Journal of information and communication convergence engineering
http://jicce.org

J. Inf. Commun. Converg. Eng. 12(2): 128-134, Jun. 2014 Regular paper

GPU-Based Optimization of Self-Organizing Map Feature Matching for Real-Time Stereo Vision

Kajal Sharma, Saifullah, and Inkyu Moon*, *Member, KIICE*

Department of Computer Engineering, Chosun University, Gwangju 501-759, Korea

Abstract

In this paper, we present a graphics processing unit (GPU)-based matching technique for the purpose of fast feature matching between different images. The scale invariant feature transform algorithm developed by Lowe for various feature matching applications, such as stereo vision and object recognition, is computationally intensive. To address this problem, we propose a matching technique optimized for GPUs to perform computations in less time. We optimize GPUs for fast computation of keypoints to make our system quick and efficient. The proposed method uses a self-organizing map feature matching technique to perform efficient matching between the different images. The experiments are performed on various image sets to examine the performance of the system under varying conditions, such as image rotation, scaling, and blurring. The experimental results show that the proposed algorithm outperforms the existing feature matching methods, resulting in fast feature matching due to the optimization of the GPU.

Index Terms: Feature matching, Graphics processing unit, Self-organizing map, Stereo vision

I. INTRODUCTION

Feature selection and matching is a key component in many computer vision tasks, such as path finding, obstacle detection, navigation, and stereo vision [1-4]. Several strategies have been already proposed for keypoint detection [5, 6]. Schmid and Mohr [7] used Harris corners as interest points in image recognition problems to match features against a large database of images. This method allows features to be matched under arbitrary orientation changes, but it is sensitive to image scaling. Lowe [8] proposed a scale-invariant feature transform (SIFT) descriptor for the extraction of interest points from an image that is invariant to both scale and rotation. The SIFT technique, however, uses a 128-dimensional vector to describe the SIFT feature that is computationally intensive. In a recent research [9], we have implemented an efficient feature matching technique, which is faster than Lowe's SIFT, with a self-organizing map (SOM) that can be used for real-time stereo matching applications. In this paper, we extended our research and implement the proposed method with graphics processing units (GPUs) to further optimize the execution time.

Due to the advancements of parallel processing techniques, multi-core CPU methods are widely used to accelerate computationally intensive tasks [10]. Modern programmable graphics hardware contains powerful co-processing GPUs with a peak performance of hundreds of GigaFLOPS, which is an order of magnitude higher than that of the CPUs [11]. For accelerating computer vision applications, many researchers are now exploiting parallelism supported by modern programmable graphics hardware that provides a

Received 09 December 2013, **Revised** 18 March 2014, **Accepted** 09 April 2014
***Corresponding Author** Inkyu Moon (E-mail: inkyu.moon@chosun.ac.kr, Tel: +82-62-230-6033)
Department of Computer Engineering, Chosun University, 375 Seosuk-dong, Dong-gu, Gwangju 501-759, Korea.

Open Access http://dx.doi.org/10.6109/jicce.2014.12.2.128 print ISSN: 2234-8255 online ISSN: 2234-8883

great scope for acceleration to run computations in parallel [12, 13]. Some researchers also use specialized and reconfigurable hardware to speed up these algorithms [14-16]. One example is discovering concurrency in parallel computing, where coloring is used to identify subtasks that can be carried out or data elements that can be updated simultaneously [17]. Yet another example of coloring is the efficient computation of sparse derivative matrices [18]. With the increasing programmability and computational power of GPUs, the recent work by Sinha et al. [10] accelerates some parts of the SIFT algorithm by using the hardware capacities of the GPU. A 10× speedup was obtained, which makes this technique feasible for video applications [10]. A variety of computer vision algorithms have been parallelized, providing significant acceleration to the computation [10, 12, 13].

In this paper, we present a technique that is designed to achieve fast feature matching in images with the use of neural networks and GPUs. Our contribution is the proposal of a GPU-optimized matching technique based on Kohonen's SOM [19]. The presented method provides a significant reduction of computation time as compared to Lowe's SIFT. In our approach, the scale space for keypoint extraction is configured in parallel for detecting the candidate points among which the number of keypoints is reduced with the SOM neural network. The descriptor vector generation is accelerated by the GPU, and matching is accomplished on the basis of competitive learning. The key idea is to optimize keypoint extraction with a GPU and to reduce the descriptor size with the winning calculation method. Similar winning pixels in the images are found and associated to accomplish feature matching. The proposed method using a GPU is faster as multi-processing is used.

The rest of this paper is organized as follows: Section II gives a brief overview of stereo vision. The procedure of feature matching with the GPU-optimized method is presented in Section III. Experimental results are shown in Section IV, while Section V presents the conclusions.

II. STEREO VISION

Stereo vision is based on acquisition of three-dimensional (3D) data from different views obtained by a single moving camera or a fixed arrangement of several cameras. The 3D position of an object is determined by triangulating the optical rays from at least two views of the same object point. Optical axes of two camera-lens units are configured in parallel, and the straight line joining the two optical centers is parallel to each image's horizontal line in order to meet the epipolar constraint (Fig. 1). The 3D image data are obtained on the basis of the position of the points in the left and the right images [20].

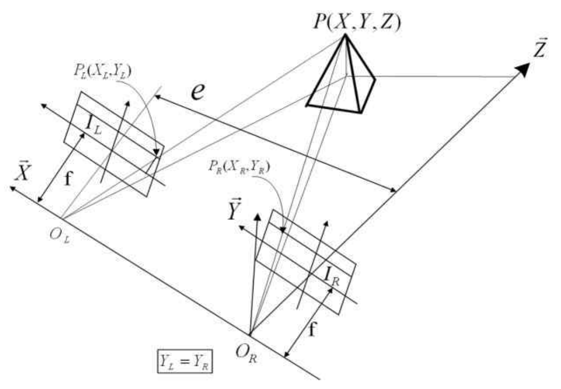

Fig. 1. Stereo vision system acquiring three-dimensional (3D) object point.

The coordinates of a 3D point P of the object (X, Y, Z) are as follows:

$$X = \frac{e(X_L + X_R)}{2\delta}, Y = \frac{e(Y_L + Y_R)}{2\delta}, Z = \frac{ef}{\delta}, \qquad (1)$$

where e denotes the distance between two optical centers, f indicates the focal length of the two lens, and δ refers to the disparity of P. Disparity is defined as the difference in the location of the object point between the left image and the right image. (X_L,Y_L) and (X_R,Y_R) are the coordinates of the projection of the point P in two images, left and right, respectively (Fig. 1). The disparity is given by $\delta = (X_L - X_R)$, i.e., the difference in position in the left and the right images.

III. PROPOSED GPU-OPTIMIZED MATCHING TECHNIQUE

In this section, we explain the proposed GPU-based method to optimize image matching along with the SOM methodology. The GPU is a special-purpose processing unit with a single instruction multiple thread parallel architecture. With the advent of multi-core CPUs and many-core GPUs, mainstream processor chips are now parallel systems. Moreover, their parallelization continues to scale with Moore's law. Compute unified device architecture (CUDA) considers GPU hardware as an independent platform that can provide a programming environment and minimize the need for understanding the graphics pipeline. A GPU hardware chip has N multiprocessors (MPs), and each MP has M scalar processors. The GPU memory is divided into global, shared, and constant. In addition to the three main memories, there are registers, which are on-chip memories (Fig. 2(a)). Variables that reside in registers are accessed at a very high speed in a highly parallel manner. Registers are allocated to individual threads; each thread can only access its own registers [21]. A kernel function typically uses registers to hold the frequently accessed variables that are private to each thread. When the kernel function to be executed with CUDA is ready, a one-block grid must be configured. The block generates a large number of threads to share data with the other threads that ensure parallel processing. The thread is composed of hierarchical SIMD architecture, and a collection of threads is called a thread block, or simply, a block. After the allocation of memory and transferring of data to the GPU, these thread blocks are initialized to execute the assigned parallel jobs. The GPU executes multiple threads in parallel and independently processes vector streams. Fig. 2(b) shows the graphic framework of our proposed GPU-based method for feature matching with SIFT and SOM.

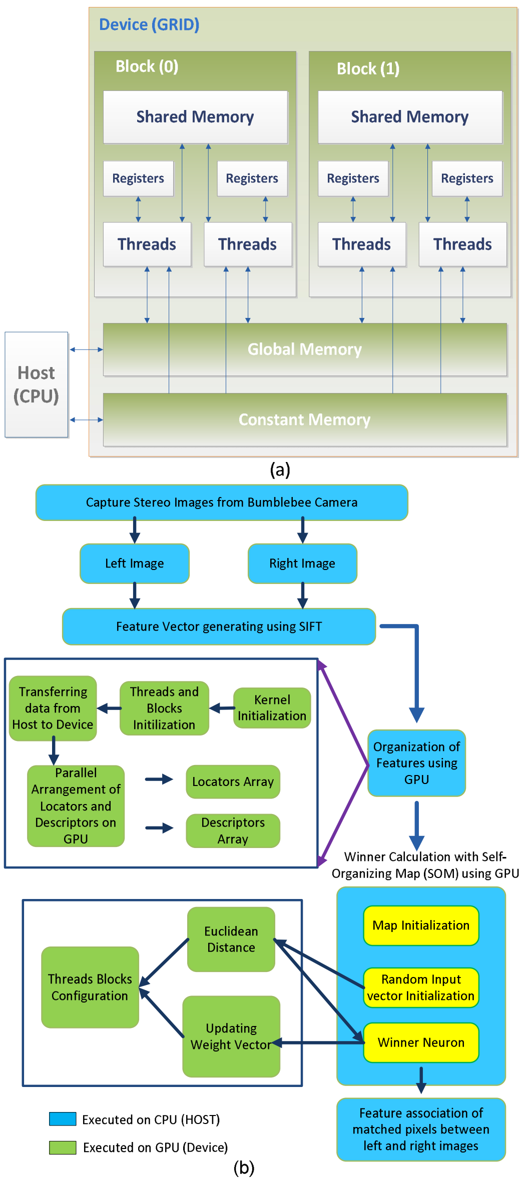
Device (GRID)
Block (0)
Block (1)
Shared Memory
Registers
Threads
Host (CPU)
Global Memory
Constant Memory
(a)
Capture Stereo Images from Bumblebee Camera
Left Image
Right Image
Feature Vector generating using SIFT
Transferring data from Host to Device
Threads and Blocks Initialization
Kernel Initialization
Parallel Arrangement of Locators and Descriptors on GPU
Locators Array
Descriptors Array
Organization of Features using GPU
Winner Calculation with Self-Organizing Map (SOM) using GPU
Euclidean Distance
Threads Blocks Configuration
Updating Weight Vector
Map Initialization
Random Input vector Initialization
Winner Neuron
Feature association of matched pixels between left and right images
Executed on CPU (HOST)
Executed on GPU (Device)
(b)

Fig. 2. Graphics processing unit (GPU) architecture. (a) Memory, thread, and block organization, (b) GPU graphics framework.

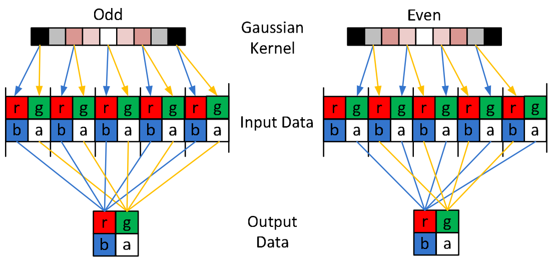

Fig. 3. Gaussian convolution in RGBA 16-GPU format for odd and even samples.

The parts executing on the GPU that would mainly reduce the computation time are described in subsections III-A and III-B. The streams of instructions and fragments (pixels) are designed to be processed in parallel on the GPU with a peak performance of GFLOPS (GigaFLOPS). Fragment processors perform the role of computational kernels, and different computational steps are often mapped to different fragment programs [10]. Texture mapping on the GPU is analogous to the CPU's random read-only memory interface; the fragment processors apply a fragment program (a pixel shader) to each fragment in the stream in order to compute the final color of each pixel. The GPU processes the pixels with parallel processing by the fragment processors and calculates the four color values at once as a vector. The image data are rearranged into a four-channel RGBA image where one color value in the RGBA image represents a 2 × 2 block of the gray-level image, thereby reducing the image size by 4. The RGBA image is processed on the GPU, and the fragment shader arranges them into one RGBA image format (Fig. 3). The Gaussian convolution is directly applied to the RGBA image format where the Gaussian kernel consists of even and odd values to perform semi-convolutions, which are added further to form the final result. The calculations are performed in the fragment shader, and the computational overhead is low because of the rearrangement of the image in the RGBA image format.

A. Parallel Scale Space Configuration and Construction of the Descriptor

In our approach, the construction of a Gaussian scale space is accelerated by the GPU by using fragment programs. In order to extract the candidate keypoint, the scale space $L(x, y, \sigma)$ is computed in parallel by the convolution of the variable-scale Gaussian $G(x, y, \sigma)$ over the input image $I(x, y)$ as follows:

$$\begin{aligned} L(x,y,\sigma) &= G(x,y,\sigma) * I(x,y) \\ &= \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} * I(x,y), \end{aligned} \tag{2}$$

```
1. In the image_i, let the weights of the node for each feature point
   that are corresponding to pixels at (i, j) with intensity $w_3^{ij}$
   be $(w_1^{ij}, w_2^{ij}, w_3^{ij})$.
   A pixel is randomly selected from the image_{i+1}, and feeds
   the corresponding descriptor vector as an input to the SOM network.
   Let the input descriptor vector corresponding to pixels at (m,n) be
   $(\alpha_1^{ij}, \alpha_2^{ij}, \alpha_3^{ij})$

2. The (x,y) winner node represents the pixel in image_i that could be
   a match for the (m,n)th pixel in image_{i+1}:
   $(x, y) = \arg\min_{i,j} \sqrt{\sum_{k=1}^{3} (w_k^{ij} - \alpha_k^{mn})^2}$.

3. Let M and N be the height and the width of the image.
   The first two components of all neuron weight vectors are updated
   as follows:
   $(w_k^{ij} \leftarrow w_k^{ij} + h_k(i', j') g_k(\Delta I)(\alpha_k^{(m+i')(n+j')} - w_k^{ij})$,
   where $i' = i - x,\ j' = j - y h_k(i', j') = \eta_k \exp\left(-\frac{i'^2 + j'^2}{2\sigma_{hk}^2}\right)$
   and $g_k(\Delta I) = \exp\left(-\frac{(\Delta I)^2}{2\sigma_{gk}^2}\right) \Delta I = (w_3^{xy} - w_3^{ij})$
   for k =1, 2, and $\forall i, m \in \{1, 2, \ldots, M\}$, $\forall j, n \in \{1, 2, \ldots, N\}$, the standard
   learning rate is $\eta_k$, and the neighborhood parameters are $\sigma_{hk}, \sigma_{gk}$
   that control the disparity propagation rate in the topological
   neighborhood. The above steps are repeated $N_x$ times where
   $N_x = 100 \times MN$.

4. For each pixel (i, j), the disparity vector $(d_p^{ij}, d_q^{ij})$ is defined as
   $d_q^{ij} = i - w_1^{ij}; d_p^{ij} = j - w_2^{ij}$, corresponding to the vertical and horizontal
   disparity.
```

Fig. 4. Pseudo-code of the presented algorithm.

where * denotes the convolution operation in x and y. Stable keypoint locations in the scale space can be computed from the difference of Gaussians (DOG) separated by a constant multiplicative factor k given by

$$L(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \\ = L(x, y, k\sigma) - L(x, y, \sigma). \tag{3}$$

The intensity image, gradients, and the DOG values are stored in the RGBA packed format and computed in parallel in the same passage by using vector operations in the fragment programs. The Hessian matrix $H(x, y, \sigma)$ is computed by the second-order derivative of the Gaussian blurred image as follows:

$$H(x, y, \sigma) = \begin{bmatrix} L_{xx}(x, y, \sigma) & L_{xy}(x, y, \sigma) \\ L_{xy}(x, y, \sigma) & L_{yy}(x, y, \sigma) \end{bmatrix}, \tag{4}$$

where L_{xy} denotes the second-order derivative of the Gaussian image in both horizontal and vertical directions. An equal number of threads is assigned $\sigma_1, \ldots, \sigma_n$ according to the number of given variables scaled for Gaussian operations in order to simultaneously construct individual pyramids of m octaves. Let mI be the number of features detected in I and D be the dimension of the descriptors (D = 128). A texture of size $mI \times D$ is created and filled with the mI descriptor values, each one occupying a column.

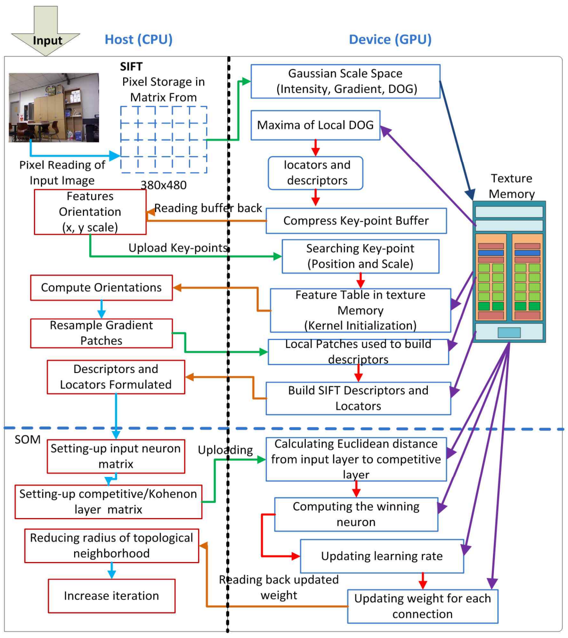
Input
Host (CPU)
Device (GPU)
SIFT
Pixel Storage in Matrix From
380x480
Pixel Reading of Input Image
Gaussian Scale Space (Intensity, Gradient, DOG)
Maxima of Local DOG
locators and descriptors
Texture Memory
Features Orientation (x, y scale)
Reading buffer back
Compress Key-point Buffer
Upload Key-points
Searching Key-point (Position and Scale)
Compute Orientations
Feature Table in texture Memory (Kernel Initialization)
Resample Gradient Patches
Local Patches used to build descriptors
Descriptors and Locators Formulated
Build SIFT Descriptors and Locators
SOM
Setting-up input neuron matrix
Uploading
Calculating Euclidean distance from input layer to competitive layer
Setting-up competitive/Kohenon layer matrix
Computing the winning neuron
Reducing radius of topological neighborhood
Updating learning rate
Reading back updated weight
Increase iteration
Updating weight for each connection

Fig. 5. Complete algorithm of the suggested approach. GPU: graphics processing unit, SOM: self-organizing map, DOG: difference of Gaussians.

B. Winner-Based Image Matching and Acceleration Using GPU

We use the SOM algorithm to map high-dimensional keypoints to a lower dimensional space by using competitive learning [8]. The pseudo-code of the proposed algorithm is shown in Fig. 4. Fig. 5 summarizes the complete algorithm.

To optimize the algorithm, the descriptor and the locator section are implemented on the GPU in order to solve the complexity of feature matching that consumes a considerable amount of time to obtain the descriptor vector. To this end, keypoints are scanned to obtain the descriptor and locator vector that can run in parallel on the GPU. In the initial stage, a function reads an image that returns the SIFT keypoints, which consist of locators and descriptors of an image that are stored in a temporary file. The temporary file is then processed separately for the scanning and organization of the locators and descriptors. The scanning and organization of the locators and descriptors is carried out in a parallel manner on the GPU. The processors consist of a front-end that reads/decodes the data, which is responsible for allocating memory on the GPU and transferring data from the CPU to the GPU, and initializes the kernel instructions. A backend is made up of a group of eight calculating units and two super-function units where instructions are executed in the SIMD mode; the same instruction is sent to all threads in the warp. NVIDIA calls this mode of execution ‘single instruction multiple threads (SIMT)’. The streaming multiprocessors’ operating mode is as follows:

1) At each cycle, a warp ready for execution is selected by the front-end, which starts execution of an instruction.
2) To send the instruction to all 32 threads in the warp, the backend will take four cycles, but since it operates at double the frequency of the front-end, from its point of view, only two cycles will be executed.

The presented approach uses scale invariant feature vectors instead of an image database as an input to the SOM network. A topological map is obtained with the use of the SOM network, and we obtain a 2D neuron grid where each neuron is associated with a weight vector containing 128 element descriptors. The 128-dimensional descriptor generation is accelerated using the GPU in order to increase the execution speed of the algorithm. The descriptor vector is read, and its value is recorded individually in an array; further, the total value is recorded using built-in libraries. The value is later used as a limit for declaring threads and blocks for the GPU. As per the limit of each block, only 512 threads can be accommodated in each block and there can be a total of 65535 blocks in the grid. Each value is assigned to each thread in the block, and the number of blocks depends on the total value divided by 512 (number of threads). An inspection of the temporary file containing the keypoints of each image indicates that the first four values are the locator values followed by the 128 descriptor values. The GPU threads are employed to arrange these locator and descriptor values accordingly.

For image matching, we use a learning algorithm based on the concept of the nearest neighbor learning. One image is considered the reference and the next image as the matching image. Both images are represented in terms of the winning neurons in the SOM network as explained in the pseudo-code. After the network is trained, input data are distributed throughout the grid of neurons. The feature vectors are arranged according to their internal similarity with the SOM, thereby forming a topological map of the input vectors. The winning neuron is found for each pixel of the next image, and the pixel value is associated to it once the winner is found. Feature matching is performed by associating the similar winning pixels between the pixels of the reference image and its neighbor image. By iteratively repeating these steps, winning pixels are obtained with the SOM and the matching between the pixels of the different image pairs is accomplished.

IV. EXPERIMENTAL RESULTS

In this section, we present some experimental results to show the performance of the proposed method. Experiments were performed using images captured with a Kinect camera designed by Microsoft; the experimental details and results of two test image sets are as follows: each image set contained 15 different images, and the performance of the proposed GPU-based method was compared with that of Lowe's method and that of the SOM-based feature matching method [9]. For a comparative analysis of the execution performance, the algorithm was run on a CPU and a GPU independently. Experiments were conducted on an Intel Core i3 processor endowed with an NVIDIA GPU, GeForce 310 with a global memory of 512 MB. The CUDA, proposed by NVIDIA for its graphics processors, exposes a programming model that integrates the host (CPU) and the GPU codes in the same code source files. The main programming introduced by the programming model is an explicitly parallel function invocation (kernel), which is executed by a user-specified number of threads.

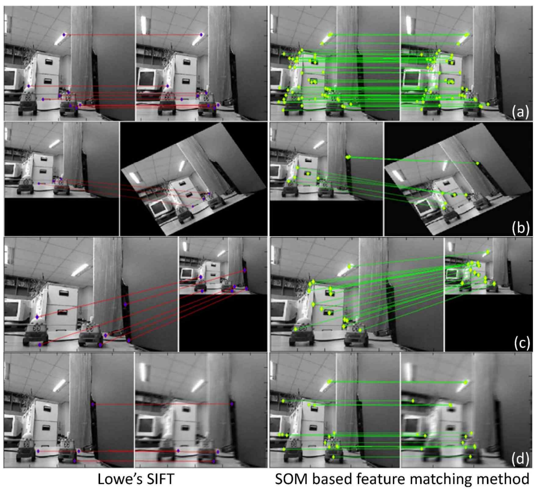

Fig. 6. Test image set 1: feature matching between image 1 and image 2 under varying conditions. (a) Matching between two images, (b) rotation of 30°, (c) scaling with a 0.5 scale factor, and (d) blurring.

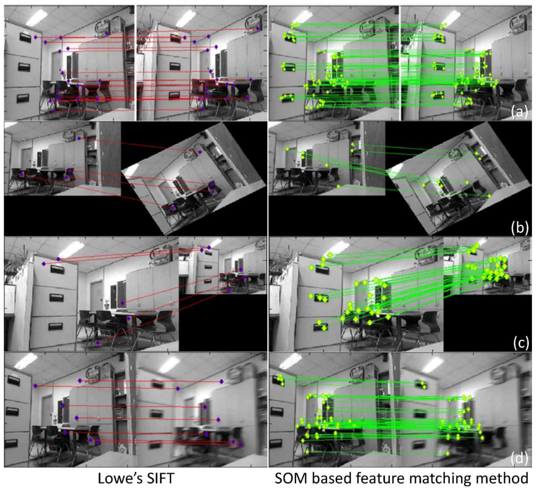

Fig. 7. Test image set 2: feature matching between image 8 and image 9 under varying conditions. (a) Matching between two images, (b) rotation of 30°, (c) scaling with a 0.5 scale factor, and (d) blurring.

Table 1. Comparison results for test image set 1 with Lowe's method, SOM-based feature matching method [9], and the proposed GPU-based method

No.	Set of images	Lowe's method		SOM-based feature matching [9] (CPU)	Proposed GPU-based method
		CPU	GPU [10]		
1	Images 1 & 2	0.1285	0.0156	0.0141	0.0017
2	Images 2 & 3	0.1345	0.0162	0.0156	0.0016
3	Images 3 & 4	0.1058	0.0177	0.0141	0.0017
4	Images 4 & 5	0.1456	0.0199	0.0167	0.0015
5	Images 5 & 6	0.1206	0.0182	0.0134	0.0014
6	Images 6 & 7	0.1612	0.0166	0.0141	0.0016
7	Images 7 & 8	0.1598	0.0199	0.0161	0.0018
8	Images 8 & 9	0.1701	0.0194	0.0145	0.0017
9	Images 9 & 10	0.1234	0.0182	0.0121	0.0014
10	Images 10 & 11	0.1314	0.0192	0.013	0.0015
11	Images 11 & 12	0.1456	0.0181	0.0145	0.0017
12	Images 12 & 13	0.1267	0.0133	0.0141	0.0017
13	Images 13 & 14	0.1312	0.0131	0.0154	0.0016
14	Images 14 & 15	0.1687	0.0155	0.0146	0.0016

GPU: graphics processing unit, SOM: self-organizing map.

Table 2. Comparison results for test image set 2 with Lowe's method, SOM-based feature matching method [9], and the proposed GPU-based method

No.	Set of images	Lowe's method		SOM-based feature matching [9] (CPU)	Proposed GPU-based method
		CPU	GPU [10]		
1	Images 1 & 2	0.1365	0.0152	0.017	0.0019
2	Images 2 & 3	0.1412	0.0165	0.0162	0.0018
3	Images 3 & 4	0.1484	0.018	0.0167	0.0018
4	Images 4 & 5	0.1586	0.0183	0.0176	0.0019
5	Images 5 & 6	0.1707	0.0169	0.0191	0.0021
6	Images 6 & 7	0.1698	0.0191	0.0183	0.0021
7	Images 7 & 8	0.1613	0.0201	0.0181	0.002
8	Images 8 & 9	0.1688	0.0187	0.0187	0.0018
9	Images 9 & 10	0.1568	0.0178	0.0151	0.0017
10	Images 10 & 11	0.1514	0.0182	0.0176	0.0019
11	Images 11 & 12	0.1595	0.0179	0.0169	0.0019
12	Images 12 & 13	0.0968	0.0111	0.01112	0.0013
13	Images 13 & 14	0.1062	0.0123	0.0133	0.0016
14	Images 14 & 15	0.1077	0.0117	0.0134	0.0016

GPU: graphics processing unit, SOM: self-organizing map.

Every CUDA kernel is explicitly invoked by the host code and executed by the device, while the host side code continues the execution asynchronously after instantiating the kernel. The image size for the two test image sets is 480 × 380 pixels. Experiments were conducted under varying situations, such as rotation, scaling, and blurring conditions. Figs. 6 and 7 show the matching results for two image sets obtained using Lowe's method and with the SOM-based feature matching method [9]. The proposed method achieves the same matching results as those described in [9] with much less computation time. It is found that the average matching time is 0.13953 s for Lowe's SIFT, while for the SOM-based feature matching method [9], it is 0.01447 seconds, which is reduced to 0.00165 seconds by using the proposed GPU-based method.

Using the SOM-based feature matching [9] method, we can accelerate Lowe's algorithm by nine times on average, and yet another nine times by using the proposed GPU-based method. Thus, the presented GPU-based method yields a performance improvement of approximately 90 times. Tables 1 and 2 show the comparison results of the computation time on two image sets for different images, in the cases of the Lowe's method, SOM-based feature matching method [9], and the proposed GPU-based method. Experimental results show that the SOM-based feature matching method [9] performs more efficient matching than Lowe's SIFT. A significant reduction in the computation time is obtained by using the proposed GPU-based method.

V. CONCLUSIONS

This paper presents a matching method to obtain the features under varying conditions with reduced processing time. The computation time of the proposed method is reduced as compared to Lowe's method and further optimized by using a GPU. Experiments on various test images have been carried out to evaluate how well the presented method performs on the matching problem as compared to Lowe's method. Experimental results show that the proposed method produces better matching results with a significant reduction in computation times.

REFERENCES

[1] M. Z. Brown, D. Burschka, and G. D. Hager, "Advances in computational stereo," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 8, pp. 993-1008, 2003.

[2] T. Pribanic, N. Obradovic, and J. Salvi, "Stereo computation combining structured light and passive stereo matching," *Optics Communications*, vol. 285, no. 6, pp. 1017-1022, 2012.

[3] C. H. Lee, Y. C. Lim, S. Kwon, and J. H. Lee, "Stereo vision–based vehicle detection using a road feature and disparity histogram," *Optical Engineering*, vol. 50, no. 2, pp. 027004-027004, 2011.

[4] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 4, pp. 509-522, 2002.

[5] J. Shi and C. Tomasi, "Good features to track," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Seattle, WA, pp. 593-600, 1994.

[6] K. Mikolajczyk and C. Schmid, "Scale & affine invariant interest point detectors," *International Journal of Computer Vision*, vol. 60, no. 1, pp. 63-86, 2004.

[7] C. Schmid and R. Mohr, "Local grayvalue invariants for image retrieval," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 5, pp. 530-534, 1997.

[8] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91-110, 2004.

[9] K. Sharma, S. G. Kim, and M. P. Singh, "An improved feature

matching technique for stereo vision applications with the use of self-organizing map," *International Journal of Precision Engineering and Manufacturing*, vol. 13, no. 8, pp. 1359-1368, 2012.

[10] S. N. Sinha, J. M. Frahm, M. Pollefeys, and Y. Genc, "Feature tracking and matching in video using programmable graphics hardware," *Machine Vision and Applications*, vol. 22, no. 1, pp. 207-217, 2011.

[11] K. A. Bjorke, "Image processing on parallel GPU pixel units," *Proceedings of SPIE*, vol. 6065, pp. 606515, 2006.

[12] J. Fung and S. Mann, "OpenVIDIA: parallel GPU computer vision," in *Proceedings of the 13th Annual ACM international conference on Multimedia*, Singapore, pp. 849-852, 2005.

[13] R. Yang and M, Pollefeys, "Multi-resolution real-time stereo on commodity graphics hardware," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition,* Madison, WI, pp. 211-217, 2003.

[14] C. Zach, K. Karner, and H. Bischof, "Hierarchical disparity estimation with programmable graphics hardware," in *Proceedings of the 12th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, Plzen-Bory, Czech Republic, pp. 275-282, 2004.

[15] M. Bramberger, J. Brunner, B. Rinner, and H. Schwabach, "Real-time video analysis on an embedded smart camera for traffic surveillance," in *Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium*, Toronto, Canada, pp. 174-181, 2004.

[16] S. Klupsch, M. Ernst, S. A. Huss, M. Rumpf, and R. Strzodka, "Real time image processing based on reconfigurable hardware acceleration," in *Proceedings of the IEEE Workshop Heterogeneous Reconfigurable Systems on Chip (SoC)*, Hamburg, Germany, p. 1-7, 2002.

[17] M. T. Jones and P. E. Plassmann, "Scalable iterative solution of sparse linear systems," *Parallel Computing*, vol. 20, no. 5, pp. 753-773, 1994.

[18] Y. Saad, "ILUM: a multi-elimination ILU preconditioner for general sparse matrices," *SIAM Journal on Scientific Computing*, vol. 17, no. 4, pp. 830-847, 1996.

[19] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464-1480, 1990.

[20] G. Toulminet, M. Bertozzi, S. Mousset, A. Bensrhair, and A. Broggi, "Vehicle detection by means of stereo vision-based obstacles features extraction and monocular pattern analysis," *IEEE Transactions on Image Processing*, vol. 15, no. 8, pp. 2364-2375, 2006.

[21] D. Kirk and W. Hwu, *Programming Massively Parallel Processors: A Hands-On Approach*. Burlington, MA: Morgan Kaufmann Publisher, 2010.

Kajal Sharma
received her B.E. in Computer Engineering from University of Rajasthan, India, in 2005, and her M.Tech. and Ph.D. in Computer Science from Banasthali University, Rajasthan, India, in 2007 and 2010, respectively. From October 2010 to September 2011, she worked as a postdoctoral researcher at Kongju National University, Korea. Since October 2011, she has been working as a postdoctoral researcher at the School of Computer Engineering, Chosun University, Gwangju, Korea. Her research interests include image and video processing, neural networks, computer vision, and robotics. She has published many research papers in various national and international journals and conferences.

Saifullah
received his B.E. in Electronics Engineering from N.E.D University of Engineering and Technology, Pakistan, in 2011, and his M.S. in Computer Engineering from Chosun University, Gwangju, Korea, in 2013. From September 2011 to June 2013, he worked as a graduate research assistant in the 3D Image Processing Lab of the Computer Engineering Department of Chosun University under the supervision of Dr. Inkyu Moon. As a research assistant, Mr. Saifullah worked on symmetrical cryptography, digital holographic encryption, neural network algorithms, and acceleration of these algorithms on a graphics processing unit. His research interests include image processing, neural networks, algorithm implementation, and computer vision. He has published research papers in various international conferences.

Inkyu Moon
received his B.S. and M.S. in Electronics Engineering from Sungkyunkwan University, Korea, in 1996 and 1998, respectively, and his M.S. and Ph.D. in Electrical and Computer Engineering from University of Connecticut in 2007. From January 2008 to January 2009, he was a researcher in a post-doctoral position at the University of Connecticut. He joined Chosun University, Korea, in 2009, and is currently, an associate professor at the School of Computer Engineering. He has to his credit more than 50 publications, including 30+ peer reviewed journal articles and 20+ conference proceedings (10+ Keynote Addresses and invited conference papers). Dr. Moon is a member of IEEE, OSA, and SPIE. He is on the Editorial Board of the Korea Multimedia Society.