A Design for the 178-MHz WXGA 30-fps Optical Flow Processor Based on the HOE Algorithm

A Design for the 178-MHz WXGA 30-fps Optical Flow Processor Based on the HOE Algorithm

Tetsuya Matsumura[1], Aoi Kurokawa[2], Kousuke Imamura[2], and Yoshio Matsuda[2]
[1] College of Engineering, Nihon University, Koriyama, Japan
[2] College of Science and Engineering, Kanazawa University, Kanazawa, Japan

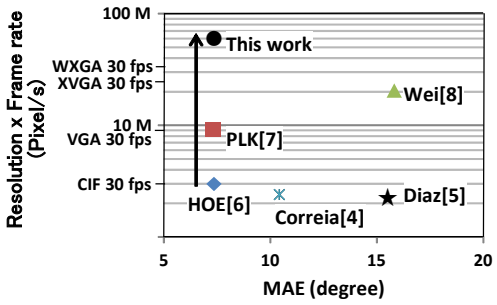
***Abstract*— We propose an optical flow processor, which allows real-time processing of WXGA 30-fps at 178.3 MHz. By introducing the SOR method and a pipeline operation for the Gauss-Seidel method to the iterative flow calculation, computational complexity can be reduced to 14.5% when compared to the previous HOE processor. We decreased the area of the embedded memory by using the image division method, applying line memory, and optimizing the computation word length. The core size of the designed processor is 16.82 mm^2 in 90 nm process technology, which is approximately 5% of the previous HOE processor. The processor can operate completely in parallel, which ensures high-resolution scalability.**

I. INTRODUCTION

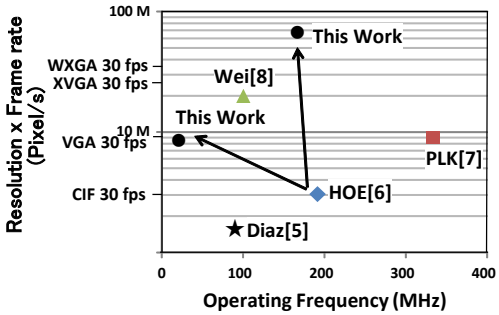
In the field of image processing, highly accurate motion estimation is very important in many applications, such as MPEG2, motion segmentation, and frame rate up-conversion. For example, optical flow, which has been widely studied in the fields of transportation and robotics, is a motion vector at each pixel and requires highly accurate image recognition. For many applications, in addition to accuracy, real-time processing is also needed.

Beginning with Horn-Schunck [1], Lucas-Kanade [2], a number of algorithms have been proposed for computing the optical flow. In optical flow calculations, iterations are generally used in many algorithms, and all algorithms involve a large number of computations. For example, even the CIF 30-fps sequence involves calculations which exceed ten GOPS. Software for the Horn-Schunck algorithm, which was published in the OpenCV Library [3], can only process 3.5 CIF size frames per second, even with a 2GHz CPU. In order to achieve real time processing using software with a general-purpose processor, we must limit the image area and/or thin out the pixels, degrading the accuracy of the optical flow. Therefore, dedicated hardware is essential in real time optical flow calculations to obtain highly accurate optical flows over full size images.

For the dedicated hardware, a number of optical flow processors have been proposed [4]-[8]. Their performances are shown in Fig. 1, as well as the performance of the processor proposed here. In Fig. 1 (a), the vertical axis is the resolution and the horizontal axis is the Mean Angle Error (MAE). In Fig. 1 (b), the vertical axis also shows the resolution while the horizontal axis shows the operating



(a) Pixel rate vs. MAE



(b) Pixel rate vs. operating frequency

Fig. 1 Performance comparison of the optical flow processor.

frequency. The Hierarchical Optical flow Estimation processor (HOE) [6] based on the Horn-Shunck algorithm, and the Pyramidal Lucas-Kanade processor (PLK) [7] based on the Lucas-Kanade algorithm, are highly accurate. However, as shown in Fig. 1 (b), the highest resolution for the HOE processor is CIF 30fps at 189MHz. On the other hand, the PLK can process a VGA with one chip, but it is necessary to operate it at 332MHz. According to Wei et al. [8], the processor can handle 20 M pixels at 100MHz, but there are problems in accuracy. For applications which require even more highly accurate optical flows in high-resolution images, a higher speed is required.

In this paper, we propose a highly accurate optical flow processor with an increased computation speed and a reduced size, based on the HOE algorithm. Section II gives a brief review of the HOE algorithm. Section III describes the most important methods introduced in order to reduce the computational complexity in optical flow calculation. It also includes the pipelining of the Gauss-Seidel method, which

contributes greatly to the realization of a high-throughput [9]. Section IV describes the image division method, which is the central method for reducing hardware volume. Section V discusses an optical flow processor design which incorporates the above-mentioned technology. Finally, the conclusions are given in Section VI.

II. HOE ALGORITHM

The HOE [6] algorithm, which is based on the Horn-Schunck algorithm, gives a highly accurate optical flow using a motion compensation frame. The luminance value of the pixels (x, y) in the time t frame is $I(x, y, t)$, and each pixel moves u and v respectively, in the direction x and y in the next frame at the time $t + 1$. This motion vector (u, v) is called the optical flow. Hereafter, it is simply referred to as the flow. Following the luminance-conservation law, a relation (1) is established between the luminance values of frame t and frame $t + 1$.

$$I(x + u, y + v, t + 1) - I(x, y, t) = -\xi. \quad (1)$$

In this case, ξ represents a luminance change unrelated to pixel motion, such as sunlight changes. If the flow (u, v) is small in (1), (2) is obtained in the first order approximation of the Taylor expansion.

$$I_x u + I_y v + I_t + \xi = 0. \quad (2)$$

I_x and $I_y$represent the spatial luminance gradient around x and y, respectively, and I_t is the temporal luminance gradient. Since (u, v) cannot be determined by only using (2), the conditions of spatial smoothness change are imposed on $u, v,$ and ξ. The flow (u, v) and luminance change ξ are chosen to minimalize the following error function (3).

$$E(u, v) = \iint (f^2 + \alpha^2 f_a^2 + \beta^2 f_b^2)^2 dxdy,$$
$$f^2 = (I_x u + I_x v + I_t + \xi)^2,$$
$$f_a^2 = u_x^2 + u_y^2 + v_x^2 + v_y^2,$$
$$f_b^2 = \xi_x^2 + \xi_y^2. \quad (3)$$

$u_x, v_x,$ and ξ_x are the spatial gradients of $u, v,$ and ξ around x, respectively, and $u_y, v_y,$ and ξ_y around y. f_a^2 and f_b^2 represent the imposed conditions for spatial smoothness for $(u, v)$and ξ respectively, and weights $\alpha$and β are adjustable parameters. The integral is taken over the entire frame. The minimum values of $(u, v)$and ξsatisfy the following equation.

$$(I_x u + I_y v + I_t + \xi) I_x = \alpha^2 \Delta u,$$
$$(I_x u + I_y v + I_t + \xi) I_y = \alpha^2 \Delta v,$$
$$(I_x u + I_y v + I_t + \xi) = \beta^2 \Delta \xi, \quad (4)$$

where Laplacian $\Delta u_{i,j}$ for the pixel (i, j) is given by (5).

$$\Delta u_{i,j} = (u_{i-1,j-1} + u_{i-1,j+1} + u_{i+1,j-1} + u_{i+1,j+1})/12 + (u_{i,j-1} + u_{i,j+1} + u_{i-1,j} + u_{i+1,j})/6 - u_{i,j} = \bar{u} - u_{i,j}. \quad (5)$$

The Laplacian of v, ξ are defined using the same equation as (5). $\bar{u}$ denotes the local average of the 8 neighboring pixels around the pixel (i, j). Finally, we obtain the simultaneous liner equation (6) from (4) and (5).

$$u = \bar{u} - I_x \frac{I_x \bar{u} + I_y \bar{v} + I_t + \bar{\xi}}{\alpha^2 + {I_x}^2 + {I_y}^2 + \lambda^2},$$
$$v = \bar{v} - I_y \frac{I_x \bar{u} + I_y \bar{v} + I_t + \bar{\xi}}{\alpha^2 + {I_x}^2 + {I_y}^2 + \lambda^2},$$
$$\xi = \bar{\xi} - \lambda^2 \frac{I_x \bar{u} + I_y \bar{v} + I_t + \bar{\xi}}{\alpha^2 + {I_x}^2 + {I_y}^2 + \lambda^2}. \quad (6)$$

In (6), the subscripts i and j of the pixel coordinates are omitted and $\lambda = \alpha/\beta$. By solving (6) using the iterative method, the flow and the luminance change are obtained.

Since we assume the flow (u, v) to be small (a linear approximation) in (2), the accuracy of the flows might degrade if there is a large movement in the images. As a countermeasure, a hierarchical motion estimation and a motion compensation frame are employed in order to improve accuracy. Also, if the spatial and temporal gradients are approximated with a two point difference, highly accurate flows cannot be obtained because that they are too sensitive to noise and luminance variation in the image. The HOE solves this problem using a multi-dimensional gradient filter.

A flow calculation using the HOE algorithm is shown in Fig. 2 for an images with two layers. First, three hierarchical images are created from each of the three images in frames $t - 1, t,$ and $t + 1$. During this process, in order to smooth out the images, a 5 x 5 Gaussian filter is simultaneously applied to the pixels when sub-sampled.

After the highest layer images have been created, the spatial and temporal luminance gradients are calculated using a multi-dimensional gradient filter, and the flow(u, v) and luminance change ξ are calculated by solving (6) with the iterative method. When the updated value of the flow in

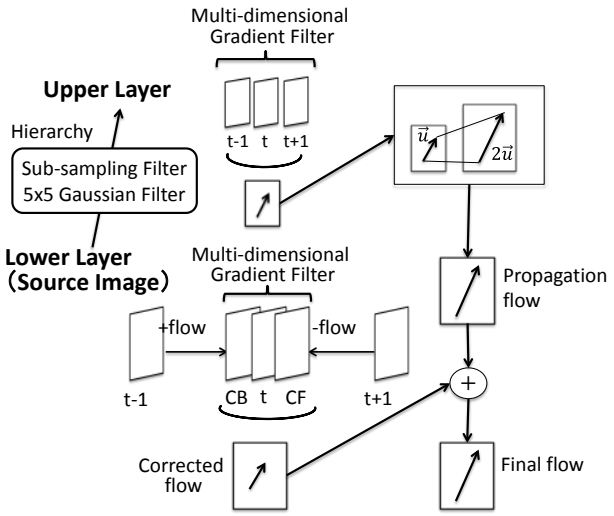


Fig. 2. Illustration of the optical flow with a hierarchical motion compensation frame in HOE.

the iteration becomes smaller than the preset threshold, or when the number of iterations reaches the preset number, the calculation ends. In [6], the updated threshold is set to 0.0001 and the number of iterations to 150 times.

The flow obtained from this hierarchy is doubled and then propagated to the lower layer. Hereafter, this is referred to as the propagation flow. After the motion compensation frames C_F and C_B are created from frames $t-1$ and $t+1$, respectively, using the propagation flow, the multi-dimensional gradient filter is again applied to the C_F, C_B and t frame of the lower layer in order to calculate the flow correction in the lower layers. As shown in Fig. 2, the motion compensation frame C_F is created from frame $t-1$, assuming that each pixel of the $t-1$ frame moves by the rate of propagation flow. The C_B is created from the frame $t+1$. Since the integer pixels of the motion compensation frame generally correspond to the subpixels in frames $t-1$ and $t+1$, the subpixels are generated by bilinear interpolation from the surrounding integer pixels. The flow in the lower layers is obtained by adding the corrected flow obtained in a lower layer to the propagation flow from the upper layers. The final flow is obtained by repeating this process until the lowest layer is reached.

III. Reducing Computational Complexity and Improving Throughput

A. Initial flow generation using the previous frame flow

In the HOE algorithm, the number of the iterations is set 150 times, which accounts for 90% of the total computational complexity. For real-time processing, it is most effective to reduce the number of iterations. Considering that the movement between two consecutive frames is small, it can be expected that the convergence speed would increase if the flow obtained from the previous frame is used as the initial value of the iteration.

Although it seems reasonable to use the flow obtained from the previous frame layer as the initial value, memory is required to store the flow of each layer and that leads to an increase in H/W. In contrast, the last and lowest layer's flow of the previous frame is stored in an external memory until the last flow of the current frame is obtained, so the last flow can be used as the initial value of the iteration. This flow is read into the chip with sub-sampling and the initial value of the upper layer is generated by reducing it to 1/2. 1/2 can be easily performed in a one bit shift operation, which also leads to a reduction in H/W.

B. SOR method and Pipelining of the Gauss-Seidel method

The Jacobi iteration method was used in previous HOE processor. For this new processor design, the Gauss-Seidel method is employed, which provides faster convergence on the iterative calculation. Since this creates a problem with pipeline operation, the diagonal average (the average over 4 diagonal pixels) is introduced to the Laplacian in (5).

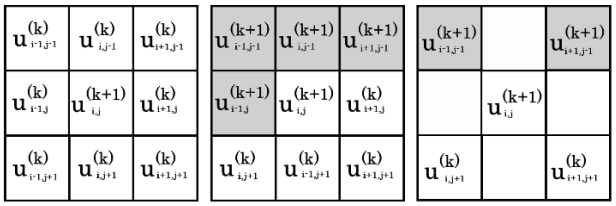


Fig. 3. Laplacian by 4 diagonal pixels.

The flow of each pixel is calculated by raster scanning from left to right and from top to bottom. The flow of the 8 neighboring pixels is required for the local average $\bar{u}$ when calculating $u_{i,j}^{(k+1)}$ at the (k+1)st iteration. When calculating the (k+1)st iteration (Fig. 3(a)) in the Jacobi method, the k-th flow is used for the 8 neighboring pixels, whereas in the Gauss-Seidel method, (k+1)st flow is used for the upper 3 pixels and the left-hand 1 pixel (Fig. 3(b)) when calculating $u_{i,j}^{(k+1)}$ at the (k+1)st iteration. Therefore, $u_{i,j}^{(k+1)}$ cannot be calculated until the flow of the left pixel is updated. This means that the Gauss-Seidel method is not compatible with the pipeline operation. Therefore, the average of 4 diagonal pixel for $\overline{u_{i,j}}$ is introduced (Fig. 3(c)). In other words, the Laplacian is defined by (7).

$$\Delta u_{i,j} = 2\left(\frac{u_{i-1,j-1}-u_{i-1,j+1}+u_{i+1,j-1}-u_{i+1,j+1}}{4} - u_{i,j}\right). \quad (7)$$

This is the same for v and ξ. Since the coefficient 2 is included in α, it does not appear explicitly. Finally, $\bar{u}$ in (5) can simply be replaced by the average over the 4 diagonal pixels.

In order to further accelerate the convergence, the Successive Over-Relaxation (SOR) method is introduced. The SOR method accelerates the convergence by multiplying the accelerating parameter ω with the difference between the value at the (k+1)st iteration and the value at the kth iteration in order to increase the corrected quantity. Convergence is guaranteed for $0 < \omega < 2$, but in general, there is no method to determine the value of ω. In this processor design, $\omega = 1.75$ is chosen by simulation. This value makes the hardware implementation very easy because multiplication can only be performed in the shift operation. Finally, our calculations of the flow, when using the 4 diagonal pixels as the local average and applying the Gauss-Seidel and SOR methods, are given in (8).

$$u^{(k+1)} = u^{(k)} + \omega\left(\bar{u}^{(k)} - I_x\frac{I_x\bar{u}^{(k)} + I_x\bar{v}^{(k)} + I_t + \bar{\xi}^{(k)}}{\alpha^2 + {I_x}^2 + {I_y}^2 + \lambda^2} - u^{(k)}\right),$$
$$v^{(k+1)} = v^{(k)} + \omega\left(\bar{v}^{(k)} - I_y\frac{I_x\bar{u}^{(k)} + I_x\bar{v}^{(k)} + I_t + \bar{\xi}^{(k)}}{\alpha^2 + {I_x}^2 + {I_y}^2 + \lambda^2} - v^{(k)}\right),$$
$$\xi^{(k+1)} = \xi^{(k)} + \omega\left(\bar{\xi}^{(k)} - \lambda^2\frac{I_x\bar{u}^{(k)} + I_x\bar{v}^{(k)} + I_t + \bar{\xi}^{(k)}}{\alpha^2 + {I_x}^2 + {I_y}^2 + \lambda^2} - \xi^{(k)}\right),$$

$$\bar{u}_{i,j}^{(k+1)} = \frac{1}{4}\left(u_{i-1,j-1}^{(k+1)} + u_{i+1,j-1}^{(k+1)} + u_{i-1,j+1}^{(k)} + u_{i+1,j+1}^{(k)}\right),$$
$$\bar{v}_{i,j}^{(k+1)} = \frac{1}{4}\left(v_{i-1,j-1}^{(k+1)} + v_{i+1,j-1}^{(k+1)} + v_{i-1,j+1}^{(k)} + v_{i+1,j+1}^{(k)}\right),$$
$$\bar{\xi}_{i,j}^{(k+1)} = \frac{1}{4}\left(\xi_{i-1,j-1}^{(k+1)} + \xi_{i+1,j-1}^{(k+1)} + \xi_{i-1,j+1}^{(k)} + \xi_{i+1,j+1}^{(k)}\right). \quad (8)$$

With these techniques, the computational complexity associated with the iterations was reduced to 5% when compared to the previous HOE processor [6] while retaining the same accuracy. Other computations, such as generation of hierarchical images and motion compensation frames, account for 10% of the total. Thus, the computational complexity was reduced to about 14.5% while maintaining the same level of accuracy.

IV. Reduction of the Embedded Memory Using the Image Division Method

In algorithms with iteration, storing the temporary data of each iteration step to the external memory would occupy the bus band width due to data transfer, so the temporary data are stored to embedded memory on the same chip and the data processing is completed within the chip. In the HOE, a great amount of memory is embedded, such as the source image memory, which stores the luminance values of the source image, the hierarchical image memory, which stores the luminance values of the hierarchy image, and the luminance gradient and motion memories, which store the luminance gradient and the flow of each iteration step, respectively. Also, as motion compensation frames are used, memory to store the image data of these compensation frames is also required. In the HOE, the flow is calculated frame by frame, so a large capacity of RAM are embedded, leading to an increased chip size.

In our optical flow processor design, the frame is divided into blocks, and the flow is calculated block-by-block. Hereafter, this method is called the image division method. The capacity of the embedded memory is reduced to 1/*N* compared to the previous HOE processor [6], where *N* is the number of blocks. If the frame is simply divided into blocks, great flow disturbance occurs in the block boundary, which degrades the accuracy of the flow. Therefore, as shown in Fig. 4 (a), overlapping pixels are added to each block (white dashed line) and the flow is calculated over the block with overlapping pixels. *o* shows the overlapping pixels. The final flow of the entire frame is generated by cutting the flow of desired block (white solid line). Simulation over multiple sequences confirm that 15 overlapping pixels are sufficient, regardless of the block size.

Figs. 4(b) and 4(c) show the simulation results of the Translation Tree flow for 150 pixels x 150 pixels, when divided into 9 blocks. The block size is 50 pixels by 50 pixels. Fig. 4(b) shows the Mean Angle Error (MAE) of the flow with non-overlapping pixels and Fig. 4(c) shows the

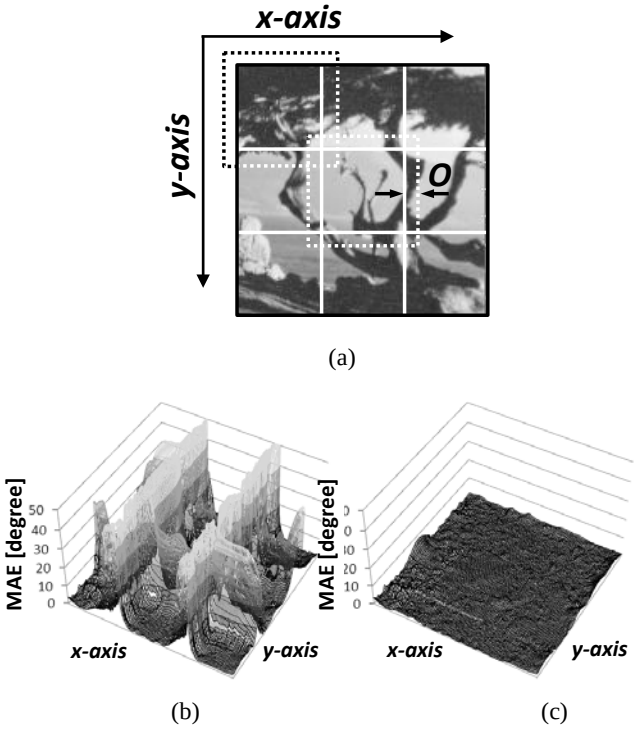


Fig. 4. Optical flows using the image division method.

image division method for 9 divisions with 16 overlapping pixels. Flow disturbance on the boundary is eliminated in the image division method. Furthermore, exactly the same flow is obtained numerically when compared to a non-divided image.

If the block size becomes smaller by increasing the number of divisions, the embedded memory can also be smaller. The proposed processor contains a pipeline operation for the divided blocks. In this case, the greater the number of divisions, that is, the smaller the number of pixels in the block, the better the throughput is. On the other hand, a large number of divisions would lead to an increase in the overlap pixel ratio and more computations. Therefore, there are a certain number of divisions that would maximize the throughput when the number of overlapping pixels is fixed. In the case of 15 overlapping pixels, a 6-9 division maximizes the throughput. Based on the above mentioned considerations, the proposed processor manages VGA (640 x 480) size images in 9 divisions. Furthermore, when processing XGA (1024 x 768), WXGA (1280 x 800), and HD (1920 x 1080) size images, a block size of 244 pixels x 192 pixels, which includes 15 overlapping pixels in the *x* direction and 16 pixels in the *y* direction, is used for the pipeline operation processing unit. In addition, by replacing some kinds of embedded memory with line memory and optimizing the computation word length, the embedded memory capacity can be reduced to about 8% compared to the previous HOE processor when converting the flow calculation of a WXGA size image.

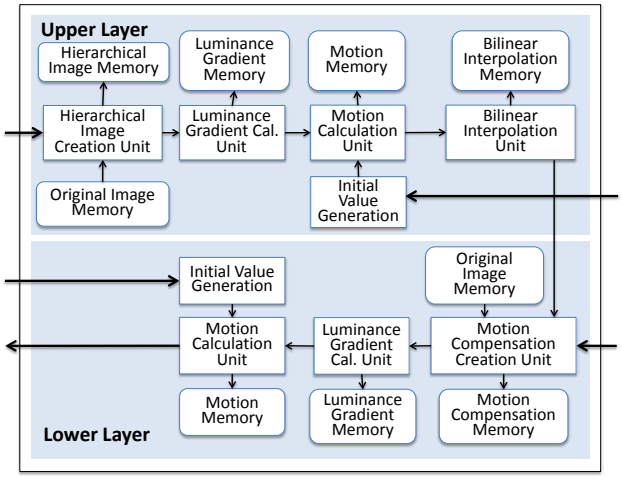
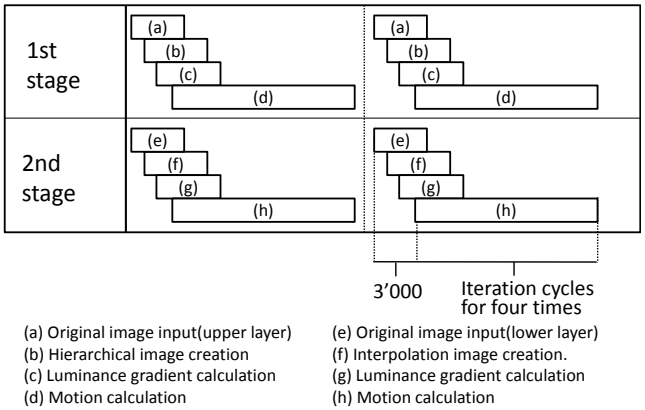


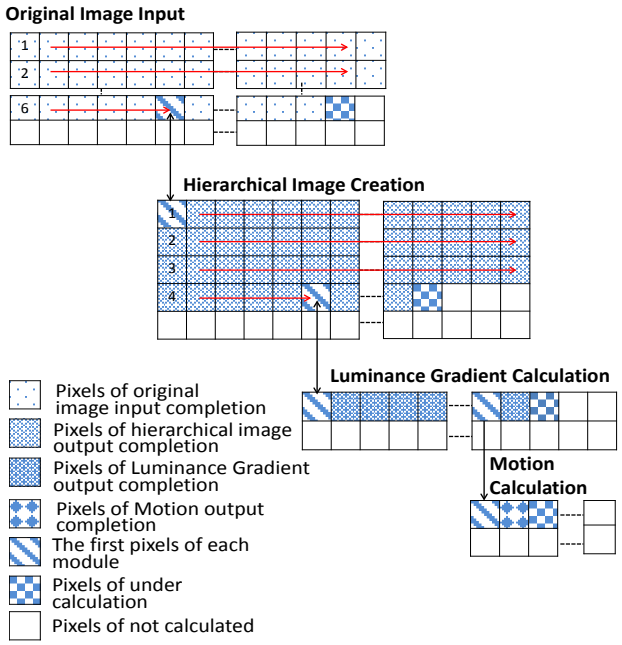
Fig. 5. Block diagram of the designed optical flow processor.

Fig. 5 shows a block diagram of the proposed processor. It consists of an upper layer module and a lower layer module. The upper layer is composed of four units, a hierarchical image creation unit, a luminance gradient calculation unit, a motion calculation unit, and a bilinear interpolation unit. The lower layer is composed of three units, a motion compensation image creation unit, a luminance gradient calculation unit, and a motion calculation unit. Each unit has computing/control units and associated memory. The source image memory and hierarchical image memory of the upper layer, and the source image memory and motion compensation image memory of the lower layer are placed in a chip as a line buffer. The motion memory and luminance gradient memory of the lower layer store the data of a 244 x 192 pixel sized block. The motion memory, luminance gradient memory, and bilinear interpolation memory of the upper layer store 61 x 48 pixel data, which is 1/4 of the lower layer. The input image memory for storing the source image data of the frame and the output motion memory for storing the final flow are external memory.

The upper and lower layer modules operate in parallel and carry out pipeline operations block-by-block in the unit of divided images. In addition, each unit carries out pipeline operations pixel-by-pixel. Fig. 6 shows the pipeline operation for a case of 2 layers. As shown in Fig. 6(a), the flow calculation of the upper layer and lower layers have a 2-stage pipeline structure in blocks. Fig. 6(b) shows the pipeline operation in pixels within each of the blocks. 5 lines of the source image are stored in the line buffer. When the memory begins to store the 6th line, the data of the 5 pixels are inputted to the hierarchical image creation unit at the same time, and a hierarchy image is created while applying the Gaussian filter. After the 25 pixels (5 x 5) are inputted, the first pixel of the hierarchical image is outputted in the next cycle. In the following cycles, one pixel of the hierarchical image data is outputted per cycle. Then, 3 lines



(a) Pipeline operation in the case of 2 layers



(b) Pipeline operation per pixels

Fig. 6. Pipeline operation of the designed processor.

of the hierarchical image are stored in the line buffer. When the storage of the 4th line begins, the luminance gradient calculation starts and the first pixel of the luminance gradient is outputted in the 6th cycle. Next, the luminance gradient is calculated by one pixel per cycle. The motion calculation immediately starts after the luminance gradient is outputted. The motion calculation results are outputted in 12 cycles after the luminance gradient is inputted, and then it is outputted by 1 pixel per cycle. When the motion calculation is finished, bilinear interpolation is performed and the flow with doubled is propagated to the lower layer.

8 lines of the source image are stored in the line buffer of the lower layer and the creation of interpolated images starts from the 9th line. Then, the same process as the upper layer is performed. The final flow is outputted to the external

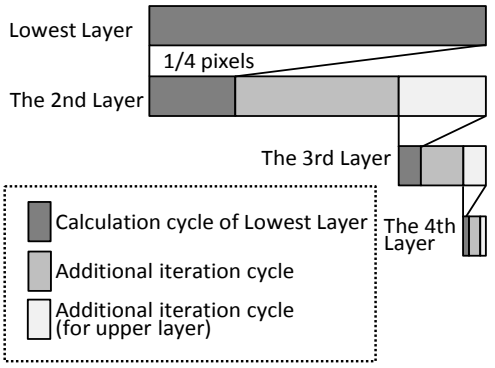


Fig. 7. Allocation of the number of iterations for multiple layers.

TABLE I
CHARACTERISTICS OF THE DESIGNED PROCESSOR

Item		Proposed processor	HOE processor [6]
Number of iterations (in the case of two layers)		Upper: 24 Lower: 6	All: 150
Performance		WXGA 30 fps	WXGA 30 fps
Number of divisions		30	-
Resolution		244 × 192	1280 × 800
Operating frequency		178.3 MHz	1909 MHz
Data transfer rate		0.58 Gbyte/s	1.21 Gbyte/s
Number of logic gates		230 kgates	311 kgates
Memory	1 port	987 kbytes	6212 kbytes
	2 port	-	6162 kbytes
Memory area		16.08 mm^2	308.9 mm^2
Core area		16.82 mm^2	310.4 mm^2

memory after adding the calculated motion and the propagation flow.

The proposed processor allows the creation of any number of layers in the same hardware. The concept is shown in Fig. 7, where the number of iterations in the upper layer and the iteration cycles of the lower layer are set to be equal. The number of pixels in the upper layer is 1/4 the number of pixels in the lower layer, so the operation cycle is also 1/4. In order to equalize the number of operation cycles in the lower layer, the remaining 3/4 is allocated to additional iterations. In the case of 3 layers, 2/4 is allocated to additional iterations for the 2nd layer, and the remaining 1/4 is allocated to additional iterations for the 3rd layer. In this way, the proposed processor handles any multiple layers without additional cycles.

This processor calculates the flow of VGA size images in 9 divided blocks of 244 pixels x 192 pixels, including overlapping pixels. The flow of VGA 30 fps can be calculated in real time at 57.5 MHz. The processing of WXGA size images is supported by a faster operating frequency. At 178.3 MHz, real-time processing of 30 fps is possible. In fact, the processor is designed to operate at this frequency. In addition, the processor can operate completely in parallel thanks to the image division method. Real-time processing of HD 30 fps is possible using only two processors in simple parallel operations at the same frequency.

TABLE II
ACCURACY COMPARISON OF THE OPTICAL FLOW

	MAE[degree]			
	Trans.	Div.	Yos.	Average
Proposed processor	0.658	2.714	5.006	2.614
HOE processor [6]	0.626	2.701	6.889	3.405

The characteristics of the proposed processor and HOE [6] processor are shown in Table I. The previous HOE processor is designed to process CIF 30 fps at 189 MHz, but in order to be compared with our processor, the characteristics are translated as "to calculate flow of WXGA 30 fps." Furthermore, the chip size of this new processor is estimated in 90 nm process technology based on the gates-to-area value of the previous HOE processor. By introducing the image division method, line buffer, and optimization of the computation word length, the capacity of the embedded memory is reduced to 8% (5% in area) and the data transfer volume by about half. Also, the previous HOE processors embeds a specially designed 2-port DRAM for a 1 chip solution, but our processor uses simple standard SRAM.

Finally, the evaluated flows are listed in Table II for three test sequences with correct flow (Translation Tree, Diverging Tree, and Yosemite). Accuracy is equal to or better than the previous HOE processor.

V. CONCLUSION

We designed an optical flow processor of WXGA 30 fps at 178.3 MHz, which allows real-time processing. This processor can handle approximately 10 times the resolution at same frequency as the previous HOE processor. The core size is 16.82 mm^2 in 90 nm process technology, which is only about 5% of the previous HOE processor. The designed processor can operate completely in parallel, which ensures high-resolution scalability.

REFERENCES

[1] B.K.P. Horn and B.G Schunck, "Determining optical flow," *Artificial Intelligence*, vo. 17, pp. 185-203, 1981.
[2] B.D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proc. Imaging Understanding Workshop*, pp. 121-130, 1991.
[3] http://opencv.jp/sample/optical flow.html
[4] M.V. Correia and A.C. Campilho, "Real-time implementation of an optical flow algorithm," in *16th Int'l. Conf. Pattern Recognition (ICPR'02)*, vol. 4, pp. 247- 250, Aug. 2002.
[5] J. Diaz, E. Ros, S. Mota, F. Pelayo, and E.M. Ortigosa, "Real-time optical flow computation using FPGAs," in *Proc. Early Cognitive Vision Workshop*, 2004.
[6] N. Minegishi, et. al, "VLSI architecture study of a real-time scalable optical flow processor for video segmentation," *IEICE Trans. Electron*., vol. E89-C, no. 3, pp. 230-242, Mar. 2006.
[7] Y. Murachi, et. al, "A VGA 30-fps realtime optical-flow processor core for moving picture recognition," *IEICE Trans. Electron*., vol. E91-C no. 4, pp.4 57-464, Apr. 2008.
[8] Z. Wei, D.J. Lee, and B.E. Nelson, "FPGA-based real-time optical flow algorithm design and implementation," *Jour. Multimedia*, vol. 2, no. 5, pp. 38-45, Sept. 2007.