Real-Time Implementation of Stereo Vision Based on Optimised Normalised Cross-Correlation and Propagated Search Range on a GPU

Rui Fan, Naim Dahnoun

***Abstract*—Due to the trade-offs between accuracy and speed, binocular stereo vision is still a challenging task in 3D computer vision research area. In this paper, an efficient stereo matching algorithm is implemented on a state-of-the-art GPU to achieve highly accurate disparity maps in real time for various autonomous vehicle applications. The proposed algorithm is developed from our previous paper, where the search range at row v is propagated from three estimated neighbourhood disparities located at row $v + 1$. In order to speed up the execution, the prevalent NCC algorithm is optimised by factorising the equation into five independent parts. The computations of μ_l, μ_r, σ_l and σ_r are accelerated by using the integral images I_l, I_r, I_{l^2} and I_{r^2}. The values of μ and σ are stored in static program storage for indexing during the stereo matching, which further reduces expensive calculations to help the system perform in real time. The main purpose of this work is to accelerate the processing speed by highly exploiting the parallel computing architectures (OpenMP and CUDA). The performance of the implementation on an NVIDIA GTX 970M GPU is compared with the performance of the implementations on an Intel Core i7-4720HQ CPU using both single thread and multiple threads. The experimental results illustrate that the GPU implementation yields 37 fps when processing the images (resolution: 1242 × 375) from the KITTI database, which is between two and nine times faster than the implementations on the CPU using a different number of threads.**

***Index Terms*—stereo matching, GPU, disparity map, real time, autonomous vehicle applications, NCC, integral images, parallel computing, OpenMP, CUDA.**

I. INTRODUCTION

AUTONOMOUS vehicles have been developing rapidly since Google launched their self-driving car project in 2009 [1]. Recently, with a number of remarkable high-technology breakthroughs being witnessed like science fictions, the race to make the driver-less cars a reality among many companies like Google, Tesla and BMW has also accelerated significantly. For example, the driver-less vehicles are able to communicate with each other via the 5G network which offers a more powerful internet access to transfer vast amounts of data with an approximately 50 times faster speed than the current 4G systems. The computer binocular stereo vision has also been prevalently used in various prototype vehicle road tests to provide the 3D information for many autonomous vehicle applications (i.e. signage detection, accident avoidance and lane detection), which helps to enhance the robustness of the ADAS (Advanced Driver Assistance Systems) dramatically.

For various computer stereo vision systems, matching speed and disparity accuracy are two key components [2]. The stereo matching algorithms can be classified as local algorithms, semi-global algorithms and global algorithms [3]. Although the global algorithms can provide a more accurate disparity map by minimising the global cost with the assistance of more sophisticated and computationally intensive optimisation approaches such as BP (Belief Propagation), GC (Graph Cut) and DP (Dynamic Programming), they are significantly challenged to achieve real-time performance without specialised hardware accelerators [4]. Although the performance of the systems can be improved with future advances in computational power in hardware techniques, it is shown that the optimisations on the algorithm side can also result in a lot of impressive increases in the stereo matching speed [2]. For example, authors in [5] proposed a quasi-dense stereo matching algorithm named GCS which propagates the search range from a collection of estimated confidential disparities to their neighbours in order to save redundant computations in block matching as well as to improve the accuracy of the estimated disparity map. Similarly, our previous work [6] presented an efficient disparity estimation algorithm where the search range at row v is propagated from three estimated neighbouring disparities located at row $v + 1$, which is more suitable for various autonomous vehicle applications with its higher accuracy and lower computational complexities [7]. After that, an optimised version was presented in our previous paper [8] with the consideration of the search range suggested by a horizontal neighbouring disparity, which further enhances the disparity map precision but is difficult to implement on various parallel computing architectures for real-time purposes. Recently, Lin et al. proposed an optimisation methodology for the NCC (Normalised Cross-Correlation) computation by dividing the standard equation into four independent parts to reduce the computational complexities [9]. However, instead of using sliding windows to accelerate the calculations of μ and σ, the integral image is a more efficient algorithm that can be used in block matching to save redundant computations by calculating the sum of pixel intensities over a rectangular region of the image with only four operations [10]. Therefore, we developed our previous algorithm [6] by factorising the NCC cost function as five independent parts and accelerating

Rui Fan is with the Visual Information Laboratory, University of Bristol, BS8 1UB, UK. Email: ranger.fan@bristol.ac.uk, URL: http://www.ruirangerfan.com

Naim Dahnoun is with the Department of Electrical and Electronic Engineering, University of Bristol, BS8 1UB, UK. Email: naim.dahnoun@bristol.ac.uk

their computations using the integral images. The proposed algorithm in this paper is based on three parts: μ and σ memorisation accelerated by integral images, row v_{max} stereo matching with a full search range, and the rest of the disparity map estimation based upon search range propagation. The disparities on the occlusion areas are removed with the left-right consistency (LRC) check to further refine the estimated disparity map. The main purpose of this paper is to accelerate the algorithm implementation by highly exploiting the parallel computing architectures. The performance of the GPU (Graphics Processing Unit) implementation is evaluated and compared with the performance of the CPU (Central Processing Unit) implementations using a different number of threads.

The remainder of this paper is organised as follows: Section II describes the proposed disparity estimation algorithm. Section III discusses the implementations on both CPU and GPU. Section IV presents the elimination of the occlusions with LRC check. Section V illustrates the experimental results. Section VI concludes the paper and proposes possible future work.

II. Algorithm Description

A. Block matching and memorisation

In this paper, the input stereo image pairs are assumed to be well calibrated. An example of block matching is illustrated in Fig. 1, where the disparity costs are calculated by shifting a series of square blocks whose side length is $2\rho+1$ ($\rho \in \mathbb{Z}^+$) from the right image between d_{min} and d_{max} and matching them with a constant square block from the left image. n is the number of pixels within a square block and it is usually an odd number because of $n = (2\rho + 1)^2$. The disparity with the lowest cost or the highest correlation is then selected as the correspondence. This optimisation is also known as winner-take-all (WTA), where d_{min} and d_{max} are decided by the furthest or the closest objects to be detected. With the consideration of the trade-offs between accuracy and speed, the block size is proposed to be 7×7 ($\rho = 3$ and $n = 49$) in this system. Due to its insensitivity to the intensity difference during the block matching, the NCC is chosen as the proposed correlation measurement approach which is depicted as:

$$c(u,v,d) = \frac{1}{n\sigma_l\sigma_r} \sum_{(u,v)\in W} (i_l(u,v) - \mu_l)(i_r(u-d,v) - \mu_r) \quad (1)$$

where $c(u,v,d)$ is defined as the cost (correlation), u and v are the horizontal and vertical coordinates of a pixel, $i_l(u,v)$ or $i_r(u,v)$ is the intensity of the pixel located at (u,v) in the left or right image. W represents the reference domain of the block. μ_l and μ_r denote the means of the intensities within the left and right blocks, respectively. σ_l and σ_r represent their corresponding standard deviations [3]:

$$\sigma_l = \sqrt{\sum_{(u,v)\in W} (i_l(u,v) - \mu_l)^2/n} \quad (2)$$

$$\sigma_r = \sqrt{\sum_{(u,v)\in W} (i_r(u-d,v) - \mu_r)^2/n} \quad (3)$$

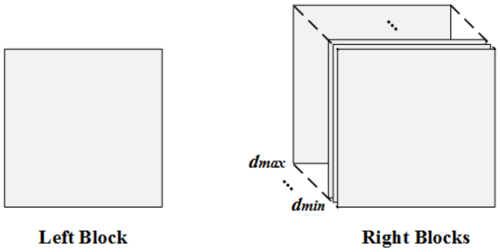


Fig. 1. Stereo block matching.

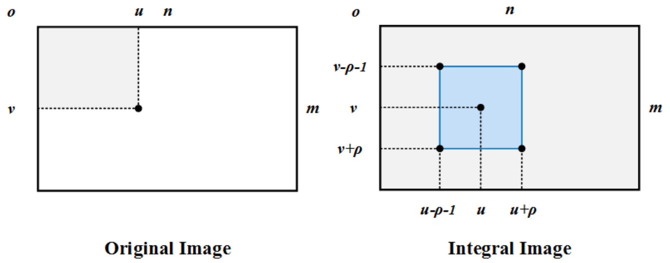


Fig. 2. Integral image.

When the left block is selected, the calculation of μ_l and σ_l are always repeated because d is only used to select the position of the right blocks for stereo matching. Therefore, the four independent parts μ_l, μ_r, σ_l and σ_r can be pre-calculated and stored in static program storage for direct indexing. The integral image algorithm can be used to compute μ_l and μ_r efficiently [11], which is illustrated in Fig. 2. The algorithm has two steps: integral image initialisation and values indexing from the initialised reference. In the first step, for a discrete image i whose pixel intensity at (u,v) is $i(u,v)$, its integral image intensity $I(u,v)$ at the position of (u,v) is defined as:

$$I(u,v) = \sum_{i\le u, j\le v} i(i,j) \quad (4)$$

Algorithm 1 details the implementation of the integral image initialisation, where I is calculated serially based on its previous neighbouring results to save unnecessary computations.

Algorithm 1: Integral image initialisation

```
Data: Original image of size $m \times n$: $i$
Result: Integral image of size $m \times n$: $I$
$I(0,0) \leftarrow i(0,0)$;
for $u \leftarrow 1$ to $n-1$ do
    $I(u,0) \leftarrow I(u-1,0) + i(u,0)$;
for $v \leftarrow 1$ to $m-1$ do
    $I(0,v) \leftarrow I(0,v-1) + i(0,v)$;
for $u \leftarrow 1$ to $n-1$ do
    for $v \leftarrow 1$ to $m-1$ do
        $I(u,v) \leftarrow I(u,v-1) + I(u-1,v)$
        $-I(u-1,v-1) + i(u,v)$;
```

With a given integral image, the sum (s) of pixel intensities within a square block whose $\rho = (\sqrt{n} - 1)/2$ and the centre is

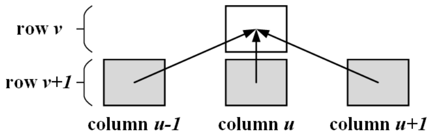


Fig. 3. Search range propagation.

(u, v) can be computed with four references $r_1 = I(u+\rho, v+\rho)$, $r_2 = I(u-\rho-1, v-\rho-1)$, $r_3 = I(u-\rho-1, v+\rho)$, $r_4 = I(u+\rho, v-\rho-1)$, which is illustrated in equation 5. The corresponding mean $\mu(u, v)$ of the intensities within the block is $s(u, v)/n$ which is then stored in static program storage for the computation of σ and c.

$$s(u, v) = r_1 + r_2 - r_3 - r_4 \tag{5}$$

In addition, equations 2 and 3 can be rearranged as equations 6 and 7 to compute σ_l and σ_r more efficiently.

$$\sigma_l = \sqrt{\sum_{(u,v)\in W} i_l^2(u, v)/n - \mu_l^2} \tag{6}$$

$$\sigma_r = \sqrt{\sum_{(u,v)\in W} i_r^2(u-d, v)/n - \mu_r^2} \tag{7}$$

where $\sum i_l^2(u, v)$ and $\sum i_r^2(u-d, v)$ are dot products. Similarly, the computations of $\sum i_l^2(u, v)$ and $\sum i_r^2(u-d, v)$ can also be accelerated by initialising two integral images I_{l^2} and I_{r^2} as references for indexing. Therefore, the standard deviations σ_l and σ_r are also calculated and stored in static program storage for the efficient computation of c, and equation 8 is obtained.

$$c(u, v, d) = \frac{1}{n\sigma_l\sigma_r}\left[\sum_{(u,v)\in W} i_l(u, v)i_r(u-d, v) - n\mu_l\mu_r\right] \tag{8}$$

From equation 8, only $\sum i_l(u, v)i_r(u-d, v)$ needs to be calculated during stereo matching. Hence, with the values of μ_l, μ_r, σ_l and σ_r able to be indexed directly, equation 1 is simplified as a dot product. In practical experiments, the factorisation of the NCC equation and the independent parts memorisation make the speed of stereo matching increase by about 36% when the block size is 7×7.

B. Search Range Propagation

The disparity is estimated iteratively row by row from row v_{max} to row v_{min}. Row $v_{max} = m - \rho - 2$ (Due to the utilisation of integral images, row $v_{max} = m - \rho - 2$ instead of $v_{max} = m - \rho - 1$) is processed with a full search range from d_{min} to d_{max}, where d_{min} is 0 and d_{max} is 70 for the KITTI datasets [12]–[15]. After that, the search range for stereo matching at row v is propagated from three neighbours' disparities on row $v+1$, which is illustrated in Fig. 3. For a pixel at the position of (u, v), its neighbouring disparities $d_1 = l(u-1, v+1)$, $d_2 = l(u, v+1)$ and $d_3 = l(u+1, v+1)$ have been estimated in the previous iteration, where l is the disparity map. Hence, the search range SR for the position of (u, v) is restricted by equation 9 [6], [8], where τ is the bound of the search range and it is usually selected as 1 or 2 in our proposed system. Algorithm 2 presents the details of the disparity map estimation, and the implementations will be discussed in Section III.

$$SR = \bigcup_{k=u-1}^{u+1} \{sr \mid sr \in [l(k, v+1) - \tau, l(k, v+1) + \tau]\} \tag{9}$$

Algorithm 2: Left disparity map estimation

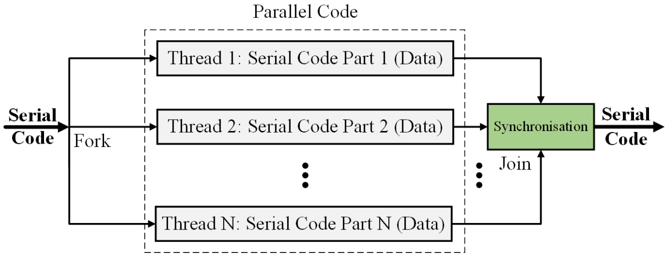
```
Data: left image of size $m \times n$: $i_l$
      right image of size $m \times n$: $i_r$
      left mean map of size $m \times n$: $\mu_l$
      right mean map of size $m \times n$: $\mu_r$
      left standard deviation map of size $m \times n$: $\sigma_l$
      right standard deviation map of size $m \times n$: $\sigma_r$
Result: left disparity map of size $m \times n$: $l^{lf}$
$v \leftarrow m - \rho - 2$;
for $u_l \leftarrow \rho + d_{max} + 1$ to $n - \rho - d_{max} - 2$ do
    $d_i \in [d_{min}, d_{max}]$;
    $u_r \leftarrow u_l - d_i$;
    if $\sigma_l(u_l, v)\sigma_r(u_r, v) \neq 0$ then
        $\sigma_{lr} \leftarrow \sigma_l(u_l, v)\sigma_r(u_r, v)$;
        $c(u_l, v, d_i) \leftarrow$
          $(\sum I_l(u_l, v)I_r(u_r, v) - n\mu_l(u_l, v)\mu_r(u_r, v))/n\sigma_{lr}$;
    $l^{lf}(u_l, v) \leftarrow \arg\max(c(u_l, v, d_i))$;
for $v \leftarrow m - \rho - 3$ to $\rho + 1$ do
    for $u_l \leftarrow \rho + d_{max} + 1$ to $n - \rho - d_{max} - 2$ do
        $SR(u_l, v) = \bigcup_{k=u_l-1}^{u_l+1}\{sr|sr \in$
          $[l(k, v+1) - \tau, l(k, v+1) + \tau]\}$;
        $d_i \in SR(u_l, v)$;
        $u_r \leftarrow u_l - d_i$;
        if $\sigma_l(u_l, v)\sigma_r(u_r, v) \neq 0$ then
            $\sigma_{lr} \leftarrow \sigma_l(u_l, v)\sigma_r(u_r, v)$;
            $c(u_l, v, d_i) \leftarrow$
              $(\sum I_l(u_l, v)I_r(u_r, v) - n\mu_l(u_l, v)\mu_r(u_r, v))/n\sigma_{lr}$;
        $l^{lf}(u_l, v) \leftarrow \arg\max(c(u_l, v, d_i))$;
```

III. Implementations

The main purpose of this paper is to accelerate the algorithm execution and further achieve real-time performance by exploiting the parallel computing architectures. In this section, the performance of the implementations on an Intel Core i7-4720HQ CPU using both single thread and multiple threads are compared with the performance of the implementation on an NVIDIA GTX 970M GPU. The results illustrate that the performance on GPU is approximately nine times faster than a single-threading CPU implementation and about two times faster than the implementation on the CPU using eight threads.

A. CPU implementation

In order to speed up the execution, OpenMP is used to break a serial code into independent chunks to process it in parallel [16]. OpenMP mainly has three components: work-sharing, data sharing and synchronisation. Work-sharing specifies a part of the serial code to be parallelised, data sharing specifies an appropriate scheduling model, and synchronisation



Parallel Code
Thread 1: Serial Code Part 1 (Data)
Thread 2: Serial Code Part 2 (Data)
Thread N: Serial Code Part N (Data)
Serial Code
Fork
Synchronisation
Join
Serial Code

Fig. 4. OpenMP structure.

determinates how data is shared [17]. The fork-join model is utilised in the implementation, which is shown in Fig. 4. In this section, we only discuss the implementation on CPU using eight threads. The integral images I_l, I_r, I_{l^2} and I_{r^2} are initialised serially using one thread. After that, the for loops in the μ_l, μ_r, σ_l and σ_r computation stage are divided among eight threads with *omp for* clause. *dynamic* is selected as the scheduling model because of its better performance with unequal subtasks distributed to each thread. When a thread finishes a chunk of data, it retrieves the next chunk. Meanwhile, μ_l, μ_r, σ_l and σ_r are declared as *private* variables to make each thread have its own copy. For synchronisation, the *nowait* clause is utilised to ignore the implicit barrier of the *for* pragma. The rest of the algorithm is parallelised with *omp sections*, and the serial code is equally divided into eight sub-blocks to execute concurrently. The results from each thread join together and the disparity map is obtained.

B. GPU Implementation

Graphic processors have been widely used to accelerate various 3D computer vision applications which are computationally intensive but able to be implemented in parallel to achieve the real-time performance. A brief overview of their general architecture is shown in Fig. 5. Compared with a CPU which consists of a low number of cores optimised for sequentially serial processing, the GPU has a massively parallel architecture which is composed of hundreds or thousands of lighter cores to handle multiple tasks concurrently.

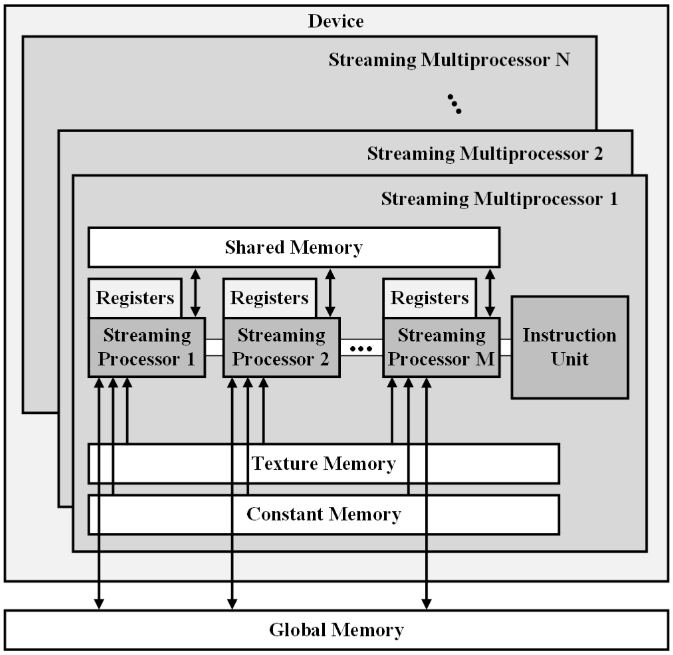


Fig. 5. Brief overview of general GPU architecture [18].

As shown in Fig. 5, a GPU consists of N streaming multiprocessors (SMs) with M streaming processors (SPs) on each of them. The SIMD (Single Instruction Multiple Data) architecture allows the SPs on the same SM to execute the same instruction but operate different data at each clock cycle [19]. The device has its own Dynamic Random Access Memory (DRAM) which consists of global memory, constant memory and texture memory that can communicate with the host memory via the GMCH (graphical/memory controller hub) and the ICH (I/O controller hub) which are also known as the Intel northbridge and the Intel southbridge, respectively. Each SM has four types of on-chip memories: register, shared memory, constant cache and texture cache. Since they are on-chip memories, the constant cache and texture cache are utilised to speed up the data fetching from the constant memory and texture memory, respectively. Due to the fact that the shared memory is small, it is used for the duration of processing a block, while the register is only visible to the thread. The details of different types of memories are illustrated in Table I.

Memory	Location	Cached	Access	Scope
Register	On-chip	N/A	R/W	one thread
Shared	On-chip	N/A	R/W	All threads in a block
Global	Off-chip	No	R/W	All threads + host
Constant	Off-chip	Yes	R	All threads + host
Texture	Off-chip	Yes	R	All threads + host

TABLE I
GPU MEMORY ARCHITECTURE.

In CUDA C programming, the threads are grouped into a set of three-dimensional thread blocks which are also organised as a three-dimensional grid. A kernel is launched as a grid of thread blocks and only one kernel can be executed at one time. Once a thread block is distributed to an SM, the threads are divided into groups of 32 parallel threads which are executed by SPs; each group with 32 parallel threads is called a warp. Therefore, the block size is usually chosen as a multiple of 32 to keep the efficiency of data processing.

The NVIDIA GTX 970M GPU has 10 SMs with 128 SPs on each of them. The maximal dimension size of a thread block is $(x : 1024, y : 1024, z : 64)$ and the maximum number of threads per block is 1024. The integral images I_l, I_r, I_{l^2} and I_{r^2} are initialised serially on the CPU and their data is then transferred to the global memory. In the GPU architecture, a thread is more likely to fetch the memory from the closest addresses that its nearby threads fetched. However, the addresses they accessed are usually not consecutive, which makes the use of the cache not possible. Therefore, the texture memory is utilised to optimise the caching for 2D spatial locality. From Fig. 5, the texture memory is read-only and cached on-chip to provide a higher effective bandwidth by reducing the memory requests from the global memory. First of all, two 2D texture reference objects are created. Then,

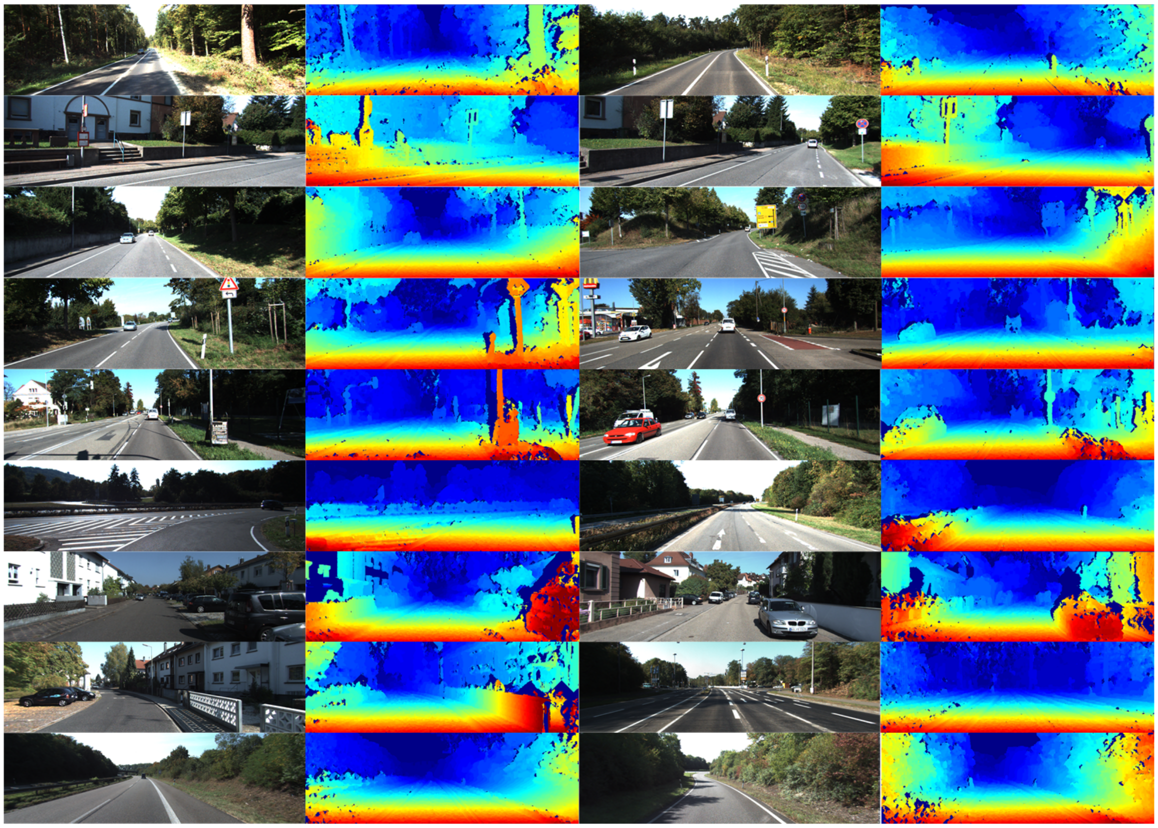


Fig. 6. Experimental results. $\rho = 3$ and $\tau = 1$. The first and third columns are the left frames. The second and fourth columns are the left disparity maps.

the texture objects are bound directly to the address of the global memory. In the μ_l, μ_r, σ_l and σ_r computation stage, the images are divided into a group of 32×8 thread blocks, and each of them are divided into eight warps in one SM to be processed by a set of SPs in parallel. After that, the disparities on row v_{max} are estimated in parallel using a set of 64×1 thread blocks. The image intensities i_l and i_r are also fetched from the texture references for $\sum i_l(u, v) i_r(u - d, v)$ calculation, whereas the values of μ_l, μ_r, σ_l and σ_r are indexed directly from the global memory to avoid unnecessary repeated computations. As for the disparity estimation for the rest of the map, the search range at the position of (u, v) is independent to their horizontal neighbouring disparities at $(u - 1, v)$ and $(u + 1, v)$, and only relies on the previously estimated disparities located on row $v + 1$. Therefore, the proposed algorithm is performed iteratively from row v_{max} to row v_{min} of the image, and each row is processed in parallel with the strategy in algorithm 2.

IV. Post Processing

For various disparity map estimation algorithms, the pixels that are only visible in one disparity map are a major source of the matching errors. Due to the uniqueness constraint of the correspondence, for an arbitrary pixel (u_i, v_i) in the left disparity map l^{lf}, there exists at most one correspondence in the right disparity map l^{rt}, namely [20]:

$$l^{lf}(u_i, v_i) = l^{rt}(u_i - l^{lf}(u_i, v_i), v_i) \qquad (10)$$

The LRC check is performed to remove half-occluded areas from the disparity map. Although the LRC check doubles the computational complexity by re-projecting the computed disparity values from one image to the other one, most of the incorrect half-occluded pixels can be eliminated and an outlier can be found [21]. For l^{rt} estimation, the memorisation of μ_l, μ_r, σ_l and σ_r is unnecessary because they have already been calculated when estimating l^{lf}. The LRC check is detailed in algorithm 3 and the results can be seen in Fig. 6.

Algorithm 3: Left-right consistency check

```
Data: left disparity map of size $m \times n$: $l^{lf}$
right disparity map of size $m \times n$: $l^{rt}$
Result: disparity map of size $m \times n$: $l$
for $v \leftarrow \rho + 1$ to $m - \rho - 2$ do
    for $u \leftarrow \rho + d_{max} + 1$ to $n - \rho - d_{max} - 2$ do
        $d_l \leftarrow l^{lf}(u, v)$;
        $d_r \leftarrow l^{rt}(u - d_l, v)$;
        if $abs(d_l - d_r) > threshold$ then
            $l(u, v) \leftarrow 0$;
        else
            $l(u, v) \leftarrow l^{lf}(u, v)$;
```

V. Experimental Results

In our previous publication [6], the performance of the proposed algorithm has already been evaluated by comparing the estimated disparity map with the ground truth from the KITTI database [12]–[15], where its overall percentage of the error pixels is approximately half of the rate obtained from GCS (When $\rho = 3$ and $\tau = 1$, the percentage of the absolute disparity error is 6.82%).

Platform	CPU threads	τ	ρ	fps
CPU	8	2	2	25
CPU	8	1	2	31
CPU	8	1	3	18
CPU	4	1	3	13
CPU	1	1	3	4
GPU	N/A	1	3	37

TABLE II
RUNTIME OF THE SYSTEM.

In this section, the performance of the implementation with different ρ and τ on Intel Core i7-4720HQ CPU and NVIDIA GTX 970M GPU is illustrated in Table II. When $\tau = 2$, the execution speed decreases by about 19% compared with the execution when $\tau = 1$; also, the runtime goes up with the increase of ρ. In addition, the performance of the implementation on the CPU with a single thread is about 4.9 times slower than the performance with eight threads processing the algorithm in parallel. Compared with the performance of the single-threading CPU, the GPU implementation is about nine times faster with 1280 CUDA cores estimating the disparities row by row from row v_{max} to row v_{min} of the image.

VI. CONCLUSION AND FUTURE WORK

This paper presented an efficient stereo matching algorithm and its real-time implementations on both i7-4720HQ CPU and GTX 970M GPU. The implementation was optimised on both algorithm level (NCC factorisation, integral images and search range propagation) and the parallel programming level (OpenMP and CUDA) to reduce the computational complexity, improve the accuracy of the disparity map estimation, and speed up the algorithm execution. The prevalent NCC algorithm was simplified as a dot product by factorising it as five independent parts where μ_l, μ_r, σ_l and σ_r are pre-calculated with the references from four integral images I_l, I_r, I_{l^2} and I_{r^2}, and their values are stored in static program storage for indexing. In addition, the search range at the position of (u, v) was propagated from three estimated neighbouring disparities $l(u-1, v+1)$, $l(u, v+1)$ and $l(u+1, v+1)$, which not only helps to minimise the ambiguities during the stereo matching and improve the accuracy of the disparity estimation, but also speeds up the execution by reducing the complicated and repeated computations of μ_l, μ_r, σ_l and σ_r. Also, the infeasible occlusions that are only visible in one disparity map were removed by conducting LRC check, which further improves the accuracy of the existing disparity map. The experimental results are better than what were obtained from the GCS. The implementations highly exploited the parallel computing architecture of OpenMP and CUDA to achieve real-time performance on both multi-threading CPU and GPU when processing the images with a resolution of 1242×375 from the KITTI database.

Our possible future work will focus on both accuracy and speed improvements of 3D information extraction for various autonomous vehicle applications. For instance, the subpixel resolution is planned to be achieved via a local quadratic polynomial interpolation. In addition, the shared memory on the GPU is planned to be used to speed up the stereo matching by sharing the data among threads within the same thread block instead of fetching them from the global memory.

REFERENCES

[1] James A Brink, Ronald L Arenson, Thomas M Grist, Jonathan S Lewin, and Dieter Enzmann, "Bits and bytes: the future of radiology lies in informatics and information technology," *European Radiology*, pp. 1–5, 2017.

[2] Beau Tippetts, Dah Jye Lee, Kirt Lillywhite, and James Archibald, "Review of stereo vision algorithms and their suitability for resource-limited systems," *Journal of Real-Time Image Processing*, vol. 11, no. 1, pp. 5–25, 2016.

[3] Zhen Zhang, *Advanced stereo vision disparity calculation and obstacle analysis for intelligent vehicles*, Ph.D. thesis, University of Bristol, 2013.

[4] Umar Ozgunalp, *Vision based lane detection for intelligent vehicles*, Ph.D. thesis, University of Bristol, 2016.

[5] Jan Cech and Radim Sara, "Efficient sampling of disparity space for fast and accurate matching," in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE, 2007, pp. 1–8.

[6] Zhen Zhang, Xiao Ai, and Naim Dahnoun, "Efficient disparity calculation based on stereo vision with ground obstacle assumption," in *21st European Signal Processing Conference (EUSIPCO 2013)*. IEEE, 2013, pp. 1–5.

[7] Umar Ozgunalp, Rui Fan, Xiao Ai, and Naim Dahnoun, "Multiple lane detection algorithm based on novel dense vanishing point estimation," *IEEE Transactions on Intelligent Transportation Systems*, vol. PP, pp. 1–12, 2016.

[8] U Ozgunalp, X Ai, Z Zhang, G Koc, and N Dahnoun, "Block-matching disparity map estimation using controlled search range," in *Computer Science and Electronic Engineering Conference (CEEC), 2015 7th*. IEEE, 2015, pp. 35–40.

[9] Chuan Lin, Ya Li, Guili Xu, and Yijun Cao, "Optimizing zncc calculation in binocular stereo matching," *Signal Processing: Image Communication*, vol. 52, pp. 64–73, 2017.

[10] Gabriele Facciolo, Nicolas Limare, and Enric Meinhardt-Llopis, "Integral images for block matching," *Image Processing On Line*, vol. 4, pp. 344–369, 2014.

[11] John P Lewis, "Fast template matching," in *Vision interface*, 1995, vol. 95, pp. 15–19.

[12] Moritz Menze and Andreas Geiger, "Object scene flow for autonomous vehicles," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3061–3070.

[13] 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*. IEEE, 2012, pp. 3354–3361.

[14] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[15] Jannik Fritsch, Tobias Kuhnl, and Andreas Geiger, "A new performance measure and evaluation benchmark for road detection algorithms," in *Intelligent Transportation Systems-(ITSC), 2013 16th International IEEE Conference on*. IEEE, 2013, pp. 1693–1700.

[16] Rui Fan, Victor Prokhorov, and Naim Dahnoun, "Faster-than-real-time linear lane detection implementation using soc dsp tms320c6678," in *Imaging Systems and Techniques (IST), 2016 IEEE International Conference on*. IEEE, 2016, pp. 306–311.

[17] Rohit Chandra, *Parallel programming in OpenMP*, Morgan kaufmann, 2001.

[18] Caio César Teodoro Mendes, Fernando Santos Osório, and Denis Fernando Wolf, "Real-time obstacle detection using range images: processing dynamically-sized sliding windows on a gpu," *Robotica*, pp. 1–16, 2015.

[19] CUDA Nvidia, "C programming guide version 4.0," *Nvidia Corporation*, 2011.

[20] Mikhail G Mozerov and Joost van de Weijer, "Accurate stereo matching by two-step energy minimization," *IEEE Transactions on Image Processing*, vol. 24, no. 3, pp. 1153–1163, 2015.

[21] Andrea Fusiello, Vito Roberto, and Emanuele Trucco, "Efficient stereo with multiple windowing," in *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*. IEEE, 1997, pp. 858–863.