Analysis of Real-Time Stereo Vision Algorithms On GPU

Ratheesh Kalarot*†, John Morris*, David Berry† and James Dunning†
*Department of Computer Science, University of Auckland, New Zealand
†Control Vision, Auckland, New Zealand

***Abstract*—Dozens of stereo correspondence algorithms whose matching performance has been measured are available, but the trade-off between speed and matching performance of viable real-time stereo has received much less attention. Here, we evaluate five correspondence algorithms(Symmetric Dynamic Programming Stereo ,SemiGlobal Matching,simple block matching, Belief Propagation, and its constant space variant) on a GPU using CUDA and report running time and matching performance. Analysis of these results leads to several insights into the advantages and limitations of the five algorithms - all on a GPU - and SemiGlobal block matching on a conventional CPU.**

I. Introduction

Passive stereo vision is an attractive solution for finding real-time dense depth information of a scene for various reasons . Unlike active depth inferring solutions such as LIDAR or *Microsoft-Kinect*, stereo vision does not need probe beams and hence is more acceptable to applications were objects are sensitive to and range limited by the probes. Active illumination systems project patterns to enhance stereo matching but do not depend on the quality of the projected patterns nor their complete penetration into the scene - if there was texture there already, the pattern is not needed. It is also much cheaper compared to high resolution time-of-flight sensors, *e.g.* Velodyne[1]. Despite these advantages, stereo vision is not yet seen as an industrial solution for real-time depth sensing applications. There are trade-offs between cost, speed and accuracy on stereo solutions which has been little studied. The availability of low cost, powerful Graphics Processing Units (GPUs) able to efficiently exploit the inherent parallelism in most stereo algorithms leads us to explore these trade-offs.

In this paper, we measured and analysed the performance of 5 stereo algorithms implemented on a GPU which can run at high frame rates. Section III gives details of these algorithms.

A. Related work

Many GPU based stereo implementations have been reported: early ones[2], [3], lacking tools supporting simple models of the target architectures - indirectly used the specialised functional units (*e.g.* vertex shader and pixel shader).

Simple correlation based block matching and dynamic programming algorithms are relatively easy to implement on a GPU because they have abundant low level data parallelism and linear relations between computing resources and number of disparity levels. Earlier, we built high resolution stereo systems based Gimelfarb's Symetric Dynamic Programming Stereo algorithm[4], [5], [6] and a multi resolution variant [7]. Implementations of dynamic programming algorithms for low resolution stereo have also been reported[8], [9].

Reports of several SemiGlobal Matching (SGM) GPU implementations may be found. Ernst and Hirschmuller describe a full SGM implementation using mutual information based cost functions[10]. Gibson *et al.* report SGM with Birchfeld-Tomasi cost functions[11]. Other adaptations of SGM have also been reported [12], [13].

Due to non-linear memory storage requirements and inherently slow access to the large memories needed, it is difficult to achieve high speeds and large disparity ranges with belief propagation or graph cut based algorithms, yet multiple CUDA based implementations have recently been reported [14], [15], [16]

Here we measure the performance of several GPU stereo algorithm implementations for *both* matching accuracy *and* speed to provide users with a basis for choosing the best implementation (with the optimum speed:matching performance:resources needed balance for high definition (Mpixel images), high fidelity (large - >100 - disparity ranges) stereo for a target application.

II. Compute Unified Device Architecture (CUDA)

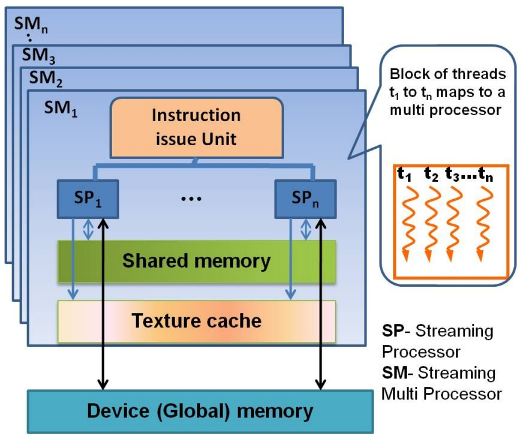


Fig. 1. Nvidia GPU architecture.

Compute Unified Device Architecture (CUDA) [17] is a unified general purpose parallel computing GPU architecture, an initiative by Nvidia Corporation. This architecture offers flexible programming with minimal extensions to the common

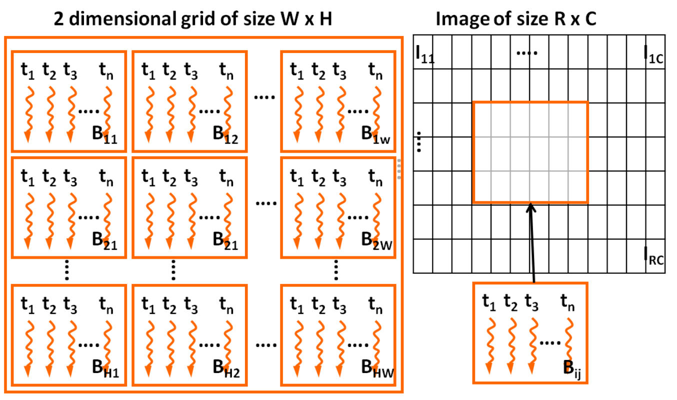


Fig. 2. Typical mapping of a block matching algorithm to a GPU. A 2D grid of threads blocks computes all disparities for 2D blocks of the image.

high-level C programming language. A CUDA enabled GPU (see Figure 1) consists of many Streaming Multiprocessors ($\mathrm{SM}_1..\mathrm{SM}_n$), each multiprocessor being a set of Streaming Processors ($\mathrm{SP}_1..\mathrm{SP}_n$) that share a fast access local memory referred to as the shared memory. All the multiprocessors can also access a global memory called the device memory. A multiprocessor has a common instruction issue unit and a very light weight thread scheduling mechanism to switch threads quickly when needed, eg during an I/O wait. A number of threads $t_1..t_n$ can be assigned to a streaming processor to form a thread block. The block maps physically to this processor although the number of threads in a block can exceed the number of the streaming processors. In such a case the threads are split into so-called warps, such that only one warp is active at a time. Threads in an active warp will execute the instructions in a parallel step locked manner but branches in the code are executed sequentially. Threads agreeing on the branch are executed in parallel while others wait until their corresponding branch is chosen for execution.
Figure 2 shows an example of stereo computation mapping to GPU. A block of threads containing thread $t_1..t_n$ is allocated to process a 2D block of image.
In this study, we used a commercial high end graphics card *GeForce GTX 480* - widely used in desk-top PCs - for our experiments. Throughput results for a low end mobile graphics card *GeForce GT 540M* were also measured for comparison - see Table I for specifications of both cards.

TABLE I
GRAPHICS CARDS USED HERE.

Features	GeForce GTX 480	GeForce GT 540M
CUDA Cores	480	96
Processor Clock	1.4 GHz	1.3 GHz
Memory Clock	1.9 GHz	0.9 GHz
Memory Size/Type	1.5 GB GDDR5	1 GB DDR3
Memory Bandwidth	177 GB/sec	29 GB/sec

III. STEREO ALGORITHMS

In this section, we briefly describe the key characteristics of the five stereo algorithms studied. All were fully realised on the GPU using CUDA. The first two implementations - $SDPS$ and SGM - are our work and will be discussed in detail. The remaining three - block matching (BM), belief propagation (BP) and belief propagation constant space (BP_CS) were taken from OpenCv [18]. We also include semi global block matching ($SGBM_{cpu}$) - a block matching based semi global algorithm, also available in OpenCv - as a candidate real-time algorithm on CPU to show GPU speedups attainable.

A. Symmetric Dynamic Programming Stereo ($SDPS$)

A detailed description of $SDPS$ and its GPU realisation can be found elsewhere (see[4], [6]). $SDPS$ is essentially a dynamic programming algorithm which finds the lowest cost path for stereo matching along a scan-line. Its key feature is that, rather than directly match the left image to the right image or vice versa, $SDPS$ reconstructs a Cyclopæan image that would be seen by a virtual camera midway between the optical centres of the two real cameras and matches it simultaneously to both the stereo images. A careful exploitation of this feature will allow half the number of disparities of a pixel in a scan-line to be calculated in parallel - fine grain parallelism readily exploited by GPUs.
As a typical dynamic programming algorithm, $SDPS$ accumulates costs for potentially optimal paths through the disparity search space (DSS) for a single Cyclopæan scan-line. The matching score is optimised in a single direction only. Both the storage needed and constraints on implementations are highlighted by outlined below basic computations for a simplified $SDPS$ that matches the Cyclopæan image signals (grey values or colours) directly to the left and right image signals. For each w pixel Cyclopæan scan-line, a potentially optimal predecessor for each DSS point is stored in the predecessor array of size $\mathcal{O}(w\Delta)$, where Δ is the disparity range.
Let x and d denote an x-coordinate and an x-disparity for a point in a scan-line based DSS. Let $D_{x,d}$ be a signal dissimilarity score for corresponding binocularly visible pixels at positions $x + \frac{d}{2}$ and $x - \frac{d}{2}$ in the scan-lines in the left and right images, respectively. If a single continuous surface is reconstructed, each DSS point (x, d) can have one of the three visibility states: $s \in \{\mathrm{B}, \mathrm{M_L}, \mathrm{M_R}\}$ where B stands for a binocularly visible point and $\mathrm{M}_{L|R}$ indicate points that are only monocularly visible - on either the left or the right image. Let D_o be a constant cost associated with a partially occluded or unmatched surface point; this cost also serves as a smoothing constraint along the scan-line. Potentially optimal costs, $C_{x,d,s}$, accumulated by $SDPS$ and predecessors, $\pi_{x,d,s} \equiv (x', d', s')$, for each DSS position (x, d, s) are:

$$C_{x,d,\mathrm{B}} = D_{x,d} + \min\{C_{x-1,d-1,\mathrm{ML}}, C_{x-2,d,\mathrm{B}}, C_{x-2,d,\mathrm{MR}})\} \quad (1)$$

$$\pi_{x,d,\mathrm{B}} = \arg\min\{C_{x-1,d-1,\mathrm{M_L}}, C_{x-2,d,\mathrm{B}}, C_{x-2,d,\mathrm{M_R}}\} \quad (2)$$

$$C_{x,d,\mathrm{M_L}} = D_\mathrm{o} + \min\{C_{x-1,d-1,\mathrm{M_L}}, C_{x-2,d,\mathrm{B}}, C_{x-2,d,\mathrm{M_R}}\} \quad (3)$$

$$\pi_{x,d,\mathrm{M_L}} = \arg\min\{C_{x-1,d-1,\mathrm{M_L}}, C_{x-2,d,\mathrm{B}}, C_{x-2,d,\mathrm{M_R}}\} \quad (4)$$

$$C_{x,d,\mathrm{M_R}} = D_\mathrm{o} + \min\{C_{x-1,d+1,\mathrm{B}}, C_{x-1,d+1,\mathrm{M_R}}\} \quad (5)$$

$$\pi_{x,d,\mathrm{M_R}} = \arg\min\{C_{x-1,d+1,\mathrm{B}}, C_{x-1,d+1,\mathrm{M_R}}\} \quad (6)$$

Visibility constraints on transitions between the single-surface DSS points considerably reduce the data needed to be stored in the predecessor array [4] differentiating $SDPS$ from other approaches. It also helps reduce the number of costly global memory access in our implementation. Each goal disparity profile is reconstructed by back-tracking over a sequence of transitions only between the potentially optimal visibility states. In this version, we used 32 bit pixels (compared our 8 bit implementation[6]). This in-turn allows high depth (*e.g.* 24 bit RGB, 16 bit grey *etc.*) to be processed natively without any performance penalty. Disparities are also 32 bits, removing practical limits on number of disparity levels that can be preserved. In this experiment, the occlusion cost D_o was set to 20.

B. Semi global matching (SGM)

Hirschmuller's original Semi Global Matching algorithm[19] uses a mutual information based pixel matching cost and aggregates matching costs in multiple directions - similar to dynamic programming above. Hirschmuller provides a detailed description. It is essentially minimising an energy

$$E(d) = \sum_{p} (C(p, D_p)) + \sum_{q \varepsilon N_p} P_1 T[|D_p - D_q| = 1] + \sum_{q \varepsilon N_p} P_2 T[|D_p - D_q| > 1] \quad (7)$$

where D_p is the disparity of pixel, p, q is a neighbour of p, C the cost functions and N_p is the neighbourhood of point, p, in all directions. Function T returns the weight of penalty cost $P_1$and P_2.

In contrast to traditional dynamic programming, SGM finds a minimum cost along a column of all possible disparity levels. This reduction is a major parallel computing hurdle for SGM . We use parallel radix sorting to engage maximum computing resources.

In the original implementation, the cost function, C, was based on mutual information but, for performance, we prefer a simple sum of absolute difference (SAD) cost function in our implementation. We also limit the number of directions along which we accumulate costs to four as the diagonal directions cause many non-coalesced memory accesses in the GPU. The left to right consistency check for occlusion detection of the original version was also discarded.

As in our $SDPS$ implementation, our SGM uses 32 bit values allowing precise pixel matching and high depth values. We set $P_1 = 5$ and $P_2 = 10$.

C. Block matching (BM)

Block Matching is a local dense stereo matching method [20] where cost of matching is found by correlation over a small window in the left image and a candidate window in the right image. The disparity assigned to a pixel in the left image is decided by the winner take all (WTA) approach. In a parallel processing view this is an excellent computation structure for fine grain parallelism where costs for all possible disparities for each pixel can be evaluated in parallel. Fig. 2 shows the mapping of a block matching algorithm to the GPU in the OpenCv implementation. Each thread block is operating on a small block of the image pixels (typically 128 × 21). The cost for each disparity is computed sequentially in a thread. This mapping saves some computation in window cost calculation by computing costs in each column only once. The cost function used in this particular implementation was sum of squared difference (SSD) since in the GPU's fast floating point ALUs the square operation is faster than an absolute difference one. It also adds some preprocessing (Sobel operation) and post processing (speckle filtering) steps. The main disadvantage of such local methods is that they perform badly in texture less regions. This can be improved with a large window but at the cost of a blurred disparity map and increased computation time. We kept the window size at $3x3$ pixels.

D. Belief propagation (BP , OpenCv)

Belief propagation algorithm is fully described elsewhere [21]. BP for correspondence is best explained as a labelling problem - one of assigning disparity labels to each pixel to minimise an energy function. We need to map a set of pixels in image, P, to a set of disparities, L, minimising the energy function:

$$E(f) = \sum^{p \varepsilon P} D_p(f_p) + \sum^{(p,q)\varepsilon A} V(f_p - f_q)) \quad (8)$$

where the first term is the data cost or dissimilarity cost ($Data\_Cost$) of each pixel and the second term is a discontinuity cost ($Disc\_Cost$). This can be done hierarchically over image size to reduce computation time. In the OpenCv implementation, the data cost is a weighted intensity difference which is thresholded to a maximum value[18].

$$Data\_Cost = data\_weight \cdot \min(|I_2 - I_1|, max\_data\_term) \quad (9)$$

The discontinuity or smoothing term is a thresholded discontinuity cost for a single jump in disparity value between neighboring pixels.

$$Disc\_Cost = \min(disc\_single\_jump \cdot |f_1 - f_2|, max\_disc\_term) \quad (10)$$

While the belief propagation algorithm has high degree of parallelism passing messages independently, it needs a large storage for passing messages:

$$MessageStorage = width\_step \cdot height \cdot ndisp \cdot 4 \cdot (1 + 0.25) \quad (11)$$

$$DataCostStorage = width\_step \cdot height$$
$$\cdot ndisp \cdot (1 + 0.25 + \cdots + \frac{1}{4^{levels}}) \qquad (12)$$

We set the window size to 3 and left the remaining parameters at their default values.

E. *Belief propagation- constant space (BP_CS , OpenCv)*

Belief propagation proves to be space intensive: the memory requirement is a multiple of size of the image and the disparity range, so OpenCv's BP scalability is very limited. For a stereo system processing 512×512 images with 64 disparities, the memory required is beyond the capacity of commercial graphic cards. In the Belief Propagation Constant Space (BP_CS) implementation [22], the storage requirement is reduced and made independent of disparity range by trading computation for accuracy. BP CS recomputes data terms at each level and also uses the course to fine computation of disparity levels along with the spatial hierarchical computation of normal BP. However, it also results in reduced depth accuracy. Again, we set the window size to 3 and left remaining parameters at default values.

F. *Semi global block matching ($SGBM_{cpu}$, OpenCv)*

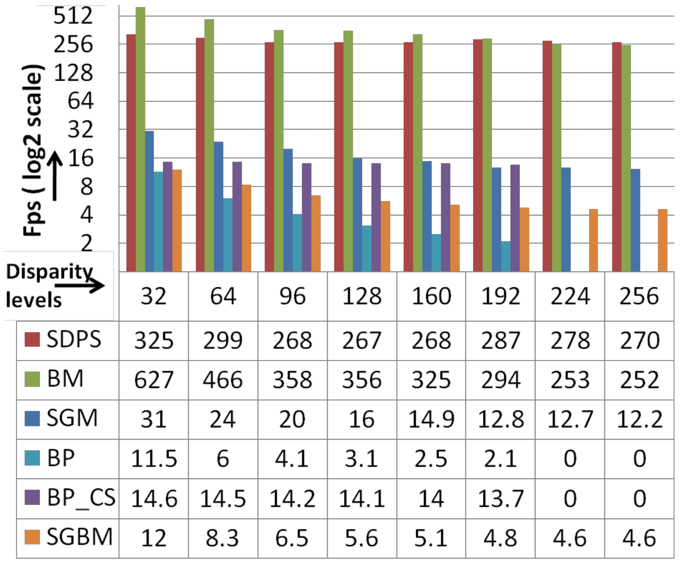
For comparison, we also included a 'close to real-time' algorithm on a high end CPU. OpenCv has an efficient semi global matching algorithm implementation which - called semi global block matching ($SGBM_{cpu}$). On lower parameter settings (small image and window size), $SGBM_{cpu}$can compute at near to video rates on a CPU. In this SGM implementation, the matching cost is derived by correlation over small blocks rather than individual pixels. The number of passes is also limited to five rather than a traditional eight passes. This implementation includes many filtering and post processing steps as options which we avoided for a fair comparison. We set $P_1 = 5$ and $P_2 = 10$ as in GPU version. Window size was set 3 - as with other window based algorithms. Other parameters were set to their defaults. Performance was reported for an Intel core i7-2.8GHZ processor using single core only.

IV. Performance analysis and comparison

In this section we compare and analyse computing performance and matching accuracy of these algorithms on a GPU and assess scalability highlighting practical limitations.

A. *Speed up*

Figure 3 shows frames per second (fps) achieved for several disparity ranges for each algorithm on a small 512×512 image. $SDPS$ and BMachieve by far the highest frame rates - as expected. Note that frame rates are shown against a log scale! This partially due to the low number of computations required by both algorithm and very efficient mapping to GPU resources and communication paths. For BMmatching (see Figure 2), each thread block is matched to a small 2D grid in the image, so the number of threads in each block is independent of the number of disparity levels. In $SDPS$ each thread block processes a scan-line and the number of threads in a block is half the number of disparity levels. It is interesting to note that, for higher disparity ranges,$SDPS$ performs better due to its efficient use of the number of threads available in each multiprocessor.

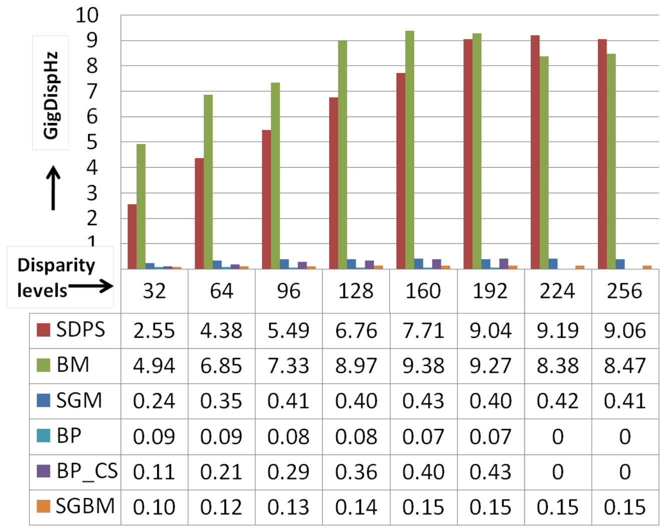


	32	64	96	128	160	192	224	256
🟥 SDPS	325	299	268	267	268	287	278	270
🟩 BM	627	466	358	356	325	294	253	252
🟦 SGM	31	24	20	16	14.9	12.8	12.7	12.2
🟦 BP	11.5	6	4.1	3.1	2.5	2.1	0	0
🟪 BP_CS	14.6	14.5	14.2	14.1	14	13.7	0	0
🟧 SGBM	12	8.3	6.5	5.6	5.1	4.8	4.6	4.6

Fig. 3. Frame rate *vs* number of disparity levels for 512×512 pixel images on a GeForce GTX 480 GPU

B. *Throughput*

A practically relevant metric is the actual throughput in number of disparities processed per second: we previously proposed a measure of $GigDispHz$ [6] or billion disparity calculation per second. For a state of the art high definition real-time stereo system, throughput should be above 1 $GigDispHz$.



	32	64	96	128	160	192	224	256
🟥 SDPS	2.55	4.38	5.49	6.76	7.71	9.04	9.19	9.06
🟩 BM	4.94	6.85	7.33	8.97	9.38	9.27	8.38	8.47
🟦 SGM	0.24	0.35	0.41	0.40	0.43	0.40	0.42	0.41
🟦 BP	0.09	0.09	0.08	0.08	0.07	0.07	0	0
🟪 BP_CS	0.11	0.21	0.29	0.36	0.40	0.43	0	0
🟧 SGBM	0.10	0.12	0.13	0.14	0.15	0.15	0.15	0.15

Fig. 4. GigDispHz *vs* disparity levels for a 512×512 images on GeForce GTX 480 GPU.

Figure 4 shows the $GigaDispHz$ of various algorithms on a 512×512 image for various disparity levels. $SDPS$ and BMhave less dependency on the number of disparities levels and their throughput increases as data parallelism increases with the number of disparity levels and more ALUs can operate in parallel. Other algorithms struggle as their memory access requirements increase exponentially with number of disparity levels.

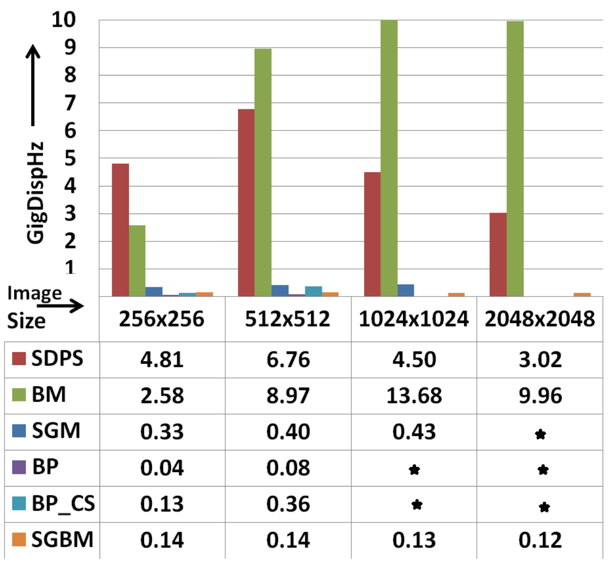


Image Size	256x256	512x512	1024x1024	2048x2048
🟥 SDPS	4.81	6.76	4.50	3.02
🟩 BM	2.58	8.97	13.68	9.96
🟦 SGM	0.33	0.40	0.43	*
🟪 BP	0.04	0.08	*	*
🟦 BP_CS	0.13	0.36	*	*
🟧 SGBM	0.14	0.14	0.13	0.12

Fig. 5. GigDispHz *vs* image size for constant number of disparity levels (128) measured on GeForce GTX 480 GPU. * Indicates tath the algorithm cannot be realised on that particular settings because of memory limitations

Figure 5 shows throughput of various algorithm *vs* image sizes for a fixed disparity range of 128. For a high definition stereo system processing of 1024×1024 images, only $BM$and $SDPS$ can achieve real-time performance. On 2048×2048 images, BMreaches its maximum throughput, while $SDPS$ performance is reduced because of poor thread occupancy. This is attributed to the increased usage of shared memory for a longer scan-line.
Table II shows the speed-up and throughput achieved on a low end mobile graphics card GeForce Gt540M.

C. Scalability and configuration limits

While different algorithms exhibits different throughput and matching accuracy , it is interesting to note their scalability to the problem size that they can handle. It is always possible to fall-back to a sub-optimal stitching of parts of stereo solution to handle bigger limits using any algorithm , still their native limits are worth understanding for practical applications.Figure 6 shows maximum achievable disparity levels of various algorithm on in different image sizes. BP and BP_CS are only usable in low resolution settings. For high definition stereo .$SDPS$ and $BM$are the only choices.

D. Accuracy

Real-time stereo algorithms trade matching accuracy for speedup. For a fair comparison, we have kept the parameters

TABLE II
THROUGHPUT AND SPEED-UP OF ALGORITHMS ON A GEFORCE GT 540M GPU

		FPS				
Image Size	Disparity levels	SDPS	SGM	BM	BP	BP_CS
512x512	32	41.5	4	187	1.90	3.1
512x512	64	37.7	3.3	116	1.00	3.1
512x512	96	35	2.8	84.8	0.70	3
512x512	128	34	2.2	75.9	0.50	3
256x256	32	158	17.3	661	7.50	10.6
256x256	64	155	15	432	3.9	10.8
256x256	96	159	14	315	2.7	10.6
256x256	128	174	12.8	417	2	10.5
		GigDispHz				
512x512	32	0.33	0.03	1.47	0.02	0.02
512x512	64	0.55	0.05	1.70	0.01	0.05
512x512	96	0.72	0.06	1.74	0.01	0.06
512x512	128	0.87	0.06	1.91	0.01	0.08
256x256	32	0.29	0.03	1.22	0.01	0.02
256x256	64	0.49	0.05	1.36	0.01	0.03
256x256	96	0.63	0.06	1.24	0.01	0.04
256x256	128	0.73	0.05	1.75	0.01	0.04

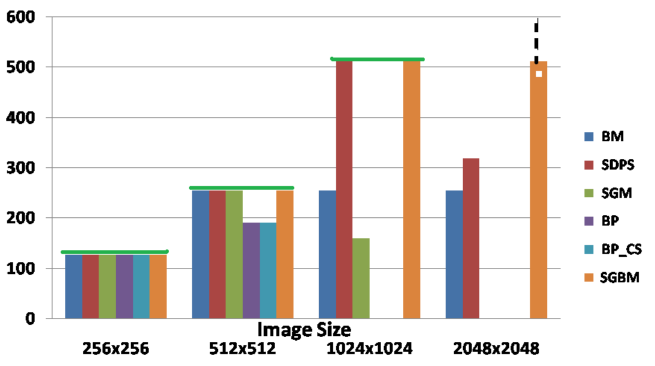


Fig. 6. Maximum achievable disparity size of various algorithm on GeForce GTX 480 GPU, for different image sizes .Green bar shows the limit of a usable configuration in every image size, beyond which the results are practically irrelevant

of these algorithm at default values and avoided any kind of post processing as far as possible. We have tested the Middlebury images[20] as well a high resolution synthetic data set 'FATBOY' generated by us[23]. Table III shows the RMS error and mismatch (BadPix -pixels with disparity error > 2 pixels) rate for various data sets on each algorithm[1]. In general, $SDPS$ is clearly more accurate than the other algorithms. SGM can perform better than $SDPS$ with more saline paths and addition of right to left scan-line consistency. Belief propagation also may perform well on many inputs but its inability to scale will make it a poor choice. Figure 7 shows false colour disparity maps for several Middlebury image pairs along with our synthetic 'FATBOY' set and some 'real' images outside our laboratory (CITR1, CITR2).

[1]We note that one default configuration for all data sets does not necessarily provide a meaningful comparison, as practical applications will adapt parameters to particular scenarios.

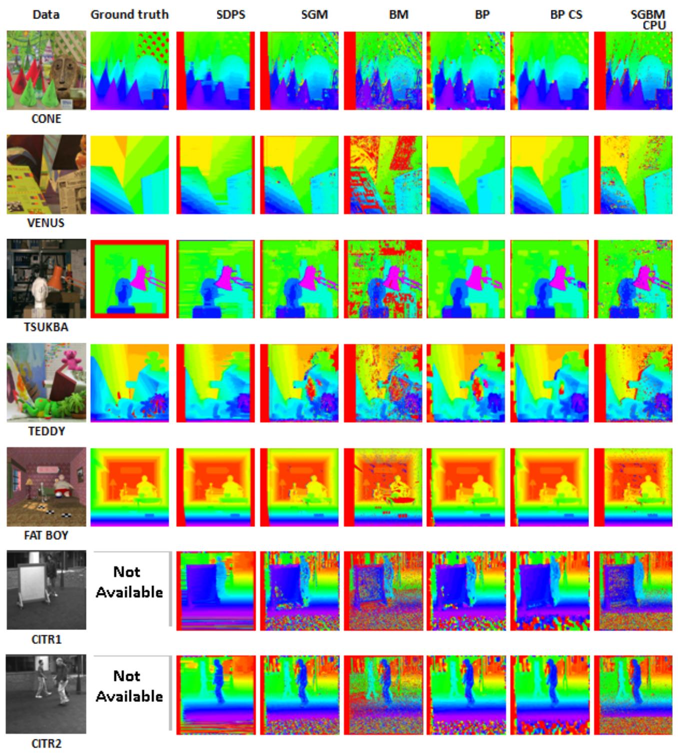


Fig. 7. Results of all stereo algorithm on various data sets

TABLE III
ACCURACY OF ALGORITHMS ON DIFFERENT DATA SETS

Algo	Error	CONE	TEDDY	TSUKUBA	VENUS	FATBOY
SDPS	Bad Pix%	8.7	6.16	5.5	4.9	1.2
	Rms Error	2.3	2.1	1.18	1.3	0.7
SGM	Bad Pix%	14.7	16	6.6	5.23	3
	Rms Error	5.7	7.3	1.51	2.4	3.9
BM	Bad Pix%	30	32.8	27.83	35.4	14.2
	Rms Err	10.5	11.5	3.3	5.6	10.3
BP	Bad Pix%	15.2	16.11	3.9	2.3	1.7
	Rms Error	5.6	8	1.27	1.25	2.9
BP_CS	Bad Pix%	7.3	11.11	4.3	2.4	1.5
	Rms Error	3.5	4.6	1.1	1.3	2.8
SGBM	Bad Pix%	10.1	14	8.6	10.1	5.9
	Rms Error	7.8	8.3	1.8	3	5.2

V. CONCLUSION

We analysed and compared algorithms capable of real-time performance on a GPU. Whilst this analysis provides a good guide to the likely performance of each algorithm, it is important to note that this is an indication only: in practice, ‘tweaking’ performance (either speed or accuracy) with special tests or optimizations applicable to one type of scene only may change the overall picture. We have also deliberately not included any pre- or post-processing and compared the basic algorithms only.

$SDPS$, $BM$and SGM emerge as good candidates for high-resolution stereo because of their scalability. BMmatches poorly on texture-less images. $SDPS$ suffers from streaking artefacts. A full version of SGM can achieve better matching accuracy but with a higher performance penalty.

REFERENCES

[1] Velodyne Lidar Inc. (2011, October) Velodyne LIDAR. [Online]. Available: http://velodyne.com/

[2] Q. X. Yang, L. Wang, and R. G. Yang, “Real-time global stereo matching using hierarchical belief propagation,” in *Proc. British Machine Vision Conf. (BMVC 2006)*, 2006, p. III:989.

[3] J. Woetzel and R. Koch, “Multi-camera real-time depth estimation with discontinuity handling on PC graphics hardware,” in *Proc. IAPR Int. Conf. on Pattern Recognition (ICPR 2004)*, vol. 1, 2004, pp. 741–744.

[4] G. Gimel’farb, “Probabilistic regularisation and symmetry in binocular dynamic programming stereo,” *Pattern Recognition Letters*, vol. 23, no. 4, pp. 431–442, 2002.

[5] R. Kalarot and J. Morris, “Implementation of symmetric dynamic programming stereo matching algorithm using cuda,” in *Proc. 16th Korea-Japan Joint Workshop on Frontiers of Computer Vision*, 2010.

[6] ——, “Comparison of FPGA and GPU implementations of real-time stereo vision,” in *Proc. 6th IEEE Workshop on Embedded Computer Vision*, 2010, pp. 9–15.

[7] R. Kalarot, J. Morris, and G. Gimel’farb, “Performance analysis of multi-resolution symmetric dynamic programming stereo on GPU,” in *25th International Conference Image and Vision Computing New Zealand (IVCNZ) 2010*, 2010.

[8] L. Wang, M. Liao, M. L. Gong, R. G. Yang, and D. Nister, “High-quality real-time stereo using adaptive cost aggregation and dynamic programming,” in *Proc. Int. Symposium on 3D Data Processing, Visualization and Transmission*, 2006, pp. 798–805.

[9] M. L. Gong and Y. H. Yang, “Real-time stereo matching using orthogonal reliability-based dynamic programming,” *IEEE Trans. on Image Processing*, vol. 16, no. 3, pp. 879–884, 2007.

[10] I. Ernst and H. Hirschmuller, “Mutual information based semi-global stereo matching on the GPU,” in *Advances in Visual Computing*, 2008, pp. I: 228–239.

[11] J. Gibson and O. Marques, “Stereo depth with a unified architecture GPU,” in *Computer Vision on GPU*, 2008, pp. 1–6.

[12] M. Humenberger, T. Engelke, and W. 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*, june 2010, pp. 77 –84.

[13] A. Woodward, D. Berry, and J. Dunning, “Real-time stereo vision on the vision server framework for robot guidance,” in *25th International Conference Image and Vision Computing New Zealand (IVCNZ) 2010*, 2010.

[14] V. Vineet and P. Narayanan, “CUDA cuts: Fast graph cuts on the GPU,” in *Proc. IEEE CS Conf. Computer Vision and Pattern Recognition (CVPR 2008)*, 2008, pp. 1–8.

[15] S. Grauer-Gray and C. Kambhamettu, “Hierarchical belief propagation to reduce search space using CUDA for stereo and motion estimation,” in *WACV*. IEEE Computer Society, 2009, pp. 1–8.

[16] Open source BSD license. (2011, October) OpenCv documentation -GPU-accelerated Computer Vision. [Online]. Available: http://opencv.itseez.com/modules/gpu/doc/gpu.html

[17] Nvidia Corporation. (2011, October) Compute Unified Device Architecture(CUDA). [Online]. Available: http://en.wikipedia.org/wiki/CUDA

[18] Open source BSD license. (2011, October) OpenCv-Open Source Computer Vision Library. [Online]. Available: http://opencv.willowgarage.com/wiki/

[19] H. Hirschmüller, “Accurate and efficient stereo processing by semi-global matching and mutual information,” in *CVPR (2)*, 2005, pp. 807–814.

[20] D. Scharstein and R. Szeliski, “A taxonomy and evaluation of dense two-frame stereo correspondence algorithms,” *Int. Journal of Computer Vision*, vol. 47, pp. 7–42, 2002.

[21] P. F. Felzenszwalb and D. P. Huttenlocher, “Efficient belief propagation for early vision,” *International Journal of Computer Vision*, vol. 70, no. 1, pp. 41–54, Oct. 2006. [Online]. Available: http://dx.doi.org/10.1007/s11263-006-7899-4

[22] Q. Yang, L. W. 0002, and N. Ahuja, “A constant-space belief propagation algorithm for stereo matching,” in *CVPR*, 2010, pp. 1458–1465.

[23] R. Kalarot and J. Morris, “Dataset for symmetric dynamic programming stereo evaluation,” 2010. [Online]. Available: http://www.cs.auckland.ac.nz/ ratheesh/SDS/sds.html