Tulips R
University of Tsukuba Repository

Real-Time Stereo Vision System: A Multi-Block Matching on GPU

著者（英）	Qiong Chang, Tsutomu MARUYAMA
journal or publication title	IEEE Access
volume	6
page range	42030-42046
year	2018-07
権利	This work is licensed under a Creative Commons Attribution 3.0 License. For more information, see http://creativecommons.org/licenses/by/3.0/
URL	http://hdl.handle.net/2241/00157229

doi: 10.1109/ACCESS.2018.2859445

Creative Commons : 表示
http://creativecommons.org/licenses/by/3.0/deed.ja

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 14, 2018, accepted July 18, 2018, date of publication July 25, 2018, date of current version August 20, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2859445

Real-Time Stereo Vision System: A Multi-Block Matching on GPU

QIONG CHANG AND TSUTOMU MARUYAMA, (Member, IEEE)
Graduate School of Systems and Information Engineering, University of Tsukuba, Tsukuba 3058577, Japan
Corresponding author: Qiong Chang (cq@darwin.esys.tsukuba.ac.jp)

ABSTRACT Real-time stereo vision is attractive in many areas such as outdoor mapping and navigation. As a popular accelerator in the image processing field, GPU is widely used for the studies of the stereo vision algorithms. Recently, many stereo vision systems on GPU have achieved low error rate, as a result of the development of deep learning. However, their processing speed is normally far from the real-time requirement. In this paper, we propose a real-time stereo vision system on GPU for the high-resolution images. This system also maintains a low error rate compared with other fast systems. In our approach, the image is resized to reduce the computational complexity and to realize the real-time processing. The low error rate is kept by using the cost aggregation with multiple blocks, secondary matching and sub-pixel estimation. Its processing speed is 41 fps for 2888×1920 pixels images when the maximum disparity is 760.

INDEX TERMS Stereo vision, GPU, multi-block, real-time.

I. INTRODUCTION

The aim of stereo vision systems is to reconstruct the 3-D geometry of a scene from images taken by two separate cameras. It can be widely used in many areas like 3D-reconstruction [22], [23], robot vision [24], self-driving cars [25] and mechanical parts inspection. Many researchers have developed a wide range of effective algorithms, including CNN (Convolutional Neural Networks), Segmentation, BF (Belief Propagation), GC (Graph Cut) and Adaptive Cost Aggregation. Although most of them provide excellent accuracy, their high computational complexity often prohibits their application in real-time systems.

Many acceleration systems with FPGAs, GPUs and dedicated hardware have been developed [1]–[4]. All of them succeeded in real-time processing of high resolution images, but the maximum ranges of their disparities are less than 128. The larger the disparities are, increasingly complex the processing becomes. Middlebury Benchmark launched a new data set that consists of high resolution images with large disparity ranges (roughly 760), aiming to promote the development of stereo vision algorithms and systems for high resolution images. Recently, many algorithms on GPU have been developed. Sophisticated algorithms such as [5] and [6] have been implemented on GPU, and they showed very high accuracy. However, their processing speed for a high resolution image set is far from the real-time requirement. This is caused by the fact that data in several lines are accessed intensively in the stereo vision. The shared memory in GPU runs very fast, however, it is too small to hold all those lines. As a result, many memory accesses to the global memory is inevitable, and they limit the acceleration by GPUs. Therefore, the key for high performance on GPU, is to reduce the number of global memory accesses by efficiently sharing the data in the shared memory among the GPU cores.

In this paper, we aim to construct a real-time GPU stereo vision system for the high resolution image set (2888×1920 pixels$\times 760$ disparities) in the Middlebury Benchmark. In order to achieve the balance of the processing speed and the accuracy, we focus on proposing a series of methods to accelerate the multi-block matching (MBM) algorithm proposed by [9] on GPU.

The reason for choosing MBM is that it shows a good performance of processing speed and accuracy on CPU, and it can be easily combined with other methods to achieve a better performance, which makes our acceleration research more meaningful. In MBM, for each pixel, the normalized cross-correlation (NCC) is calculated by using a 3×3 window centered around the pixel, and the NCCs in several different size and shape blocks centered around the target pixel is added together. Then, the product of the sums is used to calculate the disparity of each pixel. This approach enables higher matching accuracy than the approaches that use only one block. MBM requires more computation, but the size of

This work is licensed under a Creative Commons Attribution 3.0 License. For more information, see http://creativecommons.org/licenses/by/3.0/

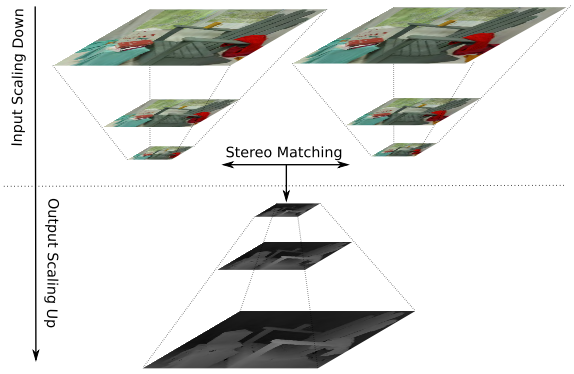

FIGURE 1. Image scaling.

data set that are intensively accessed is only slightly larger than that by single-block algorithms.

When we consider processing a high resolution image, its computational complexity is too high to achieve real-time processing by using MBM. In order to achieve high performance, in this paper, we propose the following methods:

Scaling the images: As shown in Fig.1, to reduce the computational complexity, we scale down the input images into a small size, and then scale up the disparity map to the original size. The images are scaled down to 1/4 (or 1/16) by reducing the width and height by half (or quarter), and the disparities are calculated on the scaled down image. The maximum disparity is also reduced to half (or quarter), which means that the total computational complexity can be reduced to 1/8 (or 1/64). This approach typically worsens the matching accuracy, but in our approach, an interpolation method based on the secondary matching is performed during the scaling up step, and a high matching accuracy can be maintained. This approach becomes possible because of the high quality of the high resolution images.

Making full use of on-chip memory: In addition to the shared memory, there are many registers on the GPU. The same as the shared memory, the registers can also be accessed at a high speed by the corresponding thread, but the data stored in them cannot be shared among the threads. In the stereo matching, some data of the reference image does not need to be accessed by other threads, so it becomes possible to store them in the registers. The data of the target image needs to be shared among the threads and are stored in the shared memory. By using the registers, it becomes possible to cache two-fold lines of images in the on-chip memory.

Reusing the intermediate results: To reduce the computational complexity, we reuse the intermediate results in two phases. First, in the stereo vision, the two input images are matched twice using one image as the reference and the other image as the target. In these two matchings, the matching costs calculated in the first matching can be reused in the second matching even though large memory is required to store them. The global memory of GPU is large enough for this purpose. Secondly, in the cost aggregation step, the matching costs of pixels in three different size blocks are aggregated. In this cost aggregation, the partial sums, such as the sum of the matching costs along the x-axis in smaller block, can be reused in the aggregation for the larger blocks.

By using these three methods, it becomes possible to achieve the real-time processing of the stereo vision of high resolution images. The processing speed of our system is faster than any other stereo vision systems on GPU and FPGA. The error rate evaluated using the Middlebury, KITTI2012 and KITTI2015 Benchmarks is at a medium level compared with those systems. Furthermore, we implemented our system on the Kepler series GPU Geforce GTX 780Ti and the Pascal series GPU Geforce GTX 1080Ti respectively. By using the Geforce GTX 780Ti GPU, the processing speed of our system has been achieved 18fps for high resolution image set (2888 $\times$ 1920 pixels $\times$760 disparities), while it has been improved to 41fps by using the GTX 1080Ti. These two implementations proved that our methods are suitable for different architecture GPUs.

This paper is organized as follows. Section II presents the background of our research and introduces the matching algorithms implemented on GPU. Section III describes our matching method. The detailed description of our GPU implementation is presented in Section IV. Finally, the accuracy and the processing speed are evaluated in Section V. Section VI concludes this paper.

II. BACKGROUND

Recently, many algorithms for stereo vision have been developed. These algorithms can be categorized into two groups: global algorithms and local algorithms.

In the global algorithms, the disparities of all pixels are decided by the mutual effect of all pixels. Thus, the global algorithms achieve lower error rates, but require longer computation time. References [1] and [8] implemented the minimal spanning tree (MST) and dynamic programming (DP) on FPGA respectively. They achieved the real-time processing (30fps) for the high resolution images, but all of their disparities are smaller than 64, which is not suitable to the modern requirements. Reference [10] implemented a Recurrent Neural Network (RNN) aggregation method on a high-end GPU Geforce GTX Titan X, and [11] implemented a Semi-Global Matching (SGM) method on a low-end embedded GPU Tegra X1. In both of these two GPU systems, the global algorithms are processed in real-time. However, the input image size of [10] is 1242 $\times$ 375 pixels, while that of [11] is 640 $\times$ 480 pixels. None of which are large enough. Additionally, according to the evaluation results in the KITTI Benchmark 2015, the error rates of RNN [10] (6.34%) and the SGM [11] (8.24%) is higher than the MBM [9] (6.04%), which is used in this paper, because both of them need to simplify the algorithms to fit the hardware architecture.

In the local algorithms, only the local information around the target pixel is used to decide the disparity of the pixel. This feature is suitable for their acceleration on GPUs because the small and fast memory on GPU works efficiently owing to

the high locality of the memory access. Cross-Based Aggregation (CROSS) is one of the most effective local algorithms. References [12] and [13] implemented the CROSS on GPU and FPGA, and achieved real-time processing respectively. Although both of them achieved a low error rate, their input sizes are smaller than 640 × 480. Reference [13] also runs their system for the high resolution images of Middlebury Benchmark. However, their speed is slower than 3fps for the large image set (2888 × 1920 pixels×760 disparities). Reference [4] implemented the Segmentation method on GPU for high resolution images, which is used in the cost aggregation step. The same as [10] and [11], it also simplified the original algorithms in order to implement it on GPU, which resulted in decreased accuracy.

As described previously, the existing real-time stereo vision systems face limitations such as small image size or low accuracy. None of the them can obtain a good balance.

III. OUR ALGORITHM

In our algorithm, to achieve the real-time processing of high resolution images, and low error rate for them,

1) the two input images are gray-scaled,
2) the two images are scaled down,
3) the Normalized Cross-Correlation (NCC) is calculated as the matching cost of each pixel,
4) they are aggregated using three different shape and size blocks,
5) the Multi-Block Matching (MBM) is applied, and a disparity map is generated,
6) the secondary matching, and then a disparity map is scaled up by the bilateral estimation,
7) step 4) to 6) are executed again to obtain two disparity maps using left and right image as the reference,
8) Ground control point (GCPs) are chosen using the two disparity maps, and
9) the disparity map is improved using the GCPs.

A. SCALING DOWN

In order to reduce the computational complexity, the two images are scaled down linearly in both horizontal and vertical directions using the mean-pooling method. Here, take the left image as an example:

$$L(x,y) = \frac{1}{(2m+1)^2} \times \sum_{j=-m}^{m} \sum_{i=-m}^{m} L_{org}(K \cdot x + i, K \cdot y + j) \tag{1}$$

where $L_{org}(K \cdot x + i, K \cdot y + j)$ is the pixel in the original image, and K is the factor for the scaling down. $L(x, y)$ is the pixel of the left scaled down image, and is smoothed by a mean-filter the size of which is $(2m + 1)^2$. By choosing the block size and the pooling stride carefully, we can avoid the loss of the matching accuracy, and can improve the processing speed.

B. MATCHING COST COMPUTATION

The matching cost of each pixel is calculated using the NCC. When the left image L is used as the reference, the matching cost of $L(x, y)$ and $R(x - d, y)$ is given by

$$C_L(x,y,d) = \frac{\sum_{x',y' \in S_p} (L(x',y') - \bar{L}(x,y))(R(x'-d,y') - \bar{R}(x-d,y))}{|S_p| \cdot \sigma_L(x,y) \cdot \sigma_R(x-d,y)} \tag{2}$$

where

$$\bar{L}(x,y) = \frac{1}{|S_p|} \sum_{x',y' \in S_p} L(x',y'), \tag{3}$$

and

$$\sigma_L(x,y) = \sqrt{\frac{1}{|S_p|} \sum_{x',y' \in S_p} (L(x',y') - \bar{L}(x,y))^2} \tag{4}$$

In this equation, S_p is the window used to calculate the NCC. $L(x', y')$ and $R(x' - d, y')$ are the pixels in each NCC window, while $\bar{L}(x, y)$ and $\bar{R}(x - d, y)$ are the averages of them. $\bar{R}(x - d, y)$ and $\sigma_R(x - d, y)$ are given in the same way as $\bar{L}(x, y)$ and $\sigma_L(x, y)$.

When the right image R is used as the reference, the matching cost is given as follows:

$$\begin{aligned} C_R(x,y,d) &= \frac{\sum_{x',y' \in S_p} (R(x',y') - \bar{R}(x,y))(L(x'+d,y') - \bar{L}(x+d,y))}{|S_p| \cdot \sigma_L(x+d,y) \cdot \sigma_R(x,y)} \\ &= C_L(x+d,y,d). \end{aligned} \tag{5}$$

This equation means that all C_R are already calculated when C_L were calculated, and $C_L(x, y, d)$ can be reused as $C_R(x - d, y, d)$.

For this cost calculation using NCC, the integral images can be used to reduce the computational complexity of ((2)). However, "long" or "double" (8B width) is required for the arrays to store the integral images, and their data size becomes the double of our calculation method using "float" (4B width). With this increase of the data size, only one row can be processed at the same time (as described in Section IV, it is possible to process two rows at the same time without accessing global memory in the implementation using "float"), and the total processing speed becomes worse.

C. COST AGGREGATION

The standard Block-Matching (BM) approach is widely used in the matching cost aggregation step. In this approach, the matching costs of the pixels in a rectangular block centered at the target pixel are added, and the sum is used as the matching cost of the target pixel:

$$C_{LB}(x,y,d) = \sum_{(x',y') \in b(x,y)} C_L(x',y',d). \tag{6}$$

where $b(x, y)$ is a rectangular block centered at (x, y).

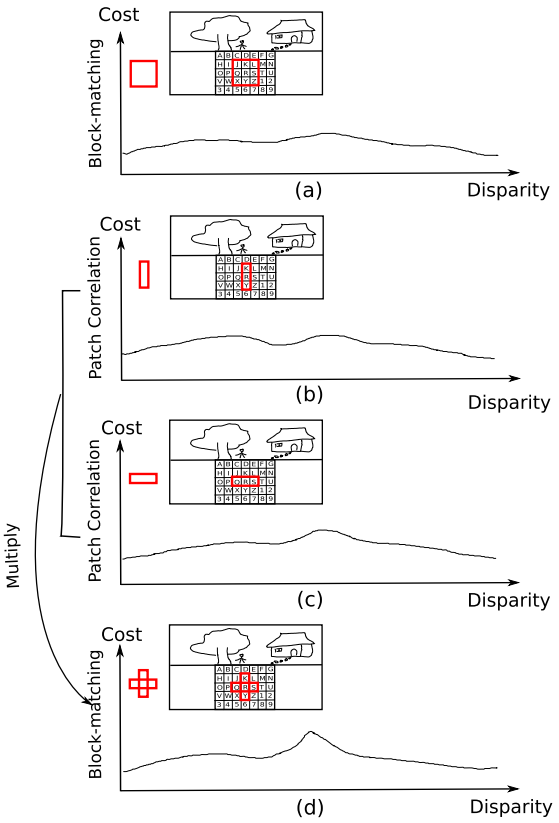

FIGURE 2. Multi-block matching.

However, the standard BM usually generates disparity maps with strong fattening. In order to obtain better disparity maps, [9] proposed the MBM approach. It combines matching blocks of different shapes and sizes and has shown significant improvement compared to the standard BM. In the MBM, a multiplicative function of block set B is used to calculate the matching cost of each pixel:

$$C_{LMBM}(x, y, d) = \prod_{b \in B} C_{LB}(x, y, d). \tag{7}$$

where $C_{LB}(x, y, d)$ is the matching cost calculated by standard BM for a block b.

Fig.2(a) shows the cost aggregation in the standard BM. The matching costs of the pixels in the block are simply added. This approach often lacks the sensitivity because of the cost averaging in the large block. Fig.2(b) and (c) show the matching costs obtained when a vertical and horizontal block are used. In this example, as shown in Fig.2(c) a horizontal bock shows higher sensitivity because the texture changes sharply along the horizontal direction in the image. In this case, in the original BM, the sensitivity that could be obtained by the aggregation along only the x-axis is lost by the averaging of the pixels in the square block. However, it is preserved in the MBM in which the aggregated cost of each block is multiplied as the final matching cost (Fig.2(d)).

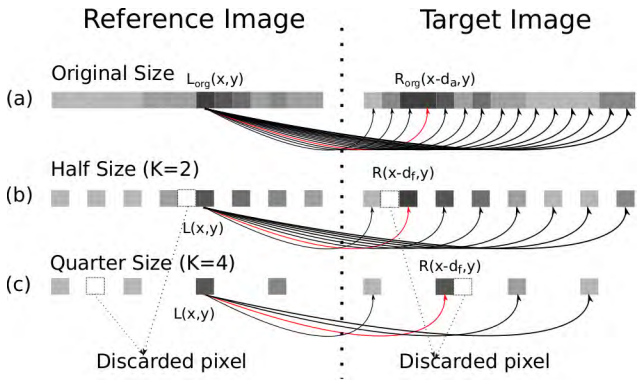

FIGURE 3. Stereo-matching in different scale.

Finally, the disparity of each pixel $D_{LMBM}(x, y)$ is decided by Winner-Take-All (WTA) as follows:

$$D_{LMBM}(x, y) = \max_{d} C_{LMBM}(x, y, d). \tag{8}$$

Here, disparity d is in the range of $[0, D'_{max}]$. D'_{max} is the maximum of the disparity of the scaled down images as following:

$$D'_{max} = \frac{D_{max}}{K}. \tag{9}$$

D_{max} is the maximum of the disparity of the original images and K is the scale factor described above.

D. SECONDARY MATCHING AND SCALING UP

As described above, in order to reduce the computation complexity, the input images are scaled down to be processed. Fig.3 shows the stereo matching along one line on different scaling images. In Fig.3, all images on the left are the reference images, and those on the right are the target images. In Fig.3(a), the matching is performed on the original size images L_{org} and R_{org}, and an accurate disparity (d_a in this figure) can be found for each pixel $L_{org}(x, y)$. Fig.3(b) and (c) show the matching on the scaled down images. In these cases, only a fuzzy disparity (d_f) can be found due to the following factors:

1) During the scaling down step, part of the pixels in the target images R are discarded, illustrated by the blank space.
2) During the scaling up step, the disparity of each discarded pixel can only be estimated by using the disparities of the retained pixels, which may not be consistent with the truth value.

Accordingly, in order to improve the accuracy, we propose two methods to deal with these issues: a secondary matching method to find an accurate disparity for each pixel in the scaled down images, and a bilateral estimation method to fill in the disparities for the discarded pixels during the scaling up step.

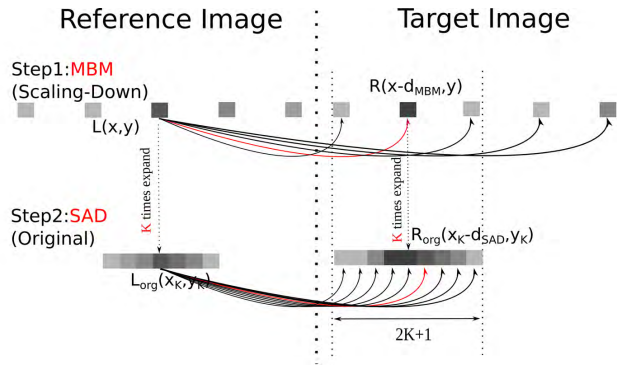

FIGURE 4. Secondary matching.

1) SECONDARY MATCHING

As the analysis above, the accurate disparity d_a cannot be found by using the scaled down images. According to ((9)), K times the fuzzy disparity d_f is closest to the accurate disparity d_a and in most cases, d_a should be in $(K \cdot (d_f - 1), K \cdot (d_f + 1))$. Based on this assumption, in order to obtain a higher accuracy, after the MBM matching, a secondary stereo matching is performed.

As shown in Fig.4, for the pixel $L(x, y)$ in the scaled down image ($K = 4$), when the fuzzy disparity d_{MBM} is found by using the MBM, the corresponding pixel $L_{org}(x_k, y_k)$ in the original image ($x_k = K \cdot x$ and $y_k = K \cdot y$) is matched again with the original size target image R_{org}. Correspondingly, the discarded pixels in the range of $(K \cdot (d_{MBM} - 1), K \cdot (d_{MBM} + 1))$ are retrieved and used to find the accurate disparity. For this secondary matching, although many matching methods can be used such as MBM, in order to reduce the amount of computation, the simple matching method SAD [15] is chosen and the accurate disparities are found as following:

$$C_{LSAD}(x_k, y_k, d) = |S_q| \cdot I_{vol} - \sum_{x_k, y_k \in S_q} |L_{org}(x_k, y_k) - R_{org}(x_k - d, y_k)| \tag{10}$$

and

$$D_{LSAD}(x_k, y_k) = \max_d C_{LSAD}(x_k, y_k, d). \tag{11}$$

In ((10)), $C_{LSAD}(x_k, y_k, d)$ is the matching cost of pixel $L_{org}(x_k, y_k)$. S_q is the window used to calculate the cost of SAD, and I_{vol} is the maximum value in the gray-scale images (in our implementation, simply, 255 is used as I_{vol}). In ((11)), $D_{LSAD}(x_k, y_k)$ is the disparity of $L_{org}(x_k, y_k)$ that maximizes the value of $C_{LSAD}(x_k, y_k, d)$.

Fig.5 shows how the disparities are fine-tuned by using the secondary matching. First, Fig.5(a) shows a typical case of two matching costs for pixel $L(x, y)$; the matching cost by MBM and that by SAD. In this figure, d_{MBM} gives the best matching cost $C_{LMBM}(x, y, d_{MBM})$, and in the range of $d_{MBM} \pm 1$, the secondary matching costs $C_{LSAD}(x, y, d)$ by

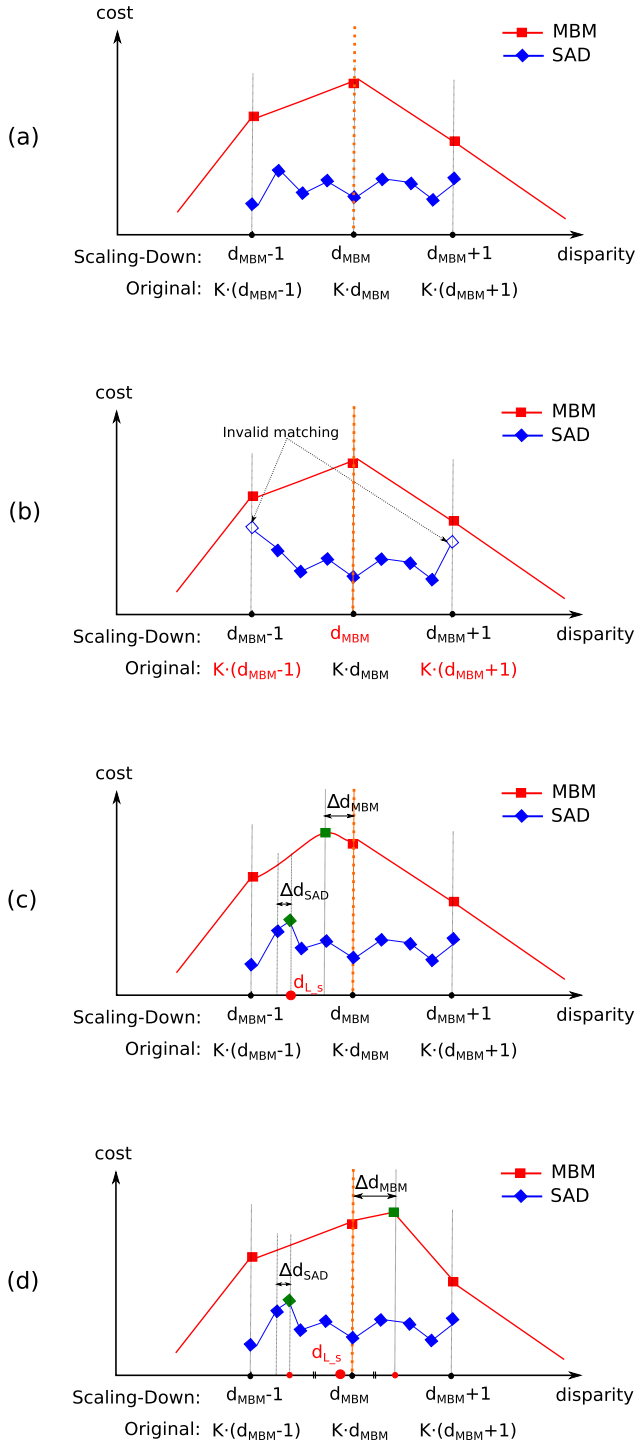

FIGURE 5. Fine-tune. (a) Normal matching. (b) Invalid matching: the result of sad is inconsistent with MBM. (c) Valid matching: the results on the same side. (d) valid matching: the results on different sides.

SAD are calculated ($K = 4$ in this example). The x-axis represents the disparities for the scaled down and original size image. The disparity $d_{\text{MBM}} \pm 1$ in the scaled down image corresponds to $K \cdot (d_{\text{MBM}} \pm 1)$ in the original size image. In order to ensure the robustness of this method, it is

necessary to confirm that for each pixel, whether the result by SAD is consistent with that by MBM or not. If it is consistent, d_{SAD}, the disparity that maximizes $C_{LSAD}(x, y, d)$, should be between $(K \cdot (d_{MBM} - 1), K \cdot (d_{MBM} + 1))$, but should not be on $K \cdot (d_{MBM} - 1)$ or $K \cdot (d_{MBM} + 1)$, because $C_{MBM}(x, y, d_{MBM} \pm 1)$ is smaller than $C_{MBM}(x, y, d_{MBM})$. In Fig.5(a), this requirement is satisfied, but it is not in Fig.5(b). In the former case, the disparities are fine-tuned as described below. but in the later case, the matching cost by SAD is not used for fine-tuning, and d_{MBM} is used as the result.

In our approach, to fine-tune the disparities, a sub-pixel estimation method proposed by [14] is used. In [14], it is supposed that the curve of the matching costs is continuous on the disparities and each small fragment of the matching costs can be approximated by a quadratic function. In our system, using $C_{LMBM}(x, y, d_{MBM})$ and $C_{LMBM}(x, y, d_{LMBM} \pm 1)$, a quadratic function that goes through these three points is calculated, and the distance from d_{MBM} to the disparity that gives the peak to the quadratic function, Δd_{MBM}, is obtained. In the same way, a quadratic function that goes through $C_{LSAD}(x_k, y_k, d_{SAD})$ and $C_{LSAD}(x_k, y_k, d_{SAD} \pm 1)$ is calculated, and the distance from d_{SAD} to the disparity that gives the peak to the quadratic function, Δd_{SAD}, is obtained. Then, the disparity is fine-tuned as follows.

1) If Δd_{MBM} and $d_{SAD} + \Delta d_{SAD} - K \cdot d_{MBM}$ have the same sign as shown in Fig.5(c), which means that both disparities are in the same side of the center line (red dotted line in Fig.5(c),(d)), the new disparity d_{L_S} for pixel $L(x, y)$ is calculated as $(d_{SAD} + \Delta d_{SAD})/K$.
2) If Δd_{MBM} and $d_{SAD} + \Delta d_{SAD} - K \cdot d_{MBM}$ have the different signs as shown in Fig.5(d), which means that the two disparities are in different side of the center line, the new disparity d_{L_S} for pixel $L(x, y)$ is calculated as the average of them $(d_{MBM} + \Delta d_{MBM} + (d_{SAD} + \Delta d_{SAD})/K)/2$.

During this matching step, although the matchings are performed twice, as the disparity range is limited in the secondary matching, only $D_{max}/K+2K+1$ matches are required for each pixel. Hence, not only an accurate matching can be ensured, but also the amount of the computation is kept small.

2) BILATERAL ESTIMATION

As a result of the sub-pixel estimation, the disparity map D_{L_S} of the scaled down images is generated. In order to revert D_{L_S} to the original size map D_L, the disparities of the discarded pixels (the blank pixels) in Fig.3(b) and (c) are regenerated by the following steps:

1) Set the disparity $D_L(x_k, y_k)$ to $K \cdot D_{L\_S}(x, y)$.
2) Fill the disparity of the discarded pixels $L_{org}(x_k + i, y_k)$ along the x-axis, where $0 < i < K$.
   a) If $|D_L(x_k, y_k) - D_L(x_k + K, y_k)| \leq T$, where T is the threshold for the difference of disparity, it can be considered that the disparity is changing continuously in this range, and $D_L(x_k + i, y_k)$ is filled as $D_L(x_k, y_k) + i \cdot ((D_L(x_k, y_k) - D_L(x_k + K, y_k))/K)$.
   b) If $|D_L(x_k, y_k) - D_L(x_k + K, y_k)| > T$, which means that the disparity changes rapidly in this range, it can be considered that an edge exists in this range. Thus, $D_L(x_k, y_k)$ is chosen as $D_L(x_k + i, y_k)$ if $L_{org}(x_k + i, y_k)$ is closer to $L_{org}(x_k, y_k)$ than $L_{org}(x_k + K, y_k)$ in color, and otherwise, $D_L(x_k + K, y_k)$ is chosen as $D_L(x_k + i, y_k)$.
3) Fill the disparity of the discarded pixels $L_{org}(x_k, y_k + j)$ along the y-axis, where $0 < j < K$.
   a) If $|D_L(x_k, y_k) - D_L(x_k, y_k + K)| \leq T$, where T is the same as above, $D_L(x_k, y_k + j)$ is filled as $D_L(x_k, y_k) + j \cdot ((D_L(x_k, y_k) - D_L(x_k, y_k + K))/K)$.
   b) If $|D_L(x_k, y_k) - D_L(x_k, y_k + K)| > T$, in the same way as above, we choose $D_L(x_k, y_k)$ or $D_L(x_k, y_k + K)$ which is similar to $L_{org}(x_k, y_k + j)$ in color, and it is copied to $D_L(x_k, y_k + j)$.
4) Fill the disparity of other discarded pixels $L_{org}(x_k + i, y_k + j)$, where $0 < i < K$ and $0 < j < K$. After step 3), for each j, the disparities of $L_{org}(x_k, y_k + j)$ and $L_{org}(x_k + K, y_k + j)$ (these two pixels are on the same line) are fixed. Then, the disparities of the pixels on this line are filled according to step 2).

With this approach, an accurate disparity map D_L can be estimated.

E. CROSS_CHECK

In our approach, $D_R(x_k, y_k)$ is also calculated in the same way as $D_L(x_k, y_k)$. Then, the ground control points (GCPs), are obtained by comparing them [17]. Here, suppose that d is the disparity obtained for $L_{org}(x_k, y_k)$, when the left image is used as the reference. This means that between the two images, $L_{org}(x_k, y_k)$ and $R_{org}(x_k - d_{int}, y_k)$ shows the best matching (d_{int} is an integer rounded to the closest one), and they are the same point of an object in the images. Therefore, $D_R(x_k - d_{int}, y_k)$ should also be $[d_{int} - 0.5, d_{int} + 0.5]$. If this requirement is satisfied, the point is called a GCP, and it is considered that GCPs have high reliability.

F. DISPARITY MAP IMPROVEMENT

Ideally, all pixels except for those in the occluded regions should be GCPs, however, in actuality more pixels become non-GCPs because of the slight change of the brightness between the input images. To achieve more reliable disparities of non-GCPs, two approaches are often used [14]. In both approaches, for each non-GCP, the closest GCPs on the left and right hand-side along the x-axis are searched first. Then, in the first approach, as shown in Fig.6(a), the closer GCP in the distance is chosen as the disparity of the non-GCP because the non-GCP and the closer GCP can be considered to belong to the same object with a higher probability. In the second approach, as shown in Fig.6(b), the smaller disparity is chosen as the disparity of the non-GCP assuming that the non-GCP is caused by the occlusion. The disparity of the

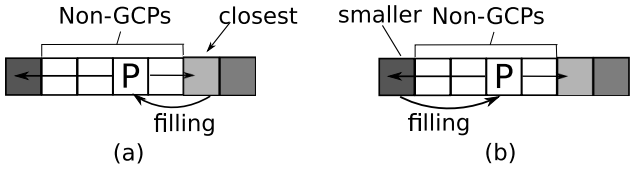

FIGURE 6. Non-GCPs filling. (a) Method 1. (b) Method 2.

occluded region is smaller than that of the foreground object because the disparity of the closer object is larger, and the non-GCP should have a smaller disparity. In our system, in order to reduce the memory usage, the second approach is used to improve the disparity map.

IV. IMPLEMENTATION ON GPU

In this section, we discuss the implementation of our MBM algorithm on GPU.

A. THE GPU ARCHITECTURE

We implement our algorithm on both NVIDIA GTX780 Ti and GTX1080 Ti GPUs, which have the different architectures. Although the number of streaming multi-processors (SMs) and the memory sizes are different, the hierarchy and the attribute of the memories have not any changed. Each SM has two types of memory: register memory and shared memory. The sizes of them are usually limited, but their access delay is very short. However, the data cached in the register memory is visible only to the thread that wrote them and last only for the lifetime of that thread, while the data cached in the shared memory are visible to all threads in the same SM and last for the duration of the kernel function which declared the threads. Each GPU also has the global memory, constant memory, and texture memory in the off-chip. The global memory is usually used to hold all data that are required for the processing, and the constant memory and texture memory are usually used to reduce the memory traffic to the global memory. By the reason of the long access delay to the off-chip memory, the most important technique to achieve high performance on GPU is how to cache a part of the data on the on-chip memory, and to hide the memory access delay to the global memory. The shared and the global memory have the restriction on the access to them. In the CUDA, which is an abstracted architecture of NVIDIA GPUs, 32 threads are managed as a set. When accessing the global memory, 32 words can be accessed in parallel if the 32 threads access to continuous 32 words on 32 word-boundary. Otherwise, the bank conflict happens, and several accesses to the global memory are issued. The shared memory consists of 32 banks, and the 32 words can be accessed in parallel if they are placed in different banks (the addresses of the 32 words do not need to be continuous).

B. SYSTEM PIPELINE

Fig.7 shows the pipeline of our stereo vision system. The original color images are first converted to gray-scale on

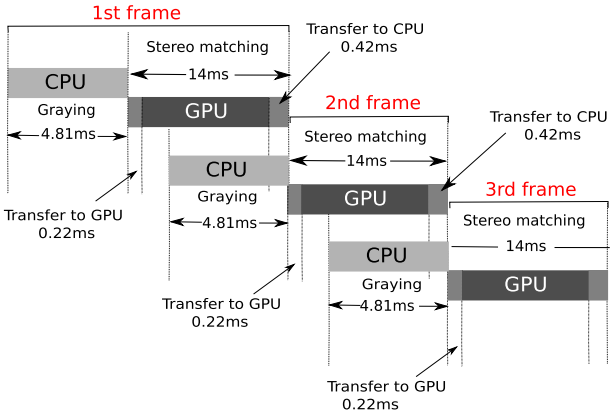

FIGURE 7. System pipeline.

CPU, and then they are transferred to the global memory of GPU. The data size can be reduced to 1/3 (24bit to 8bit), and their transfer time also can be reduced. In our system, taking the GTX1080 Ti as an example, for the 2888 $\times$ 1920 pixels images, the processing on CPU takes about 4.81 milliseconds, and the processing on GPU takes about 13.7 milliseconds. Although the delay for the first frame takes more than 18 milliseconds, as shown in Fig.7, the calculation on the two devices can be run in parallel and the stream processing makes it possible to achieve the real-time processing as its throughput.

Since the main processing of our system is on the GPU side, we focus on how to implement our system on GPU.

C. TASK ASSIGNMENT AND DATA MAPPING ON GPU

Because of the high locality of the MBM algorithm, there exist many alternatives for how to process the pixels in parallel by using many cores on GPU. In our implementation, as shown in Fig.8, the image is divided by N along the y-axis. Here, Y is the height of the image and N is the number of the SMs of the target GPU. As shown in Fig.8, Y/N lines are assigned to each SM. In each SM, Y/N lines are processed from the top to bottom line by line. In this line by line processing, one pixel is assigned to one thread as shown in Fig.8-top and the pixels on the same line are processed in parallel. In our implementation, the whole work-flow is divided into five steps as shown in Fig.8, and in each step, the outputs of the previous step are fetched from the global memory into the on-chip memory, and the outputs of the current step are sent to the global memory for the next step. By reducing the number of steps, higher processing speed is expected because the number of the memory accesses to the global memory can be reduced. However, the size of our target image is large, and only the data required for processing one line can be held on on-chip shared memory and registers. Under this limitation, the five steps are the minimum set. In these five steps, the procedures that can be performed by using the data of the same line are packed in the same step

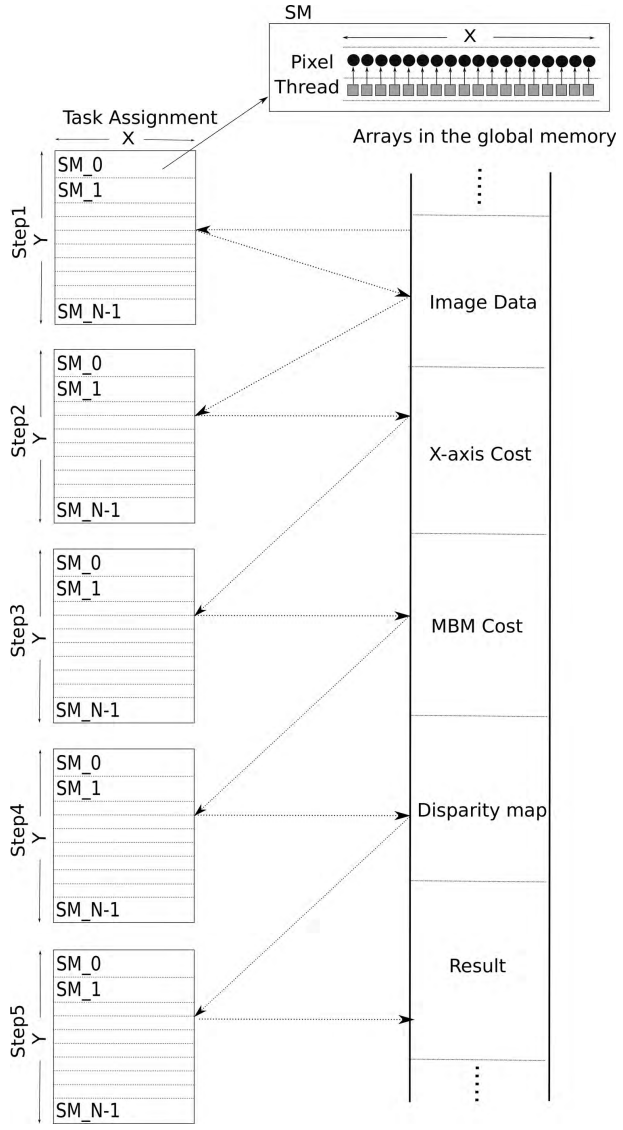

FIGURE 8. Task assignment to each step on GPU. *Step1:* Smoothing & Scaling Down. *Step2:* NCC calculation & Cost Aggregation along the *x*-axis. *Step3:* Cost Aggregation along the *y*-axis. *Step4:* WTA, Secondary Matching & Scaling Up. *Step5:* Cross-Check & Improvement.

such as the calculation of NCC cost and their aggregation along the x-axis in the step 2.

D. EFFECTIVE MATCHING PROCESSING ON GPU

The most time consuming steps in our algorithm are step 2 and step 3, because in these steps, D matching costs are calculated. In order to achieve high performance, the optimization of these two steps is highly important. The following describes the details of our methods to improve the performance of these two steps.

1) MAKING FULL USE OF ON-CHIP MEMORY

Each SM in GPU has on-chip shared memory and registers in it. By using them efficiently, the access to global memory can be reduced. The data in the shared memory can be accessed

TABLE 1. Memory sharing in the Ncc cost calculation.

Data Type	Data Size	
	Shared Memory	***Register***
Image Data	$768\times3\times4\times16(bit)=18(KB)$	
Ave(T)	$768\times2\times32(bit)=6(KB)$	
SD(T)	$768\times2\times32(bit)=6(KB)$	
Ave(R)		$768\times2\times32(bit)=6(KB)$
SD(R)		$768\times2\times32(bit)=6(KB)$
Costtemp	$768\times2\times32(bit)=6(KB)$	
Result	$768\times2\times32(bit)=6(KB)$	
Total	***42KB***	***12KB***

Image Data: *The image data with integer type which are generated in the step1 (Both left and right image).* ***Ave***: *The average of the image data.* ***SD***: *Standard deviation of the image data.* ***T***: *Target image.* ***R***: *Reference image. The "Ave" and the "SD" of the target image are stored in shared memory. On the other hand, those of the reference images are stored in the register.* ***Costtemp***: *The result of the NCC.* ***Result***: *the costs aggregated along the x-axis.*

from any threads in the same SM, but the data in the registers can be accessed only by the thread that wrote them in the registers. Table.1 shows the memory usage in the NCC calculation step. In this step, it is necessary to store 3×2 lines of the reference (left) and target (right) image in the shared memory. $\sigma_L(x,y)$, $\bar{L}(x,y)$, $\sigma_R(x-d,y)$ and $\bar{R}(x-d,y)$ are used for calculating NCC as described in Section III-B, and each of them is accessed D times for calculating all NCCs. During the computations, $\sigma_L(x,y)$ and $\bar{L}(x,y)$ are accessed by only one thread, while $\sigma_R(x-d,y)$ and $\bar{R}(x-d,y)$ are accessed from D threads. Thus, $\sigma L(x,y)$ and $\bar{L}(x,y)$ can be stored in the on-chip registers. As shown in Table.1, the intermediate results for calculating NCCs (Costtemp and Result) are stored in the shared memory and the total size of the data stored in the shared memory reaches 42KB for calculating one line. This means that data for one line in the original size image cannot be stored in the shared memory, and reducing the image size is a must for the efficient computation by using only the on-chip memory.

In our implementation, the NCCs are calculated line by line. First, two sets of three lines $y-1$, y and $y+1$ are held in the shared memory, and the NCC costs of the center line y are calculated. Then, the pixels of the next line $y+2$ are fetched from the global memory, and the pixels of the oldest line $y-1$ are replaced by the new ones. Using the new line $y+2$ and the two lines that are already held on the chip y and $y+1$, the NCC costs of the next line $y+1$ are calculated.

2) REUSING THE INTERMEDIATE RESULTS

After the NCC cost calculation, they are aggregated using the MBM algorithm. As described in Section IV-C, the NCC costs of the next line cannot be calculated at the same time, which means that the NCC cost cannot be aggregated along the y-axis without using the global memory. Thus, two steps are required for the cost aggregation. In our MBM, three blocks with different size and shape are used. By choosing the block width properly, the intermediate results of the cost

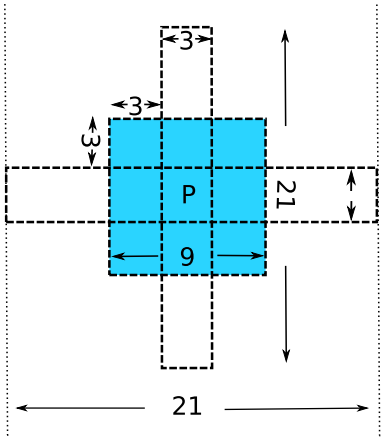

FIGURE 9. Multi-block matching.

aggregation for the small blocks can be reused for larger blocks. Fig.9 shows a set of the blocks, the sizes of which are 3×21, 21×3, 9×9 respectively. With this combination, the partial sums calculated for the smallest block 3×3 can be reused for other blocks. Additionally, according to our evaluation, this combination also shows a good accuracy. By using larger blocks, higher matching accuracy can be expected, but according to our experience, the improvement is marginal though it requires more operations (addition) which may effect the processing speed.

Fig.10(a) shows the cost aggregation along the x-axis. For pixel $L(x, y)$, three cost aggregation steps are taken. First, the costs of itself $C(x, y, d)$ and its two neighbors, $C(x-1, y, d)$ and $C(x+1, y, d)$, are aggregated by the corresponding thread. Then, its sum $C_3(x, y, d)$ is kept in the shared memory to be reused for other size blocks. Secondly, each thread aggregates $C_3(x, y, d)$, $C_3(x-3, y, d)$ and $C_3(x+3, y, d)$, and the $C_9(x, y, d)$ is obtained. $C_9(x, y, d)$ is stored into the register instead of the shared memory because it is not accessed from other threads in the following steps. Thirdly, $C_{21}(x, y, d)$ is calculated by adding $C_9(x, y, d)$ and its four neighbors, $C_3(x \pm 6, y, d)$ and $C_3(x \pm 9, y, d)$. All the costs are stored in the global memory to be used for the aggregation along the y-axis. This sequence is repeated D times ($d = [0, D)$) for each pixel.

Fig.10(b) shows the layout of the partial sums along the x-axis in the global memory. For each pixel, the cost of each disparity is stored in the global memory in the order that the next aggregation step runs efficiently. Fig.10(c) shows the costs aggregation along the y-axis. Before entering this step, the cost aggregation of all lines along the x-axis is finished, and the all sums are stored in the global memory. For each pixel $L(x, y)$, its three aggregation costs are given as follows:

$$C_{21\times3}(x, y, d) = \sum_{i=-1}^{1} C_{21}(x, y+i, d) \tag{12}$$

$$C_{9\times9}(x, y, d) = \sum_{i=-4}^{4} C_9(x, y+i, d) \tag{13}$$

$$C_{3\times21}(x, y, d) = \sum_{i=-10}^{10} C_3(x, y+i, d) \tag{14}$$

In these equations, C_{21}, C_9, C_3 are transferred from the global memory to the shared memory and are aggregated line by line. Here, suppose that $C_{21\times3}(x, y, d)$, $C_{9\times9}(x, y, d)$ and $C_{3\times21}(x, y, d)$ are held on the shared memory. Then, the three sums for the next line can be calculated efficiently as follows:

$$\begin{aligned} &C_{21\times3}(x, y+1, d) \\ &= \sum_{i=0}^{2} C_{21}(x, y+i, d) \\ &= C_{21\times3}(x, y, d) + C_{21}(x, y+2, d) - C_{21}(x, y-1, d) \end{aligned} \tag{15}$$

$$\begin{aligned} &C_{9\times9}(x, y+1, d) = \sum_{i=-3}^{5} C_9(x, y+i, d) \\ &= C_{9\times9}(x, y, d) + C_9(x, y+5, d) - C_9(x, y-4, d) \end{aligned} \tag{16}$$

$$\begin{aligned} &C_{3\times21}(x, y+1, d) = \sum_{i=-9}^{11} C_3(x, y+i, d) \\ &= C_{3\times21}(x, y, d) + C_3(x, y+11, d) - C_3(x, y-9, d) \end{aligned} \tag{17}$$

By this calculation method, the computation order of the aggregation along the y-axis can be reduced to O(1).

E. SUBSEQUENT PROCESSING ON GPU

After MBM, a series of processing shown in Fig.8 are executed line by line. The task assignment to the threads is the same as the NCC cost calculation step, and all of the steps are executed one by one in each line:

1) The costs of MBM are transferred from the global memory to the shared memory repeatedly, and two initial disparity lines D'_{LMBM} and D'_{RMBM} are generated by the WTA. During the WTA, when the matching cost for $d_c + 1$, namely $C(x, y, d_c + 1)$, is calculated, $C(x, y, d_c - 1)$ and $C(x, y, d_c)$ are being held on the registers, and the sub-pixel estimation is performed for d_c. With this implementation, when the integer disparity d_{MBM} which gives the maximum matching cost is obtained, an offset Δd_{MBM} is also obtained through the sub-pixel estimation.
2) Based on d_{MBM}, the secondary matching is executed in the range of $[K \cdot (d_{MBM} - 1), K \cdot (d_{MBM} + 1)]$. The same as the MBM, in the SAD matching, one thread corresponds to one pixel. For each pixel, d_{SAD} and Δd_{SAD} are calculated by using the same method as 1). Unlike the first matching, in the secondary matching, several lines of original image need to be transferred to the on-chip memory, and the amount of the data becomes usually several times the data that were used in the first matching. By using more data in this step, higher matching accuracy can be expected, but it requires more arithmetic operations and more memory space. Hence, according to the limit of the hardware resources and the processing speed, choosing a suitable amount of data is very important. Here, one thing to note is that

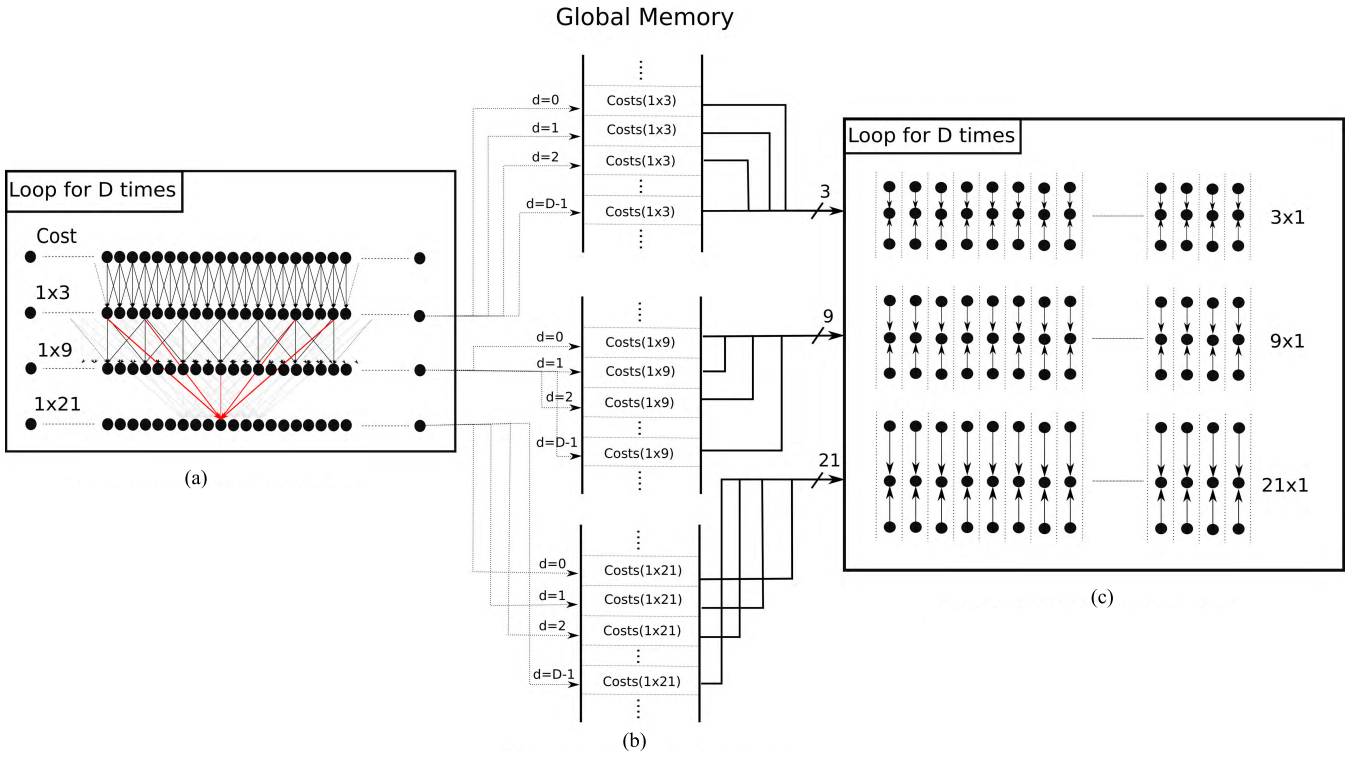

FIGURE 10. System pipeline. (a) Cost aggregation along x-axis. (b) Layout of the global memory. (c) Cost aggregation along y-axix.

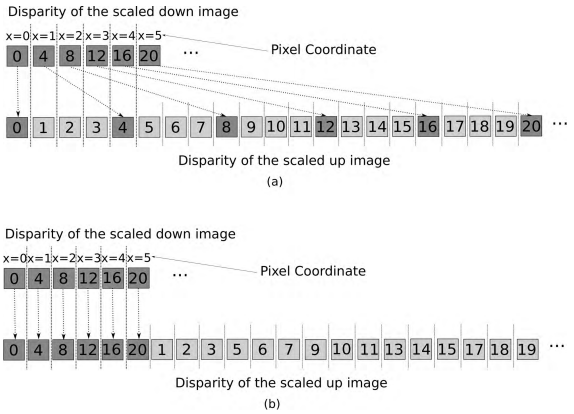

FIGURE 11. Effective scaling-up (K = 4).

since the image data are also used during the Scaling-up step, the number of lines must be K at least.

3) The remaining steps are executed. During the Scaling up step, instead of storing the scaled up disparity map data successively (Fig.11(a)), the interpolated data are stored separately as shown in Fig.11(b). These data are reverted to the original order before the Cross-Check step. This method is used to avoid shifting the disparity of the scaled down images when they are fetched from the global memory.

V. EXPERIMENTAL RESULTS

We have implemented the algorithm on a middle-end GPU NVIDIA GTX780 Ti and a high-end GPU NVIDIA

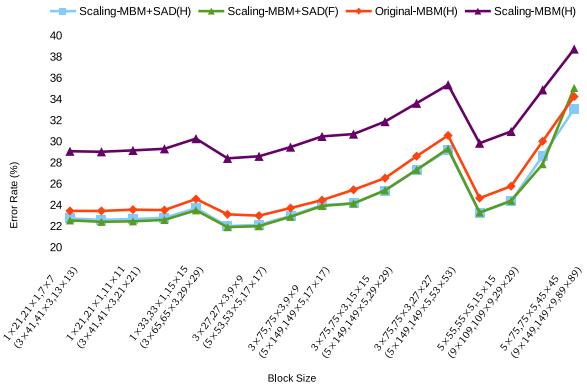

FIGURE 12. Accuracy comparison. ***Scaling-MBM+SAD*: Result of MBM on scaled down images with a secondary SAD matching. *Original-MBM*: Result of MBM on original images. *Scaling-MBM*: Result of MBM on scaled down images without secondary matching. *(H)*: H-size dataset. *(F)*: F-Size dataset.**

GTX1080 Ti respectively, and evaluated the processing speed and the error rate using the Middlebury V3 [16], KITTI2012 [26] and KITTI2015 [27] benchmarks. In this section, we first evaluate the accuracy and processing speed of our system using each benchmark, and then make a comprehensive comparison with other systems.

A. MIDDLEBURY BENCHMARK

Fig.12 shows an accuracy comparison of our proposed method with the *Original-MBM* (MBM on the original image

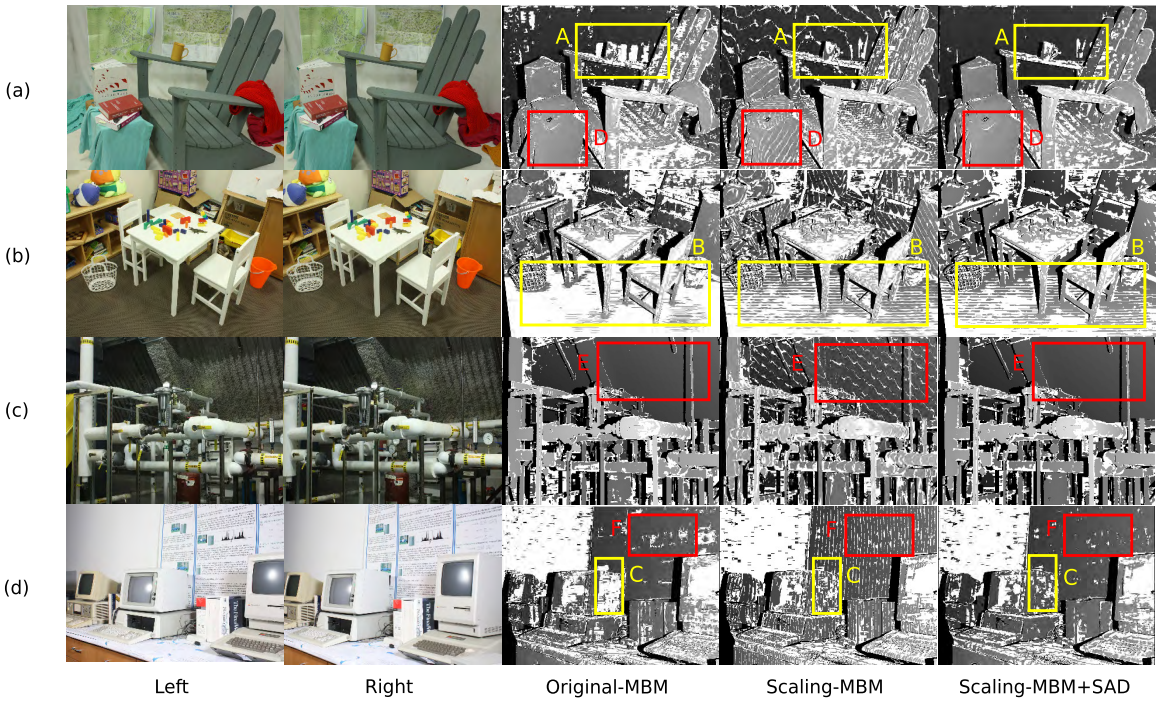

FIGURE 13. Matching Result. (a) Adirondack (b) Playtable (c) Pipes (d) Vintage. *A*: Repetitive patterns. *B*: Perspective distortions & Uniform regions. *C*: Uniform regions. *D*,*E*,*F*: Gradient regions.

set) and the *Scaling_MBM* (MBM on the scaled down image set without secondary matching). In Fig.12, H and F are the image sizes. Images in F are larger than those in H. Their sizes are shown in Table.3. In this evaluation, $K = 2$ for the H-size images, and $K = 4$ for F-size images. For achieving higher processing speed, the block size of SAD is fixed to 3×3. T (a threshold introduced in Section III-D.2) is set to K. In Fig.12, the x-axis shows several combinations of block sizes that are used in MBM. The sizes of all blocks are chosen so that the intermediate results of smaller blocks can be reused for larger blocks. The block sets shown in parentheses are used for the original size images and their sizes are two times those for the scaled down images. The y-axis shows the bad 2.0 error rate (percentage of "bad" pixels whose disparity are different more than 2.0) of the above three algorithms for the training image set. As shown in this graph, when the blocks are too large, their error rates become worse. For the *Scaling-MBM* and *Scaling-MBM+SAD*, the $3 \times 21, 21 \times 3, 9 \times 9$ block set shows the lowest error rate, and for the *Original-MBM*, the $5 \times 53, 53 \times 5, 17 \times 17$ shows the lowest error rate. *Scaling-MBM* shows the worst error rate (roughly 6% higher than other methods), and our method *Scaling-MBM+SAD* shows the lowest for both of H-size and F-size data sets. Furthermore, the error rates for both size images are almost the same, which shows that our method is very robust. Here, it can be noted that the accuracy of our methods are always better than the *Original-MBM*, even though the information of original images is lost by down-scaling.

Fig.13 shows the results of four H-size images in the Middlebury Benchmark [16]. The block size used in *Scaling-MBM* and *Scaling-MBM+SAD* is $3 \times 21, 21 \times 3, 9 \times 9$, and the block size of SAD is 3×3, while the block size used in *Original-MBM* is $5 \times 41, 41 \times 5, 17 \times 17$. Three areas, A, B and C, show the repetitive patterns, perspective distortions and uniform regions respectively, which represent the three kinds of difficult problems for the block matching method. D, E and F show the gradient regions for which it is required to decide their disparities considering their continuity. The white pixels represent the matching errors. As shown in this figure, our approach shows better results than other approaches specially in those marked areas. This means that in our approach, GCPs are correct as well as the *original-MBM*, and the disparities of non-GCPs are improved better. For A,B and C regions, because of the scaling down, some information that makes the matching difficult in the original size images, such as repetition of patterns and a serious of similar pixels, are discarded, and better matching becomes possible. Furthermore, because of the secondary matching, our method, Scaling-MBM+SAD, shows a better result than the Scaling-MBM in these regions. For D, E and F, a secondary matching generates the continuous disparities, and the disparities are improved to the same level as matching by *Original-MBM*.

Table.2 shows the accuracy comparison among several stereo vision systems. In this table, systems that were evaluated using Middlebury and KITTI benchmarks are listed by their average error rates on Middlebury benchmark. Our error

TABLE 2. Accuracy Comparison on Middlebury Benchmark.

Date	avgerr (pixels) Name	Res	Weight Avg	Adiron MP: 5.7 nd: 290 im0 im1 GT nonocc	ArtL MP: 1.5 nd: 256 im0 im1 GT nonocc	Jadepl MP: 5.2 nd: 640 im0 im1 GT nonocc	Motor MP: 5.9 nd: 280 im0 im1 GT nonocc	MotorE MP: 5.9 nd: 280 im0 im1 GT nonocc	Piano MP: 5.4 nd: 260 im0 im1 GT nonocc	PianoL MP: 5.4 nd: 260 im0 im1 GT nonocc	Pipes MP: 5.7 nd: 300 im0 im1 GT nonocc	Playrm MP: 5.3 nd: 330 im0 im1 GT nonocc	Playt MP: 5 nd: 290 im0 im1 GT nonocc	PlaytP MP: 5 nd: 290 im0 im1 GT nonocc	Recyc MP: 5.6 nd: 260 im0 im1 GT nonocc	Shelvs MP: 5.9 nd: 240 im0 im1 GT nonocc	Teddy MP: 2.7 nd: 256 im0 im1 GT nonocc	Vintge MP: 5.5 nd: 760 im0 im1 GT nonocc
05/31/18	IResNet_ROB	H	**2.50** 1	1.17 12	2.80 5	8.93 5	2.16 9	2.16 9	1.59 3	2.12 5	4.05 6	2.63 4	1.44 2	1.17 2	1.04 4	1.81 5	1.22 5	1.97 6
01/24/17	3DMST	H	4.59 2	1.16 10	5.25 25	23.9 19	2.99 16	2.94 13	2.01 13	3.68 17	6.17 17	3.53 9	1.68 6	1.62 8	1.30 8	4.77 18	1.83 21	2.82 13
04/19/15	MeshStereo	H	7.58 3	2.39 32	6.41 35	36.4 60	5.40 50	5.71 53	3.25 29	5.45 28	11.6 53	6.34 32	4.92 29	2.73 25	2.25 27	11.1 72	1.85 23	5.62 21
04/08/15	REAF	H	8.49 4	4.07 56	7.29 43	32.4 46	4.76 38	4.70 38	5.00 54	11.7 65	10.9 42	8.61 55	15.1 62	4.04 42	2.54 33	9.72 62	3.30 50	9.29 42
07/28/14	SGM	F	8.53 5	4.90 67	4.65 17	30.3 39	4.70 35	4.32 32	3.77 39	5.76 29	10.4 36	7.04 39	25.4 82	4.27 49	4.36 68	9.34 60	2.33 30	17.7 77
05/10/18	MSMD_ROB	F	9.22 6	2.85 40	8.58 57	45.1 77	5.12 43	4.99 42	3.75 38	7.65 42	11.0 44	6.86 37	9.76 48	9.32 77	2.74 37	3.58 14	3.02 47	9.59 44
01/15/17	IGF	Q	9.49 7	4.56 61	7.33 44	28.8 36	5.87 57	5.91 54	6.36 74	12.1 67	11.5 52	7.16 41	27.3 87	8.64 73	3.85 62	9.19 55	3.30 50	9.12 38
04/24/16	HLSC_cor	H	9.61 8	3.35 50	9.70 63	35.0 56	6.85 70	6.87 65	3.92 41	7.30 40	13.8 73	10.1 65	16.6 67	3.90 40	3.55 56	11.7 77	2.99 45	14.6 61
09/18/14	SNCC	H	10.4 9	3.63 54	6.74 37	39.8 66	5.12 43	5.11 44	4.65 49	8.23 46	11.8 56	8.05 50	45.6 96	4.36 54	3.29 51	8.10 40	2.52 32	14.8 62
03/26/18	ELAS_ROB	H	10.5 10	4.08 57	7.18 42	52.8 81	5.40 50	5.47 47	5.00 54	9.14 52	10.7 38	8.03 49	23.3 79	3.83 38	3.79 60	9.40 61	3.27 49	10.5 48
06/08/18	MBM	H	10.7 11	4.31 60	9.25 60	37.0 61	5.84 56	5.65 51	7.10 76	13.9 71	11.5 51	12.6 70	24.4 81	7.68 69	3.49 54	12.5 81	3.90 60	13.3 57
10/13/15	MDP	H	10.8 12	1.56 16	7.37 45	53.8 82	5.89 58	6.18 61	4.04 43	8.81 50	14.2 75	11.0 67	15.8 66	4.19 45	4.00 65	9.24 56	3.95 62	15.2 64
11/13/17	CBMV	H	11.5 13	5.98 76	11.0 67	41.2 70	5.26 47	5.51 48	4.03 42	14.1 72	11.2 46	11.5 68	8.45 39	6.89 66	3.84 61	8.45 44	7.11 73	40.2 90
08/28/15	MC-CNN-acrt	H	11.8 14	4.24 59	18.7 86	34.1 53	7.21 72	7.22 68	6.00 68	9.35 53	13.5 68	18.3 76	9.71 47	9.37 78	4.64 70	6.62 29	9.31 80	21.6 84
11/24/16	ADSM	Q	12.3 15	14.3 90	10.6 66	34.1 52	6.00 62	8.00 72	7.37 77	20.4 82	12.1 60	16.9 72	25.5 83	5.84 61	5.83 81	17.2 88	4.11 63	11.1 53
05/01/18	PSMNet_ROB	Q	13.3 16	8.83 84	13.9 70	68.4 91	8.26 76	9.16 79	5.89 67	10.5 59	14.4 76	9.38 59	5.54 32	5.52 60	4.98 72	11.6 76	3.87 59	9.66 45
11/15/16	MC-CNN-WS	H	13.7 17	5.73 73	20.5 89	36.3 59	9.39 82	9.37 80	8.13 80	16.1 77	16.7 85	18.7 78	11.5 51	10.1 84	5.05 75	9.83 63	11.0 88	20.8 83
09/10/14	LAMC_DSM	H	14.6 18	7.65 81	21.8 91	37.9 63	11.3 87	11.1 82	8.81 82	11.7 66	17.4 87	22.7 84	15.4 63	10.6 85	5.91 82	13.4 83	10.2 82	15.5 65
08/31/14	BSM	Q	23.5 19	12.7 87	28.7 95	58.7 87	14.8 93	14.7 86	16.0 93	35.8 94	24.5 91	29.4 91	31.0 90	20.2 95	12.1 91	19.2 93	14.3 95	39.3 89
06/09/18	MBM	F	10.8 11	4.40 60	9.22 60	36.6 61	5.76 53	5.58 50	7.10 76	14.4 72	11.5 51	13.3 70	25.4 81	7.52 69	3.58 58	13.1 81	3.87 59	13.8 58

Size(MBM): $3 \times 21, 21 \times 3, 9 \times 9$. Size(SAD): 3×3. iResNet_ROB [29], 3DMST [41], MeshStereo [37], REAF [39], SGM [19], MSMD_ROB [36], IGF [40], HLSC_cor [31], SNCC [20], ELAS_ROB [21], MDP [35], CBMV [34], MC-CNN-arct [5], ADSM [32], PSMNet_ROB [38], MC-CNN-WS [18], LAMC_DSM [30], BSM [33]

TABLE 3. execution time with the Middlebury benchmark set (ms).

Image	Size	Dmax	Gray(CPU)	HtoD	SS	NCC&AggH	AggV	WTA&SeM	CC	DtoH	Overall(GPU)
Adirondack(H)	1436×992	145	5.31	0.226	0.37	1.135	2.68	0.684	0.274	0.432	5.801
MotorcycleE(H)	1482×994	128	4.98	0.233	0.039	1.073	2.652	0.678	0.254	0.447	5.366
Teddy(H)	900×750	128	1.9	0.11	0.029	0.51	0.875	0.309	0.205	0.205	2.243
Vintage(H)	1444×960	380	4.81	0.219	0.038	3.096	7.6	1.779	0.545	0.422	13.699
Adirondack(F)	2872×1984	290	23.88	0.921	0.052	1.191	2.693	1.029	1.334	1.764	8.984
MotorcycleE(F)	2964×1988	256	23.02	0.931	0.054	1.118	2.63	1.007	1.162	1.788	8.69
Teddy(F)	1800×1500	256	10.29	0.428	0.037	0.537	1.271	0.539	0.82	0.82	4.452
Vintage(F)	2888×1920	760	21.43	0.889	0.053	3.248	8.189	2.288	2.322	1.699	18.688

Size(MBM): 3×21, 21×3, 9×9 ***Size(SAD)***: 3×3 ***Dmax***: *Maximum Disparity* ***Gray***: *Graying.* ***HtoD***: *Data transmission time from CPU to GPU.* ***SS***: *Smoothing & Scaling-Down.* ***NCC&AggH***: *NCC cost calculation & Aggregation along the x axis.* ***AggV***: *Aggregation along the y axis.* ***WTA&SeM***: *WTA & Secondary Matching.* ***CC***: *Cross_Check.* ***DtoH***: *Data transmission time from GPU to CPU.* **Overall**: The overall time taken on GPU.

rate for F-size is listed at the bottom, because Table.2 is a hard copy of the benchmark evaluation site [16], and in this site, it is not allowed to upload more than one result at a time. Our error rates for H-size and F-size image sets are not the top, but they are not bad compared with other systems.

Table.3 shows the details of the processing speed of our system on GTX1080 Ti. We select 8 representative image sets from the Middlebury Benchmark [16]. 4 sets of them are H-size, and 4 sets are F-size. In the evaluated images, for example, Adirondack and MotorcycleE are large images with small disparity, Vintage is a large image with large disparity, and Teddy is a small image with small disparity. For each image set, the execution time of each processing step is shown. The graying step (*Gray*) which takes the large portion of the execution time, is executed on CPU. *HtoD* and *DtoH* steps show the data transmission time between the CPU and GPU. *DtoH*, the transmission time from GPU to CPU, is roughly twice as *HtoD*, CPU to GPU, because *HtoD* is for

TABLE 4. Processing speed comparison (sec).

System	Adirondack	ArtL	Jadeplant	Motorcycle	MotorcycleE	Piano	PianoL	Pipes
IresNet [29]	0.35 (23.33)	0.28 (70)	0.35 (12.96)	0.35 (23.33)	0.35 (23.33)	0.34 (24.29)	0.34 (24.29)	0.34 (21.25)
ELAS [21]	0.52 (34.67)	0.13 (32.5)	0.49 (18.15)	0.54 (36)	0.55 (36.67)	0.5 (35.71)	0.49 (35)	0.49 (30.63)
MSMD [36]	0.9 (60)	0.2 (50)	0.8 (29.63)	0.9 (60)	0.9 (60)	0.8 (57.14)	0.8 (57.14)	0.9 (56.25)
SNCC [20]	1.03 (68.67)	0.24 (60)	1.67 (61.85)	1.05 (70)	1.09 (72.67)	0.82 (58.57)	0.82 (58.57)	1.07 (66.88)
MC-CNN-WS [18]	2.65 (176.67)	0.63 (157.5)	4.14 (153.33)	2.58 (172)	2.58 (172)	2.25 (160.71)	2.25 (160.71)	2.59 (161.88)
REAF [39]	4.37 (291.33)	0.78 (195)	9.73 (360.37)	4.43 (295.33)	4.43 (295.33)	3.32 (237.14)	3.31 (236.43)	4.7 (293.75)
MDP [35]	55.5 (3700)	12.8 (3200)	82.2 (3044)	71.8 (4787)	71.6 (4773)	60 (4286)	64.1 (4578)	78.3 (4894)
Meshstereo [37]	59.1 (3940)	49.8 (12450)	87 (3222)	63 (4200)	63 (4200)	62.2 (4443)	62.5 (4464)	66.6 (4163)
MC-CNN-acrt [5]	100 (6666.67)	23.9 (5975)	206 (7630)	109 (7267)	108 (7200)	99 (7071)	96.5 (6893)	108 (6750)
3DMST [41]	178 (11867)	39 (9750)	203 (7519)	203 (13533)	199 (13267)	187 (13357)	184 (13143)	171 (10688)
LAMC-DSM [30]	519 (34600)	158 (39500)	1034 (38296)	493 (32867)	495 (33000)	477 (34071)	542 (38714)	573 (35813)
HLSC [31]	1285 (85667)	147 (36750)	2281 (84481)	1933 (128867)	1899 (126600)	1340 (95714)	1279 (91357)	1689 (105563)
CBMV [34]	424 (28266.67)	245 (61250)	1207 (44703)	605 (40333)	618 (41200)	520 (37143)	526 (37571)	609 (38063)
our system(1080Ti)	0.005 (0.33)	0.001 (0.25)	0.01 (0.37)	0.005 (0.33)	0.005 (0.33)	0.004 (0.29)	0.005 (0.36)	0.005 (0.31)
our system(780Ti)	0.015 (1)	0.004 (1)	0.027 (1)	0.015 (1)	0.015 (1)	0.014 (1)	0.014 (1)	0.016 (1)

System	Playroom	Playtable	PlaytableP	Recycle	Shelves	Teddy	Vintage	Geometric Mean
IresNet [29]	0.35 (21.88)	0.34 (24.29)	0.35 (25)	0.35 (26.92)	0.35 (25)	0.32 (45.71)	0.35 (10.29)	**20.8**
ELAS [21]	0.5 (31.25)	0.49 (35)	0.46 (32.86)	0.5 (38.46)	0.54 (38.57)	0.23 (32.86)	0.51 (15)	**26.13**
MSMD [36]	0.8 (50)	0.7 (50)	0.7 (50)	0.8 (61.54)	0.9 (64.29)	0.4 (57.14)	0.8 (23.53)	**41.18**
SNCC [20]	1.08 (67.5)	0.9 (64.29)	0.91 (65)	0.95 (73.08)	0.94 (67.14)	0.42 (60)	2.06 (60.59)	**49.3**
MC-CNN-WS [18]	2.67 (166.88)	2.21 (157.86)	2.24 (160)	2.22 (170.77)	2.31 (165)	1.06 (151.43)	4.84 (142.35)	**116.19**
REAF [39]	4.75 (296.88)	4.07 (290.71)	4.17 (297.86)	3.69 (283.85)	3.78 (270)	1.32 (188.57)	12.4 (364.71)	**185.88**
MDP [35]	62.7 (3919)	48.8 (3486)	55.5 (3964)	55.4 (4262)	79.8 (5700)	26.1 (3729)	28.6 (841)	**2358**
Meshstereo [37]	66.2 (4138)	58.4 (4171)	57.9 (4136)	56.9 (4377)	59.5 (4250)	50.4 (7200)	71.5 (2103)	**2648**
MC-CNN-acrt [5]	122 (7625)	104 (7429)	101 (7214)	92.6 (7123)	94.5 (6750)	40.5 (5786)	264 (7765)	**3846**
3DMST [41]	171 (10688)	171 (12214)	189 (13500)	177 (13615)	178 (12714)	87 (12429)	208 (6118)	**6354**
LAMC-DSM [30]	567 (35438)	486 (34714)	482 (34429)	481 (37000)	578 (41286)	217 (31000)	961 (28265)	**17730**
HLSC [31]	1514 (94625)	1146 (81857)	1177 (84071)	1112 (85538)	1470 (105000)	378 (54000)	1905 (56029)	**40440**
CBMV [34]	602 (37625)	559 (39929)	565 (40357)	9999 (769154)	493 (35214)	382 (54571)	1464 (43059)	**24310**
our system(1080Ti)	0.005 (0.31)	0.004 (0.29)	0.005 (0.36)	0.004 (0.31)	0.004 (0.29)	0.002 (0.29)	0.013 (0.38)	**0.34**
our system(780Ti)	0.016 (1)	0.014 (1)	0.014 (1)	0.013 (1)	0.014 (1)	0.007 (1)	0.034 (1)	**1**

Geometric Mean: The Geometric Mean is calculated from "Adirondack" to "Vintage" image sets.

two unsigned char images and *DtoH* is for one float image. The transmission time of F-size data set is roughly 4 times that of H-size, because F-size images are 4 times larger than H-size images. Among the steps executed on GPU, the cost calculation and its aggregation (*NCC&AggH* and *AggV*) take most of the computation time, because frequent data transfer (*D* times loops) between the on-chip memory and off-chip memory is required. The execution time of the secondary matching step (*WTA&SeM*) is relatively small, although several loops of matching are still needed. This is because SAD does not require the global memory access to save/read the intermediate results owing to the small amount data required in this step. The overall time required on GPU is proportional to the image size and the maximum disparity. As shown in *overall* field, even for the largest image set "Vintage", its processing time is only 18.688ms, which makes our real-time processing possible. However, in the processing for F-size data set, it can be noted that the time required by CPU is longer than GPU, and the processing speed depends on the graying step on CPU.

Fig.14 compares the processing speed when the block sizes are changed in our system. As shown in this figure, in all cases, more computation time is required for larger blocks. Considering the error rate shown in Fig.12, the block sizes used in our implementation, $3 \times 21, 21 \times 3, 9 \times 9$, gives the good balance of the processing speed and the matching accuracy. Table.4 shows the comparison of our processing speed with other systems listed in Table.2. Here, because the image sizes and the disparity ranges are different, in order to make a clear comparison, we chose the systems that can process the H-size image sets, and used the *Geometric Mean* method [28] to compare them. Due to the space limitation,

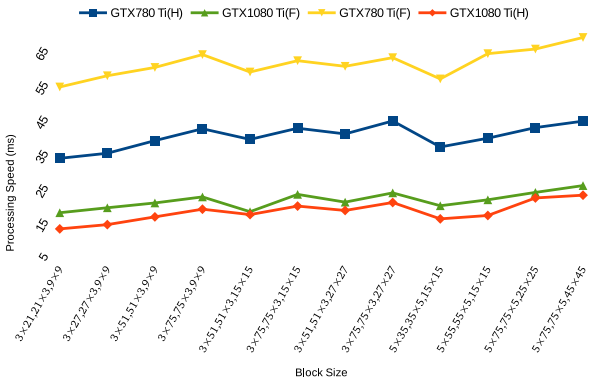

FIGURE 14. Processing speed comparison.

Table.4 is folded into two (each system on the leftmost column is evaluated using 15 images (8 in the upper part, and 7 in the lower part)). In this table, for each system, two values are shown for each corresponding image. The first one is its runtime, and the second one in the parentheses is the normalized value by our system (GTX 780Ti). From these values, one *Geometric Mean* value can be calculated for each system. Based on this value, the difference of the processing speed among these systems can be shown obviously. As shown in the column *Geometric Mean* of Table.4, the processing speed of our system is much faster than other system. The processing speed by GTX 1080Ti is three times faster than GTX 780Ti.

B. KITTI BENCHMARK

We also evaluated our system using KITTI2012 [26] and KITTI2015 [27] benchmarks, separately. In these

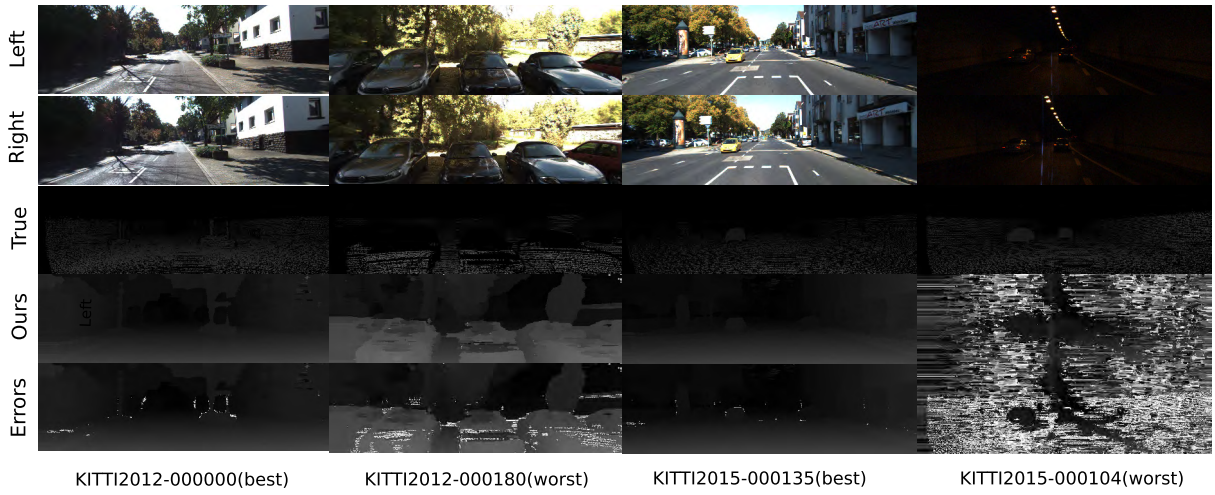

FIGURE 15. Evaluation results using the KITTI benchmarks.

TABLE 5. KITTI2012.

Error	Out-Noc	Out-All	Avg-Noc	Avg-All
2 pixels	7.86 %	9.53 %	1.2 px	1.4 px
3 pixels	5.45 %	6.88 %	1.2 px	1.4 px
4 pixels	4.38 %	5.56 %	1.2 px	1.4 px
5 pixels	3.70 %	4.67 %	1.2 px	1.4 px

TABLE 6. KITTI2015.

Error	D1-bg	D1-fg	D1-all
All / All	6.06 %	14.13 %	7.40 %
All / Est	6.06 %	14.13 %	7.40 %
Noc / All	5.41 %	12.72 %	6.62 %
Noc / Est	5.41 %	12.72 %	6.62 %

benchmarks, hundreds of images are divided into two groups. For the first group (“training dataset”), their true disparity maps are given, and this group is used to tune the parameters of the stereo vision systems. The users are required to upload their disparity maps for the second group (“testing dataset”) to the website, and their matching accuracy are evaluated on the website. The image sizes are close to 1250 × 375 and the ranges of disparities are always 256. Here, since the numbers of lines in KITTI are small, we only scaled down the images along the x-axis (K = 2). According to our evaluation, the runtime of our system on GTX 780Ti is roughly 0.015s (66.7 fps), and it is 0.005s (200 fps) on GTX 1080Ti. For the “testing dataset” (the second group), our accuracy is shown in Table.5 and Table.6, which was ranked 73rd out of 108 systems in KITTI2012 and ranked 94th out of 119 systems in KITTI2015. This accuracy is not good, but for this evaluation, the same parameters as *Middlebury* are used, and they are not tuned for KITTI benchmarks. As for the accuracy of the “training dataset” (for this first group, we can know the error rate of each image), we achieved the lowest error rates 0.084% for “KITTI2012-000000” and 0.058% for “KITTI2015-000135”, while the worst error rate 20.6% for “KITTI2012-000180” and 67.8% for “KITTI2015-000104” as shown in Fig.15. According to our results, except the “KITTI2015-000104” that was taken in a tunnel, most of the error rates are kept between 1% and 5% on both benchmarks. We think that this error rate is enough for most practical use.

C. ACCURACY COMPARISON BETWEEN DIFFERENT SYSTEMS

In this subsection, we compare the error rates of all systems that are evaluated not only by using Middlebury, but also by using at least one of KITTI benchmarks. For this comparison, the *Geometric Mean* is used. As shown in Table.7, the 2nd, 3rd and 4th columns show the error rates (and the normalized one by our system) for Middlebury, KITTI 2012 and KITTI 2015 benchmark sets, and 5th, 6th and 7th columns show the Geometric Mean for Middlebury with KITTI 2012, KITTI 2015, and both. According to these results, it can be noted that the accuracy of our system in KITTI2012 and Middlebury is close to the medium level, and as described above, the accuracy of KITTI2015 is lower, which leads to a decrease in overall performance.

D. SPEED COMPARISON BETWEEN DIFFERENT SYSTEMS

Table.8 compares the processing speed of stereo vision systems on different architectures. All the systems achieved a real-time processing as shown in *FPS* field, but their target image size (*Size*) and disparity range (*Dmax*) are different. *MDE/s* means mega disparity evaluation per second, and shows the true processing speed of each system. As shown in this table, *MDE/s* of our system is much higher than other systems. For calculating a disparity map of large size image such as 2888 × 1920, larger *Dmax* (760) is required, and with GTX 780Ti, its real-time processing cannot be achieved. However, by using faster GPU, GTX 1080 Ti, it becomes possible by our approach, and its *MDE/s* is 12x to 1060x faster than other systems. To compare the performance on different architectures, another criterion, *disparities/cycle*, is calculated by using equation (18), which means the number of disparities that can be processed per clock cycle on each

TABLE 7. Accuracy comparison.

System	KITTI 2012 (%)	KITTI 2015(%)	Middlebury (%)	GM2012	GM 2015	GM Overall
Our system(H)	6.88 (1)	7.4 (1)	10.7 (1)	1	1	1
Our system(F)	6.88 (1)	7.4 (1)	10.8 (1.01)	1	1	1
MC-CNN-Acrt [5]	3.63 (0.53)	3.89 (0.53)	11.8 (1.1)	0.76	0.76	0.68
CBMV [34]	4.73 (0.69)	5.06 (0.68)	11.5 (1.07)	0.86	0.85	0.79
SGM [19]	7 (1.02)	6.38 (0.86)	8.51 (0.8)	0.9	0.83	0.89
PSMNET [38]	1.89 (0.27)	2.31 (0.31)	13.3 (1.24)	0.58	0.62	0.47
SNCC [20]	6.44 (0.94)	7.14 (0.96)	10.4 (0.97)	0.95	0.96	0.96
ELAS [21]	9.66 (1.4)	9.72 (1.31)	10.5 (0.98)	1.17	1.13	1.22
3DMST [41]	-	4.97 (0.67)	4.59 (0.43)	-	0.54	-
MDP [35]	-	5.36 (0.72)	10.8 (1.01)	-	0.85	-
MSMD [36]	-	4.16 (0.56)	9.22 (0.86)	-	0.69	-
Meshstereo [37]	-	8.38 (1.13)	7.58 (0.71)	-	0.9	-
REAF [39]	-	10.11 (1.37)	8.49 (0.79)	-	1.04	-
IGF [40]	-	10.84 (1.46)	9.49 (0.89)	-	1.14	-
IresNet [29]	2.16 (0.31)	-	2.5 (0.23)	0.27	-	-
MC-CNN-WS [18]	4.45 (0.65)	-	13.7 (1.28)	0.91	-	-
LAMC-DSM [30]	11.49 (1.67)	-	14.6 (1.36)	1.51	-	-
HLSC [31]	12.82 (1.86)	-	9.61 (0.9)	1.29	-	-
ADSM [32]	10.05 (1.46)	-	12.3 (1.15)	1.3	-	-
BSM [33]	13.44 (1.95)	-	23.5 (2.2)	2.07	-	-

Our system(H): The sizes of Middlebury image sets are H-size. **Our system(F)**: The sizes of Middlebury image sets are F-size. **GM 2012**: The Geometric Mean of the KITTI 2012 and Middlebury benchmarks. **GM 2015**: The Geometric Mean of the KITTI 2015 and Middlebury benchmarks. **GM Overall**: The Geometric Mean of the KITTI 2012, KITTI 2015 and Middlebury benchmarks.

TABLE 8. Comparison with high-speed stereo vision systems.

System	Size	Dmax	FPS	MDE/s	Hardware	Frequency	Disparities/Cycle	#Cores	Disparities/Cycle/Core
RT-FPGA [1]	1920 × 1680	60	30	5806	Kintex 7	160Mhz	36.288	-	-
FUZZY [2]	1280 × 1024	15	76	1494	Cyclone II	100Mhz	14.942	-	-
FPGA-Road [7]	1920 × 1080	128	50	13271	Virtex-6	110Mhz	120.645	-	-
Low-Power [8]	1024 × 768	64	30	1510	Virtex-7	50Mhz	30.198	-	-
FlexibleSV [42]	2048 × 1024	255	22	11765	Zynq 7045	200Mhz	58.82	-	-
FPGART [43]	1600 × 1220	128	42	10321	Stratix-IV	180Mhz	57.344	-	-
FPGA-SOC [44]	1280 × 720	256	60	14156	Zynq XC7Z030	200Mhz	70.77	-	-
RT-MULTI [45]	640 × 480	64	325	6390	Virtex-6	100Mhz	63.89	-	-
H-GF [48]	1280 × 720	60	60	3539	Kintex-7	145Mhz	24.40	-	-
ScalSGM [52]	1280 × 720	128	30	3438	VC709	130Mhz	27.20	-	-
LowEng [53]	1024 × 768	128	127	12784	Cyclone-4	100Mhz	127.84	-	-
RT-GF [54]	1920 × 1080	48	80	7962	Cyclone-4	108.5Mhz	73.38	-	-
BP-Stereo [49]	450 × 375	60	33	334	I5-4570	3200Mhz	0.104	-	-
Blind-RT [50]	640 × 360	99	30	684	Zynq ZC702	150Mhz	4.56	-	-
ADAS [51]	1280 × 720	256	30	7078	ADAS+FPGA	250Mhz	28.31	-	-
RGB-SV [47]	450 × 375	60	23	233	Quad-Core	2660Mhz	0.0875	4	0.021886
Comp-GPU [55]	640 × 480	128	55	2163	GTX 280	604Mhz	3.581	240	0.014919
IDR [13]	450 × 375	42	23	163	GTX TITAN BlACK	889Mhz	0.18	2880	0.000064
BF-DP [46]	340 × 240	16	192	251	GTX 580	772Mhz	0.324	512	0.000634
ETE [3]	1242 × 375	256	29	3458	GTX TITAN X	1000Mhz	3.45	3072	0.001126
EmbeddedRT [11]	640 × 480	128	81	1652	Tegra X1	1000Mhz	1.651	256	0.006415
Our System 1	1444 × 960	380	30	15803	GTX 780 Ti	876Mhz	18.04	2880	0.006264
Our System 2	1444 × 960	380	72	37927	GTX 1080 Ti	1480Mhz	25.626	3854	0.006649
Our System 3	2888 × 1920	760	18	75855	GTX 780 Ti	876Mhz	86.592	2880	0.030067
Our System 4	2888 × 1920	760	41	172781	GTX 1080 Ti	1480Mhz	116.743	3854	0.030292

architecture.

$$Disparities/cycle = \frac{ImageSize \times D_{max} \times FPS}{Frequency} \quad (18)$$

For GPU systems, the *frequency* refers to the frequency of each core, and for other systems, it means the frequency of the overall system. By comparing *disparities/cycle*, we can understand the performance gain by the algorithms implemented on each device. As shown in Table.8, for some systems, although their *MDE/s* are similar, there exists a gap in *disparities/cycle* such as "FPGA-Road [7]" and "FPGA-SOC [44]", because their frequency is different. Our rate 116.743 (for F-size images on GTX 1080Ti) is much faster than other GPU systems (even 18.04 for H-size images on GTX-780 Ti is faster). This means that our algorithm works very well on GPUs. However, FPGAs shows higher rate than our system. This comes from the fact that higher parallelism is possible on FPGAs than GPUs because the data width required in the stereo vision systems is less than 2B in many cases. To compare the performance of the systems on CPU and GPU, *disparities/cycle/core* is also shown in Table.8. This comparison shows that in our system, each core works more efficiently than other GPU systems, and even than CPU system for H-size images.

VI. CONCLUSION

In this paper, we have proposed a real-time stereo vision system on GPU. Its processing speed is much faster than previous ones, and it is the first GPU system that enables the real-time processing of high resolution images. The acceleration method was originally designed for GTX 780 Ti, but it could be easily ported on GTX 1080 Ti, and showed good performance on the two different GPU architectures. The performance gain by new architecture is 3X, and this is that we can expect from the difference of their maximum throughput. This means that it can be expected that our method can also achieve higher performance on upcoming new GPUs.

In our current implementation, the accuracy and the processing speed are still limited by the size of the shared memory. The improvement of the memory usage by further tuning the algorithms used in our implementation is one of our main future work.

REFERENCES

[1] D. Zha, X. Jin, and T. Xiang, "A real-time global stereo-matching on FPGA," *Microprocessors Microsyst.*, vol. 47, pp. 419–428, Nov. 2016.

[2] M. P. Patricio, A. Aguilar-González, M. Arias-Estrada, H. R. Hernández-De León, J. L. Camas-Anzueto, and J. A. de Jesús Osuna-Coutiño, "An FPGA stereo matching unit based on fuzzy logic," *Microprocessors Microsyst.*, vol. 42, pp. 87–99, May 2016.

[3] A. Kendall *et al.* (2017). "End-to-end learning of geometry and context for deep stereo regression." [Online]. Available: https://arxiv.org/abs/1703.04309

[4] W. Qiao and J. C. Créput, "Stereo matching by using self-distributed segmentation and massively parallel GPU computing," in *Proc. Int. Conf. Artif. Intell. Soft Comput.*, 2016, pp. 723–733.

[5] J. Žbontar and Y. LeCun, "Stereo matching by training a convolutional neural network to compare image patches," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 2287–2318, 2016.

[6] X. Ye, J. Li, H. Wang, H. Huang, and X. Zhang, "Efficient stereo matching leveraging deep local and context information," *IEEE Access*, vol. 5, pp. 18745–18755, 2017.

[7] M. Dehnavi and M. Eshghi, "FPGA based real-time on-road stereo vision system," *J. Syst. Archit.*, vol. 81, pp. 32–43, Nov. 2017.

[8] L. Puglia, M. Vigliar, and G. Raiconi, "Real-time low-power FPGA architecture for stereo vision," *IEEE Trans. Circuits Syst., II, Exp. Briefs*, vol. 64, no. 11, pp. 1307–1311, Nov. 2017.

[9] N. Einecke and J. Eggert, "A multi-block-matching approach for stereo," in *Proc. Intell. Vehicles Symp.*, Jun./Jul. 2015, pp. 585–592.

[10] A. Kuzmin, D. Mikushin, and V. Lempitsky. (2016). "End-to-end learning of cost-volume aggregation for real-time dense stereo." [Online]. Available: https://arxiv.org/abs/1611.05689

[11] D. Hernandez-Juarez, A. Chacón, A. Espinosa, D. Vázquez, J. C. Moure, and A. M. López, "Embedded real-time stereo estimation via semi-global matching on the GPU," in *Proc. Int. Conf. Comput. Sci.*, 2016, pp. 143–153.

[12] M. Jin and T. Maruyama, "Fast and accurate stereo vision system on FPGA," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 7, no. 1, 2014, Art. no. 3.

[13] J. Kowalczuk, E. Psota, and L. C. Perez, "Real-time stereo matching on CUDA using an iterative refinement method for adaptive support-weight correspondences," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 1, pp. 94–104, Jan. 2013.

[14] Q. Yang, L. Wang, R. Yang, H. Stewénius, and D. Nistér, "Stereo matching with color-weighted correlation, hierarchical belief propagation, and occlusion handling," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 3, pp. 492–504, Mar. 2009.

[15] H. Hirschmüller and D. Scharstein, "Evaluation of cost functions for stereo matching," in *Proc. CVPR*, Jun. 2007, pp. 1–8.

[16] D. Scharstein *et al.*, "High-resolution stereo datasets with subpixel-accurate ground truth," in *Proc. German Conf. Pattern Recognit. (GCPR)*, Münster, Germany, Sep. 2014.

[17] P. Knöbelreiter, C. Reinbacher, A. Shekhovtsov, and T. Pock, "End-to-end training of hybrid CNN-CRF models for stereo," in *Proc. CVPR*, Jun. 2017, pp. 2339–2348.

[18] S. Tulyakov, A. Ivanov, and F. Fleuret, "Weakly supervised learning of deep metrics for stereo reconstruction," in *Proc. ICCV*, Oct. 2017, pp. 1348–1357.

[19] H. Hirschmüller, "Accurate and efficient stereo processing by semi-global matching and mutual information," in *Proc. CVPR*, vol. 2, Jun. 2005, pp. 807–814.

[20] N. Einecke and J. Eggert, "A two-stage correlation method for stereoscopic depth estimation," in *Proc. Int. Conf. Digit. Image Comput., Techn. Appl.*, Dec. 2010, pp. 227–234.

[21] A. Geiger, M. Roser, and R. Urtasun, "Efficient large-scale stereo matching," in *Proc. Asian Conf. Comput. Vis.*, 2010, pp. 25–38.

[22] S. Zhang and P. Huang, "High-resolution, real-time 3D shape acquisition," in *Proc. CVPR*, Jun./Jul. 2004, p. 28.

[23] H. Nguyen, D. Nguyen, Z. Wang, H. Kieu, and M. Le, "Real-time, high-accuracy 3D imaging and shape measurement," *Appl. Opt.*, vol. 54, no. 1, pp. A9–A17, 2015.

[24] X. Luo, U. L. Jayarathne, S. E. Pautler, and T. M. Peters, "Binocular endoscopic 3-D scene reconstruction using color and gradient-boosted aggregation stereo matching for robotic surgery," in *Proc. Int. Conf. Image Graph.*, 2015, pp. 664–676.

[25] C. C. Pham, V. Q. Dinh, and J. W. Jeon, "Robust non-local stereo matching for outdoor driving images using segment-simple-tree," *Signal Process., Image Commun.*, vol. 39, pp. 173–184, Nov. 2015.

[26] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *Proc. CVPR*, Jun. 2012, pp. 3354–3361.

[27] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *Proc. CVPR*, Jun. 2015, pp. 3061–3070.

[28] P. J. Fleming and J. J. Wallace, "How not to lie with statistics: The correct way to summarize benchmark results," *Commun. ACM*, vol. 29, no. 3, pp. 218–221, 1986.

[29] Z. Liang *et al.* (2018). "Learning for disparity estimation through feature constancy," [Online]. Available: https://arxiv.org/abs/1712.01039

[30] C. Stentoumis, L. Grammatikopoulos, I. Kalisperakis, and G. Karras, "On accurate dense stereo-matching using a local adaptive multi-cost approach," *ISPRS J. Photogramm. Remote Sens.*, vol. 91, pp. 29–49, May 2014.

[31] S. Hadfield, K. Lebeda, and R. Bowden, "Stereo reconstruction using top-down cues," *Comput. Vis. Image Understand.*, vol. 157, pp. 206–222, Apr. 2017.

[32] N. Ma, Y. Men, C. Men, and X. Li, "Accurate dense stereo matching based on image segmentation using an adaptive multi-cost approach," *Symmetry*, vol. 8, no. 12, p. 159, 2016.

[33] K. Zhang, J. Li, Y. Li, W. Hu, L. Sun, and S. Yang, "Binary stereo matching," in *Proc. ICPR*, 2012, pp. 356–359.

[34] K. Batsos, C. Cai, and P. Mordohai. (2018). "CBMV: A coalesced bidirectional matching volume for disparity estimation." [Online]. Available: https://arxiv.org/abs/1804.01967

[35] A. Li, D. Chen, Y. Liu, and Z. Yuan, "Coordinating multiple disparity proposals for stereo computation," in *Proc. CVPR*, Jun. 2016, pp. 4022–4030.

[36] H. Lu, H. Xu, L. Zhang, and Y. Zhao. (2018). "Cascaded multi-scale and multi-dimension convolutional neural network for stereo matching." [Online]. Available: https://arxiv.org/abs/1803.09437

[37] C. Zhang, Z. Li, Y. Cheng, R. Cai, H. Chao, and Y. Rui, "MeshStereo: A global stereo model with mesh alignment regularization for view interpolation," in *Proc. ICCV*, Dec. 2015, pp. 2057–2065.

[38] J.-R. Chang and Y.-S. Chen. (2018). "Pyramid stereo matching network." [Online]. Available: https://arxiv.org/abs/1803.08669

[39] C. Çiğla, "Recursive edge-aware filters for stereo matching," in *Proc. CVPR*, Jun. 2015, pp. 27–34.

[40] R. A. Hamzah, H. Ibrahim, and A. H. A. Hassan, "Stereo matching algorithm based on per pixel difference adjustment, iterative guided filter and graph segmentation," *J. Vis. Commun. Image Represent.*, vol. 42, pp. 145–160, Jan. 2017.

[41] L. Li, X. Yu, S. Zhang, X. Zhao, and L. Zhang, "3D cost aggregation with multiple minimum spanning trees for stereo matching," *Appl. Opt.*, vol. 56, no. 12, pp. 3411–3420, 2017.

[42] S. K. Gehrig, R. Stalder, and N. Schneider, "A flexible high-resolution real-time low-power stereo vision engine," in *Proc. Int. Conf. Comput. Vis. Syst.*, 2015, pp. 69–79.

[43] W. Wang, J. Yan, N. Xu, Y. Wang, and F.-H. Hsu, "Real-time high-quality stereo vision system in FPGA," *IEEE Trans. Circuits Syst. Video Technol.* vol. 25, no. 10, pp. 1696–1708, Oct. 2015.
[44] S. Michalik, S. Michalik, J. Naghmouchi, and M. Berekovic, "Real-time smart stereo camera based on FPGA-SoC," in *Proc. Humanoids*, Nov. 2017, pp. 311–317.
[45] K.-R. Bae and B. Moon, "An accurate and cost-effective stereo matching algorithm and processor for real-time embedded multimedia systems," *Multimedia Tools Appl.*, vol. 76, no. 17, pp. 17907–17922, 2017.
[46] L. Wang, R. Yang, M. Gong, and M. Liao, "Real-time stereo using approximated joint bilateral filtering and dynamic programming," *J. Real-Time Image Process.*, vol. 9, no. 3, pp. 447–461, 2014.
[47] S. Madeo, R. Pelliccia, C. Salvadori, J. M. del Rincon, and J.-C. Nebel, "An optimized stereo vision implementation for embedded systems: Application to RGB and infra-red images," *J. Real-Time Image Process.* vol. 12, no. 4, pp. 725–746, 2016.
[48] C. Ttofis and T. Theocharides, "High-quality real-time hardware stereo matching based on guided image filtering," in *Proc. DATE*, Mar. 2014, pp. 1–6.
[49] M. Nguyen, W. Q. Yan, R. Gong, and P. Delmas, "Toward a real-time belief propagation stereo reconstruction for computers, robots, and beyond," in *Proc. IVCNZ*, Nov. 2015, pp. 1–6.
[50] V. C. Sekhar, S. Bora, M. Das, P. K. Manchi, S. Josephine, and R. Paily, "Design and implementation of blind assistance system using real time stereo vision algorithms," in *Proc. VLSID*, Jan. 2016, pp. 421–426.
[51] K. J. Lee, K. Bong, C. Kim, J. Park, and H.-J. Yoo, "An energy-efficient parallel multi-core ADAS processor with robust visual attention and workload-prediction DVFS for real-time HD stereo stream," in *Proc. IEEE Symp. Low-Power High-Speed Chips*, Apr. 2016, pp. 1–3.
[52] J. Hofmann, J. Korinth, and A. Koch, "A scalable high-performance hardware architecture for real-time stereo vision by semi-global matching," in *Proc. CVPR*, Jun./Jul. 2016, pp. 845–853.
[53] L. F. S. Cambuim, J. P. F. Barbosa, and E. N. S. Barros, "Hardware module for low-resource and real-time stereo vision engine using semi-global matching approach," in *Proc. SBCCI*, Aug./Sep. 2017, pp. 53–58.
[54] C. Yang, Y. Li, W. Zhong, and S. Chen, "Real-time hardware stereo matching using guided image filter," in *Proc. Int. Great Lakes Symp. VLSI (GLSVLSI)*, May 2016, pp. 105–108.
[55] R. Kalarot and J. Morris, "Comparison of FPGA and GPU implementations of real-time stereo vision," in *Proc. CVPR*, Jun. 2010, pp. 9–15.

QIONG CHANG received the master's degree in engineering from the University of Tsukuba in 2013, where he is currently pursuing the Ph.D. degree with the Graduate School of Systems and Information Engineering. His research interest is in reconfigurable parallel computing systems.

TSUTOMU MARUYAMA received the Ph.D. degree in engineering from The University of Tokyo in 1987. He is currently a Professor with the Graduate School of Systems and Information Engineering, University of Tsukuba. His research interest is in reconfigurable parallel computing systems.