Journal of Engineering and Applied Sciences 11 (7): 1480-1487, 2016
ISSN: 1816-949X

Stereo Matching Performance Analysis of Cost Functions on the Graphic Processing Unit (GPU) for Pervasive Computing

[1]Gwang-Soo Hong, [1]Woong Hoe, [1]Byung-Gyu Kim, [2]Jang-Woon Beak and [2]Kee-Koo Kwon
[1]Department of Computer Engineering, SunMoon University, Asan, Republic of Korea
[2]Automotive IT Platform Section, ETRI, Daegu, Republic of Korea

Abstract: Stereo imaging is a powerful technique fordetermining the distance to objects using a pairs of cameraspaced apart. The extremely high computational requirements ofstereo vision limit application to non realtime applications wherehigh computing power is available. To overcome the limitation, we utilized the general strategy for parallelization of dense cost functions on Compute Unified Device Architecture (CUDA) with Graphic Processing Unit (GPU), especially for pervasive environment. The challenges of mapping a sequential stereo matching algorithm to a massively parallel thread environment are considered. Compared to the CPU counterpart, the processing speed of the stereo matching algorithm based on CUDA programming can be improved by about from 107-369 times.

Key words: Stereo vision, CUDA, GPU, cost function, stereo matching, pervasive computing

INTRODUCTION

Smart object and its interaction with our daily lives funnel a phenomenal amount of data around well-developed pervasive environments. Data from sensors (environmental sensors such as motion sensors; smart phone sensors such as accelerometers and GPS; and object sensors such as RFID tags) require a careful analysis to extract interesting and relevant information and the most important of all is how the extracted information can help develop well-being in human society. The sheer volume of sensor data, as well as its streaming and distributed nature, poses many challenges to the data analysis, mobile sensing and knowledge discovery community. Analyzing these data or big data, trails can support different applications in a novel way.

In term of human perception, it is attempt to realize in a digital way technology when it possible to update map during stereoscopic fusion of this map with newer aerial image. These kinds of processes were state of the art at times when human perception was deployed in research and production environments. Described here approach can be termed as "technology fusion" encompassing photogrammetry, cognitive neuroscience and computer vision.

The human visual system calculates the relative depth of an object with respect to the object that the eyes are fixated on. The perception of three-dimensional depth from two disparate retinal projections is called human-centric stereomatching (Zarnowski *et al*., 2010).

Human-centric stereo matching in stereo vision has been widely studied a topic of computer vision with a lot of surveyed in (Yoon and Kweon, 2005). State of art algorithms as evaluated in the stereo benchmark (Middlebury) have improved the matching cost accuracy.

Zhang *et al*. (2006) compute simultaneously the disparity image and an illumination ratio map in a BP framework for handling complex local intensity variations. (Gautama *et al*., 1999) compared Zero mean Normalized Cross-Correlation (ZNCC) and Census for car-seat occupancy detection using window-based stereo vision. For their application, (Banks and Corke, 2001) compared Sum of Absolute Intensity Differences (SAD), Sum of Squared intensity Differences (SSD), Normalized Cross-Correlation (NCC) their zero mean variants, Rank and Census for window-based stereo matching. The evaluation includes visual inspection and the count of pixels that passed the left/right consistency check on images. Fookes *et al*. (2004) compared SAD, Zero mean Sum of Absolute Differences (ZSAD), NCC, ZNCC, Rank and MI for window-based stereo matching. Their evaluation also measures the number of pixels that passes the validity check. They concluded that ZNCC performs best on images without radiometric changes.

But, these algorithms are generally computational complicated. To obtain fast speed, stereo on Graphic Processing Units (GPUs) is an attractive trend, as the successful exemplar of computer vision on GPUs (OpenNVIDIA). GPUs which can utilize the horsepower of massive parallel processors are effective to accelerate

Corresponding Author: Gwang-Soo Hong, Department of Computer Engineering, SunMoon University, Asan, Republic of Korea

stereo algorithms by exploiting their parallelism. Several recent methods have reached fast speed on GPUs while maintaining matching quality (Yang *et al*., 2006). In addition, these methods typically avoid image segmentation, which requires a large number of computationally demanding iterations and plane-fitting, which lacks the computational regularity necessary for parallelization (Kalus *et al*., 2006; Bleyer *et al*., 2010).

large memory consumption could impede its prevalence in matching high-resolution images with a large disparity range (Lu *et al*., 2009) obtained fast speed on a GPU using a cross-based local approach (Zhang *et al*., 2009). Excluding the refinement stage of Zhang *et al*. (2009) and Lu *et al*. (2009) cannot handle occluded regions and large homogeneous regions accurately.

In this study, the cost functions and computationally efficient window-based cost aggregation and a low-complexity optimization are introduced. The scope of this paper is the evaluation and comparison of some wieldy used stereo matching costs on images on CUDA.

MATERIALS AND METHODS

Background: The basic structure of a stereo vision application is shown in Fig. 1. Two cameras are spaced apart by the baseline distance B. Each camera images the object but from a slightly different angle. The distance to the object can then be computed by:

$$D = \frac{Bf}{d} \quad (1)$$

Where:

D = The distance to the object
B = The baselinedistance between the stereo images, is the focal length ofthe camera
D = The disparity
D = The difference inlocation of the image of the object between the left and right images
D = Finding by determining the correspondence between pixels in the stereo pair is the primary problem to be solved by this application

Researach computed in units of pixels which can be converted to distance units by multiplying by the pixel size. Also, this simple formula assumes idealized optics.

In order to find corresponding pixels in target and reference image, there is obviously need for pixel similarity measure. It is used the term as dissimilarity measure or matching cost which increases as the similarity between two compared pixels decreases. Matching cost is a function of reference image coordinates and disparity.

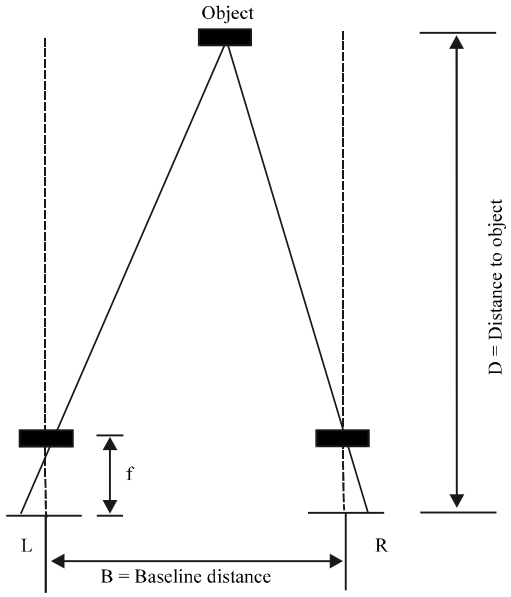

Fig. 1: The basic structure of a stereo vision

The cost function returns the value of dissimilarity for a coordinate in the disparity space. The disparity space is made by the available image pixel coordinates and disparity search range. Generally, disparity search range has to be set manually and 0depends on the characteristics of the input image pair. The cost is referred to as Disparity Space Image (DSI). This is due to the fact that for each fixed integer disparity value, the function represents an image visualizing the cost for every pixel location.

Common measures that are used for pixel-wise comparison are Absolute intensity Difference (AD) (Kanade, 1994) and Squared intensity Difference (SD). (Anandan, 1989). The cost functions for pixel-wise comparison are written as:

$$C_{AD}(p,d) = \left|I_L(p) - I_R(p-d)\right| \quad (2)$$

Where:

$I_L(p)$ = The intensity of all pixels p in left image
$I_R(p\text{-}d)$ = The intensity of corresponding pixel p-d calculated using disparity component d, respectively

Also, the square difference value can be defined as the following:

$$C_{SD}(p,d) = \left|I_L(p) - I_R(p-d)\right|^2 \quad (3)$$

By extending the comparison to square window regions centered about the search and reference pixels,

these measures are turned into SAD, SSD and Normalized Sum of Squared Intensity Differences (NSSD). The SAD, SSD and NSSD cost functions based for square comparison are defined as:

$$C_{SAD}(p,d)=\sum_{q\in Nq}(|I_L(q)-I_R(q-d)|) \quad (4)$$

where, all pixels of certain neighborhood are calculated over disparities between left and right image. Also SSD and NSSD can be written as:

$$C_{SSD}(p,d)=\sum_{q\in Nq}\left[|I_L(q)-I_R(q-d)|\right]^2 \quad (5)$$

$$C_{NSSD}(p,d)=\frac{\sum_{q\in Nq}\left[|I_L(q)-I_R(q-d)|\right]^2}{\sqrt{\sum_{q\in Np}I_L(q)^2.\sum_{q\in N_p}I_R(q-d)^2}} \quad (6)$$

Next following cost measurements relies on calculating at each position of the image under examination a correlation or distortion function that measures the degree of similarity or dissimilarity to a template sub image. Among the correlation/distortion functions proposed in literature, NCC (Hirschmuller *et al*., 2002) and ZNCC (Krattenthaler *et al*., 1994) are widely used due to their robustness in template matching. In fact, the normalization embodied into the NCC and ZNCC allows for tolerating linear brightness variations. The NCC and ZNCC cost functions for square comparison are written as:

$$C_{Ncc}(p,d)=\frac{\sum_{q\in Nq}I_L(q).I_R(q-d)}{\sqrt{\sum_{q\in Np}I_L(q)^2\cdot\sum_{q\in Np}I_R(q-d)^2}} \quad (7)$$

$$C_{ZNCC}(p,d)=\frac{\sum_{q\in Nq}\left(I_L(q)-\overline{I}_L(p)\right)\left(I_R(q-d)-I_R(p-d)\right)}{\sqrt{\sum_{q\in Np}\left(I_L(q)-I_L(p)\right)^2\sum_{q\in Np}\left(I_R(q-d)-\overline{I}_R(p-d)\right)^2}} \quad (8)$$

where, $\overline{I}_L(p)$ and $\overline{I}_R(p-d)$ denote the mean intensity value of and respectively. The focus of this study is on matching costs between sequential and parallel programming. This excludes popular methods like the correlation-based weighting according to proximity and color similarity (Yoon and Kweon, 2005) since this is an aggregation approach rather than a matching cost. Mentioned cost functions have been evaluated in term of the complexity.

Cuda parallel processing architecture: The CUDA computing engine visualizes graphics HARDWARE available to the programmer through the use of uniquely numbered threads that are organized from one-dimensional to three dimensional blocks of arbitrary size.

The thread can be thought of as a scalar arithmetic processor whereas a block of threads is an abstract representation of a multiprocessor composed of multiple scalar processors and capable of performing operations in parallel. The threads are executed on the graphics device equipped with a GPU, hereafter referred to as the device, serving as a coprocessor that enhances the computational capabilities of the workstation, referred to as the host.

The memory of the device is disjoint from the memory of the host, making it necessary to allocate and transfer blocks of data to the device prior to executing threads. In addition to off-chip random access memory, termed global memory, the device offers a limited amount of low-latency on-chip memory accessible to all threads within a block, referred to as shared memory. On-chip memory is also available in the form of registers which are only accessible to individual threads. The device code is encapsulated in special functions called kernels that are invoked by the host and executed in parallel by multiple threads. At runtime, each block of threads gets mapped to a single multiprocessor on the device and the threads within the block are executed in groups of 32 called warps.

The execution follows the Single Instruction Multiple Thread (SIMT) model which guarantees parallel execution as long as the threads in a warp do not experience a divergence of coding due to branching instructions. To ensure peak performance, it is imperative to maximize the occupancy of the multiprocessors and to minimize the latency associated with global memory access by selecting the appropriate granularity of computations and the proper assignment of thread block dimensions.

The implementation of the cost functions utilizes the NVIDIA Tesla C2050 (GF100GL) computing processor, equipped with GPU cores. Tesla c2050 allows it to accelerate local and global memory references that exhibit spatial or temporal locality through the use of memory caching.

Figure 2 shows overall procedure of parallelization and computing flow of kernels on CUDA. We exploit the intrinsic parallelism exposed by the GPU assigning to each thread on pixel of the reference image. For each pixel the thread will process the entire disparity range d. We split entire stereo matching algorithm in 3 main operations, each one associated with a kernel.

The first kernel, referred to as matching cost calculation, takes as input the reference and target images in order to calculate matching costs for each block.The second kernel, referred to as aggregation, takes as calculated matching costs between reference and target image are aggregated within each block of size Aggregation is done by summing matching cost over square window with constant disparity.The third kernel referred to as optimization, for each pixel of reference image, the cost of each possible correspondence with the pixels of the target within the disparity range .Once these operations are completed, the kernels searches for the candidate with the minimum cost within the disparity range.

Algorithm 1 (Matching cost calculation kernel):

1: procedure CostFuncKernel(input, output, C, arguments) ▹each thread do
2: tIdx = blockIdx.x * blockDim.x + threadIdx.x
3: tIdy = blockIdx.y * blockDim.y + threadIdx.y
4: for (x, y-w/2) to (x, y+w) do ▹ compute matching costs at the position of [tidx, tidy]
5: C(tIdx, tIdy) = costfunc(tIdx + x, tIdy +y)
6: end for
7: end procedure

Algorithm 2 (Aggregation kernel):

1: procedure AggregationKernel (input, output, AggrC, arguments) ▹each thread do
2: tIdx = blockIdx.x * blockDim.x + threadIdx.x
3: tIdy = blockIdx.y * blockDim.y + threadIdx.y
4: for (x, y-w/2) to (x, y+w) do ▹ aggregate matching costs
5: AggrC (tIdx, tIdy) = AggregateFunc(tIdx + x, tIdy +y,C(tIdx,tIdy))
6: end for
7: end procedure

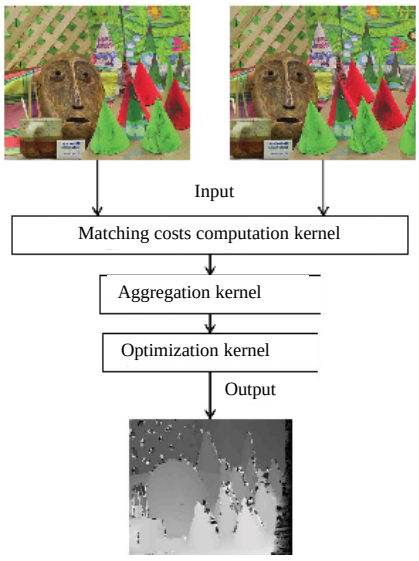

Fig. 2: Parallelization and computation flows of processing kernels on CUDA

Algorithm 3 (Optimization kernel):

1: procedure OptimizationKernel(input, output, AggrC, arguments) ▹each thread do
2: tIdx = blockIdx.x * blockDim.x + threadIdx.x
3: tIdy = blockIdx.y * blockDim.y + threadIdx.y
4: for d∈δ do ▹ select the index having minimum of cost
5: d = Opt(tIdx, tIdy, d, Aggrc(tIdx,tIdy))
6: end for
7: end procedure

The Pseudo-code in algorithms 1, 2 and 3 describes the operations executed by the matching cost calculation, aggregation and optimization kernels, respectively. In order to reduce the global memory latency, it is important to bind the texture memory, as well as the reference and target images. The intensity values should be stored in CUDA arrays. The image data is loaded into the extended window in a way that avoids bank conflicts, since no two threads in the block write to the same bank in shared memory. Although the device allows the maximum of 1024 threads within the block, corresponding to the block size of , due to limitations in the amount of shared memory available to each multiprocessor. This high multiprocessor occupancy makes it possible for the hardware to effectively hide the latency of memory access.

RESULTS AND DISCUSSION

This secretion will discuss the performance of all of the methods discussed so far and some of the fast implementations available in literature. We have used NVIDIA Tesla C2050 (GF100GL) on Intel quad core system for all these experiments.

We tested all combinations of all matching costs with the window-based aggregation and Winner-Take-All)

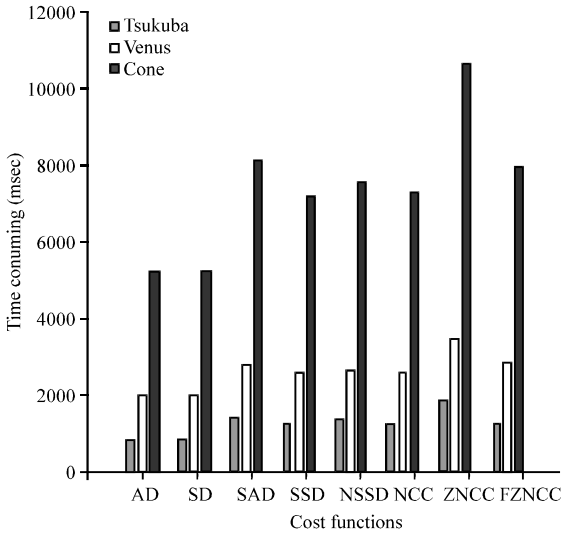

Fig. 3: Execution time of full matching computation in CPU programming

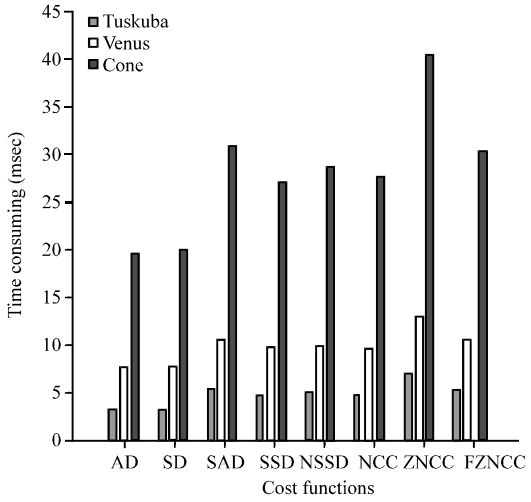

Fig. 4: Execution time of full matching computation in GPU programming

optimization on images to analyze computational complexity (Scharstein and Szeliski, 2002). And we have tested the speed of these methods using the Middlebury stereo database which have varying the maximum disparity range as shown Table 1.

Figure 3 and 4 show execution time of full matching computation in CPU and GPU, respectively. Matching costs are calculated one by one in sequential programming. Since time-consuming of each cost function is significantly different in CPU, it is similar in GPU regales of cost functions. As each block of threads gets mapped each operation of stereo matching, latency is minimized in GPU programming.

Table 1: Middlebury stereo database

Sequences	Image size	Disparities
Tsukuba	384×288	16
Venus	434×383	20
Cone	450×375	60
Teddy	450×375	60

Table 2: Execution times of CPU implementations of each operation in Stereo Matching Algorithm

Sequences	Cost functions (msec)							
(operation)	AD	SD	SAD	SSD	NSSD	NCC	ZNCC	FZNCC
Tsukuba								
Matching cost	20	18	570	740	1400	1300	1550	1160
Aggregation	470	470	460	460	470	470	470	470
Optimization	20	30	30	30	20	20	30	30
Allocation	1	1	1	1	1	1	1	1
Full matching	511	519	1061	1221	1881	1791	2051	1661
Venus								
Matching cost	40	100	1120	1200	2470	2220	2950	1960
Aggregation	710	700	700	700	700	700	700	700
Optimization	60	30	70	70	70	70	60	70
Allocation	3	3	3	3	3	3	3	3
Full matching	813	833	1893	1973	3243	2993	3723	2733
Cone								
Matching cost	210	190	3480	3690	7360	6740	9160	5700
Aggregation	2210	2210	2210	2210	2210	2210	2210	2210
Optimization	200	200	200	190	200	200	200	200
Allocation	5	5	5	5	5	5	5	5
Full matching	2625	2605	589	6095	9775	9155	11515	8115

Table 3: Execution times of GPU implementations of each operation in Stereo Matching Algorithm

Sequences	Cost functions (msec)							
(operation)	AD	SD	SAD	SSD	NSSD	NCC	ZNCC	FZNCC
Tsukuba								
Matching cost	0.17	0.18	2.43	1.67	1.98	1.77	4.13	2.20
Aggregation	1.41	1.41	1.39	1.41	1.41	1.40	1.40	1.40
Optimization	0.16	0.16	0.16	0.16	0.16	0.16	0.16	0.16
Allocation	1.51	1.52	1.49	1.58	1.54	1.52	1.45	1.59
Full matching	3.25	3.27	5.47	4.83	5.09	4.85	7.14	5.35
Aggregation	1.99	1.97	2.04	2.03	1.98	2.05	2.03	1.94
Venus								
Matching cost	0.55	0.54	3.56	2.54	2.90	2.67	6.07	3.32
Aggregation	1.99	1.97	2.04	2.03	1.98	2.0	2.03	1.94
Optimization	0.28	0.28	0.28	0.28	0.28	0.28	0.28	0.28
Allocation	4.76	4.89	4.71	4.97	4.88	4.71	4.72	4.97
Full matching	7.58	7.68	10.59	9.82	10.04	9.71	13.10	10.51
Cone								
Matchingcost	8.89	8.30	13.47	9.61	11.13	10.23	22.86	12.72
Aggregation	7.77	7.91	7.52	7.89	7.89	7.70	7.65	7.56
Optimization	1.32	1.35	1.38	1.31	1.33	1.35	1.35	1.33
Allocation	8.21	8.24	8.24	8.23	8.19	8.11	8.21	8.20
Full matching	19.59	19.80	30.61	27.04	28.54	27.39	40.07	29.81

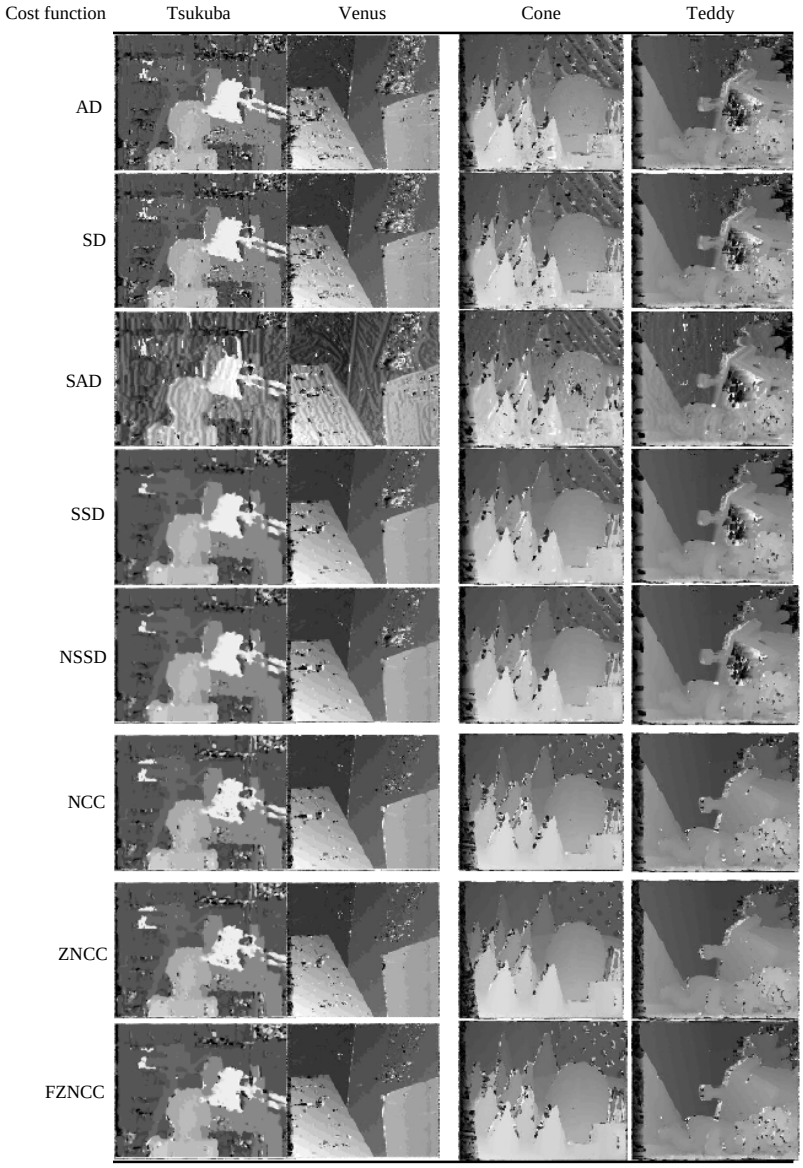

Fig. 5: The results of depth map V

Tables 2 and 3 give details of the performance comparison of different methods available in literature based on CPU and CUDA, respectively. Although, pixel- wise cost functions such as AD and SD are faster than other cost functions based on CPU, By comparing to existing real-time stereo matching methods, the proposed method is evaluated in terms of speed.

Computations of stereo matching based on CUDA coding is faster than based on CPU coding about from to. Each part of stereo matching algorithm on GPU shows the high computation complexity on sequential programming.However, since most time is consumed by memory allocation and memory copy from DRAM to CUDA memory, the results of time-consuming on CUDA coding are similar about from ms to ms. The part calculating matching cost is actually performed quickly by splitting the thread unit. The cost functions based on square window have high computational complexity in CPU coding. In the CUDA coding, the cost functions based on square window make no differences in term of speed.

Figure 5 shows the results of the computed depth map according to the cost functions. The cost functions

based on pixel-wise show the worse results than the cost functions based on square window in subjective point of view. ZNCC and NC show better results than the other cost functions in term of uniform region and depth discontinuities region. The cost functions implemented by CUDA give the real-time performance and prove to be faster than the state of art implementation based on CPU.

When the display is low, we have obtained very high performance. However, as the value of disparity goes on increasing the performance degrades. But, it still provides the enough performance for real-time applications, especially for pervasive environment.

CONCLUSION

We have developed the stereo matching algorithm on GPU platform. The performance comparison between CPU and CUDA coding in stereo matching in terms of computational complexity has been analyzed to provide the real-time stereo vision in the pervasive computing environment. We have tested our implementation on sample images from the Middlebury database. We have achieved very high speed-up factor when compared to the conventional sequential implementation.

Although, the part of memory allocation on the GPU was slower than the CPU programming, the other parts of stereo matching based on CUDA coding was much faster than based on CPU coding by amount of about from 107-369×. In the CUDA development, pixel-wise cost functions as well as square window cost functions make depth map rapidly.

Hence, the proposed approach can be applied for the real-time depth map generation based on the CUDA programming.

REFERENCES

Anandan, P., 1989. A computational framework and an algorithm for the measurement of visual motion. Int. J. Comput. Vision, 2: 283-310.

Banks, J. and P. Corke, 2001. Quantitative evaluation of matching methods and validity measures for stereo vision. Int. J. Rob. Res., 20: 512-532.

Bleyer, M., C. Rother and P. Kohli, 2010. Surface stereo with soft segmentation. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, June 13-18, 2010, San Francisco, CA., USA., pp: 1570-1577.

Fookes, C., A. Maeder, S. Sridharan and J. Cook, 2004. Multi-spectral stereo image matching using mutual information. Proceedings of the 2nd International Symposium on 3D Data Processing, Visualization and Transmission, September 6-9, 2004, Thessaloniki, Greece, pp: 961-968.

Gautama, S., S. Lacroix and M. Devy, 1999. Evaluation of stereo matching algorithms for occupant detection. Proceedings of the International Workshop on Recognition, Analysis and Tracking of Faces and Gestures in Real-Time Systems, September 26-27, 1999, Corfu, Greece, pp: 177-184.

Hirschmuller, H., P.R. Innocent and J. Garibaldi, 2002. Real-time correlation-based stereo vision with reduced border errors. Int. J. Comput. Vision, 47: 229-246.

Kalus, A., M. Sormann and K. Karner, 2006. Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure. Proceedings of the 18th International Conference on Pattern Recognition, Volume 3, August 20-24, 2006, Hong Kong, pp: 15-18.

Kanade, T., 1994. Development of a video-rate stereo machine. Proceedings of the ARPA Image Understanding Workshop, November 14-16, 1994, Monterrey, CA., USA., pp: 549-557.

Krattenthaler, W., K.J. Mayer and M. Zeiller, 1994. Point correlation: A reduced-cost template matching technique. Proceedings of the IEEE International Conference Image Processing, Volume 1, November 13-16, 1994, Austin, TX., USA., pp: 208-212.

Lu, J., K. Zhang, G. Lafruit and F. Catthoor, 2009. Real-time stereo matching: A cross-based local approach. Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, April 19-24, 2009, Taipei, Taiwan, pp: 733-736.

Scharstein, D. and R. Szeliski, 2002. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. Int. J. Comput. Vision, 47: 7-42.

Yang, Q., L. Wang, R. Yang, S. Wang, M. Liao and D. Nister, 2006. Real-time global stereo matching using hierarchical belief propagation. Proceedings of the British Machine Vision Conference, Volume 3, September 4-7, 2006, Edinburgh, UK., pp: 989-998.

Yoon, K.J. and I.S. Kweon, 2005. Locally adaptive support-weight approach for visual correspondence search. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Volume 2, June 20-25, 2005, San Diego, CA., USA., pp: 924-931.

Zarnowski, A., E. Levin and S. Skubiev, 2010. [Human centric approach to inhomogeneous geospatial data fusion and actualization]. Archiwum Fotogrametrii Kartografii Teledetekcji, 21: 523-533, (In Polish).

Zhang, J., L. McMillan and J. Yu, 2006. Robust tracking and stereo matching under variable illumination. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Volume 1, June 17-22, 2006, New York, USA., pp: 871-878.

Zhang, K., J. Lu and G. Lafruit, 2009. Cross-based local stereo matching using orthogonal integral images. IEEE Trans. Circuits Syst. Video Technol., 19: 1073-1079.