A VGA 30-fps Optical-Flow Processor Core Based on Pyramidal Lucas and Kanade Algorithm

Hajime Ishihara, Masayuki Miyama, Yoshio Matsuda
Graduate School of Natural Science and Technology
Kanazawa University
Kanazawa, Ishikawa, 920-1192, Japan
E-mail: hajime@mics.ee.t.kanazawa-u.ac.jp

Yuichiro Murachi, Yuki Fukuyama, Ryo Yamamoto,
Junichi Miyakoshi, Hiroshi Kawaguchi,
Masahiko Yoshimoto
Department of Computer and Systems Engineering
Kobe University
Kobe, Hyogo, 657-8501, Japan

***Abstract*-This paper describes an optical-flow processor core for real-time video recognition. The processor is based on the Pyramidal Lucas and Kanade algorithm. It has small chip area, a high pixel rate, and high accuracy compared to conventional optical-flow processors. Introduction of search range limitation and the Carman filter to the original algorithm improves the optical-flow accuracy and reduces the processor hardware cost. Furthermore, window interleaving and window overlap methods can reduce the necessary clock frequency of the processor by 70%. The proposed processor can handle a VGA 30-fps image sequence with 332 MHz clock frequency. The core size and power consumption in 90-nm process technology are estimated respectively as 3.50 × 3.00 mm^2 and 600 mW.**

I. INTRODUCTION

An optical flow is a motion vector of a pixel between two successive images; that flow is the basis of video recognition. Fig. 1 depicts an image sequence: *Yosemite* and its optical flows. Using the optical flow, moving objects in an image sequence or movement of a camera itself can be detected. Fig. 2 shows various applications using optical flows. An optical flow is useful for vehicle safety systems, robot systems, medical systems, and surveillance systems.

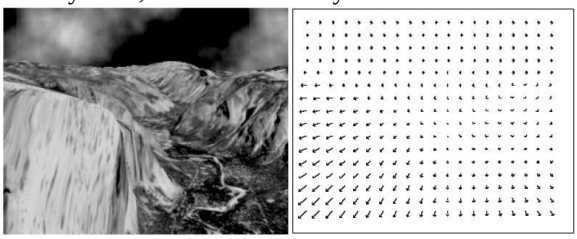

Fig. 1. *Yosemite* and its optical flows.

In optical-flow calculations, several equations must be solved for every pixel. The computational cost reaches a few tens of GOPS, even in a CIF 30-fps (352 × 288 pixels per frame and 30 frames per second) image sequence. Consequently, software approaches using generally available processors have examined only a small part of an image; alternatively, such approaches have neglected accuracy. For higher resolution real-time operation, dedicated hardware is required. Moreover, scalability in the pixel rate and accuracy is preferable for an optical flow processor because the required pixel rate and accuracy differ among application areas.

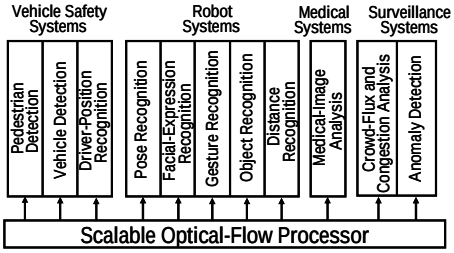

Fig. 2. Applications using optical flows.

Several optical-flow processors have been developed [1]–[3]. Fig. 3 depicts a comparison of the proposed processor to conventional ones in terms of accuracy (MAE: Mean Angle Error) and the pixel rate. Here, *L*, *W*, and *K* respectively denote a hierarchical level, a window size, and an iteration count, as stated later. The HOE processor can handle a CIF 30-fps image sequence, but it needs a large memory capacity of about 1.2 MBytes [1], which will result in a huge memory for higher resolution image sequence.

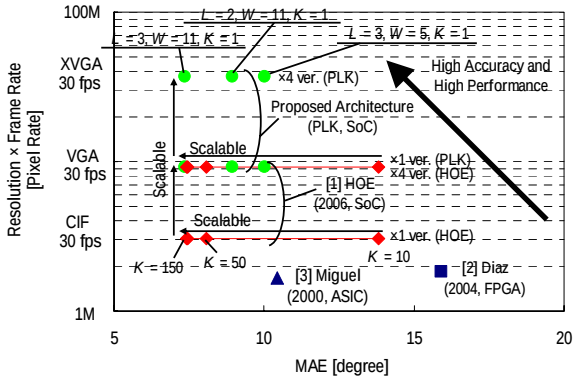

Fig. 3. Performance comparison.

This paper describes the optical-flow processor core based on the Pyramidal Lucas and Kanade (PLK) algorithm [4]. The PLK processor can handle a VGA 30-fps image sequence with far less memory capacity. Its MAE is 7.36° for the *Yosemite* image sequence, which is equal accuracy to that of the HOE processor. The PLK processor provides both the highest pixel rate and accuracy. The PLK processor architecture has wide scalability in terms of pixel rate and accuracy: it can handle an

XVGA 30-fps image sequence by connecting four processors in parallel. In addition, it can save its power consumption in low accuracy applications by appropriately choosing the values of algorithm parameters.

II. Pyramidal Lucas and Kanade (PLK) Algorithm

The PLK algorithm is released in the OpenCV library by Intel Corp.; it applies a hierarchical scheme to the Lucas and Kanade algorithm [5] to handle large movement of objects. The PLK algorithm has been adopted for our VLSI implementation because this algorithm has lower computational cost, less memory size, and higher accuracy than other optical-flow algorithms [5]–[8].

An optical flow **u** in the PLK algorithm is defined as a vector to minimize the following residual function $E(\mathbf{u})$:

$$E(\mathbf{u}) = \sum (I(\mathbf{r}) - J(\mathbf{r}+\mathbf{u}))^2, \quad (1)$$

where I and J are luminance values of the first and second images of two successive ones, respectively and the summation is over a region centering at position **r**. The region is referred as a window A on the first image and a window B on the second image. The window B has displacement **u** (the optical flow) to the window A. In a linear approximation, (1) leads to (2):

$$\mathbf{Gu} = \mathbf{b}, \quad \mathbf{G} = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}, \quad (2)$$

where the spatial gradient matrix and a mismatch vector are respectively represented as **G** and **b**. The luminance gradients of x, y, and t coordinates are respectively denoted as I_x, I_y, and I_t. Using (2), the optical flow is computed iteratively with the Newton-Raphson method. Fig. 4 depicts a flowchart of the PLK algorithm, where L denotes a hierarchical level and K an iteration count. First, hierarchical images are generated in a recursive fashion. Then, I_x and I_y are computed from pixel data in the window A and I_t from ones in the windows A and B. Then, **G** and **b** are computed to produce an optical flow. These steps are repeated iteratively at a hierarchical level. The position of the window B varies at every iteration step depending on the previous optical flow. Furthermore, this procedure is repeated from the uppermost level to the 1st level (raw image). Finally, the optical flow is obtained.

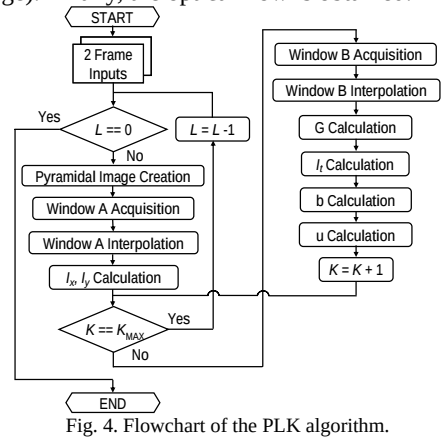

Fig. 4. Flowchart of the PLK algorithm.

III. PLK Algorithm Optimization for VLSI Implementation

In the PLK algorithm, a window B at the L-th level is determined using the (L+1)-th level optical flow. The search range of an optical flow will become large proportionately if the computed optical flow will be large. This increases the size of the memory on a chip. Fig. 5 shows a proposed search range limitation method, which configures the upper limit value of an optical-flow. The method reduces the number of pixels necessary to compute the optical flow of one pixel. It requires only 18 kBytes memory. The method also enhances optical-flow accuracy. The PLK algorithm assumes small movement of the flow, so a large value of the flow is likely to be false. The method described here reduces false detections and enhances the flow accuracy. In addition, the Carman Filter [9] is adopted for computing I_t, as shown in (3). The filter improves MAE by 0.1°. Introduction of these methods to the original PLK algorithm both improves accuracy and reduces the memory size.

$$I_t = \frac{3}{4}(I(\mathbf{r}) - J(\mathbf{r}+\mathbf{u})). \quad (3)$$

Fig. 6 shows an accuracy comparison of the PLK algorithm according to parameters. The parameter set of L (hierarchical level) = 3, W (window size) = 11, K (iteration count) = 1 is adopted for our VLSI implementation. The algorithm optimization with the above parameter set improves MAE by 0.59° and reduces the memory size by 96% compared to the original PLK algorithm.

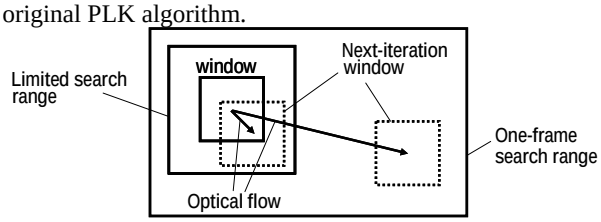

Fig. 5. Search range limitation method.

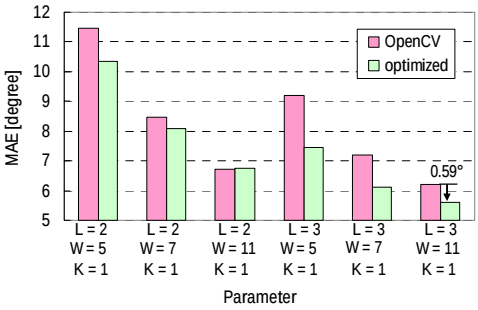

Fig. 6. Accuracy comparison of the PLK algorithm. This MAE is an average value of MAEs of four image sequences: Translating-tree (Trans), Diverging-tree (Div), *Yosemite* (Yos), and Original Composite Sequence.

IV. VLSI Architecture

A. PLK Optical-flow Processor

Fig. 7 shows a block diagram of the PLK optical-flow processor, which comprises a pyramidal image creation (PIC), a spatial gradient matrix (SGM), a mismatch vector (MMV),

an optical flow (OPF), and so on. Each block is a pipeline stage and operates in parallel. Because the window B at the L-th level is determined using the (L+1)-th level optical flow, the MMV can not start computing the L-th level optical flow until the (L+1)-th level optical flow is obtained. It causes pipeline stall, as shown in Fig. 8(a). The window interleaving method is proposed as shown in Fig. 8(b). Because an optical-flow calculation corresponding to a pixel is independent of a calculation corresponding to another pixel, the optical-flow calculation of the other pixel can be inserted into idle cycles in Fig. 8(a). Thanks to this method, pipeline stall does not occur and the clock frequency is reduced by 65%.

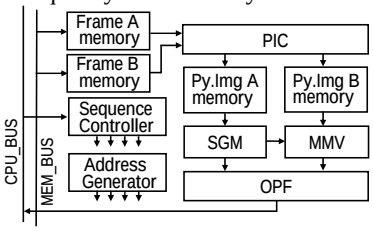

Fig. 7. Block diagram of the PLK optical-flow processor core.

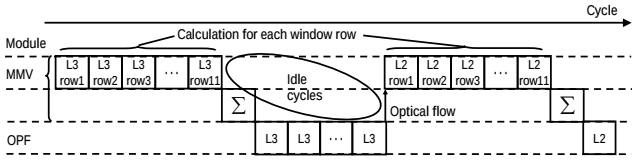

Fig. 8(a). Timing chart (conventional).

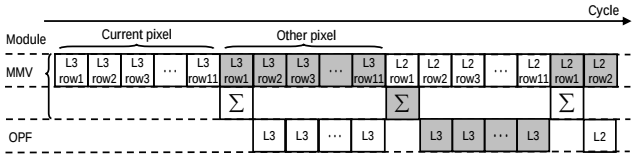

Fig. 8(b). Timing chart (window interleaving method).

B. Pyramidal Image Creation (PIC)

Fig. 9 shows a block diagram of the PIC. This block generates a hierarchical image by sub-sampling and 5 × 5 Gaussian filtering, as shown in Fig. 10. The filtering is made by addition and bit-shifting.

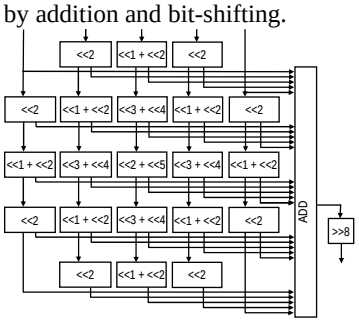

Fig. 9. Block diagram of the PIC.

1/256	4/256	6/256	4/256	1/256
4/256	16/256	24/256	16/256	4/256
6/256	24/256	36/256	24/256	6/256
4/256	16/256	24/256	16/256	4/256
1/256	4/256	6/256	4/256	1/256

Fig. 10. 5 × 5 Gaussian filter.

C. Spatial Gradient Matrix (SGM)

Fig. 11 shows a block diagram of the SGM, which comprises interpolation A (IPAs), spatial gradient (SPGs), gradient matrix element (GMEs), and a summation (SUM). First, pixel data of a window A are acquired from the Py.Img A memory (a first frame hierarchical image corresponding to the window A), as shown in Fig. 7. Next, the IPAs interpolate luminance values at decimal pixels. The SPGs calculate I_x and I_y. The GMEs calculate elements of a spatial gradient matrix; they are added for each row at the SUM. By repeating these steps for iterations equal to the number of window rows, **G** is derived; it is the total of spatial gradient matrices.

Most pixels in the window corresponding to a calculating pixel and in the next window corresponding to a next pixel are identical. Most pixel data can be used in common in computing the optical flow at neighboring pixels. Fig. 12 portrays a window overlap method. This method reduces the memory size and clock cycles to read pixels. It reduces the clock frequency by 15% from that using the window interleaving method alone.

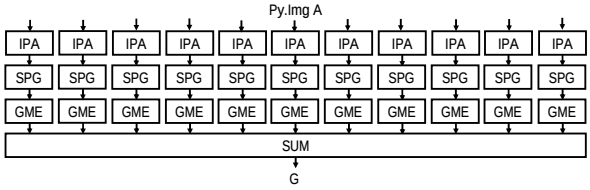

Fig. 11. Block diagram of the SGM.

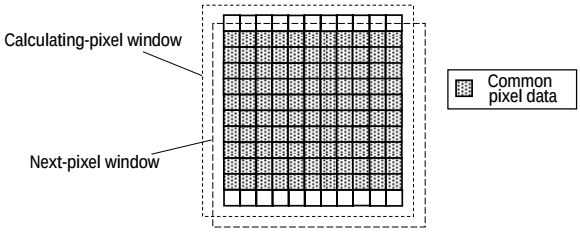

Fig. 12. Window overlap method.

D. Mismatch Vector (MMV)

Fig. 13 shows a block diagram of the MMV, which comprises interpolation B (IPBs), mismatch vector (MVs) and a summation (SUM). First, pixel data of window B are acquired from the Py.Img B (a second-frame hierarchical image corresponding to the search range), as shown in Fig. 7. Next, IPBs interpolate luminance values at decimal pixels estimated using an upper level optical flow. The MVs calculate I_t from pixel data of both window A and window B with the Carman filter in (3). Then, mismatch vectors of each pixel are calculated from I_x, I_y, and I_t. Finally, as with the SGM, by adding mismatch vectors, **b** is derived, which is the total of mismatch vectors.

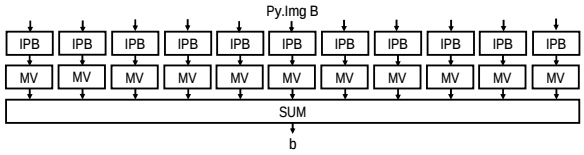

Fig. 13. Block diagram of the MMV.

E. Optical Flow (OPF)

Fig. 14 shows a block diagram of the OPF. This comprises a calculation of the denominator and numerator (CDN), divider (DIVs), and an UPDATE (update). First, the CDN executes a 32-bit multiplication with four 16-bit multipliers in a four-stage pipelined multiplication. Next, DIVs execute 32-bit

division using a subtraction shift recovery algorithm. The DIV can calculate a 1-bit quotient per clock cycle. Because the bit-length of an optical flow is 24, division with the DIV requires 24 clock cycles per optical flow. The DIV must finish calculation at every six clock cycles when $W = 5$ (the smallest window size in the proposed processor) because an optical flow is produced per six clock cycles. Four DIVs are placed in parallel to handle this occasion. Finally, at the UPDATE, an optical flow is updated by adding an upper level optical flow and this optical flow.

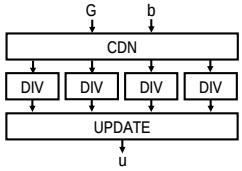

Fig. 14. Block diagram of the OPF.

F. Scalability

The proposed processor has scalability in terms of pixel rate and accuracy. By connecting four processors in parallel, it can handle an XVGA 30-fps image sequence. Fig. 15 shows its scalable architecture. Each processor is independent except for input from a memory bus. Each processor receives pixel data of the same region and calculates optical flows at different places. Therefore, optical flows can be computed with reduced clock frequency and without increasing the bus-bandwidth. In addition, the processor operates at lower clock frequency in less accuracy applications by setting the values of the algorithm parameters to be smaller than those of the optimized ones; as a result, power consumption can be saved.

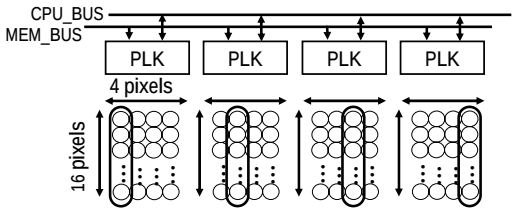

Fig. 15. Scalable Architecture.

V. VLSI Implementation

Fig. 16 shows a PLK processor core layout, which is designed in 90-nm process technology. Then, the respective area sizes of the SGM, the MMV, and the OPF are 1.50 × 0.84 mm^2, 1.50 × 0.70 mm^2, and 0.50 × 0.84 mm^2. The core size, which includes all blocks, is estimated within 3.50 × 3.00 mm^2. Table I shows a performance comparison of the PLK and the HOE processors. The PLK processor achieves real-time processing of a VGA30-fps image sequence with smaller chip size than that of the HOE. Total power consumption of the SGM, the MMV, and the OPF is estimated at 600 mW with the parameters of $L = 3$, $W = 11$ and $K = 1$. Accuracy of the PLK is equivalent to that of the HOE.

VI. Conclusion

An optical-flow processor core for real-time video recognition based on the PLK algorithm is described in this paper. For VLSI implementation, introducing the search range limitation and the Carman filter as computing the temporal luminance gradient optimizes the PLK algorithm. The optimized PLK algorithm provides accuracy which is equivalent to that of the HOE algorithm, with improved MAE by 0.59°, and memory size reduced by 96% for parameters of $L = 3$, $W = 11$ and $K = 1$ (see TABLE I). Moreover, introduction of window overlap and window interleaving methods reduces the PLK processor clock frequency by 70%. The core size is estimated as 3.50 × 3.00 mm^2 in 90-nm process technology; it can handle a VGA 30-fps image sequence at 332 MHz clock frequency and 600 mW power consumption. Therefore, the proposed optical-flow processor is applicable to several application fields of real-time video recognition tasks such as those for vehicle safety, robotics, medical care, and surveillance.

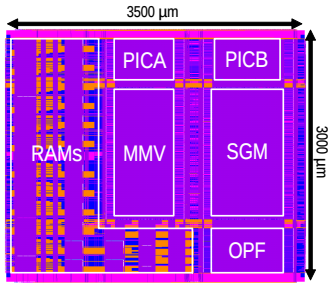

Fig. 16. PLK Processor Core Layout.

TABLE I
Comparison of Performance

		PLK		HOE
Prameter		$L = 3, W = 11, K = 1$		$K = 150$
Resolution		VGA 30 fps	CIF 30 fps	CIF 30 fps
Frequency		332 MHz	110 MHz	189 MHz
Power		600 mW	198 mW	500 mW
Size	Logic	590 kGate		311 kGate
	Memory	18 kByte		1200 kByte
	Area	10.5 mm^2		30 mm^2
MAE	Trans	0.47 °		0.65 °
	Div	2.87 °		2.75 °
	Yos	7.36 °		7.44 °

Acknowledgment

This work was supported by the Semiconductor Technology Academic Research Center (STARC), and VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Cadence Design Systems, Inc. and Synopsys, Inc.

References

[1] N. Minegishi et al., "VLSI Architecture Study of a Real-time Scalable Optical Flow Processor for Video Segmentation", IEICE Transactions on Electronics, Vol.E89-C, No. 3, pp.230-242, 2006.

[2] J. Diaz, E. Ros, S. Mota, F. Pelayo, and E. M. Ortigosa, "Real-time optical flow computation using FPGAs", Proc. Early Cognitive Vision Workshop, 2004.

[3] M. V. Correia and A. C. Campilho, "Real-Time Implementation of an Optical Flow Algorithm", ICPR, Vol.4, pp.247-250, 2000.

[4] J. Y. Bouguet, "Pyramidal Implementation of the Lucas Kanade Feature Tracker Description of the Algorithm", Intel Corporation, Microprocessor Research Labs, OpenCV Documents, 1999.

[5] B. D. Lucas and T. Kanade, "An Iterative Image Registration Technique With An Application to Stereo Vision", Proc. DARPA Image Understanding Workshop, pp.121-130, 1981.

[6] T. Yamamoto, K. Imamura and H. Hashimoto, "Improvement of Optical Flow by Moving Object Detection using Temporal Correlation", Trans. IIITE, Vol.55, No.6, pp.907-911, 2001.

[7] B. K. P. Horn and B. G. Schunck, "Determining Optical Flow", Artificial Intelligence, Vol.17, pp.185-204, 1981.

[8] J. L. Barron, D. J. Fleet and S. S. Beauchemin, "Performance of Optical Flow Techniques", International Journal of Computer Vision, Vol.12, No. 1, pp.43-77, 1994.

[9] Carman. Neustaedter, "An Evaluation of Optical Flow using Lucas and Kanade's Algorithm", University of Calgary Department of Computer Science, Calgary, AB, T2N 1N4, Canada, 2002.