COMPUTER GRAPHICS AND IMAGE PROCESSING 17, 71–78 (1981)

NOTE

Getting the Median Faster

PER-ERIK DANIELSSON*

IBM Research Division, 5600 Cottle Road, San Jose, California 95193

Received July 11, 1980; revised September 22, 1980

Median filters and rank-order filters are generally considered to be more time consuming than, e.g., averaging filters. With the new algorithm presented in this report the opposite turns out to be true. The median (or any other rank-order value) is obtained by examining the bits of the arguments columnwise starting with the most significant bits. Hardware implementations suitable for parallel processing are presented.

1. INTRODUCTION

Median filters (or the more general rank-order filters) have been proved to be very powerful in image processing [1]. However, established computational algorithms of type sorting are rather time consuming. Furthermore, most time-efficient sorting mechanisms require intermediate data structures in the form of pointers and the like which are not very suitable for hardware implementations.

Nevertheless, one straightforward hardware device implemented in CCD has been suggested by Nudd *et al.* [2]. It utilizes $n(n-1)/2$ comparators and multiplexors to deliver a completely sorted array of n output values. For image processing this is not as bad as it might seem since it has been shown by Narendra [3] that median filters behave reasonably well even if they are separated into a succession of two one-dimensional median filters orthogonally oriented.

To perform sorting of a set of arguments is actually a large overkill if the goal is to find the median (or a percentile) value. Huang [4] has suggested a running median filter algorithm that utilizes a continuously updated histogram as the main instrument. The histogram needs only to be partially updated when we move from one neighborhood to the next. Hence, by observing how many pixels fall above and below the previous median value we can rather simply adjust this value to the presently correct median value by stepping upward or downward in the histogram. This algorithm requires a fast RAM for the histogram. The time for the stepping procedure is data dependent. Thus in a parallel processing scheme one has to allow for a worst case number of steps equal to the number of buckets in the histogram. Huang's algorithm belongs to the large group of algorithms called bucket sort. In the next section we will study another form of bucket sorting mechanism called lexicographic sort.

2. LEXICOGRAPHIC SORT METHODS

The well-known lexicographic sorting methods have been used ever since the time of punched card machines. An overview can be found, for instance, in Aho *et al.* [5].

*On leave of absence from: Department of Electrical Engineering, Linkoeping University, S-58183 Linkoeping, Sweden.

0146-664X/81/090071-08$02.00/0

i	B_1	B_2	B_3	B_4	
1	0	0	1	1	A_1
2	1	1	0	1	A_2
3	1	1	0	0	A_3
4	0	1	1	0	A_4
5	1	0	0	0	A_5
6	0	1	1	0	A_6
7	1	1	1	0	A_7

FIGURE 1

The arguments $A_1, A_2, \ldots, A_n$ to be sorted are considered as an $n \times k$ array, each A_i being a k-tuple

$$A_i = (a_{i1}, a_{i2}, \ldots, a_{ik})$$

of integers. In our case A_i are the pixel values and a_{ij} are binary integers. As our running example, let us take the seven four-bit vectors of Fig. 1.

Lexicographic sort means that we arrange for two buckets $Q[0]$ and $Q[1]$. These are stored with a sorted sequence of the i-numbers 1 to n as shown in Fig. 2.

In the first step the sequences are ordered according to the bit vector B_4 in Fig. 1, in the next according to B_3 *and* the existing sequence from step 1, etc.

The algorithm has the advantage of terminating in a fixed number of steps $= k$. However, the implementation in fast hardware for picture processing applications is far from obvious. One attempt is shown in Fig. 3. The two FIFOs hold the two buckets shown with contents after step 1. The bit-vector B_3 is used to steer the numbers 3, 4, 5, 6, 7, 1, 2 to the next bucket configuration. Varying length of the FIFOs can be avoided if we allow for $2n$ cycles at each step.

After completion of the sorting procedure one has to circulate the buffers one extra step to bring out the number n of the median or any other rank-order. Finally this number has to be used as an *address* to the arguments to get the final result. In parallel image processing we would prefer this addressing to be independent of the original access mechanisms of the pixel arguments (all data-dependent addressing should be avoided) which means that we have to store the pixel values in an extra buffer in a predetermined order.

3. SUCCESSIVE APPROXIMATION IDEAS

Danielsson [6] suggested that the median or any other rank-order value could be found by collecting a *set* of histograms for the arguments:

$$\text{hist } 1, \text{hist } 2, \ldots, \text{hist } k,$$

	step 1	step 2	step 3	step 4
Q[0]	3,4,5,6,7	3,5,2	5,1	1,4,6
Q[1]	1,2	4,6,7,1	3,2,4,6,7	5,3,2,7

Answer: 1,4,6,5,3,2,7

FIGURE 2

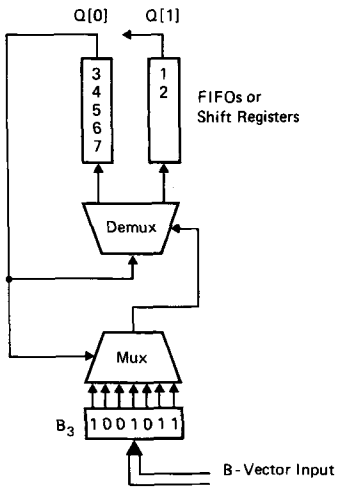

FIGURE 3

where hist 1 contains only 2 bins according to the most significant bit of the argument, hist 2 contains 4 bins due to the two most significant bits, etc. After histogram collection, this information could be addressed from hist 1 to hist k with successively better approximation to find the bin with the median value. Only one bin of hist 1 has to be visited and after checking this value we known which bin in hist 2 to visit for further refinement, etc. The drawback of this method is the rather cumbersome histogram collection process.

Kruse [7] suggested that instead of arranging for histograms one should use the arguments repeatedly k times, comparing them to a successively refined median. After each such read-out we know whether the assigned median is too high or too low and can adjust it appropriately. Unfortunately each argument has to be fetched k times.

4. THE NEW ALGORITHM

Consider again the seven arguments of Fig. 1 and assume that we want to find the fifth A_i in size. By inspecting the vector B_1 of most significant bits we can see that n_0, the number of arguments with leading zeros, is 3, which is less than 5, meaning that the set $d_0 = \{A_1, A_4, A_6\}$ can be discarded from further discussion. This is the general idea behind the new algorithm. The following parameters and input arguments are involved:

T = rank-order number of the wanted pixel value,
$\{A_1, A_2, \ldots, A_n\}$ = set of input arguments,
$\{B_1, B_2, \ldots, B_k\}$ = column bit vectors of the A_i's,
B_0 = initial column vector,
N = accumulated sum of arguments considered to be converging toward T,
$S_0, S_1, \ldots, S_k$ = bit vectors indicating the set of discarded arguments in different steps of the algorithm.

	B_0	S_0	B_1	S_1	B_2	S_2	B_3	S_3	B_4	S_4
	1	1	0	0		0		0		0
	1	1	1		1		0		1	0
	1	1	1		1		0		0	
	1	1	0	0		0		0		0
	1	1	1		0	0		0		0
	1	1	0	0		0		0		0
	1	1	1		1		1	0		0
N=	0		3		4		6		5	

FIG. 4. Note that for the sake of readability, the 1's of S_1 to S_4 were omitted as well as discarded parts of B_1 to B_4.

See Fig. 4. The counter N is initially set to 0 while B_0 and S_0 are filled with 1's.

In the first step, already described, three 0's are found in the first vector B_1. The set d_0 of discarded arguments are marked by 0's in the vector S_1.

The next step in the algorithm is to increase the count N from n_0 to $n_0 + n_{10}$ by accumulating all 0's in B_2, not considering those arguments with a 0 in the S_1-vector. This accumulates to 4 which tells us that our wanted number has 11 as leading bits. All other arguments receive 0's in the S-vector as can be seen from S_2 in Fig. 4.

In the third step, $N = n_0 + n_{10} + n_{110} = 6 \geq 5$, which means that the leading bits are 110 and all other arguments can be zeroed in the S-vector.

Finally, in the fourth step $N = n_0 + n_{10} + n_{110} - n_{1101} = 5 \geq T$ which gives us the final result 1100.

For those from Missouri who still have some doubts about the algorithm we will give an explanation in the form of a binary decision tree. See Fig. 5. The algorithm is "zooming in" in a binary search which can be illustrated as in Fig. 6a for the example above.

As another example, let the set of arguments consist of one argument 1111 and all the other six = 0000. As can be seen from Fig. 6b the final result will be correct.

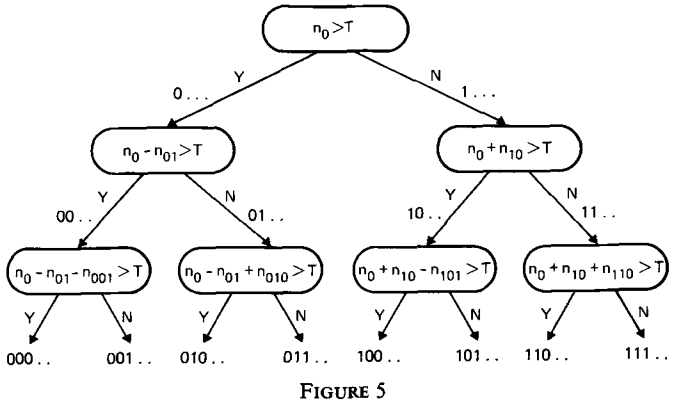

FIGURE 5

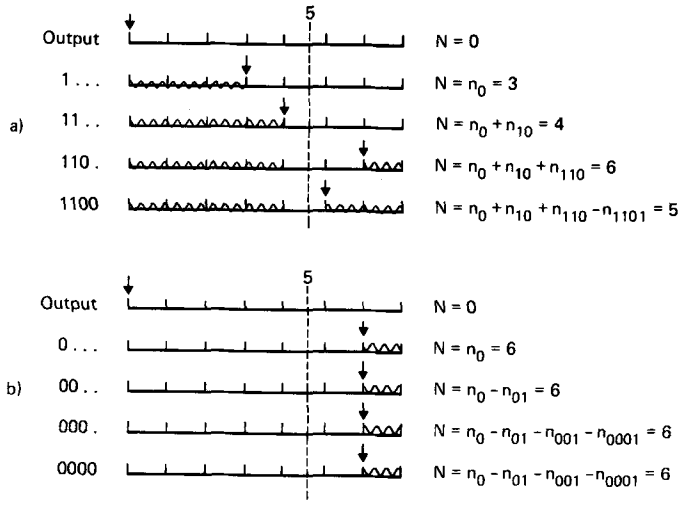

FIGURE 6

5. HARDWARE IMPLEMENTATIONS

The novel algorithm can very easily be implemented in hardware as shown by Fig. 7. The input data is stored in shift register B, bit-serial columnwise, so that the incoming bit can be compared to the previous bit of the same argument (Exclusive OR-gate). Only if the present bit and the previous one are different should the N-counter be incremented or decremented as shown by the decision tree, Fig. 6, where all the n-entities have index numbers $n_{..10}$ or $n_{..01}$. The S-vector is gradually filled with more and more 0's, the conditions for a new 0-bit being that $(N < T) = 1$ and the B-bit $= 0$ or that $(N < T) = 0$ and the B-bit $= 1$. This logic is implemented by the leftmost Exclusive OR-gate in Fig. 7.

With the implementation of Fig. 7, one median is computed in $O(k \cdot n)$ time. This is the same time complexity as in the scheme suggested by Kruse [7]. However, in Fig. 7 we are only using bit-serial input and output.

In Fig. 8 is shown a hardware solution for computing the median in $O(k)$ time. As before we assume that there are n input arguments $A_1, A_2, \ldots, A_n$ constituting the input data. To make things easy we have assumed that the input data is a digitized one-dimensional signal. In this case we can feed the input data bit serially into a set of n shift registers having n outputs. In sequence, the k column vectors of the arguments will appear at these outputs. Conversion from a possible parallel access of input data to the serial input of this device is trivial. Also, if data is two-dimensional this scheme is easily extended to a multitude of separate chains of shift registers. For an $n \times m$ window we then need m chains of n shift registers. In this case all the $m \times n$ binary outputs of the shift register package are of course channelled to a single B-register as in Fig. 8.

The logic of Fig. 7 is duplicated n times in Fig. 8. The "count" outputs are summed up in a Count network and accumulated in an accumulator. Note that the width of the data paths in this part is only $O(\log n)$ bits.

It is possible to fold out the algorithm even more as shown by Fig. 9. With this hardware the median is computed in $O(1)$ time. The k-bit argument enters at the

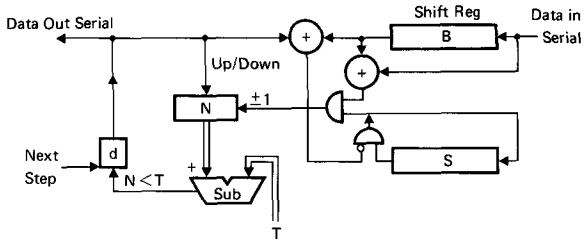

FIGURE 7

data input and is pushed down step by step in the k shift registers $B_1, B_2, \ldots, B_k$. *At each step* one new median value is computed with the logic networks which are nothing but the iterative version of Fig. 8. Initial values are supplied from the top. The comparison curcuit of Fig. 8 is replaced by simply detecting the sign after the subtraction/addition operations.

To be really implementable, the scheme of Fig. 9 should be properly pipe-lined with delay elements and intermediate registers that intersect the vertical data flow. However, this is a trivial matter that has been left out for the sake of brevity.

Finally, it is interesting to discuss the *space complexity* of the hardware in Fig. 9 as a function of the size parameters n and k. The B-registers and the "logic" units all have the obvious space complexity $O(n)$ equal to the width of the data paths. The count network has an input width of n and an output width of $\log n$ which could possibly imply a space complexity of $O(n \cdot \log n)$. However, it can easily be shown that a combinatorial count of n binary input values always can be computed by $n - 1$ full adders. Therefore, the count network also has the space complexity $O(n)$. The adder units are obviously $O(\log n)$. Since there are k units of this kind we

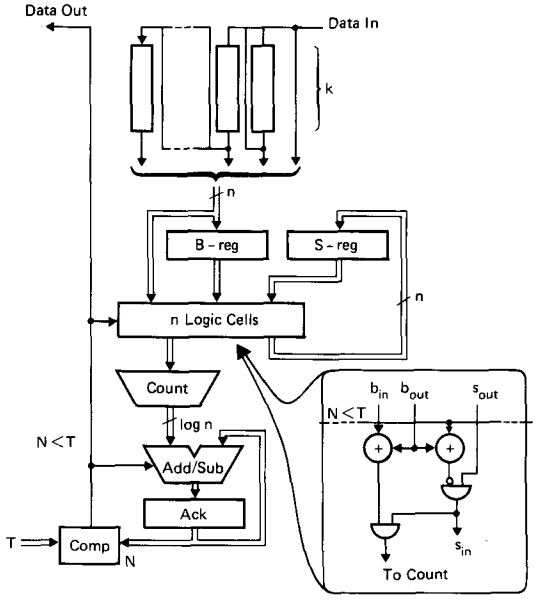

FIGURE 8

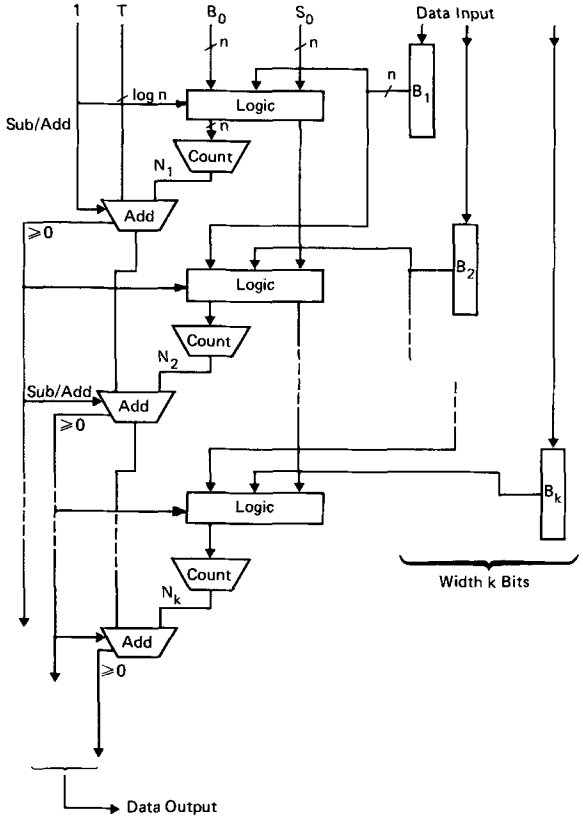

FIGURE 9

conclude that the total space complexity of Fig. 9 is $O(n \cdot k)$. Compared to the strictly serial implementation of Fig. 7, space and time complexity are completely reversed.

The previously mentioned hardware implementation [2] employs $n(n-1)/2$ comparators. Since each comparator has width k bits, this device has space complexity $O(n^2 k)$.

We feel that the lower complexity of the algorithm presented in this paper is a good indicator of the possibility of implementing it effectively in both serial and parallel hardware.

6. CONCLUSIONS

Like so many other algorithms in the present era of growing interest in fast processing and VLSI design, the new algorithm in this paper is based on the possibility of manipulating not only words of data but individual bits. In its strictly bit-serial implementation (Fig. 7) its time complexity is $O(k \cdot n)$, where k is the number of bits in the argument and n is the number of arguments. Thus, its time complexity is directly proportional to the total number of input bits. Also, since the algorithm produces the output pixels with most significant bit first, the user may discontinue the computation at the level of precision he finds appropriate. The

algorithm is open for a more parallel computation as shown by Fig. 8 in which case the time complexity drops to $O(k)$ or as shown by Fig. 9 that computes the result in $O(1)$ time.

Median filtering belong to the broad class of local neighborhood operations. As such, the new algorithm can be implemented as a program for SIMD-Machines (Single Instruction Multiple Data Stream) that utilize several processors to operate on different neighborhoods of data computing several output results simultaneously.

REFERENCES

1. B. Justusson, Noise reduction by median filtering, Proc. 4th Int. Conf. on Patt. Recogn., pp. 502–504, 1978.
2. G. R. Nudd *et al.*, Implementation of advanced real-time understanding algorithms, in Semiannual Technical Report, Image Proc. Institute, Univ. Southern Calif., March 1979.
3. P. M. Narendra, A separable median filter for image noise smoothing, in Proc. IEEE Conf. on Patt. Recognition and Image Analysis, pp. 137–141, 1978.
4. T. S. Huang, A fast two-dimensional median filter algorithm, Proc. IEEE Conf. on Patt. Recognition and Image Analysis, pp. 128–131, 1978.
5. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison–Wesley, Reading, Mass., 1974.
6. P. E. Danielsson, *Histogram Calculations and Median Filter*, PICAP II memo nr 28, 1978. [in Swedish]
7. B. Kruse, Personal communication.