Real-time Stereo Vision Framework for Mobile Device Using GPGPU

안드레 이반, 김태인, 박인규
인하대학교 정보통신공학과
andreivan13@gmail.com, sappho192@gmail.com, pik@inha.ac.kr

Abstract

This paper presents a fast stereo matching framework running on widely available mobile device using GPU. Our approach consists of three main steps: pre-processing, stereo matching, and pre-processing where each step employs multiple algorithms. This multiple approach enables us to analyze and choose the best approach for tradeoff between speed and accuracy. Experimental results show that the system able to run in real-time yet generating a reasonable disparity map through some optimization technique, quantitative evaluation is also performed to check the system accuracy.

1. Introduction

Stereo matching or stereo correspondences has been a continuously explored topic in the computer vision field. However, there are not many stereo matching works focusing on mobile device due to computational limitation on mobile device. Naturally stereo matching algorithm consists of algorithm with high computation cost, coming from number of loops involved while searching for correspondences.

With general-purpose computing on graphics processing units (GPGPU) we can overcome the limitation. While not all algorithm can be implemented in GPU, stereo matching algorithms are mostly suitable for GPU implementation. Exploiting this we can solve the performance issue on a resource constrained device (mobile). Even though GPGPU can solve the computation problem, implementing the algorithm on mobile GPU is not an easy feat, as it consists of many constraints and limitation compared to desktop environment. We utilize OpenCL library as the parallelization library. OpenCL is a widely known heterogenous and cross-platform parallel programming library suitable for mobile device GPU.

Thus, we present a fast stereo matching system and optimization methods for real-time processing with reasonable disparity map on widely available mobile device. The accuracy of proposed framework is evaluated using dataset from [1].

2. Proposed method

The proposed framework consists of three main steps. At the beginning gaussian and gradient filter is performed alongside calibration and rectification on pre-processing stage. For matching cost computation sum of absolute difference (SAD) or adaptive support weight (ASW) [2] is used. Note that cost aggregation is not included to reduce the computation time. Final disparity map is obtained by performing winner-take-all (WTA), the whole process is considered as stereo matching stage. Finally weighted median filter (WMF) [3] is performed to improve the disparity map as post-processing stage. Whole system pipeline is shown in Fig. 1. Each step is optimized and described in the following subsections.

2.1 Pre-processing

First Gaussian filter is performed to remove existing noise captured in the real-world environment. Note that we use small σ to make sure the image did not become ambiguous for stereo correspondence search yet enough to remove some noise. Gradient filter is then performed to deal with ambiguity from intensity difference between left and right camera. Gradient filter can also deal with texture less or homogenous region. Calibration and rectification is needed since we are dealing with real stereo camera. Both process is done at the same time to save computation time. We also compare the gradient filter approach with normalized cross correlation (NCC) matching in handling different intensity view, we found that gradient filter able to handle the intensity difference better and with less computation time.

2.2 Stereo matching

Two approaches are considered for matching cost computation: SAD, ASW. Problem in SAD mostly relies on nested loops for image width, height, and disparity search range, while the computation complexity for each loop is quite low as only absolute difference is performed. ASW also has the same problem but with more computation complexity as it deals with exponential calculation. On GPU implementation these nested loops can be omitted and performed in parallel. SAD performs faster compared to ASW but ASW produces better disparity map. Hence, SAD is more suitable matching cost algorithm for real-time system, even though our ASW implementation also run in real-time. Computation time for each method is discussed

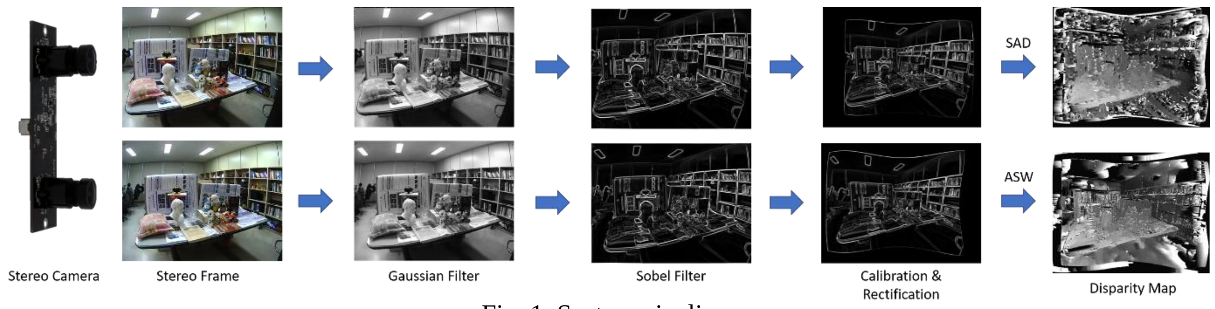

Fig. 1. System pipeline

on section 3. Cost volume filtering or WTA is performed to find each pixel disparity value associated with its minimum cost. Stereo matching part is the bottleneck of our proposed system; therefore, we optimize those steps. More detailed optimization method employed is discussed on section 2.4.

2.3 Post-processing

WMF is performed to further refine the estimated disparity map. Unweighted median filter or just median filter is known to remove salt and pepper noise which commonly found in disparity map. In refining disparity map just removing noise is not enough, we want to fill some holes from error in matching cost computation. WMF is a great approach in smoothing image while preserving edges which is important in stereo matching. Drawback of WMF is the computation times since its basically performing cost aggregation on disparity image.

We implement WMF on GPU using bilateral filter as the weight. Since WMF is not feasible for real-time computation we employ WMF to achieve high accuracy disparity map. Bilateral filter is used due to its simplicity in implementing it on GPU and it preserves edges better than guided or domain transform filter [3].

2.4 Optimization

Since we are dealing with stereo images number of search in width and height doubles than normal, in GPU this translates to amount of global work size. This related to the amount of points in an array of work dimension that must be executed. More work size means more pixel to work with, this can be solved by stacking or computing both left and right image parallelly. Since on pre-processing stage left and right image is independent, this approach is efficient as we can perform pre-processing on both stereo image in one go. We also ignore boundary pixel to removes insufficient computation and (IF) branching.

Utilizing efficient memory access by reducing the amount of global memory access as much as possible. By using local memory, we can reduce computation time as it is known to be much faster than global memory access. However, size of memory available inside GPU is limited. Thus, we need to design the usage of this variable well. Global memory access mostly happened when we try to access pixel value of an image passed as global variable. Store frequently accessed pixels in local memory to avoid repeated global memory access.

It is important to find suitable local work size or thread block size in GPGPU programming. To find the best size, we evaluate the computational time for each candidate and choose the fastest one empirically. Note that local work size has the most affect in reducing computation time. We found that 16x16 and 8x8x4 local work size for 2D and 3D global work size respectively are the best for fastest computation time. 2D global work size is for case involving width and height only while 3D includes disparity range. The whole optimization approach improves our computation time from 7.5 FPS to 10.44 FPS.

3. Experimental result

The proposed framework is implemented on Samsung S7 with Octa-core (4x2.3 GHz & 4x1.6 GHz) CPU and Mali-T880 MP12 GPU on Exynos 8890 Octa chipset. We use OpenCL ver 1.2. and code is implemented in Android Studio. Fig. 2 shows the result of our framework using real stereo image captured from USB stereo camera and image from [1]. We suggest gradient filter, SAD, and WTA for real-time processing purposes and gradient filter, ASW, WTA, WMF for high accuracy purposes (3.54% bad pixel).

Table 1 shows the computational time comparison between CPU implementation and GPU implementation. Even with considering the OpenCL memory allocation between GPU and host, it still performs faster than CPU implementation.

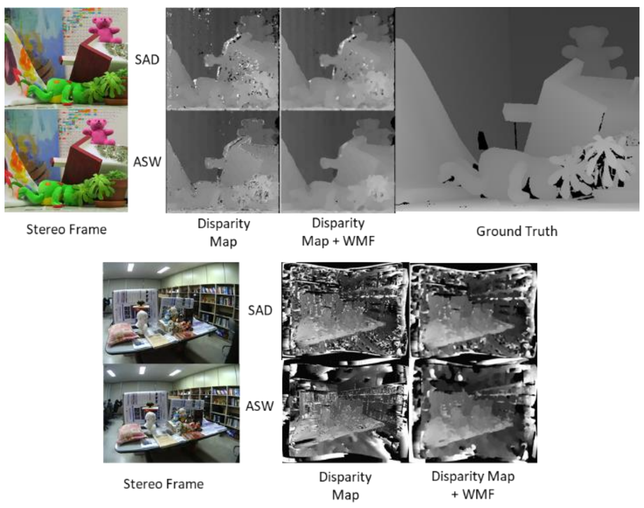

Fig. 2. Results of proposed framework

	CPU	GPU
Host→GPU	0	0.000525
Gradient filter	0.14684	0.016442
Rectification	0.09854	0.003358
Stereo matching	5.43070	0.085134
GPU→Host	0	0.004366
Total time	5.67608	0.109825

Table 1. Comparison of computational time in seconds

4. Conclusion

We introduced a stereo matching framework on mobile GPU to achieve real-time computation yet producing reasonable disparity map. Our framework works with real stereo camera and stereo image. GPU optimization were done throughout the whole framework. Experimental results show that the proposed framework can obtain faster speed and produces dense disparity map. Our framework can also be used for multiple purposes for speed or accuracy.

Acknowledgement

이연구는정보통신기술진흥센터(IITP)와과학기술정보통신부(MSIT)의지원으로수행되었습니다(2017-0-00142).

References

[1] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *Int'l J. Computer Vision*, 2002.
[2] K. J. Yoon and I. S. Kweon, "Adaptive support-weight approach for correspondence search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2006.
[3] Z. Ma, K. He, Y. Wei, J. Sun, E. Wu, "Constant Time Weighted Median Filtering for Stereo Matching and Beyond," *IEEE International Conference on Computer Vision*, 2013.