Stereo Matching

Stereo image matching entails discovering the most likely matches between pixels in two images. Typically these are captured simultaneously from cameras in different spatial locations. The technique is widely used in computer vision and robotics, including: data visualization, three dimensional map building and robot pick and place. It has been studied over several decades in computer vision and many researchers have worked at solving it [Scharstein and Szeliski, 2002]. The stereo problem is easy to state, but not easy to solve, when considering the geometry behind it. A typical example of a single 3D point and two perspective cameras [Hartley and Zisserman, 2003] is presented in Figure 2.1. The basis of inferring actual depth is to search for the projection of the same 3D point across the two images and calculate the difference in image coordinates between those projections (disparity). Once the corresponding projections are found, the absolute 3D coordinates of the world point are completely determined via triangulation [Hartley and Sturm, 1997], provided that the stereo cameras are calibrated. Within this framework, triangulation and calibration are relatively straightforward, while correspondence remains challenging. Stereo matching is a method which aims to solve the correspondence problem.

There are two main classes of stereo matching algorithms: local methods and global meth-

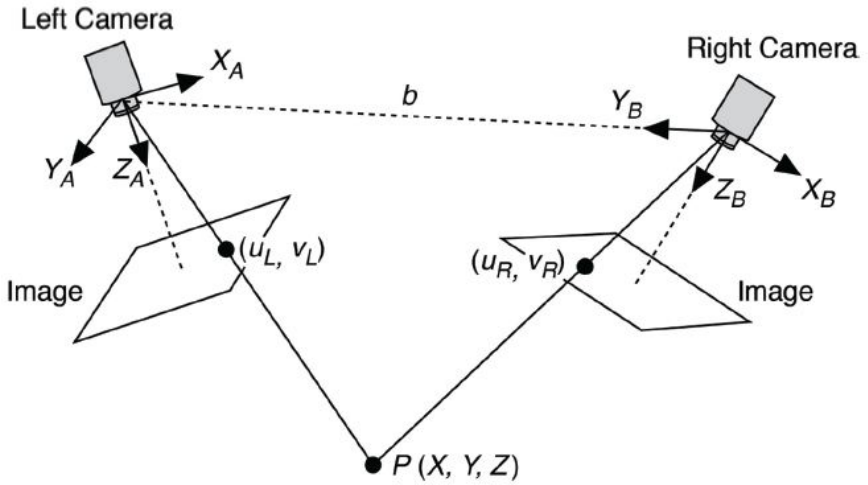


Figure 2.1: Typical Stereo Vision System

ods. Local algorithms are statistical methods and are usually based on correlation. For global algorithms, the task of computing disparities is cast in terms of energy minimization, and is solved by various optimization techniques [Klaus et al., 2006][Kolmogorov and Zabih, 2001]. Compared with local algorithms, global algorithms are normally computationally much more expensive. We briefly explain below algorithms of those two classes.

2.1.1 Local Methods

Local algorithms can be subdivided into two categories: feature-based algorithms and area-based algorithms.

Feature-based Algorithms

Feature-based matching algorithms [Birchfield and Tomasi, 1999][Venkateswar and Chellappa, 1995] attempt to establish a correspondence by matching a sparse sets of image features. Feature points may include edges, lines, regions, and gradient peaks. But only feature points with high texture diversity are included, areas of low texture diversity are not matched with another. The number of points used is related to the number of image features identified. Since only the feature points are matched, the computation cost is greatly reduced. Although, feature-based algorithms work very fast, complete dense disparity maps cannot be obtained. They can only generate sparse disparity maps.

Feature-based methods are applied to reduce processing time for stereo matching, so consideration is taken on choosing a feature extractor that does not consume more computation time than it saves. But these feature-based algorithms are not suitable for many applications (e.g. reconstructing surfaces) that require dense disparity maps.

Area-based Algorithms

When considering generating dense disparity map, area-based stereo matching method is the most common way to accomplish the task. Area-based algorithms [Faugeras et al., 1993][Zabih and Woodfill, 1994] are also called correlation-based algorithms. These methods merge the feature detection step with the matching part, which means it aims to deal with the images without attempting to detect salient objects in the images.

Correlation-based stereo methods match neighbouring pixel values, within a window, between images. They assume that for the neighbourhood of homologous pixels, the colour intensities are also similar. Moreover, they assume that the colour intensities of neighbours of a pixel in the left image are close to those of the same neighbours of its homologous pixel in the right image. Therefore, the matching cost is defined between the window around the left pixel and the windows around the corresponding candidate pixels in the right image as shown in Figure 2.2. The window is a square window, defined by its centre point P and its radius r. Matching cost functions are evaluated over this window of neighbouring pixels in the image. The best correlation between windows in the left and right images is searched for all possible disparity positions. Note that most of algorithms only focus on horizontal disparities, because they are normally using rectified input stereo images that are pre-processed using a procedure called image rectification [Loop and Zhang, 1999]. Image rectification is a transformation process used to project two-or-more images onto a common image plane. Therefore, below, only 1D (horizontal) disparities are considered when we introduce those algorithms.

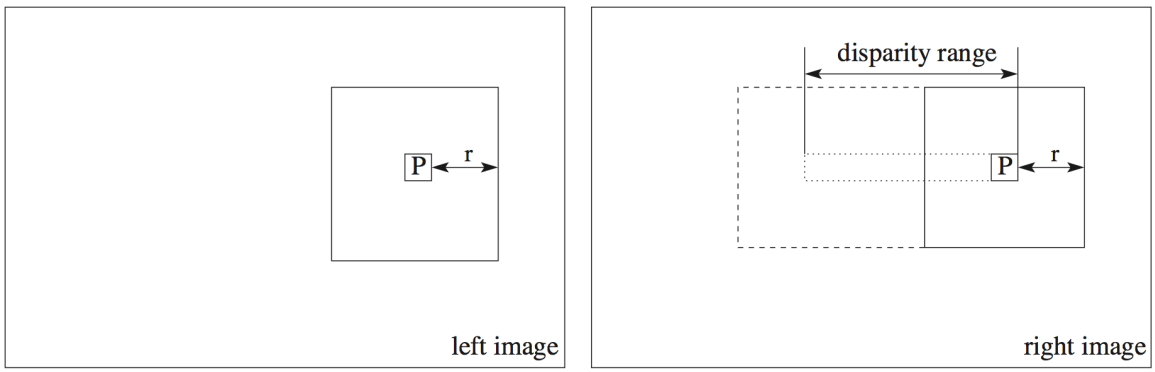


Figure 2.2: Correlation between left and right image at point P

Common correlation-based matching costs [Hirschmüller and Scharstein, 2007] include the Sum of Squared Differences (SSD), Sum of Absolute Differences (SAD), Normalized Cross Correlation (NCC) and rank and census transforms.

SSD for grey-scale images is defined as

$$SSD_g^w(x_l, y, d) = \sum_{u=-w}^{w} \sum_{v=-w}^{w} (I_l(x_l + u, y + v) - I_r(x_l + u - d, y + v))^2 \tag{2.1}$$

where $I_l(x_l, y)$ means the intensity of pixel with coordinate (x_l, y) in the left image. d represents the value of disparity candidate, g means grey-scale, and w is the radius of the window. The window size is $(2w + 1) \times (2w + 1)$.

The SAD cost can be formulated as

$$SAD_g^w(x_l, y, d) = \sum_{u=-w}^{w} \sum_{v=-w}^{w} |I_l(x_l + u, y + v) - I_r(x_l + u - d, y + v)| \tag{2.2}$$

While the NCC matching is defined as

$$NCC_g^w(x_l, y, d) = \frac{\sum_{u=-w}^{w} \sum_{v=-w}^{w} |I_l(x_l + u, y + v) \times I_r(x_l + u - d, y + v)|}{NormC_{lg}^w(x_l, y) \times NormC_{rg}^w(x_l - d, y)} \tag{2.3}$$

where $NormC_{lg}^w(x_l, y)$ and $NormC_{rg}^w(x_l - d, y)$ are normalization coefficients of grey-scale corresponding to pixels within the $(2w + 1) \times (2w + 1)$ aggregation window, centered on (x_l, y) in the left image and $(x_l - d, y)$ in the right image respectively. The normalization coefficients are expressed as

$$NormC_{lg}^w(x_l, y) = \sqrt{\sum_{u=-w}^{w} \sum_{v=-w}^{w} (I_l(x_l + u, y + v)^2} \tag{2.4}$$

and

$$NormC_{rg}^w(x_l - d, y) = \sqrt{\sum_{u=-w}^{w} \sum_{v=-w}^{w} (I_r(x_l + u - d, y + v)^2} \tag{2.5}$$

where lg and rg refer to the left and right grey-scale images respectively.

Some costs are insensitive to differences in camera gain or bias, such as rank and census transforms [Zabih and Woodfill, 1994]. This non-parametric matching costs are based on the local order of intensities. The Rank transform replaces the intensity of a pixel with its rank among all pixels within a certain neighbourhood to increase the robustness of window-based methods. Since the non-parametric costs only depend on the ordering of intensities and not the magnitude of intensities, they tolerate all radiometric distortions that preserve this ordering. It is defined as

$$I_{Rank}(x, y) = \sum_{u=-w}^{w} \sum_{v=-w}^{w} T[I(x + u, y + v) < I(x, y)] \tag{2.6}$$

The function $T[\,]$ is defined to return 1 if its argument is true and 0 otherwise. The transformed images are matched with the absolute difference. Figure 2.3 shows an example. Because only 51 and 49 are smaller than 55 (pixel of interest), so the sum would be 2, which means the rank cost is 2.

89	63	72	89	63	72
67	55	$64 \Rightarrow 2$	67	55	$64 \Rightarrow 00000011$
58	51	49	58	51	49

Figure 2.3: Example rank (left) and census (right) transforms in a 3×3 window

Zabih and Woodfill [Zabih and Woodfill, 1994] also propose a variation of the rank transform, called the census transform. A bit string is defined where each bit corresponds to a certain pixel in the local neighbourhood around the pixel of interest. If a pixel has a lower intensity than the pixel of interest, then the corresponding bit is set to 1 as shown in Figure 2.3. Thus, unlike Rank, Census not only stores the intensity ordering, but also the spatial structure of the local neighbourhood. The transformed images are matched by computing the Hamming distance between corresponding bit strings. As found by Hirschmüller and Scharstein [Hirschmüller and Scharstein, 2009], the census transform is quite robust against large scale non-stationary exposure and illumination changes.

Using a square window as described above is the most straightforward matching cost aggregation approach, but it is not the only one. Many works propose different aggregation approaches and there are several classic ones. The shiftable window approach considers multiple square windows centered at different locations. It retains the window with the smallest cost [Bobick and Intille, 1999] [Fusiello et al., 1997]. Windows with adaptive sizes make it possible to automatically select the window size and/or shape based on local information [Kanade and Okutomi, 1994]. The window size should be large for areas with low textures and small for areas with fine details. Yoon and Kweon [Yoon and Kweon, 2006] proposed an approach for correspondence search using adaptive support-weight. This method is recognised as having the best performance to speed tradeoff [Tombari et al., 2008]. Instead of using square windows with uniform weighting, each pixel within an aggregation window influences the final matching cost based on the color similarity and spatial distance.

After computing and aggregating the matching costs, the disparity of a given pixel can be simply computed by performing a Winner-Takes-All (WTA) optimization, i.e. choose at each pixel the disparity associated with the minimum cost value. Repeating this process for every pixel of the image, a disparity map accurate to integer pixel could be produced.

However, integer disparity accuracy is not sufficient for some applications, such as image-based rendering or 3D modelling. To remedy this situation, many local stereo algorithms apply a sub-pixel refinement step after the initial discrete correspondence computation [Scharstein

and Szeliski, 2002]. One simple way is to fit a curve to the matching costs at discrete integer disparity levels neighbouring the best integer matching and then interpolate to find a more precise local minimum. In this way, only little additional computation need to be added to the stereo algorithm, but the intensities being matched should vary smoothly, to make the algorithm work effectively.

Apart from sub-pixel computations, there are other post-processing steps available, such as cross-checking between the left-to-right and right-to-left disparity maps to detect occluded areas. Applying a median filter could help eliminating mismatches while holes due to occlusion can be filled by surface fitting.

2.1.2 Global Methods

Global stereo matching methods consider nonlocal constraints, in order to reduce the sensitivity to local regions that fail to match due to occlusion or uniform texture. These methods are formulated in an energy minimization framework to find the disparity solution d that minimizes the global energy function. This energy function $E(d)$ consists two terms: data $E_d(d)$ and smoothness $E_s(d)$.

$$E(d) = E_d(d) + \lambda E_s(d) \tag{2.7}$$

where λ is a regularization parameter that controls the influence of the smoothness term.

$E_d(d)$ measures the degree of similarity between each pixel in the reference image and its corresponding pixel in the other image at the current disparity position d. The smoothness term $E_s(d)$ considers the smoothness assumptions made by the algorithm. If two pixels are adjacent, they usually move about the same amount. Thus, the smoothness term is often restricted to only measuring the differences in disparities between neighbouring pixels.

The solution of the global energy minimization problem is based on Markov Random Field (MRF). A number of efficient optimization methods have been developed and applied to global stereo such as dynamic programming [Bobick and Intille, 1999][Cox et al., 1996][Ohta and Kanade, 1985], belief propagation [Felzenszwalb and Huttenlocher, 2006][Sun et al., 2003], graph cuts [Boykov et al., 2001] [Kolmogorov and Zabih, 2001], and others. These optimization methods in general give good empirical results compared to local methods and forms the basis for many of the state-of-the-art stereo methods today.

On the contrary, global methods can not compete with local algorithms in terms of speed, memory storage requirements, and computational complexity. This is the reason that in practice, especially for real-time applications, researchers still prefer to use local methods, as they require less processing and are parallel friendly. Therefore, for this thesis, we will mainly focus on local area-based methods.

2.1.3 Coarse-to-fine Approach

So far, we have introduced both local and global stereo matching methods. Other method such as hierarchical (coarse-to-fine) algorithm is a good method to reduce computation time. Such algorithms always operate on image pyramid, where the base level is the original image, and the coarser resolution levels are smaller images subsampled from original image. The most widely used pyramids are Gaussian, Laplacian and Quadtree. The initial estimate for the disparity is computed at low resolution, and results from coarser levels is refined at finer levels until the original resolution achieved. Because of refinement step, the search range for each pyramid level could be smaller in comparison to non hierarchical algorithms that calculated from scratch. Both local and global methods can be fit into this coarse-to-fine framework [Quam and Center, 1984][Witkin et al., 1987][Yang and Pollefeys, 2003][Felzenszwalb and Huttenlocher, 2006].

The coarse-to-fine approach also has limitations. Mistakes occurred at the coarse resolution are difficult to correct. Meanwhile, this framework can not handle depth discontinuities well, due to high-frequency details lost at coarser scales. But tasks like robotic cloth manipulation which requires accurate depth map generated for continuous cloth surface is a good exception because it normally does not contain depth discontinuity [Sun et al., 2015].

2.1.4 Evaluation

To evaluate the accuracy of stereo matching algorithms, in the vision community, Middlebury stereo benchmark [Scharstein and Szeliski, 2002] is the most popular and widely accepted evaluation benchmark. It provides several stereo datasets with ground-truth disparities, as examples shown in Figure 2.4. This benchmark also gives online evaluation for researchers to submit and compare their stereo matching algorithms with others. The benchmark contains many stereo pairs with different image size, disparity range, as well as with various scene and texture. Among the stereo pairs, Tsukuba, Venus, Teddy and Cones (Figure 2.4) compose the dataset for the most popular Middlebury Stereo Evaluation - Version 2 [1], where more than 160 algorithms are ranked.

The evaluation metrics used for the version 2 is the percentage of bad matching pixels, while for most recent version - Version 3[2], more comprehensive and complex measurements are added (e.g. root mean square disparity error, average absolute error, time, etc.). More detailed description of the classic evaluation metrics are presented in section 5.3.

Note that although Middlebury benchmark datasets are useful to evaluate the performance of stereo matching algorithms in general, it does not specifically focus on evaluation in the

[1] http://vision.middlebury.edu/stereo/eval/
[2] http://vision.middlebury.edu/stereo/eval3/

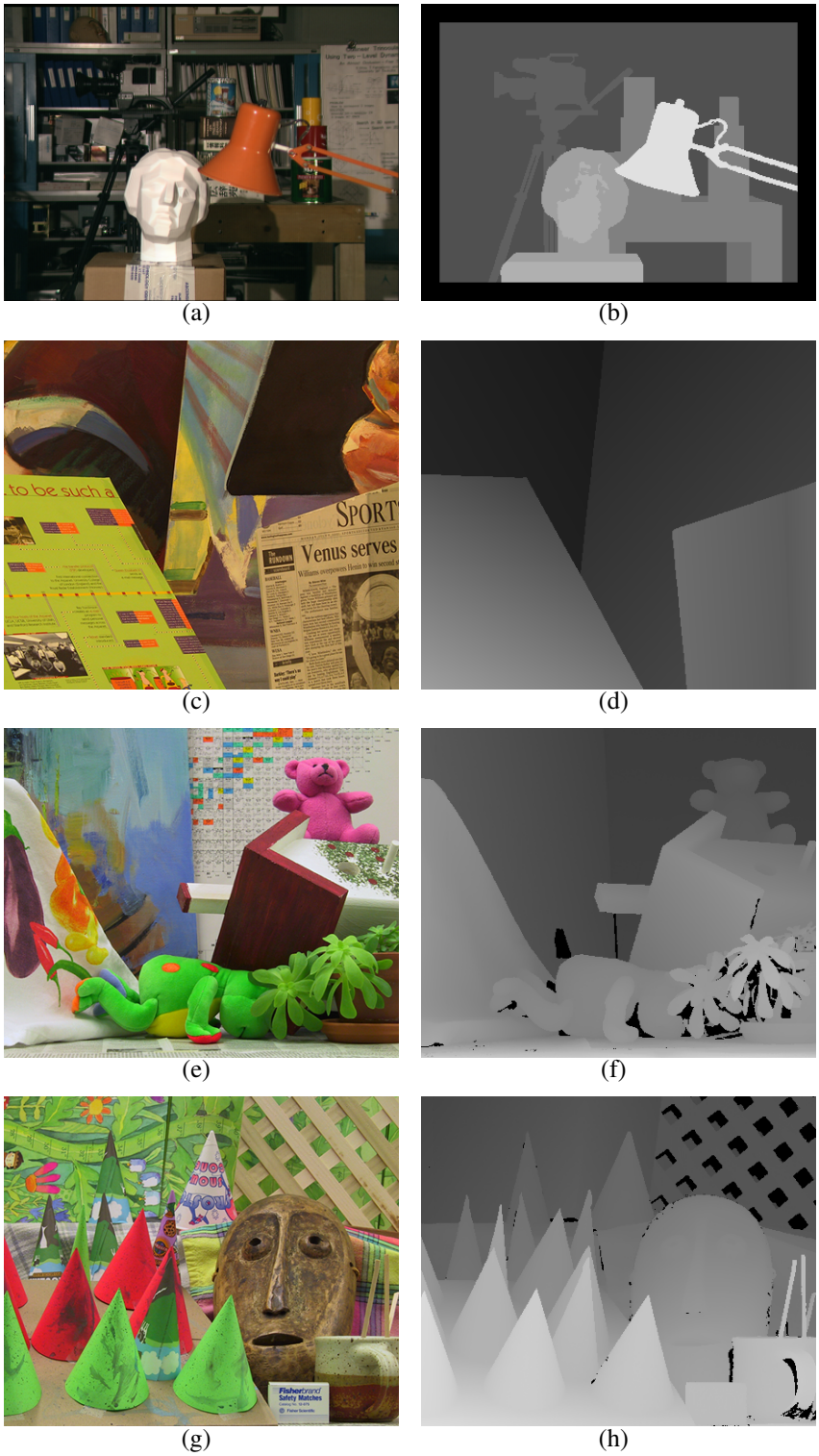


Figure 2.4: Left views and disparity maps of Tsukuba, Venus, Teddy and Cones in the Middlebury stereo matching benchmark dataset Version 2

context of robotic cloth manipulation. Therefore, in this thesis, we aim to bridge this gap by creating a benchmark (several testbeds) for evaluating specifically stereo matching algorithms on garment images.

2.2 Acceleration

Recent applications (e.g. mobile robots and autonomous vehicles) require fast stereo capture. A single core scalar CPU is too slow to give high resolution stereo with modern cameras. This has motivated research into parallelizing the problem. Zhang et. al [Zhang et al., 2008] proposed two parallel SIFT (Scale Invariant Feature Transform) algorithms with optimization techniques to improve the application's performance on multi-core systems. Jang et. al [Jang et al., 2008] implemented neural networks-based text detection system on both GPU and multi-core CPU processors. The paper of Yang et al. [Yang et al., 2003] is the first one to explore the potential of GPUs to accelerate depth estimation. They effectively used the capability of graphics hardware to warp and process images. Zhang et. al [Zhang et al., 2012] applied a parallel Motion Estimation algorithm on a heterogeneous computing system, and compared its performance on a CPU and a GPU. Below we introduce more details on those two common hardware architectures used in previous work for acceleration.

2.2.1 Multi-core CPU

Due to the large number of data running on computer, nowadays the performance of single-core CPU (Central Processing Unit) is not satisfied. The clock speeds of fastest commercial CPUs have been rising to 3 or 4 GHz for a number of years now, but it is difficult to be higher because of the physical constraint. Therefore, an alternative approach, multi-core CPU draws people's attention.

Architecture

A multi-core processor is a single computing component with several independent actual processors [Barroso et al., 2000], which is also called cores. Generally, a CPU contains two to eight cores while now for example Intel even produced an 80-core chip. Hence, multi-core CPU becomes an important new trend in computer architecture.

Multiple cores can run multiple instructions at the same time, increasing the overall speed for programs to compute. Usually, the multiple processors are integrated into one single chip. Figure 2.5 is an example of a generic dual-core processor. The processors have their own private level 1 cache, and a shared level 2 cache. L1 caches are closer to cores and faster, so

the speed of access proceeds at very high speed. L2 caches are not as fast as L1 caches, but they can share data on different cores.

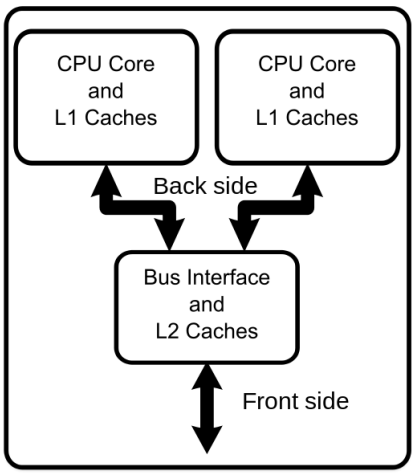


Figure 2.5: General model of dual-core CPU

Different Forms of Parallel Computing

Bit-level Parallelism It is a form of parallelism which is based on increasing processors word size. Increasing the word size shortens the number of instructions that the system must run in order to perform an operation on variables which are greater in size. For example, when adding two 16-bit integers on an 8-bit processor, the processor must add twice, because it needs two instructions to complete this single operation. The processor would first add 8 lower-order bits from each integer, then add 8 higher-order bits. However, for a 16-bit processor, it would be able to complete the operation with single instruction. As the development of processor, 64-bit processor becomes commonplace.

Instruction-level Parallelism Pipelining can overlap the execution of instructions when they are independent from one another. This potential overlap among instructions is called instruction-level parallelism (ILP). Here is an example:

$$c = a + b \tag{2.8}$$

$$f = d + e \tag{2.9}$$

$$g = c * f \tag{2.10}$$

The third operation depends on the result of the first two operations, so it cannot be calculated until the first two are computed. Nevertheless, since the first two operations are independent, they can be computed at the same time. One goal of compiler and processor designers is to use ILP as much as possible. Micro-architectural techniques that use ILP include:

- Instruction pipelining where the execution of multiple instructions can be partially overlapped.
- Superscalar execution in which multiple execution units are used to execute multiple instructions in parallel.
- Out of order execution where instructions execute in any order but without changing data dependencies.
- Register renaming which is a technique used to avoid unnecessary serialization of program operations caused by the reuse of registers by those operations, in order to enable out-of-order execution.
- Speculative execution which allows the execution of complete instructions or parts of instructions before being sure whether this execution is required.
- Branch prediction which is used to avoid delays cause of control dependencies to be resolved. Branch prediction is used with speculative execution.

Instruction-level parallelism rapidly increases the speed of processor over the last 15 years.

Data parallelism Data parallelism (also known as loop-level parallelism) is a form of parallel computing for multiple processors using a technique for distributing the data across different parallel processor nodes. In a multiprocessor system executing a single set of instructions (SIMD), data parallelism is accomplished when each processor performs the same task on different part of distributed data. In some situations, a single execution thread controls operations on all the data. In others, different threads control the operation, but they execute the same code. For instance, we have code running on a 2-core CPU in a parallel computing environment. It is possible to tell one processor to do that task on one part of data and other processor on another part of data simultaneously, in order to reduce execution time. Data parallelism has many applications, especially data processing applications. Real world programs usually use a combination of data parallelism and task parallelism. In this thesis, we mainly focus on data parallelism as this is suitable for image data processing algorithms.

Task parallelism Task Parallelism (also known as Thread level parallelism) is a form of parallelization in which different processors execute different thread (or process) on the same or different data. The threads may execute the same or different code. Different execution threads communicate with each other usually to share data. For example, server can serve each client in a separate thread (database server, Web server); a computer game can do graphics, AI and physics in three separate threads. Single-core superscalar processors cannot fully exploit this kind of parallelism. Thus, multi-core architectures will be the next stage in processor evolution to explicitly exploiting parallel processing algorithm.

Limitation

The performance of a multi-core chip depends very much on the software algorithms used and their implementation. Software are divided into two part. One part can be parallelized to run on multiple cores simultaneously, while the remaining part cannot be parallelized. Amdahl's Law [Amdahl, 1967] has described this effect and is used to find the maximum expected improvement to an overall system when only part of the system is improved. It is often used to predict the theoretical maximum speedup in multi-core parallel computing. Figure 2.6 indicates the limitation of speedup using multiple processors in parallel computing. If the fraction of the program that can be parallelized was decided, as the number of processors grows, the theoretical maximum speedup using parallel computing would rise to a certain value and remain unchanged, no matter how many processors are used. So what we can do is to increase the paralleled portion of the program to make efficiently speedup.

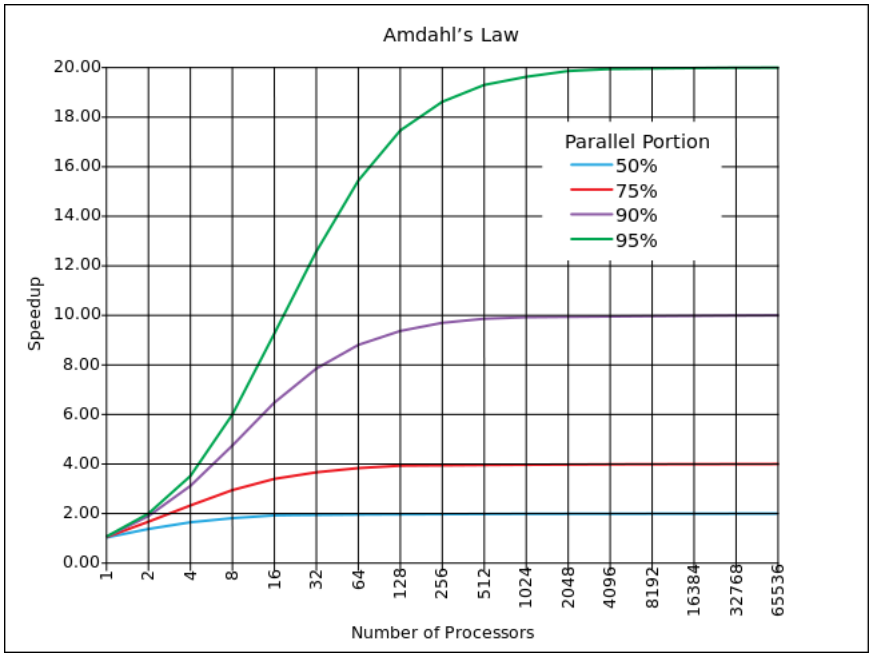


Figure 2.6: Amdahl's Law: predicting the theoretical maximum speedup using multiple processors.

2.2.2 GPU

GPU (Graphic Processing Unit) is initially designed for image rendering, but nowadays, its highly parallel structure makes it even more effective than general-purpose CPU when dealing with large blocks of data. Current GPUs have hundreds to thousands of processors that are capable of performing highly parallel floating computation [Fung and Mann, 2008]. Another feature is that GPU has its own graphics card memory that adds the whole

system memory bandwidth. Figure 2.7[3] indicates the trend rate of CPU and GPU growth with respect to floating-point operations. Therefore, GPU becomes an alternative choice to accelerate the calculation of large amount of data, especially in the image processing area.

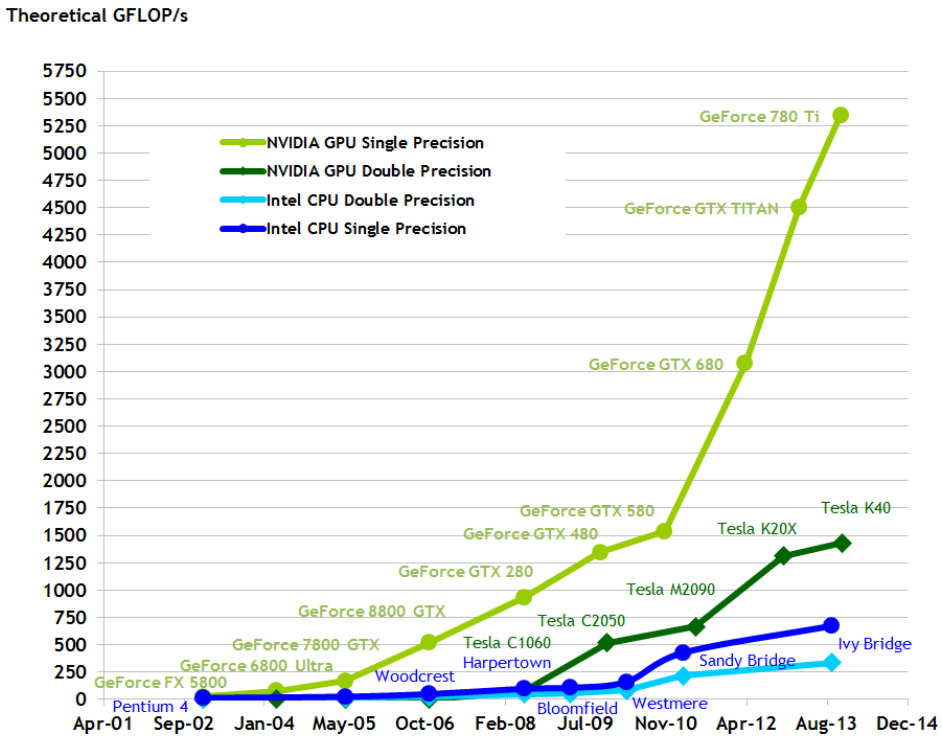


Figure 2.7: Floating-Point Operations per Second for CPU and GPU Achitectures

NVIDIA GPU

GPU Architecture A popular GPU Architecture is CUDA (Compute Unified Device Architecture), which is the model of NVIDIA GPGPU (General-Purpose Computing on Graphics Processing Units).

Although there are many different graphics cards available, they all share similar processing flow, but with different computational capabilities. Here we use a classic GeForce 8800 GTX as an example to explain the architecture. Figure 2.8 shows the architecture of CUDA. Streaming Processor (SP) is a fully pipelined, single-issue, inorder microprocessor. Although SP has its own register and program counter, it is not an independent processor, since it does not include instruction fetch and scheduling. 8 SPs make up a Streaming Multiprocessor (SM). The function of SM in GPU is similar to the core of CPU, which means it has independent instruction and constant cache, read/write shared memory. Every two or three SMs form a Texture Processing Cluster (TPC) and several TPCs form Streaming Processor Array.

[3]https://docs.nvidia.com/cuda/cuda-c-programming-guide/

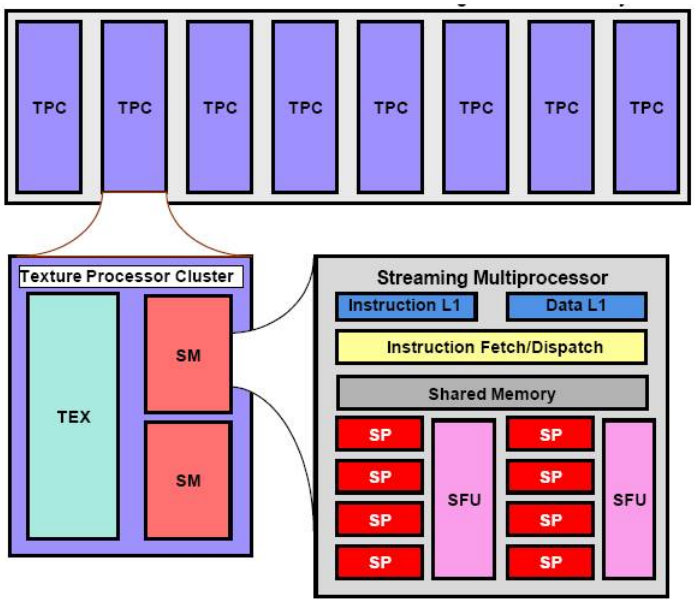


Figure 2.8: CUDA Architecture (SP=Streaming Processor)(SM=Streaming Multiprocessor)(TPC=Texture Processing Cluster)(SFU=Special Function Unit)(TEX=texture processing unit)

Parallel Programming Organization In CUDA Structure, a thread is the fundamental unit of parallel programming. A thread block is a batch of threads that can cooperate with each other by synchronizing their execution, efficiently sharing data through shared memory, with the restriction that two threads from different blocks cannot cooperate. Each grid contains several thread blocks and a kernel is executed as a grid of thread blocks. All threads in a block execute the same thread program. They have thread ID numbers within their block and the thread program uses its thread ID to select work and address shared data, as observed in Figure 2.9.

AMD GPU

AMD GPUs follow a more classic graphics processing design [Daga et al., 2011]. The computing unit is a SIMD (Single Instruction Multiple Data) engine and contains several thread processors, which each contain four processing cores as well as a special-purpose core and a branch execution unit as Figure 2.10 shows (Model: Radeon HD 5870). The special-purpose core, or special function unit (SFU) executes certain mathematical functions in hardware, such as transcendentals like $sin()$, $cos()$ and $tan()$. There is only one branch unit (BU) exists for every five processing cores, so any branch, divergent or not, leads to some amount of serialization in order to determine what path each thread will take. Figure 2.10 presents that there are totally 1600 SIMD stream cores (SC) from the equation below.

$$20\ \textit{SIMD Engines} \times 16\ \textit{Thread Processors} \times 5\ \textit{SCs} = 1600\ \textit{SIMD SCs} \tag{2.11}$$

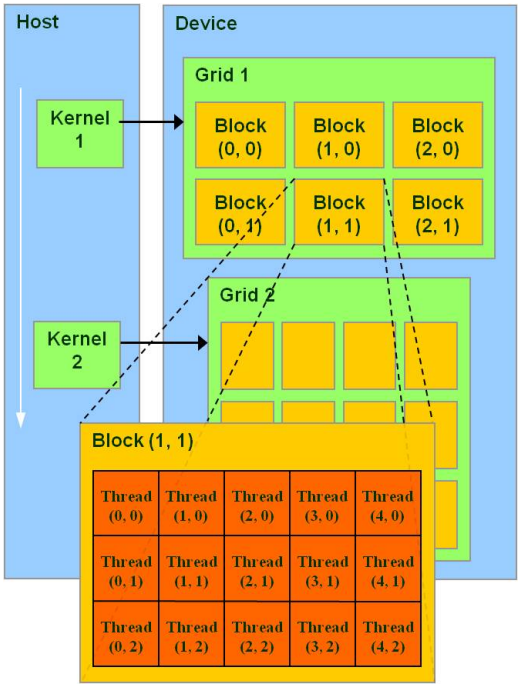


Figure 2.9: Parallel Thread Organization

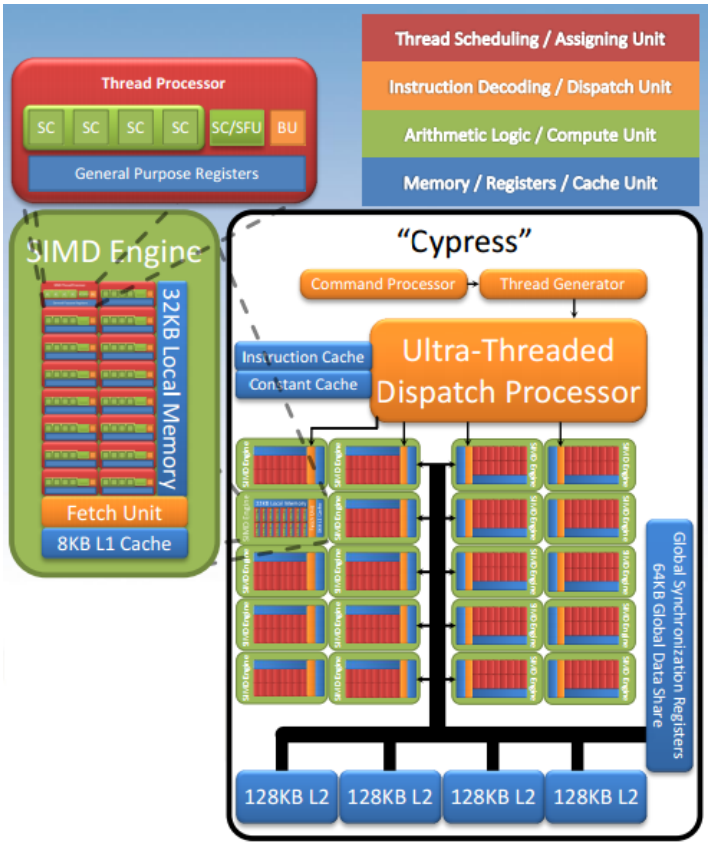


Figure 2.10: AMD Evergreen Series GPU Architecture (BU=Branch Unit)(SFU=Special Function Unit)(SC=Stream Core)

This hardware implementation allows a high number of stream processing units per core, but the card offers lower memory bandwidth and processor clocking, so the overall performance is comparable with other GPUs (e.g. NVIDIA GPUs).

OpenCL vs CUDA

As we can see, CUDA is only supported by NVIDIA GPUs, so for other GPUs (e.g. AMD GPUs), we need an alternative. Open Computing Language (OpenCL) is a framework for writing programs that execute across heterogeneous platforms consisting of CPUs, GPUs, DSPs, FPGAs and other processors or hardware accelerators. It also provides a standard interface for parallel computing using task-based and data-based parallelism. However, in this thesis, we only focus on parallel strategies only on NVIDIA GPU using CUDA, as our research focus is not on accelerating across heterogeneous platforms. And more importantly, since CUDA provides better support and optimization for NVIDIA GPUs (for example, CUDA has many libraries come for free, e.g. cuBLAS, cuSPARSE, etc.), we therefore choose CUDA to perform our experiments in the later chapters.

2.2.3 Discussion

As shown in Figure 2.7, the theoretical operation performance for the latest GPU can reach more than 5000 GFLOP/s, while Ivy bridge CPU can only achieve about 750 GFLOP/s. However, the theoretical performance can only be achieved in the optimal conditions:

- Processing units 100% used
- All parallelism 100% exploited
- All data transfers at the maximum bandwidth

However, in the real world, all the above conditions can not be satisfied most of the time. For example, some coordinations from other tasks (e.g. data-intensive tasks, such as data transfer from the hard-disk to memory) can result in less than 100% of the processing units to be utilized. Therefore, the hardware theoretical values are not realistic estimates of upper bounds of performance.

According to TOP500 [4], the state of the art high performance computers can achieve only about 60% to 90% of theoretical peak performance, for the 500 most powerful commercially available computer systems known to public. While for GPU, as investigated by Zhang and

[4] https://top500.org

Owens [Zhang and Owens, 2011], the real performances are less close to the peak performances. A simple dense matrix multiplication application only achieves around 60% of GPU peak performance [Volkov and Demmel, 2008]. For a tridiagonal solver, even worse, GPU can only achieve less than 10% of theoretical performance [Zhang and Owens, 2011].

In general, theoretical peak performance can only give a guidance of computation capability. The more important challenge is to understand the realistic attainable performance given the algorithm.

2.3 Robotic Cloth Manipulation

Robotic cloth manipulation has become a research area in recent years with various attempts being made to enable a robot to *grasp* [Maitin-Shepard et al., 2010][Ramisa et al., 2012], *flatten* [Sun et al., 2015][Cusumano-Towner et al., 2011], *fold* [Bersch et al., 2011][Van Den Berg et al., 2011] or *unfold* [Willimon et al., 2011a][Doumanoglou et al., 2014b] clothes with wrinkles on them.

Grasping is the basic task in the context of robotic cloth manipulation. Maitin-Shepard et al. [Maitin-Shepard et al., 2010] develop the first system with two generic robotic arms and a pair of cameras that is able to pick up a towel reliably. The robot detects the corners of an article of clothing based on stereo depth information to decide grasp points. This system gives reasonable performance on picking up a randomly dropped towel from a table and grasping corner point when hanging the towel from its gripper. Ramisa et al. [Ramisa et al., 2012] target on grasping highly wrinkled clothes and demonstrated a good performance on detecting collars in deformed polo shirts, using the depth acquired from a Kinect camera.

Cusumano-Towner [Cusumano-Towner et al., 2011] proposed a method that brings garment from an unknown configuration into a desired configuration while simultaneously determining its identity, which gives promising results. This system also involves a pair of cameras to gain death information. Sun et al. [Sun et al., 2015] present an accurate garment surface analysis using an active stereo robot head that applied to dual-arm on-table cloth flattening. The system is able to parse the high-dimensional configuration space of a garment by detecting and quantifying wrinkles, and also gives efficient flattening strategy. The authors also compared the flattening performance between using stereo cameras and when using a kinect-like depth sensor, and show that their stereo cameras outperforms the kinect-like depth sensor, due to the depth sensor produces noisier depth map.

Bersch [Bersch et al., 2011] presented a robotic system capable of autonomously folding a T-shirt that is placed onto a table in a random configuration. Their algorithm includes a novel fold detection and grasp generation strategy, which suggests grasp poses on cloth folds. To let the robot get perception about the cloth, the shirt is held up by one arm and

rotated by 360 degrees to record stereo image data, and furthermore, 3D model of the shirt would be generated. Van Den Berg et al. [Van Den Berg et al., 2011] use a gravity-based folding algorithm to fold clothes like short and long sleeve tops and pants. The core of their approach is to give the geometry of the cloth via the stereo information extracted from robot head.

Willimon et al. [Willimon et al., 2011a] define a model that describes how to unfold laundry into a flat canonical position automatically. First, initial wrinkles/folds are removed without depth information (accomplished with a single overhead camera). Then the second step is to remove more difficult folds using depth information. Doumanoglou et al. [Doumanoglou et al., 2014b] propose a framework that could be applied to autonomously robot cloth unfolding, addressing the problem of best viewpoint selection in classification, grasp point and pose estimation of garments. Their raw depth data of the garment is captured from a depth sensor.

One common challenge of all those tasks is to develop a robotic vision system that is able to perceive the clothes with sufficient detail, to allow the robotic components to manipulate them (e.g. grasping the cloth wrinkles). In general, there are three classic types of sensor devices that can be mounted on a range of robots to acquire 3D data.

- Stereo camera: A sensor in which 3D measurements are obtained by finding correspondence between two slightly offset images.
- Kinect: A sensor which obtains 3D data by projecting a structured light pattern onto the scene and infer depth from the deformation of that pattern.
- Laser: A sensor which uses a laser beam to determine the distance to an object by firing a laser into the environment and measuring time-of-flight of reflection.

Different sensors have different characters. The laser sensor is accurate on large scales (i.e. up to 80 m) and does not require too much post-processing to produce 3D points. However, this does not suit well for short distance tasks (i.e. robotic cloth manipulation), The raw range image data generated directly by Kinect is noisy, but by providing post-processing, excellent depth data could be achieved. Other than that, Kinect could only provide low-resolution depth map, which is not suitable for some of the applications that require high resolution and accurate depth maps. Due to high resolution and close to camera scene requirement, stereo camera would be the best choice, because it normally provides high resolution image and the depth maps calculated by state-of-the-art stereo matching algorithms are also quite accurate [Scharstein and Szeliski, 2002].

Part II

On the Acceleration of Stereo Matching

Chapter 3

CPU Acceleration

We start our Part II of this thesis, aiming to understand the mechanism of multi-core CPU and GPU parallel processing, and how these two different parallelism enabled architectures could benefit a dense stereo-correspondence algorithm. In this chapter, we focus our investigations on studying multi-core CPUs for accelerating stereo matching algorithms using parallel computing.

Since it normally does not require lots of programming work to parallelize an algorithm on a multi-core CPU [Gadioli et al., 2014], several researchers [Mattoccia, 2010][Gehrig and Rabe, 2010] have attempted to investigate the performance improvement of stereo matching when using several cores (e.g. 4 cores). However, to author's knowledge, there is no research performed on how matching algorithm performances scales when there are many more cores (e.g. 64 cores), when we start our research. Therefore we perform our acceleration research on a stereo matching algorithm as an example and try to find out ultimately how many times can the algorithm be speed up, and the underlying reasons. We specifically utilize Vector Pascal, a dialect of Pascal designed to make efficient use of the multi-media instruction sets of recent multi-core processors, to accelerate a correlation-based stereo matching algorithm, MSSM (Multiple Scale Signal Matching) [van Hoff, 1993], on various multi-core CPU architectures. This chapter addresses our research questions **RQ 1**, **RQ 2** and **RQ 3**, as specified in Section 1.2.1.

> **RQ 1**: How much speed-up can multi-core CPUs offer on accelerating stereo matching algorithms? Is there an acceleration plateau regardless of further increase of the number of CPU cores?
>
> **RQ 2**: How do different CPU system architectures (such as 486, Pentium, Pentium 4 and Sandybridge) affect the acceleration performance?
>
> **RQ 3**: After decomposing the stereo matching algorithms into different sub-components, how do different sub-tasks perform differently in terms of acceleration? Can we further

infer from the algorithmic perspective, which characteristics can benefit more from the multi-core CPU acceleration?

The remainder of this chapter is organized as follows. In Section 3.1, we discuss the previous research work relating to accelerate algorithms using multi-core CPU. Section 3.2 describes the pyramid image matching algorithm, whereas in Section 3.3 we present Vector Pascal and gives detailed explanation on how to utilize it to implement the algorithm on Multi-core CPU. Section 3.4 presents efficiency performance for overall and sub-algorithm, as well as efficiency comparison for different CPU architectures. We conclude the chapter in Section 3.5 by summarising all the findings and discussing the implications of our results.

3.1 Related Work

Recent applications (e.g. mobile robots and autonomous vehicles) require fast stereo matching processing. A single core scalar CPU is too slow to efficiently provide high resolution stereo with modern cameras. This has motivated research into parallelizing the problem.

Mattoccia [Mattoccia, 2010] takes advantage of coarse-grained thread-level parallelism on multi-core CPU for stereo matching algorithm. The stereo pair is split according to the number of available cores, and by comparing execution time on different number of cores, the author found that the execution matching time can be dramatically reduced. He also points out that thread-level parallelism can be effectively coupled with data-level parallelism (i.e. executing the same operation on multiple data by means of SIMD instructions). Gehrig and Rabe [Gehrig and Rabe, 2010] carefully analyze a semi-global matching algorithm, and implement parallelization for the most time consuming part via SIMD and multi-core CPU parallelism. They announce that their CPU implementation outperforms an early version GPU variant in terms of speed.

Other than the works above, many researchers [Arndt et al., 2013][Gadioli et al., 2014][Spangenberg et al., 2014][Humenberger et al., 2010] implement the same algorithm on different architectures (i.e. multi-core CPU, GPU, embedded system) to compare the performance. These platforms expose different programmability complexities. In general, GPU architecture achieves much higher throughput, however, it requires more specialized application code, while multi-core platforms are more flexible and provide good code portability.

3.2 Parallel Pyramid Matcher

The initial idea of this matcher comes from the Multiple Scale Signal Matching (MSSM) algorithm [van Hoff, 1993], a correlation-based matching algorithm, which we also refer as

pyramidal stereo matching algorithm in the rest of the thesis. Most of the matching algorithms are applied on rectified stereo images with small image sizes (e.g. 640 $\times$ 480, 320 $\times$ 240), but MSSM algorithm can be applied to large unrectified images (e.g. 4928 $\times$ 3264) which fits the requirement of CloPeMa project very well. Furthermore, MSSM algorithm has been proved to be quite effective, which can provide accurate disparity maps [Siebert and Urquhart, 1995]. However, one shortage of the serial MSSM algorithm is that for the current application, it was no longer adequate due to its slow processing time. For the original Java version on a single core CPU, it takes around 20 minutes to process a single pair of 16 MegaPixel images [Cockshott et al., 2012]. Therefore, we tend to find a way to accelerate the whole process. Parallel processing is widely used in the image processing field. And for stereo matching algorithm, we normally assign the same task to all pixels, which is very suitable for parallel processing. So in this chapter, we will discuss the Parallel Pyramid Matcher (PPM) with various optimizations for multi-core CPU architectures, which improve the speed of MSSM algorithm dramatically.

The aim of this pyramidal algorithm is to compute a disparity map for a stereo-pair of images. The disparity map refers a two dimensional array of displacement vectors which map the pixels in one image onto their corresponding pixel in the other. In the case of un-rectified images, i.e. images from converged stereo cameras, there are two disparity maps, one for horizontal displacements and one for vertical displacements, specifying orthogonal components of the disparities. The algorithm also produces a confidence map, which assigns a confidence to each disparity vector. The confidence map is a measure of the likelihood that the found correspondence is the best match for the pixel. In general, only pixels with high confidences lead to reliable results.

Pyramid Creation

The matching algorithm employs a pyramid representation (Figure 3.2), which is applied to the input images and then serves to facilitate signal matching at multiple scales [van Hoff, 1992]. In this scheme, an initial estimate for the disparity is computed at a low resolution and the initial disparity estimate from this scale is refined at higher resolutions until the target resolution is achieved.

Before we proceed, we briefly introduce the image pyramid algorithm (i.e. how to generate the image pyramid for one image, such as the left image in Figure 3.2). Pyramid is a type of multi-scale signal representation, in which a signal or an image is subject to repeated smoothing and sub-sampling. A variety of different smoothing kernel was proposed for generating pyramid. Among these, Gaussian kernel is the most supportable and widely used one. Pyramids provide increased storage efficiency and a uniform basis for analysis over scales. There are two main types of pyramids [Cyganek and Siebert, 2009] (shown in Figure 3.2):

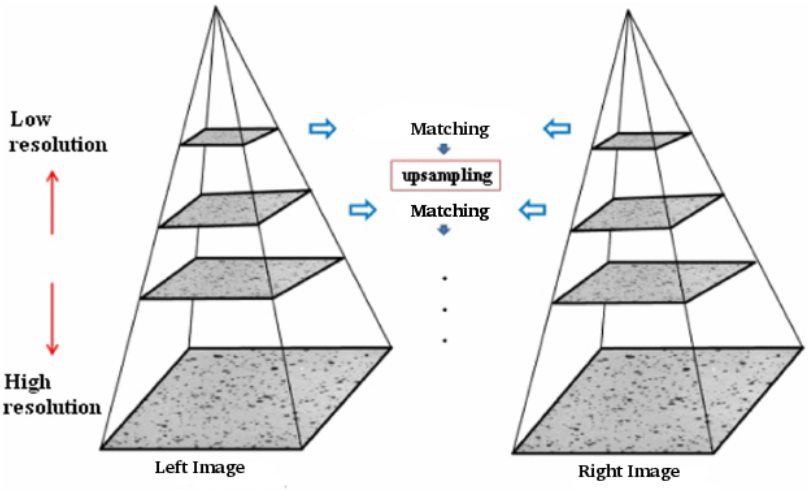


Figure 3.1: Pyramid representation of stereo input image to perform matching at multiple scales

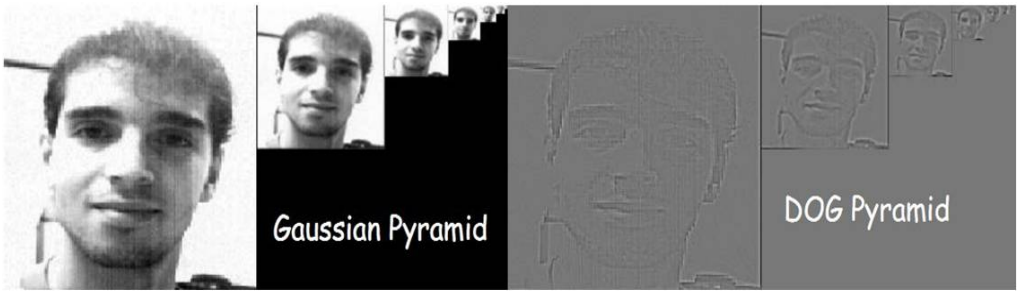


Figure 3.2: Image Pyramid

Gaussian (low pass) pyramids and DOG (Difference of Gaussian) (band pass) pyramids.

- A low pass (Gaussian) pyramid is generated by first smoothing the image with an appropriate smoothing filter and then sub-sampling the smoothed image, usually by a factor of two along each coordinate direction. This smoothed image is then subjected to the same processing, resulting in a yet smaller image. As this process proceeds, the result will be a set of gradually more smoothed images, where in addition the spatial sampling density decreases level by level. If illustrated graphically, this multi-scale representation will look like a pyramid, from which the name has been obtained.

- A band pass (DOG) pyramid is obtained by forming the difference between adjacent levels in a pyramid, where in addition some kind of interpolation is performed between representations at adjacent levels of resolution, to enable the computation of pixel-wise differences.

Pixel Correlation

In order to refine the disparities at each level of the input pyramids, the algorithm attempts to maximize the correlation between pixels in windowed regions of the left and right images. The windowed correlation is weighted by doing a convolution with a Gaussian kernel, which is computed as follows

$$cor_{l,r} = \frac{cov_{l,r}(x,y)}{\sqrt{var_l(x,y)}\sqrt{var_r(x,y)}} \tag{3.1}$$

where the covariance is computed as

$$cov_{l,r}(x,y) = \sum_u \sum_v f_l(x+u, y+v) f_r(x+u, y+v) w(u,v) \tag{3.2}$$

and the variance is computed as

$$var_i(x,y) = \sum_u \sum_v f_i(x+u, y+v) f_i(x+u, y+v) w(u,v) \tag{3.3}$$

In the above equations, u and v define the size of the window, $w(u,v)$ define the Gaussian weight and $f(x,y)$ define the pixel intensity at the (x,y) coordinates. The window in the left image is named as reference window and the window in the right image is called search window. By moving the search window, a local correlation surface is created for each pixel position. The search window is moved in four directions (up, down, left and right) using a one pixel search step. If we include the null move, there are in total five local correlation maps.

Polynomial Maximization

Having obtained five correlation coefficient matrices, 2nd order polynomial maximization is applied to the corresponding elements of the matrices. The task is to find, for each pixel, the local maximum of the curve. If the local maximum is found to be more than one pixel away from the current position the relative displacement is clipped to ± 1. The confidence map, is computed from the local correlations after the move has been done.

Inter-Scale Disparity Refinement

At each scale-level, the disparities are anisotropically diffused using the confidence map. The effect is that disparities found in areas of high confidence tend to diffuse into adjacent areas of low confidence. Suppose $InitConf$ is the confidence before search and $InitDisp_{xy}$

is the disparity and $ConfHV$ is the confidence after search, then

$$InitConf = ConfHV + 0.75 * (InitConf - ConfHV) \tag{3.4}$$

The disparity matrices are then weighted by the confidence matrix as the equation below.

$$\begin{aligned} Disp_{xy} &= \sum_{I \in \Delta} InitDisp_{(xy+I)} \times InitConf_{(xy+I)} \\ \Delta &= \{[0,0],[0,1],[0,-1],[1,0],[-1,0]\} \end{aligned} \tag{3.5}$$

In the above equation, $Disp_{xy}$ is the output disparity. This smoothing process is iterated for a certain number of times to refine disparity map.

Rescale

From current scale to higher resolution scale, interpolation is necessary for each pixel in the disparity map. An interpolated pixel is derived from its four neighbours. If

$$x_0 = floor(x) \quad y_0 = floor(y) \tag{3.6}$$

$$a = x - x_0 \quad b = y - y_0 \tag{3.7}$$

then

$$\begin{aligned} f_{interpo}(x,y) = (1-a)(1-b)f(x_0,y_0) + (1-a)bf(x_0,y_0+1) \\ + a(1-b)f(x_0+1,y_0) + a \cdot b \cdot f(x_0+1,y_0+1) \end{aligned} \tag{3.8}$$

The interpolated result is the initial disparity map for the next higher scale resolution.

Iteratively implementing the same matching algorithm on each scale until the highest resolution scale, the final and more accurate disparity map is produced.

3.3 Vector Pascal Multi-core CPU Parallel Theory

Vector Pascal[1] [Cockshott, 2004][Cockshott, 2011] is used for implementation of parallelization of the stereo matching algorithm described above on the multi-core CPU architectures. Vector Pascal, an open source compiler for Pascal, is designed to support efficient expression of algorithms using the SIMD (Single Instruction, Multiple Data) model of computation. It

[1] https://sourceforge.net/projects/vectorpascalcom/

is a dialect of Pascal designed to make efficient use of the multi-media instruction sets of recent processors. It supports data parallel operations and saturated arithmetic.

3.3.1 Compiler Options

In Vector Pascal, all operators are overloaded, so that they can operate on arrays and vectors as well as scalars. Using compiler flags, a single program can be compiled, with differing levels of parallelism, to target a range of microprocessors.

- *-cpu **CGFLAG*** specify the code generator to be used. In our study, four classic options shown in table 3.1 are tested and reported. Full list of supported code generators can be found in [Cockshott, 2011]. The selected four code generators are all using NASM[2] assembler, but with different instruction-set.

  The four instruction-sets in the table are IA-32 (Intel Architecture, 32-bit), MMX (Multimedia Extensions), SSE2 (Streaming SIMD Extensions 2), AVX (Advanced Vector Extensions). Basically, MMX, SSE2 and AVX are all able to do parallel SIMD operations, but with different capability. IA-32 holds 32-bit registers; MMX holds 64-bit registers; SSE2 holds 128-bit registers; AVX holds 256-bit registers. For example, applying SIMD on MMX means that instead of using the whole register for a single 64-bit integer, it is possible to process two 32-bit integers, four 16-bit integers, or eight 8-bit integers concurrently. Same for SSE2 and AVX, each generation doubles the capability of parallel SIMD operation.

Table 3.1: Code generators supported

CGFLAG	Description
IA32	generates code for the Intel 486 with IA-32 instruction-set uses the NASM assembler
Pentium	generates code for the Intel P6 with MMX instruction-set uses the NASM assembler
P4	generates code for the Intel PIV family and Athlon XP with SSE2 instruction-set uses the NASM assembler
AVX32	generates 32 bit code for the AVX instruction-set uses the NASM assembler

- *-cores **n*** generate code for n threads executing in parallel. (Cores means threads for this compiler flag.)

  Figure 3.3 is an example to illustrate how single-core and multi-core CPU works. Here we use AVX instruction as an example. The data is a digital representation of image.

[2]The Netwide Assembler (NASM) is an assembler and disassembler for the Intel x86 architecture.

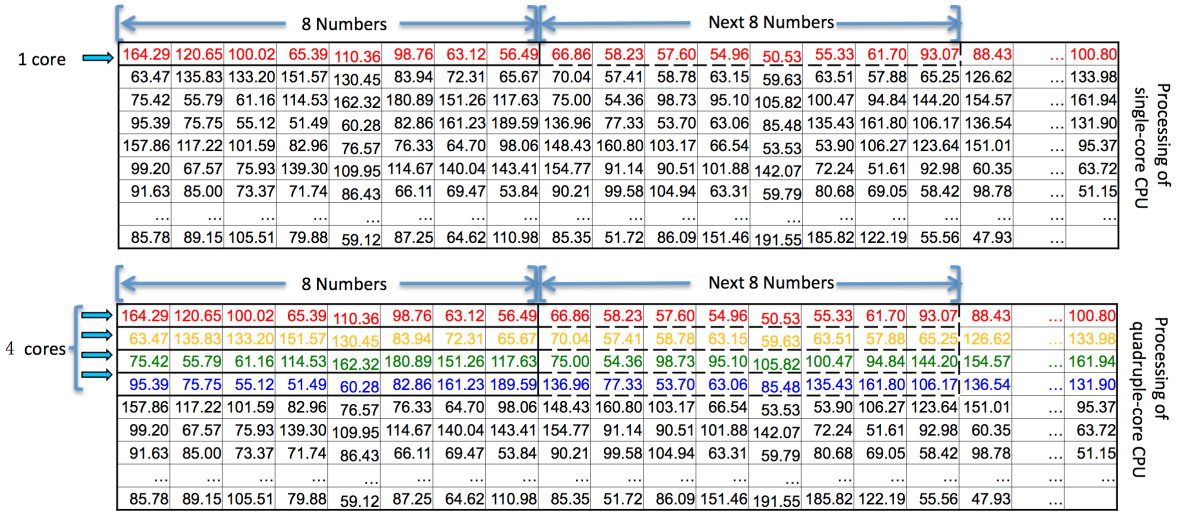


Figure 3.3: Processing of CPUs

Each number represents the color level of a pixel and it ranges from 0 to 255 (e.g. 0 means pure black while 255 means pure white in the gray image). Suppose all the pixel values in the matrix are divided by 2, CPU with different number of cores are used to achieve this.

Single-core CPU with AVX instruction set deals with pixels row by row, and in each row, it runs 8 floating point numbers by 8 numbers, because AVX can process 8 floating point numbers once. For multi-core CPU (cores 4 for example), one core deals with one row, so four rows can be processed at the same time. In total, it can process 32 floating point numbers data at a given time. In this way, theoretically the speed of 4-core CPU is four times faster than single-core CPU. However, since not all the instructions can be paralleled, we therefore cannot assume that we can achieve the theoretical acceleration performance.

Here we only use 4-core CPU as an example. In the experiments, "4" can be instead by any integers that equal or smaller than the total available threads of a multi-core CPU. This means in Vector Pascal, it is possible to control the number of active cores/threads. Here we use an example to explain the strategy of thread assignment. If we have a 4-core CPU, but only 2 threads are needed, then core 1 and core 2 would be active, but core 3 and core 4 do nothing. While if we have a 4-core CPU, which support multi-thread (e.g. 2 threads for each core), then the number of total available threads will be 8. And if 5 threads are needed, the CPU will first assign tasks to one thread of each core (core 1 to 4), then for the fifth thread, it will be assigned to core 1 again but use the other available thread.

- *-D **symbol*** define compiler pre-processor flag for conditional compilation. The option *-DBUSYWAIT* is able to change default setting of task scheduling from pthread

semaphores to spin-locks (busy waiting).

The Vector Pascal compiler uses pthreads to support multi-core parallelism over matrices. The default setting of task scheduling is using pthread semaphores, but another option, spin-locks (busy waiting), can be used for machines with very large numbers of cores.

The default setting "semaphore" is a variable or abstract data type that is used for controlling access, by multiple processes, to a common resource in a parallel programming or a multi user environment. It has a counter and will allow itself being acquired by one or several threads, depending on what value being posted and what its maximum allowable value is. If a semaphore cannot be acquired, it blocks, giving up CPU time to a different thread that is ready to run. This means that a few milliseconds pass before the thread is scheduled again.

Unlike semaphores, a spin-lock is a lock which causes a thread trying to acquire it to simply wait in a loop while repeatedly checking if the lock is available. Once acquired, spin-locks will usually be held until they are explicitly released. Although they avoid overhead from operating system process re-scheduling or context switching, spin-locks are efficient if threads are only likely to be blocked for a short period, since they prevent the core in question from being used by any other process.

3.3.2 Implementation

Image convolution and interpolation are the most two important sub-algorithms for the pyramidal stereo matching. Their implementations can largely decide the overall performance, so special analyses for them are worth taken.

Convolution of an image by a matrix of real numbers can be used to smooth an image, depending on the matrix used. If A is an output image, K a convolution matrix, then if B is the convolved image

$$B_{y,x} = \sum_i \sum_j A_{y+i,x+j} K_{i,j} \tag{3.9}$$

where i and j are the height and width of the filter kernel [Cockshott et al., 2014]. In general, $i * j$ multiplications are required for each output pixel. Whereas the separable filter, a specialisation of the more general convolution matrix, is algorithmically more efficient to implement. It can be expressed as two consecutive one-dimensional convolution operations on the data, one on the rows on the image, and one on the columns. Then only $n+m$ multiplications for each output pixel are required. If $\mathbf{k}$ is a convolution vector, then the corresponding matrix K is such that $K_{i,j} = \mathbf{k}_i \mathbf{k}_j$.

Given a starting image A as a two dimensional array of pixels, and a three element kernel c_1, c_2, c_3 for example, the algorithm first forms a temporary array T whose elements are the weighted sum of adjacent rows.

$$T_{y,x} = c_1 A_{y-1,x} + c_2 A_{y,x} + c_3 A_{y+1,x} \tag{3.10}$$

Then in a second phase it sets the original image to be the weighted sum of the columns of the temporary array.

$$A_{y,x} = c_1 T_{y,x-1} + c_2 T_{y,x} + c_3 T_{y,x+1} \tag{3.11}$$

Since the convolution is defined over the neighbours of the pixel, so the boundary pixels show as a special case who misses one neighbour. Our solution is simplicity performs only vertical convolutions on the left and right edges and horizontal convolutions on the top and bottom lines of the image. Below is the algorithm used in Vector Pascal for the parallel convolution of images. Most of the work is done by the function `MA5` (Kernel size: five) which assigns the sum of the input planes `p1` to `p5` that weighted by the elements of the kernel `k`, to the output image plane `acc` ;

Listing 3.1: Vector Pascal code for convolution

```
PROCEDURE MA5(VAR acc, p1,p2,p3,p4,p5:plane; k:kernel);
BEGIN
  acc:= p1*k[1] +p2*k[2] +p3*k[3] +p4*k[4] +p5*k[5];
END;
```

The next procedure convolves a plane by applying the vertical and horizontal convolutions in turn. Flags `writetop` and `writebottom` indicate if the top and bottom margins are to be written or ignored.

```
PROCEDURE  convp(VAR p,T:plane; writetop,writebottom:boolean);
VAR r,c,k,i,lo,hi,bm,tm,mid,l:integer;
BEGIN
```

This sequence performs a vertical convolution of the rows of the plane `p` and places the result in the temporary plane `T`. On successive calls to `MA5`, the procedure is passed first a set of vertical subplanes offset by one row, and then a set of horizontal subplanes offset by one column.

```
margin:=(kernel_size div 2);
(* margin specifies how many rows at top and bottom
are left out *)
r:=p.rows; lo:= margin; hi:=r-margin;
MA5(T[lo..hi],p[0..hi-lo+0],p[1..hi-lo+1],p[2..hi-lo+2],
     p[3..hi-lo+3],p[4..hi-lo+4],kerr)
l:=margin -1;
```

```
    if l > T.rows then l:= T.rows;
    for k:=0 to l do BEGIN (* handle top and bottom margin *)
       T[k]:=p[k];
       T[r-k]:=p[r-k];
    END;
    p:=T;
```

Now perform a horizontal convolution of the plane `T` and place the result in `p`.

```
    c:=p.cols;  hi:= c-margin;
    bm:=0;tm:=r;
    if not writetop then bm:=margin;
    if not writebottom then tm:=r-margin;
    MA5(p [bm..tm,lo..hi],t[bm..tm,0..hi-lo+0],
        t[bm..tm,1..hi-lo+1], t[bm..tm,2..hi-lo+2],
        t[bm..tm,3..hi-lo+3],t[bm..tm,4..hi-lo+4], kerr)
    for k:=0 to margin-1 do BEGIN (* left and right margin *)
       p[bm..tm,k]:=T[bm..tm,k];
       p[bm..tm,c-k]:=T[bm..tm,c-k];
    END;
  END;
```

For image convolution, separable filters offers benefit to reduce the arithmetic complexity, and operations from our Vector Pascal implementation can be vectorised well on various instruction set.

Compared to image convolution, the implementation of interpolation algorithm is simpler. Basically for each pixel, it requires the scale rate for both vertical (`dy`) and horizontal (`dx`) directions, as well as information of all four neighbours as shown in Listing 3.2. This uses array `vert` and `horiz` to index the source image array.

Listing 3.2: Vector Pascal interpolation routine

```
PROCEDURE planeinterpolate ( var a,b:Plane);
  VAR dy,dx:real;
          vert,horiz:pivec;  vr,hr:prvec;
```

Type `pivec` is a pointer to an array of integer, while `prvec` demonstrates a pointer to an array of real.

```
  BEGIN
   dy:=(a.rows+1)/(b.rows+1);
   dx:=(a.cols+1)/(b.cols+1);
   new (vert, b.maxrow); new(vr,b.maxrow);
   vert^:=trunc( iota [0] *dy); vr^:=iota [0]*dy -vert^;
   new( horiz,b.maxcol); new(hr,b.maxcol);
   horiz^:=trunc( iota [0] *dx); hr^:=iota [0] *dx-horiz^;
```

Until now, we have computed vectors of horizontal and vertical positions in `horiz` and `vert`, the horizontal and vertical residuals in `hr` and `vr`. Note that `iota` is the implicit index vector for the expression, so `iota[0]`, `iota[1]` give the x,y position in the image array.

```
B [ 0..B.maxrow-1,0..B.maxcol-1]:=
     (A[vert^[iota[0]], horiz^[iota[1]]]* (1-hr^[iota [1]]) +
      A[vert^[iota[0]], horiz^[iota[1]]+1]* hr^[iota [1]] ) *
     (1-vr^[iota[0]]) +
     (A[vert^[iota[0]]+1, horiz^[iota[1]]]* (1-hr^[iota [1]]) +
      A[vert^[iota[0]]+1, horiz^[iota[1]]+1]* hr^[iota [1]] ) *
      vr^[iota[0]];
 dispose(vert);dispose(horiz);dispose(vr);dispose(hr);
END;
```

From the equation above, interpolation result could be achieved. For the multi-core parallelisation, SIMD construct is used, to split the data to run across cores.

Moreover, the Vector Pascal version of whole Parallel Pyramid Matcher source code is public available on Sourceforge[3] for researchers to use.

3.4 Experiments

In this section, we first describe the experimental setups (Section 3.4.1), i.e. the systems and data used in our experiments. Then we discuss our findings on accelerating the matching algorithm on multi-core CPUs (Section 3.4.2). We finally analyze the acceleration impact on CPU (Section 3.4.3) and compare performances on various CPU architectures (Section 3.4.4).

3.4.1 Experimental Setup

The experiments have been conducted on four different multi-core CPU systems. The CPU systems are: a four chip AMD Opteron processor 6366HE system, a two chip Intel Xeon processor E5-2440 system, a single chip system using an Intel Xeon processor E5-2620 and an Intel Core i5-2400 processor (See Table 3.2 for details).

The test data for the matching algorithm originates from a stereo-pair of cameras of the binocular robot-head used for CloPeMa. The head is fitted with Nikon DSLR D5100 cameras that are capable of capturing images at 16 mega pixels [Cockshott et al., 2012]. The dataset consists of a set of stereo-pairs of images, covering a range of different cloth textures. These

[3] https://sourceforge.net/p/vectorpascalcom/code/HEAD/tree/tests/matchtest.pas

Table 3.2: Specification of different processors (S for per socket, C for per core) used to accelerate stereo matching

Parameter	AMD Opteron 6366HE	Intel Xeon E5-2440	Intel Xeon E5-2620	Intel Core i5-2400
Chips	4	2	1	1
Cores and Threads	16 and 16 (S)	6 and 12 (S)	6 and 12	4
Clock Speed	1.8GHz	2.4 GHz	2 GHz	3.1 GHz
Memory Capacity	504GB	24GB	16 GB	8 GB
Memory Technology	DDR3	DDR3	DDR3	DDR3
Memory Speed	1333 MHz	1333 MHz	1333 MHz	1333 MHz
Memory Data Width	4×64 bits (S)	2×64 bits (S)	64 bits	64 bits
Peak Memory Bandwidth	51.2GB/s (S)	32 GB/s (S)	42.6 GB/s	21 GB/s
Data Caches	16 x 16 KB L1 (S)	384 KB L1 (C)	32 KB L1 (C)	32 KB L1 (C)
	8 x 2 MB L2 (S)	1.50 MB L2 (C)	256 KB L2 (C)	256 KB L2 (C)
	2 x 8 MB L3 (S)	15 MB L3 (S)	15 MB L3	6 MB L3

images were taken as full high resolution colour images with 4928 by 3264 pixels. In the system, the images are represented as arrays of 32 bit floating point numbers to maintain accuracy.

3.4.2 General Results

Figure 3.4 shows the overall performance of the parallelised pyramid matching algorithm on three multi-core CPUs specified in Table 3.2, with the number of threads varied from the minimum to the maximum of a given system. The x-axis corresponds to the number of threads, while the y-axis is the execution time in seconds. The axes are shown in log scale in order to present the figure more concisely. Note that in this experiments, all runs are done using pthread spin-locks as the setting of task scheduling. Also, all the CPUs support AVX instructions and the code was compiled to use these instructions. Since these instructions allow 8 fold SIMD parallelisation of vector operations, the single core performance is already substantially faster than the scalar Java performance. For fair comparison, single core performance without SIMD is shown in Figure 3.4, which uses X87 instructions (32-bit instructions). It can be observed that the single-thread using AVX instructions is twice as fast as the one using X87 instructions.

The performance of three machines show a similar trend. Not surprisingly, as the number of threads increased, execution time decreased. In the initial range, up to 10 threads the performance follows an approximately linear trend in log log space, which is equivalent to a power-law functional form. However, one interesting finding for the 64 cores AMD is, that after around 30 threads, the running time stopped decreasing, and increased slightly. This trend could not be verified on the other two machines, since the lower number of cores did not allow us to detect whether a similar turning point exists.

This scenario can not simply be explained by Amdahl's Law [Amdahl, 1967]. Amdahl's law

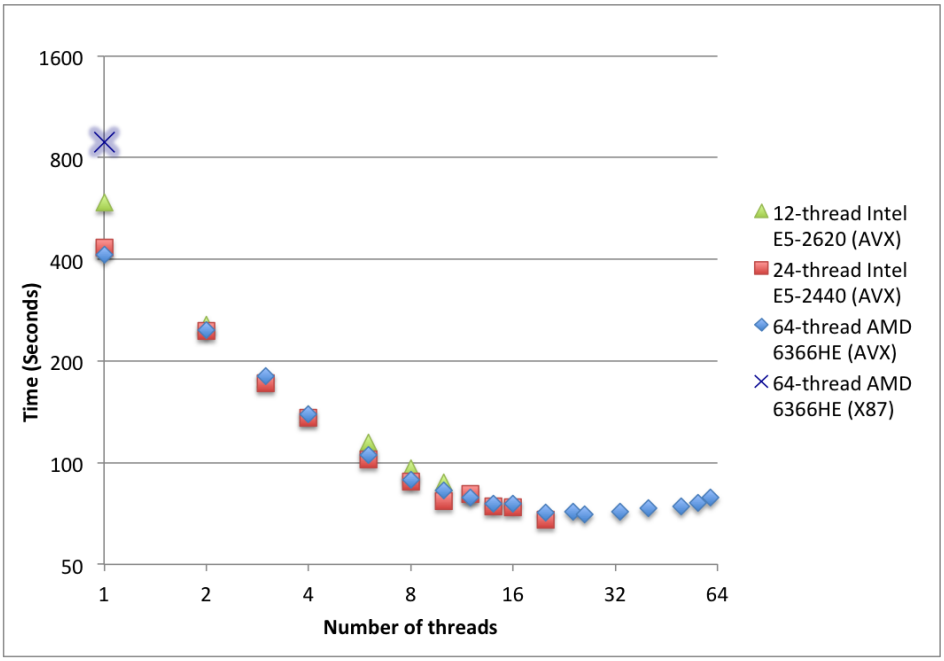


Figure 3.4: Log/Log plot of matching performance on different CPU based machines

states that if P is the proportion of a program that can be parallelized, and $(1 - P)$ is the proportion that cannot be parallelized, the maximum speed-up that can be achieved using N processors is

$$S(N) = \frac{1}{(1 - P) + \frac{P}{N}} \tag{3.12}$$

In the limit, as N tends to be infinity, the maximum speed-up tends to be $1/(1 - P)$.

Theoretically, if there are enough cores (threads) to run an algorithm, the execution time of the part that can be parallelized will tend to be zero. So in theory, the optimal performance is predictable, because it only depends on the serial part. But in practice, in the multi-core environment, the CPU takes time to assign different tasks to different cores. As the number of used cores grows, the allocation time increases, eventually the extra thread scheduling time can be expected to outweigh the speed-up due to parallelization.

However, it is not clear that this is necessarily the cause of the bottoming out of run time at 30 threads. Another possible cause relates to memory bandwidth. Depending on how the mapping of physical to virtual memory is carried out, the images being processed may have been spread over the memory channels of all 4 processor chips on the Opteron system, or may have been entirely allocated within the memory controlled by one of the chips. In the latter case the bottoming out of performance after 30 threads could represent the saturation of the memory bandwidth of that chip. The input and output pyramids occupy around 2.5 GB memory in total. In addition, there are in the CPU version, various temporary buffers of comparable sizes that have to be manipulated repeatedly as the algorithm iterates. It thus places considerable demands on the memory channels.

Overall, the best multi-core CPU architecture gives about $12\times$ acceleration of the stereo matching algorithm compared to the original single-core CPU performance.

3.4.3 Acceleration Analysis of Sub-Algorithms

The matching algorithm can be separated into several parts, as mentioned in Section 3.2: pyramid creation, pixel correlation, polynomial maximization, inter-scale disparity refinement, rescale and others. Since similar results were observed for up to 30 threads across all three machines and only the 64-core AMD machine showed a performance bottleneck, the tests for the key sub-parts of the algorithm were only performed on the 64-core system.

Execution time for each of the six parts over different number of cores (threads) is shown in Figure 3.5. The figure illustrates that as the number of cores varies, different parts show different performance changes. The two outstanding parts in Figure 3.5 are pixel correlation and polynomial maximization. Those two parts have similar execution times for single thread execution. However, when executed with multiple-threads, there are two major differences that can be observed: (1) the acceleration rate of polynomial maximization continues to improve until about 40 cores while the speed-up of pixel correlation reaches a peak and remains stable after around 12 cores; (2) the acceleration rate of polynomial maximization is much faster than pixel correlation.

To look a bit closer, among the sub-tasks, polynomial maximization gives the best parallel performance with 23 times speed-up on CPU. On the other hand, the most time consuming part of the algorithm is pixel correlation. The multi-core CPU performance reaches a 11 times speed-up against single thread.

This demonstrates that polynomial maximization and pixel correlation exhibit different acceleration rates. Two basic image processing algorithms, linear interpolation and image convolution that used in our matching algorithm, also has similar characteristic as polynomial maximization and pixel correlation respectively. Since we want to understand the underlying reasons for the differences in acceleration, linear interpolation and image convolution are examined in more detail. These operations are much simpler, allowing for a more refined performance analysis.

Figure 3.6 shows the linear interpolation and image convolution performance on different number of cores. Y axis is scaled by Log function. As the number of cores increases, both execution times decrease. The acceleration of interpolation demonstrates a good, power law like, curve with a performance improvement of 36 times. Linear interpolation only contains one step of array access and is easily parallelized on a multi-core architecture. Polynomial maximization and rescale, which rely on linear interpolation, follow therefore a similar speed-up trend.

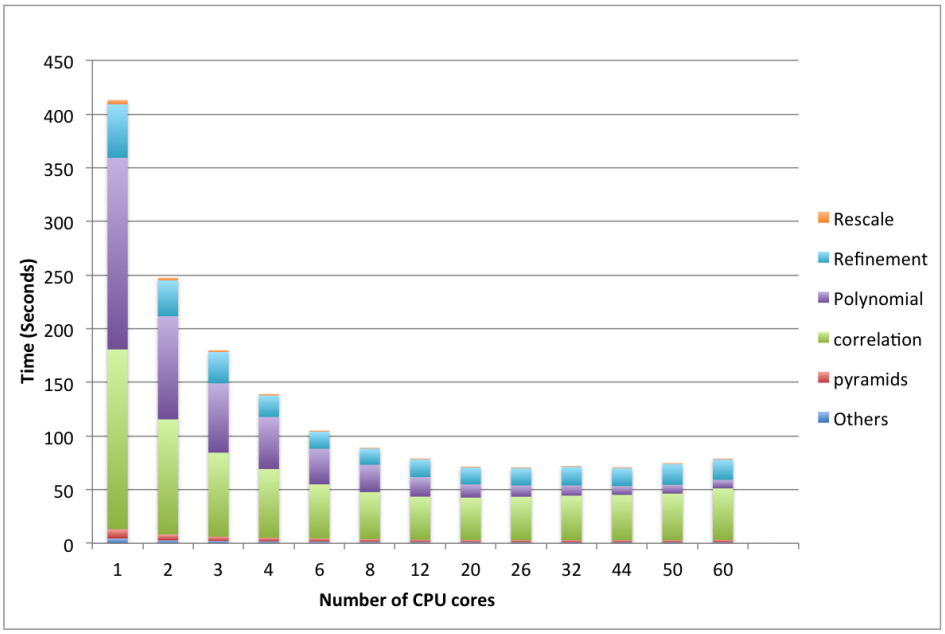


Figure 3.5: Performance of matching algorithm breakdown on AMD 6366HE CPU (AVX instructions)

The performance trend for convolution, on the other hand, is similar to the trends for overall matching performance and pixel correlation performance. Execution time decreases as the number of cores increases, until about 10 cores are reached. After that, the execution time decreases very slowly. It only gives about 3 times speed-up on CPU, which is close to pixel correlation performance. Note that all these tests are using AVX instructions.

A possible explanation is that to perform image convolution, a one dimensional convolution is first applied horizontally and then vertically to the image. Although this method allows straight forward parallelization, the creation of an intermediate result array will limit the speed-up gain since the image data size is typically larger than the CPU cache. Writing to the intermediate array will tend to flush the original data buffer from the caches.

We can observe that although the interpolation is slower than convolution on a single-core CPU, as the number of cores increases, interpolation overtakes convolution. From the previously described experiment, it is understood that different algorithms impose different loads on the memory subsystems. This appears to be the reason as to why the different algorithms show different trends. To summarize, compared to interpolation, although two pass convolution still benefits from multi-core parallelization, this type of convolution algorithm is not well suited to effective parallel acceleration.

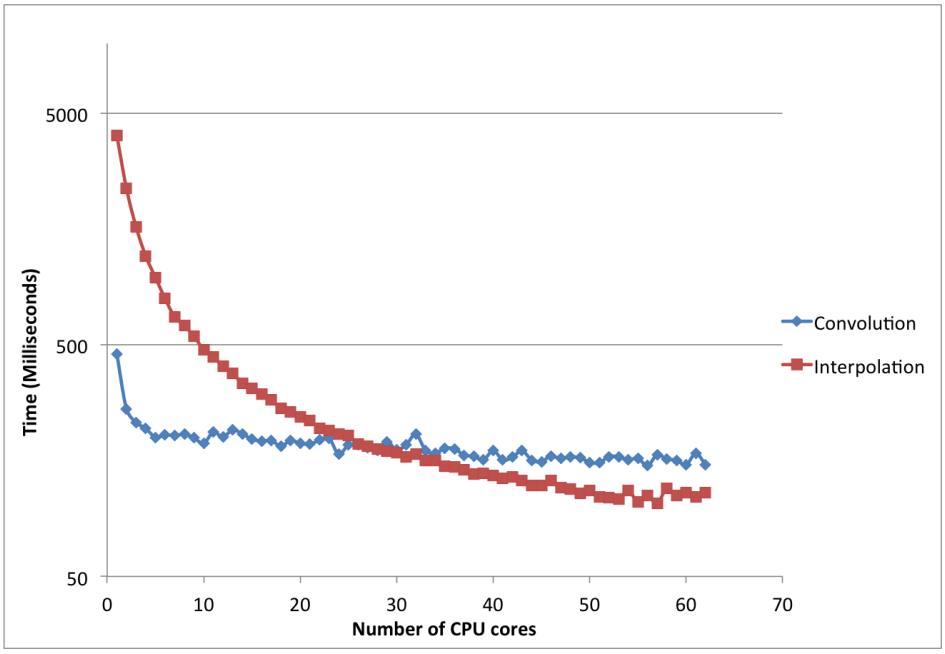


Figure 3.6: Interpolation and convolution performance on different number of CPU (AMD 6366HE) cores using AVX instructions

3.4.4 Comparison of CPU Architectures

We also aim to understand whether different CPU architectures can influence the acceleration. We expect the more advanced the hardware architecture, the greater the acceleration will be. To simplify the process for this sanity check, we perform the experiments only on the image pyramid algorithm (a sub-component of the pyramidal stereo matching). We conduct the experiment on the machine with Intel Core i5-2400 CPU. In this experiment, we use various Gaussian kernel sizes (3 and 5) and image sizes (512×512 pixels, 1024×1024 pixels and 2048×2048 pixels), and aim to show their effects.

Table 3.3: Efficiency performance of image pyramid algorithm in various system architectures (in seconds)

Kernel Size	Image Size (Pixels)	CPU			
		Sandybridge (*AVX32*)	Pentium 4 (*P4*)	Pentium (*Pentium*)	486 (*IA32*)
3	512×512	**0.039**	0.052	0.133	0.133
	1024×1024	**0.147**	0.202	0.533	0.537
	2048×2048	**0.628**	0.808	2.136	2.138
5	512×512	**0.07**	0.075	0.191	0.194
	1024×1024	**0.21**	0.292	0.777	0.776
	2048×2048	**0.858**	1.181	3.137	3.174

Due to the fact that the previous CPU architectures do not support parallel processing, therefore we only compare the performance using one core. Different architectures and corresponding code generators are shown in Table 3.3. Sandybridge is the newest architecture and 486 is the oldest one. In each row, the number in bold denotes the calculation time executed by the best performed (i.e. least time) CPU. Notice that in this experiment, all runs are done using default setting of task scheduling "semaphore", because when the processing time is short and the number of cores is not sufficient, different task scheduling settings make no significant difference.

From the tables, we can observe that, not surprisingly, the newest system architecture has a significant improvement, compared with the previous versions. Performance on Sandybridge and Pentium 4 architecture is about 4 times and twice faster than on 486 respectively, while the performance of Pentium is similar to 486. This is due to the development of instruction optimization these years.

3.4.5 Discussion of Accuracy and Validation

Until now, we have reported our main focus, the efficiency performance. However, as an accelerated version of the original sequential algorithm, validation is required in order to make sure that both versions achieve the similar performance in terms of accuracy (effectiveness). We select several image pairs from Glasgow garment dataset [Aragon-Camarasa et al., 2013] as our testbed (more details will be introduced in Section 5.2.3). The percentage of bad pixels (details can be found in Section 5.3) is utilized as our metric to evaluate the output (disparity maps). By comparing the outputs from both the multi-core CPU accelerated and the original sequential algorithms, we found that the results are quite similar and can match to each other for the cloth areas, while for the boundary area of the table and the periphery of the image, the disparities are sometimes not coherent. On average, the bad pixel rate between the original algorithm and this accelerated Vector Pascal version is around 17%.

In theory, the two versions should give exactly the same result. Although the author devotes every efforts to make sure the codes are doing the same thing, the results are still partly different. This may due to the numerical settings of different programming languages, and the accumulation of nuances.

3.5 Conclusions

The objective of this chapter is to study utilizing the multi-core CPU to parallelize stereo matching. The experimental results (Figure 3.4) show significant performance accelerations with multi-core CPUs giving $12\times$ speedup on average (using Vector Pascal programming

language on Sandybridge architecture), compared to SISD single-thread CPU performance. The acceleration plateau of the MSSM stereo matching algorithm (Section 3.2) is achieved at around 30 cores while further increasing the number of cores (even to 64 cores) does not further reduce runtime. When comparing different architectures, it is not surprising that the recent Sandybridge outperforms Pentium 4, Pentium and 486 (Table 3.3). Performance on Sandybridge and Pentium 4 architecture is about 4 times and twice faster than on 486 respectively, while the performance of Pentium is similar to 486. In addition, we found that the optimized acceleration of one stereo matching sub-task "interpolation" has a greater impact on the overall speed-up than another sub-task "convolution" (Figure 3.6). This is because "interpolation" is more suited for parallel execution, but "convolution" currently has a speed-up limitation due to its high demands on storage of intermediate results. This finding can help us better understand the accelerated matching algorithm from the efficiency perspective and potentially can be used to guide the development of better algorithms that are easier to be paralleled.

Chapter 4

GPU Acceleration

In the previous chapter, we focused on utilizing the mechanism of multi-core CPU parallel processing paradigm to accelerate a pyramidal stereo matching algorithm. In this chapter, we start our investigations by studying the difference between GPU and CPU architecture and exploit the specialties of GPU. The same stereo matching algorithm (Parallel Pyramid Matcher) mentioned in Section 3.2 is applied on the GPU platform with various facilities enabled by CUDA, in order to have fair comparison with the multi-core CPU platform. This chapter addresses our research questions **RQ4** and **RQ5**, as specified in Section 1.2.1.

> **RQ 4**: How many times speed up can GPUs achieve on accelerating stereo matching algorithms? How does it compare with multi-core CPUs?
>
> **RQ 5**: How do different stereo matching sub-tasks perform differently in terms of acceleration on GPUs? Does the GPU acceleration trend vary from the one of multi-core CPUs?

The remainder of this chapter is organized as follows. In Section 4.1, we discuss the previous research work relating to accelerate algorithms using GPU. Section 4.2 compares GPU architecture for parallel computing with CPU and present the GPU Strategies for Parallel Pyramid Matcher. Section 4.3 provides the parallel computing analysis of PPM algorithm for GPU. Finally, Section 4.4 outlines our conclusions.

4.1 Related Work

Various researchers have prevoiusly attempted to accelerate stereo matching algorithms using GPUs. The paper of Yang et al. [Yang et al., 2003] is the first one to explore the potential of GPUs to accelerate depth estimation. They effectively used the capability of graphics hardware to warp and process images.

Wang et al. [Wang et al., 2006] use an adaptive aggregation step in a dynamic-programming (DP) stereo framework, which achieved high quality results. Their efficient use of both CPU and GPU as a co-operative system allows it to maintain real-time performance. The GPU performs image warping and aggregation in massive parallelism, and the CPU performs DP which requires more flexible looping and branching capability.

Mei et al. [Mei et al., 2011] also present a GPU-based matching system that could effectively handle various errors in a multi-step refinement process, which gave good performance in both accuracy and speed. Their GPU-friendly system design brings an impressive 140 $\times$ speedup in the processing speed, compared with a CPU implementation.

Zhao and Taubin [Zhao and Taubin, 2011] introduce a real-time dense stereo matching algorithm based on a progressive multi-resolution pipeline which includes background modeling and dense matching with adaptive windows. They also give detailed GPU implementation on GPU buffer and thread block optimization.

Although many works attempt to improve the efficiency of stereo matching algorithms using either multi-core CPU or GPU, and some of them compare the overall performance on several different platforms, there has been no systematic study of the acceleration impact of the individual key sub-parts of the stereo matching algorithm in relation to the type of hardware used.

4.2 GPU Accelerated Parallel Pyramid Matcher

In this section, we focus on describing our strategies for accelerating the stereo matching algorithm using a GPU with the CUDA platform.

4.2.1 GPU Versus CPU

The CPU has often been called the brains of the PC. But increasingly, that brain is being enhanced by another part of the PC - the GPU. The architectures of CPU and GPU are presented in Figure 4.1. These different architectures indicate the difference of CPU and GPU.

GPUs are designed for parallel computations with lots of arithmetic operations, and therefore it contains a lot more processor cores than a CPU. They process data arrays instead of flow control of several sequential computing threads. Traditional CPUs can execute 1 or 2 threads per core, while GPUs can maintain hundreds (or even thousands) of threads per multiprocessor. CPUs need hundreds of cycles to switch from different threads, while GPUs can switch several threads every cycle. Therefore, GPUs are specialized for fast execution of

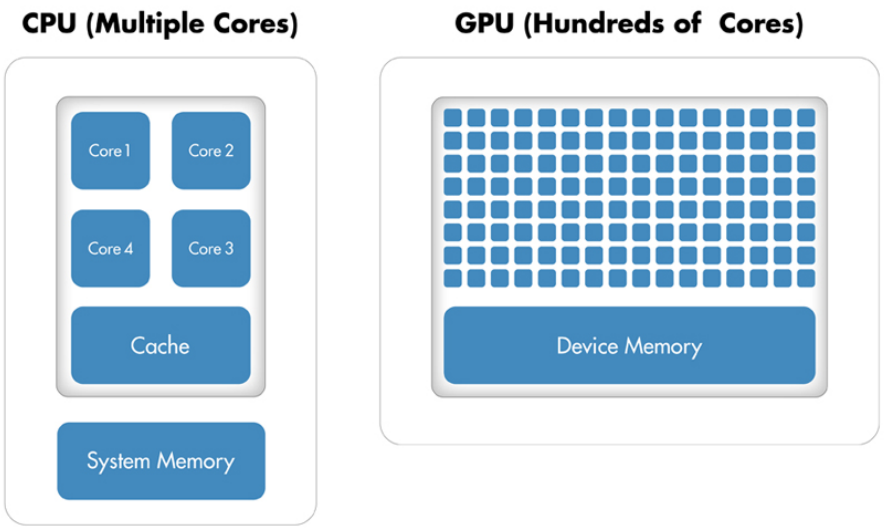


Figure 4.1: Different Architectures of CPU and GPU

many parallel instruction threads, while CPU cores are designed to execute a single thread of sequential instructions with maximum speed.

Nowadays, most computers have multi-core CPUs, which normally have several cores with some cache and control units. However, the number of processing units (CPU cores) is still much fewer than the number of GPU cores. Each CPU core operates independently, which makes it possible to perform different instructions with different data, while all GPU cores executes same instruction simultaneously, but works on different data. However, when comparing performance of Single Instruction Multiple Data (SIMD) tasks (e.g. image data processing) on both platform, in general, multi-core CPUs processing can only provide little parallelism due to the small number of cores.

Thus, understanding the characters of CPUs and GPUs could give us a clearer idea of how to utilize them in combination, to get the best system performance, price, and power.

4.2.2 GPU Parallelization

Besides the acceleration of stereo matching algorithm on multi-core CPUs described in the Chapter 3, in this section, we describe the process of implementing Parallel Pyramid Matcher (PPM) algorithm on a GPU to exploit its facilities for parallel computation. The GPU parallelization consists of distributing PPM algorithm over three types of memories on a GPU with CUDA architecture (shared memory, global memory and texture memory) (Figure 4.2) and pinned (or "page-locked") memory on CPU. Each type of memory and its subsequent parallelization is discussed in the following sections. The integrated source code[1] is public

[1] http://www.dcs.gla.ac.uk/~tian/datasets/gpuMatcher.zip

available for researchers to use.

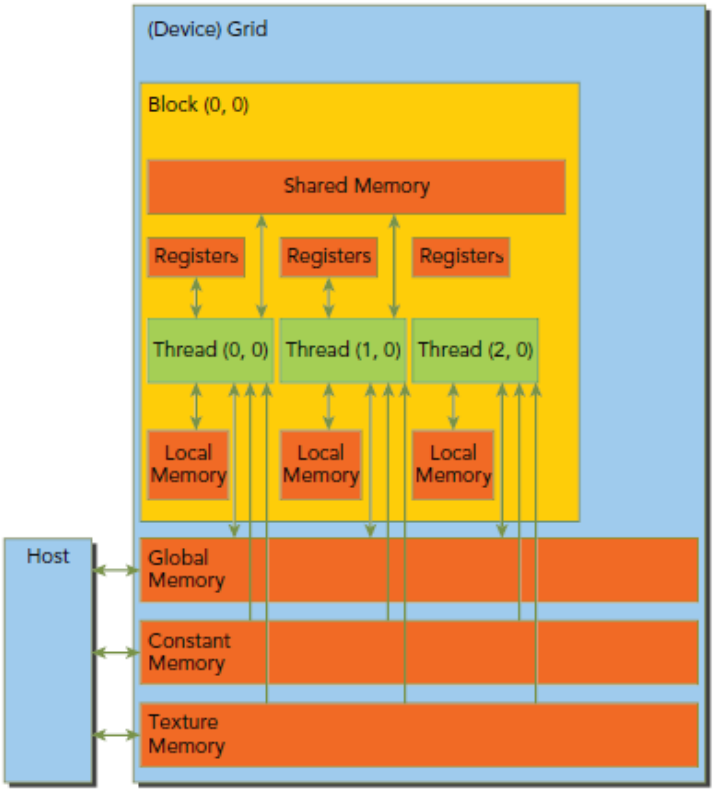


Figure 4.2: GPU Architecture

Shared Memory

GPU shared memory is extremely fast and is shared between all streaming processors in a multiprocessor. A streaming processor is a fully pipelined, single-issue, in-order microprocessor, and several streaming processors make up a streaming multiprocessor. To use shared memory, the data is first copied from host memory to global memory on the GPU device, and then indexed by block and thread ID. Threads can access data in shared memory loaded from global memory by other threads within the same thread block. Unlike cache that everything is done automatically, with shared memory, we get full control as to what gets stored where and what threads will do what. A single streaming processor only executes one thread program at a time, but all streaming processors in all multiprocessor work simultaneously, which results in a time effective execution. After execution, data in shared memory is copied back to global memory, in order to communicate with host.

To accelerate the pyramidal stereo matching algorithm described in Section 3.2, for example, convolution procedure, the component being used several times in the stereo matching algorithm, is implemented using shared memory. Convolution is used in the sub-components of pyramid creation, pixel correlation and inter-scale disparity refinement in PPM. This is an

important part for parallelisation optimization and it demonstrates significant performance improvement in the CPU acceleration.

Similar to our CPU acceleration implemented in Vector Pascal as described in Section 3.3.2, the convolution process is divided into two passes. One pass is performed for each of the two dimensions in a separable image filter. Listing 4.1 shows the row filter part of the convolution algorithm in CUDA to run on an Nvidia GPU. The column filter pass operates much like the row filter pass.

Figure 4.3 demonstrates the layout of the thread block grid for the row filtering pass. Since the convolution is defined over the neighbours of the pixel, an apron for each block is needed. Each block is allocated to a multiprocessor. All threads within one block execute on the allocated multiprocessor.

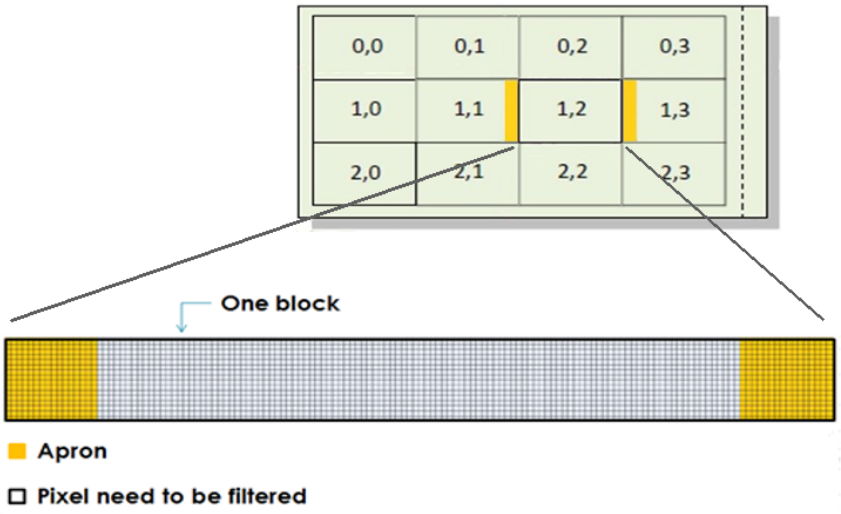


Figure 4.3: Layout of the thread block grid for the row filtering pass

Now we describe the detailed implementation of convolution through the CUDA framework, as shown in Listing 4.1.

Listing 4.1: CUDA code for the row filter kernel of convolution

```
__global__ void convolutionRowsKernel(
    float *d_Dst,
    float *d_Src,
    int imageW,
    int imageH,
    int pitch
)
{
  __shared__ float s_Data[ROWS_BLOCKDIM_Y][(ROWS_RESULT_STEPS +
                                            2 * ROWS_HALO_STEPS) *
```

```
                                                ROWS_BLOCKDIM_X];

  const int baseX = (blockIdx.x * ROWS_RESULT_STEPS -
                     ROWS_HALO_STEPS) * ROWS_BLOCKDIM_X +
                     threadIdx.x;
  const int baseY = blockIdx.y * ROWS_BLOCKDIM_Y + threadIdx.y;
  d_Src += baseY * pitch + baseX;
  d_Dst += baseY * pitch + baseX;
```

s_Data is located on shared memory. Halo is the apron in the code. baseX and baseY give the offset to the left halo edge.

```
  int Idx,remainX,tempIdx, tempRm;
  remainX = imageW % (ROWS_RESULT_STEPS * ROWS_BLOCKDIM_X);
  Idx = imageW / (ROWS_RESULT_STEPS * ROWS_BLOCKDIM_X);
  tempIdx = remainX / ROWS_BLOCKDIM_X;
  tempRm = remainX % ROWS_BLOCKDIM_X;

#pragma unroll
  for (int i = ROWS_HALO_STEPS;
           i < ROWS_HALO_STEPS + ROWS_RESULT_STEPS; i++)
  {
      s_Data[threadIdx.y][threadIdx.x + i * ROWS_BLOCKDIM_X] =
                                          d_Src[i * ROWS_BLOCKDIM_X];
  }
```

This equation helps to load main data for each block. Since the innermost processing loop of row filter performs very few computations per iteration, the loop overhead is very big, so in order to achieve higher efficiency, each thread must process more than one pixel, thus we unroll the loop.

```
#pragma unroll
  for (int i = 0; i < ROWS_HALO_STEPS; i++)
  {
    s_Data[threadIdx.y][threadIdx.x + i * ROWS_BLOCKDIM_X] =
                                (baseX >= -i * ROWS_BLOCKDIM_X) ?
                                     d_Src[i * ROWS_BLOCKDIM_X] : 0;
  }

#pragma unroll
  for (int i = ROWS_HALO_STEPS + ROWS_RESULT_STEPS;
           i < ROWS_HALO_STEPS + ROWS_RESULT_STEPS +
                                   ROWS_HALO_STEPS; i++)
  {
    s_Data[threadIdx.y][threadIdx.x + i * ROWS_BLOCKDIM_X] =
                         (imageW - baseX > i * ROWS_BLOCKDIM_X) ?
                                     d_Src[i * ROWS_BLOCKDIM_X] : 0;
```

```
  }
    __syncthreads();
```

This procedure is to load data for left and right halo, so that each pixel of the main data has neighbours. The overlap for blocks make sure that the convolution results are correct.

```
  if(blockIdx.x >= Idx && baseY < imageH)
  {
#pragma unroll
        for (int i = ROWS_BLOCKDIM_X * ROWS_HALO_STEPS;
                 i < ROWS_BLOCKDIM_X * (ROWS_HALO_STEPS + tempIdx) +
                                                          tempRm; i++)
        {
          float sum = 0;
#pragma unroll
          for (int j = -KERNEL_RADIUS; j <= KERNEL_RADIUS; j++)
          {
                sum += c_Kernel[KERNEL_RADIUS - j] *
                       s_Data[threadIdx.y][threadIdx.x + i + j];
          }
          d_Dst[i] = sum;
        }
  }
```

The last few pixels in each row may not full fill the right most blocks. This part of the code deals with this scenario. Convolution is only applied on image pixels, not padding pixels, and the computing result would be copied back to GPU global memory from shared memory.

```
  else if (baseY < imageH){
#pragma unroll
        for (int i = ROWS_HALO_STEPS; i < ROWS_HALO_STEPS +
                                          ROWS_RESULT_STEPS; i++)
        {
          float sum = 0;
#pragma unroll
          for (int j = -KERNEL_RADIUS; j <= KERNEL_RADIUS; j++)
          {
                sum += c_Kernel[KERNEL_RADIUS - j] *
                       s_Data[threadIdx.y][threadIdx.x +
                                        i * ROWS_BLOCKDIM_X + j];
          }
          d_Dst[i * ROWS_BLOCKDIM_X] = sum;
        }
  }
}
```

In this procedure, convolution is applied on each block for most pixels of the image, and then

the result would be copied back to the device global memory.

Texture Memory

GPU texture memory is available for reading by all multiprocessors. Data is fetched by texture units in a GPU and textures are accessed through a dedicated read-only cache which is optimized for spatial locality, so the data can be interpolated linearly without extra overheads (see Figure 4.4).

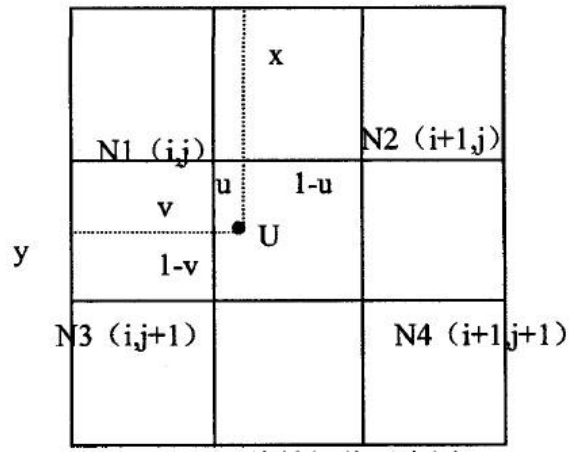


Figure 4.4: Floating Point Interpolation

Suppose the four points N1, N2, N3 and N4 are known. In order to determine the interpolated U, normally the equation below is executed, which is a cost expensive calculation.

$$\begin{aligned} F(x,y) = uvF(i,j) + (1-u)vF(i+1,j) \\ + u(1-v)F(i,j+1) + (1-u)(1-v)F(i+1,j+1) \end{aligned} \tag{4.1}$$

On a GPU, this interpolation is implemented in hardware while the execution time cost for texture memory to perform linear floating point interpolation is negligible. Floating point interpolation is an important element in the pyramidal stereo matching algorithm. Polynomial maximization and rescaling image size both can benefit from the fast access provided by the texture memory.

Listing 4.2 shows an interpolation algorithm in CUDA to run on an Nvidia GPU. The kernel function is in principal invoked in parallel on all pixel positions. On the GPU it is not necessary to explicitly code for the interpolation. Instead, texture memory allows implicit linear interpolation of a two dimensional array of data. The interpolation is performed when the array is indexed using the `tex2D` function. Note that before executing `interpolationKernel`, data has been stored in texture memory named `texSrc` already.

Listing 4.2: CUDA code for the interpolation kernel. Note the use of 'texture memory' which can be indexed by real valued coordinates.

```
//Maps to a single instruction on G8x / G9x / G10x
#define IMAD(a, b, c) ( __mul24((a), (b)) + (c) )

__global__ void interpolationKernel(
    float *d_Dst,
    int imageW,
    int imageH,
    float scalefactorW,
    float scalefactorH,
    int imageW2,
    int imageH2
)
{
    const int ix = IMAD(blockDim.x, blockIdx.x, threadIdx.x);
    const int iy = IMAD(blockDim.y, blockIdx.y, threadIdx.y);
    const float  x = (float)ix + 0.5f;
    const float  y = (float)iy + 0.5f;
    if (ix >= imageW2 || iy >= imageH2)
    {
        return;
    }
    d_Dst[IMAD(iy, imageW2, ix)] = tex2D(texSrc, x / scalefactorW,
                                         y / scalefactorH);
}
```

Note that for floating-point coordinate, 0.5f offset needs to be added to the texture coordinate, in order to get the identity of exact center. An example of this is shown in Figure 4.5.

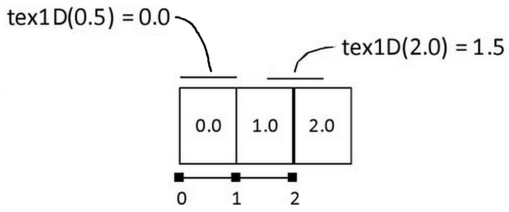


Figure 4.5: Texturing pixels in CUDA

Global Memory

GPU Global memory is the largest volume of memory available to all multiprocessors in a GPU, but the latency (about 400-800 cycles) is higher than shared memory. Theoretical global memory bandwidth can be calculated using hardware specification. For example, the

peak bandwidth of NVIDIA GeForce GTX 770 with a memory speed of 7.0 Gbps and a 256-bit wide memory interface is

$$Bandwidth = 7.0Gbps \times 256/8 = 224GB/s \tag{4.2}$$

where memory data width is divided by 8, in order to convert bits to bytes.

However, this is not the speed for data transfer between CPU and GPU. The speed is limited by PCI-e (Peripheral Component Interconnect Express) throughput, which varies depending on the version of PCI-e. For example, for GPUs, the bandwidth of PCI-e 3.0 is 32GB/s, while PCI-e 2.0 is 16GB/s.

As we can see, PCI-e throughput is much less than the bandwidth of global memory, so in practice, global memory is mainly used to keep intermediate results for later use either by shared memory or by texture memory, in order to reduce the time spent on data exchange between CPU and GPU. Every step of the pyramidal stereo matching algorithm requires support from global memory.

Pinned Memory

CPU data is allocated on pageable host memory by default (Figure 4.6), where GPU cannot access data directly. When the data transfers from pageable host memory to device memory, the CUDA driver must first allocate a temporary pinned buffer, copy the host data to the pinned buffer, and then transfer the data from the pinned buffer to device memory [Harris, 2012]. This consumes precious host time, but the cost of the transfer between pageable and pinned host arrays can be avoided by directly allocating host arrays in pinned memory in CUDA C/C++.

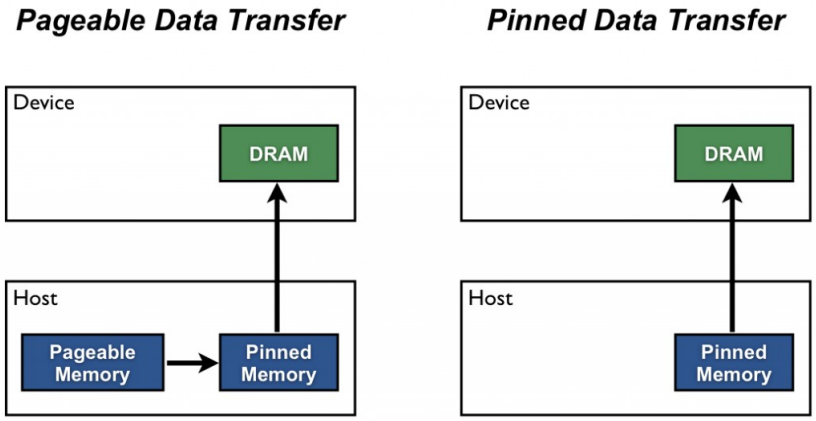


Figure 4.6: Pageable and pinned memory transfer

4.3 Experiments

In this section, the performance of the GPU version of Parallel Pyramid Matcher is reported. We first describe the experimental setup (Section 4.3.1). Then we discuss our findings on accelerating the matching algorithm on the GPU (Section 4.3.2). We finally analyze the acceleration impact on GPU (Section 4.3.3).

4.3.1 Experimental Setup

The experiment has been conducted on the Nvidia GeForce GTX 770 GPU. To compare with multi-core CPU systems, a similar specification Table of GPU is presented in Table 4.1. CUDA C is the language used for coding acceleration on the GPU while our test image data is exactly the same as used in multi-core CPU experiments described in Section 3.4.1.

Table 4.1: Specification of GPU used in this work (SMX for Streaming Multiprocessor)

Parameter	Nvidia GeForce GTX 770
Chips	1
Cores/Threads	1536
Clock Speed	1046 MHz
Memory Capacity	4 GB
Memory Technology	GDDR5
Memory Clock	1753 MHz
Memory Speed	7.0 Gbps
Memory Data Width	256 bits
Peak Memory Bandwidth	224.3 GB/s
Data Caches	64 KB L1 per SMX, 512 KB L2

4.3.2 General Results

The PPM stereo matching algorithm and their key sub-parts are implemented for, and run on, the GPU. To compare GPU acceleration performance with the CPU results (specification can be found on Table 3.2), Table 4.2 presents in detail, single-thread AMD CPU performance (using both AVX and x87 instructions), best AMD CPU performance (using 44 threads with AVX instructions) and GPU performance for each of the six key sub-parts. In the table, "Total" shows the time needed to deal with the whole matching algorithm, which is also the sum of the execution time of all six sub-parts. Besides the execution time for each sub-part, the table provides the corresponding speed-up of the AMD single-thread SIMD,

best AMD CPU and GPU performance over the single-thread AMD CPU performance with X87 instructions. It should be noted that single threaded AVX timing benefits from SIMD parallelisation - with a vector width of 8. The acceleration given by AVX code turns out to be at most 3.5, since not all operations are fully vectorizable.

Table 4.2: Single-thread AMD CPU with X87 and AVX instructions, 44-thread AMD CPU and GPU Performance

	CPU					GPU	
	Single-thread			44-thread (best)			
	X87	AVX					
	Time(s)	Time(s)	Speedup	Time(s)	Speedup	Time(s)	Speedup
Pyramids	19.94	8.61	2.32	1.68	11.87	0.43	45.88
Correlation	486.56	167.69	2.90	42.34	11.49	2.90	167.89
Polynomial	192.19	178.51	1.08	8.18	23.50	0.26	731.12
Refinement	174.93	49.93	3.50	16.80	10.41	1.04	168.65
Rescale	5.57	3.91	1.42	0.39	14.28	0.19	29.89
Other tasks	7.95	4.46	1.78	1.16	6.85	0.20	39.86
Total	887.14	413.11	2.15	70.55	12.57	5.02	176.77

Overall, CPU gives about $12\times$ and GPU gives $176\times$ acceleration. Among the sub-tasks, polynomial maximization gives the best parallel performance with 23 times and 731 times speed-up on CPU and GPU respectively. This process is full of array calculations, and every single step can be parallelized. This type of process suits well for parallelization with both architectures. Simple linear interpolation, which is an important and basic step in the matching algorithm, also has this characteristic.

On the other hand, the most time consuming part of the algorithm is pixel correlation. The best GPU and CPU performance reaches respectively 167 time and 11 times speed-up against single thread (X87 instructions). This sub-part of the algorithm contains many iterations of image convolution, another basic image algorithm, which we aim to understand in-depth in terms of GPU acceleration next.

From the performance table, we notice that the speed-up for polynomial maximization on a GPU is much higher than for other components. This is due to the polynomial maximization being the only part which runs entirely on the GPU without any data transfer to CPU.

The experiment is designed to let GPU deal with all the data processing procedures and let CPU undertake memory allocation, reading/writing data from/to files, etc. However, due to the size limitation of the graphic card memory, intermediate variables may need to transfer to CPU memory for a while. This consumes non-negligible time. So if we can have a GPU with larger memory or deal with smaller images, it is possible to change the framework a little bit to further accelerated the whole process.

Next, we will demonstrate how much data transfer between the CPU and GPU can influence the overall performance.

4.3.3 Acceleration Analysis

Similar to CPU performance, polynomial maximization and pixel correlation exhibit different acceleration rates through GPU. To study the underlying reasons for the differences in acceleration, linear interpolation and image convolution are examined in more detail.

Interpolation and image convolution performance on GPU is presented in Figure 4.7. On

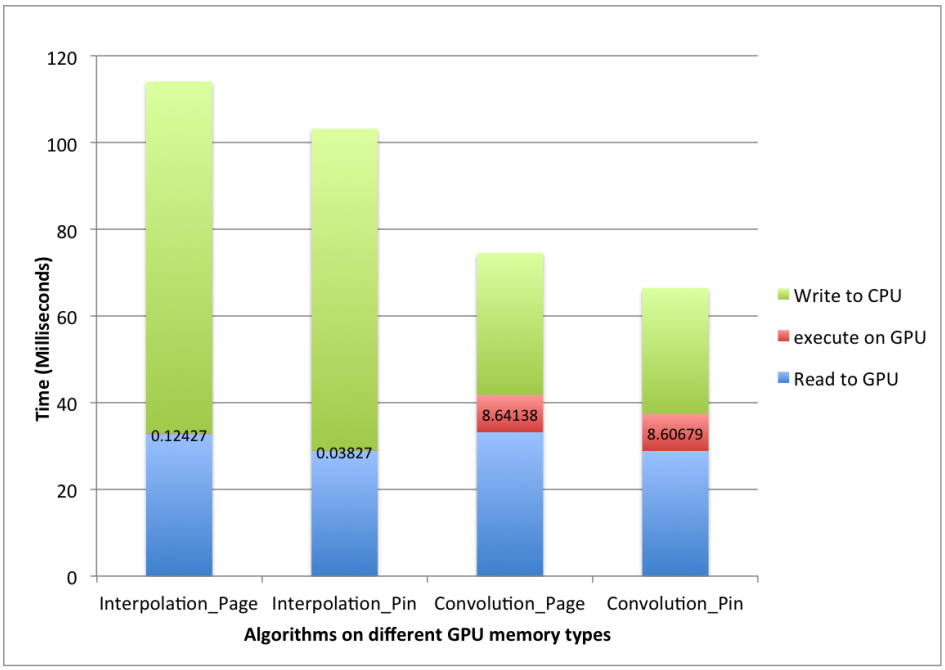


Figure 4.7: Interpolation and convolution performance on GPU

the GPU, interpolation gives about 39 times speed-up compared to a single-core CPU using AVX instructions. It is not as big an improvement as on polynomial maximization, but after breaking down the interpolation process on GPU, it can be seen that almost all the time is consumed on data transfer between the CPU and GPU. Although, using pinned memory, rather than paged memory, to reduce the transfer time it is still the most time consuming part. Comparatively speaking, the execution time on the GPU is minimal. This is due to one of the main advantages of the GPU. The GPU has texture memory facilitating the interpolation. As the size of image increases, there is no significant execution time increment shown on the GPU, which demonstrates that doing floating point interpolation on texture memory is almost cost-free, as long as the image data can fit into texture memory.

With convolution, the GPU gives about a 7 times speed-up over a single-core CPU using AVX. Compared to interpolation, this is only a marginal improvement. Breaking down the

convolution process, the most time consuming part is again the transfer time. However, for convolution, execution time on the GPU is short but nonnegligible. The main function of the convolution, executes on the shared memory, which is extremely fast. But similar to the CPU implementation, the algorithm generates intermediate variables in global memory. Therefore, data transfer within the GPU memories is inevitable. This accounts for most of the execution time on the GPU.

Both GPU versions of interpolation and convolution give significant acceleration improvements compared to CPU performance. Although as an isolated algorithm, convolution's performance is bounded by CPU to GPU transfers, when operating as a component of the matcher it is invoked repeatedly using data that has already been transferred to the GPU. This would explain why GPU gives 176 times speed-up on the complete matching algorithm.

This scenario suggests, when implementing algorithms on GPU, one should let GPU do parallel computing as much as possible, and try the best to keep intermediate variables on GPU only, because the data transfer within GPU is much faster than transfer to CPU.

4.3.4 Discussion of Accuracy and Validation

Similar to multi-core CPU, we conduct validation analysis for GPU as well. Using the same methodology described in Section 3.4.5, we select several image pairs from Glasgow garment dataset [Aragon-Camarasa et al., 2013] as our testbed and utilize the percentage of bad pixels as our metric to evaluate the output (disparity maps). By comparing the outputs from both GPU and original sequential algorithm, we found that the disparity maps can basically match each other for the cloth and table area, while for the periphery of the image, the performance is not that promising. On average, the difference of bad pixel rate for the disparity maps of the GPU CUDA version vs. the original sequential version is approximately 8%, which shows better performance than the Vector Pascal CPU version (17%). This may be because compared to Vector Pascal, the numerical settings of CUDA is closer to the original Java setting.

4.4 Conclusions

The objective of this chapter is to study utilizing a GPU to parallelize stereo matching. We found that overall the Nvidia GeForce GTX 770 GPU (1536 cores) that programmed by CUDA C language, gives $176\times$ acceleration (Table 4.2), which is significantly better than the AMD 6366HE CPU (64 cores) programmed by Vector Pascal language ($12\times$ speed-ups). Processes of array calculations (such as polynomial maximization as one sub-task) can be best paralleled with a 731 times speed-up on GPU (Table 4.2). Furthermore, we found that

when implementing algorithms on GPU, one should let GPU do parallel computing as much as possible, and try the best to keep intermediate variables on GPU only, because the data transfer within GPU is much faster than the data transfer to CPU (Table 4.1).

Part III

Accurate Stereo Matching for Cloth Manipulation

Chapter 5

Cloth Manipulation Stereo Matching Evaluation

In the previous two chapters (Part II), we have attempted to improve the *efficiency* of stereo matching algorithm using multi-core CPU and GPU architectures, now in Part III of the thesis, we move our attention towards the *effectiveness* (accuracy) aspect of stereo matching algorithms in the context of cloth manipulations.

Before proposing new stereo matching algorithm, we need to be able to evaluate the accuracy of the stereo matching performance. Although several datasets have been developed [Scharstein and Szeliski, 2002] [Geiger et al., 2012] to measure stereo matching effectiveness in general, within the context of our interests (cloth images), to our knowledge, no public testbeds are available with disparity or depth map ground-truths for the evaluation purposes.

The main aim of this chapter is to understand how to effectively evaluate the accuracy of stereo matching algorithm performance in the robotic cloth manipulation context. In order to thoroughly evaluate in this context, we need to construct a set of testbeds, consisting of images with different cloth materials and different pose configurations for the thorough evaluation. In addition, in most circumstances, as the robot normally aims to deal with clothes in the chaotic status (e.g. for unfolding or grasping), we also make sure there are various wrinkle types on the clothes. We mainly aim to answer:

> **RQ 6**: How to construct the proper datasets to evaluate stereo matching algorithms in the context of cloth manipulations? Can we also provide disparity ground-truths for both unrectified and rectified images?
>
> **RQ 7**: What evaluation metrics should be used to evaluate the accuracy performance of cloth related stereo matching algorithms given the testbeds we developed?

The remainder of this chapter is organized as follows. We first review the popular stereo benchmark in Section 5.1. We then describe five datasets that designed to evaluate stereo matching algorithms for unrectified images in the context of cloth manipulations in Section 5.2. Finally, metrics for measure the quality of output disparity map and depth map are studied (Section 5.3), after which the chapter is concluded (Section 5.4).

5.1 Related Work

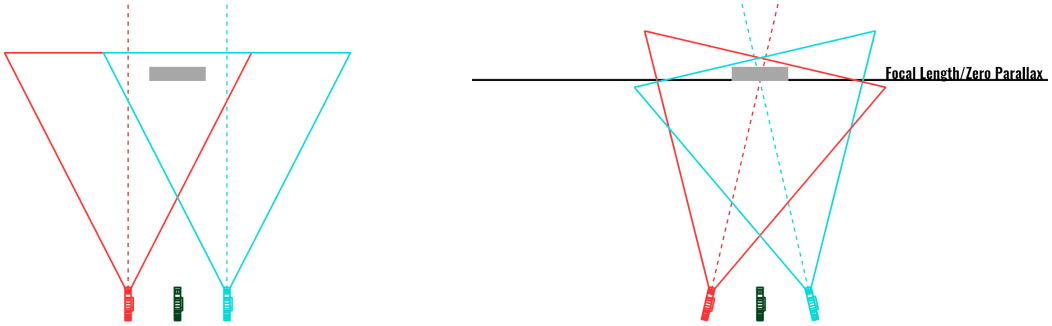
Several research efforts [Geiger et al., 2012, Ladickỳ et al., 2012, Pfeiffer et al., 2013, Saxena et al., 2007, Scharstein and Szeliski, 2002, Tombari et al., 2010] have been devoted in the past decades to develop datasets and methods for evaluating stereo matching. For example, KITTI stereo / flow benchmark [Geiger et al., 2012] consists of 194 training and 195 test image pairs developed in the context of autonomous driving for evaluating stereo matching.

Middlebury stereo benchmark [Scharstein and Szeliski, 2002] is the most popular dataset to evaluate stereo matching algorithm's performance, as mentioned in Section 2.1.4. There are various versions of Middlebury Stereo Evaluation datasets while the most popular version (Version 2, which is no longer active) generally contains low resolutions images of less than 640×480 pixels. Version 3 maintains images of higher resolutions, up to around 3000×2000 pixels. Low resolution could not satisfy today's resolution requirement for stereo matching as it may cause the loss of image details. From both accuracy and efficiency perspective, the algorithms that perform well for small images may not perform well for high resolution images.

However, we do not use Middlebury datasets or any other existing datasets for our stereo matching evaluation. This is due to the fact that those datasets are not suitable to evaluate stereo matching algorithm performance in the context of cloth manipulations. Although Middlebury datasets contain a few cloth related images, they do not cover a wide range of materials and cloth pose configurations for thorough evaluation. And for the latest version (Version 3), there are no scene designed for cloth, only several image pairs are partly containing cloth, which are also not sufficient to evaluate the performance on cloth.

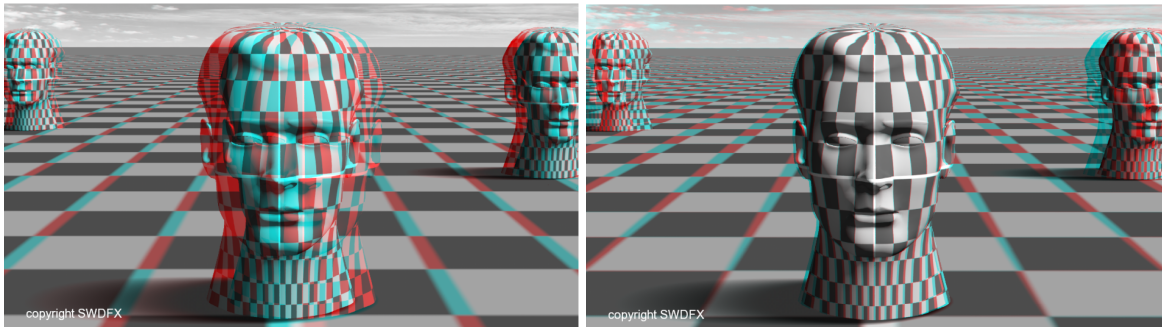
Furthermore, within these datasets, all the stereo images are rectified images, which makes it not suitable for evaluating matching algorithms for unrectified images. There are two main camera/stereo configurations, which are parallel and converged (as shown in Figure 5.1). An example to demonstrate the effect of different configurations are shown in Figure 5.2. All Middlebury datasets are using rectified image pairs with parallel configuration, which means the strategy of matching algorithms designed for Middlebury datasets only need to consider horizontal disparities, but with larger range of offset compared to converged setting. While in this thesis, we try to develop algorithm for converged configuration, thereby we have

different strategy. We can see that for converged camera setting, especially for the central area of images (normally the area of interesting) only have small or no disparities, so we only need to consider a small range of the offsets around each pixel (i.e. left, right, up and down). For this purpose, we choose to develop several datasets to thoroughly evaluate stereo matching algorithms designed for unrectified images using converged configuration.



(a) Parallel (b) Converged

Figure 5.1: Two main camera configurations



(a) Stereo render with parallel cameras (b) Stereo render with converged cameras

Figure 5.2: Example of effect of two different configurations [Wright, 2011]

5.2 Stereo Image Datasets

To facilitate the evaluation of stereo matching algorithm performance, we describe our methodology to create five datasets : real scene with box on table dataset, simulated cloth dataset, garments dataset, simulated cloth wrinkle dataset and enriched garments dataset.

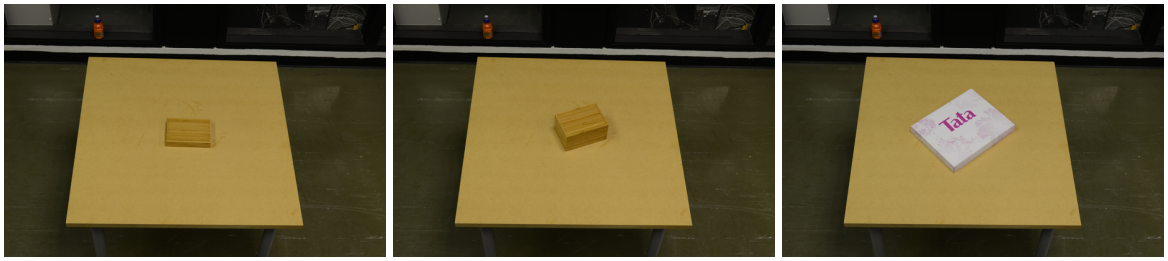
5.2.1 Real scene with box on table dataset

This dataset[1] is formed by 24 images, and each scene shows a table with a rectangle box placed on it. Three boxes are used for this dataset, and each box is placed on the table with

[1] http://www.dcs.gla.ac.uk/~tian/datasets/RealSceneWithBoxOnTableDataset.zip

several different positions (see Figure 5.3 for an example). In order to show more details of the table and the box, the size of stereo images we captured is 4928×3264. The three boxes have different length, width and height, which have been carefully measured by vernier caliper before generating the dataset. A pair of cameras, focusing on the same point of table, are used to simultaneously capture stereo images.

Although this dataset does not directly relate to cloth images, this dataset consists of high resolution unrectified images, which make it possible to evaluate stereo matching algorithms that can directly apply to unrectified images. With known camera parameters, it is also possible to rectify these images in order to apply matching algorithms that are only available for rectified images. It is also possible to rectify the images (e.g. by utilizing the rectification algorithm provided by openCV[2]) for evaluating those stereo matching algorithms applied to rectified images.



(a) First Image (b) Second Image (c) Third Image

Figure 5.3: Examples of real scene with box on table dataset

The ground truth for disparity map is difficult to generate, but since the depth of the box is known, we could use depth map to evaluate the algorithm performance. No matter the algorithms are applied on rectified or unrectified images, the depth of the box remains unchanged. So for each algorithm, we first try to fit the table surface and top box surface using least squares fitting method. And then we use both surface equations to calculate the distance between the box and the table using the central point of the box's top surface. By comparing this estimated depth of box with the ground truth depth of the box, we could measure how accurate is the depth map generated by a given stereo matching algorithm.

To make the evaluation more comprehensive, the depth map for the table and the box are generated for comparison. Because every image pair has exactly the same table placed at the identical location, it is easier to use the whole dataset to estimate the equation of the table plane given an assumed coordinate system. Several algorithms are applied on these image pairs to estimate the table surface, but when fitting the plane, only the root mean square error (RMS) less than a pre-defined threshold would be counted. The average of the counted estimation is recognised as the ground truth of the table. Assuming the top box surface is

[2]http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

parallel to the table surface, the top box surface could be calculated by moving the table plane with the value of box height according to perpendicular direction.

After knowing table's estimated plane (ground-truth), we aim to also estimate the box's top surface plane (ground-truth) using geometry. Let us assume the box's top surface is parallel to the table surface, as shown in Figure 5.4. Suppose point A is the left camera, located on x-y plane, facing z axis (the red line is x axis, the green line is y axis and the blue line is z axis). Plane HIJK is the table plane, and plane LMNO is the box's top surface plane. Plane LMNO and plane x-y intersect on line PE, and Plane HIJK and plane x-y intersect on line BG. We draw line AP that perpendicular to line PE. Because plane HIJK $\parallel$ plane LMNO, PE $\parallel$ BG and BC $\parallel$ DE.

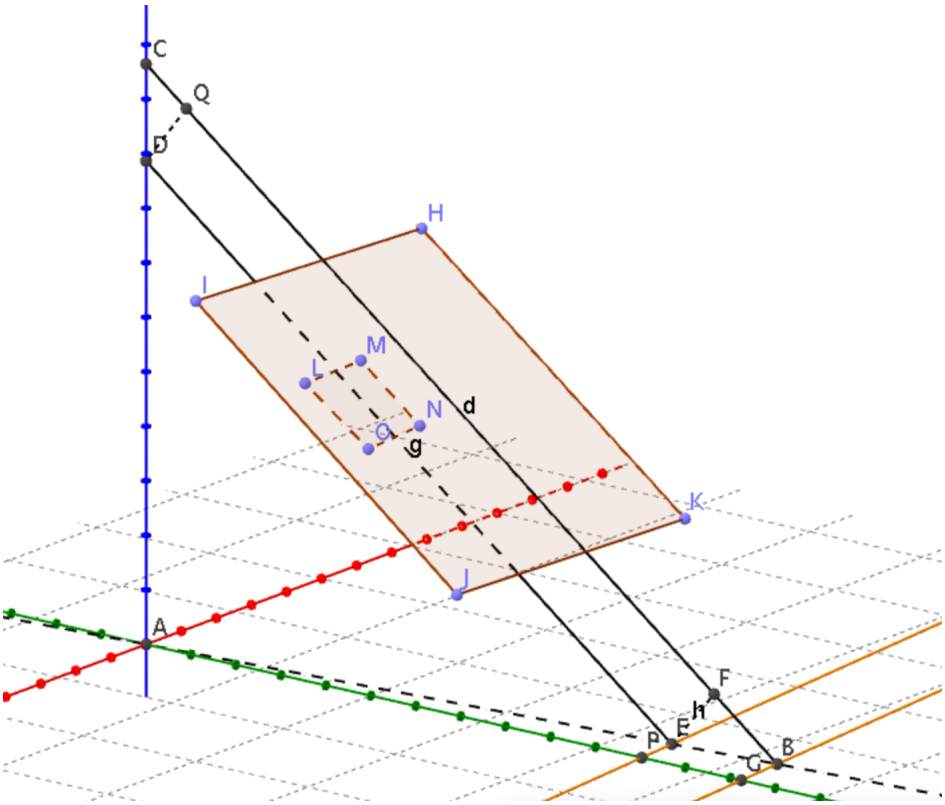


Figure 5.4: Model of camera-table-box system

Suppose the equation of table plane HIJK is

$$z = a + bx + cy \tag{5.1}$$

then

$$AC = a \tag{5.2}$$

x-y plane 's equation is

$$z = 0 \tag{5.3}$$

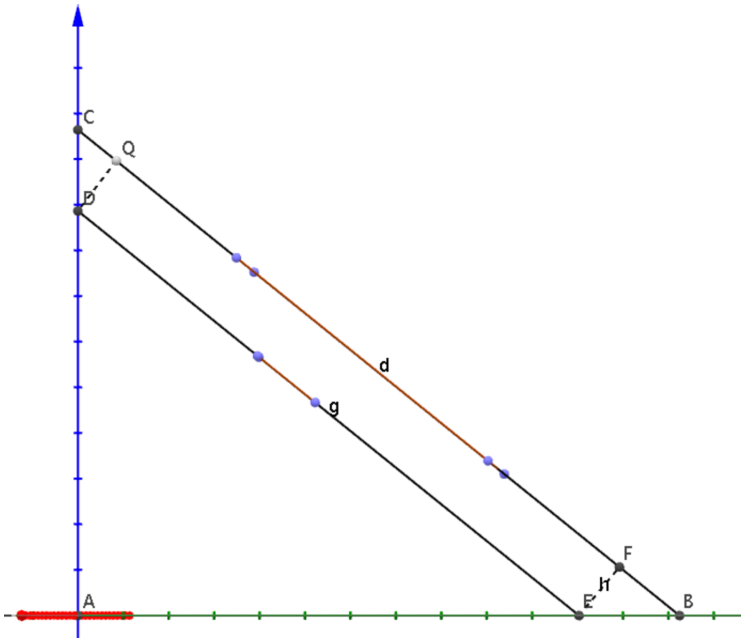


Figure 5.5: Model of camera-table-box system (side view)

so dihedral angle[3]

$$\begin{aligned} \angle ABC &= arccos(\frac{b*0+c*0+1*1}{\sqrt{b^2+c^2+1^2}*\sqrt{0^2+0^2+1^2}}) \\ &= arccos(\frac{1}{\sqrt{b^2+c^2+1}}) \end{aligned} \tag{5.4}$$

Then we draw line EF, and line DQ to make both of them perpendicular to plane HIJK. From the side view (Figure 5.5) we can see that

$$\triangle ABC \sim \triangle FBE \sim \triangle QDC \tag{5.5}$$

so

$$\angle QDC = \angle ABC \tag{5.6}$$

because

$$QD = EF = h \tag{5.7}$$

and h is the height of box, which is known, so

$$cos\angle QDC = cos\angle ABC = \frac{h}{CD} = \frac{1}{\sqrt{b^2+c^2+1}} \tag{5.8}$$

then

$$CD = h * \sqrt{b^2+c^2+1} \tag{5.9}$$

[3]https://en.wikipedia.org/wiki/Plane_(geometry)

because plane HIJK || plane LMNO, then the gradients of two plane are the same, so equation of the top box surface is

$$z = d + bx + cy \tag{5.10}$$

because

$$d = AD = AC - CD = a - h * \sqrt{b^2 + c^2 + 1} \tag{5.11}$$

so the final equation for the top box surface is

$$z = a - h\sqrt{b^2 + c^2 + 1} + bx + cy \tag{5.12}$$

Therefore, the ground truth depth map for each box's top surface can be generated.

5.2.2 Simulated cloth dataset

Because our stereo matching task is based on clothes, an image dataset[4] for cloth textures is generated in order to to test whether the stereo matching algorithms can handle all kinds of cloth textures successfully. This dataset contains 78 images. A collection of different types of cloth images (e.g. cotton, wool, jeans, leather, carpet, etc.) (Figure 5.6) is used for this dataset. Images are cropped into three different sizes, including small size (512×512 pixels), medium size (1024×1024 pixels), and large size (2448×2050 pixels), and each of these single image is treated as left image (reference image).

Unlike left images, which are real cloth textures images, right images are the simulated ones (Figure 5.7). It is difficult to obtain depth or disparity ground truth, because clothes are always soft and deformable. Thus, we try to simulate cloth wrinkle waves by the sin transform. Horizontal sin transform and diagonal sin transform are applied on the x-axis of left images to generate right images. In addition to the right image generated with no transformation on the reference image, we generate in total three kinds of right images with known disparities.

Suppose the intensity of pixel (x, y) in the left image is $I(x, y)$, then for the case of identical right image, we have intensity $I_{(a)}$

$$I_{(a)}(x, y) = I(x, y) \tag{5.13}$$

while for horizontal sin transform one, the disparity $D_{(b)}(x, y)$ can be represented as

$$D_{(b)}(x, y) = -A \times sin(\frac{\pi}{0.25 \times H} \times y) + A \tag{5.14}$$

[4]http://www.dcs.gla.ac.uk/~tian/datasets/SimulatedClothDataset.zip

(a) Cotton (b) Wool (c) Jeans

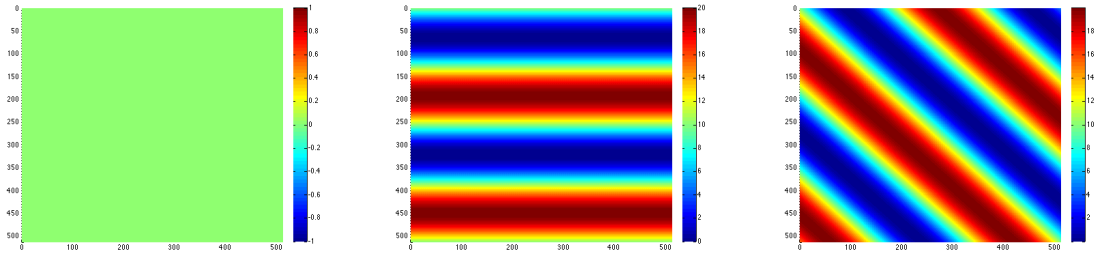
(d) Leather (e) Carpet

Figure 5.6: Example left images of five cloth patterns

(a) Left Image (Identical Right Image)

(b) Right Image with Horizontal sin Transform

(c) Right Image with Diagonal sin Transform



(d) Disparity map of (a)

(e) Disparity map of (b)

(f) Disparity map of (c)

Figure 5.7: Example of cloth's left and simulated right image

and the corresponding intensity $I_{(b)}$ is

$$I_{(b)}(x, y) = I(x + D_{(b)}, y) \tag{5.15}$$

where A is the amplitude for sin function, H is the image height in pixel. Specifically, $A = 10$, which initially makes the disparities range from -10 to 10. However, because some of the stereo matching algorithms for rectified images can not handle negative and/or vertical disparities, another A is added to change the disparities range to [0, 20]. And all disparities are only applied on horizontal direction to make sure that this dataset is suitable to evaluate and compare the performance for all kinds of stereo matching algorithms, in the context of cloth textures. The wrinkle is designed to include two sin waves, so the parameter inside the sin function is $\frac{\pi}{0.25*H}$.

Similar to horizontal sin transform, the equation for generating diagonal sin disparity map is

$$D_{(c)}(x, y) = -A \times sin(\frac{\pi}{0.25 \times \sqrt{2} \times H} \times (x - y)) + A \tag{5.16}$$

and the corresponding intensity $I_{(c)}$ is

$$I_{(c)}(x, y) = I(x + D_{(c)}, y) \tag{5.17}$$

For the same reason of horizontal sin disparity, the diagonal sin disparities are only applied on X direction as well.

5.2.3 Garments dataset

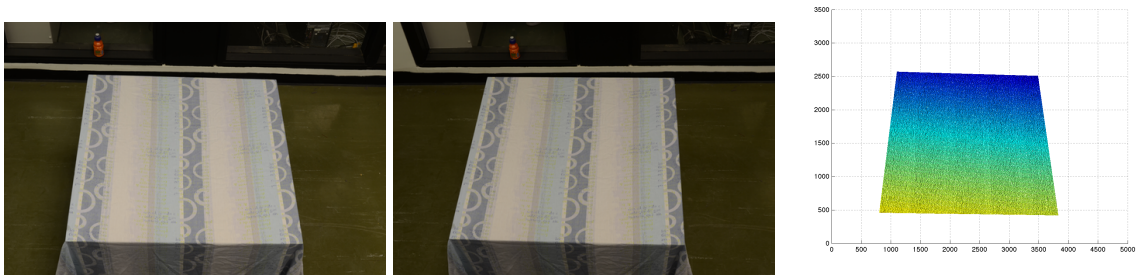
To provide insight into cloth perception and manipulation with an active binocular robotic vision system, a database[5] of 80 stereo-pair colour images [Aragon-Camarasa et al., 2013] is compiled with corresponding horizontal and vertical disparity maps and mask annotations. This database is based on 16 different off-the-shelve garments, consisting of a wide variety of textile materials with different texture, colour and reflectance characteristics in order to give a realistic sample of the real world clothing variety. Each garment has been imaged in five different pose configurations on the project's binocular robot head, including: flat on the table, folded in half, completely folded, randomly wrinkled and hanging over the robot's arm. These configurations are an approximation of the most representative pose configurations a robot may encounter while sorting and folding clothes. Note that there are no depth map ground truths for this dataset and we show several examples of the garment images in Figure 5.10.

Figure 5.8: Example of the original garments dataset

5.2.4 Simulated Cloth Wrinkle Dataset

To better understand the effect of cloth wrinkle on stereo matching algorithm in our application, we generate a dataset[6] to imitate the simplest case, which is a cloth with one wrinkle on it. A pair of real images (Resolution: 4928 $\times$ 3264) with table covered by cloth (Figure 5.9) is used to create the simulated dataset. The cloth is flat and there is no visible wrinkle on the table area. To generate the disparity map, Parallel Pyramid Matcher (PPM) is applied on the stereo images. Knowing the camera parameters, the corresponding depth map from the left viewpoint could be generated [Hartley and Sturm, 1997] (Figure 5.9(c)). Notice that the generated depth map is not the ground truth, but this serves as the flattened plane for our wrinkle simulation, i.e. adding wrinkles to this depth map.

[5]https://sites.google.com/site/ugstereodatabase/
[6]http://www.dcs.gla.ac.uk/~tian/datasets/SimulatedClothWrinkleDataset.zip



(a) Left image (b) Right image (c) Depth map of left view

Figure 5.9: The image pair obtained from the stereo robotic camera with the corresponding depth map

To imitate a "bell curve" shaped cloth wrinkle, in this work, we simply use the Gaussian function for the simulation. This simulated wrinkle is then added to the depth map (Figure 5.9) in order to generate a simulated depth map with one wrinkle on the cloth (Table 5.1). When the intrinsic and extrinsic camera parameters are known[7], the 3D points from depth map could be projected to the left and right camera view plane using a perspective transformation:

$$sx = PX_w = A[R|t]X_w \tag{5.18}$$

or

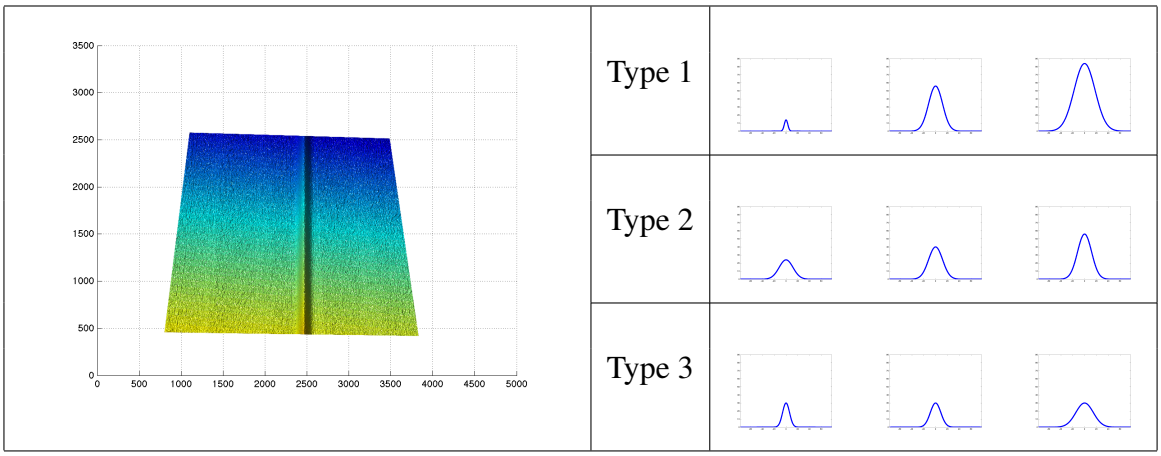
$$s\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_1 & r_2 & r_3 & t_1 \\ r_4 & r_5 & r_6 & t_2 \\ r_7 & r_8 & r_9 & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{5.19}$$

where P is a projection matrix; A is a 3×3 matrix of camera intrinsic parameters; Extrinsic parameters are combined by 3×3 rotation matrix R and 3×1 translation matrix t; s is a skew parameter; (X, Y, Z) are the coordinates of the 3D point in the world coordinate space while (x, y) are the coordinates of the projection point in pixels. Knowing the projection positions on the left and right view plane, the position difference is treated as the disparity map. By warping the right image with disparity map, the left image could be generated. The simulated depth map used to create the images can later be used as the ground truth depth map.

To better understand whether the wrinkle characteristics have an effect on the stereo matching performance, image pairs with different wrinkle width and height, and also the corresponding ground truth depth maps are created. Normally the height of wrinkle would be smaller than the width of wrinkle, because clothes are soft and deformable. If the wrinkle height is larger than the width, the wrinkle ridge may not be stable and may fall down to

[7]Camera calibration and 3D reconstruction: http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html.

Table 5.1: The depth map of simulated cloth wrinkles using Gaussian function, simulating various cross section shapes (height and width)



reduce the height. According to this, three groups of wrinkle are designed. Examples of the cloth wrinkle depth maps are shown in table 5.1. In total, we simulate three different types of wrinkles:

- (**Type 1**) Similar wrinkle curve: there are three wrinkles in this group, which have similar shapes. The height of wrinkles are about equal to the width: width 14 millimetre (mm) and height 14 mm; width 58 mm and height 56 mm; width 87 mm and height 84 mm.
- (**Type 2**) Same width with different heights: the width of three wrinkles are all 58 mm, but the heights are 24 mm, 40 mm and 56 mm respectively.
- (**Type 3**) Same height with different widths: the height of all three wrinkles are 30 mm, but the widths are 29 mm, 43 mm and 72 mm respectively.

5.2.5 Enriched Garments Dataset

To comprehensively study different cloth wrinkle characters, we generate an enriched garments dataset[8] following Glasgow's stereo image garments dataset [Aragon-Camarasa et al., 2013] mentioned in Section 5.2.3. We use exactly the same resolution as well as the same 16 different off-the-shelve garments and 1 additional towel to generate stereo images for our dataset. Motivated by [Ramisa Ayats et al., 2011], for each article we create four shapes, which are one single wrinkle in the middle (shape 1), twisted multiple wrinkles (shape 2), random multiple wrinkles (shape 3) and the last one is part of the article concealed by a fold (shape 4). The wrinkle size can be either large or small, and the article can be placed either

[8] http://www.dcs.gla.ac.uk/~tian/datasets/EnrichedGarmentDataset.zip

(a) shape 1, large, middle (b) shape 2, large, middle (c) shape 3, large, middle (d) shape 4, large, middle

(e) shape 1, small, middle (f) shape 2, small, middle (g) shape 3, small, middle (h) shape 4, small, middle

(i) shape 1, large, side (j) shape 2, large, side (k) shape 3, large, side (l) shape 4, large, side

(m) shape 1, small, side (n) shape 2, small, side (o) shape 3, small, side (p) shape 4, small, side

Figure 5.10: Example of enriched garments dataset (Each subfigure shows different wrinkle character)

in the middle or on one side. Figure 5.10 presents the examples of all of those different settings. Note that there are also no depth map ground truths for this dataset.

5.3 Evaluation Metrics

Normally there are two forms of output for stereo matching algorithms: disparity map and depth map. Different evaluation metrics are applied correspondingly according to different forms.

5.3.1 Evaluation for Disparity Map

Normally, for rectified image pairs, only horizontal disparity map is needed, because theoretically the position of corresponding pixels are the same in y direction. However, the stereo

rectification resulting from a standard calibration procedure may not be accurate enough (e.g. incorrect lens distortion estimation), this error would cause small y direction disparity. So recently, not only for the unrectified image pairs, but also for the rectified image pairs, researchers start to take vertical disparity map into consideration [Scharstein et al., 2014].

For disparity evaluation, horizontal and vertical disparities are normally considered separately. For example, the horizontal disparity of a feature is the difference in its horizontal position as observer from left and right cameras, and the difference is presented in terms of image pixels. Same for the vertical disparity, but only vertical position difference is taking into account. To quantitatively estimate the quality of the computed correspondences for each direction, one common technique is to compute error statistics with respect to ground truth data [Barron et al., 1994]. Normally the quality of disparity map are measured using two general approaches, percentage of bad matching pixels and root mean square (RMS) error [Scharstein and Szeliski, 2002].

1. Percentage of bad matching pixels (B)

$$B = \frac{1}{N} \sum_{(x,y)} (|d_C(x,y) - d_T(x,y)| > \delta_d) \tag{5.20}$$

   where $d_C(x,y)$ is the computed disparity map and $d_T(x,y)$ is the ground truth disparity map. N is the total number of pixels and δ_d is a disparity error tolerance, which is set to δ_d = 1.0 in our experiments. This measure defines when comparing algorithm's output disparity with ground truth disparity, how many percentages of pixel's error are larger than a threshold (1 pixel in this experiment, as most previously studies use this setting).

2. Root Mean Square (RMS) error (E_{RMS})

$$E_{RMS} = (\frac{1}{N} \sum_{(x,y)} |d_C(x,y) - d_T(x,y)|^2)^{\frac{1}{2}} \tag{5.21}$$

   RMS evaluates the square root of the mean of the squares of the disparities difference between the output and the ground truth disparity. Moreover, Average Error (E_{Avg}), a variant of RMS error may be used to provide more comprehensive measurement. Unlike RMS, it measures the average absolute difference.

$$E_{Avg} = \frac{1}{N} \sum_{(x,y)} |d_C(x,y) - d_T(x,y)| \tag{5.22}$$

5.3.2 Evaluation for Depth Map

Only evaluating the quality of disparity map is not sufficient to understand the effect of matching algorithm on robotic manipulation tasks, because the robot needs to know the depth of view in millimetres, in order to decide for example where to grasp. So rather than measuring the disparity map, we choose to evaluate the depth map in order to gain insight with respect to measures that are more related to and interpretable, in terms of the robot manipulation tasks.

For rectified images, it is quite simple to convert horizontal disparity map to depth map, since there is no vertical disparity. But if both horizontal and vertical disparities are considered, we need to use a simplified least-square stereo triangulation routine [Hartley and Sturm, 1997] to convert to depth map, when knowing intrinsic and extrinsic parameters of the cameras.

Root mean square error is also applicable for evaluating depth map. So an equation similar to equation 5.21 is presented to compute the RMS error (E_{RMS}) between the calculated depth map $d_D(x, y)$ and the ground truth depth map $d_{GT}(x, y)$

$$E_{RMS} = (\frac{1}{N} \sum_{(x,y)} |d_D(x, y) - d_{GT}(x, y)|^2)^{\frac{1}{2}} \tag{5.23}$$

5.4 Conclusions

In total, we have constructed five datasets (Section 5.2) for evaluating stereo matching in the context of cloth manipulation (two simulated and three real-world), not only considering drastic depth change, but also measuring micro depth change such as cloth wrinkles. We also include both unrectified and rectified images, and their corresponding ground-truths in some of our created testbeds. The proposed evaluation methodology can be useful and the datasets can serve as standard testbeds for measuring stereo matching for cloth manipulations.

With respect to the five constructed testbeds, specifically, the first dataset (Section 5.2.1) is created by automatically deriving the ground-truths of the depth map, on drastic depth change (box on the table). The second one (Section 5.2.2) is created by generating disparity maps to simulate micro depth changes (small cloth wrinkles). The third dataset (Section 5.2.3) is a real-world cloth manipulation dataset that we aim to present the performance in the real task. Other than that, we also create a simulated dataset (Section 5.2.4) to better understand the effect of simple garment wrinkle on stereo matching algorithm for cloth manipulation applications with depth map ground-truth. For the final dataset (Section 5.2.5), we enrich the third dataset by add more wrinkle shapes, sizes and positions to comprehensively study cloth wrinkle characters.

In addition to the created testbeds, we also present two sets of evaluation metrics (Section 5.3) to quantify the accuracy of disparity map and depth map.

Chapter 6

Guided Filtering based Pyramidical Stereo Matching for Unrectified Images

In the previous chapter, we presented five datasets for stereo matching algorithms in the context of cloth manipulations and proposed methodology to evaluate the accuracy performance of algorithms. In this chapter, we aim to improve the *effectiveness* of stereo matching algorithm and evaluate its performance on the proposed datasets.

Commonly, stereo matching requires rectified images that are computed from calibrated cameras, because this image rectification process could essentially simplify the 2D stereo correspondence problem to 1D and various algorithms have been proposed to rectify images [Tsai, 1987, Zhang, 2000]. However, parametric camera models used for calibration are only approximations of physical cameras while the rectification process itself can add computational complexity to the algorithm. Even cameras that are calibrated to sub-pixel accuracy can cause large matching errors. Hirschmuller and Stefan [Hirschmuller and Gehrig, 2009] present the example of a service robotics scene with round objects, e.g. glasses, where calibration errors of just 0.25 pixel cause artificial disparity discontinuities of 2 pixel. Although the normal process is to first rectify image, and then apply existing stereo matching algorithms, nevertheless, this procedure could be expensive and the non-ideal stereo configurations usually produce inferior results.

On the other hand, stereo matching in the context of cloth perception and manipulation is quite challenging. First of all, since the cloth manipulation requires more accurate representation for the stereo matching (e.g. we need to capture cloth wrinkles), therefore, dealing with high-resolution images is required. This requires the stereo matching algorithm to have low memory cost when dealing with high resolution images. In addition, the cloth manipulation also requires the image matching algorithm not only to be capable of deriving disparity

maps from the objects with drastic depth changes (e.g. a box on a table), but also dealing with cloth containing smooth depth changes (such as small cloth wrinkles). Furthermore, due to the real-time nature of the cloth manipulation task, this requires the stereo matching algorithm to be both efficient and accurate. An algorithm that is easily to parallelize is preferred.

To cope with the above challenges and remedy the rectification issue, in this chapter we aim to answer research questions **RQ 8** and **RQ 9**.

> **RQ 8**: Can we propose a new stereo matching algorithm that better suits to cloth images?
>
> **RQ 9**: Can we effectively derive accurate depth maps by directly matching unrectified cloth images?

The remainder of this chapter is organized as follows. Related work is reviewed in Section 6.1. In Section 6.2, we describe our adapted stereo matching algorithm. We report the experimental results in Section 6.3 and conclude this chapter in section 6.4.

6.1 Related Work

Stereo matching algorithms can be divided into two categories: local methods and global methods. Compared with local algorithms, global algorithms are normally computationally much more expensive, as mentioned in Section 2.1. So in this chapter we focus on local methods that could provide accurate and comparatively speaking, efficient performance.

Adaptive support weight [Yoon and Kweon, 2006] approach is one of the classic correlation-based algorithms which gives good quality results. The idea is that if a pixel inside the support window is close in both color and spatial distance to the central pixel of the window, it receives a higher support weight. The mechanism is that pixels with similar color to the central pixel, are likely to be same object, so there is a large chance that they have similar disparity. Thus, pixels inside the support window have different influence in aggregation.

However due to the large amount of time and memory required in both the pixel-wise support window weight computation and the aggregation processes, adaptive support weight approach is hardly to be considered as an efficient algorithm. The guided filter [He et al., 2013] is a fast and non-approximate linear-time algorithm, whose computational complexity is independent of the filtering kernel size. The idea of this state-of-the-art filter is similar to adaptive support weight algorithm, which has edge-preserving property, that preserves object boundaries and depth discontinuities. Some researches [Mei et al., 2011, Rhemann

et al., 2011] have demonstrated that the guided filter is both effective and efficient in stereo matching applications.

To the authors' knowledge, all of previous matching algorithms using guided filter are applied on rectified stereo images, however, the facilities of guided filter could also be applied on matching algorithms for unrectified images. In this chapter, we demonstrate that adapting guided filter to pyramidical correlation based stereo matching framework [Xu et al., 2014] could also achieve high quality result when applying on large unrectified images. In addition, we also apply this stereo matching to the cloth manipulation context that recently becomes popular [Maitin-Shepard et al., 2010, Sun et al., 2015].

6.2 Unrectified Adapted Matching Algorithm

A pyramidal framework that is similar to Parallel Pyramid Matcher (PPM) [Xu et al., 2014] (also described in Chapter 3.2) is applied on our unrectified matching algorithm, but different strategies (through guided filtering algorithm) are performed to compute the correlation between each pixel in the left image and its corresponding candidate pixels in the right image. We name our algorithm as Adapted Parallel Pyramid Matcher (APPM). This approach contains several steps.

1) Pyramid Creation: As shown in Figure 3.2, our matching algorithm employs a pyramid representation, which is applied to the input images. Pyramid level is adjustable to make sure the image size from top pyramid level is small enough, and the disparities for this level are considered to be equal or less than one pixel. In this multiple scale scheme, an initial estimate for the disparity is computed at a low resolution and the initial disparity estimate from this scale is refined at higher resolutions until the target resolution is achieved. The strategy to generate and refine the estimate is described as below.

2) Cost Computation and Aggregation: In order to generate or refine the disparities at each level of the input pyramid, the algorithm attempts to maximize the similarity between pixels in windowed regions of the left and right images.

First, color difference is calculated between pixel p of the left image and the pixel at coordinates $(p-d)$ of the right image. d is the disparity between two coordinates of the pixels from both images. The color differences $D(p,d)$ for matching pixel p at disparity d are computed as:

$$D(p,d) = \sum_{i=1}^{3} |I^i_{left}(p) - I^i_{right}(p-d)| \tag{6.1}$$

where i demonstrates the ith color channel in RGB space. Then the aggregated matching

costs $C(p, d)$ at pixel p and disparity d are computed as:

$$C(p, d) = \sum_{q \in W_p} W(p, q) D(q, d) \tag{6.2}$$

In the equation above, weighting function $W(p, q)$ computes the likelihood that pixel q lies on the same disparity with the window's central pixel p. The windowed correlation is supported by guided filter [He et al., 2013], whose weighting function $W(p, q)$ is defined as follows:

$$W_{(p,q)} = \frac{1}{|w|^2} \sum_{k:(p,q) \in w_k} (1 + \frac{(I_p - \mu_k)(I_q - \mu_k)}{\sigma_k^2 + \epsilon}) \tag{6.3}$$

Here, I is the reference image, μ_k and σ_k^2 are the mean and variance of I in window w_k centered at the pixel k, $|w|$ is the number of pixels in this window, and ϵ is a regularization parameter.

The window in the left image is named as reference window and the window in the right image is called search window. Because for the unrectified images, both horizontal and vertical disparities need to be considered. So the search window is moved in four directions (up, down, left and right) using a one pixel search step. If we include the null move, there are in total five positions. For each position, aggregated matching costs $C(p, d)$ is computed.

3) Polynomial Minimization: Similar to the PPM, after obtained five aggregated matching costs matrices, 2nd order polynomial minimization is applied to the corresponding elements of the matrices, in order to find the position that has the minimum cost for each pixel. The position could be sub-pixel position to achieve high accuracy. If the local minimum is found to be more than one pixel away from the current position, the relative displacement is clipped to ± 1.

4) Rescale: From current scale to higher resolution scale, interpolation is necessary for each pixel in the disparity map. An interpolated pixel is derived from its four neighbours. The interpolated result is the initial disparity map for the next higher scale resolution.

Iteratively implementing the same matching algorithm on each scale until the highest resolution scale, the final and more accurate disparity map is produced. Notice that in the context of cloth manipulation, we generally deal with garments with continuous surface placed on the table where no occlusion occurs in the area of interests (i.e. table area, see Figure 6.2 for an example). Therefore, we do not need to explicitly consider occulsion detection in our stereo matching.

6.3 Matching Algorithm Experiments

Our proposed stereo matching algorithm is implemented using a 4-core Intel Core i5-2400, 3.1 GHZ computer. The GPU is a GeForce GTX770 graphics card with 4GB of memory from NVIDIA, as specified in Section 4.3.1. We used the CUDA (Compute Unified Device Architecture) technique of NVIDIA Corporation for implementation on the GPU.

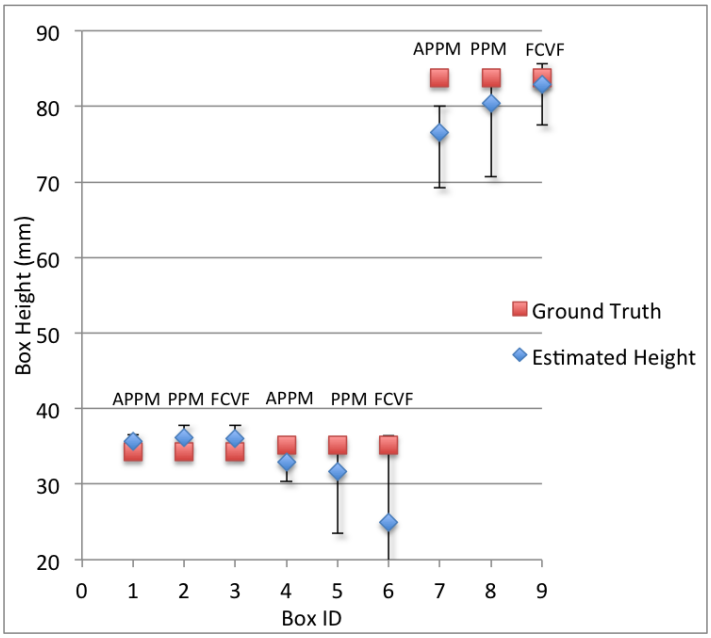
The experiment is based on 3 different datasets as described in Section 5.2. They are real scene with box on table dataset, simulated cloth dataset and garments dataset. Our algorithm (APPM) is evaluated by comparing to another two methods. One is Parallel Pyramid Matcher (PPM) [Xu et al., 2014], which is also a GPU based matching algorithm that utilizes pyramid based stereo matching framework. Another one is Fast Cost-Volume Filtering (FCVF) [Rhemann et al., 2011] for stereo matching on rectified images, which is one of the state-of-the-art algorithm that also involves guided image filtering [He et al., 2013] as part of it. This method is parallelable using GPU, but the authors only made the MATLAB code available. So in this experiment, we mainly focus on accuracy comparison. All experiments are conducted on the same computer.

6.3.1 Evaluation on real scene with box on table dataset

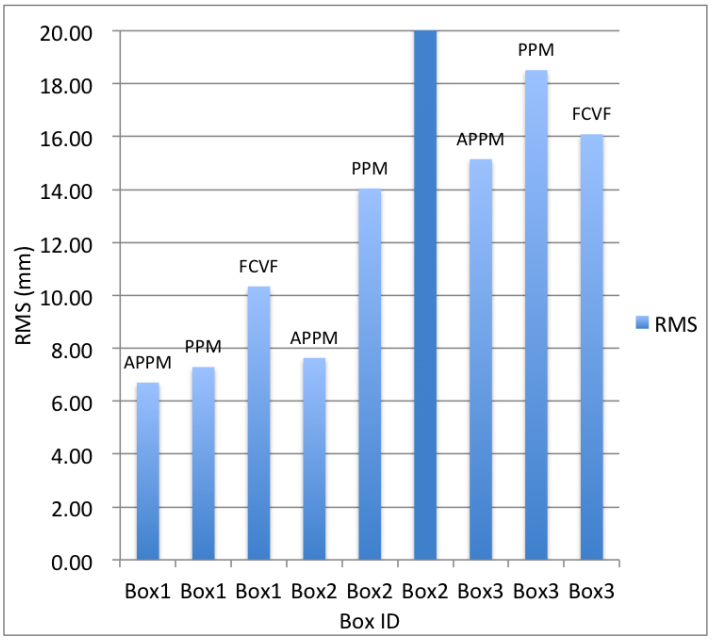
To preserve details, the images of this dataset are quite large, each image contains 4928×3264 pixels, as mentioned in Section 5.2.1. However, due to the huge memory requirement of Fast Cost-Volume Filtering (FCVF) algorithm to which applying on high resolution images are not applicable, we shrunk the images to 1/16 size (1232×816 pixels) for fair comparison to the other two algorithms. In addition, we also report the matching performance on the images of the original size for PPM and our proposed algorithm.

First the three box's height estimation performance is reported, shown in Figure 6.1(a). Ground truth represents the real box height while the estimated height provides the average estimated height based on the matching algorithms. The black lines around the estimated height provide the range of estimated height (i.e. variance). Our algorithm performs the best when applied on boxes with smaller height, maintaining the least error range as well. For the box that has higher height (i.e. more drastic depth change), our algorithm could not perform as well as the other two algorithms. However, in the context of cloth manipulation, the depth changes on the cloth are normally smooth and do not maintain drastic change, we believe our algorithm can suit better and provide more accurate depth map dealing with cloths (as we also empirically demonstrate from the experiments conducted below).

Along with the box's height estimation, we also investigated the root mean square error (RMS) for the box's depth map (Figure 6.1(b)), as defined in Section 5.3. This provides



(a) Three box height estimates



(b) Root Mean Square error (RMS) in pixel for three algorithms

Figure 6.1: Evaluation on real scene with box on table dataset (1/16 size images)

the more refined evaluation than the single point height estimate. On average, our algorithm gives the least root mean square error, which indicates that our algorithm is the most stable matching algorithm among the three.

Due to the huge memory requirement, Fast Cost-Volume Filtering algorithm can not run on the original size of the images. But we conduct another experiment to test whether the high resolution images (with more details) could lead to a better accuracy. Parallel Pyramid Matcher and our algorithm are both applied on the original images and the shrunk images. In addition, the output depth map of the shrunk images are enlarged to the original size to compare with ground truth depth map. The average performance is given on Table 6.1.

Table 6.1: Effect of sub sampling on height errors

	APPM (Ours)		PPM	
	Avg. % of Height Error	Plane Fitting RMS	Avg. % of Height Error	Plane Fitting RMS
1/16 Size	6.38%	11.47 mm	6.60%	12.16 mm
Full Size	**4.78%**	**9.60** mm	6.55%	9.91 mm

Among all, our algorithm with the full size images achieves the best performance on both the percent of height error and plane fitting RMS. The table also indicates that images with more details could help improving the matching accuracy, which is not surprising. Compared with 1/16 size images, our algorithm on the original size images reduced 25% of percent of height error and 16% of plane fitting root mean square error.

6.3.2 Evaluation on simulated cloth dataset

Unlike real scene with box on table dataset, which uses depth map for evaluation, this simulated cloth dataset uses disparity map for accuracy measurement. To test how well an algorithm could handle cloth textures, we introduce three evaluation criteria as shown in Table 6.2, which are Bad Pixel Rate, Average Error and Root Mean Square error.

Table 6.2: Disparity map evaluation on simulated cloth dataset

	Bad Pixel Rate(%)	Avg. Error (Pixel)	RMS (Pixel)
APPM (Ours)	**1.184**	**0.083**	**0.431**
PPM	8.519	1.453	6.139
FCVF	4.443	0.711	1.309

The table shows the average performance of all kinds of different cloth textures. Among the table, our proposed algorithm achieves the best result. Only about 1% of pixels are bad

pixels. In addition, the average error and root mean square error of disparity are both less than 0.5 pixel. This indicates that for cloth materials that have continuous surfaces without drastic depth change, our proposed stereo matching algorithm is able to achieve high quality disparity maps.

6.3.3 Performance on garments dataset

This is the specific task our algorithm really focus on. Due to lack of ground truths for this dataset, we therefore show several examples of one garment on the table in Figure 6.2. The figure shows that our approach is able to produce detailed and accurate depth maps, which is able to preserve fine shape and tiny wrinkles of clothes.

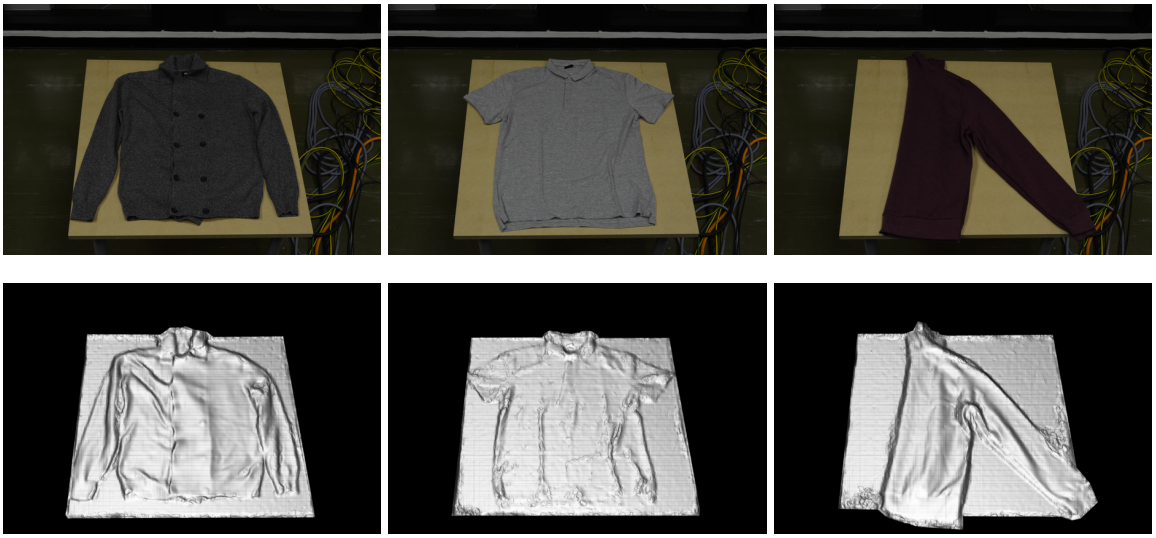
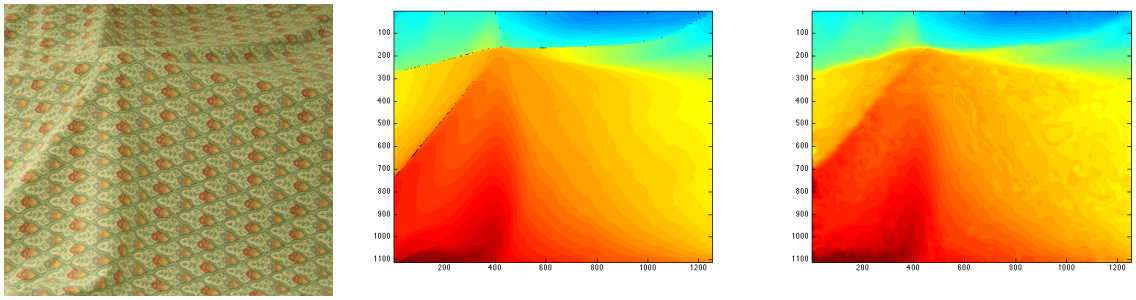


Figure 6.2: Example range map of garments dataset

For reader's interests, we also present the performance on cloth related image pair from the standard Middlebury dataset. Although it is not a suitable dataset to evaluated our algorithm, as we explain in Section 5.1, we can still obtain relatively good performance as shown in Figure 6.3.

6.3.4 Evaluation on Efficiency

Efficiency evaluation is based on the three image datasets mentioned above. Because our algorithm and PPM algorithm are both implemented by GPU code, which is quite efficient, while FCVF one is using Matlab code, so timing comparison is not our main focus. From Figure 6.4, we found that all the algorithms show linear trend, however, timing of our algorithm and PPM one are independent from the number of disparity levels, while timing for



(a) Left Image (b) Ground Truth Disparity Map (c) APPM Disparity Map

Figure 6.3: Example of disparity map for cloth related image from MiddleBury dataset

FCVF is related to not only image size, but also disparity levels. So if the number of disparity levels increases, FCVF algorithm is expected to spend more time to compute. From this perspective, our algorithm is an efficient algorithm.

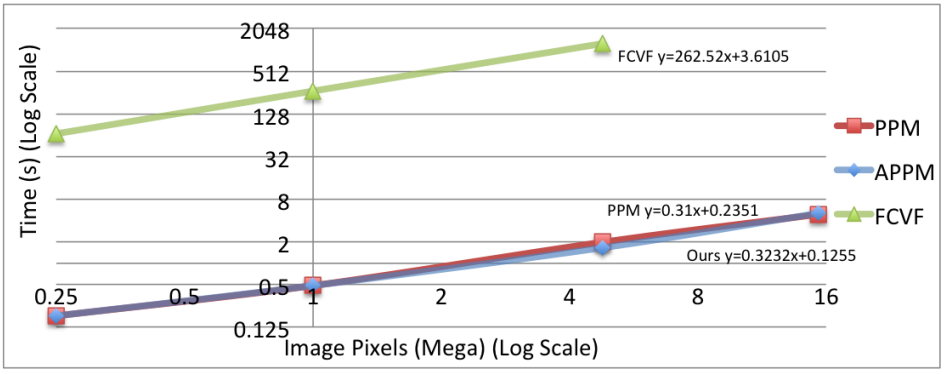


Figure 6.4: Relationship between Image Size and Execution Time

6.4 Conclusions

As robotics becomes popular, conducting stereo matching in the context of cloth manipulation is useful and challenging. This is a consequence of the requirement for high efficiency, accuracy and low memory requirement under the usage of high resolution unrectified images for robotic cloth manipulations.

Therefore, we propose a new stereo matching algorithm Adapted Parallel Pyramid Matcher (APPM) (Section 6.2) that works directly for unrectified images, and is shown empirically to be better suited to the characteristics of the task of cloth manipulations and easily paralleled. By comparing the proposed algorithm with two baseline algorithms on various datasets, we find that this proposed algorithm is capable of processing high resolution images with low memory cost, is easy to parallelize, and accurately derives depth information for small cloth wrinkles. The proposed algorithm suits better with the cloth characteristics and outperforms

the other state-of-the-art stereo matching algorithms (such as guided filtering and pyramidical stereo matching algorithms) (Section 6.3). We also demonstrate that rather than relying on image rectification, directly applying stereo matching through the unrectified images can be also quite effective and meanwhile efficient. Our algorithm on the full size (4928 $\times$3264) images reduced 25% of percent of height error and 16% of plane fitting root mean square error (Table 6.1), while maintains similar efficiency, compared to PPM algorithm (Figure 6.4).

Part IV

Trading off Cloth Stereo Matching Accuracy for Efficiency

Chapter 7

Foveated Stereo Matching for Cloth Manipulation

In the previous chapters, we have attempted to accelerate stereo matching algorithm utilising multi-core hardware architectures, and proposed method to improve the matching quality. However, in some circumstances, where high resolution images are required, stereo matching efficiency still cannot satisfied the requirement (e.g. real-time applications), which becomes a rate limiting factor.

For example, for the task of cloth flattening [Sun et al., 2015], the robotic vision system needs to detect any wrinkles that are larger than 5mm in order to smooth the wrinkles. Therefore, high resolution images (16 mega pixels) are captured to allow analysis of the garment's surface. It was shown previously that performing stereo matching on the Middlebury high resolution full size images (around 6 mega pixels) [Scharstein et al., 2014] can cost almost around 30 seconds to finish one matching for most of the proposed algorithms[1]. Although there have been a few attempts to utilize multi-core architectures such as GPU or CPU to accelerate matching [Mei et al., 2011, Xu and Cockshott, 2015, Xu et al., 2014], in order to achieve the level of efficiency required, the best performing techniques are still too slow for real-time use.

Rather than relying on faster computational hardware, another straightforward way to reduce computation time is to work with coarse resolution images but this restricts the acquisition of detailed information. A better solution, inspired by biological systems, is the use of eye movement together with foveated retinas [Bernardino and Santos-Victor, 2002, Boyling and Siebert, 2000]. The visual system of amniotes has a space-variant nature where the resolution is high on the fovea (the center of the retina) and decreases gradually to the periphery of the visual field. By moving the high resolution fovea we get a detailed representations of our environment. A few approaches have already been proposed to compute disparity maps

[1] http://vision.middlebury.edu/stereo/eval3/.

for foveated active vision systems using, for example, a foveated pyramid representation [Boyling and Siebert, 2000] or a logmap based dense representation [Bernardino and Santos-Victor, 2002]. These approaches can be referred as *foveated stereo matching*.

In this work, we aim to evaluate the performance of the pyramid-based foveated matching in terms of both matching accuracy and efficiency, in the context of robotic cloth manipulations. Specifically, we aim to answer **RQ 10** and **RQ 11**.

> **RQ 10**: Whether foveated matching can be used to accelerate the process while being sufficiently accurate, for two robot cloth manipulation tasks: *flattening* and *grasping*?
>
> **RQ 11**: How to determine when foveated matching is sufficient to accomplish the given robotic cloth grasping and flattening task?

The chapter is organized as follows. We discuss the foveated stereo matching algorithms in section 7.1. In section 7.2, we describe the evaluation metrics used for evaluating stereo matching effectiveness, especially in the context of robot cloth manipulation tasks. We discuss the experiments in section 7.3 and conclude this chapter in section 7.4.

7.1 Foveated Matching Algorithm

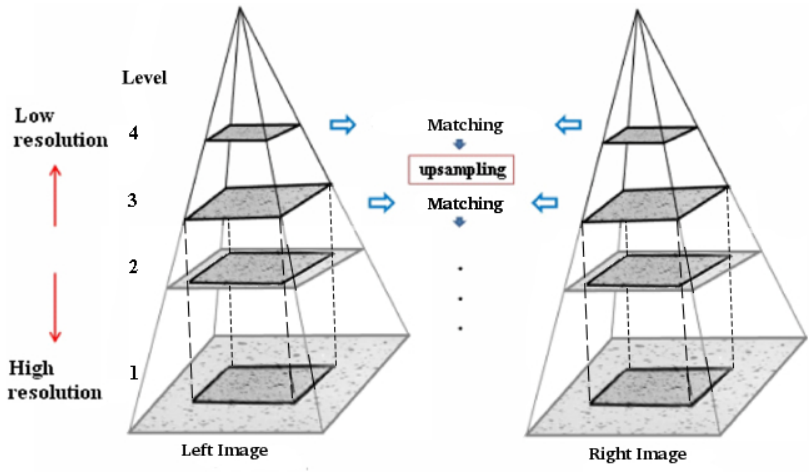


Figure 7.1: Pyramid representation of stereo input image to perform foveated matching at multiple scales

We implemented a parallel extension of Boyling's foveated pyramid matching algorithm [Boyling and Siebert, 2000]. Two image pyramids with gaussian smoothing are first built for the input left and right image. A correlation-based matching process described in [Xu et al., 2014] is computed at a low resolution to generate initial estimate for the disparity and

the initial disparity estimate from this scale is refined at higher resolutions until the target resolution is achieved.

In this chapter, as we descend the pyramid, the correlation-based matching is performed until we reach a particular level, which is called the foveated level f (for example f=3 in Figure 7.1). Here we manually choose the foveated level, but in next Chapter (Section 8.1), we will let the algorithm learn how to automatically select the foveated level.

Let us, for the sake of argument, assume that the foveated level is 400 by 300 pixels. Clearly, by the properties of an image pyramid, this level contains information about the whole scene but with reduced resolution. With the standard pyramid matcher, the following level, called level $f-1$ ($f-1$=2 in Figure 7.1), would be larger by a factor of $\sqrt{2}$ on edge, giving a pyramid plane of 566 by 424. In the foveated matcher [Boyling and Siebert, 2000], we retain only a central block of the same size as level f. The margins between this 400x300 block and the 566x424 plane are simply discarded as shown in Figure 7.1.

Clearly the 400x300 block in level $f-1$ will only include information about the central part of the scene in level f, but will contain this in more detail than in level f. The field of view shrinks but the level of detail increases.

Suppose we have some matching algorithm M which can be applied to a full resolution pyramid, then this same algorithm can be applied to the foveated pyramid, except that with the foveated case, it works on less data, because part of the image data has been discarded. In this chapter, the same correlation-based matching algorithm that was used on the full resolution pyramid is applied to the foveated image pyramid. This gives a stack of disparity maps for each level $l \in 0..f$. This stack is the output.

This stack of foveated matching disparity maps contains less data than the normal pyramid of disparity maps. Suppose the original size image contains N pixels, and the pyramid's scale factor for width and length is $\sqrt{2}$, then the scale factor for image is 2. If there is in total K levels for the pyramid and level F is chosen to be the foveated level, then the sum of pixel number processed for the normal pyramid (S_{Nor}) could be calculated as

$$\begin{aligned} S_{Nor} = N + \frac{1}{2}N + (\frac{1}{2})^2 N + \cdots + (\frac{1}{2})^{(F-1)} N \\ + \cdots + (\frac{1}{2})^{(K-1)} N \\ = N \cdot \frac{1-(\frac{1}{2})^K}{1-(\frac{1}{2})} \end{aligned} \tag{7.1}$$

and the sum of pixel number processed for the foveated pyramid (S_{Fov}) is

$$\begin{aligned} S_{Fov} = (\frac{1}{2})^{(F-1)}N + (\frac{1}{2})^{(F-1)}N + \cdots + (\frac{1}{2})^{(F-1)}N \\ + \cdots + (\frac{1}{2})^{(K-1)}N \\ = N \cdot F \cdot (\frac{1}{2})^{(F-1)} + N \cdot (\frac{1}{2})^{F} \cdot \frac{1-(\frac{1}{2})^{(K-F)}}{1-(\frac{1}{2})} \end{aligned} \tag{7.2}$$

so the ratio of processed number of pixels between those two methods can be calculated as

$$\frac{S_{Fov}}{S_{Nor}} = \frac{(\frac{1}{2})^{F}(F+1) - (\frac{1}{2})^{K}}{1-(\frac{1}{2})^{K}} \tag{7.3}$$

The ratio S_{Fov}/S_{Nor} for the example in Figure 7.1 is 0.47, which means more than half of the data has been discarded. If the pyramid level increases to for example 14 levels, and we choose level 5 as foveated level, then the ratio will be 0.19, which decreases the data size significantly for stereo matching process.

Another data representation is to post process the stack of foveated disparity maps to generate the full size disparity map of foveated matching, which has the same size as the original input left and right images. This option makes it easier to compare the accuracy performance with the standard full field matcher, so it is chosen for our experiments. The disparity map of foveated level f (f=3 as shown in Figure 7.2), is expanded to the original resolution of the next level $f-1$. Pixels of the central area from $f-1$ are superimposed on the centre of this expanded disparity map, which is then expanded again. This process is repeated until the highest resolution level is reached. The result is an disparity map which contains high resolution information in the central area and bordered by concentric areas of decreasing resolution information.

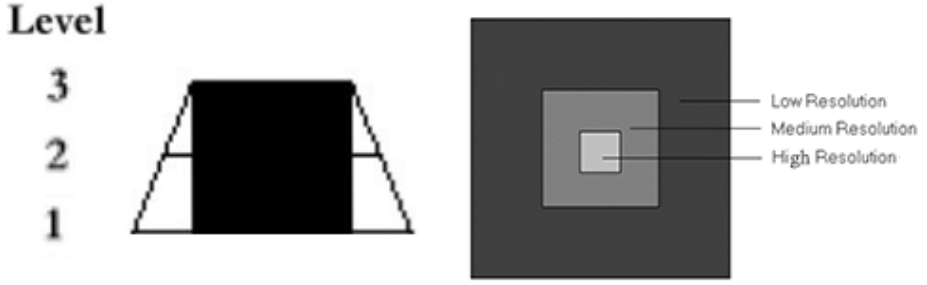


Figure 7.2: Structure of the foveated pyramid and a map of the spatial resolution of a disparity map created by matching the foveated pyramid

The disparity map thus obtained could be further converted to depth map using a simplified least-square stereo triangulation routine [Hartley and Sturm, 1997], given the intrinsic and extrinsic parameters of the cameras. The depth map is necessary because it can give a direct

measure of how depth changes in millimetres, rather than in pixels. This output depth map would have similar character as disparity map, which contains high resolution information in the central area and bordered by concentric areas of decreasing resolution information.

7.2 Evaluation Metrics

After introducing foveated matching algorithm, we utilize a set of evaluation metrics to measure stereo matching effectiveness, both in terms of the depth map and its effects on robotic manipulation tasks.

Stereo Matching Effectiveness

We use RMS error (E_{RMS}) to evaluate depth map quality in terms of robotic manipulation tasks, as mentioned in Section 5.3. Depending on the area where we calculate E_{RMS} upon, we can obtain different insights on the stereo matching performance. Since our work mainly focuses on recognizing wrinkles for robotic manipulation tasks, we are mostly interested in the performance within the wrinkle area and the wrinkle ridge. The wrinkle area is the area of the entire wrinkle that corresponds to the wrinkle width and length we simulated. The wrinkle ridge only focuses on the wrinkle ridge points (highest points) and its small surrounding area ($\pm$ 3 pixels). In addition, we can also report the E_{RMS} in the plane area (the entire cloth area on the table except for the wrinkle area) for comparison.

Robotic Manipulation Tasks

Our ultimate goal is to evaluate the effect of foveated stereo matching on the precision of various robotic manipulation tasks. The most straightforward way to evaluate this is to plug the different foveated matching of various levels (with different accuracy and efficiency) into the robotic system and then track the manipulation task performance. Although this provides the real effects of stereo matching on robotic manipulations, it is time-consuming to conduct this evaluation with lots of garment images. In addition, when evaluating stereo matching, we may also need to take other factors that may affect the robotic performance (such as camera calibration errors) into account. Therefore, as our first step, this chapter focuses on simulating this evaluation. Specifically, we conduct the simulation by assuming the relationships (following previous work) between the stereo matching and two robotic manipulation tasks, i.e. grasping and flattening.

For a given version of the foveated stereo matching, by having several trials on multiple cloth images with different wrinkles in our datasets (shown in Table 5.1), we are able to track the

failure rate (FR) of the robotic manipulation ($1 - precision$), i.e. out of how many trials (number of stereo image pairs in our dataset in this case) the given foveated stereo matching can fail the robotic manipulation task given the assumptions we made (as described below).

$$FR = \frac{1}{N} \sum_{i \in I_N} failure(i) \tag{7.4}$$

where N is the number of trials, I_N is the set of stereo image pairs in our dataset and $failure(i)$ is the indicator function representing whether a given trial for the image pair i is a failure. Next, we describe below how we define this $failure(i)$ function for both cloth grasping and flattening tasks respectively in our simulation.

For the cloth grasping task, a graspable point is selected based on depth information. The most commonly used way to ensure that a point is graspable is by selecting the one that maximizes height [Cutkosky, 2012, Maitin-Shepard et al., 2010, Ramisa et al., 2012, Willimon et al., 2011b]. Therefore, we make several assumptions on whether the robotic gripper can succeed in grasping the cloth given the estimated depth map:

(1). the gripper can not grasp the cloth if the vertical difference between the height of the estimated highest depth point and the ground-truth height of the highest point is more than 10mm;

(2). the gripper can not grasp the cloth if the horizontal difference between the estimated highest point and the ground-truth highest point is more than 5mm.

Only when both criteria are satisfied, is the robot able to succeed in grasping the cloth. Since this setting can vary across different robotic grippers or systems, we therefore assume the more restrictive setting.

For the cloth flattening task, based on the average of manually flattened garment examples performed by a human, it has been shown that [Sun et al., 2015] if the detected wrinkles are less than 5mm, the garment is deemed to be flattened. Therefore, we assume that the stereo matching is required to at least achieve the RMS below 5mm on the wrinkle ridges in order to recognize small wrinkles of around 5mm whilst not falsely recognizing plane areas as wrinkles.

7.3 Foveated Matching Algorithm Experiments

In this section, we describe our experimental results when evaluating foveated matching in terms of stereo matching effectiveness (Section 7.3.1), how matching effectiveness reacts to different wrinkle shapes (Section 7.3.2) and the effect of foveated matching on both robotic cloth grasping and flattening tasks (Section 7.3.3).

Our experiments are based on simulated cloth wrinkle dataset (Section 5.2.4) and implemented in the following ways. The resolution of the stereo image pairs used in our work is fairly large, i.e. we utilize 16 Mega (4928×3264) pixel colour images. The subsample factor of image pyramid is $\sqrt{2}$ in linear dimensions, and in total we utilize 14 levels for the pyramid. The foveated matching algorithm is implemented using a 4-core Intel Core i5-2400, 3.1 GHZ computer. The GPU is a GeForce GTX770 graphics card with 4GB of memory from NVIDIA. We used the CUDA (Compute Unified Device Architecture) technique from NVIDIA Corporation for implementation on the GPU.

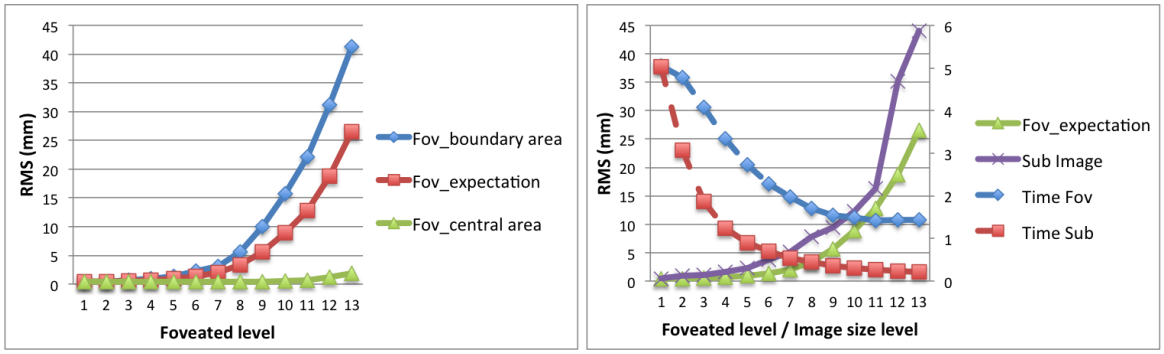
7.3.1 Evaluating Matching Effectiveness

To evaluate how effectiveness (accuracy) of the depth map changes with the foveation level, in this section, we rate the foveated stereo matching in terms of both the wrinkle area and the wrinkle ridge (defined in Section 7.2). Note that when the foveated level is selected to be 1, the effectiveness of the foveated matching is the same as the one applying non-foveated matching algorithm. In this experiment, for the sake of simplicity and clarity, we choose to only perform on a wrinkle of 87 mm width and 84 mm height we simulated (as we found similar results for other simulated wrinkles, see Figure 7.5). The effect of how foveated matching performance change according to different wrinkle types is studied in Section 7.3.2.

Wrinkle Area Evaluation

Figure 7.3(a) presents the RMS evaluation results of the foveated matching algorithm applied to full size images for each selected foveated level within the wrinkle area. As for foveated matching (see Figure 7.2), since the depth map contains its finest information in the central area and more coarse information in the border area, we therefore report three different performance aspects. "Fov_boundary area" only considers points that contain coarsest wrinkle information while "Fov_central area" focuses on the evaluation of central points only that contain finest wrinkle information. "Fov_expectation" represents the mathematical expectation of the performance over the whole wrinkle.

Comparing those three aspects, not surprisingly, we observe that for the foveated matching algorithm at various levels, the central area can always achieve very high accuracy, with RMS always below 5mm in the wrinkle area. As the foveated level increases, the RMS error on the coarsest level of the foveated matcher ("Fov_boundary area") increases dramatically. This trend is not so significant when the foveated level increases from 1 to 7 and the RMS error is relatively stable and still always below 5mm. However, for the foveation above 10 levels, the RMS error increases rapidly (varying from around 20mm to 40mm). The "Fov_expectation"



(a) Accuracy evaluation of foveated matching algorithm applied to full size images

(b) Efficiency and accuracy comparison between two strategies (Foveated & Low resolution matching)

Figure 7.3: Accuracy and efficiency performance on foveated stereo matching on wrinkle area

follows the similar trend. This suggests that foveated matching with 10 and more levels according to our settings is definitely not recommended for accuracy.

Applying foveated matching algorithm on stereo images is not the only way to improve the efficiency as the most straightforward approach is using smaller resolution images, then conducting standard (i.e. non-foveated) stereo matching. In order to have fair comparison between the two strategies, the original size (4928×3264 pixels) image is continually subsampled by a factor of $\sqrt{2}$ in linear dimensions, to build image pairs of different resolutions, following the 14-level image pyramid strategy. Table 7.1 presents the mapping between the foveation level to the image resolution. Note that for each level, the image resolution (size) corresponds to coarsest (boundary) image resolution within the given foveated level.

Table 7.1: Mapping between each foveation level to the image resolution

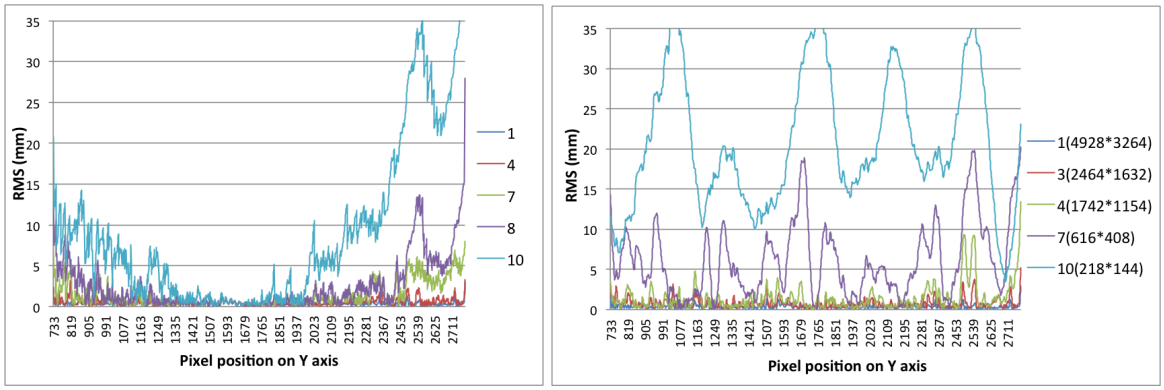
Foveated Level	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Height	4928	3485	2464	1742	1232	871	616	436	308	218	154	109	77	54
Width	3264	2308	1632	1154	816	577	408	288	204	144	102	72	51	36

Therefore, non-foveated matching algorithm (i.e. foveated stereo matching algorithm assuming foveation level is 1) is applied on image pairs of different resolutions ("Sub Image") and the effectiveness performance is shown in Figure 7.3(b), in comparison with "Fov_expectation". We also report the efficiency results. Note that the images of different resolutions is obtained by subsampling from the original stereo images with Gaussian smoothing to attenuate high frequency noises. In terms of matching effectiveness, we can observe that the performance curve of "Sub Image" shows the trend that when the image size decreases, the RMS error of depth map increases. It is clear that "Sub image" always has larger error than "Fov_expectation" for all different foveation level (or image resolution) while it has comparable performance with "Fov_boundary area". With respect to efficiency, "Time sub" is the corresponding efficiency for the matching with different image resolutions

while "Time Fov" is the efficiency trend for foveated matching process. We can observe that comparing two efficiency curve, it indicates that "Time Sub" decreases faster than "Time Fov" while "Time Fov" can reach the maximum efficiency of around 1 second. This is because the input images for foveated matching algorithm are all full size images (16 mega pixels), therefore part of the process could not be accelerated, e.g. integrating stack of disparity maps into full size disparity map, reading and writing full size image data between CPU and GPU, etc. Decreasing the foveated matching base image to smaller resolution (e.g. 4 mega pixels) can reduce the converging efficiency (time) meanwhile still achieve better accuracy performance than simply matching using images of low resolution.

Wrinkle Ridge Evaluation

In the above experiments, RMS error is used to evaluate the depth map accuracy, however, this only shows the average performance for all points in the specified wrinkle area. It is hard to know whether the ridge has been preserved well. In this section, only points close to the ridge are taken into consideration. Since our simulated wrinkle is placed vertically along the table, as shown in Table 5.1, for all the depth map points on the ridge, the X coordinates are all the same. So we only need to present the depth performance along the Y-axis (on the table area, ranged from 733 to 2790 pixels). The results are shown in Figure 7.4.



(a) Performance for different foveated levels (b) Performance for different image size levels

Figure 7.4: Accuracy performance with different foveated/ image size level along with the wrinkle ridge

Figure 7.4(a) presents the results of foveated stereo matching on different foveated levels (1, 4, 7, 8, 10) on the wrinkle ridge. The figure demonstrates that the error is usually small in the central area but larger in the borders. This is because central area contains finer information than borders. We can observe that almost all the points from foveated level 1 and 4 are less than 5 mm, while about 55% of points from level 10 do not meet this. For foveation level 7, 4.8% of the points have an error larger than 5 mm while most of these points are close to border of the wrinkle. We also notice that the distribution of RMS error in Figure 7.4(a)

is not symmetric, because the fovea of our algorithm focuses on the center of the image, rather than the central of the table. Therefore, when comparing performance surrounding the central Y coordinate, which is 1632, the RMS error is roughly symmetric.

For the same wrinkle, ridge performance is also evaluated for the non-foveated matching algorithm on images of different resolutions in Figure 7.4(b). The RMS error of depth map for level 3 (2464x1632 pixels) are all below 5 mm while the RMS error of a small part of level 4 depth map is sharp and non-neglectable. Compared to the performance on the corresponding foveated level shown in Figure 7.4(a), the performance on the wrinkle ridge is worse for lower resolution (such as 7 and 10) while the RMS is generally above 5 to 10mm. This demonstrate that applying foveated matching can achieve better effectiveness than simply applying non-foveated matching on low resolution images.

7.3.2 Wrinkle Characteristic Effects on Foveated Matching

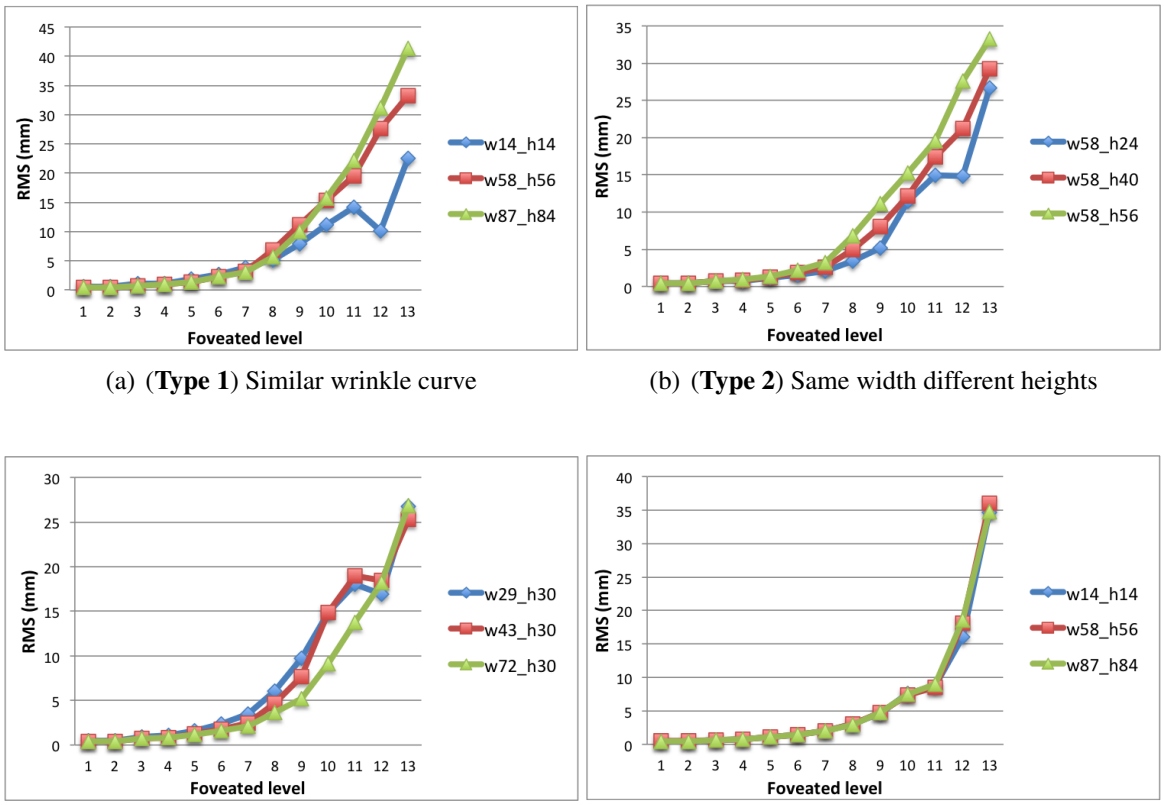
In order to understand how the foveated matching algorithm reacts to different types of cloth wrinkles, the algorithm is applied to various wrinkle types with different width and height created in our simulated cloth wrinkle dataset. Figure 7.5(a)(b)(c) show the corresponding performances of three wrinkle groups mentioned in table 5.1. In addition, the accuracy performance on the plane area is also reported, as shown in figure 7.5(d). In figure 7.5, the RMS represents the wrinkle area evaluation results for various foveated level while for the foveation of each level, we only present the results within the "Fov boundary area".

We can observe from figure 7.5(b) and 7.5(c) respectively that foveated matching can perform relatively better on the wrinkles with smaller heights and larger width for various foveation level. This trend is especially more significant when the foveation level is above 7. This implies that with "smoother" wrinkles, foveated stereo matching algorithm can achieve better performance in terms of effectiveness. In that case, we can even be more aggressive and select higher foveation level in order to tradeoff accuracy for efficiency.

Comparing figure 7.5(a) with 7.5(d), we find that the RMS error in the wrinkle area is generally larger than the plane area. In the plane area, a foveation level of 9 is still be sufficient to achieve an RMS error below 5 mm. This implies again that we can be more aggressive in trading off efficiency for accuracy in the plane area or area with very small and smooth wrinkles.

7.3.3 Effects of Foveated Matching to Robot Cloth Manipulation

In this section, we aim to evaluate the effect of foveated stereo matching on the final robot manipulation task. Figure 7.6 presents the results on the failure rate ($1 - precision$) of



(a) (**Type 1**) Similar wrinkle curve

(b) (**Type 2**) Same width different heights

(c) (**Type 3**) Same height different widths

(d) Plane area without wrinkle

Figure 7.5: The effects of different wrinkle shapes (w means width, h means height) on the foveated matching algorithm in terms of RMS in the wrinkle area

robotic manipulation for both cloth grasping and flattening tasks for various foveated levels. Y-axis is the failure rate of the given robotic manipulation task for the cloth images with different wrinkles and the x-axis is the foveation level. Details on the assumptions made for the robotic manipulation task and how we obtain the failure rate are given in Section 7.2.

We can observe that, to achieve better than 80% cloth flattening task completion precision (i.e. less than 20% failure rate), it requires the foveated matching to be at most on the 5th level to achieve the least required matching accuracy to finish the flattening task. With respect to cloth grasping task, at least foveated stereo matching with 7 levels is required to achieve at least 80% of precision of task completion.

This is not surprising. It implies that the flattening task is generally more difficult than the grasping task, and thus requires lower levels of foveation (more accurate representation) in order to finish the task. This further justifies our choices of assumptions made for the two tasks. From the efficiency perspective, this shows that by using foveation to achieve the same level of accuracy for completing the robotic cloth flattening and grasping tasks, we can reduce the running time by approximately two and three times.

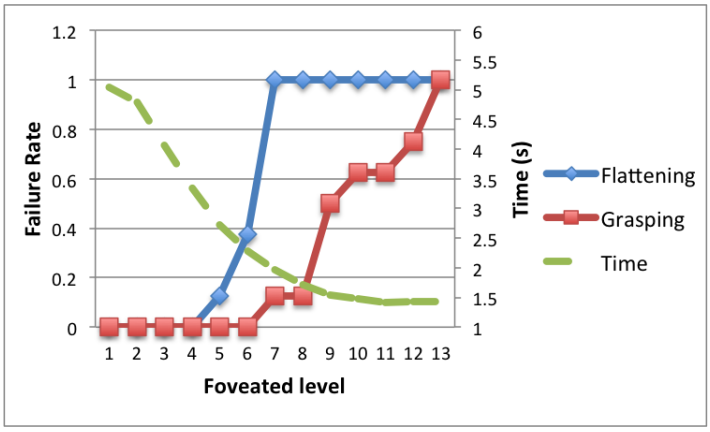


Figure 7.6: Accuracy performance (failure rate) of two robotic manipulation tasks (cloth grasping and flattening) and matching efficiency given various foveation levels based on the simulated cloth wrinkle dataset (clothes of single wrinkle in the middle)

7.4 Conclusions

This Chapter evaluates the performance of a pyramid-based foveated matcher in terms of both accuracy and efficiency, in the context of robotic cloth manipulations. By conducting simulations of cloth wrinkles we obtained depth map ground-truths for our evaluation. Using this simulated dataset, we found that foveated matching is effective in trading off accuracy for efficiency for stereo matching, especially for those cloth wrinkles that are "smooth" (wrinkles of low height and wide width) (Figure 7.5). In addition, we compare foveated matching with the simple solution of just utilizing low resolution images, in terms of the accuracy versus efficiency trade off (Figure 7.3). In addition, by assuming the robotic behavior following previous work for both cloth grasping and flattening tasks, we demonstrate that using foveated matching can achieve the similar level of accuracy for completing both tasks with a two to three times speedup (Figure 7.6).

Note that our work has several limitations: Firstly, we assume that there is only one wrinkle on the cloth. Secondly, we assume a certain stereo matching accuracy is required to achieve the robotic manipulation tasks. This might vary according to different cloth materials, wrinkle properties, etc. We conduct the work of dealing with more real cloth wrinkles with real robotic manipulations in next Chapter.

Chapter 8

Learning to Trade-off Accuracy for Efficiency in Robot Cloth Manipulation

The previous Chapter uses simulation to demonstrate that foveated stereo matching can be useful in trading off accuracy for efficiency for both robotic cloth grasping and flattening tasks. Specifically, with a preliminary study on one wrinkle on a cloth material shown at the center of the foveation (assuming perfect selection of fovea on the camera focus), we demonstrate that it is possible to make the stereo matching more efficient with identical or slightly worse robotic manipulation accuracy.

As we already pointed out, firstly, all the above analysis is based on simulation and we have not verified the above results in a real-world robotic manipulation setting. In addition, although we utilize different cloth materials, we assume they are flattened cloth materials lying on the table while it may be more difficult to deal with the real garments such as trousers and skirts. Secondly, we also assume the perfect foveation selection, which may not be easy. Although previous work has shown by dynamically tuning the camera configurations [Boyling, 2002], it is possible to dynamically tune the foveation focus point (i.e. camera focus point), however, there are inherent difficulties lie into this solution.

To address the above limitations of our previous study, we firstly propose a learning framework to predict the foveation level given a fixed camera-setting. The prediction is made on low resolution images and we demonstrate that it is possible to use the image visual features to predict the appropriate foveation level with certain confidence (e.g. robotic manipulation accuracy above 90%). Here, appropriateness is defined in terms of selecting the (sub)optimal foveation level to keep the robotic manipulation accuracy while improving efficiency. We apply this learning framework again to two common robotic manipulation tasks: cloth grasping and flattening.

Secondly, we also apply this learned framework to real grasping task using only a towel with real-world robots in the CloPeMa project. We verify our simulation above and demonstrate our proposed framework can be effective in the real-world robotic manipulation setting.

We aim to address to following research questions:

> **RQ 12**: Can we simulate more complex scenarios of garments with various wrinkle properties and can we simulate the task success for grasping and flattening under those scenarios?
>
> **RQ 13**: Can we learn to effectively predict which foveation level is required given the image features of the current configuration?
>
> **RQ 14**: Can we apply this learning framework to the foveation level and obtain better efficiency while keeping the same task accomplishment rates? If so, how much can we further accelerate the stereo matching performance?

The chapter is organized as follows. We present how to learn to select foveation level in section 8.1. We discuss the experiments in section 8.2. In section 8.3, we verify the experiment on real-world grasping task. We conclude this chapter in section 8.4.

8.1 Learning to Select Foveation Level

Our aim is to develop a learned classifier to predict for a given image pair, which foveation level should be performed in order to achieve the best tradeoff between efficiency and accuracy on robotic manipulation tasks. We conduct this experiment through both simulation and real-world robotic manipulation task. Specifically we focus again on two common robotic cloth manipulation tasks, grasping and flattening, in order to demonstrate the feasibility of the proposed approach.

8.1.1 Learning Framework Overview

We first introduce an overview of our learning framework for trading off efficiency for accuracy, applied to foveated stereo matching algorithm we described in the previous chapter. Again, we aim to predict the foveation level to be employed in order to successfully accomplish the given cloth robotic manipulation task with least time.

The learning framework can be separated into the following components:

- **Learning target**: this is to provide the cloth manipulation task success labels (i.e. success or failure) in order to serve as the learning target for the machine learning algorithm. Either simulation or real robotic task performance is required to obtain this. For most of this work, we use simulation to obtain the task success label on whether the robot can accomplish the given task for a given setting (e.g. there are multiple wrinkles on a garment positioned at the side further away from the camera focus).

- **Features**: in order to train the learning algorithms, we also need to develop a set of features in order to represent the situation of interests for learning and prediction. In our case specifically, we need to extract a set of features from images taken from the camera and hopefully some of the developed features can capture and distinguish the difficulty of the robotic manipulation task given the setting.

- **Classifier**: after we obtain the features to represent each instance and the learning target, we need a machine learning algorithm to learn a model in order to predict when encountering new instances. Specifically, we treat our problem as a classification problem (i.e. which is the optimal foveation level to perform a given robotic manipulation task successfully meanwhile most efficient) and exploit a machine learning based classifier to learn this.

- **Evaluation**: Given the learned classifier, we need to evaluate the performance in order to understand how it performs. We need two types of evaluation. First of all, since we treat the problem as a classification problem, therefore, we can utilize the traditional classification evaluation metrics for our evaluation (in our case we exploit AUC). Secondly, we also need to understand how our prediction perform in the robotic manipulation settings, rather than only evaluating on learning performance. Specifically, we are interested in two aspects of the robotic manipulation performances: efficiency and accuracy.

Below we describe each component in more details respectively.

8.1.2 Learning Target: Robotic Task Success Label

For a given captured image and robotic cloth manipulation task, our aim is to obtain the success labels as our learning target. Normally, learning requires at least hundreds (and preferably thousands or even more) of instances in order to train properly. Although performing hundreds or thousands of robotic manipulation tasks is possible, it is generally time-consuming, especially if we would like to focus on more than one task. Therefore, we instead exploit simulation as our main tool to obtain the task success labels for both

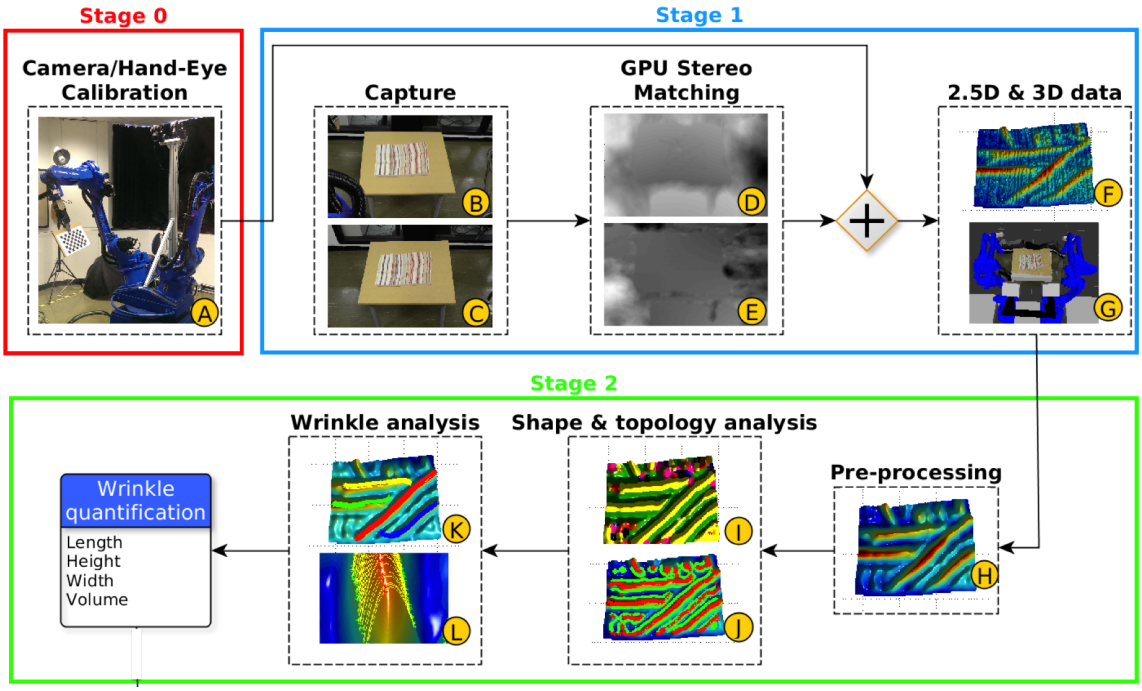


Figure 8.1: The Overall processing pipeline [Sun et al., 2015] for analyzing wrinkles used for generating the cloth grasping and flattening task success labels. The variants we manipulate is the foveation level of the foveated stereo matching component described in Stage 1 and we track the performance of robotic task based on the characteristics of the corresponding outputs of the quantified wrinkles.

robotic cloth grasping and flattening tasks. Specifically, we obtain the detailed characteristics of the cloth (e.g. wrinkles on it) by exploiting the output from the stereo matching, and make several assumptions to derive the labels for a given tasks.

Before we introduce our assumptions, we first present the output we utilize as our sources of evidence to derive labels. Figure 8.1 provides the overall processing pipeline for our analysis. This pipeline mainly includes stereo matching, shape, topology and wrinkle analysis components. The details of the shape, topology and wrinkle analysis components (Stage 2) can be found in [Sun et al., 2015] while the basic idea is that after the stereo matching, a segmented clothing range map is B-Spline smoothed prior to being parsed by means of shape and topology analysis into wrinkle structures. The length, width and height of each wrinkle is used to quantify the topology of each wrinkle and thereby rank wrinkles by size such that a greedy algorithm can identify the top largest wrinkle(s) present.

We apply the same pipeline in this work while the only component we change is the stereo matching component and remain all other components fixed. We then utilize the outputs from the wrinkle analysis component in order to derive the labels. Note that the performance of those components has been proved to be very effective [Sun et al., 2015].

Again, given the output, we make several assumptions in order to obtain the task success labels with a given cloth manipulation task. This label is then used in our framework as the

ground-truth for learning purposes (i.e. our learning target). Although this is still simulation rather performing on the real robot, however, compared to the work we conducted in the previous chapter (Chapter 7), our assumptions are more realistic in the sense that they are based on the actual characteristics of the multiple real wrinkles on garments, rather than solely stereo matching depth map on simulated single-wrinkle garments, and are closer to the real world setting.

Given the wrinkle analysis output for a given image, for any robotic task, we first make one assumption with respect to the ground-truth of the task success label. Specifically, since we do not have disparity or depth ground-truths for a large number of garments (also due to the difficulty in generating disparity ground-truths for deformable materials as we described in Chapter 5), we assume that the original stereo matching algorithm (i.e. foveation level is 1) with the high resolution images (around 16M pixels) is the best a stereo matching can perform and utilize it as the ground-truths (i.e. success). Then we compare the output from the foveated stereo matching algorithms with various foveation levels to the ground-truths and track the discrepancy.

Then for a given robotic cloth manipulation tasks (e.g. grasping or flattening), according to the task characteristics, we make corresponding assumptions to derive whether the discrepancy between the foveated stereo matching and the original matching can result in a task failure. Below we focus on describing the assumptions made and procedures in obtaining the robotic task success binary labels (i.e. success or failure) for cloth grasping and flattening tasks respectively. Note that the source code to generate task success labels for grasping and flattening is public available[1].

Cloth Grasping

For the cloth grasping task, a graspable point is selected based on depth information. The most commonly used way to ensure that a point is graspable is by selecting the one that maximizes height [Cutkosky, 2012, Maitin-Shepard et al., 2010, Ramisa et al., 2012, Willimon et al., 2011b].

For grasping (with various multiple wrinkles on the cloth), the basic idea is to compare the wrinkles quantified with the foveated stereo matching with the wrinkles of the ground-truth (original stereo matching). If the highest point of wrinkles from the foveated stereo matching does not appear on the wrinkle of the ground-truth, then we deem the grasping would not be successful. The detailed pseudo code for generating task success labels for the cloth grasping task is presented in Listing 8.1.

Listing 8.1: Pseudo code for generating task success labels for the cloth grasping task

[1] http://www.dcs.gla.ac.uk/~tian/datasets/taskSimulation.zip

```
For stereo matching depth map originated from foveation level k
        find pixel (x,y) maintains highest height of the wrinkle areas;

        graspLabel(k)=0;

        For all pixel set S in square area (x-theta,y-theta) to
        (x+theta,y+theta) on the original stereo matching depth map
                if pixel (x1,y1) belongs to wrinkle area and S
                        if abs(depth(x,y) - depth(x1,y1)) < 10 mm
                                graspLabel(k)=1;
                        end
                end
        end
end
```

Here we set $theta$ as 5 mm displacement to set S within the original stereo matching depth map. Basically, we assume that if a given wrinkle highest point on the foveated level resides on the original stereo matching wrinkle area (within a 5mm square surrounding) and within a certain height difference (10mm), then the grasping can succeed, otherwise fail. This is similar to the assumption we made in Chapter 7 on obtaining grasping label using the real depth map ground-truth with one single wrinkle on the cloth.

Cloth Flattening

For the cloth grasping task, it is assumed that [Sun et al., 2015] the cloth is flattened with largest wrinkle detected per flattening iteration. Since the flattening decisions are based on this quantified wrinkle analysis, therefore, compared to the original stereo matching, if the foveated stereo matching can derive the same set of wrinkles and provide the identical quantification ranking of the wrinkles that are identified, then the flattening with the foveation can be successful. Therefore, we need to propose a methodology to compare the wrinkles from the foveated stereo matching and the wrinkles from the original stereo matching. The pseudo code for generating task success labels for the cloth flattening task is presented in Listing 8.2.

To quantify the differences between the wrinkles, we need to calculate the distance between two wrinkles. We utilize discrete Frechet Distance [Eiter and Mannila, 1994] to achieve this. Mathematically, the Frechet distance is a measure of similarity between curves that takes into account the location and ordering of the points along the curves. The Frechet distance between the two curves is the length of the shortest distance that is sufficient for traversing both curves and the discrete Frechet distance, also called the coupling distance, is an approximation of the Frechet metric for polygonal curves. This special structure allows the discrete

Frechet distance to be computed in polynomial time by an easy dynamic programming algorithm.

Listing 8.2: Pseudo code for generating task success labels for the cloth flattening task

```
For stereo matching depth map originated from foveation level k

        flattenLabel(k)=0;
        flattenScore(k)=0;

        generate top i wrinkle set W(k) from foveation level k
        where wrinkle w(k) belongs to W(k);

        generate top j wrinkle set V from orginal stereo matching
        where wrinkle v belongs to V;

        for wrinkle w(k) originated from W(k)
                for wrinkle v originated from V
                        if FrechetDistance(w(k),v) / wrinkleLength(w(k))
                        < delta and avgWrinkleHeightDiff(w(k),v) < 5mm
                                flattenScore(k)+=1;
                                break;
                        end
                end
        end
        if flattenScore(k) equals to i
                flattenLabel(k)=1;
        end
end
```

Given this distance calculation, we assume that if the distance between two wrinkles are within a given threshold $delta$ and those two wrinkles' height are within 5mm, then those two wrinkles are "matched". For those matched wrinkles if all the top i wrinkles from the foveated stereo matching can be matched on the top j wrinkles from the original stereo matching ($i <= j$), we deem the flattening task of the given foveated level would succeed, otherwise fail. Here we assume $i = 3$ and $j = 5$ in our setting (i.e. top 3 wrinkles in foveated stereo matching and top 5 wrinkles in original stereo matching). The detailed quantification of those wrinkles for ranking again can be found in [Sun et al., 2015].

8.1.3 Features

To train the classifier, for each image, we need to represent them in a way to capture their differences that reflect the task accomplishment performance. Note that our feature extraction (and also latter the learning algorithm) is supposed to perform on low resolution images

in order to be very efficient when applying and the execution time can be negligible.

Since the focus of our work is to provide a proof-of-concept of this work, we use the set of manually annotated features. This set consists of features directly reflect the wrinkle properties and we aim to study whether they can be effectively used for the learning and prediction. More details of those manual annotated features can be found in Section 5.2.5 while a brief summary is presented below.

- *wrinkle shape*: one wrinkle, twisted multiple wrinkle, random multiple wrinkle, wrinkled garment concealed by a fold.
- *wrinkle position*: middle or side of the camera focus.
- *wrinkle size*: large or small.

Note that ultimately, experiment should be conducted to study visual features that can be automatically and efficiently extracted based on the low resolution images or depth map. Those set of features can be visual features such as HoG (Histogram of oriented Gradients) [Dalal and Triggs, 2005] or even deep learning based CNN visual features [Donahue et al., 2013] that can effectively reflect the wrinkle properties that we describe above. However, investigations on what visual features can reflect those wrinkle properties is out of the scope of this thesis and we perform the learning on those annotated features as a proof-of-concept.

For the feature set, we use all the first and second order features (i.e. the feature themselves and any two feature combinations). An example of the second-order feature is a factor reflects whether a wrinkle is small and positioned at the side, away from the camera focus.

8.1.4 Classifier

Our aim is to develop a learned classifier to predict for a given image pair, which foveation level should be performed in order to achieve the best tradeoff between efficiency and accuracy. This problem can be treated as a multi-class classification problem.

We solve this problem by aggregating a set of binary classifier. Specifically, we train a binary classifier to predict the task success or failure with foveation level as one of its feature. Then for a given image and a given foveation level, we use the trained classifier to obtain the probability of task success. We obtain this probability for each of the foveation level. Afterwards, by an aggregation approach based on heuristic, we obtain the predicted optimal foveation level to perform the task. Next we describe respectively the model and the aggregation methodology.

Binary Classification Model

We use logistic regression to learn the binary classification model as it is a simple, efficient model that has been proved to be useful in lots of applications [Hosmer Jr and Lemeshow, 2004]. Logistic regression is parameterized through a weight vector w. We assume that the posterior probabilities can be estimated through a linear combination of the input features x, passed through a sigmoidal function:

$$P(y = 1|x) = f(x, w) = \frac{1}{1 + exp(-x^T w)}$$

To estimate the parameters w, we minimize the loss function

$$\min_{w \in R^K} \frac{1}{N} \sum_{i=1}^{N} m_i (y_i - f(x_i, w))^2 + \lambda ||w||_1$$

where the hyper-parameter λ controls the L1-regularization.

Given the trained logistic regression model, we estimate the posterior robotic cloth manipulation task successful accomplishment probabilities as $f(x_i, w) \in [0, 1]$, and obtain the predicted class y_i (success, failure) by thresholding the obtained probabilities: $y_i = sign(f(x_i, w) - \theta)$, where threshold θ is usually set to 0.5. However, θ can be chosen anywhere between 0 and 1 to ensure desired precision. We use 5-fold cross-validation to train and test the model, and report AUC (area under the ROC curve) as our performance metric.

Aggregation for Multi-Class Prediction

After we obtain the binary classifier, we can inject different foveation level as its feature and output the probability score of task success for each foveation level. After the probability scores are thresholded to obtain the binary decision (success or failure) for various foveation levels, we simply use a max methodology to aggregate in order to obtain the highest foveation level that is predicted still to succeed in accomplishing the task while mostly efficient[2]. For example, if three of the foveation levels such as 1, 3 and 5 are all predicted to be success by the classifier, then we select the optimal predicted foveation level as 5.

8.1.5 Evaluation

We evaluate on both efficiency and accuracy in terms of the robotic cloth manipulation tasks. The efficiency reflects the execution time of performing the stereo matching, including both

[2]This is because higher foveation level implies higher efficiency, as demonstrated in Section 7.3.3.

the prediction framework (i.e. feature extraction and classifier prediction) and performing corresponding foveated stereo matching at a given predicted foveation level.

We measure on both aspects and aim to obtain the expected performance against the baseline (stereo matching using the full resolution images for all the circumstances). For example, we calculate the efficiency performance using the fraction of expected time of the new approach (learned foveated stereo matching) against the baseline (original stereo matching). With respect to accuracy, we assume the baseline achieves the best performance we can get and aim to find out the fraction of failures using the new framework against the baseline.

$$metric(x) = eval_x(fov)/eval_x(baseline) \tag{8.1}$$

With respect to efficiency, apparently, the lower this fraction, the more efficient the proposed learned approach is. On the contrary, in terms of accuracy, the higher this fraction (closer to 1), the better successful rate of the robot is able to accomplish the task.

8.2 Learning based Experiments

8.2.1 Data and Analysis

Here we briefly recap the enriched garment dataset we have described in Section 5.2.5. We simulate 16 configurations: four different wrinkle types, two wrinkle sizes and two wrinkle positions, which also consists of 17 types of garments. Figure 5.10 presents the examples of all of those different settings in more details. In total, we have 272 stereo image pairs for all those 17 garments under different configurations.

For each image pair, we perform foveated stereo matching for seven different foveation levels (1,3,5,7,9,11,13) and our aim is to select the optimal foveation level from these levels. Therefore, in total, we deal with $272 \times 7 = 1904$ image pairs. We use five cross-validation to train and test the performance of the classification.

Before we perform the classification, we first verify the task success labels generated for both robot manipulation tasks (see Section 8.1.2) by ploting the distribution of failure rates according to different foveation levels we use. Here failure rate is calculated by the number of image pairs for a given foveation level that are labeled as failure divided by the total number of image pairs within that foveation level. Since the image pairs originate from the enriched garment dataset (see Section 5.2.5), they consist of clothes of a variety of materials, wrinkle properties such as shape, size and position.

Figure 8.2 presents the results and we can observe that again, not surprisingly, flattening is deemed to be more difficult than grasping, with faster failure rate degradation as the foveation

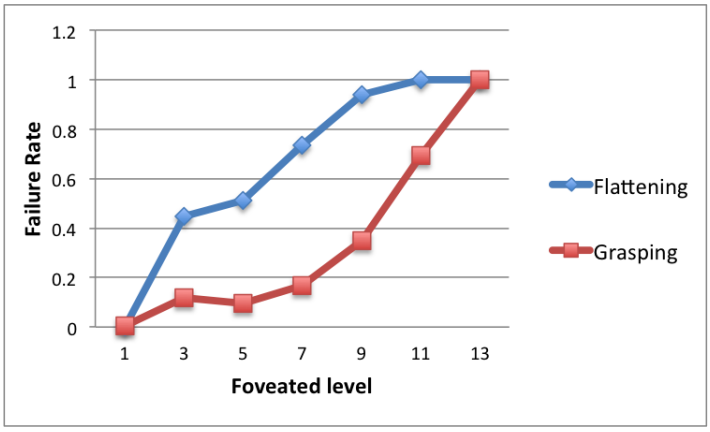


Figure 8.2: Accuracy performance (failure rate) of two robotic manipulation tasks (cloth grasping and flattening) given various foveation levels and generated task success label based on wrinkle analysis for the enriched garments dataset (clothes of various wrinkle properties).

level increases. Compared to the results in Figure 7.6, we found that for the performance of the small foveation levels (such as 3 or 5), the failure rate is higher. This is expected since we use the enriched garment dataset while the images can contain multiple wrinkles placed at the side away from the camera focus, which are more difficult to be grasped or flattened. In addition, since our simulation in Figure 8.2 is based on wrinkle analysis, compared to only utilizing depth map information in Figure 7.6, we believe this conforms better with the real-world scenarios.

8.2.2 Classification Performance

Grasping

Given the overall 1904 image pairs for training and testing, we have in total 1238 image pairs (65%) as positive example for successfully grasping the clothes and the rest as negative examples.

For the grasping classification (five-fold cross validation), the obtained AUC is 0.84, which is highly performing. This demonstrates that using the wrinkle properties in addition to the foveation level can be very effective in predicting the success or failure for the cloth grasping task.

When looking at the top important features based on the classification coefficients (based on normalized features) that are significant ($p\text{-}value < 0.05$), we can observe that "foveation level" (negative contribution to success), "size(smallWrinkle)" (negative contribution to success) and "Type(ConcealedFoldWrinkle)" (negative contribution to success) are the top features for predicting grasping success.

Table 8.1: Robotic cloth manipulation performance (efficiency and accuracy) using the learned foveation level selection classifier

Probability	Grasping		Flattening	
Threshold	Efficiency	Accuracy	Efficiency	Accuracy
0.30	33.4%	75.7%	48.0%	54.0%
0.40	37.0%	90.1%	60.8%	62.9%
0.50	41.8%	93.8%	73.0%	68.4%
0.60	49.5%	94.1%	76.5%	70.2%
0.70	63.3%	97.0%	82.0%	78.3%
0.75	69.3%	97.4%	91.3%	89.7%
0.80	72.8%	98.2%	96.3%	96.7%
0.90	85.8%	98.5%	96.3%	93.8%

Flattening

Given the overall 1904 image pairs for training and testing, we have in total 650 image pairs (34%) as positive example for successfully flattening the clothes and the rest as negative examples. Not surprisingly, the percentage of positive examples for cloth flattening is smaller than the grasping task as flattening is a more difficult task.

For the flattening classification (five-fold cross validation), the obtained AUC is 0.90, which is even better than grasping. This shows that compared to grasping, using the wrinkle properties in addition to the foveation level can be even more effective in predicting the success or failure for the cloth flattening task.

When looking at the top important features based on the classification coefficients (based on normalized features) that are significant ($p\text{-}value < 0.05$), we can observe that "foveation level" (negative contribution to success), "foveationLevel:Position(side)" (negative contribution to success) and "Type(singleWrinkle):Position(side)" (positive contribution to success) are the top features for predicting flattening success.

8.2.3 Robotic Cloth Manipulation Performance

Now we present the results on applying the trained classifier for the robotic manipulation tasks and track their efficiency and accuracy. Table 8.1 present the results for both grasping and flattening. Here, the threshold refers to the threshold set on the obtained classifier probabilities to infer the predicted class (success or failure).

We can observe that for grasping, when setting the threshold to be 0.40, with over 90.0% of the accuracy achieved compared to the original stereo matching, we can only spend 37.0% of it original stereo matching time. This implies that we can accelerate around three times

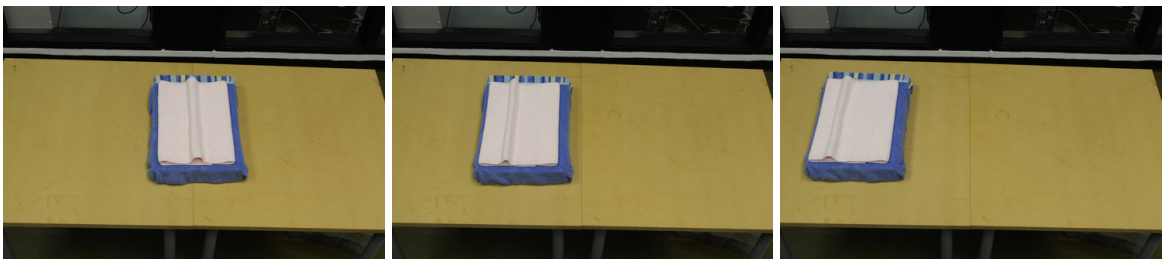
for grasping while remaining over 90% of accuracy. On the other hand, for flattening, this task seems to be more difficult. To maintain the flattening accuracy to be around 90.0% (threshold equals to 0.75), we can only save around 8.7% of the execution time.

8.3 Real-world Robotic Cloth Manipulation

8.3.1 Cloth Grasping Task

In this section, we outline our verification experiments of real-world robot (CloPeMa robot) to grasp the clothes using foveated stereo matching. In order to know the performance of all the foveation levels, we perform the robotic manipulation grasping tasks for different foveation levels (i.e. 1,7,9,11,13). Note that we decide this due to the time constraints and based on results from our pilot study investigating how the foveation affect the robotic cloth manipulation performance.

We design to evaluate grasping performance using one simple wrinkle with three different positions and two wrinkle sizes as shown in Figure 8.3. For each case, the experiment is repeated for 5 times for stable results (e.g. to avoid random error). Towel is used in this task because the material of towel is neither too soft (e.g. T-shirt) nor too hard (e.g. jeans), which makes it easy to form different wrinkle sizes and shapes.



(a) Position 1 with big wrinkle (b) Position 2 with big wrinkle (c) Position 3 with big wrinkle

(d) Position 1 with small wrinkle (e) Position 2 with small wrinkle (f) Position 3 with small wrinkle

Figure 8.3: Three wrinkle positions and two wrinkle sizes tested for the grasping task

The grasping strategy applied in our robot system is to grasp wrinkle point which has the largest depth-width ratio. One straight wrinkle shape is chosen because this can ensure that

the grasping point is on the wrinkle ridge. If there are multiple wrinkles, then it is harder to guarantee the manipulated shape is the same every time and it is not easy to decide which wrinkle ridge is the right one to grasp. As for the two wrinkle sizes, we name it big wrinkle and small wrinkle respectively. Every time the wrinkle height is measured by vernier caliper. However, because the towel is soft and deformable, the wrinkle size is not guaranteed to be a fixed number, but within a small range. The big wrinkle ranges from 30 mm to 40 mm, while the small wrinkle ranges from 10 mm to 20 mm. For positions, Figure 8.3(a)(d) demonstrate position 1 (Center); Figure 8.3(b)(e) demonstrate position 2 (Middle); Figure 8.3(c)(f) demonstrate position 3 (Side). All three positions are carefully marked on the table to maintain repeatability.

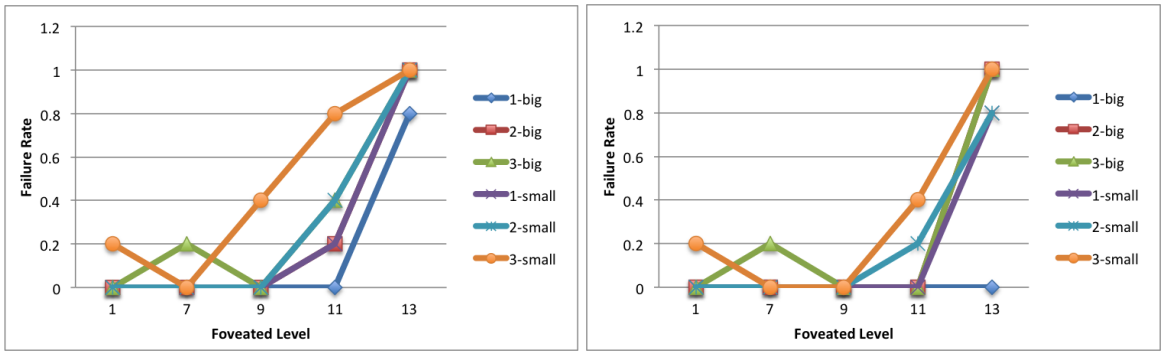
As shown in our simulated grasping task, if the foveated matching is applied (e.g. foveated level is 13), the error may increase dramatically. In that case, the estimated towel depth might be lower than the true depth of the table. Therefore, in order to protect the robot gripper and the table, the towel is placed on an empty paper box covered with cloth (blue).

8.3.2 Experiments

The grasping point should be on the wrinkle ridge as mentioned in the last section, so for this grasping task, only if the gripper grasp the wrinkle area, it can be considered as a successful grasp, otherwise a failure. The concept of "touching" is defined in the similar fashion. For foveated stereo matching, due to the generated depth map is not be accurate enough, the gripper may able to touch the towel wrinkle, but might not in the right position to pick it up. The "Touching" is also considered for grasping task evaluation.

Figure 8.4 presents the results of failure rate for grasping task (including the touching based evaluation) when applying foveated matching algorithm with different foveated levels (1, 7, 9, 11, 13). Matching performance with 1 foveated level is equal to applying non-foveated matching algorithm. Each experiment is repeated for 5 times, but foveated level 1 for "3-small" (wrinkle position 3 with small wrinkle size) and foveated level 7 for "3-big" (wrinkle position 3 with big wrinkle size) failed once, so for these two, the failure rate are 0.2.

If assuming that the failure rate of 0.2 is reasonable, then for all cases, 7 foveation level is sufficient. This also verifies the conclusion made for simulated robotic grasping task (similar setting of single wrinkle as shown in Figure 7.6). Specifically, if the cloth is placed at the center, choosing 9 foveation level is acceptable. Comparing 8.4(a) and (b), we find that position 1 (center) normally have relative accurate performance, which can be explained by the foveation character (the fovea contains the highest resolution information in the central area of the image and bordered by concentric areas of decreasing resolution information). The big wrinkle size always performs better than the small wrinkle, because when the wrinkle is



(a) Grasping Failure Rate (b) Touching Failure Rate

Figure 8.4: Grasping Task Failure Rate (i.e. "1-big" means wrinkle position 1 with big wrinkle size)

small, it becomes hard to recognise as a wrinkle, and this may lead to misunderstanding in the wrinkle analysis process.

8.4 Conclusions

By utilizing the created more complex enriched garment datasets (Section 5.2.5) with more wrinkle types, positions and sizes simulated, we propose thorough methods to derive the success labels given the quantified wrinkles for both robotic cloth grasping and flattening tasks (Section 8.1.2). Then we propose a machine learning based classification approach that is very effective in predicting the success and failure of a given robotic manipulation task. We use our simulated grasping instances as our proxy to train the prediction model. Specifically, we treat the grasping prediction as a binary classification task and consider all instances that fall within the height inaccuracy range as *positive*. For *negative* examples, we treat all the other cases (either the highest point does not reside on the real wrinkle or there is inaccurate estimation of the wrinkle height). We use foveation level as one of the features to make the prediction, and for a given image we use the trained classifier to provide the probability of successful grasping given all the different foveation levels. Then we propose a simple aggregation strategy to determine the optimal foveation level to be selected. With the traditional classification metric AUC (area under the ROC curve) of 0.82 and 0.90 achieved respectively for grasping and flattening tasks (Section 8.2.2). When applying this learned foveated stereo matching to the robotic manipulation tasks, we found that for grasping, with our learned classifier, with over 90.0% of the accuracy achieved for the dynamic foveated stereo matching, compared to the original stereo matching, we only need to spend 37.0% of it original stereo matching time (Section 8.2.3). This implies that we can accelerate almost three times for grasping while retaining over 90% accuracy. On the other hand, for flattening, this task seems to be more difficult. To maintain the flattening accuracy over 90.0% for the

dynamic foveated stereo matching, we can only save around 8.7% of the execution time. We also verify this using the real-world CloPeMa robot on the towel grasping task (Section 8.3.2).

Part V

Conclusions and Discussion

Chapter 9

Conclusions

In this thesis, we presented work towards extensively studying stereo matching, contributing to the long-term goal of developing effective ways for efficient and accurate robotic stereo matching for cloth manipulation. Although stereo matching has been extensively investigated in the past, however, in the context of robotic cloth manipulation, how to *effectively* and *efficiently* apply stereo matching for cloth materials is still an open question.

To address this challenge, the thesis work consists of accelerating the stereo matching algorithms using various hardware architectures, developing accurate stereo matching algorithms and evaluation methodology that fit well with robotic cloth manipulation tasks, and optimally trading off accuracy for efficiency in order to adjust stereo matching algorithm to still successfully accomplish robotic task but with much less time. The six research chapters addressed the challenges of stereo matching in the context of robotic cloth manipulation as follows, organized into three parts:

- **(Efficiency)** how to accelerate stereo matching algorithms using various hardware architectures;
- **(Accuracy)** how to evaluate stereo matching for cloth images and can we propose better stereo matching algorithms that well fit for cloth characteristics;
- **(Trade-off Accuracy for Efficiency)** how to trade-off stereo matching accuracy for efficiency for robotic cloth manipulation so that the robot can successfully accomplish the given task with least time.

With respect to *efficiency*, firstly, by exploring a variety of different hardware architectures (Part II) such as multi-core CPU (Chapter 3) and GPU (Chapter 4) to accelerate stereo matching, we demonstrate that the parallelised stereo-matching algorithm can be significantly accelerated, maintaining $12\times$ and $176\times$ speed-up respectively for multi-core CPU and GPU,

compared with SISD (Single Instruction, Single Data) single-thread CPU. Moreover, to analyze the origin of the speed-up and gain deeper understanding about the choice of the optimal hardware, the stereo matching algorithm is broken into key sub-tasks and the efficiency performance of those sub-components is tested for different hardware architectures.

In terms of *accuracy* (Part III), due to the fact that there are no cloth based testbeds with depth map ground-truths for evaluating the accuracy of stereo matching performance in this context, we create five different testbeds to facilitate evaluation of stereo matching in the context of cloth manipulation (Chapter 5). In addition, we propose to adapt guided filtering algorithm into the pyramidical stereo matching framework that works directly for unrectified images, and evaluate its accuracy performance utilizing the created cloth testbeds (Chapter 6). We demonstrate that our proposed approach is not only efficient, but also accurate and suits well especially to the characteristics of the task of cloth manipulations. This also shows that rather than relying on image rectification, directly applying stereo matching through the unrectified images can be also quite effective and meanwhile efficient.

With respect to *accuracy-efficiency trading off* (Part IV), rather than accelerating stereo matching algorithm using current multi-core hardware infrastructure, we further explore algorithmically whether we can improve the efficiency while maintaining reasonable accuracy for robotic cloth manipulations. We propose to use foveated matching algorithm, an algorithm inspired by biological vision systems, and found that it is effective in trading off accuracy for efficiency for stereo matching (Chapter 7). By assuming the robotic behavior following previous work for two common robotic manipulation tasks, i.e. cloth grasping and flattening, we demonstrate that using foveated matching with an optimal selection of its foveation level can almost achieve the same level of accuracy for completing both tasks with two to three times of acceleration applied on the single-wrinkle garments. Furthermore, we aim to evaluate this foveated stereo matching algorithm with more wrinkle types, positions and sizes simulated and test its real robotic cloth manipulation performance (Chapter 8). By extracting simple features from the cloth images very efficiently, we demonstrate that we can use machine learning techniques to predict the optimal foveation level in order to accomplish the robotic cloth manipulation tasks successfully (above certain accuracy level) meanwhile most efficiently. We apply this learned foveated matching algorithm on simulated images of robotic grasping task and demonstrate its effectiveness, with the acceleration of stereo matching of almost three times and accomplish the grasping tasks of over 90% of accuracy.

Below, we provide a more detailed summary of the contributions and results of our research, and answer the research questions set out at the beginning of this thesis (Section 9.1). We conclude with an outlook on future research directions (Section 9.2).

9.1 Summaries of Main Findings

The broad question that motivates the research in this thesis is: *Can we develop a robotic stereo matching based vision system that is both sufficiently efficient and accurate for the task of robotic cloth manipulation?* We answer this question as follows.

9.1.1 On the Acceleration of Stereo Matching

Chapter 3 and 4 improve the *efficiency* of image matching approach under two specific parallel environments (i.e. multi-core CPU and GPU), to enable it to run in real-time.

Multi-core CPU Acceleration

We utilize the multi-core CPU to parallelize stereo matching and specifically answer the following research questions (RQs).

> **RQ 1**: How much speed-up can multi-core CPUs offer on accelerating stereo matching algorithms? Is there an acceleration plateau regardless of further increase of the number of CPU cores?

The experimental results (Figure 3.4) show significant performance accelerations with multi-core CPUs giving $12\times$ speedup on average (using Vector Pascal programming language on Sandybridge architecture), compared to SISD single-thread CPU performance. The acceleration plateau of the MSSM stereo matching algorithm (Section 3.2) is achieved at around 30 cores while further increasing the number of cores (even to 64 cores) does not further improve the performance.

> **RQ 2**: How do different CPU system architectures (such as 486, Pentium, Pentium 4 and Sandybridge) affect the acceleration performance?

When comparing different architectures, it is not surprising that the recent Sandybridge outperforms Pentium 4, Pentium and 486 (Table 3.3). Performance on Sandybridge and Pentium 4 architecture is about 4 times and twice faster than on 486 respectively, while the performance of Pentium is similar to 486.

> **RQ 3**: After decomposing the stereo matching algorithms into different sub-components, how do different sub-tasks perform differently in terms of acceleration? Can we further infer from the algorithmic perspective, which characteristics can benefit more from the multi-core CPU acceleration?

We found that the optimized acceleration of one stereo matching sub-task "interpolation" has a greater impact on the overall speed-up than another sub-task "convolution" (Figure 3.6). This is because "interpolation" is more suited for parallel execution, but "convolution" currently has a speed-up limitation due to its high demands on storage of intermediate results.

GPU Acceleration

As stereo matching is an image processing algorithm and it is natural to assume that utilizing GPUs can better accelerate stereo matching algorithms than multi-core CPUs. In order to study whether this is the case, we conduct experiments to answer the following questions.

> **RQ 4**: How many times speed up can GPUs achieve on accelerating stereo matching algorithms? How does it compare with multi-core CPUs?

We found that overall the Nvidia GeForce GTX 770 GPU (1536 cores) that programmed by CUDA C language, gives $176\times$ acceleration (Table 4.2), which is significantly better than the 64-core AMD CPU programmed by Vector Pascal language ($12\times$).

> **RQ 5**: How do different stereo matching sub-tasks perform differently in terms of acceleration on GPUs? Does the GPU acceleration trend vary from the one of multi-core CPUs?

We found that processes of array calculations (such as polynomial maximization as one sub-task) can be best paralleled with a 731 times speed-up on GPU (Table 4.2). Furthermore, we found that when implementing algorithms on GPU, one should let GPU do parallel computing as much as possible, and try the best to keep intermediate variables on GPU only, because the data transfer within GPU is much faster than the data transfer to CPU (Table 4.1).

9.1.2 Accurate Stereo Matching for Cloth Manipulation

Chapter 5 and 6 address the *effectiveness* aspect, i.e. how can we improve the stereo matching algorithms that perform more accurately on cloth images? We propose evaluation methodologies and novel stereo matching algorithms that suits the characteristics of cloth images.

Cloth Manipulation Stereo Matching Evaluation

To evaluate the performance of stereo matching in the context of cloth manipulation, ground-truth disparity is required for those images with garments within it. We construct various testbeds and aim to answer the following questions.

> **RQ 6**: How to construct the proper datasets to evaluate stereo matching algorithms in the context of cloth manipulations? Can we also provide disparity ground-truths for both unrectified and rectified images?

We have constructed five datasets (Section 5.2) for evaluating stereo matching in the context of cloth manipulation (two simulated and three real-world), not only considering drastic depth change, but also measuring micro depth change such as cloth wrinkles. We also include both unrectified and rectified images, and their corresponding ground-truths in some of our created testbeds. The proposed evaluation methodology can be useful and the datasets can serve as standard testbeds for measuring stereo matching for cloth manipulations.

> **RQ 7**: What evaluation metrics should be used to evaluate the accuracy performance of cloth related stereo matching algorithms given the testbeds we developed?

We review a set of evaluation metrics (Section 5.3) that are based on either disparity map or depth map ground-truths for quantifying the stereo matching accuracy on the clothes.

Guided Filtering based Pyramidcal Stereo Matching

To study whether it is possible to propose a stereo matching algorithm that directly applies to unrectified images and perform well on cloth images, we aim to answer the following questions.

> **RQ 8**: Can we propose a new stereo matching algorithm that better suits to cloth images?

We adopt a single stage stereo matching methodology that derives the disparity information directly from the unrectified images and adapt a state-of-the-art guided filtering algorithm within a pyramid based stereo matching framework (Section 6.2). We found that this proposed algorithm is capable of processing high resolution images with low memory cost, is easy to parallelize, and accurately derives depth information for small cloth wrinkles. The proposed algorithm suits better with the cloth characteristics and outperforms the other state-of-the-art stereo matching algorithms (such as guided filtering and pyramidical stereo matching algorithms) (Section 6.3).

> **RQ 9**: Can we effectively derive accurate depth maps by directly matching unrectified cloth images?

We show that rather than relying on image rectification, directly applying stereo matching through the unrectified images can be also quite effective and efficient. Our algorithm on the full size (4928 $\times$3264) images reduced 25% of percent of height error and 16% of plane fitting root mean square error (Table 6.1), while maintains similar efficiency, compared to PPM algorithm (Figure 6.4).

9.1.3 Trading off Cloth Stereo Matching Accuracy for Efficiency

Chapter 7 and 8 apply the stereo matching algorithms to various robotic cloth manipulation scenarios and study how the stereo matching performance translates into the robotic task accomplishment, in terms of both effectiveness and efficiency.

Foveated Stereo Matching for Cloth Manipulation

To tailor the stereo matching to tradeoff accuracy for efficiency for the given robotic manipulation task, motivated by biological system, we exploit utilizing foveated stereo matching to accomplish this, rather than working with coarse resolution images which restricts the acquisition of detailed information, and aim to answer the following questions.

> **RQ 10**: Whether foveated matching can be used to accelerate the process while being sufficiently accurate, for two robot cloth manipulation tasks: flattening and grasping?

We found that foveated matching is effective in trading off accuracy for efficiency for stereo matching, especially for those cloth wrinkles that are "smooth" (wrinkles of low height and wide width) (Figure 7.5). In addition, we compare foveated matching with the simple solution of just utilizing low resolution images, in terms of the accuracy versus efficiency trade off (Figure 7.3).

> **RQ 11**: How to determine when foveated matching is sufficient to accomplish the given robotic cloth grasping and flattening task?

By conducting simulation by assuming the robotic behavior following previous work for both cloth grasping and flattening tasks, we demonstrate that using foveated matching can achieve the similar level of accuracy for completing both tasks with a two to three times speedup (Figure 7.6).

Learning to Trade-off Accuracy for Efficiency in Robot Cloth Manipulation

We deal with more realistic cloth wrinkles with real robotic manipulations to study on the effectiveness of this foveated stereo matching framework in achieving better accuracy-efficiency tradeoff, and aim to answer the following questions.

> **RQ 12**: Can we simulate more complex scenarios of garments with various wrinkle properties and can we simulate the task success for grasping and flattening under those scenarios?

By utilizing the created more complex enriched garment datasets with more wrinkle types, positions and sizes simulated, we propose thorough methods to derive the success labels given the quantified wrinkles for both robotic cloth grasping and flattening tasks (Section 8.1.2).

> **RQ 13**: Can we learn to effectively predict which foveation level is required given the image features of the current configuration?

We propose a machine learning based classification approach that is very effective in predicting the success and failure of a given robotic manipulation task, with an traditional classification metric AUC (area under the ROC curve) of 0.84 and 0.90 achieved respectively for grasping and flattening tasks (Section 8.2.2).

> **RQ 14**: Can we apply this learning framework to the foveation level and obtain better efficiency while keeping the same task accomplishment rates? If so, how much can we further accelerate the stereo matching performance?

When applying this learned foveated stereo matching to the robotic manipulation tasks, we found that for grasping, with our learned classifier, with over 90.0% of the accuracy achieved for the dynamic foveated stereo matching, compared to the original stereo matching, we only need to spend 37.0% of it original stereo matching time (Section 8.2.3). This implies that we can accelerate almost three times for grasping while remaining over 90% of accuracy. On the other hand, for flattening, this task seems to be more difficult. To maintain the flattening accuracy over 90.0% for the dynamic foveated stereo matching, we can only save around 8.7% of the execution time. We also verify this to the real-world CloPeMa robot on the towel grasping task (Section 8.3.2).

9.1.4 Main Conclusion

In this thesis, we accelerate stereo matching algorithms using various hardware architectures, and propose better stereo matching algorithms that well fit for cloth characteristics. We also find a way to trade-off stereo matching accuracy for efficiency for robotic cloth manipulation so that the robot can successfully accomplish the given task with least time. To summarize, we extensively study stereo matching, contributing to the long-term goal of developing effective ways for efficient meanwhile accurate robotic stereo matching for cloth manipulation in the research community.

9.2 Future Work

The work presented in this thesis suggests the following subjects for further research.

9.2.1 Acceleration on Various Parallelizable Architectures

Currently our algorithms are only applied on multi-core CPU and GPU architectures for acceleration, but there are many other parallel architectures, such as FPGA, DSP and XeonPhi. In addition, these architectures can be combined as heterogeneous systems. Depending on the task requirement, different parallel architectures can be chosen as the best fitted architecture to achieve higher acceleration rate.

9.2.2 Robust Stereo Matching Algorithm

In this thesis, we present the performance of our stereo matching algorithms in the context of robot cloth manipulation. However, we do not have knowledge whether these algorithms are robust enough to accomplish various tasks in various scenarios. For example, if there are other objects placed on the clothes to be manipulated, is the stereo matching algorithm sufficiently robust to still capture the wrinkle details. Therefore we would like to evaluate our stereo matching algorithms on various environments to study the robustness. Moreover, how to adapt our algorithms to further improve the robustness whilst maintain efficiency is an interesting question to further investigate.

9.2.3 Dynamic Vergence System for Foveation

Our previous study on foveated matching algorithm demonstrates that it is an efficient algorithm and the matching accuracy is higher for the area that is close to the fovea (camera focal point). However, for the boundary area, the matching accuracy would not be promising. So if we can perfectly select the fovea, which always focus on the view of interest, we will not suffer from boundary area's low accuracy. Previous work [Boyling, 2002] attempted to tune the camera configurations dynamically, but this process takes non-neglectable time. In order to make our system efficient, this dynamic vergence process need to be accomplished in short time, which is quite challenging and requires additional research.

9.2.4 Efficient Visual Features for Learning

Our learning framework to tradeoff accuracy for efficiency is still a proof-of-concept research. Ultimately, experiments should be conducted in order to mine visual features that

can be automatically and efficiently extracted [Vedaldi and Fulkerson, 2010] based on the low resolution images or depth map. Those set of features can be visual features such as HoG (Histogram of oriented Gradients) [Dalal and Triggs, 2005] or even deep learning based CNN visual features [Donahue et al., 2013] that can effectively reflect the wrinkle properties. However, investigations on what visual features can reflect those wrinkle properties effectively and what set of visual features can be efficiently extracted requires further research.

Bibliography

Gene Amdahl. Validity of the single processor approach to achieving large-scale computing capabilities. In *AFIPS Conference Proceedings*, number 30, pages 483–485, 1967.

Gerardo Aragon-Camarasa, Susanne B Oehler, Yuan Liu, Li Sun, Paul Cockshott, and J Paul Siebert. Glasgow's stereo image database of garments. *arXiv preprint arXiv:1311.7295*, 2013.

Oliver Jakob Arndt, Daniel Becker, Christian Banz, and Holger Blume. Parallel implementation of real-time semi-global matching on embedded multi-core architectures. In *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII), 2013 International Conference on*, pages 56–63. IEEE, 2013.

Stephen T Barnard and Martin A Fischler. Computational stereo. *ACM Computing Surveys (CSUR)*, 14(4):553–572, 1982.

John L Barron, David J Fleet, and Steven S Beauchemin. Performance of optical flow techniques. *International journal of computer vision*, 12(1):43–77, 1994.

Luiz André Barroso, Kourosh Gharachorloo, Robert McNamara, Andreas Nowatzyk, Shaz Qadeer, Barton Sano, Scott Smith, Robert Stets, and Ben Verghese. Piranha: a scalable architecture based on single-chip multiprocessing. In *ISCA*, pages 282–293, 2000.

Alexandre Bernardino and José Santos-Victor. A binocular stereo algorithm for log-polar foveated systems. In *Biologically Motivated Computer Vision*, pages 127–136. Springer, 2002.

Christian Bersch, Benjamin Pitzer, and Sören Kammel. Bimanual robotic cloth manipulation for laundry folding. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 1413–1419. IEEE, 2011.

Stan Birchfield and Carlo Tomasi. Multiway cut for stereo and motion with slanted surfaces. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, pages 489–495. IEEE, 1999.

Aaron F Bobick and Stephen S Intille. Large occlusion stereo. *International Journal of Computer Vision*, 33(3):181–200, 1999.

Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(11): 1222–1239, 2001.

Timothy Boyling and J Siebert. A fast foveated stereo matcher. In *Proc. Conf. on Imaging Science Systems and Technology*, pages 417–423, 2000.

Timothy A Boyling. *Active vision for autonomous 3d scene reconstruction*. PhD thesis, University of Glasgow, 2002.

Paul Cockshott, Susanne Oehler, and Tian Xu. Developing a compiler for the xeonphi. In *Technical Report*. University of Glasgow, 2014.

W. Paul Cockshott, Susanne Oehler, Tian Xu, J. Paul Siebert, and Gerardo Aragon-Camarasa. Parallel stereo vision algorithm. In *Many-Core Applications Research Community Symposium 2012*, pages 45–50, 2012.

W.Paul Cockshott, editor. *SIMD Programming Manual for Linux and Windows*. Springer, London, 2004.

W.Paul Cockshott, editor. *Glasgow Pascal Compiler with vector extensions*. Glasgow, 2011.

Ingemar J Cox, Sunita L Hingorani, Satish B Rao, and Bruce M Maggs. A maximum likelihood stereo algorithm. *Computer vision and image understanding*, 63(3):542–567, 1996.

Marco Cusumano-Towner, Arjun Singh, Stephen Miller, James F O'Brien, and Pieter Abbeel. Bringing clothing into desired configurations with limited perception. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3893–3900. IEEE, 2011.

Mark R Cutkosky. *Robotic grasping and fine manipulation*, volume 6. Springer Science & Business Media, 2012.

Boguslaw Cyganek and J.Paul Siebert, editors. *An Introduction to 3D Computer Vision Techniques and Algorithms*. Wiley, 2009.

Mayank Daga, Thomas Scogland, and Wu chun Feng. Architecture-aware mapping and optimization on a 1600-core gpu. In *ICPADS*, pages 316–323, 2011.

Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *arXiv preprint arXiv:1310.1531*, 2013.

Alexandros Doumanoglou, Andreas Kargakos, Tae-Kyun Kim, and Sotiris Malassiotis. Autonomous active recognition and unfolding of clothes using random decision forests and probabilistic planning. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 987–993. IEEE, 2014a.

Andreas Doumanoglou, Tae-Kyun Kim, Xiaowei Zhao, and Sotiris Malassiotis. Active random forests: An application to autonomous unfolding of clothes. In *Computer Vision–ECCV 2014*, pages 644–658. Springer, 2014b.

Thomas Eiter and Heikki Mannila. Computing discrete fréchet distance. Technical report, Citeseer, 1994.

Olivier Faugeras, Thierry Viéville, Eric Theron, Jean Vuillemin, Bernard Hotz, Zhengyou Zhang, Laurent Moll, Patrice Bertin, Herve Mathieu, Pascal Fua, et al. Real-time correlation-based stereo: algorithm, implementations and applications. 1993.

Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient belief propagation for early vision. *International journal of computer vision*, 70(1):41–54, 2006.

James Fung and Steve Mann. Using graphics devices in reverse: Gpu-based image processing and computer vision. In *ICME*, pages 9–12, 2008.

Andrea Fusiello, Vito Roberto, and Emanuele Trucco. Efficient stereo with multiple windowing. In *cvpr*, page 858. IEEE, 1997.

Davide Gadioli, Simone Libutti, Giuseppe Massari, Edoardo Paone, Michele Scandale, Patrick Bellasi, Gianluca Palermo, Vittorio Zaccaria, Giovanni Agosta, William Fornaciari, et al. Opencl application auto-tuning and run-time resource management for multicore platforms. In *Parallel and Distributed Processing with Applications (ISPA), 2014 IEEE International Symposium on*, pages 127–133. IEEE, 2014.

Stefan K Gehrig and Clemens Rabe. Real-time semi-global matching on the cpu. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, pages 85–92. IEEE, 2010.

Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3354–3361. IEEE, 2012.

Donald B Gennery. Modelling the environment of an exploring vehicle by means of stereo vision. Technical report, DTIC Document, 1980.

Michael Goesele, Noah Snavely, Brian Curless, Hugues Hoppe, and Steven M Seitz. Multi-view stereo for community photo collections. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

G Gordon, Trevor Darrell, Michael Harville, and John Woodfill. Background estimation and removal based on range and color. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 2. IEEE, 1999.

Mark Harris. How to optimize data transfers in cuda c/c++. *Technical report*, 2012.

Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

Richard I Hartley and Peter Sturm. Triangulation. *Computer vision and image understanding*, 68(2):146–157, 1997.

Kaiming He, Jian Sun, and Xiaoou Tang. Guided image filtering. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(6):1397–1409, 2013.

Stephan Hengstler, Daniel Prashanth, Sufen Fong, and Hamid Aghajan. Mesheye: a hybrid-resolution smart camera mote for applications in distributed intelligent surveillance. In *Proceedings of the 6th international conference on Information processing in sensor networks*, pages 360–369. ACM, 2007.

Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments. In *In the 12th International Symposium on Experimental Robotics (ISER*. Citeseer, 2010.

H Hirschmuller and Stefan Gehrig. Stereo matching in the presence of sub-pixel calibration errors. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 437–444. IEEE, 2009.

Heiko Hirschmüller and Daniel Scharstein. Evaluation of cost functions for stereo matching. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

Heiko Hirschmüller and Daniel Scharstein. Evaluation of stereo matching costs on images with radiometric differences. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(9):1582–1599, 2009.

David W Hosmer Jr and Stanley Lemeshow. *Applied logistic regression*. John Wiley & Sons, 2004.

Martin Humenberger, Tobias Engelke, and Wilfried Kubinger. A census-based stereo vision algorithm using modified semi-global matching and plane fitting to improve matching quality. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, pages 77–84. IEEE, 2010.

Honghoon Jang, Anjin Park, and Keechul Jung. Neural network implementation using cuda and openmp. In *Digital Image Computing: Techniques and Applications (DICTA), 2008*, pages 155–161. IEEE, 2008.

Takeo Kanade and Masatoshi Okutomi. A stereo matching algorithm with an adaptive window: Theory and experiment. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16(9):920–932, 1994.

RE Kelly, PRH McConnell, and SJ Mildenberger. The gestalt photomapping system. *Photogrammetric Engineering and Remote Sensing*, 43:1407–1417, 1977.

Andreas Klaus, Mario Sormann, and Konrad Karner. Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 3, pages 15–18. IEEE, 2006.

Vladimir Kolmogorov and Ramin Zabih. Computing visual correspondence with occlusions using graph cuts. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 2, pages 508–515. IEEE, 2001.

Danica Kragic, Mårten Björkman, Henrik I Christensen, and Jan-Olof Eklundh. Vision for robotic object manipulation in domestic settings. *Robotics and autonomous Systems*, 52 (1):85–100, 2005.

Lubor Ladickỳ, Paul Sturgess, Chris Russell, Sunando Sengupta, Yalin Bastanlar, William Clocksin, and Philip HS Torr. Joint optimization for object class segmentation and dense stereo reconstruction. *International Journal of Computer Vision*, 100(2):122–133, 2012.

Charles Loop and Zhengyou Zhang. Computing rectifying homographies for stereo vision. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 1. IEEE, 1999.

Jeremy Maitin-Shepard, Marco Cusumano-Towner, Jinna Lei, and Pieter Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2308–2315. IEEE, 2010.

Davide Maltoni, Dario Maio, Anil K Jain, and Salil Prabhakar. *Handbook of fingerprint recognition*. Springer Science & Business Media, 2009.

Yoshio Matsumoto and Alexander Zelinsky. An algorithm for real-time stereo vision implementation of head pose and gaze direction measurement. In *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*, pages 499–504. IEEE, 2000.

Stefano Mattoccia. Fast locally consistent dense stereo on multicore. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, pages 69–76. IEEE, 2010.

Xing Mei, Xun Sun, Mingcai Zhou, Shaohui Jiao, Haitao Wang, and Xiaopeng Zhang. On building an accurate stereo matching system on graphics hardware. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 467–474. IEEE, 2011.

Sushmita Mitra and Tinku Acharya. Gesture recognition: A survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 37(3):311–324, 2007.

Thomas B Moeslund, Adrian Hilton, and Volker Krüger. A survey of advances in vision-based human motion capture and analysis. *Computer vision and image understanding*, 104(2):90–126, 2006.

Shunji Mori, Hirobumi Nishida, and Hiromitsu Yamada. *Optical character recognition*. John Wiley & Sons, Inc., 1999.

Don Murray and James J Little. Using real-time stereo vision for mobile robot navigation. *Autonomous Robots*, 8(2):161–171, 2000.

Yuichi Ohta and Takeo Kanade. Stereo by intra-and inter-scanline search using dynamic programming. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (2): 139–154, 1985.

David Pfeiffer, Stefan Gehrig, and Nicols Schneider. Exploiting the power of stereo confidences. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 297–304. IEEE, 2013.

Lynn H Quam and Artificial Intelligence Center. Hierarchical warp stereo. *Readings in computer vision*, pages 80–86, 1984.

Arnau Ramisa, Guillem Alenya, Francesc Moreno-Noguer, and Carme Torras. Using depth and appearance features for informed robot grasping of highly wrinkled clothes. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1703–1708. IEEE, 2012.

Arnau Ramisa Ayats, Guillem Alenyà Ribas, Francesc Moreno-Noguer, Carme Torras, et al. Determining where to grasp cloth using depth information. 2011.

Christoph Rhemann, Asmaa Hosni, Michael Bleyer, Carsten Rother, and Margrit Gelautz. Fast cost-volume filtering for visual correspondence and beyond. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3017–3024. IEEE, 2011.

Ashutosh Saxena, Jamie Schulte, and Andrew Y Ng. Depth estimation using monocular and stereo cues. In *IJCAI*, volume 7, 2007.

Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1-3): 7–42, 2002.

Daniel Scharstein, Heiko Hirschmüller, York Kitajima, Greg Krathwohl, Nera Nešić, Xi Wang, and Porter Westling. High-resolution stereo datasets with subpixel-accurate ground truth. In *Pattern Recognition*, pages 31–42. Springer, 2014.

JP Siebert and CW Urquhart. C3d^{TM}: a novel vision-based 3-d data acquisition system. In *Image Processing for Broadcast and Video Production*, pages 170–180. Springer, 1995.

Robert Spangenberg, Tobias Langner, Sven Adfeldt, and Renan Rojas. Large scale semi-global matching on the cpu. In *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*, pages 195–201. IEEE, 2014.

Jian Sun, Nan-Ning Zheng, and Heung-Yeung Shum. Stereo matching using belief propagation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(7):787–800, 2003.

Li Sun, Gerardo Aragon-Camarasa, Simon Rogers, and J.Paul Siebert. Accurate garment surface analysis using an active stereo robot head with application to dual-arm flattening. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 185–192, May 2015. doi: 10.1109/ICRA.2015.7138998.

Li Sun, Gerardo Aragon-Camarasa, Bing Shuai, Simon Rogers, and J.Paul Siebert. Clothing category recognition in free-configurations with application to autonomous clothes sorting (under submission). In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, 2016.

Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

Federico Tombari, Stefano Mattoccia, Luigi Di Stefano, and Elisa Addimanda. Classification and evaluation of cost aggregation methods for stereo correspondence. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

Federico Tombari, Stefano Mattoccia, and Luigi Di Stefano. Stereo for robots: quantitative evaluation of efficient and low-memory dense stereo algorithms. In *Control Automation Robotics & Vision (ICARCV), 2010 11th International Conference on*, pages 1231–1238. IEEE, 2010.

Roger Y Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *Robotics and Automation, IEEE Journal of*, 3(4):323–344, 1987.

Jur Van Den Berg, Stephen Miller, Ken Goldberg, and Pieter Abbeel. Gravity-based robotic cloth folding. In *Algorithmic Foundations of Robotics IX*, pages 409–424. Springer, 2011.

Arthur A. van Hoff. Efficient computation of gaussian pyramids. *Technical report*, 1992.

Arthur A. van Hoff. An efficient implementation of mssm. *Technical report*, 1993.

Andrea Vedaldi and Brian Fulkerson. Vlfeat: An open and portable library of computer vision algorithms. In *Proceedings of the international conference on Multimedia*, pages 1469–1472. ACM, 2010.

V Venkateswar and Rama Chellappa. Hierarchical stereo and motion correspondence using feature groupings. *International Journal of Computer Vision*, 15(3):245–269, 1995.

Vasily Volkov and James W Demmel. Benchmarking gpus to tune dense linear algebra. In *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, pages 1–11. IEEE, 2008.

Liang Wang, Miao Liao, Minglun Gong, Ruigang Yang, and David Nister. High-quality real-time stereo using adaptive cost aggregation and dynamic programming. In *3D Data Processing, Visualization, and Transmission, Third International Symposium on*, pages 798–805. IEEE, 2006.

Bryan Willimon, Stan Birchfield, and Ian D Walker. Model for unfolding laundry using interactive perception. In *IROS*, pages 4871–4876, 2011a.

Bryan Willimon, S Birchfield, and Ian Walker. Classification of clothing using interactive perception. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1862–1868. IEEE, 2011b.

Andrew Witkin, Demetri Terzopoulos, and Michael Kass. Signal matching through scale space. *International Journal of Computer Vision*, 1(2):133–144, 1987.

Steve Wright. Parallel vs converged. *Technical report*, 2011.

Tian Xu and Paul Cockshott. Guided filtering based pyramidical stereo matching for unrectified images. In *Image and Vision Computing New Zealand, International Conference, IVCNZ 2015, Auckland, New Zealand*, 2015.

Tian Xu, Paul Cockshott, and Susanne Oehler. Acceleration of stereo-matching on multicore cpu and gpu. In *High Performance Computing and Communications, 2014 IEEE Intl Conf on*, pages 108–115. IEEE, 2014.

Ruigang Yang and Marc Pollefeys. Multi-resolution real-time stereo on commodity graphics hardware. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1, pages I–211. IEEE, 2003.

Ruigang Yang, Greg Welch, and Gary Bishop. Real-time consensus-based scene reconstruction using commodity graphics hardware? In *Computer Graphics Forum*, volume 22, pages 207–216. Wiley Online Library, 2003.

Kuk-Jin Yoon and In So Kweon. Adaptive support-weight approach for correspondence search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):650–656, 2006.

Ramin Zabih and John Woodfill. Non-parametric local transforms for computing visual correspondence. In *Computer Vision ECCV'94*, pages 151–158. Springer, 1994.

Jinglin Zhang, J-F Nezan, and J-G Cousin. Implementation of motion estimation based on heterogeneous parallel computing system with opencl. In *High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, pages 41–45. IEEE, 2012.

Qi Zhang, Yurong Chen, Yimin Zhang, and Yinlong Xu. Sift implementation and optimization for multi-core systems. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–8. IEEE, 2008.

Yao Zhang and John D Owens. A quantitative performance analysis model for gpu architectures. In *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, pages 382–393. IEEE, 2011.

Zhengyou Zhang. A flexible new technique for camera calibration. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(11):1330–1334, 2000.

Wenyi Zhao, Rama Chellappa, P Jonathon Phillips, and Azriel Rosenfeld. Face recognition: A literature survey. *ACM computing surveys (CSUR)*, 35(4):399–458, 2003.

Yong Zhao and Gabriel Taubin. Real-time stereo on gpgpu using progressive multi-resolution adaptive windows. *Image and Vision Computing*, 29(6):420–432, 2011.