IEICE TRANS. INF. & SYST., VOL.E93–D, NO.12 DECEMBER 2010

PAPER

A VGA 30 fps Affine Motion Model Estimation VLSI for Real-Time Video Segmentation

Yoshiki YUNBE[†], *Nonmember*, **Masayuki MIYAMA**[†a)], *Member*, *and* **Yoshio MATSUDA**[†], *Nonmember*

SUMMARY This paper describes an affine motion estimation processor for real-time video segmentation. The processor estimates the dominant motion of a target region with affine parameters. The processor is based on the Pseudo-M-estimator algorithm. Introduction of an image division method and a binary weight method to the original algorithm reduces data traffic and hardware costs. A pixel sampling method is proposed that reduces the clock frequency by 50%. The pixel pipeline architecture and a frame overlap method double throughput. The processor was prototyped on an FPGA; its function and performance were subsequently verified. It was also implemented as an ASIC. The core size is $5.0 \times 5.0\,\mathrm{mm}^2$ in $0.18\,\mu$m process, standard cell technology. The ASIC can accommodate a VGA 30 fps video with 120 MHz clock frequency.

key words: *affine motion model, motion estimation, video segmentation, real-time processing, VLSI, FPGA*

1. Introduction

Video recognition is necessary for various applications such as vehicle safety systems, robot systems, and surveillance systems [1], [2]. Motion estimation is an important technology for video segmentation, which is an important basis of video recognition [3]–[5]. Affine motion model estimation is a motion estimation technique [6]. With affine motion model estimation, one estimated model corresponds to one target region in an image. The estimated model with six affine parameters can express motions of all pixels in the target region.

Several affine motion model estimation techniques have been proposed [6]–[8]. In general, accurate motion model estimation is difficult if the target region includes complex motion or multiple objects, because one motion model can not express two or more motions. Therefore, the Pseudo M-estimator (PSM) algorithm has been proposed as the robust affine motion model estimation technique [7]. The PSM algorithm is a robust estimation method using the M-estimator concept. It can estimate only a dominant motion of a target region with elimination of outlier motions by weighting for each pixel. Therefore, a region with the dominant motion can be distinguished from regions with other motions. The PSM algorithm has been applied to video segmentation [9]. This video segmentation algorithm gives good results in comparison with the other methods [10], [11].

Manuscript received September 2, 2009.
Manuscript revised June 8, 2010.

[†]The authors are with the Graduate School of Natural Science and Technology, Kanazawa University, Kanazawa-shi, 920–1192 Japan.
a) E-mail: miyama@t.kanazawa-u.ac.jp

DOI: 10.1587/transinf.E93.D.3284

However, the affine motion model estimation based on the PSM algorithm entails enormous computational costs. The applications of the PSM algorithm require real-time operation with high resolution to increase the recognition accuracy. Real-time processing is impossible using software approaches in this case. Dedicated hardware is required for real-time operation with high resolution. A dedicated VLSI processor for the PSM algorithm has not been proposed.

This paper describes an affine motion estimation VLSI based on the PSM algorithm. Introduction of an image division method and a binary weight method to the original algorithm reduces data traffic between the processor and external memories. Furthermore, these methods reduce the hardware cost. In addition, a pixel sampling method can reduce the clock frequency by 50%. Pixel pipeline architecture and a frame overlap method double throughput of the processor. The proposed VLSI architecture can handle a VGA 30 fps video with 120 MHz clock frequency. The processor was prototypically implemented on an FPGA and verified in terms of function and performance. It was also implemented as an ASIC. The core size is $5.0 \times 5.0\,\mathrm{mm}^2$ in $0.18\,\mu$m process, standard cell technology.

This paper is organized as follows: the following section introduces the motion models with affine parameters, and the robust multi-resolution method used for the estimation. Section 3 describes the algorithm optimization for VLSI implementation. In Sect. 4, VLSI architecture that accommodates a VGA 30 fps video with 120 MHz clock frequency is presented. Section 5 describes our VLSI implementation and its experimental results. Finally, Sect. 6 contains concluding remarks. This paper is an extension work of our previous paper [12]. An ASIC implementation is newly presented. Simulation results and experimental results are added.

2. Affine Motion Estimation Algorithm

2.1 Motion Model with Affine parameters

In affine motion estimation, a motion is expressed using two-dimensional affine transformation. An affine motion model $\boldsymbol{A}$, estimated from two successive images, comprises six affine parameters as follows.

$$\boldsymbol{A}^t = (a_1\ a_2\ a_3\ a_4\ a_5\ a_6) \tag{1}$$

The combination of these parameters can express various motions. Figure 1 shows motion expression with the affine

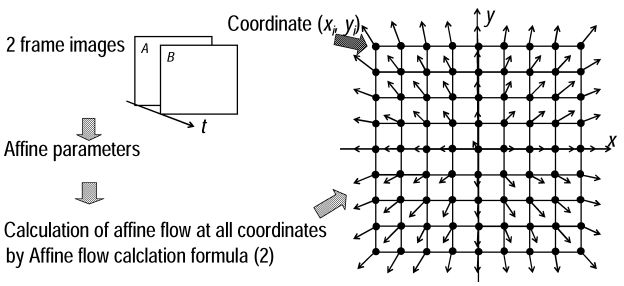


Fig. 1 Example of motion expression by affine parameters.

parameters. Motion vectors for each pixel are calculable as

$$\begin{cases} u_i = a_1 + a_2 x_i + a_3 y_i \\ v_i = a_4 + a_5 x_i + a_6 y_i \end{cases} \tag{2}$$

where u and v respectively signify motion vector elements in the x and y directions. The suffix i is a pixel index. Here, a set of affine parameters expresses an outspreading motion.

2.2 Pseudo M-estimator (PSM) Algorithm

The PSM algorithm is a robust estimation method using the M-estimator concept. To derive a motion model, an error function is defined as follows.

$$r_i = J(x_i + u_i, y_i + v_i) - I(x_i, y_i) + \xi \tag{3}$$

Therein, I and J respectively represent a current image and the subsequent image. This equation fundamentally assumes the luminance conservation law. Parameter ξ represents the global luminance change, arising irrespective of the object motion. A motion model $\boldsymbol{\theta}$ of a region F is defined as a vector to minimize Eq. (3) over all pixels in F with the weighted least squares method. It is expressed as follows.

$$\begin{aligned} \boldsymbol{\theta} &= \boldsymbol{G}^{-1} \cdot \boldsymbol{G_s} \\ &= \left(\sum_{(x_i,y_i)\in F} w_i \boldsymbol{\chi}_i^t \boldsymbol{\chi}_i \right)^{-1} \left(\sum_{(x_i,y_i)\in F} \boldsymbol{\chi}_i^t \delta_i \right) \\ &\begin{cases} \boldsymbol{\theta} = (\boldsymbol{A}^t\ \xi) = (a_1\ a_2\ a_3\ a_4\ a_5\ a_6) \\ \boldsymbol{\chi}_i = (I_x\ I_x x_i\ I_x y_i\ I_y\ I_y x_i\ I_y y_i\ 1) \\ \delta_i = -I_t \end{cases} \end{aligned} \tag{4}$$

In those equations, the luminance gradients in x, y, and t directions at the point indexed by i are denoted respectively as I_x, I_y, and I_t. The weight for each pixel is represented as w_i. The luminance gradient matrices are represented as $\boldsymbol{G}$ and $\boldsymbol{G_s}$.

The PSM algorithm introduces multi-resolution images to cope with large motion. The algorithm also introduces a weight for each pixel to estimate a motion model with eliminating outlier motions. Weights for each pixel are calculated at the bottom resolution level. Then they are propagated to the upper resolution level. An evaluation measure for each pixel q_i is defined as follows.

$$q_i = \frac{\sum_{(x_i,y_i)\in\eta_i} \|\nabla I(x_i, y_i)\| DFD(x_i, y_i)}{\sum_{(x_i,y_i)\in\eta_i} \|\nabla I(x_i, y_i)\|}$$

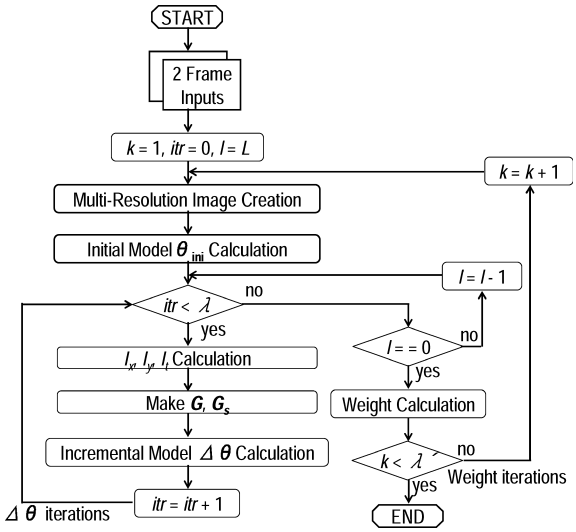


Fig. 2 Flowchart of the PSM algorithm.

$$\begin{aligned} &DFD(x_i, y_i) \\ &\quad = I(x_i + u_i, y_i + v_i, t + 1) - I(x_i, y_i, t) + \xi \end{aligned} \tag{5}$$

Therein, ∇I represents the spatial gradient and η_i represents the eight-neighborhood of the target pixel. Then, the weight w_i is determined with comparison of q_i to a threshold value C as follows.

$$w_i = \begin{cases} (C^2 - q_i^2)^2 & (\text{if } |q_i| < C) \\ 0 & (\text{otherwise}) \end{cases} \tag{6}$$

The threshold value C is selected a priori as $0 < C \leq 20$.

Figure 2 portrays a flowchart of the PSM algorithm. Introduction of the multi-resolution estimation scheme, which estimates a motion model at each resolution, can handle large movement of objects. First, multi-resolution images are generated in a recursive fashion. Then the estimation is started from the top resolution level. A variable l is a level counter. A parameter L represents the uppermost level. The motion model is calculated iteratively at each resolution level to improve the estimation accuracy ($\Delta\boldsymbol{\theta}$ iterations). A variable itr is a loop counter for the $\Delta\boldsymbol{\theta}$ iteration. A parameter λ is the maximum number for the itr. After the motion model calculation at the bottom resolution level finishes, weight calculation begins. These steps are repeated iteratively (Weight iterations); then the correct motion model is finally obtained. A variable k is a loop counter for the Weight iteration. A parameter λ' is the maximum number for the k.

3. VLSI Algorithm

3.1 Image Division Method

We next consider the affine motion estimation of a VGA 30 fps video. An internal memory to store the whole VGA image is impractical because it would require a large chip. An external memory to store the VGA image is also difficult to implement because of the huge data traffic between

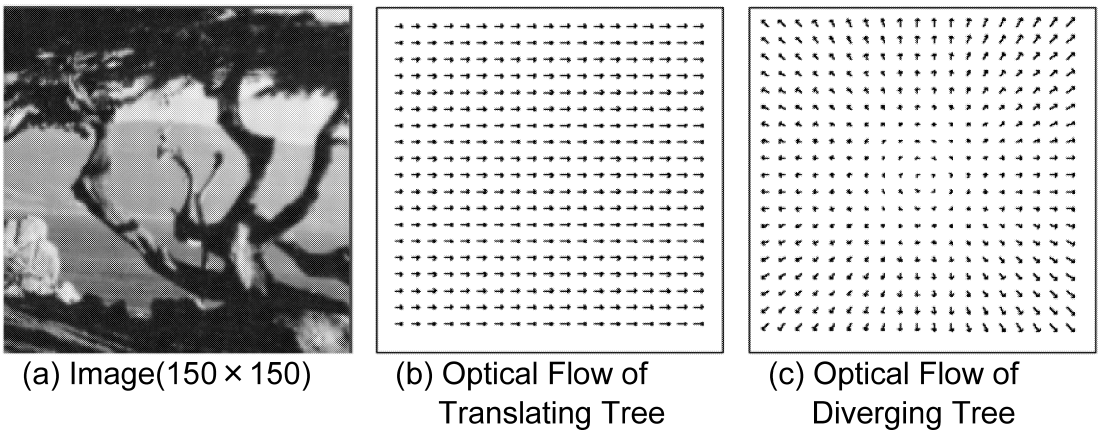


Fig. 4 Translating tree and diverging tree test sequences.

Table 1 Comparison of data traffic and memory amount.

	Original	Proposed
Internal memory (kbyte)	50.4	41.6
External memory (kbyte)	787.2	0
Data traffic (Mbps)	18,855.936	82.944

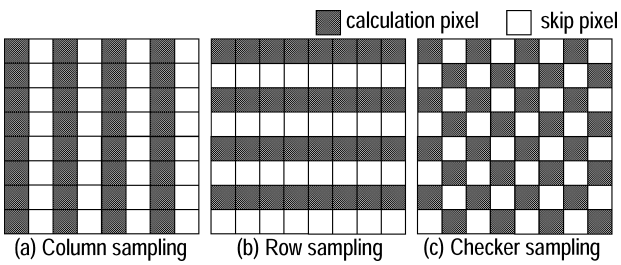


Fig. 3 Three-pattern pixel sampling method of motion model calculation.

the processor and the memory. The image division method is proposed to solve these problems. This technique divides a large image into numerous small images; then the motions for each divided image are estimated independently. Table 1 presents a comparison of the data traffic and memory amount between the original and proposed method. It is assumed that the VGA image is divided into 20 (128×128) images, and that 12 pixels are transferred from the memory to the processor in every cycle for parallel processing. The proposed method reduces both the memory amount and data traffic.

3.2 Pixel Sampling Method

Computational costs of the motion model calculation depend on the number of pixels. Figure 3 shows three pixel-sampling methods: column sampling (a), row sampling (b), and checker sampling (c). Introduction of the proposed methods can halve the number of calculation pixels. Consequently, the computational cost of motion model calculation can be approximately halved.

3.3 Binary Weight Method

The original method shown in Eq. (6) gives large weight to dominant motion; the weight gradually decreases concomitantly with the outlier influence. The threshold value C is selected within $0 < C \leq 20$. The maximum weight (=20) takes 18-bit length per pixel. Weights of all pixels must be kept in memory until completion of the motion estimation. Consequently, enormous amounts of memory are required to store all weights. In addition, calculation of the gradient matrix element uses the weight; thereby the bit length of the matrix element also increases. To solve these problems, binary weighting is proposed as

$$w_i = \begin{cases} 1 & (\text{if } |q_i| < C) \\ 0 & (\text{otherwise}). \end{cases} \tag{7}$$

3.4 Simulation Results

Test sequences of Translating Tree, Diverging Tree, and Yosemite [13] were simulated to verify the proposed method. The Translating Tree and the Diverging Tree are depicted in Fig. 4. The Translating Tree is a sequence with motion panning to the right (Fig. 4 (b)). The Diverging Tree is a sequence with motion outspreading from the center (Fig. 4 (c)). The Yosemite is a sequence that includes complex motion, as depicted in Fig. 5. The upper part moves to the right, the lower right part moves to the lower right, and the lower left part moves to the lower left. In this case, one motion model cannot express all these motions. These sequences were synthesized by computers; and then a correct optical flow for each sequence is known before estimation. The mean angle error MAE and the mean magnitude error MME are adopted to evaluate the motion accuracy. The MAE and MME are obtained with comparison of calculated motion vectors to correct motion vectors as

$$\text{MAE} = \frac{1}{N} \sum_x \sum_y \left(\arccos \frac{\boldsymbol{v}_c \cdot \boldsymbol{v}_e}{\|\boldsymbol{v}_c\| \|\boldsymbol{v}_e\|} \right), \tag{8}$$

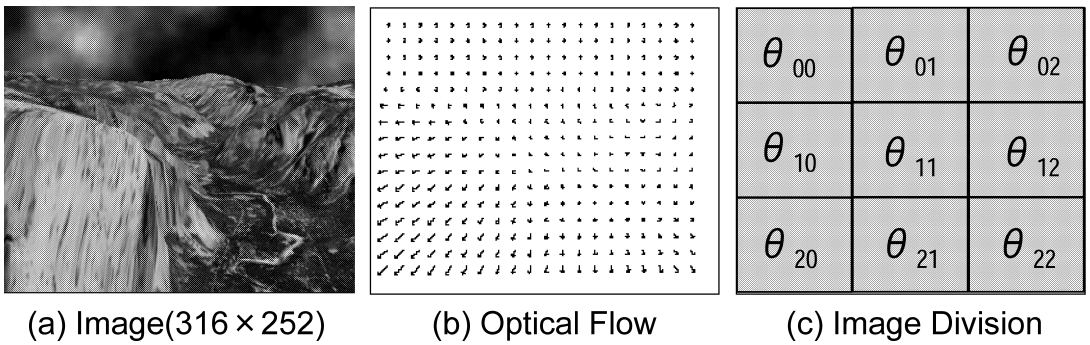


Fig. 5 Yosemite sequence and image division method.

Table 2 Simulation result of image division method.

Test Sequence	MAE(degree)			MME(pixel)		
	Original	4	9	Original	4	9
Translating	0.145	0.237	0.917	0.016	0.020	0.057
Diverging	2.940	1.474	2.008	0.058	0.022	0.019
Yosemite	36.707	17.210	11.152	0.613	0.357	0.393

Table 3 Simulation result of pixel sampling methods.

Test Sequence	MAE(degree)				MME(pixel)			
	Original	Column	Row	Checker	Original	Column	Row	Checker
Translating	0.145	0.252	0.211	0.139	0.016	0.021	0.017	0.014
Diverging	2.940	3.170	2.990	2.934	0.058	0.065	0.069	0.058
Yosemite	36.707	36.798	37.159	36.814	0.613	0.619	0.437	0.433

$$\mathrm{MME} = \frac{1}{N}\sum_{x}\sum_{y}(\|\boldsymbol{v}_c - \boldsymbol{v}_e\|). \tag{9}$$

The algorithm parameters L, λ, λ', C were 1,4,4, and 20, respectively.

The simulation result of the image division method is shown in Table 2. In the Table, 4 and 9 are the number of divided images. Degradation of the motion accuracy in the Translating Tree is slight. The method rather improves the accuracy in the Yosemite and the Diverging Tree sequences. In general, it is difficult to represent the whole complex motion in an image only by a motion model. The image division method fits a motion model to each divided image, whose motion become relatively simple through the division. The whole complex motion can be represented by the set of motion models better than a motion model obtained with the original method. The PSM algorithm also has an advantage of estimating smooth motion at the boundary between the divided images [7]. The PSM algorithm with the division method does not degrade the motion accuracy thanks to these features. Table 3 shows a comparison of the accuracy among pixel sampling methods. Compared to the original, the accuracy degradation for each method is slight. Table 4 presents a comparison of the accuracy between two methods related to the weight. The binary weight method's accuracy almost equals that of the original method. Comparison between the original method and the proposed method composed of the all above methods is presented in Table 5. In the proposed method, the checker sampling is applied. The numbers of image divisions in the Translating Tree, the Diverging Tree, and the Yosemite are 4,4, and 9, respectively. Without accuracy degradation, the proposed method decreases the circuit scale and the operating frequency.

Table 4 Simulation result of binary weight method.

Test Sequence	MAE(degree)		MME(pixel)	
	Original	Proposed	Original	Proposed
Translating	0.145	0.153	0.016	0.009
Diverging	2.940	2.940	0.058	0.046
Yosemite	36.707	35.240	0.613	0.605

Table 5 Accuracy comparison of original and proposed.

Test Sequence	MAE(degree)		MME(pixel)	
	Original	Proposed	Original	Proposed
Translating	0.145	0.252	0.016	0.017
Diverging	2.940	1.664	0.058	0.022
Yosemite	36.707	11.624	0.613	0.452

4. VLSI Architecture

4.1 PSM Processor Architecture

Figure 6 shows a block diagram of the PSM processor, which is comprised of Multi Reso. Image Creations, a Motion Model Calculation, a Weight Calculation, a sequence controller, and memories for images and weights. Each processing block operates pixel-by-pixel in a pipeline fashion. The sequence controller controls a sequence of calculations. It generates control signals connecting to other blocks and instructs them to behave properly with the signals.

As described previously, the weight calculation uses

the motion model; motion model calculation uses the weight. Therefore, the dependency between these calculations causes pipeline stall, as portrayed in Fig. 7 (a). The frame overlap method is proposed as depicted in Fig. 7 (b). The motion model calculation of the next frame is inserted into idle cycles, as shown in Fig. 7 (a). By virtue of this method, pipeline stall does not occur: the processor throughput approximately doubles. The processor based on the pixel pipeline architecture and the frame overlap method estimates the motion model of a VGA 30 fps video with 120 MHz clock frequency.

The control module of the proposed processor comprises MR_control and MW_controls. The MR_control controls the Multi Reso. Image Creations. Also, MW_control_A controls the Motion Model Calculation and the Weight Calculation between the image A and B; MW_control_B controls the same circuits between images B and C. Frame overlap processing is controlled with mutual signaling, as presented Fig. 8.

4.2 Multi Reso Image Creation

Multi Reso Image Creation generates multi-resolution images by sub-sampling and 5×5 Gaussian filtering as shown in Fig. 9 (a). Figure 9 (b) shows a block diagram of the filter, which comprises adders and bit-shifting. In the truth, shifters are not necessary because the shift numbers are fixed. The Multi Reso Image Creation operates pixel-by-pixel in a pipeline fashion, and calculates four pixel values of upper level image in parallel.

4.3 Motion Model Calculation

Figure 10 shows a block diagram of the Motion Model Calculation, which calculates the gradient matrix elements with all pixels of the current frame in the raster scan order. First, a destination coordinate after moving is calculated using the

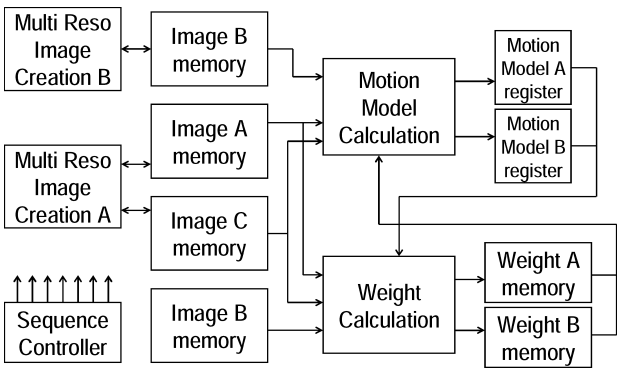


Fig. 6 Block diagram of the PSM processor.

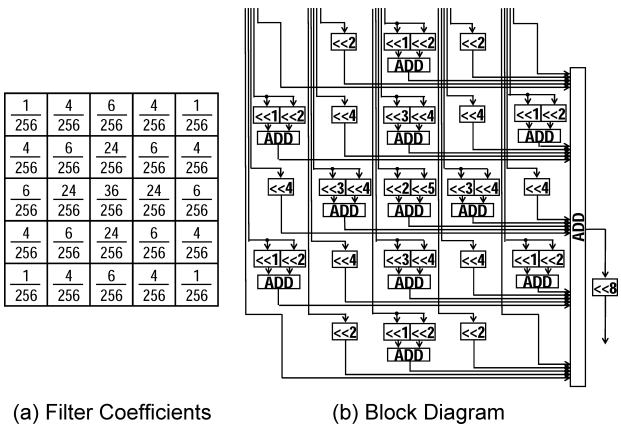


Fig. 9 5×5 Gaussian filter.

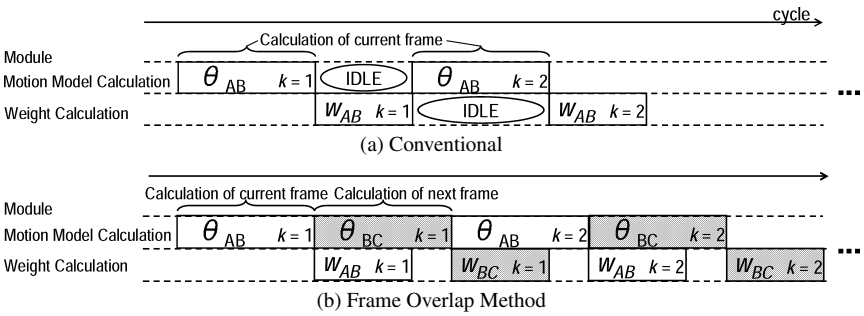


Fig. 7 Timing diagram of the PSM processor.

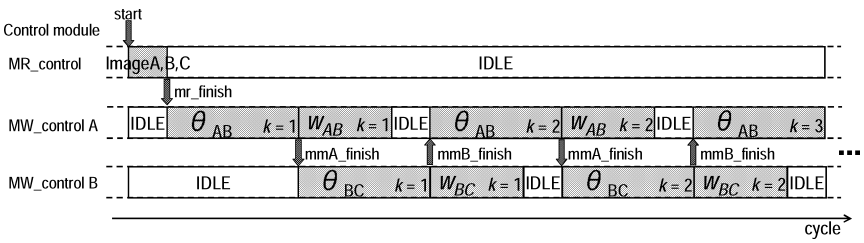


Fig. 8 Frame overlap control.

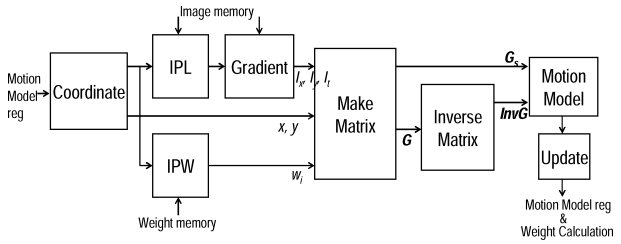


Fig. 10 Block diagram of the motion model calculation.

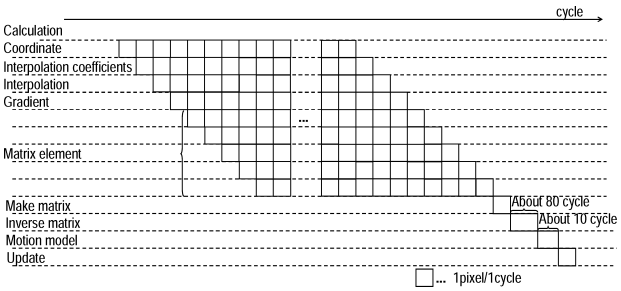


Fig. 11 Timing diagram of the motion model calculation.

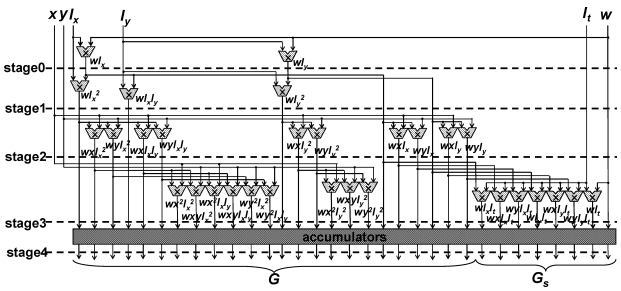


Fig. 12 Block diagram of the make matrix.

motion model by Coordinate. Next, the luminance and the weight of the destination at the decimal coordinate are interpolated using IPL and IPW. Luminance gradient I_x, I_y, and I_t are calculated by Gradient in parallel using the interpolated luminance from the IPL. The luminance gradients feed into Make Matrix. All elements of the matrices $\boldsymbol{G}$ and $\boldsymbol{G_s}$ are calculated by the Make Matrix in parallel. Inverse Matrix calculates the inverse matrix $\boldsymbol{InvG}$ from the gradient matrix $\boldsymbol{G}$, and Motion Model calculates an incremental motion model from $\boldsymbol{InvG}$ and $\boldsymbol{G_s}$. Finally, Update updates the motion model.

The timing chart of the motion model calculation is presented in Fig. 11. The gradient matrix is calculated pixel by pixel in a pipeline fashion. This calculation uses most of the cycles. The inverse matrix calculation, the motion model calculation, and the motion model update are done only once for every iteration.

4.4 Make Matrix

Figure 12 shows a block diagram of the Make Matrix. The circuit is composed of 5 pipeline stages. The circuit calculates all elements corresponding to 1 pixel every 1 clock cycle in parallel. Common multipliers among the elements are shared in the early stages of the pipeline to reduce the circuit scale.

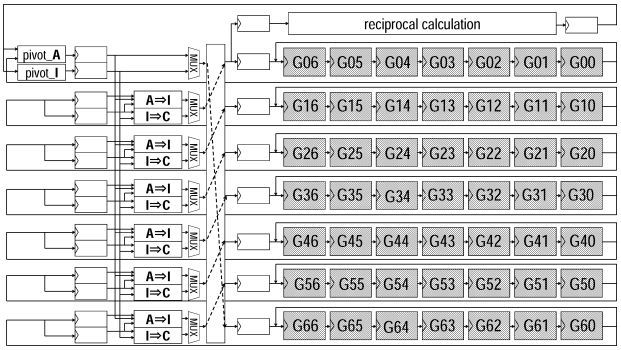


Fig. 13 Inverse matrix arithmetic circuit.

4.5 Inverse Matrix

The Motion Model Calculation has an Inverse Matrix circuit based on the Gauss-Jordan algorithm. This algorithm produces $N \times 2N$ matrix $\boldsymbol{GI}$ by concatenating two $N \times N$ matrices $\boldsymbol{G}$ and $\boldsymbol{I}$. The corner elements $\boldsymbol{GI}(i, i)$ of the matrix $\boldsymbol{GI}$ are called pivots. First, for $i = 0$, each element $\boldsymbol{GI}(i, j)$ of the row including the current pivot (pivot row) is calculated as

$$\boldsymbol{GI}(i, j) = \boldsymbol{GI}(i, j)/pivot. \tag{10}$$

The element $\boldsymbol{GI}(i, i)$ becomes 1 by Eq. (10). Next, each element $\boldsymbol{GI}(i', j)$ of the row i' other than the pivot row is calculated as follows.

$$\boldsymbol{GI}(i', j) = \boldsymbol{GI}(i', j) - \boldsymbol{GI}(i, j) \times \boldsymbol{GI}(i', i) \tag{11}$$

The elements $\boldsymbol{GI}(i', i)$ becomes 0 by the Eq. (11). These calculations are repeated until $i = N - 1$ with incrementing i by 1. Finally, the $\boldsymbol{G}$ part of the matrix $\boldsymbol{GI}$ is transformed into $\boldsymbol{I}$; the $\boldsymbol{I}$ part of the matrix $\boldsymbol{GI}$ is transformed into the inverse matrix $\boldsymbol{InvG}$.

The proposed circuit comprises a reciprocal calculation circuit, an inverse calculation circuit, and a matrix element array, as presented in Fig. 13. First, the reciprocal calculation circuit calculates the reciprocal number of a pivot. Next, the inverse calculation circuit calculates one-column elements by one cycle. The elements $\boldsymbol{GI}(i, j)$ of the pivot row i are calculated according to the Eq. (10). The elements $\boldsymbol{GI}(i', j)$ of the row i' other than the pivot row are calculated according to Eq. (11). These calculations are repeated over all columns. Each element of the obtained columns is moved up by one row and is stored in the row of the matrix element array. Using this mechanism, the next pivot row $i + 1$ is always stored in the top row of the matrix element array. The uppermost circuit in the inverse calculation circuit always computes Eq. (10). The other lower circuits always compute Eq. (11). Each circuit need not switch the calculations

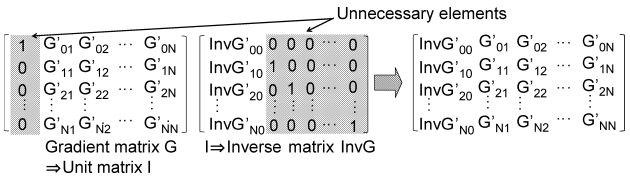


Fig. 14 Register reduction technique.

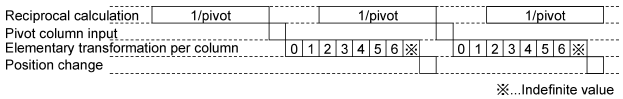


Fig. 15 Timing chart of the inverse matrix arithmetic circuit.

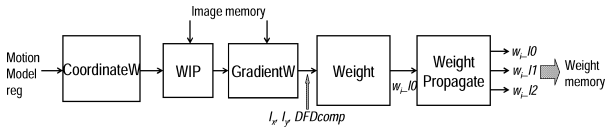


Fig. 16 Block diagram of the weight calculation.

of Eqs. (10) and (11). Consequently, the circuit scale is reduced. The original algorithm processes $2N$ columns to one pivot. However, the calculation and reservation of $N \times N$ elements are unnecessary, as indicated in Fig. 14. Thereby the matrix element array registers are halved. Figure 15 portrays a timing chart of the inverse matrix calculation. The reciprocal and inverse calculations are executed in parallel to reduce idle cycles. The proposed circuit calculates the inverse matrix of 7×7 in 77 cycles.

4.6 Weight Calculation

Figure 16 shows a block diagram of the Weight Calculation. In this block, the weight is calculated at every pixel of the bottom resolution level using an estimated motion model. First, a destination coordinate after moving is calculated by Coordinate. Next, a luminance value of the destination at the decimal coordinate is interpolated by WIP, and GradientW calculates the luminance gradients using the interpolated luminance values. Subsequently, the weight evaluation measure qi is calculated; the weight w_i is determined by Weight. Finally, Weight Propagate propagates the calculated weights to the upper resolution level. One pixel weight on the upper level is calculated by the rounding operation of four pixel weights on the lower level. The Weight Calculation calculates four pixels in parallel and operates in a pipeline fashion. Figure 17 shows an overlap method of weight calculation and weight propagation. Using this method, the weight calculation and the weight propagation can be processed in parallel.

4.7 Memory Composition

The luminance gradient at the destination coordinate after moving must be calculated. In most cases, the destination coordinate is decimal, so the luminance at the decimal coordinate is interpolated using the surrounding pixels. Therefore, 12 pixels around the target pixel must be read as presented in Fig. 18. For higher performance, a memory composition that can read several pixels from the arbitrary part simultaneously is desirable. The random access capability is crucial because the destinations for each pixel mutually differ. Figure 19 depicts the proposed memory division method. The necessary 12 pixels are legible from the arbitrary part by dividing the image memory into 16, as presented in Fig. 19.

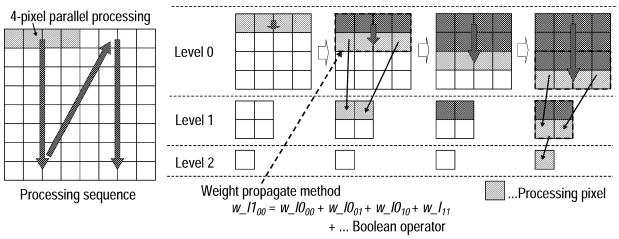


Fig. 17 Weight calculation procedure.

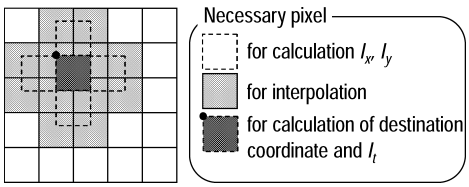


Fig. 18 Necessary pixels for calculation of luminance gradient.

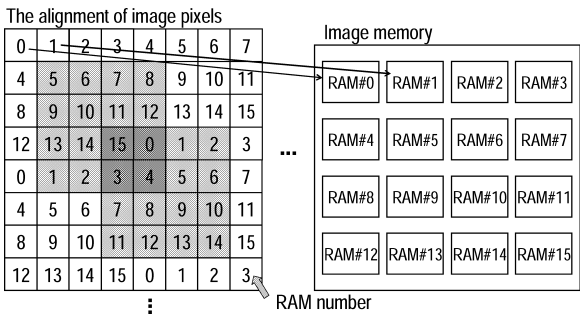


Fig. 19 16 Divisions of image memory.

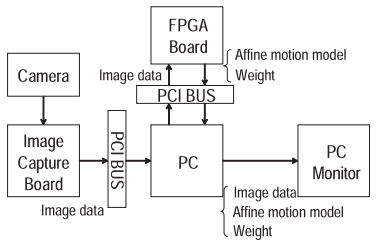
5. VLSI Implementation

5.1 FPGA Implementation

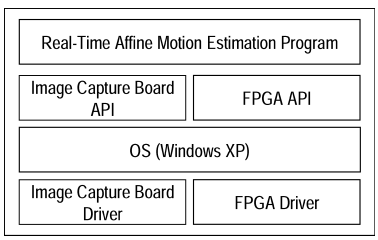
An affine motion estimation processor that can handle an (128×128) 30 fps image sequence based on the proposed architecture was implemented on an FPGA board. The FPGA board (RC2000; Celoxica Inc.) includes an FPGA (Virtex-4 XC4VLX160; Xilinx Inc.). External memory is divided into six banks on the board. Table 6 shows the result of the FPGA implementation. Usages of LUT, 18 kB RAM, and an 18×18 Multiplier are, respectively, 47%, 44%, and 98%. Furthermore, the maximum operation frequency is 36.7 MHz. This frequency is much higher than 6.5 MHz

Table 6 Resource utilization of FPGA.

Resource	Utilization	Precentage
4 input LUT	63,779 out of 135,168	47%
18×18 Multiplier	95 out of 96	98%
Block RAM	128 out of 288	44%



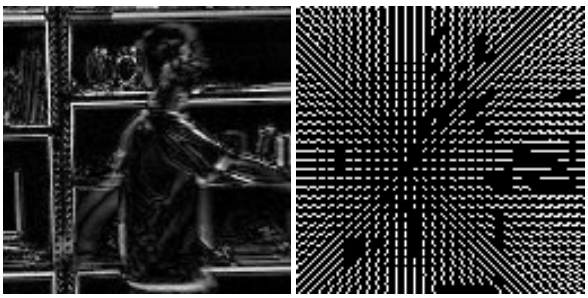
(a) Hardware Construction



(b) Software Construction

Fig. 20 Real-time verification system.

(a) Image #0 (b) Image #1



(c) Subtraction Image (d) Optical Flow

Fig. 21 Experimental results of real-time system.

which is required for the 30 fps motion estimation.

Figure 20 (a) presents a block diagram of a verification system for real-time motion estimation including image capture and display. The camera (SV642M; Silicon Video) and an image capture board (PIXCI SI Digital Frame Grabber) were used. In this system, the images taken by the camera are transferred to the FPGA through the image capture board under control of a PC as shown in Fig. 20 (b). The FPGA estimates the motion model using these image data. The estimated motion model and the weight data are transferred from the FPGA to the PC. Then the PC calculates motion vectors of all pixels from the motion model and the weight data. The motion vectors superimposed on the original image are displayed on the PC monitor in real-time.

5.2 Experimental Results

Figure 21 shows the operation results of the real-time verification system. Figure 21 (a) and (b) depict image examples taken by the system. Those images are successive in time. While the camera zooms in, the human moves to the right. The image background has a dominant motion outspreding from the center. Figure 21 (c) shows a subtraction image between the two images. Both the background and the human are shown in the image. Figure 21 (d) shows the optical flow derived from the affine motion model obtained by the estimation. The background is estimated as a region of a dominant motion. The weight in the human contour is 0, and the motion vectors in the part are not displayed. The silhouette of a human is extracted well. The experimental results show the dominant motion of the dynamic scene can be estimated in real-time. The proposed processor will be applied to real-time video segmentation.

5.3 ASIC Implementation

The PSM processor LSI was designed using 0.18 μm process technology. A main purpose of our ASIC implementation was to show its practicalness, including low cost feature. We completed its layout design using common 0.18 μm technology and estimated the chip area for the purpose. We also estimated the operating frequency to show the real-time performance for VGA 30 fps video.

There is no difference between the FPGA implementation and the ASIC implementation in architecture, excluding the Image B memory. The implementation eliminates the Image B memory for weight calculation shown in Fig. 6 to reduce chip area. We chose the memory because the number of times for weight calculation is fewer than that for motion model calculation. Pixels in the Image B are read from the external. The amount of data traffic is still within the data bandwidth.

Figure 22 shows a LSI layout of the processor. Table 7 shows the LSI specification. The core size is $5.0 \times 5.0\,\mathrm{mm}^2$. The estimated operating frequency is 120 MHz.

Comparison of throughput performance among different implementations is presented in Table 8. The throughput performance of a general purpose CPU with 3.15 GHz frequency is lower than the PSM processor implemented on the FPGA with 30 MHz frequency. The ASIC implementation with 120 MHz frequency can achieve the throughput of VGA 30 fps. Such high performance cannot be achieved by the CPU because it requires about 400 GHz frequency.

Table 8 Comparison of throughput performance.

	Frequency (MHz)	Throughput (pixels/s)	128×128 (fps)	VGA (fps)
CPU(Intel Xeon)	3,150	74,056	4.52	0.24
FPGA(Xilinx XC4VLX160)	30	2,304,000	140.63	7.50
ASIC(0.18 μm SC)	120	9,216,000	562.50	30.00

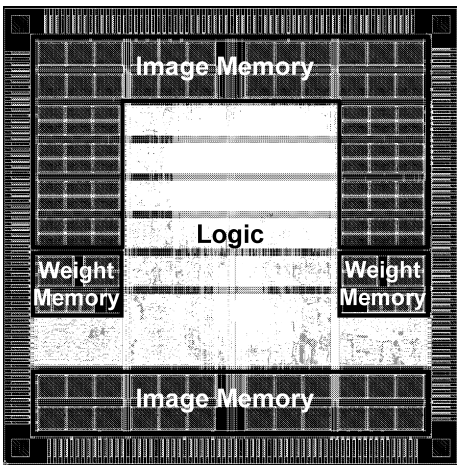


Fig. 22 PSM processor LSI layout.

Table 7 Specification of PSM processor.

Technology	CMOS 0.18 μm 6 metals
Logic Gates	559,263
Memory	Image Level0:8×1 K×48 Image Level1:8×256×48 Weight Level0:2×2 K×8 Weight Level1:2×512×8
Core Size	$5.0 \times 5.0\,\mathrm{mm}^2$
Frequency	120 MHz
Throughput	VGA 30 fps

6. Conclusion

An affine motion estimation processor for real-time video segmentation based on the PSM algorithm was described in this paper. Introduction of the image division method and the binary weight method to the original algorithm reduces the data traffic and the hardware cost. In addition, the proposed pixel sampling method halves the necessary clock frequency. The processor throughput was approximately doubled using pixel pipeline processing and the frame overlap method. The proposed VLSI architecture can handle a VGA 30 fps image sequence with 120 MHz clock frequency, which divides a VGA image into 20 (128×128) images. The processor was implemented on an FPGA and verified in terms of function and performance. The experimental results show that the proposed processor is applicable to real-time video segmentation. The processor was also implemented as an ASIC. The core size is $5.0 \times 5.0\,\mathrm{mm}^2$ in 0.18 μm process, standard cell technology. Application of the processor to real-time video segmentation is a future work.

Acknowledgements

This research was supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research (C), 19560339, 2007-2008. This work was supported by the VLSI Design and Education Center (VDEC) and The University of Tokyo, in collaboration with Celoxica, Ltd. and Synopsys, Inc.

References

[1] B. Siciliano and O. Khatib, Springer Handbook of Robotics, Springer-Verlag, Berlin Heidelberg, 2008.
[2] S.A. Velastin and P. Remagnino, Inteligent Distributed Video Surveillance Systems, The Institution of Electrical Engineers, London, United Kingdom, 2006.
[3] D.A. Forsyth and J. Ponce, Computer Vision A Modern Approach, Pearson Education, USA, 2003.
[4] Al Bovik, Handbook of Image and Video Processing, Elsevier Academic Press, USA and UK, 2005.
[5] Y.-J. Zhang, Advances in Image and Video Segmentation, IRM Press (an imprint of Idea Group Inc.), USA and UK, 2006.
[6] J.R. Bergen, P. Anandan, K. Hanna, and R. Hingorani, "Hierarchical model-based motion estimation," Proc. ECCV-92, pp.237–252, 1992.
[7] J.M. Odobez and P. Bouthemy, "Robust multiresolution estimation of parametric motion models," J. Vis. Commun. Image Represent., vol.6, no.4, pp.348–365, Dec. 1995.
[8] M.J. Black and P. Anandan, "The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields," Computer Vision and Image Understanding, vol.63, no.1, pp.75–104, Jan. 1996.
[9] J.M. Odobez and P. Bouthemy, "Direct incremental model-based image motion segmentation for video analysis," Signal Processing 66, pp.143–155, 1998.
[10] S.M. Smith and J.M. Brady, "ASEET-2: Real-time motion segmentation and shape tracking," IEEE Trans. Pattern Anal. Mach. Intell., vol.17, no.8, pp.814–820, Aug. 1995.
[11] S. Araki, T. Matsuoka, N. Yokoya, and H. Takemura, "Real-time tracking of multiple moving object contours in a moving camera image sequence," IEICE Trans. Inf. & Syst., vol.E83-D, no.7, pp.1583–1591, July 2000.
[12] Y. Yunbe, M. Miyama, and Y. Matsuda, "A VGA 30 fps affine motion estimation processor for real-time video segmentation," IASTED Circuits & Systems, no.625-010, Kailua-Kona, Hawaii, USA, Aug. 2008.
[13] ftp://ftp.csd.uwo.ca/pub/vision, John Barron the University of Western Ontario Department of Computer Science.

Yoshiki Yunbe was born in Toyama, Japan, on January 7, 1985. In 2007, he received the B.S. degree in Department of Electrical and Electronic Engineering from Kanazawa University, where he is currently working toward the M.S. degree in Division of Electrical and Computer Engineering. His research interests include VLSI algorithms and architecture for real-time video segmentation.

Masayuki Miyama was born on March 26, 1966. He received a B.S. degree in Computer Science from the University of Tsukuba in 1988. He joined PFU Ltd. in 1988. He received an M.S. degree in Computer Science from the Japan Advanced Institute of Science and Technology in 1995. He joined Innotech Co. in 1996. He received a Ph.D. degree in electrical engineering and computer science from Kanazawa University in 2004. He is an assistant professor at Kanazawa University Graduate School of Science and Technology. His present research focus is VLSI designs for real-time image processing.

Yoshio Matsuda was born in Ehime, Japan, on October 26, 1954. He received the B.S. degree in physics and the M.S. and Ph.D. degree in applied physics from Osaka University in 1977, 1979, and 1983, respectively. He joined the LSI Laboratory, Mitsubishi Electric Corporation, Itami, Japan, in 1985. He was engaged in development of DRAM, advance CMOS logic, and high frequency devices and circuits of compound semiconductors. Since 2005, he has been a professor of Graduate School of Natural Science and Technology at Kanazawa University, Japan. His research is in the fields of integrated circuits design where his interests have includes multimedia system, low power SoC, and image compression processors.