North-Holland
Microprocessing and Microprogramming 26 (1989) 119–128

Hardware Implementation of a Parallel Noise Clearing Algorithm

M. Atiquzzaman

Department of Electrical Engineering, King Fahd University of Petroleum & Minerals, Dhahran 31261, Saudi Arabia

Dedicated hardware can be used to release the CPU of an image processing system, of routine image pre-processing tasks, thereby speeding up the operational speed of the system. This paper describes the hardware implementation of a one-pass fill, shrink and expand algorithm to clear random noise from an image. The three operations, comprising the algorithm, operate in parallel on a raster-scan image in a pipelined system. Simulation results are given, and the operating speed and cost are compared with those of a multiprocessor system using 8086.

Keywords: Image processing, Noise removal, Image filtering, Fill-shrink-and-expand, Hardware implementation, 2-D convolution, Performance evaluation.

Submitted: 3 June 1988
Revised: 23 October 1988
Accepted: 28 October 1988

1. Introduction

Depending on the type and quality of the visual sensor used for capturing images, there is usually a varying amount of noise prevalent in pictures. The noise makes it difficult for further processing and therefore has to be cleaned up.

The algorithms for noise removal are usually simple; the only constraint being its demand for huge amount of computing power especially for real time operation. For example, the volume of data from a camera operating at 300 frames/sec and digitized to $512 \times 512 \times 8$-bit resolution is 7.5Mb/sec.

Multiprocessor systems are generally employed for real time processing of images. Due to the expensive computing time of multiprocessor systems, simple pre-processing algorithms like noise removal can be done by very cheap hardwired systems, thereby permitting the multiprocessor system to be used for more complicated algorithms.

Different types of noise are encountered in images. Some of them are independent of the picture signal, while others are related in some form to the picture signal. The following are a few types of noise frequently encountered in images:

(a) Channel noise introduced when a picture is transmitted
(b) Coherent noise due to raster-scan nature of television picture, or due to graininess of photographic films
(c) Quantization noise due to digitizing a picture
(d) Salt-and-pepper noise due to thresholding of a noisy picture
(e) Random walk noise.

A detailed treatment of the different types of noise, their sources, and techniques to remove them are given in [1].

The noise-clearing algorithms depend on the type of noise. Both spatial and frequency domain analysis can be used to remove noise [2]. Spatial domain analysis is less computation-intensive and also can be more readily implemented in hardware. Several methods of removal of random noise can be found in the literature. Three methods of removing noise based on its random distribution are discussed in [3]. A method to smooth noise by the successive application of a one-dimensional filter is given in [4]. Smoothing usually results in a blurring of the object. An algorithm based on the neighbour weighting function is given in [5]. Software implementation of these algorithms using a single processor is extremely slow, and hence is not suitable for real time operation. Hardware implementations have not been suggested in any of them.

The implementation of algorithms into dedicated hardware is still in an early stage. Implementations of convolvers for image enhancement have been described in [6,7]. Very few researchers have attempted to implement, in hardware, noise-clearing algorithms.

Salt-and-pepper noise [8] can be effectively removed by a sequence of fill, shrink and expand operations on the image. An implementation has been suggested in [9] for removal of salt-and-pepper noise by using look-up tables.

This paper describes the hardware implementation of the fill, shrink and expand algorithm (shrink and expand operation is often referred to in the literature as erosion and dilation operations) using hardware simpler than in [9]. The hardware is specifically geared towards clearing salt-and-pepper noise in real time from a raster-scan image, as is usually obtained from a conventional video camera. Clean pictures are obtained from the output of the system in real-time, i.e. at the same rate at which the pixels are output from the camera. The three operations (fill, shrink and expand) are carried out in parallel in a pipelined architecture. Due to its high speed and low cost the hardware can be used as a real time pre-processor in image processing systems.

Results of simulation of the architecture using test data are given in section 5. The speed at which the system can operate is also given and compared with the speed achieved by multiprocessor systems using the Intel 8086 processor.

2. Definitions

- A picture can be represented by a two-dimensional array of integers. Elements of the array are called pixels. Let the binary picture G, from which noise is to be removed, be represented by an $N \times N$ array of pixels as shown in Fig. 1. The pixel at the ith row and the jth column of the array is represented by $G_{i,j}$, where $i,j \in [0, N-1]$, $N = 2^n$ and n is an integer.
- A two-dimensional template T will be represented by an $M \times M$ array where $M < < N$ and M is an odd integer. The elements of the template are referenced by $T_{k,l}$ where $k,l \in [0, M-1]$. The template is referenced by its centre element $T_{m,m}$, where $m = (M-1)/2$. When it is said that the template T is over pixel $G_{i,j}$ it is meant that $T_{m,m}$ is over the pixel $G_{i,j}$.
- In a raster-scan image the pixels are output from the camera serially one after another from left to right and top of bottom of the image. A pixel $G_{i,j}$ is said to be preceding $G_{m,n}$ if the pixel $G_{i,j}$ is output from the camera before $G_{m,n}$ and $G_{i,j}$ is said to be succeeding $G_{m,n}$ if $G_{i,j}$ is output from the camera after $G_{m,n}$.
- Neighbouring pixels of $G_{i,j}$ are those that are covered by the template T when $T_{m,m}$ is placed over $G_{i,j}$, i.e. the elements $G_{p,q}$ where $p \in [i-m, i+m]$, $q \in [j-m, j+m]$, and $G_{p,q} \neq G_{i,j}$.
- One pixel-unit time is defined to be the time required to output one pixel. One pixel-unit time is equal to

  $\dfrac{1}{F \times H \times V}$ where F = Frame rate of the camera, H, V = horizontal and vertical resolution of the digitized picture.

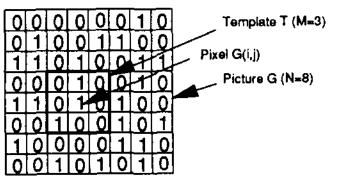


Fig. 1. Picture G and template T.

3. Algorithms

This section presents the fill-shrink-and-expand algorithm for removal of salt-and-pepper noise from a binary picture. Blobs of 1's are considered as objects while pixels with 0's are considered as background. The algorithm consists of three steps:

- *Fill* – filling isolated holes in an object
- *Shrink* – removing isolated noise from the background by shrinking the objects in size
- *Expand* – expanding the objects to recover the original size while isolated noise in the background is removed.

3.1 The Fill Algorithm

A binary picture can be obtained by thresholding a grey-level picture [10]. In such a picture the objects of interest are shown as groups of pixels of value '1' against a background of pixels of value '0'. During the thresholding operation some pixels within the object may show up as isolated 0's within the object. The fill operation is used to remove the isolated 0's from within the object. The isolated 0's are detected and then filled with 1's. The template T is successively moved over the different pixels of the picture from left to right and top to bottom. At any instant let the template be over pixel $G_{i,j}$, i.e. $T_{m,m}$ is over pixel $G_{i,j}$. The fill algorithm can be stated as follows.

Algorithm 1: If the pixels $G_{p,q}$ under the template T are equal to 1, i.e. $G_{p,q} = 1$, where $p \in [i - m, i + m]$, $q \in [j - m, j + m]$ and $G_{i,j} \neq G_{p,q}$ then replace $G_{i,j}$ by 1, else leave $G_{i,j}$ as it is.

This corresponds to finding the AND of all the pixels $G_{p,q}$.

3.2 The Shrink Algorithm

An effect opposite to what has been stated in section 3.1 also occurs during the thresholding of a picture. In a binary picture there is usually some noise in the form of isolated 1's scattered in the background. The shrink operation removes these isolated 1's which do not belong to any object.

The template T is successively moved over the picture from left to right and top to bottom. Let the template be over pixel $G_{i,j}$ at any instant. The shrink algorithm can be stated as follows:

Algorithm 2: If $G_{i,j} = 1$ and not all $G_{p,q} = 1$ for $p \in [i - m, i + m]$, $q \in [j - m, j + m]$, then $G_{i,j} = 0$.

The effect of the shrink operation on an object is to replace the border pixels of an object by 0's thereby shrinking the object size by one pixel from all of its sides. The effect of the shrink operation on an isolated '1' (or a group of 1's of width less than M pixels) is to replace it (or the group) by a '0' (or a group of 0's). The shrinked picture is subsequently expanded as will be described in section 3.3 in order to recover the original size of the object, but the isolated 1's (or the group of 1's) which are considered as noise do not reappear.

3.3 The expand Algorithm

Due to the shrink operation, the size of an object is reduced. The expand algorithm restores the original size of the object, but the noise (isolated 1's) which were removed by the previous shrink operation are not restored.

The template T is moved over the picture G from left to right and from top to bottom. At any particular instant let $T_{m,m}$ be over the pixel $G_{i,j}$. The expand algorithm can be stated as follows:

Algorithm 3: If $G_{i,j} = 1$, then $G_{p,q} = 1$, where $p \in [i - m, i + m]$, $q \in [j - m, j + m]$.

The elements $G_{p,q}$ are to be modified in such a manner that the modified values are not considered when the template later moves over any of the elements in $G_{p,q}$. When the template moves over any of the elements of $G_{p,q}$, the original values of $G_{p,q}$ should be considered but not the modified values. This creates a problem in real time operation on a raster-scan image. The problem is explained in detail later.

4. Hardware Implementations

The fill, shrink and expand operations are performed in parallel on a raster scan image. The operations are successively applied on the stream of pixels emerging from a camera. The three sections performing the three operations are connected in a pipeline as shown in Fig. 2. Pixels are pumped into one end of the system and clean pictures are obtained from the other end.

In this paper, very simple hardware like flip flops, shift registers, and gates has been employed to implement the algorithms in the different stages. This

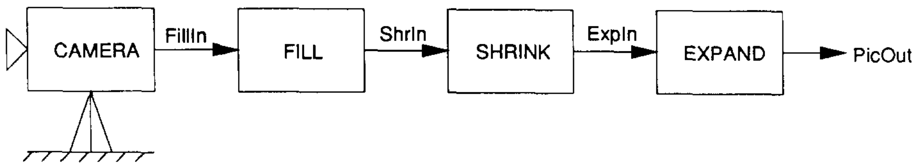


Fig. 2. Block diagram of the system.

1	1	1
1	X	1
1	1	1

X = don't care

Fig. 3. Template for the fill operation ($M = 3$).

has resulted in a very simple, fast and inexpensive piece of hardware which can be used to reduce the computing load on the CPU of an image processing system. The system described here demonstrates the fact that low-level image processing operations can usually be implemented by a very simple form of hardware.

There is a time lag between the input and the output of pixels in each of the stages. The time lag in the different stages depends on the size of the picture and the template, and is discussed in the following sections. The total time delay in the system is the sum of the time delays in the different stages.

4.1 *Fill Operation*

The fill operation, as stated in section 3.1, consists of replacing a pixel $G_{i,j}$ having a value of '0' by '1' if all the surrounding pixels are 1's. The template for the fill operation is shown in *Fig. 3* for $M = 3$. *Fig. 4* shows a serpentine memory structure [11,12] to register/align the pixels arriving from a raster-scan camera. The memory structure comprises of the flip flops F1, F2,.... F8, F9 and the shift registers SR1 and SR2 each of length $N - M$. $M = 3$ has been assumed corresponding to the template in *Fig. 3*. A line of pixels are stored in F1, F2, F3, and SR1, while pixels of the immediately succeeding line are stored in F4, F5, F6, and SR2.

Pixels from the camera enter the system through the 'FillIn' line. The outputs of F1, F2, F3,... F9 are ANDed and the result ORed with F5 as shown in *Fig. 5* which is a hardware implementation of the template matching operation for fill algorithm. The effect of pumping the pixels serially through 'FillIn' and shifting the pixels through the flip flops and shift registers is, in effect, like moving the template T over the picture G from left to right and top to bottom.

The system is synchronous; a master clock is used throughout the system. The shifting of pixels in the flip flops & the shift registers and the input & output of the pixels occur under the control of the master clock.

Every time a pixel is input through 'FillIn', a pixel is output from the line 'ShrIn' which is connected to the input line of the shrink block. There is a time lag between the input and the output of the fill

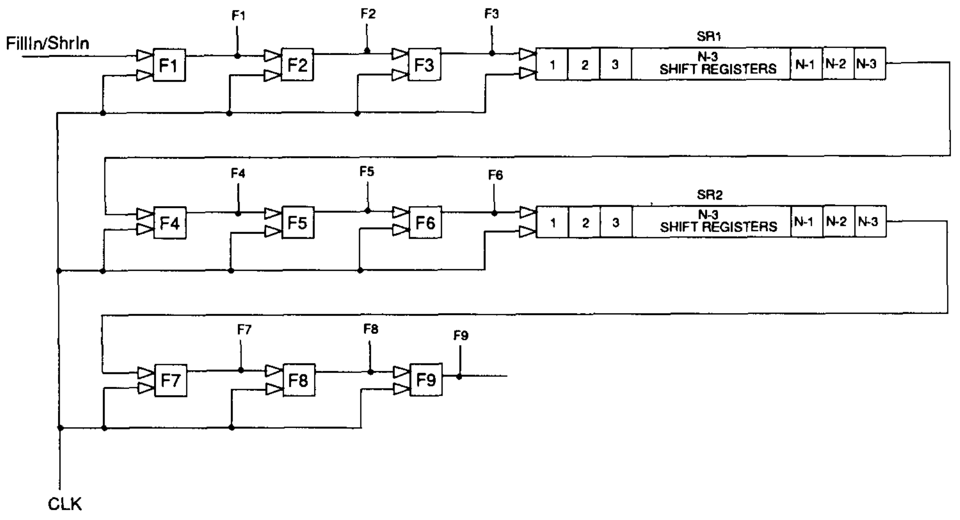


Fig. 4. Arrangement of shift registers and flip flops for proper alignment of pixels for fill and shrink operation.

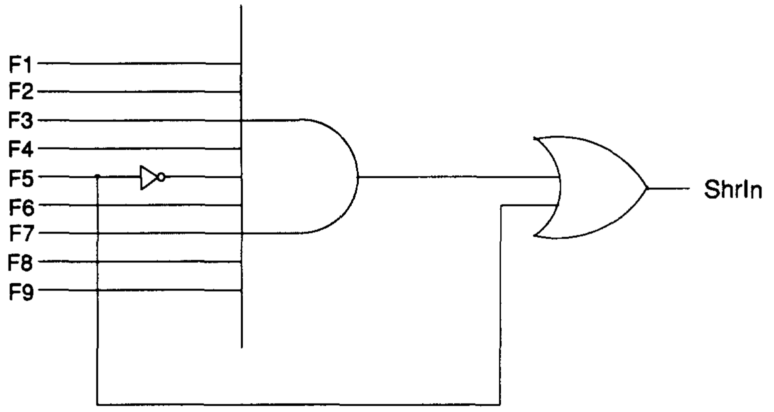


Fig. 5. Convolution hardware for the fill operation.

stage. The delay in pixel units is $(N+1)(M-1)/2$. This means that, in *Fig. 4*, a pixel going through the 'FillIn' line, after modification, emerges out of the 'ShrIn' line after a time equal to output $(N+1)$ pixels from the camera.

Texas Instruments 74164 is an 8-bit shift register, and the values in the individual flip flops are externally available. The 74164 has been used to implement the flip flops F1, F2, F3,.... F9 and the shift registers SR1 and SR2. The maximum clock frequency of the 74164 is 36 MHz, which sets the maximum speed of operation of the fill stage.

4.2 Shrink Operation

Image thinning operation is in fact the shrinking operation described in section 3.2, with the difference that in the thinning operation the picture is successively shrinked until the objects in the image are skeletonized. The thinning operation is often applied to images for character recognition purposes, and for reducing the amount of data to be handled.

A one-pass thinning algorithm has been described in [11]. The hardware implementation for the thinning operation discussed in [11] consists of flip flops and shift registers. Similar hardware configuration is used in this paper for the shrink operation. *Fig. 4* shows the connection of the shift registers to properly align the data in the shrink stage for performing the shrink operation, and *Fig. 6* shows the hardware realization for the thinning algorithm. The inputs to *Fig. 6* are taken from *Fig. 4* as shown. In this case the input to *Fig. 4* will be ShrIn. There is a time lag of $(N+1)(M-1)/2$ pixel units between the input and the output. The hardware has been implemented using the 8-bit shift register (74164) described in the previous section.

Different types of thinning algorithms using 3 × 3 templates can be found in the literature [13,14]. Most of them operate on the picture in parallel, and also lend themselves to implementations of the type described in this paper.

4.3 Expand Operation

It has been stated in section 3.3 that depending on the value of a pixel, the value of the neighbouring pixels may have to be changed in an expand operation. Consider the template for $M = 3$ shown in *Fig. 7* which is moved successively over the pixels of G from left to right and top to bottom. Let the template be over $G_{i,j}$. If the value of the pixel $G_{i,j}$ is 1

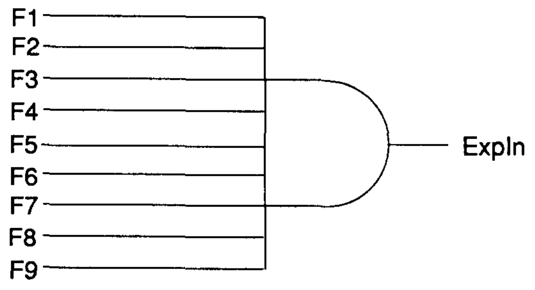


Fig. 6. Convolution hardware for the shrink operation.

X	X	X
X	1	X
X	X	X

X = don't care

Fig. 7. Template for the expand operation.

then all the neighbours of $G_{i,j}$ have to be changed to 1's, irrespective of what the previous values were. The elements that are to be changed are given later. The whole system operates synchronously in real time on a raster-scan image as stated in section 4.1.

The hardware implementation of the expand operation consists of two shift registers SR3 and SR4, each of length $(M-1)(N+1)/2 = \mathrm{m(N+1)}$ where $m(M-1)/2$, as shown in *Fig. 8* for $M = 3$ and $N = 16$. The outputs of the flip flops in shift register 4 (SR4) are available to the external circuitry, while the flip flops in shift register 3 (SR3) have separate external reset inputs. Texas Instrument's 5-bit shift register (74LS96) and the 8-bit parallel out shift register (74164) satisfy the above conditions, and therefore have been used for SR1 and SR2 respectively. The 74LS96 and the 74164 have maximum clock speed of 25 MHz and 36 MHz respectively.

It has been stated in section 3.3 that the new value of a pixel should not be determined depending on the modified values of its or its neighbouring pixels. The unmodified values of the pixel and its neighbouring pixels should be used for calculating the new value of a pixel. Therefore, any pixel succeeding $G_{i,j}$ cannot be replaced immediately. The template must have moved over a pixel before it can be replaced so that the neighbouring pixels are replaced depending on the original value but not the modified value. This is illustrated in *Fig. 9*. *Fig. 9a* shows an object covering an area of 6 pixels. After the expand operation the resulting picture is shown in *Fig. 9b*, where the objects have expanded by one pixel on all its sides.

Fig. 10 shows the expand operation when the neighbouring pixels are replaced immediately. The template T is over pixel $G_{i,j}$ in *Fig. 10a*. All the neighbouring pixels of $G_{i,j}$ are replaced by 1's. When the template moves over $G_{i+1,j}$, the pixel $G_{i+2,j}$ is replaced by 1 as shown in *Fig. 10b*. $G_{i+2,j}$ should have been 0 as is evident from *Fig. 9b*. If the neighbouring pixels are replaced immediately, the object will keep on growing in size.

In order to avoid this, neighbouring pixels, for example $G_{i+1,j}$ should not be immediately replaced, but the decision regarding the value of $G_{i+1,j}$ (when the template is over $G_{i,j}$) should be stored. The stored value of $G_{i+1,j}$ should be used to replace $G_{i+1,j}$ after

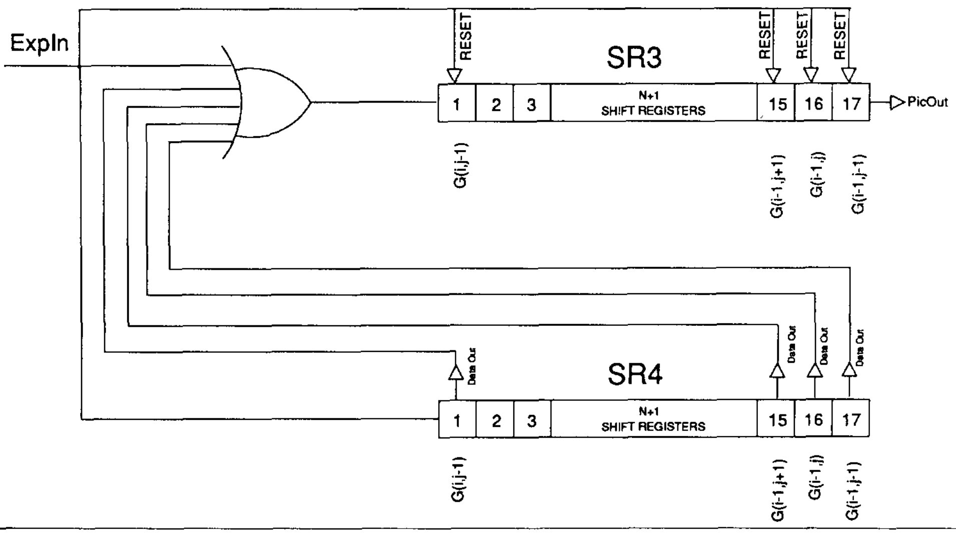


Fig. 8. Hardware for the expand operation, $M=3, N=16$.

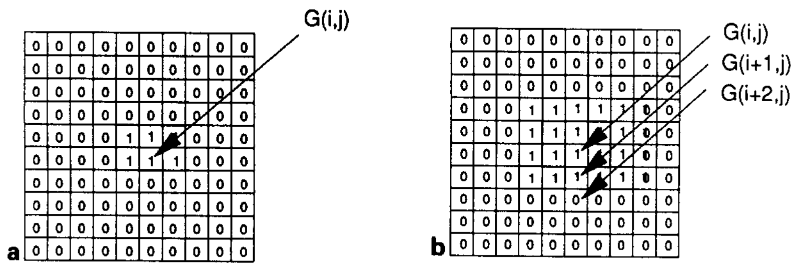


Fig. 9. Expansion of an object.

(1) the template has moved over $G_{i+1,j}$,
(2) the original value of $G_{i+1,j}$ has been considered in determining the neighbouring values and
(3) the template has then moved over to $G_{i+1,j+1}$.
In fact, for $M = 3$, the value of $G_{i+1,j}$ depends not only on the value of $G_{i,j}$ but on the values of $G_{i,j-1}$, $G_{i,j+1}$ and $G_{i+1,j-1}$ which are preceding it, and also on the succeeding values as will be explained in the next few paragraphs.

This problem arises only for real time operations on a raster-scan image where the pixels of a picture are continuously pumped in through one end and pushed out through the other end. It does not arise when a full picture is stored in a frame buffer and then operated on by a single microprocessor. All the pixel values required to decide the value of a pixel are already stored.

Shift register SR4 stores values of pixels which are required for determining the values of succeeding pixels. When T is over $G_{i,j}$ a copy of $G_{i,j}$ is stored in SR4 while $G_{i,j}$ is replaced by the logical OR of the values

$$PV = \left[\begin{array}{cccccccc} G_{i,j-m} & G_{i,j-m-1} & \cdots & G_{i,j-1} & & & & \\ G_{i-1,j-m} & G_{i-1,j-m-1} & \cdots & G_{i-1,j-1} & G_{i-1,j} & \cdots & G_{i-1,j+m-1} & G_{i-1,j+m} \\ \cdot & \cdot & & \cdot & \cdot & & \cdot & \cdot \\ \cdot & \cdot & & \cdot & \cdot & & \cdot & \cdot \\ \cdot & \cdot & & \cdot & \cdot & & \cdot & \cdot \\ G_{i-m,j-m} & G_{i-m,j-m-1} & \cdots & G_{i-m,j-1} & G_{i-m,j} & \cdots & G_{i-m,j+m-1} & G_{i-m,j+m} \end{array}\right].$$

The modified value of $G_{i,j}$ (let us say $G^m{}_{i,j}$) depends on the values of all those pixels which are its preceding neighbours, and as stated above the values of $G_{i,j}$ could not changed unless the template has moved over $G_{i,j}$. These preceding neighbouring pixels were stored in SR4 and the values are therefore obtained from the following-numbered flip flops of SR4.

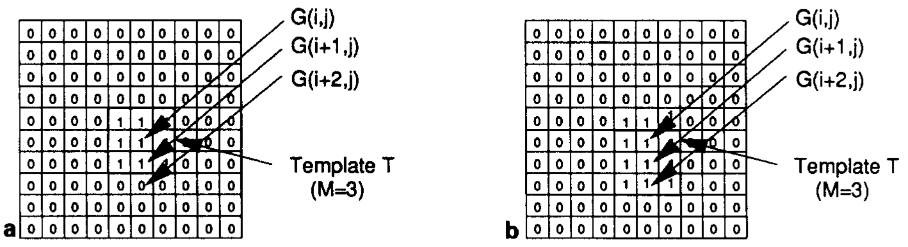


Fig. 10. Expansion of an object by replacing neighbouring pixels immediately.

When the template is over $G_{i,j}$ three actions are to be taken:

(i) The value of $G_{i,j}$ has to be modified depending on the values of the set [PV], and the modified value $G^m{}_{i,j}$ has to be stored in SR3 for subsequent modification depending on the values of the succeeding neighbouring pixels.

(ii) The value of $G_{i,j}$ has to be stored in SR4. $G_{i,j}$ is required for modification of succeeding neighbours which are not available now due to the raster-scan nature of images.

(iii) The values of $G_{i,j}$ should be used to modify the values of the preceding neighbouring pixels, i.e. the pixels in [PV]. The preceding neighbouring pixels are stored in SR3.

The modified values of the pixels of [PV], let us say $[\mathrm{PV}]^m$, are actually stored in SR3 as stated in step 1. So if $G_{i,j} = 1$, then the proper elements in SR3 are reset to 1. The elements of $[\mathrm{PV}]^m$ are stored in the following numbered flip flops of SR1.

$$\begin{array}{ccccccccc}
1 & \ldots & m-2 & m-1 & m & & & & \\
N-m & \ldots & N-2 & N-1 & N & N+1 & N+2 & \ldots & N+m \\
2N-m & \ldots & 2N-2 & 2N-1 & 2N & 2N+1 & 2N+2 & \ldots & 2N+m \\
\cdot & & \cdot & \cdot & \cdot & \cdot & \cdot & & \cdot \\
\cdot & & \cdot & \cdot & \cdot & \cdot & \cdot & & \cdot \\
\cdot & & \cdot & \cdot & \cdot & \cdot & \cdot & & \cdot \\
mN-m & \ldots & mN-2 & mN-1 & mN & mN+1 & mN+2 & \ldots & mN+m
\end{array}$$

The reset lines of the above numbered flip flops are therefore connected to the line having the value of $G_{i,j}$ ('ExpIn' in *Fig. 8*).

There is a time lag between the input and the output of a picture. This arises due to the fact that the modified value of a pixel depends on the value of its succeeding neighbours. Therefore, a pixel cannot be output until its succeeding neighbours have been received from the raster-scan camera. The bigger the size of the picture $\boldsymbol{G}$ and the template $\boldsymbol{T}$, i.e. the higher the value of N and M, the more the time lag. The time lag (in pixel units) is the length of the shift register SR3, i.e. $(M-1)(N+1)/2$.

5. Results

Extensive simulations have been performed to test the functionality of the system described in the previous sections. The simulations were carried out using Pascal. The shift registers and flip flops have been simulated by Pascal arrays, and the wires connecting the different components by global variables. This section presents the results of simulation, the operating speed achieved by this system, and an approximate cost.

Fig. 11a shows the input binary picture $\boldsymbol{G}(N = 64)$ with apparently three objects of varying sizes. A pixel having a value of 1 or 0 is considered as object or background pixel respectively. Object 3 (representing the letter 'A') is the actual object and the others are noise arising during digitization, thresholding etc. There is some noise (one-point holes) even within object 3. The picture is input to the system for the purpose of clearing up noise. Templates of size $M = 3$ have been used in all the three stages viz. fill, shrink and expand.

Fig. 11b shows the output of the fill stage. The one-point holes in object 3 (*Fig. 11a*) have been replaced by 1's. Other parts of the picture remain unchanged.

The filled picture of *Fig. 11b* is input to the shrink stage and the output of the stage is shown in *Fig. 11c*. Notice that objects 1 and 2 which were actually noise have disappeared due to the shrinking operation. Also object 3 has reduced in size.

The shrinked picture is then input to the expand stage and output is shown in *Fig. 11d*. It is seen that object 3 has regained its original size, and the noises have been eliminated.

The whole system operates at a speed equal to the speed of the slowest component of the system. The OR and AND gates can operate at much higher speed than the shift registers. The slowest shift register used in the system is the 74LS96 having a clock speed of 25 MHz. Therefore, the maximum operating speed of the system is 25 MHz. This means that pixels can be clocked into the system at a maximum rate of 25×10^6 pixels/sec.

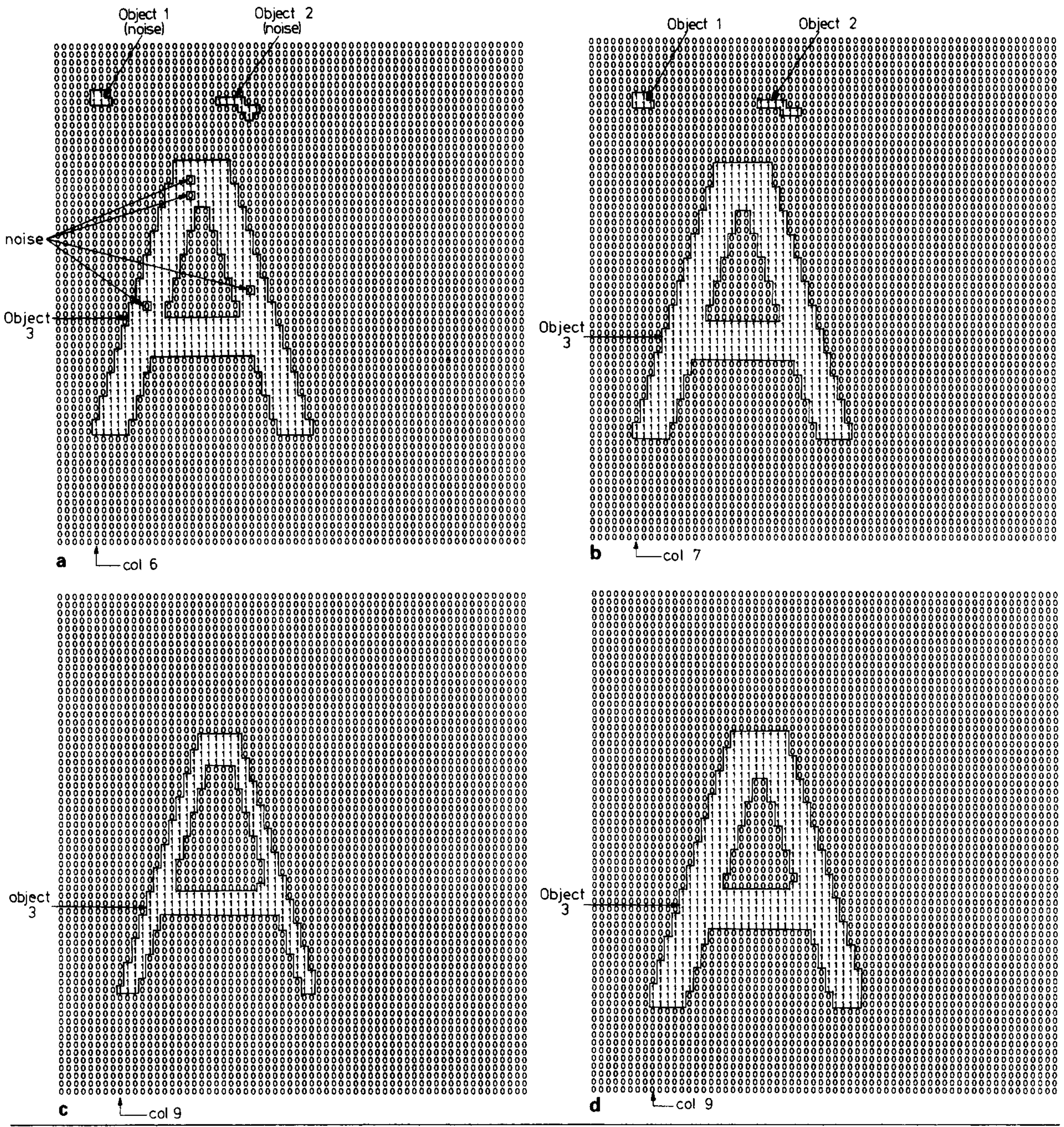


Fig. 11. Simulation results. (a) Original picture. (b) Picture after fill operation. (c) Picture after shrink operation. (d) Picture after expand operation.

A conventional TV camera outputs raster-scan images at frame rates of 30 frames/sec. If the picture is digitized at a resolution of 512 × 512 pixels, the maximum rate of arrival of pixels is 7.5×10^6 pixels/sec as stated in section 1. Therefore, the maximum speed of the system of 25×10^6 pixels/sec is more than adequate for real time operation on pictures output from a conventional TV camera. The overall component cost of the system has been calculated to be only $5.

Dedicated hardware has made possible operation at such a high speed at low cost. Contrary to this, a multiprocessor system employing 512 Intel 8086 processors and costing $134,000 can perform the fill, shrink and expand operation at a maximum speed of 2.9×10^6 pixels/sec [15].

6. Conclusions

A hardware implementation of the fill, shrink and expand algorithm to eliminate random noise in digitized pictures has been described. The three operations are carried out in parallel on a raster-scan image from a conventional TV camera. The implementation consists of gates and shift registers, and permit operations at a speed of more than three times that of real time.

Simple and cheap hardware like the one described in this paper can be used for pre-processing of pictures in a real time image processing system. This will relieve the CPU (of the system) of routine low level pre-processing tasks allowing it (the CPU) to concentrate on mid and high level processing of the picture.

Results of simulation of the system using test pictures containing random noise have been presented. A 3×3 template has been assumed for the simulations. The size of the template depends on the size of the random noise. If the random noise is found to occur in clusters (which is not usually the case), then the size of the template can be increased to 5×5 or even more by merely adding more shift registers in the system.

References

[1] A. Rosenfeld and A.C. Kak, *Digital Picture Processing*, Vol 1 (London: Academic Press, 1982).

[2] G.A. Mastin, Adaptive filters for digital image noise smoothing: An evaluation, *Computer, Vision, Graphics and Image Processing*, 31, 1, (July 1985) 103–121.

[3] L. Norton-Wayne, On the removal of random noise from two-dimensional binary patterns: A comparison study, *5th Int. Conf. on Pattern Recognition*, Florida (Dec 1-4, 1980) 1180–1183.

[4] P.M. Narendra, A separable median filter for image noise smoothing, *IEEE Conf. on Pattern Recognition and Image Processing*, Chicago, Illinois, (May 31-June 2, 1978) 137–141.

[5] K.J. Overton and T.E. Weymouth, A noise reducing preprocessing algorithm, *IEEE Conf. on Pattern Recognition and Image Processing*, Chicago, Illinois, (May 31-June 2, 1978) 498–507.

[6] Z. Orbach, Real-time video image enhancement using hardware convolver, *IEEE Conf. on Pattern Recognition and Image Processing*, Las Vegas, Nevada (June 14-17, 1982) 390–392.

[7] T. Mimaroglu, A high-speed two-dimensional hardware convolver for image processing, *IEEE Conf. on Pattern Recognition and Image Processing*, Las Vegas, Nevada (June 14-17, 1982) 386–388.

[8] A. Wood, The interaction between hardware, software and algorithms, in *Languages and Architectures for Image Processing*, M.J.B. Duff and S. Levialdi (eds) (London, Academic Press, 1981).

[9] F.A. Gerritsen and P.W. Verbeek, Implementation of cellular-logic operators using 3•3 convolution and table lookup hardware, *Computer, Vision, Graphics and Image Processing*, 27, (July 1984) 115–123.

[10] R.C. Gonzalez and P. Wintz, *Digital Image Processing*, (Massachussetts, Addison-Wesley 1977).

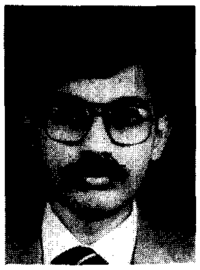
[11] R.T. Chin and H.K. Wan, A one-pass thinning algorithm and its parallel implementation, *Computer, Vision, Graphics and Image Processing*, 40, 1 (Oct 1987) 30–40.

[12] S. Grinaker, Real-time processing of raster-scan images, in *Image Sequence Processing and Dynamic Scene Analysis*, T.S. Huang (ed) (Springer Verlag, 1983) 549–561.

[13] R. Stefanelli and A. Rosenfeld, Some parallel thinning algorithms for digital pictures, *J. Association for Computing Machinery*, 18, 2 (April 1971) 255–264.

[14] H.E. Lu and P.S.P. Wang, An improved fast thinning algorithm for digital pattern, *IEEE Conference on Computer Vision and Pattern Recognition*, San Fransisco, California (June 19-23, 1985) 364–367.

[15] M. Atiquzzaman, Algorithms and architectures for automatic traffic analysis, *Ph.D. Thesis, Department of Electrical Engineering and Electronics*, University of Manchester Institute of Science & Technology, England (Dec 1986).



M. Atiquzzaman was born in Dhaka, Bangladesh, on June 6, 1959. He received the B.Sc. (Engg) degree in electrical and electronic engineering from Bangladesh University of Engineering and Technology, Dhaka, Bangladesh in 1982, and the M.Sc. and Ph.D. degrees both in electrical engineering and electronics from University of Manchester Institute of Science and Technology, Manchester, England, in 1984 and 1987 respectively. He joined the King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, in 1987 where he is presently an Assistant Professor. His current research interests are image processing and pattern recognition, computer architectures, multiprocessor systems, and parallel processing.