GPU IMPLEMENTATION OF PHASE-BASED STEREO CORRESPONDENCE AND ITS APPLICATION

Mamoru Miura[*], *Kinya Fudano*[†], *Koichi Ito*[*§], *Takafumi Aoki*[*§],
Hiroyuki Takizawa[*§] *and Hiroaki Kobayashi*[‡§]

[*]Graduate School of Information Sciences, Tohoku University, Japan.
E-mail: miura@aoki.ecei.tohoku.ac.jp
[†]NEC Software Tohoku, Ltd., Japan.
[‡]Cyberscience Center, Tohoku University, Japan.
[§]Japan Science and Technology Agency, CREST.

ABSTRACT

This paper proposes a Graphics Processing Unit (GPU) implementation of the stereo correspondence matching using Phase-Only Correlation (POC). The use of high-accuracy stereo correspondence matching based on POC makes it possible to measure accurate 3D shape of the object using stereo vision, while the drawback of POC-based approach is its high computational cost. Addressing this problem, we propose a GPU implementation of POC-based correspondence matching. Through a set of experiments using a variety of GPUs, we demonstrate that the proposed implementation is high-speed and high-efficiency compared with the CPU implementation. We also apply the proposed approach to a real-time 3D measurement system.

Index Terms— stereo vision, stereo correspondence, phase-only correlation, GPU, real-time 3D measurement

1. INTRODUCTION

Image correspondence is one of the important key techniques in the field of computer vision [1]. Especially, the 3D measurement using stereo vision requires an accurate stereo correspondence algorithm, since the accuracy of 3D measurement depends on that of image correspondence between a stereo image pair. The speed of image correspondence is also important in practical applications of the 3D measurement using stereo vision.

To achieve high-accuracy 3D measurement, we have proposed a high-accuracy stereo correspondence matching method using Phase-Only Correlation (POC) and have developed a passive 3D measurement system using stereo vision whose accuracy is comparable with the active 3D measurement system [2]. On the other hand, the high computational cost of the POC-based correspondence matching limits the area of applications, since the computation of POC is based on Fourier transform. Also, in practice, we need to find a lot of corresponding points to measure a high-quality 3D shape.

Addressing this problem, we use a Graphics Processing Unit (GPU) implementation for accurate and fast stereo correspondence matching. The GPU has been very efficient at manipulating and displaying computer graphics. Recently, the highly parallel structure of GPU makes it more effective than general-purpose CPUs for algorithms where processing of large blocks of data can be done in parallel. This effort is known as General-Purpose computation on Graphics Processing Unit (GPGPU) [3]. The GPGPU has been applied to scientific computing and video processing [4]. In this paper, we propose a GPU implementation of the phase-based stereo correspondence matching, since the correspondence can be obtained for each reference point and the most of operations such as Fourier transform can be done in parallel. Through experiments using a variety of GPUs, we demonstrate that the proposed approach is high-speed and high-efficiency compared with the CPU implementation. We also apply the proposed approach to a real-time 3D measurement system.

2. PHASE-BASED CORRESPONDENCE MATCHING

We briefly introduce a Phase-Only Correlation (POC) function (which is sometimes called the "phase-correlation function") [5, 6]. Let $f(n)$ and $g(n)$ be the 1D image signals, where $-M \leq n \leq M$ and the signal length is $N = 2M + 1$. Then, the normalized cross-power spectrum $R(k)$ is defined as

$$R(k) = \frac{F(k)\overline{G(k)}}{|F(k)\overline{G(k)}|} = e^{j(\theta_F(k) - \theta_G(k))}, \quad (1)$$

where $F(k)$ and $G(k)$ are the 1D DFTs of $f(n)$ and $g(n)$, $\overline{G(k)}$ denotes the complex conjugate of $G(k)$, and $-M \leq k \leq M$. The 1D POC function $r(n)$ between $f(n)$ and $g(n)$ is given as the 1D Inverse DFT (1D IDFT) of $R(k)$. When two images are similar, their POC function gives a distinct sharp peak. When two images are not similar, the peak drops significantly. The height of the peak gives a good similarity measure for image matching, and the location of the peak shows the translational displacement between the images. We have also proposed the important techniques for improving the accuracy of 1D image matching for sub-pixel correspondence matching: (i) function fitting for high-accuracy estimation of peak position, (ii) windowing to reduce boundary effects, (iii) spectral weighting for reducing aliasing and noise effects and (iv) averaging 1D POC functions to improve peak-to-noise ratio [2].

In the case of a rectified stereo image pair, the disparity can be limited to horizontal direction [1]. The use of 1D POC makes it possible to achieve high-accuracy correspondence matching with low computational cost. In order to find the accurate correspondence from a stereo image pair, we employ the sub-pixel correspondence matching using POC, which employs a coarse-to-fine strategy using image pyramids for robust correspondence search (Fig. 1) [2]. Let $\boldsymbol{p}$ be a coordinate vector of a reference pixel in the reference image $I(n_1, n_2)$. The problem of sub-pixel correspondence search is to

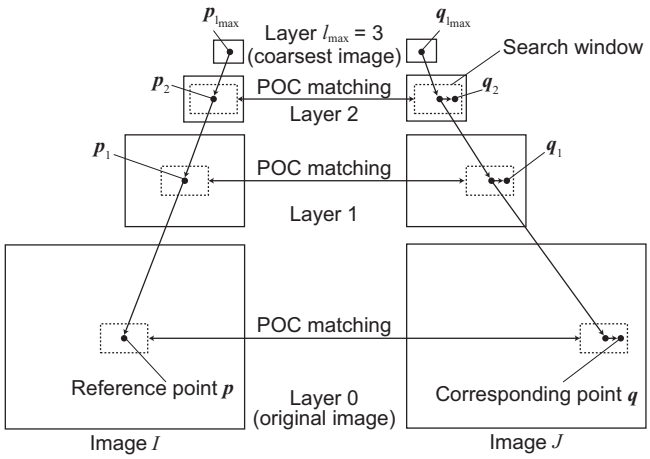

Fig. 1. Framework for high-accuracy correspondence matching.

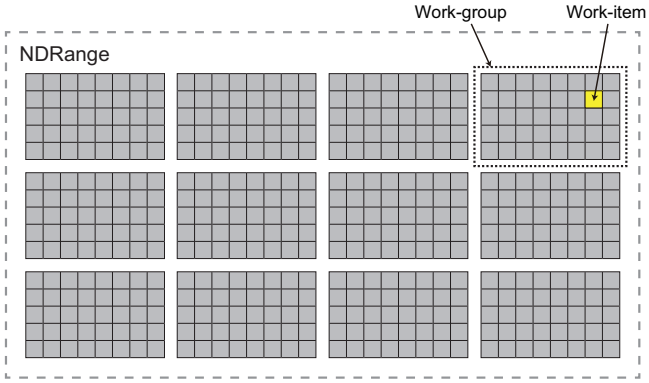

Fig. 2. Data parallel programming model in OpenCL [7].

find a real-number coordinate vector $\boldsymbol{q}$ in the input image $J(n_1, n_2)$ that corresponds to the reference pixel $\boldsymbol{p}$ in $I(n_1, n_2)$. We briefly explain the procedure as follows.

Step 1: For $l = 1, 2, \cdots, l_{\max} - 1$, create the l-th layer images $I_l(n_1, n_2)$ and $J_l(n_1, n_2)$, i.e., coarser versions of $I_0(n_1, n_2)$ and $J_0(n_1, n_2)$, recursively as follows:

$$I_l(n_1, n_2) = \frac{1}{4} \sum_{i_1=0}^{1} \sum_{i_2=0}^{1} I_{l-1}(2n_1 + i_1, 2n_2 + i_2),$$

$$J_l(n_1, n_2) = \frac{1}{4} \sum_{i_1=0}^{1} \sum_{i_2=0}^{1} J_{l-1}(2n_1 + i_1, 2n_2 + i_2).$$

Step 2: For every layer $l = 1, 2, \cdots, l_{\max}$, calculate the coordinate $\boldsymbol{p}_l = (p_{l1}, p_{l2})$ corresponding to the original reference point $\boldsymbol{p}_0$ recursively as follows:

$$\boldsymbol{p}_l = \lfloor \tfrac{1}{2} \boldsymbol{p}_{l-1} \rfloor = (\lfloor \tfrac{1}{2} p_{l-1\,1} \rfloor, \lfloor \tfrac{1}{2} p_{l-1\,2} \rfloor), \quad (2)$$

where $\lfloor z \rfloor$ denotes the operation to round the element of z to the nearest integer towards minus infinity.

Step 3: We assume that $\boldsymbol{q}_{l_{\max}} = \boldsymbol{p}_{l_{\max}}$ in the coarsest layer. Let $l = l_{\max} - 1$.

Step 4: From the l-th layer images $I_l(n_1, n_2)$ and $J_l(n_1, n_2)$, extract two sub-images (or search windows) $f_l(n_1, n_2)$ and $g_l(n_1, n_2)$ with their centers on $\boldsymbol{p}_l$ and $2\boldsymbol{q}_{l+1}$, respectively. The image blocks consist of L lines of N-point 1D signal.

Step 5: Estimate the displacement between $f_l(n_1, n_2)$ and $g_l(n_1, n_2)$ with pixel accuracy using POC-based image matching. Let the estimated displacement vector be $\boldsymbol{\delta}_l$. The l-th layer correspondence $\boldsymbol{q}_l$ is determined as follows:

$$\boldsymbol{q}_l = 2\boldsymbol{q}_{l+1} + \boldsymbol{\delta}_l. \quad (3)$$

Step 6: Decrement the counter by 1 as $l = l - 1$ and repeat from Step 4 to Step 6 while $l \geq 0$.

Step 7: From the original images $I_0(n_1, n_2)$ and $J_0(n_1, n_2)$, extract two image blocks with their centers on $\boldsymbol{p}_0$ and $\boldsymbol{q}_0$, respectively. Estimate the displacement between the two blocks with sub-pixel accuracy using POC-based image matching. Let the estimated displacement vector with sub-pixel accuracy be denoted by $\boldsymbol{\delta} = (\delta_1, \delta_2)$. Update the corresponding point as

$$\boldsymbol{q} = \boldsymbol{q}_0 + \boldsymbol{\delta}. \quad (4)$$

3. GPU IMPLEMENTATION

3.1. GPU Programming Model in OpenCL

In this paper, we employ the GPU implementation based on OpenCL [7] to perform unified performance evaluation of the proposed approach on GPUs developed by NVIDIA and AMD. OpenCL is a framework supporting parallel programming in heterogeneous computational environments such as multi-core CPUs and GPUs. It provides efficient parallel computing using both task-based and data-based parallelism.

Fig. 2 shows a data parallel programming model in OpenCL. The execution model in OpenCL consists of two concepts: a compute kernel and a program. A compute kernel is a basic execution unit in OpenCL. A program is a collection of compute kernels and internal functions. The program invokes a kernel over an index space called an N-Dimensional Range (NDRange). A single kernel instance at a point in the index space is called a work-item. Work-items are also grouped into work-groups as shown in Fig. 2. The data-parallel execution is achieved by executing multiple work-groups in parallel.

Fig. 3 shows a memory model in OpenCL. OpenCL handles four memory spaces such as private, local, constant and global. The global memory permits access to all work-items in all work-groups and has a large amount of capacity with long latency. The local memory is shared by all the work-items in each work-group. The private memory can only be used by a work-item. The constant memory may be used by all the work-groups to store read-only data. The local and private memories can be accessed faster than the global memory, since the substance of the local and private memories is located on each core. Note that the capacity of the local and private memories is small. In the case that the amount of local memory per work-group or private memory per work-item is increased, it results in reducing the number of parallel execution of work-items for each core.

3.2. Implementation Techniques

We present the GPU implementation techniques for the phase-based stereo correspondence algorithm described in Sect. 2.

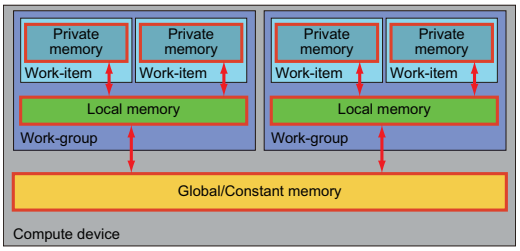

Fig. 3. Memory model in OpenCL [7].

Table 1. Processing time [ms] for each implementation.

	Radeon HD 5870	Radeon HD 6970	GeForce GTX 480	GeForce GTX 580
I1	59.9	58.3	100.2	86.2
I2	41.7	37.9	71.2	60.4
I3	34.1	34.5	39.6	33.6
I4	33.6	31.2	39.5	33.4
I5	32.7	30.2	27.5	23.6

According to the features of OpenCL in Sect. 3.1, the following 5 techniques are used.

(T1) Parallel execution for each reference point and pixel

The stereo correspondence matching is to find a corresponding point on the input image that corresponds to the reference pixel in the reference image. The correspondence matching for each reference point is done in parallel, since this process is independent for each reference point. So, we assign one work-group to the correspondence matching for one reference point. In the process for each reference point, we extract the local block around the reference point and the search window in the input image, and perform the local block matching using POC. The above process is independent for each pixel in the search window. So, we assign one work-item to the process for one pixel in the search window.

(T2) Integration of kernels for POC computation

In general, it is important for the GPU implementation to make as smaller kernels as possible in order to improve the performance of GPU computing with reducing the resources of work-groups and work-items. According to the above knowledge, the computation of POC can be separated into 4 kernels: (i) extracting a search window, (ii) computing FFT, (iii) computing the normalized cross-power spectrum and (iv) computing IFFT. However, in this case, the performance is degraded, since the data for the POC computation are transferred using the global memory whose latency is long. Addressing this problem, we integrate 4 kernels (i)–(iv) into one kernel and transfer the data for the POC computation using the local memory whose latency is shorter than the global memory.

(T3) Look-up table for twiddle factors

In GPUs, the computational cost of transcendental functions such as sin and cos is higher than that of floating-point operations such as addition and multiplication. Also, depending on the variables for the transcendental function, the global memory may be used as a buffer to compute the transcendental function. In the computation of DFT, if the length of signals is known, the twiddle factors can be fixed in advance. When invoking a kernel, twiddle factors are calculated only once, are stored into the local memory, and are referred from the local memory.

(T4) Loop unrolling

In the kernels for stereo correspondence, the process for each line and FFT are implemented by using "for" loops. In GPU, the process with conditional branching such as "if" and "for" is serialized for each branch. In order to avoid such a serialization, we employ the loop unrolling for the "for" loops with the constant number of loops.

(T5) Size of work-group

In T1, the parallel processing is performed using $N \times L$ work-items per one work-group for the search window with $N \times L$ pixels. Depending on the size of the search windows, a large nubmer of resources for each work-group may be consumed and then the number of active work-groups in each core may be reduced. Addressing this problem, we use $N \times L'$ work-items per one work-group and repeat $\lceil \frac{L}{L'} \rceil$ times. Note that we have to empirically determine the optimal L', since the optimal L' depends on the GPU architectures and the size of search windows.

4. EXPERIMENTS AND DISCUSSION

We implement the POC-based stereo correspondence matching on GPUs using the implementation techniques as mentioned in Sect. 3. In the experiments, we use NVIDIA GeForce GTX 480, GeForce GTX 580, AMD Radeon HD 5870 and Radeon HD 6970.

4.1. Performance evaluation of GPU implementations

Table 1 shows the processing time for each implementation, where I1, I2, I3, I4 and I5 in the first column indicate the GPU implementation using only T1, T1–T2, T1–T3, T1–T4 and T1–T5, respectively. The parameters for the POC-based stereo matching are as follows: The size of the search window is 32 pixels $\times$ 15 lines, the number of layers is 4 and the number of reference points is 10,000. As a result, the use of the implementation techniques makes it possible to achieve a 1.9–3.6 times speed-up as compared with the simple GPU implementation.

4.2. Performance evaluation of CPU and GPU implementations

We implement the POC-based stereo correspondence matching on CPUs and GPUs and evaluate the computation time and the power-delay product. The CPU used in the experiments is Intel Core i7-975 (3.3 GHz). We use a single thread (1 core) of the CPU and 8 threads (4 cores) of the CPU. The most time-consuming process for the CPU implementations is FFT. So, we use FFTW [8], which is known as the fastest FFT library for the CPU implementation. The parameters for the stereo correspondence matching is the same in Sect. 4.1. To evaluate the performance, we use 1,000, 5,000 and 10,000 reference points. We measure the power consumption of the whole system during execution with the power meter (HIOKI AC/DC POWER HiTESTER 3334). The power-delay product is the product of the processing time and the power consumption during processing, which measures the energy.

Table 2 shows the processing time and the power-delay product for each implementation. The GPU implementations are 16–23 times faster than the CPU implementation with a single thread and also 4–5 times faster than the CPU implementation with 8 threads (4 cores). The power-delay product of the GPU implementations is smaller than that of the CPU implementations. The above results indicate that the GPU implementations of the POC-based stereo cor-

Table 2. Experimental results (upper: processing time [ms], lower: power-delay products [W×s]).

# of points	CPU 1 thr.	CPU 8 thr.	HD 5870	HD 6970	GTX 480	GTX 580
1,000	59.1	18.4	10.4	8.7	5.7	5.1
	10.9	4.2	2.9	2.8	2.2	2.0
5,000	278.1	74.6	18.8	17.7	15.4	13.3
	51.2	16.9	5.4	5.6	5.8	5.3
10,000	548.8	126.8	32.7	30.2	27.5	23.6
	101.0	28.7	9.3	9.6	10.4	9.3

respondence matching are faster and more efficient than the CPU implementations.

5. APPLICATION TO REAL-TIME 3D MEASUREMENT

In this paper, we develop a real-time 3D face measurement system as one of the applications of the proposed approach.

As shown in Fig. 4, the developed system consists of a stereo camera pair and a computer. The flow diagram of the developed system is shown in Fig. 5. The step of "Face detection" is to extract the face region from the captured image. In the system, we employ the face detection method using the Haar-like features and AdaBoost provided by OpenCV [9]. The step of "Placement of reference points" is to set the reference points on the extracted face region in a reticular pattern with a spacing of 5 pixels. The camera parameters for the stereo rectification and the 3D reconstruction are obtained by the camera calibration in advance.

Fig. 6 shows examples of 3D measurement results. As a result, the accurate 3D shape of the face is measured. Also, the system can measure the 3D face in real-time even if the facial expression is changed. The system can measure 6,000–7,000 3D points in 15 fps which is the frame rate of the camera. As is observed in the above results, the GPU implementation of the POC-based stereo correspondence matching makes it possible to achieve real-time accurate 3D measurement even with the general-purpose computer.

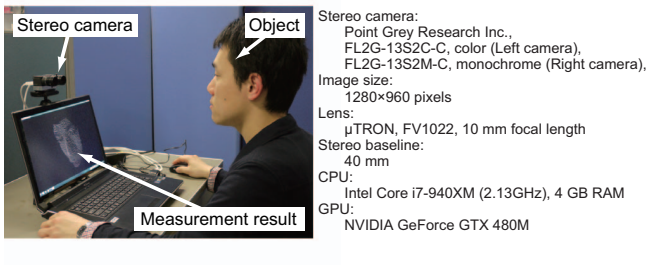

Fig. 4. Real-time 3D face measurement system.

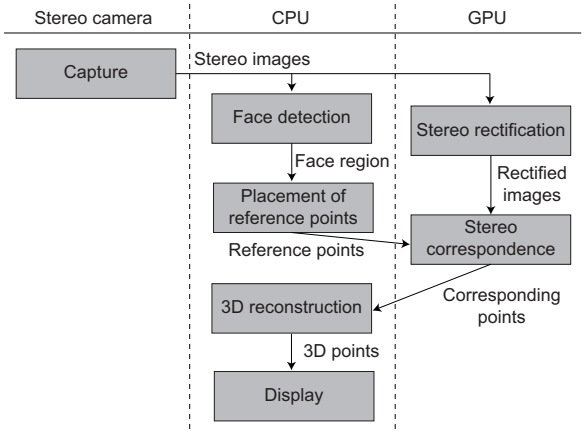

Fig. 5. Processing flow of real-time 3D face measurement system.

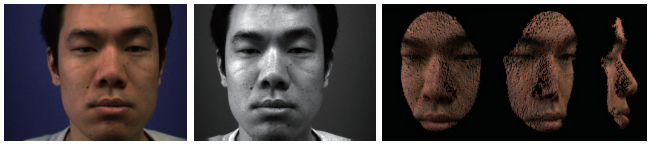

Fig. 6. 3D face measurement: (a) left camera image, (b) right camera image and (c) measurement result.

6. CONCLUSION

This paper has proposed the GPU implementation of the POC-based stereo correspondence matching. Using the optimal implementation techniques for each GPU architecture, the GPU implementations are 16–23 times faster than the CPU implementation with a single thread (1 core) and 4–5 times faster than the CPU implementation with 8 threads (4 cores). We have also developed the real-time 3D face measurement system and demonstrated its effectiveness.

7. REFERENCES

[1] R. Szeliski, *Computer Vision: Algorithms and Applications*, Springer-Verlag New York Inc., 2010.

[2] T. Shibahara, T. Aoki, H. Nakajima, and K. Kobayashi, "A sub-pixel stereo correspondence technique based on 1D phase-only correlation," *Proc. Int'l Conf. Image Processing*, pp. V–221–V–224, 2007.

[3] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A.E. Lefohn, and T.J. Purcell, "A survey of general-purpose computation on graphics hardware," *Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, 2007.

[4] "GPGPU.org," `http://gpgpu.org`.

[5] C. D. Kuglin and D. C. Hines, "The phase correlation image alignment method," *Proc. Int'l Conf. Cybernetics and Society*, pp. 163–165, 1975.

[6] K. Takita, T. Aoki, Y. Sasaki, T. Higuchi, and K. Kobayashi, "High-accuracy subpixel image registration based on phase-only correlation," *IEICE Trans. Fundamentals*, vol. E86-A, no. 8, pp. 1925–1934, Aug. 2003.

[7] Khronos Group Std., "The OpenCL Specification, Version 1.2," `http://www.khronos.org/registry/cl/specs/opencl-1.2.pdf`.

[8] M. Frigo and S.G. Johnson, "The design and implementation of FFTW3," *Proc. the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.

[9] "Open Computer Vision Library," `http://sourceforge.net/projects/opencvlibrary/`.