COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING 43, 178–204 (1988)

A VLSI Architecture for Computing Scale Space*

N. RANGANATHAN AND MUBARAK SHAH

Department of Computer Science, University of Central Florida, Orlando, Florida 32816

Received May 7, 1987; revised February 24, 1988

Meaningful information about a scene is captured in the intensity changes in an image. These intensity changes occur at various scales depending on their physical origins. Scale-space generated by applying the Laplacian of Gaussian edge detector to the image at a continuum of scales is a powerful representation for detecting and organizing these intensity changes symbolically and has proved to be very useful for one-dimensional signals. The high computational cost of generating scale-space in two dimensions has restricted its use in images. This paper proposes a very efficient single chip VLSI design for scale-space computation in one and two dimensions. The architecture of the chip is based on an algorithm that can provide speeds that are of an order of magnitude higher than the speeds obtainable from other systems proposed in the literature. The design uses the principles of *modularity*, *expandability*, and *parallelism*, and fully utilizes the three properties of Gaussian *symmetry*, *separability*, and *scaling*. Moreover, our proposed algorithm does not approximate the Laplacian of the Gaussian operator; it uses instead four one-dimensional convolutions to obtain the computations in two dimensions. The proposed architecture has not been built. © 1988 Academic Press, Inc.

1. INTRODUCTION

1.1. *Motivation*

The goal of computer vision is to build algorithms for recovering explicit and meaningful information about a scene from the images. This information is useful to recognize instances of real world objects. Information about the scene is captured in the intensity changes across time and space. The intensity changes in space signify the discontinuities due to boundaries of objects, depth, and surface orientation. The changes across the time axis, on the other hand, carry the information about the motion of objects and cues to occlusion. These intensity changes occur at various scales depending on the sizes and motion characteristics of objects. Sometimes, it is possible to find an instance of change at a particular scale, when it can not be found at other scales.

Marr [17] argued that attributes carrying the valuable information about the physical process—e.g., a regular patchwork of wheat, a stalk of wheat, and the grains on the stalk of wheat—may emerge at any range of scales in the real world

*The research reported in this paper was supported in part by Center for Research in Electro Optics and Lasers (CREOL), University of Central Florida under Grant 20-52-043 and by the National Science Foundation under Grant IRI 87-13120.

and more so in images because of additional transformation introduced during the imaging process. In order to capture the useful information in an image, Marr and Hildreth [14] proposed the use of the Laplacian of the Gaussian (LG) edge detector.[1] In their scheme, they convolve an image with a number of LG operators of different scales and detect zerocrossings in the result. They found that spatial coincidence of zero crossings at all scales may signal the presence of a physical edge.

Marr and Hildreth [14] used the LG operator at four different scales with Gaussians having standard deviations $\sigma = 1, 2, 4, 8$. This was motivated neurophysiologically because of the evidence of four bandpass channels found in the retina. Recently, Witkin [28] proposed the scale-space approach in which he used a continuum of scales. The scale-space representation is obtained by plotting the locations of zero crossings along the x-axis as a function of scale parameter σ in the x-σ space. Witkin found that these zero crossings form contours like arches, and as one sweeps across the apex of an arch with increasing σ, the zero crossings disappear in pairs. By proposing the notion of scale-space, Witkin not only reaffirmed the importance and strength of scale-space representations, but he also intrigued many researchers, resulting in several papers: [32, 30, 26, 31, 1, 5, 15, 16, 6, 25, 23, 24, 22, 20].

1.2. *Purpose*

Several researchers have proposed hardware implementations of the LG operator, but none fully exploited the advanced VLSI technology and the properties of the Gaussian, namely scaling, separability, and symmetry. Therefore, no suitable architecture *for the LG operator exists which can be extended for scale-space.* The previous architectures have used off-the-shelf LSI components like multiplier chips which forced them to implement the system with several machine wire-wrapped boards [19]. In [11], a hardware system for the Gaussian convolution has been proposed. Each processor in the systolic array is implemented as a basic convolver board and the circuitry for each board is complex. Also, the need for a huge PLA to produce the complex control logic makes their architecture inefficient in space and time. In [2], the zero crossing detector is implemented using a PLA and the critical delay for the PLA determined the clock for the whole chip. Another approach proposed in [10] uses a processor per pixel, but it will work only when the mask size is 3 by 3 and the operator coefficients are powers of two. With the advanced VLSI technology [18] and experience available today, it is possible to integrate more hardware into a single chip and obtain faster and more cost efficient systems.

Previously, the researchers have limited the use of the powerful scale-space representation technique to only one dimension. The high computational cost of scale space in two dimensions has restricted its use in two-dimensional images. In fact, the first implemented version of the LG operator took about three hours to compute the zero crossings in the coarse channel of an image 512 pixels square, the smallest operator being roughly 35 picture elements [4]. The computation of

[1] Laplacian of Gaussian is defined as $\nabla^2 g(x, y) = (\partial^2/\partial x^2)g(x, y) + (\partial^2/\partial y^2)g(x, y)$, where $g = e^{-(x^2+y^2)/2\sigma^2}$ is a bivariate Gaussian filter. The standard deviation σ of the Gaussian is related to the scale parameter of the filter. The application of LG edge detector is equivalent to the convolution of an image with $\nabla^2 g$, i.e., $I * \nabla^2 g$.

scale-space requires even more time, because it involves the applications of LG at many scales.

The *separability*, *symmetry*, and *scaling* properties of Gaussian can be exploited in order to obtain an efficient implementation of scale-space in hardware. The purpose of this paper is to propose a special-purpose architecture that will use these properties of the Gaussian and the power of VLSI to the maximum extent in order to achieve high throughput as well as speed for the scale-space.

1.3. *Outline of Paper*

This paper proposes an efficient single chip VLSI design for scale-space computation in both one and two dimensions. The architecture of the chip is based on an algorithm that can provide speeds that are an order of magnitude higher than the speeds obtainable from other systems proposed in the literature [19]. The design uses the principles of *modularity*, *expandability*, and *parallelism*, and fully utilizes the three properties of Gaussian *symmetry*, *separability*, and *scaling*. In our approach, the entire convolver is implemented as a set of processing elements (PEs) working in parallel, with each PE organized as a pipeline. The circuit for the PE is simple with minimal control logic. The host broadcasts a new pixel value during each clock cycle, and after the pipe in the first PE is filled, the host receives back a result during each cycle. Each PE is a pipeline of several stages consisting of registers, adders, and a *Wallace multiplier*. The *Wallace multiplier* [27] itself consists of isolated adder stages. Thus, the clock for the chip is dependent entirely on the delay of an adder and we can obtain a fast implementation of our scheme. Our architecture is flexible and can be adapted to implement systems that use different mask sizes.

The organization of the rest of the paper is as follows. In the next section, we review the related work on the hardware implementation of the LG operator and identify the short comings of the proposed architectures. We will outline our approach in Section three. The hardware design for one-dimensional scale-space is presented in Section four, while the extension of this design to two dimensions is reported in Section five. Finally, the chip implementation issues and performance estimates are explored in Section six.

2. RELATED WORK

Nishihara and Larson [19] built prototype hardware for implementing the LG operator which is based on the difference of Gaussian approximation. Their approach allows masks up to 32 by 32 in size. They have used *separability* to reduce the two-dimensional convolution into two one-dimensional convolutions. However, it appears that the *symmetry* and *scaling* properties were not used. They have used off-the-shelf components like the TRW multiplier chips. Also they mention the use of a hardware zero-crossing detector module, but do not give any design details. Their project goal was to gain the experience of building hardware for a vision system. They built a hardware system using machine wire-wrapped boards for the stereo matcher problem. It is possible to build more efficient and compact hardware systems by taking advantage of the VLSI technology.

An important point to be noted here is that Nishihara and Larson based their implementation on the difference of Gaussian approximation of the LG operator. Marr and Hildreth reported that the Laplacian of the Gaussian could be approximated by the difference of two Gaussians with standard deviation ratio of 1.6, but this was never formally justified [14]. Due to this, many researchers have used different ratios of standard deviations for this approximation. For instance, Crowley and Parker [7] used a ratio of 1.4 and Fleet *et al.* [9] used ratios of 1 and 2.5. Recently, it has been reported in the literature that the LG operator can also be approximated by a difference of offset Gaussians D.O.O.G. [29]. In this paper, however, we will not deal with an approximation. We will show that the LG can be computed without approximations using four one-dimensional convolutions. Our method has the same computational complexity as the ones that use approximations.

Batali [2] describes the implementation of a chip that computes the approximation to the two-dimensional gradient and detects zero crossings. The input to this chip is a raster-scanned digital stream of a two-dimensional video image that has already been convolved with the Laplacian of the Gaussian operator. The problem of gradient computation is beyond the scope of our paper. The chip uses a PLA implemented as a finite-state machine for detecting zero crossings. The author himself points out that the critical path in the AND plane of the PLA affects the clock for the whole chip. The design which we propose uses precharge logic and has a critical delay equal to that of an exclusive-OR stage, thus it is faster than that of Batali.

A VLSI-based systolic architecture for Gaussian convolution has been proposed by Giordano *et al.* in [11]. Their architecture uses a Booth's multiplier [3], ripple carry adder, and a microprogrammed ROM with complex control logic for the basic convolver board. The clock is dependent on the multiplier unit which was predicted as 125 ns. However, each basic convolver operation, a multiplication plus an addition, requires several cycles before the product is broadcast on the output bus. The choice of Booth's multiplier in such a situation where a large number of multiplications have to be performed is not good, since it may take anywhere from 16 to 24 cycles to perform one 8-bit multiplication. Moreover, the authors have not taken advantage of the symmetry and scaling properties of the Gaussian, which we will show simplifies the overall hardware significantly.

Georgiou and Anastassiou [10] have proposed a single chip architecture for the Laplacian operator which is suitable only for masks of size 3 by 3 and cannot be used for larger masks. In a 3 by 3 convolution, there is communication only to the eight nearest neighbors, whereas for larger masks, communication with PEs farther away is required. A PE organization with such capability would be complex and therefore, it is not economical to assign one PE per pixel. Furthermore, the paper assumes that the weights are powers of two in order to do multiplication by shift left. However, this assumption does not hold for all the operators, for example, the LG operator. There are two important points to be noted here. First, in general, masks of much larger size, as in the Gaussian filter, are desirable. Second, the weights in most operators are not powers of two. The first factor raises the need for VLSI architectures that are general enough for application to larger masks. The second factor dictates the use of a hardware multiplier within each PE. If each PE needs to have a hardware multiplier, then the concept of a PE per pixel is ruled out since it is extremely expensive in terms of hardware. Thus, we need an architecture that will exploit the power of VLSI and at the same time will be suitable for convolution of any size window.

3. APPROACH

Our design is based on the principles of *modularity*, *expandability*, and *parallelism*. The system architecture should be modular in nature, so that each module of the system can be treated separately from the rest in terms of its input, output and the function. Our system has three main modules, the convolver, the normalizer, and the zero-crossing detector. The convolver, in turn, is a set of processing elements (PEs), where each PE has four submodules: FIFO buffers, two adders, and a multiplier. With a modular system it is easier to obtain different functions by doing simple changes to the system. The design we propose in this paper, can be used to compute the Gaussian filter, the Laplacian of the Gaussian edge detector, and scale-space.

Expandability is a very important feature in hardware systems; that gives the flexibility to expand the basic design to solve problems of similar nature, but of larger dimensions. The basic design in our approach for one dimension can be easily expanded for computations of higher dimensions. We will discuss in Section 5 how the hardware for one-dimensional convolution can be used to compute two-dimensional convolution and scale space. Finally, the purpose of our proposed architecture is to design a highly parallel real-time hardware. In our architecture, each PE is organized as a pipeline of $8 + m$ stages, where m is the size of the mask and a total of $1 + (m/2)$ PEs are used. For instance, in the example discussed in the next section, m is equal to 5, and hence each PE consists of 13 stages and three such PEs are required by our algorithm. The pipeline architecture makes it possible for our algorithm to run in linear time ($O(n + m)$), where n is the number of pixels and m is the size of the mask. The host broadcasts one pixel value per clock cycle to the convolver chip. It takes $8 + m$ clock cycles to fill the pipe in the first PE, after which the host starts receiving the resultant pixel values at the rate of one per clock cycle. Thus, it takes $n + 8 + m - 1$ cycles to perform one-dimensional convolution with a mask of size m on n pixels. The relationship between the mask size m and σ depends upon many factors including the number of bits used in a particular implementation and has been discussed extensively in [13, 12]. Approximately, the value of the Gaussian $e^{-x^2/2\sigma^2}$ becomes almost zero for $x > |3\sigma|$. Therefore, we suggest using $m = 7\sigma$ for odd values of σ and $m = 7\sigma + 1$ when σ is even.

Besides the above three principles, our implementation fully utilizes the three properties of the Gaussian, i.e., *scaling*, *symmetry*, and *separability*. When convolved with itself, the Gaussian of standard deviation σ, yields a larger Gaussian of standard deviation $\sqrt{2}\,\sigma$. That is, if an image has been filtered with a Gaussian at a certain spread σ and if the same image must be filtered with a larger Gaussian with spread $\sqrt{2}\,\sigma$, then, instead of filtering the image with the larger Gaussian, the previous result can just be filtered with the same filter of spread σ used to obtain the desired image. Thus, the total number of operations for filtering the image by Gaussian of σ and $\sqrt{2}\,\sigma$ will be equal to $2\,n\,\sigma$. The above process is called scaling. However, without scaling, the number of operations will be approximately equal to $2.4\,n\,\sigma$. This produces a significant reduction in the number of operations needed for computations like generating scale-space, where operators of multiple sizes are applied to the same image.

The scaling property of the Gaussian also holds in two dimensions. Although, the second derivative of the Gaussian in one dimension and the Laplacian of Gaussian in two dimensions do not possess this scaling property, it is possible to obtain the effect of applying bigger operators by repeatedly applying the smaller operators to the image. First apply the second derivative of the Gaussian operator of size σ to the image and then apply the Gaussian of size σ to the output. The result will be equivalent to the output obtained by applying the second derivative of the Gaussian operator of size $\sqrt{2}\,\sigma$. In the Appendix, we state and prove a proposition for the general case.

Our approach is to design an architecture that can be used for convolutions of windows of any size. Once the chip is built for a particular mask size, say 7, it can be used as shown in the above paragraph for masks of any bigger size. For masks of smaller size, we need to turn "OFF" the unneeded PEs and use smaller size FIFO buffers. This can be done easily by replacing the FIFOs in our design with variable size FIFOs.

A two-dimensional Gaussian filter can be separated into two one-dimensional Gaussians, one along the x direction and the other along the y direction. Therefore, the Gaussian filter can be applied to an image by convolving first with a one-dimensional Gaussian along each row and then convolving the result again with a one-dimensional Gaussian along each column. Each one-dimensional convolution with an operator of size m requires m multiplications per pixel. Hence, two one-dimensional convolutions require $2m$ multiplications, which is a significant improvement over the m^2 multiplications needed for a two-dimensional convolution. Unfortunately, the LG operator is not separable into two single-dimensional operators which is due to the fact that the Laplacian is not separable, even though the Gaussian is. We give an algorithm to decompose the two-dimensional LG convolution into four one-dimensional convolutions. This scheme requires $4m$ multiplications. For a large m, $4m$ multiplications are less than m^2 multiplications. Therefore, the number of multiplications is significantly reduced for larger images. This will also be shown in the Appendix. We can summarize the algorithm for the decomposition of the operator as follows, refer to Fig. 9b for details:

(1) Convolve the image with a second derivative of Gaussian mask along each row.

(2) Convolve the resultant image from step (1) by a Gaussian mask along each column. Call the resultant image I^x.

(3) Convolve the original image with a second derivative of Gaussian mask along each column.

(4) Convolve the resultant image from step (3) by a Gaussian mask along each row. Call the resultant image I^y.

(5) Add I^x and I^y.

The Gaussian is symmetric around the origin, i.e., $g(x) = g(-x)$ for any x. This property can be used to reduce the number of multiplications as follows. Assume that the operator of size 5 is to be applied to the input sequence $X_1, X_2, X_3, X_4, X_5, X_6, \ldots, X_n$ in order to get the output sequence $Y_4, Y_5, Y_6, \ldots$. For instance, the equation for the computation of Y_5 is

$$Y_5 = w_0 X_1 + w_1 X_2 + w_2 X_3 + w_3 X_4 + w_4 X_5.$$

Due to the symmetry, $w_0 = w_4$ and $w_1 = w_3$. The convolution equations can be

$$\begin{aligned}
Y_4 &= w_0X_4 + w_1(X_3 + X_1) + w_2 X_2 \\
Y_5 &= w_0(X_5 + X_1) + w_1(X_4 + X_2) + w_2X_3 \\
Y_6 &= w_0(X_6 + X_2) + w_1(X_5 + X_3) + w_2X_4 \\[6pt]
Y_7 &= w_0(X_7 + X_3) + w_1(X_6 + X_4) + w_2X_5 \\
Y_8 &= w_0(X_8 + X_4) + w_1(X_7 + X_5) + w_2X_6 \\
Y_9 &= w_0(X_9 + X_5) + w_1(X_8 + X_6) + w_2X_7 \\[6pt]
Y_{10} &= w_0(X_{10} + X_6) + w_1(X_9 + X_7) + w_2X_8 \\
Y_{11} &= w_0(X_{11} + X_7) + w_1(X_{10} + X_8) + w_2X_9 \\
Y_{12} &= w_0(X_{12} + X_8) + w_1(X_{11} + X_9) + w_2X_{10}
\end{aligned}$$

FIG. 1. The equations for LG convolution for mask size = 5.

simplified as shown in Fig. 1. Utilizing symmetry, we reduce the number of multiplications to compute all the Y_i elements, from $5n$ to $3n$, where n is the total number of elements. In general, when using the property of symmetry, we only have to perform $(m/2 + 1) * n$ multiplications which is a significant reduction from $m * n$, where m is the number of weights in the mask.

4. ONE DIMENSION

In this section, we will present the hardware design for computing the scale-space in one dimension and the design will be extended to two dimensions in the next section. The different stages in computing one-dimensional scale-space are shown in Fig. 2a. They are four main stages: Laplacian of the Gaussian LG(x), Gaussian filter GF(x), normalization NM(x), and zero-crossing detector ZC(x). Also, we can obtain a one-dimensional edge detector and a Gaussian filter with stages as depicted in Figs. 2b,c. The one-dimensional edge detector is realized by applying the Laplacian of the Gaussian operator to the image, normalizing the resultant pixel values, and then applying the zero-crossing detector. The Gaussian filter is obtained by convolving the image with the Gaussian and normalizing the resultant pixel values.

4.1. *VLSI Architecture for the LG Convolution*

The VLSI architecture for the computation of the Gaussian and the LG convolutions is given in Fig. 3. The architecture consists of a certain number of PEs organized in parallel with a common input and output bus as shown in the figure. A zero-crossing detector module is also included which will output the binary image. In a binary image, a "1" represents the presence of a zero crossing, while a "0" represents the absence. The chip needs a total of 38 pins. Eight of the input pins are for the pixel data, eight for the weights, one pin for the switch bit needed by the PEs, one for multiplexing the output, and three for VDD, GND, and CLOCK. There is one output pin for the edge image and 16 pins for the convolved image. The 16-bit pixel values obtained from the convolution are normalized to 8-bit results by the normalization module and then input to the zero-crossing detector circuit. The design of the normalization module is discussed later in Subsection 4.4. The 2:1 Multiplexer selects between the 16-bit pixel values and the normalized 8-bit results.

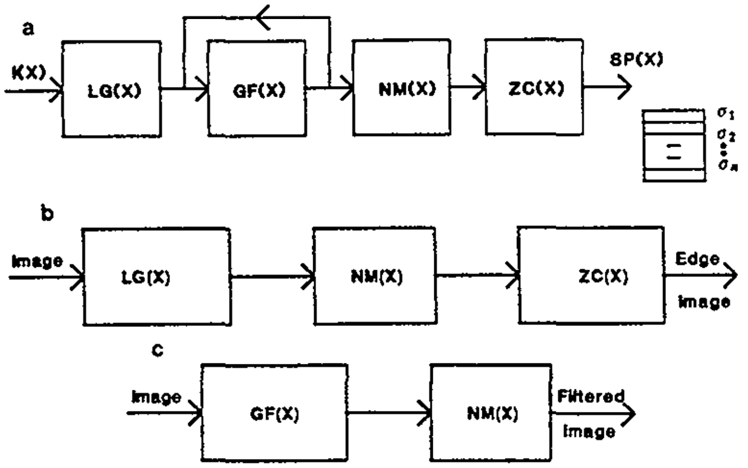


FIG. 2.(a). One-dimensional scale-space. The image I(x) is convolved with the Laplacian of the Gaussian LG(x) and then convolved with Gaussian GF(x) repeatedly depending on the scale needed. The resultant pixel values are normalized in the next stage NM(x) and in the final stage ZC(x), the zero crossings are detected to produce the edge image. SP(x) is the scale-space generated at various scales. (b) One-dimensional edge detector. The one-dimensional edge detector is realized by applying the Laplacian of Gaussian operator LG(x) to the image I(x), normalizing the resultant pixel values (shown as NM(x) stage), and then applying the zero-crossing detector ZC(x). (c) One-dimensional Gaussian filter. The Gaussian filter is realized by convolving with the Gaussian, shown as GF(x), and normalizing the resultant pixel values (shown as NM(x) stage).

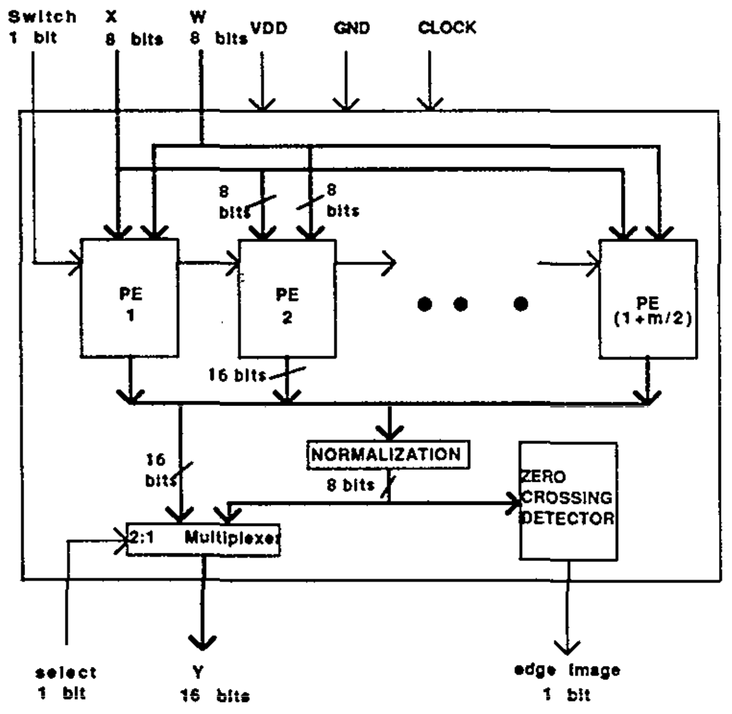


FIG. 3. Architecture of the proposed VLSI chip.

When the normalized results are selected, the most significant 8 bits of the output (Y) will provide the normalized result and the other 8 bits shall be ignored. The required output can be obtained by setting the select pin appropriately. The number of processing elements (PEs) will be equal to $1 + m/2$, where m is the size of the mask as well as the number of weights required for the convolution.

The parallel algorithm for one-dimensional LG convolution will be explained with the help of an example where the size of the mask is 5, and therefore the number of PEs is 3. The equations for this example are given in Fig. 1. The image pixel values, called X_i, are broadcast to all the PEs at once. The host sends a new X_i to the chip during each cycle and when the pipe within the PE is filled, each PE starts producing a Y_i element every third cycle, which is relayed on the output bus. In general, a PE will produce Y_i values with a gap of cycles equal to $\lfloor m/2 \rfloor$ which is one less than the number of PEs and hence the output bus is time-shared by the PEs. Thus, a resultant Y_i value is produced at the rate of one per clock cycle. In our example, first, PE1 computes Y_4; during the next cycle PE2 computes Y_5; and then PE3 follows with Y_6 during the third. After this, PE1 is ready again with Y_7 and so on. We will demonstrate how we can obtain this synchronization by suitably designing the PE architecture.

TABLE 1a
Trace of Clock-Step Execution of PE1 for the Architecture of Fig. 3

Clock	Inbus	FIFO-1					FIFO-2			Start
cycle	X	1	2	3	4	5	6	7	CLA-1	multiply
1	X_1	X_1	∅	∅	∅	∅	∅	∅	—	—
2	X_2	X_2	X_1	∅	∅	∅	∅	∅	—	—
3	X_3	X_3	X_2	X_1	∅	∅	∅	∅	—	—
4	X_4	X_4	X_3	X_2	X_1	∅	X_3	X_4	—	—
5	X_5	X_5	X_4	X_3	X_2	X_1	∅	X_3	$\varnothing + X_4$	—
6	X_6	X_6	X_5	X_4	X_3	X_2	∅	∅	$X_1 + X_3$	$W_0(X_4)$
7	X_7	X_7	X_6	X_5	X_4	X_3	X_6	X_7	$X_2 + \varnothing$	$W_1(X_1 + X_3)$
8	X_8	X_8	X_7	X_6	X_5	X_4	∅	X_6	$X_3 + X_7$	$W_2(X_2)$
9	X_9	X_9	X_8	X_7	X_6	X_5	∅	∅	$X_4 + X_6$	$W_0(X_3 + X_7)$
10	X_{10}	X_{10}	X_9	X_8	X_7	X_6	X_9	X_{10}	$X_5 + \varnothing$	$W_1(X_4 + X_6)$
11	X_{11}	X_{11}	X_{10}	X_9	X_8	X_7	∅	X_9	$X_6 + X_{10}$	$W_2(X_5)$
12	X_{12}	X_{12}	X_{11}	X_{10}	X_9	X_8	∅	∅	$X_7 + X_9$	$W_0(X_6 + X_{10})$
13	X_{13}	X_{13}	X_{12}	X_{11}	X_{10}	X_9	X_{12}	X_{13}	$X_8 + \varnothing$	$W_1(X_7 + X_9)$
14	X_{14}	X_{14}	X_{13}	X_{12}	X_{11}	X_{10}	∅	X_{12}	$X_9 + X_{13}$	$W_2(X_8)$
15	X_{15}	X_{15}	X_{14}	X_{13}	X_{12}	X_{11}	∅	∅	$X_{10} + X_{12}$	$W_0(X_9 + X_{13})$

4.2. *PE Architecture and Operation*

The PE is organized as a pipeline of several stages as shown in Fig. 4. This is essential in order to get high throughput and speed while convolving image data of the order of 1000 by 1000 pixels. The hardware organization of a processing element is given in Fig. 5. Each processing element consists of a pipeline Wallace multiplier, two carry look-ahead adders, an accumulator, a circular FIFO buffer for the weights, and two FIFO buffers for the incoming X_i values. The weights are pre-loaded into the circular FIFO buffer before the computation starts. The X_i values enter the **FIFO-1** at the rate of one every clock cycle. In order to use the symmetry property of the Gaussian, $\lfloor m/2 \rfloor$ of the X_i values are needed in the reverse order. This is done by setting the control signal "switch" high during every third clock cycle (once the computation starts), for our example. The "switch" is connected to the gates of a set of enhancement mode n-channel MOSFET transistors [18] as shown in Fig. 5. The MOSFET transistors act as pass transistors that help to copy the data from **FIFO-1** to **FIFO-2**. When the control signal "switch" is high, the pass transistors act as closed circuits allowing the data to pass through and when the control signal is low, the pass transistors act as open circuits and no data can pass through. The **FIFO-1** buffer consists of five 8-bit registers organized in a

TABLE 1b

Clock cycle	CLA-2 addition	Outbus Y
8	—	—
9	—	—
10	—	—
11	$W_0X_4 + (\emptyset)$	—
12	$W_1(X_1 + X_3) + (W_0X_4)$	—
13	$W_2X_2 + (W_1(X_1 + X_3) + W_0X_4)$	—
14	$W_0(X_3 + X_7) + (\emptyset)$	Y_4
15	$W_1(X_4 + X_5) + (W_0(X_3 + X_7))$	—
16	$W_2X_5 + (W_1(X_4 + X_6) + W_0(X_3 + X_7))$	—
17	$W_0(X_6 + X_{10}) + (\emptyset)$	Y_7
18	$W_1(X_7 + X_9) + (W_0(X_6 + X_{10}))$	—
19	$W_2(X_8) + (W_1(X_7 + X_9) + W_0(X_6 + X_{10}))$	—
20	$W_0(X_9 + X_{13}) + (\emptyset)$	Y_{10}
21	$W_1(X_{10} + X_{12}) + (W_0(X_9 + X_{13}))$	—
22	$W_2(X_{11}) + (W_1(X_{10} + X_{12}) + W_0(X_9 + X_{13}))$	—

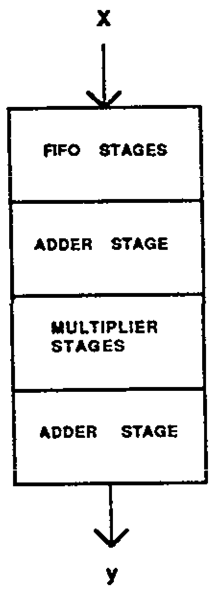


FIG. 4. The pipeline stages in a PE.

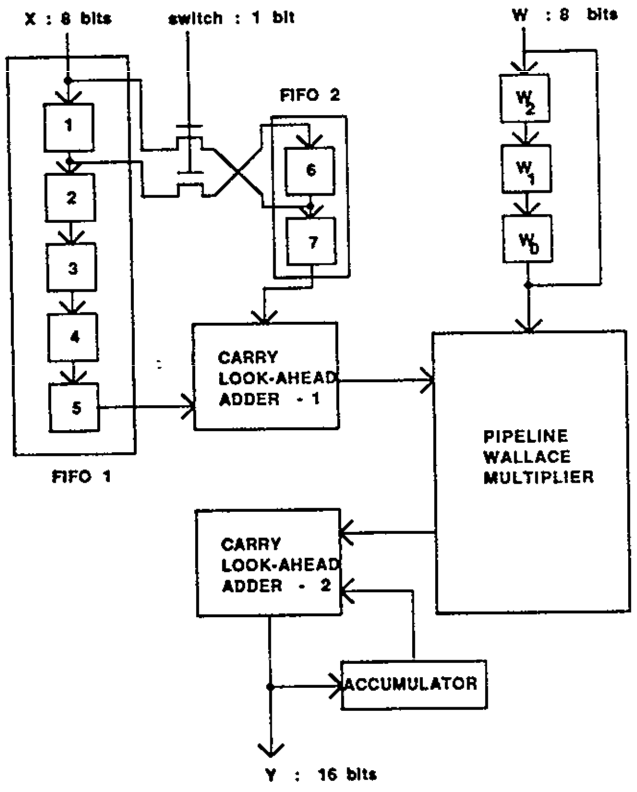


FIG. 5. Architecture of a processing element (PE).

first-in first-out fashion and the **FIFO-2** buffer consists of two 8-bit registers with additional load capability as shown in Fig. 5.

The 8-bit multiplier will have a five-stage pipeline where the first stage is the partial product generator and the second through fourth stages consist of carry save adders and the last stage is a 16-bit carry look-ahead adder [27]. As can be seen from the equations of Fig. 1, the same weight needs to be multiplied with two different pixel values. During each clock cycle, a new pair of pixel values are added in the carry look-ahead adder. The sum is input to the Wallace multiplier during the following clock cycle along with the weight to multiply it with. The weights are stored in a circular FIFO buffer. The weight input to the multiplier from the buffer is also loaded back into the buffer in a circular fashion, while the rest of the weights shift to the next stage nearer the multiplier. A new multiplication is initiated during each clock cycle. As the pixels are shifted into the carry look-ahead adder from **FIFO-2**, zeros are inserted in the other end. Thus, a zero is output by **FIFO-2** during those cycles when no actual addition is required, which corresponds to the situation when a weight has to be multiplied with only one pixel in an equation. The outputs of the multiplier are summed up and stored in the accumulator in order to obtain the Y_i value which is to be relayed on the output bus. A sample trace of the execution of the three PEs is given in the Tables 1–3. These tables illustrate the

TABLE 2a
Trace of Clock-Step Execution of PE2 for the architecture of Fig. 3

Clock	Inbus	FIFO-1					FIFO-2			Start
cycle	X	1	2	3	4	5	6	7	CLA-1	multiply
1	X_1	X_1	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	—	—
2	X_2	X_2	X_1	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	—	—
3	X_3	X_3	X_2	X_1	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	—	—
4	X_4	X_4	X_3	X_2	X_1	$\emptyset$	$\emptyset$	$\emptyset$	—	—
5	X_5	X_5	X_4	X_3	X_2	X_1	X_4	X_5	—	—
6	X_6	X_6	X_5	X_4	X_3	X_2	$\emptyset$	X_4	$X_1 + X_5$	—
7	X_7	X_7	X_6	X_5	X_4	X_3	$\emptyset$	$\emptyset$	$X_2 + X_4$	$W_0(X_1 + X_5)$
8	X_8	X_8	X_7	X_6	X_5	X_4	X_7	X_8	$X_3 + \emptyset$	$W_1(X_2 + X_4)$
9	X_9	X_9	X_8	X_7	X_6	X_5	$\emptyset$	X_7	$X_4 + X_8$	$W_2(X_3)$
10	X_{10}	X_{10}	X_9	X_8	X_7	X_6	$\emptyset$	$\emptyset$	$X_5 + X_7$	$W_0(X_4 + X_8)$
11	X_{11}	X_{11}	X_{10}	X_9	X_8	X_7	X_{10}	X_{11}	$X_6 + \emptyset$	$W_1(X_5 + X_7)$
12	X_{12}	X_{12}	X_{11}	X_{10}	X_9	X_8	$\emptyset$	X_{10}	$X_7 + X_{11}$	$W_2(X_6)$
13	X_{13}	X_{13}	X_{12}	X_{11}	X_{10}	X_9	$\emptyset$	$\emptyset$	$X_8 + X_{10}$	$W_0(X_7 + X_{11})$
14	X_{14}	X_{14}	X_{13}	X_{12}	X_{11}	X_{10}	X_{13}	X_{14}	$X_9 + \emptyset$	$W_1(X_8 + X_{10})$
15	X_{15}	X_{15}	X_{14}	X_{13}	X_{12}	X_{11}	$\emptyset$	X_{13}	$X_{10} + X_{14}$	$W_2(X_9)$

TABLE 2b

Clock cycle	CLA-2 addition	Outbus Y
8	—	—
9	—	—
10	—	—
11	—	—
12	$W_0(X_1 + X_5) + (\varnothing)$	—
13	$W_1(X_2 + W_4) + (W_0(X_1 + X_5))$	—
14	$W_2X_3 + (W_1(X_2 + X_4) + X_0(X_1 + X_5))$	—
15	$W_0(X_4 + X_8) + \varnothing$	Y_5
16	$W_1(X_5 + X_7) + (W_0(X_4 + X_8))$	—
17	$W_2X_6 + (W_1(X_5 + X_7) + W_0(X_0(X_4 + X_8))$	—
18	$W_0(X_7 + X_{11}) + (\varnothing)$	Y_8
19	$W_1(X_8 + X_{10}) + (W_0(X_7 + X_{11}))$	—
20	$W_2X_9 + (W_1(X_8 + X_{10}) + W_0(X_7 + X_{11}))$	—
21	$W_0(X_{10} + X_{14}) + (\varnothing)$	Y_{11}
22	$W_1(X_{11} + X_{13}) + (W_0(X_{10} + X_{14}))$	—

working of the algorithm and the synchronization of the various components of our chip.

The hardware of each PE requires no control logic except for the signal "switch" which is used to copy part of the X_i values into the buffer **FIFO-2** to aid in the addition of these values before they are multiplied by the weights. In the PE architecture shown in Fig. 5, the two sets of lines connecting **FIFO-1** and **FIFO-2** cross each other and this organization is possible for small mask sizes since current VLSI technologies support double metal layers commonly. But for very large mask sizes where the number of cross-overs increases considerably, the organization becomes unsuitable for VLSI implementation. This problem can be solved by modifying the PE architecture to include a third FIFO as given in Fig. 6. As the X_i values enter **FIFO-1**, they also get loaded into **FIFO-2** and, once every third cycle, the values in **FIFO-2** are copied with the parallel load capability into **FIFO-3**. This type of organization is suitable for masks of any size. Note that the size of **FIFO-2** and **FIFO-3** will be the same and equal to $\lfloor m/2 \rfloor$ where m is the mask size. The size of **FIFO-1** is m.

4.3. *Systolic Version of the Proposed Architecture*

The efficient implementation of an algorithm is VLSI depends on several factors like the use of cells that can be repeated in space, extensive use of pipelining and

TABLE 3a

Trace of Clock-Step Execution of PE3 for the architecture of Fig. 3

Clock	Inbus	FIFO-1					FIFO-2			Start
cycle	X	1	2	3	4	5	6	7	CLA-1	multiply
1	X_1	X_1	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	—	—
2	X_2	X_2	X_1	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	—	—
3	X_3	X_3	X_2	X_1	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	—	—
4	X_4	X_4	X_3	X_2	X_1	$\varnothing$	$\varnothing$	$\varnothing$	—	—
5	X_5	X_5	X_4	X_3	X_2	X_1	$\varnothing$	$\varnothing$	—	—
6	X_6	X_6	X_5	X_4	X_3	X_2	X_5	X_6	—	—
7	X_7	X_7	X_6	X_5	X_4	X_3	$\varnothing$	X_5	$X_2 + X_6$	—
8	X_8	X_8	X_7	X_6	X_5	X_4	$\varnothing$	$\varnothing$	$X_3 + X_5$	$W_0(X_2 + X_6)$
9	X_9	X_9	X_8	X_7	X_6	X_5	X_8	X_9	$X_4 + \varnothing$	$W_1(X_3 + X_5)$
10	X_{10}	X_{10}	X_9	X_8	X_7	X_6	$\varnothing$	X_8	$X_5 + X_9$	$W_2(X_4)$
11	X_{11}	X_{11}	X_{10}	X_9	X_8	X_7	$\varnothing$	$\varnothing$	$X_6 + X_8$	$W_0(X_5 + X_9)$
12	X_{12}	X_{12}	X_{11}	X_{10}	X_9	X_8	X_{11}	X_{12}	$X_7 + \varnothing$	$W_1(X_6 + X_8)$
13	X_{13}	X_{13}	X_{12}	X_{11}	X_{10}	X_9	$\varnothing$	X_{11}	$X_8 + X_{12}$	$W_2(X_7)$
14	X_{14}	X_{14}	X_{13}	X_{12}	X_{11}	X_{10}	$\varnothing$	$\varnothing$	$X_9 + X_{11}$	$W_0(X_8 + X_{12})$
15	X_{15}	X_{15}	X_{14}	X_{13}	X_{12}	X_{11}	X_{14}	X_{15}	$X_{10} + \varnothing$	$W_1(X_9 + X_{11})$

parallelism, and avoidance of global communication. The chip proposed in Section 4.1 uses a PE cell that can be repeated in space and the algorithm has a high degree of pipelining and parallelism. But the algorithm requires global broadcast of the input to all the PEs in parallel and, similarly, the output from any PE has to be connected to the output pads through a common bus. This can be achieved without degradation of performance by using powerful bus drivers and precharge logic. However, a better alternative is to use a systolic algorithm where global communication is avoided. We will show that our algorithm can be modified such that is can be implemented using a systolic array without changing the PE architecture.

The systolic organization of the PEs is shown in Fig. 7. The basic PE architecture remains the same and the figure depicts the organization for the example discussed in the previous section where a mask of size 5 is assumed. The bus that connected the outputs of all the PEs in Fig. 3 is replaced by a set of multiplexers and registers as shown in Fig. 7. This modification is necessary since in the new algorithm, all the PEs output their results simultaneously during the same clock cycle. During this cycle, the 2 : 1 multiplexers select the Y_i values from the PEs to be loaded into the corresponding registers. During the rest of the cycles, the results stored in the registers are shifted left to be output sequentially. When the results are being shifted out, the PEs continue to compute the next set of Y_i values. Our algorithm works

TABLE 3b

Clock cycle	CLA-2 addition	Outbus Y
8	—	—
9	—	—
10	—	—
11	—	—
12	—	—
13	$W_0(X_2 + X_6) + (\varnothing)$	—
14	$W_1(X_3 + X_5) + (W_0(X_2 + X_6))$	—
15	$W_2X_4 + (W_1(X_3 + X_5) + W_0(X_2 + X_6))$	—
16	$W_0(X_5 + X_9) + (\varnothing)$	Y_6
17	$W_1(X_6 + X_8) + (W_0(X_5 + X_9))$	—
18	$W_2X_7 + (W_1(X_6 + X_8) + W_0(X_5 + X_9))$	—
19	$W_0(X_8 + X_{12}) + (\varnothing)$	Y_9
20	$W_1(X_9 + X_{11}) + (W_0(X_8 + X_{12}))$	—
21	$W_2X_{10} + (W_1(X_9 + X_{11}) + W_0(X_8 + X_{12}))$	—
22	$W_0(X_{11} + X_{11}) + (\varnothing)$	Y_{12}

such that when all the results have been output, the PEs are ready to output a new set of results. Thus, the chip outputs a resultant pixel value Y_i at the rate of one per cycle.

The algorithm works as follows: a new pixel value X_i is input to the right most processing element PE3 during each clock cycle. In the following cycle, the PE3 will pass on the value to the neighboring PE2 while it receives a new one. This can be done by connecting the output of the first register in FIFO-1 to the input of the neighboring PE. Tables 4–6 illustrate the working of the algorithm as well as the synchronization of the PEs. Since the signal "switch" within each PE has to be "on" at the same time and at regular intervals, the signal can be controlled globally. Always, this signal is high only during the cycle that precedes the one when the PEs produce a new set of results. Hence the same signal can be delayed by one cycle to control the multiplexer at the corresponding stage. The multiplexer forwards the result from the neighboring PE during each clock cycle, except when new results are generated. Then the result from the PE in the corresponding stage is forwarded.

4.4. *Normalization*

The convolved image consists of 16-bit values which are output by the PEs. If the system is built to obtain a Gaussian filter, then the resultant image pixels have to be normalized to 8-bit values in order to display the filtered image. Since the original

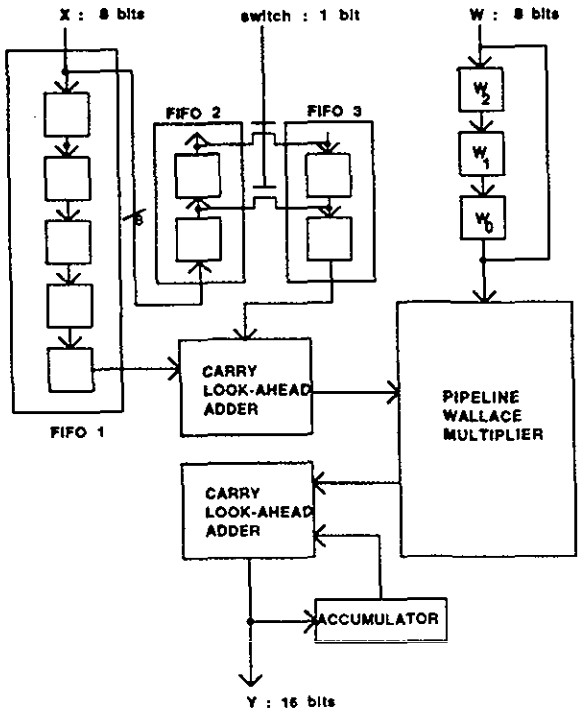


FIG. 6. A modified architecture for the processing element.

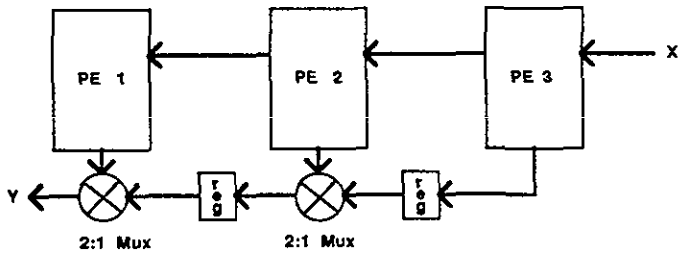


FIG. 7. A systolic version of the proposed chip. This is an alternate proposal to that of Fig. 3.

pixels are 8-bit values, the resultant pixel values after the Gaussian convolution can be divided by 256 to yield values less than 255. This can be achieved by performing arithmetic shift-right eight times, which is equivalent to dividing by 2^8. However, by doing this we will lose some accuracy if the value being shifted is negative. To avoid this, we will use an alternate method which is to take the 8 most significant bits and add the sign bit to it. Thus, the normalization requires an 8-bit adder circuit and can be performed in one clock cycle.

4.5. *Zero-Crossing Detector*

The circuit for a zero-crossing detector is given in Fig. 8a. The circuit consists of two exclusive-OR gates, an 8-bit register, precharge logic to test if the pixel value is

TABLE 4
Trace of Clock-Step Execution of PE1 for the Architecture of Fig. 7

Clock cycle	Switch	FIFO-1					FIFO-2		Output of PE1	Chip output
		1	2	3	4	5	6	7		
1	0	0	0	0	0	0	0	0	—	—
2	0	0	0	0	0	0	0	0	—	—
3	0	X_1	0	0	0	0	0	0	—	—
4	0	X_2	X_1	0	0	0	0	0	—	—
5	0	X_3	X_2	X_1	0	0	0	0	—	—
6	1	X_4	X_3	X_2	X_1	0	X_3	X_4	—	—
15	1	X_{13}	X_{12}	X_{11}	X_{10}	X_9	X_{12}	X_{13}	—	—
16	0	X_{14}	X_{13}	X_{12}	X_{11}	X_{10}	0	X_{12}	Y_4	Y_4
17	0	X_{15}	X_{14}	X_{13}	X_{12}	X_{11}	0	0	—	Y_5
18	1	X_{16}	X_{15}	X_{14}	X_{13}	X_{12}	X_{15}	X_{16}	—	Y_6
19	0	X_{17}	X_{16}	X_{15}	X_{14}	X_{13}	X_0	X_{15}	Y_7	Y_7
20	0	X_{18}	X_{17}	X_{16}	X_{15}	X_{14}	0	0	—	Y_8

TABLE 5
Trace of Clock-Step Execution of PE2 for the Architecture of Fig. 7

Clock cycle	Switch	FIFO-1					FIFO-2		Output of PE2	Chip output
		1	2	3	4	5	6	7		
1	0	0	0	0	0	0	0	0	—	—
2	0	X_1	0	0	0	0	0	0	—	—
3	0	X_2	X_1	0	0	0	0	0	—	—
4	0	X_3	X_2	X_1	0	0	0	0	—	—
5	0	X_4	X_3	X_2	X_1	0	0	0	—	—
6	1	X_5	X_4	X_3	X_2	X_1	X_4	X_5	—	—
15	1	X_{14}	X_{13}	X_{12}	X_{11}	X_{10}	X_{13}	X_{14}	—	—
16	0	X_{15}	X_{14}	X_{13}	X_{12}	X_{11}	0	X_{13}	Y_5	X_4
17	0	X_{16}	X_{15}	X_{14}	X_{13}	X_{12}	0	0	—	Y_5
18	1	X_{17}	X_{16}	X_{15}	X_{14}	X_{13}	X_{16}	X_{17}	—	Y_6
19	0	X_{18}	X_{17}	X_{16}	X_{15}	X_{14}	0	X_{16}	Y_8	Y_7
20	0	X_{19}	X_{18}	X_{17}	X_{16}	X_{15}	0	0	—	Y_8

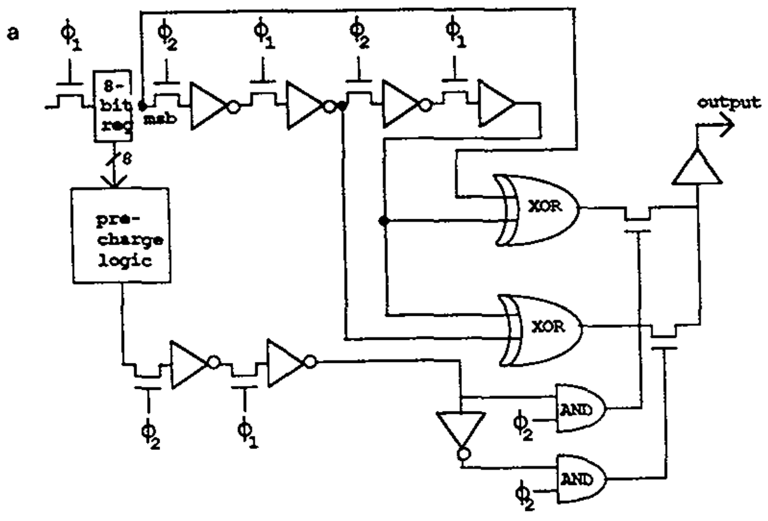
TABLE 6
Trace of Clock-Step Execution of PE3 for the Architecture of Fig. 7

Clock cycle	Switch	FIFO-1					FIFO-2		Output of PE3	Chip output
		1	2	3	4	5	6	7		
1	0	X_1	0	0	0	0	0	0	—	—
2	0	X_2	X_1	0	0	0	0	0	—	—
3	0	X_3	X_2	X_1	0	0	0	0	—	—
4	0	X_4	X_3	X_2	X_1	0	0	0	—	—
5	0	X_5	X_4	X_3	X_2	X_1	0	0	—	—
6	1	X_6	X_5	X_4	X_3	X_2	X_5	X_6	—	—
15	1	X_{15}	X_{14}	X_{13}	X_{12}	X_{11}	X_{14}	X_{15}	—	—
16	0	X_{16}	X_{15}	X_{14}	X_{13}	X_{12}	0	X_{14}	Y_6	Y_4
17	0	X_{17}	X_{16}	X_{15}	X_{14}	X_{13}	0	0	—	Y_5
18	1	X_{18}	X_{17}	X_{16}	X_{15}	X_{14}	X_{17}	X_{18}	—	Y_6
19	0	X_{19}	X_{18}	X_{17}	X_{16}	X_{15}	0	X_{17}	Y_9	Y_7
20	0	X_{20}	X_{19}	X_{18}	X_{17}	X_{16}	0	0	—	Y_8

zero, and 1-bit register stages to insert delays. We need a look-ahead of size 2, since there is a zero crossing if one of the following situations occur: $\{-,+\}$, $\{+,-\}$, $\{-,0,+\}$, or $\{+,0,-\}$, where $-$ and $+$ indicate the sign of the pixel value and 0 indicates that the value is zero. A new pixel value is loaded into the 8-bit register during each ϕ_1 phase and the precharge logic outputs a "1" whenever the loaded value is zero. The precharge logic is quite simple and is given in Fig. 8b.

During the ϕ_2 phase the bus is precharged to high, and during the ϕ_1 phase if any of the 8 pixel bits is non-zero then the bus is pulled down to zero. If all the pixel bits are "0," the output of the precharge logic remains high. The value is delayed by one cycle and used to multiplex between the outputs of the two exclusive-OR gates which detect sign change in the consecutive pixels or in those on the opposite sides of a zero. If a pixel is zero, then the exclusive-OR of the pixels adjacent to it are connected to the output. Otherwise, the exclusive-OR of the consecutive pixels is connected to the output.

As it is apparent from Figs. 8a and b that the circuit is quite simple in terms of the logic as well as the amount of hardware required. One could design a finite state machine for the problem and generate a programmable logic array with feedback. Due to the availability of automated PLA generator software, it is easy to generate PLAs for complex logic equations. But it will not be optimal in terms of the layout for our circuit, since 8-bit pixel values must be input to the PLA, the PLA will be large in size. The speed of a PLA reduces as its size increases. Out circuit is definitely better than the PLA implementation since its critical delay is only that of a two-bit exclusive-OR stage.



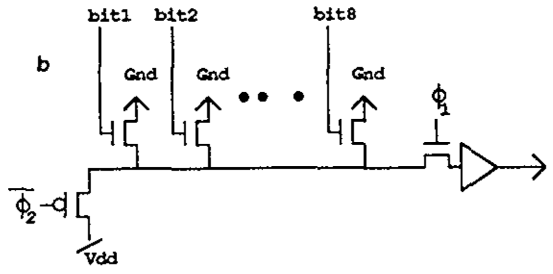


FIG. 8.(a). The zero-crossing detector circuit. (b) Precharge logic to detect pixel value "0".

5. TWO DIMENSIONS

Once we have the hardware system that can compute scale-space in one dimension, we can expand it to compute two-dimensional scale-space, as summarized in Fig. 9a. The idea is to compute the Laplacian of the Gaussian, LG(x, y) and then compute the Gaussian filter, GF(x, y), repetitively depending on the scale needed. After the convolution, the resultant pixel values are normalized to 8-bit values before the zero-crossings are detected.

The sequence of computations for the two-dimensional LG convolution is given in Fig. 9b. The computation consists of four different stages as shown in the figure. However, we need not use four different convolver chips since the computations of LG(y) and LG(x) cannot begin until the completion of GF(x) and GF(y) computations, respectively. Therefore, we suggest using two convolver chips which can be done by doubling the bandwidth of the connecting bus, since they operate, at most, on 16-bit data. The interface of two convolver chips with the host is shown in Fig. 10. After the two chips compute GF(x) and GF(y) in parallel, the new weights can be loaded into the chips to perform parallel computation of LG(y) and LG(x), respectively. The resultant images I^y and I^x will be summed to obtain LG(x, y), from which the zero crossing can be detected. The computational stages in a two-dimensional zero-crossing detector are shown in Fig. 9c. The zero-crossing

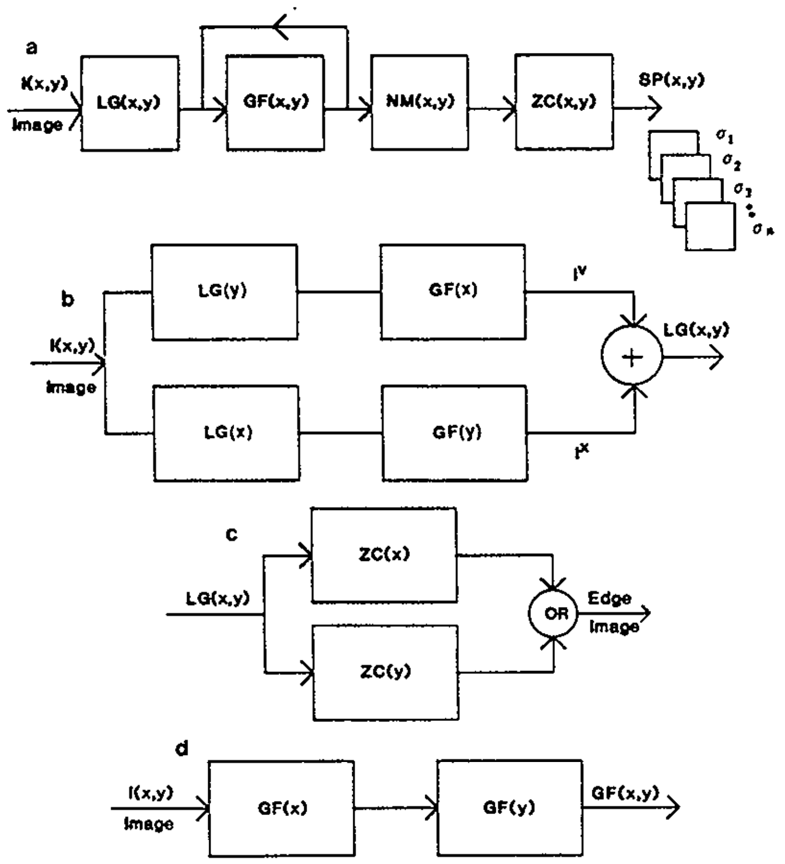


FIG. 9.(a). Two-dimensional scale-space. LG(x, y): Laplacian of the Gaussian, GF(x, y): Gaussian filter; NM(x, y): normalization; and ZC(x, y): zero-crossing detection are functions in two dimensions with axes x and y; SP(x, y): scale-space in two dimensions generated at different scales. (b) Two-dimensional LG convolution. Laplacian of Gaussian in two dimensions LG(x, y) is computed by first applying two one-dimensional Laplacian of Gaussian operators, LG(y) and LG(x) in parallel, and then applying Gaussian filters GF(x) and GF(y), respectively, as shown in figure. The resultant images are summed to obtain the Laplacian of the Gaussian LG(x, y) in two dimensions. (c) Two-dimensional zero-crossing detector. The zero-crossing detection in two dimensions can be computed row-wise and column-wise, ZC(x) and ZC(y), respectively, and the resultant images are logically OR-ed to get the final edge image ZC(x, y). (d) Two-dimensional Gaussian convolution. Two-dimensional Gaussian convolution GF(x, y) is computed as two one-dimensional Gaussian convolutions GF(x) and GF(y) in a sequential manner. Though a single convolver chip will compute both GF(x) and GF(y), the two stages of computation are shown in the figure for clarity.

detection can be computed row-wise and column-wise separately and the two resultant images can be logically OR-ed to get the final edge image.

As previously discussed, the two-dimensional Gaussian convolution can be separated into two one-dimensional convolutions. Since, the two convolutions must be performed serially, a single chip can be interfaced to the host to perform this computation. As shown in Fig. 9d, the image is filtered with GF(x) followed by GF(y) and the result is equivalent to applying the two-dimensional filter GF(x, y).

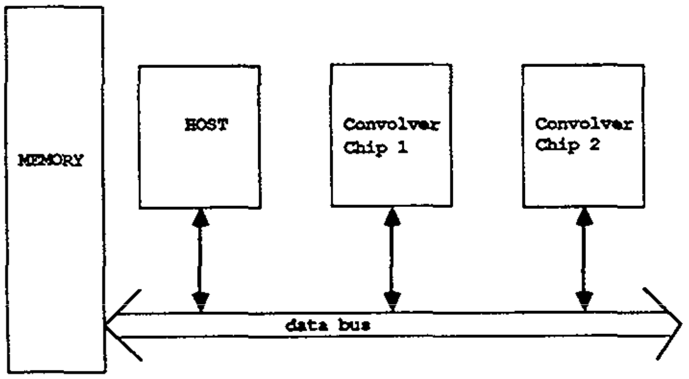


FIG. 10. Interface of the chips for two-dimensional convolution.

Thus, our proposed chip can be used to compute the two-dimensional Gaussian convolution with maximum possible parallelism.

The two-dimensional scale-space computation can be performed with two chips for computing LG(x, y), as discussed above. Though we can compute LG(x, y) in half the time by using two convolver chips, it is not economical since one of the chips has to remain idle when the Gaussian filter is being applied several times in a serial manner. Therefore, a single chip will suffice for the computation of two-dimensional scale-space.

6. CHIP IMPLEMENTATION ISSUES

The chip could be implemented in either CMOS or nMOS technology. Today, CMOS technology is more popular than the nMOS technology because of its low static power consumption. The nMOS chips consume more power, but require less silicon area compared to the CMOS. Although most new designs use CMOS technology, we will relate our discussions in this section to nMOS for the following two reasons. First, a 4-bit pipeline Wallace multiplier with an 8-bit ripple carry adder for the last stage, has been implemented using nMOS 2 μ technology [21]. Second, we can compare speed and performance with other hardware implementations that are discussed under the section on related work.

The organization of our chip was discussed in Section 4.2. Each PE consists of four FIFO buffers (as in the case of Fig. 6), one 8-bit adder, an 8-bit Wallace multiplier, a 16-bit adder, and an accumulator. The critical path delay for the carry chain in an 8-bit ripple carry adder designed was about 50 ns and the Wallace multiplier circuit for 4-bit two's complement multiplication required about 1000 transistors. Hence, we assume that we can implement an 8-bit Wallace multiplier in nMOS technology with roughly 2000 transistors in the circuit. The 8-bit and the 16-bit carry look-ahead adders will need about 300 and 600 transistors, respectively. If we consider the case where each PE needs four FIFOs and if the mask size is m, the total number of transistors required can be calculated as follows. Each FIFO consists of 8-bit registers, where each bit in a register is a shift register stage. A shift register stage can be implemented with roughly 6 transistors and hence we need 48 transistors per 8-bit register in a FIFO. We have one FIFO of size m, two of size

$\lfloor m/2 \rfloor$ and one of size $\lfloor 1 + m/2 \rfloor$ for the weights. So the total number of transistors required for the FIFOs is 48 $(m + m + 1 + m/2)$ which is $24*(5m + 2)$. If we assume a value of 34 for m, the FIFOs will need 4128 transistors for a total of about 7000 transistors per PE. This would mean about 126,000 transistors for 18 PEs and it is possible to build a chip with this number of devices with the currently available technology. The circuit for the zero-crossing detector and the normalizer can each be implemented with less than 100 transistors.

Our algorithm can be implemented as a systolic chip as discussed in Section 4.3. In the systolic design, the critical delay of the chip depends on the 16-bit carry look-ahead adder circuit. A 16-bit carry look-ahead adder circuit can be implemented using efficient circuits in order to have a delay of about 50 ns. Thus, we can realize the chip with a clock rate of about 20 MHz. We will roughly estimate the time required by the chip to compute the one-dimensional convolution with a mask of size 34 on a 512 by 512 image. The PE in this case will have 42 stages $(8 + m)$ in the pipe. In a pipeline of k stages, where $k = 8 + m$, it will take $(n + k - 1)$ cycles to perform n computations. The number of multiplications required to get each result of a convolution depends on the size of the mask being applied. By organizing $1 + \lfloor m/2 \rfloor$ PEs in parallel, our algorithm computes a resultant pixel value, one per clock cycle, after the pipe setup time. Thus, we can perform convolution in linear time $O(n + k)$, which is almost equal to $O(n)$, since k is very small compared to n. To convolve a 512 by 512 image, the number of operations is $(512*512 + 42 - 1)$ which will take about 12.5 ms with the chip operating at the rate of 20 MHz. Similarly, to perform a two-dimensional convolution on a 1000 by 1000 image, it will take about 0.1 s. This is comparable to the performance of the connection machine implementation which takes 0.107 s for a mask of size 31 [8]. The connection machine architecture consists of a processor per pixel and the machine is built with several chips where each chip consists of 16 processors arranged as a two-dimensional array. The convolution module discussed by Nishihara and Larson was estimated to perform a 1000 by 1000 convolution in 1.5 s with the whole module operating at 1 MHz rate. Thus, our convolver is not only cheaper to realize because it is a single chip, but it is also faster than the other systems studied in the literature.

7. CONCLUSIONS

In this paper, we have proposed a VLSI chip that can be used to compute scale-space in one and two dimensions with high speed, efficiency, and throughout. The same VLSI architecture can also be used for implementing the Gaussian filter and the Laplacian of the Gaussian edge detector. The proposed algorithm and the hardware architecture exploit a very high degree of pipelining and parallelism. The chip can be implemented in either nMOS or CMOS technology. The implementation can be done fast, since the PE module, once laid out, can be replicated in space and the PE itself is of simple architecture with very little control logic. We have also shown how our algorithm can be implemented using a systolic array. Our architecture is adaptable for convolutions with masks of any size, which is an important advantage over other implementations such as [10], where the computation is performed bit-serially, with a single-bit processor array of the size of the image. In such architectures, including machines like the GAPP, it becomes increasingly difficult and complex to use those machines for larger masks due to the nearestneighbor interconnection of the processors. The other important advantage is that we are not restricted to powers of two for weights. Finally, our algorithm is not only fast and efficient, but also economical, since we can perform any of these computations using a single chip.

Further research in this direction must concentrate on actually designing and building the chip which can be interfaced to a host and tested in real time in order to obtain measures of speed and performance.

APPENDIX

PROPOSITION I. *The Laplacian of the Gaussian can be written as*

$$\nabla^2 g(x, y) = g(x) * \frac{\partial}{\partial y^2} g(y) + g(y) * \frac{\partial}{\partial x^2} g(x), \tag{1}$$

where $*$ *means convolution.*

Proof. The above can be shown easily by using Fourier transform theory. Let

$$g(x) \leftrightarrows G(\omega)$$

denote a Fourier transform pair. Since the two terms in Eq. (1) are symmetric in x and y, let us compute the first term only. The second term can be found from the first by replacing x with y and y with x. Assume that $G(\omega_1)$ and $G(\omega_2)$ are the Fourier transforms of Gaussians $g(x)$ and $g(y)$. By using the convolution property of the Fourier transform we can write

$$\begin{aligned} g(x) * \frac{\partial}{\partial y^2} g(y) &\leftrightarrows G(\omega_1) * (-i\omega_2)^2 G(\omega_2) \\ &\leftrightarrows G(\omega_1, \omega_2) * (-i\omega_2)^2 \\ &\leftrightarrows (-i\omega_2)^2 * G(\omega_1, \omega_2). \end{aligned} \tag{2}$$

Where, $G(\omega_1, \omega_2)$ is the Fourier transform of bivariate Gaussian $g(x, y)$. Now, by taking the inverse Fourier transform of the right-hand side we get

$$(-i\omega_2)^2 \cdot G(\omega_1, \omega_2) \leftrightarrows \frac{\partial}{\partial y^2} g(x, y). \tag{3}$$

By comparing pairs (2) and (3) we have

$$g(x) * \frac{\partial}{\partial y^2} g(y) = \frac{\partial}{\partial y^2} g(x, y).$$

Similarly, by replacing x by y and y by x in the above equation we have

$$g(y) * \frac{\partial}{\partial x^2} g(x) = \frac{\partial}{\partial x^2} g(x, y).$$

Finally, summing the above two equations we get

$$g(x) * \frac{\partial}{\partial y^2} g(y) + g(y) * \frac{\partial}{\partial x^2} g(x) = \frac{\partial}{\partial y^2} g(x, y) + \frac{\partial}{\partial x^2} g(x, y)$$
$$= \nabla^2 g(x, y),$$

which is exactly Eq. (1). □

PROPOSITION II. *Consider a Gaussian $g^{\sigma}(x)$ of standard deviation σ. The convolution of this Gaussian with itself yields $g^{\sqrt{2}\sigma}$.*

Proof.

$$\begin{aligned} g^{\sigma}(x) * g^{\sigma}(x) &= \int_{-\infty}^{\infty} e^{-\eta^2/2\sigma^2} e^{-(x-\eta)^2/2\sigma^2}\, d\eta \\ &= \int_{-\infty}^{\infty} e^{-(2\eta^2 + x^2 - 2\eta x)/2\sigma^2}\, d\eta \\ &= \int_{-\infty}^{\infty} e^{-(\sqrt{2}\eta - x/\sqrt{2})^2/2\sigma^2} e^{-x^2/4\sigma^2}\, d\eta \\ &= \sqrt{\pi}\,\sigma e^{-x^2/4\sigma^2} \\ &= \sqrt{\pi}\,\sigma g^{\sqrt{2}\sigma}(x). \end{aligned}$$

PROPOSITION III. *Consider a second derivative of Gaussian $\nabla^2 g^a(x)$ of standard deviation a and a Gaussian $g^b(x)$ of standard deviation b. The convolution of these two functions results in a second derivative of Gaussian $\nabla^2 g^{\sqrt{a^2+b^2}}(x)$ of standard deviation $\sqrt{a^2 + b^2}$.*

Proof. By definition we have

$$\nabla^2 g^a(x) * g^b(x) = \left(1 - \frac{x^2}{a^2}\right) g^a(x) * g^b(x).$$

Taking the Fourier transform of the right-hand side we get

$$\left(1 - \frac{x^2}{a^2}\right) g^a(x) * g^b(x) \leftrightarrows \sqrt{2\pi}\,(i\omega)^2 \exp\left(-\frac{\omega^2 a^2}{2}\right) \cdot \sqrt{2\pi} \exp\left(-\frac{\omega^2 b^2}{2}\right)$$
$$\leftrightarrows \sqrt{2\pi}\left(\sqrt{2\pi}\,(i\omega)^2 \exp\left(-\frac{\omega^2(\sqrt{a^2 + b^2})^2}{2}\right)\right).$$

Now, taking the inverse Fourier transform of the right-hand side we get

$$\left(1 - \frac{x^2}{a^2}\right) g^a(x) * g^b(x) = \sqrt{2\pi}\left(1 - \frac{x^2}{(\sqrt{a^2 + b^2})^2}\right) g^{\sqrt{a^2+b^2}}(x)$$
$$\nabla^2 g^a(x) * g^b(x) = \sqrt{2\pi}\,\nabla^2 g^{\sqrt{a^2+b^2}}(x). \ \square$$

TABLE 7
Successive Operator Sizes

Iterations	$a = 1, b = 1$	$a = 1, b = 2$	$a = 1, b = 3$	$a = 2, b = 3$
1	1.4	2.2	3.1	3.6
2	1.7	3.0	4.3	4.7
3	2.0	3.6	5.3	5.6
4	2.2	4.1	6.0	6.3
5	2.4	4.6	6.8	7.0
6	2.6	5.0	7.4	7.6
7	2.8	5.4	8.0	8.2
8	3.0	5.7	8.5	8.7
9	3.1	6.0	9.0	9.2
10	3.3	6.4	9.5	9.7
11	3.5	6.7	10.0	10.2

When $a = b = \sigma$, then the result will be the second derivative of the Gaussian operator of spread $\sqrt{2}\,\sigma$.

In a particular implementation, values of a and b can be chosen so as to suit the required range of values of the operator. The value of b is important, since the successive operator sizes depend on it. In order to give the reader an idea about how the size of the operator increases depending on the initial values of a and b, we have tabulated a few cases in Table 7.

In Table 8, we show one of the possible schemes for achieving the operator sizes of the range $2, 3, 4, 5, \ldots, 10$. In this scheme, we have used two sets of initial values of a and b. The first four sizes of operator are achieved by using the values from column II, while the remaining sizes are obtained from column III.

TABLE 8
Successive Operator Sizes

Iterations	$a = 1, b = 2.2$	$a = 1, b = 3.5$
1	2	5
2	3	6
3	4	7
4		8
5		9
6		10

ACKNOWLEDGMENTS

Initial research on some parts of the work reported in this paper was conducted during M. Shah's stay at the University of Michigan, Ann Arbor. The author is thankful to Professor Ramesh Jain for his guidance and for providing the computing facilities which were supported in part under the U.S. AFSOR Contract F49620-82-0089. The authors are thankful to Ms. Donna Williams for her comments and suggestions.

REFERENCES

1. H. Asada and M. Brady, *The Curvature Primal Sketch*, MIT AI memo 758, 1984.
2. J. Batali, *A Vision Chip*, MIT AI Memo 869, May 1981.
3. A. D. Booth, A signed binary multiplication technique, *Q. J. Mech. Appl. Math.* **4**, 1951, 236–240.
4. M. Brady, Computational approaches to image understanding, *Comput. Surveys* **14**, 1982, 3–71.
5. J. Babaud, A. Witkin, and R. Duda, *Uniqueness of the Gaussian Kernel for Scale-Space Filtering*, Fairchild TR 645; *Flair* **22**, 1983.
6. M. Carlotto, Histogram analysis using a scale-space approach, in *Proceedings, Comput. Vision Pattern Recog.*-2, June 1985, pp. 334–340.
7. J. L. Crowley and A. Parker, A representation for shape based on peaks and ridges in the difference of low-pass transform, *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-6**, No. 2, 1984, 156–170.
8. M. Drumheller, Connection machine stereomatching, in *Proceedings, Natl. Conf. Artif. Intell.*-86, August 1986, pp. 748–753.
9. D. J. Fleet et al., Spatiotemporal inseparability in early visual processing, *Biol. Cyber.* **52**, 1985, 153–164.
10. C. J. Georgiou and D. Anastassiou, An architecture for real-time, single chip, Laplacian image edge detector, in *Proceedings, 2nd Intl. Conf. on Image Processing and its Applications*, June 1986, pp. 107–111.
11. A. Giardano, VLSI-based systolic architecture for fast Gaussian convolution, *Opt. Eng.* January 1987, 63–68.
12. W. E. Grimson and E. C. Hildreth, Comments on digital step edges from zerocrossings of second directional derivative, *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-7** 1985, 121–127.
13. E. C. Hildreth, Detection of intensity changes by computer and biological vision systems, *Comput. Vision Graphics Image Process.* **22**, 1983 1–27.
14. D. Marr and E. Hildreth, Theory of edge detection, *Proc. R. Soc. London B* **207**, 1980, 187–217.
15. D. H. Marimont, A representation for image curves, in *Proceedings Natl. Conf. Artif. Intell., 1984*, pp. 237–242.
16. A. K. Mackworth and F. Mokhtarian, *Scale-Based Description of Planar Curves*, Technical Report 84-1, UBC, 1985.
17. D. Marr, *Vision*, Freeman, San Francisco, 1982.
18. A. Mukherjee, *Introduction to nMos and CMos VLSI Systems Design*, Prentice–Hall, Englewood Cliffs, NJ, 1986.
19. H. K. Nishihara and N. G. Larson, Towards a real time implementation of the Marr and Poggio stereo matcher, in *Proceedings, SPIE Techniques and Applications of Image Understanding, April* 1981, pp. 299–305.
20. (Special issue on scalespace) *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-8**, 1986, 2–43.
21. N. Ranganathan, VLSI implementation of pipeline Wallace multiplier, in preparation, 1987.
22. M. Shah and A. Sood, Fitting primitives in scale-space, in *Proceedings, Soc. Photo-Opt. Instrum. Eng. Technical Symposium*, May 1987.
23. M. Shah, A. Sood, and R. Jain, *Pulse and Staircase Models for Detecting Edges at Multiple Resolution*, University of Michigan Center for Research on Integrated Manufacturing Technical Report RSD-RT-85, July 1985.
24. M. Shah, A. Sood, and R. Jain, Pulse and staircase models for detecting edges at multiresolutions, in *Proceedings, IEEE Workshop on Computer Vision: Representation and Control*, October 1985.
25. M. Shah, A. Sood, and R. Jain, Pulse and staircase edge models, *Computer Vision Graphics Image Process.* **34**, 1986, 321–341.

26. J. Stansfield, *Conclusions from the Commodity Expert Project*, MIT AI Lab Memo 722, 1980.
27. C. Wallace, A suggestion for a fast multiplier, *IEEE Trans. Electron. Comput.* **EC-13**, 1964, 14–17.
28. A. Witkin, Scale-Space Filtering, in *Proceedings Intl. Joint Conf. Artif. Intell., 1983*, pp. 1019–1021.
29. R. A. Young, The Gaussian derivative theory of spatial vision, General Motors Research Laboratories GMR-4920, March 1985.
30. A. L. Yuille and T. Poggio, *Scaling Theorems for Zero Crossings*, MIT AI Memo 722, 1983.
31. A. L. Yuille and T. Poggio, *Fingerprints Theorems for Zero Crossings*, MIT AI Memo 730, 1983.
32. S. Zucker and R. Hummel, Receptive Fields and The Representation of Visual Information, *7th Intl. Conf. Pattern Recognit., 1984*, pp. 515–517.