Algorithms And Hardware For Efficient Image Smoothing

Anup Basu, Department of Statistics
Christopher M. Brown, Department of Computer Science
University of Rochester

TR 149

December 1984

Abstract

This work develops some algorithms for efficient implementation of mean and separable median filters for image noise smoothing. The procedures suggested use one-dimensional algorithms repeatedly to obtain two-dimensional mean and separable median in ≤ 4 and $4\lceil \log_2(n+1) \rceil - 2$ operations, respectively, for an n x n square neighborhood (nbhd). A simple generalization of the algorithms to k dimensions shows that at most 2k and $k(2\lceil \log_2(n+1) \rceil - 1)$ operations are required for obtaining the mean and separable median of an n^k nbhd. However, parallel processing can be easily applied to these algorithms. Given only five parallel processors, less than one operation is required to obtain the sum of an n x n nbhd in two dimensions, and for $n \leq 31$ the maximum number of comparisons required to obtain the two-dimensional separable median filter of an n x n nbhd is reduced to less than four. Experimentally we demonstrate that the separable median filter performs better than median filter for certain classes of images. A simple hardware implementation of the above algorithm is also outlined. It is seen that the complexity of this implementation increases linearly depending on the number of inputs, unlike sort-based implementations whose complexity increases exponentially.

1. Introduction

Mean and median type filters are widely used for image noise smoothing and enhancement [1-5, 7, 8, 11]. Various optimization schemes have been suggested by several authors [1, 2, 5, 7, 8] for improving the efficiency of these algorithms. This work discusses some efficient implementations of mean and separable median [8] filters using a two-pass algorithm and some simple parallel processing techniques for two-dimensional neighborhoods. It is seen that whereas approximately four additions are required on the average by the algorithm suggested in Box-Filtering Techniques [7] to obtain the sum of an n x n square nbhd of a point of a digital picture, the procedure given here requires at most 4/p additions given p parallel processors. That is, given only five parallel processors, the number of additions can be reduced below one. Also the given algorithm does not require any additional array as used in the previous one.

Several attempts have been made to reduce the time complexity of median type filters. Huang et al. [5] and P.M. Narendra [8] give algorithms that have O(n) time complexity for an n x n square nbhd. The current work gives a simple two-pass $O(\log_2 n)$ algorithm for obtaining an approximate median filter similar to the one discussed by Narendra. However, given p parallel processors, the number of comparisons required is reduced to $(4\lceil\log_2(n+1)\rceil-2)$ / p. Thus given only five parallel processors, the number of comparisons is less than four when $n \leq 31$. The algorithm given here can be easily implemented in hardware and an implementation is described.

The algorithms can be easily extended to k dimensions where at most 2k/p and $k(2\lceil\log_2(n+1)\rceil-i)$ / p operations are needed for mean and separable median filters, respectively, using an n^k nbhd with p parallel processors.

Section 2 describes the efficient mean filtering algorithm. The efficient separable median filter is discussed in Section 3. Section 4 compares certain properties of the separable median filter with those of a true median filter. Some implementations results are given in Section 5, where a simple hardware implementation is also outlined.

2. An Efficient and Generalized Mean Filtering Algorithm

2.1. Some One-Dimensional Addition Schemes for a Sequence of Numbers

Suppose there are M numbers $x_1, \ldots, x_M$, and it is required to find the sum of every n consecutive numbers in the sequence where n < M (n assumed to be negligibly small compared to M); then the following schemes may be used. For n = 3 to obtain $y_i = x_i + x_{i+1} + x_{i+2}$ and $y_{i+1} = x_{i+1} + x_{i+2} + x_{i+3}$, three additions suffice:

$$z_{i+1} = x_{i+1} + x_{i+2}$$

$$y_i = x_i + z_{i+1}$$

$$y_{i+1} = z_{i+1} + x_{i+3}$$

Thus, on the average, 1.5 additions are required to find the sum of three consecutive numbers. To obtain a similar scheme when n = 4, consider $x_1, ..., x_8$ and $y_i = y_i + y_{i+1} + y_{i+2} + y_{i+3}$; i = 1, ..., 5. Then y_i's may be obtained by the following nine steps:

$$z_2 = x_3 + x_4$$

$$z_1 = x_2 + z_2$$

$$z_3 = x_5 + x_6$$

$$z_4 = z_3 + x_7$$

$$y_1 = x_1 + z_1$$

$$y_2 = z_1 + x_5$$

$$y_3 = z_2 + z_3$$

$$y_4 = x_4 + z_4$$

$$y_5 = z_4 + x_8$$

This implies that, on the average, 1.8 additions are sufficient to obtain the sum of every four consecutive numbers. For n $\geq$ 5, on the average, 2 additions are necessary for obtaining the sum of n consecutive numbers by the following relation:

$$\sum_{j=1}^{n} x_{i+j} = \sum_{j=1}^{n} x_{(i-1)+j} - x_i + x_{i+n}$$

2.2. An Algorithm for Efficient Two-Dimensional Mean Filtering

To obtain an efficient two-dimensional mean filtering algorithm, we first observe that the sum of the square neighborhood (nbhd) of size (2n+1) around the point x_{ij} of a matrix X of dimension M x N is (ignoring boundary effects):

$$S_{ij} = \sum_{k=-n}^{n} \sum_{l=-n}^{n} x_{i+k,j+l} = \sum_{l=-n}^{n} C_{i,j+l} = \sum_{k=-n}^{n} R_{i+k,j} \qquad (2.2.1)$$

where

$$C_{i,j+l} = \sum_{k=-n}^{n} x_{i+k,j+l} \qquad \textbf{(2.2.2)}$$

and

$$R_{i+k,j} = \sum_{l=-n}^{n} x_{i+k,j+l} \qquad (2.2.3)$$

Now, R_{ij}'s and C_{ij}'s are nothing but the sum of consecutive numbers, and hence the algorithms in the previous section can be used for computing them. Given all R_{ij}'s or all C_{ij}'s, S_{ij}'s can be expressed as sums of consecutive numbers which are R_{ij}'s or C_{ij}'s, as shown in Equation 2.2.1.

Thus, given a two-dimensional array containing the pixel values of a digital picture and a similar output array for storing the smoothed image, the procedure for filtering will simply be to obtain all C_{ij}'s or all R_{ij}'s first and store them in the output array, and then use the C_{ij}'s or R_{ij}'s to obtain S_{ij}'s, which may be stored back in the output array, since the R_{ij}'s or C_{ij}'s are required only temporarily.

To obtain the number of additions required to implement this algorithm, first observe that to obtain R_{ij}, 1.5, 1.8, or 2 additions are required, depending on whether the size of the square nbhd is 3 x 3, 4 x 4, or n x n ($n \geq 5$), respectively. To obtain an S_{ij} given R_{ij}'s, a similar number of additions are required. But each S_{ij} requires $(2n+1)$ R_{ij}'s, and also each R_{ij} except the ones around the boundaries are used for computing $(2n+1)$ S_{ij} values. Thus, on the average, the total number of additions required to obtain an S_{ij} equals the number of additions required to obtain an R_{ij} plus the number of additions required to compute an S_{ij} given R_{ij}'s. This implies that 3, 3.6, and 4 additions are required to obtain the sum of a 3 x 3, 4 x 4, and n x n ($n \geq 5$) square nbhd, respectively.

It may be observed that the total number of computations is reduced if R_{ij}'s are obtained in the intermediate step when the number of rows is greater than the number of columns in the matrix representing the digital picture: otherwise C_{ij}'s are computed.

The average number of additions required to obtain the sum of an n x n square nbhd ($n \geq 5$) for an M x N-dimensional matrix is given by:

$$4 + 2(n-1) \,/\, \{ \text{ maximum } (M,N) - n+1 \}$$
$$+ \{(n-3)(M+N-n+2)\} \,/\, \{(M-n+1)(N-n+1)\} \qquad (2.2.4)$$

if R_{ij} or C_{ij} are used appropriately in the intermediate step as mentioned above. The second term in 2.2.4 is minimized by the proper choice of terms to be computed in the intermediate step. It can be seen that the above expression is $\simeq$ 4 when n/N and n/M are negligibly small.

2.3. A Generalization of the Efficient Mean Filtering Algorithm to Three and Higher Dimensions

Consider the problem of obtaining the sum of each $(2n+1)^k$ nbhd of an array P given an additional array S for storing the sums, both arrays being k-dimensional of dimension $N_1 \times N_2 \times \cdots \times N_k$. This problem can be solved as a simple extension of the two-dimensional algorithm observing that here $(k-1)$ intermediate levels of computations are required before the required sums are finally obtained. The procedure is as follows:

1) Arrange $N_1, ..., N_k$ in non-decreasing order of magnitude, i.e., obtain $I_1, ..., I_k$ such that $N_{I_1} \leq N_{I_2} \leq \cdots \leq N_{I_k}$.

2) Obtain the partial sums with respect to the I_1^{th} dimension, storing the sums in the array S, say, in which the sums are to be stored.

3) Repeat the procedure of obtaining partial sums on the $I_2^{th}, ..., I_k^{th}$ dimensions of S, each time storing the sums back into S.

Lemma 2.3.1:

To compute the sum of each n^k nbhd of an array P, given an additional array S for storing the sums (both arrays P, S being k-dimensional of dimensions $N_1 \times N_2 \times \cdots \times N_k$), the average number of additions required by the above algorithm is:

1) k for n = 2,

2) 1.5k for n = 3,

3) 1.8k for n = 4, and

4) 2k for $n \geq 5$,

provided n/N_k is negligibly small for i = 1, ..., k.

Proof:

By induction.

2.4. An Efficient Weighted Mean Filtering Algorithm

For the, weighted mean filter considered here, the weights depend on the (city block) distance of a pixel (picture element) from the pixel being smoothed.

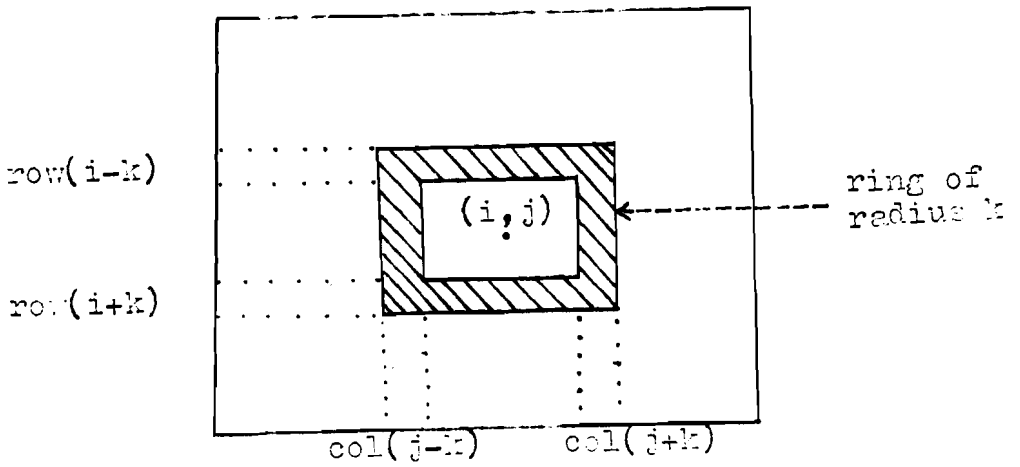

Figure 1: Ring of Radius k Around Point (i, j)

The variables and parameters in weighted mean filtering may be defined as the set {P, S, M, N, n, w_0, ..., w_n}. This means that the image P has to be smoothed using a square nbhd of size (2n+1) x (2n+1) by weighted mean filtering with weights w_0, w_1, ..., w_n. The ring of radius k around the point (i, j) has weight w_k, k = 0, 1, ..., n (Figure 1). The smoothed image is to be stored in S, where both P and S have dimensions M x N. So the following equations can be obtained:

$$\text{WMEAN}\{P, n, (i, j), w_0, ..., w_n\} = \sum_{k=0}^{n} w_k \, SR_k \{P, (i, j)\}$$

$$= \sum_{k=0}^{n} (w_k - w_{k+1}) \, SQNSUM\{P, k, (i, j)\} \tag{2.4.1}$$

where SR_k {P, (i, j)} = sum of the ring of radius k around point (i, j) of P

$$= \sum_{l=i-k}^{i+k} \{P(l, j-k) + P(l, j+k)\}$$

$$+ \sum_{l=j-k+1}^{j+k-1} \{P(i-k, l) + P(i+k, l)\} \tag{2.4.2}$$

SQNSUM {P, k, (i, j)} = sum of square nbhd of radius k around point (i, j) of P

$$= \sum_{k=i-n}^{i+n} \sum_{l=j-n}^{j+n} P(k, l) \tag{2.4.3}$$

and $w_{n+1} = 0$.

Thus the algorithm for weighted mean filtering initializes S to $(w_1 - w_2)P$, and then adds $(w_k - w_{k+1})$ SQNSUM {P, k, (i, j)} to S(i, j) varying k from 1 to n

for all valid (i, j) values. S cannot be used for computing SQNSUMs as in the previous algorithms, since the current value in S(i, j) needs to be stored and incremented by a weighted factor of the nbhd sums that are computed, so a separate array needs to be kept and an algorithm similar to the one discussed by McDonnell [4] has to be used.

The values $w_i^1 = w_i - w_{i+1}$ need to be obtained only once, and hence their computation does not contribute to the number of additions required on the average for each point. For each point being smoothed the SQNSUMs are obtained for one 3 x 3 nbhd and (n-1) k x k ($k \geq 5$) nbhds, so the total number of additions per point is 3 + 4(n-1) = (4n-1). Thus the above procedure needs (4n-1) additions and (n+1) multiplications instead of $(2n+1)^2$ additions and $(2n+1)^2$ multiplications.

2.5. A Parallel Algorithm for Efficient Mean Filtering

Consider the algorithm discussed in Section 2.2. Here the $\{R_{ij}$, j varying$\}$ can be computed in parallel for i = 1, ..., M. Subsequently, $\{S_{ij}$, i varying$\}$ can be computed in parallel for different j's (j varying over the valid range of values of j), from the R_{ij}'s. Otherwise $\{C_{ij}$, i varying$\}$ for j = 1, ..., N and subsequently $\{S_{ij}$, j varying$\}$ for varying i, can be computed in parallel. Thus, if there are L parallel processors $Z_1, Z_2, ..., Z_L$ and if R_{ij}'s and then S_{ij}'s are to be obtained for a given digital picture P of resolution M x N, the following procedure may be adapted (exchange rows and columns if M < N):

1) Use processor Z_i to process row (i+kL) of P (i = 1, ..., L; k = 0, ..., k_i where $i + LK_i \leq M$ and $i + L(K_i + 1) > M$) to obtain R_{ij}'s, storing them in S, as in the algorithm in Section 2.2.

2) Use processor Z_j to process column (j+kL+n) of S (j = 1, ..., L; k = 0, ..., k_j, where $j + LK_j + n \leq N\text{-}n$ and $j + L(K_j + 1) + n > N\text{-}n$, assuming filtering is done using a (2n+1) x (2n+1) nbhd) to obtain the final sums, S_{ij}'s, as in the algorithm in Section 2.2 (Figure 2).

(Here an extra one-dimensional array,which for convinience may be appended to the array storing the smoothed image,is needed for implementation using parallel processors in order that the values to be subtracted are not overwritten by the currently computed values. When filtering using a single processor the need for extra storage can be avoided by carefully using the rows or columns around the boundary that cannot be filtered using the normal nbhd.)

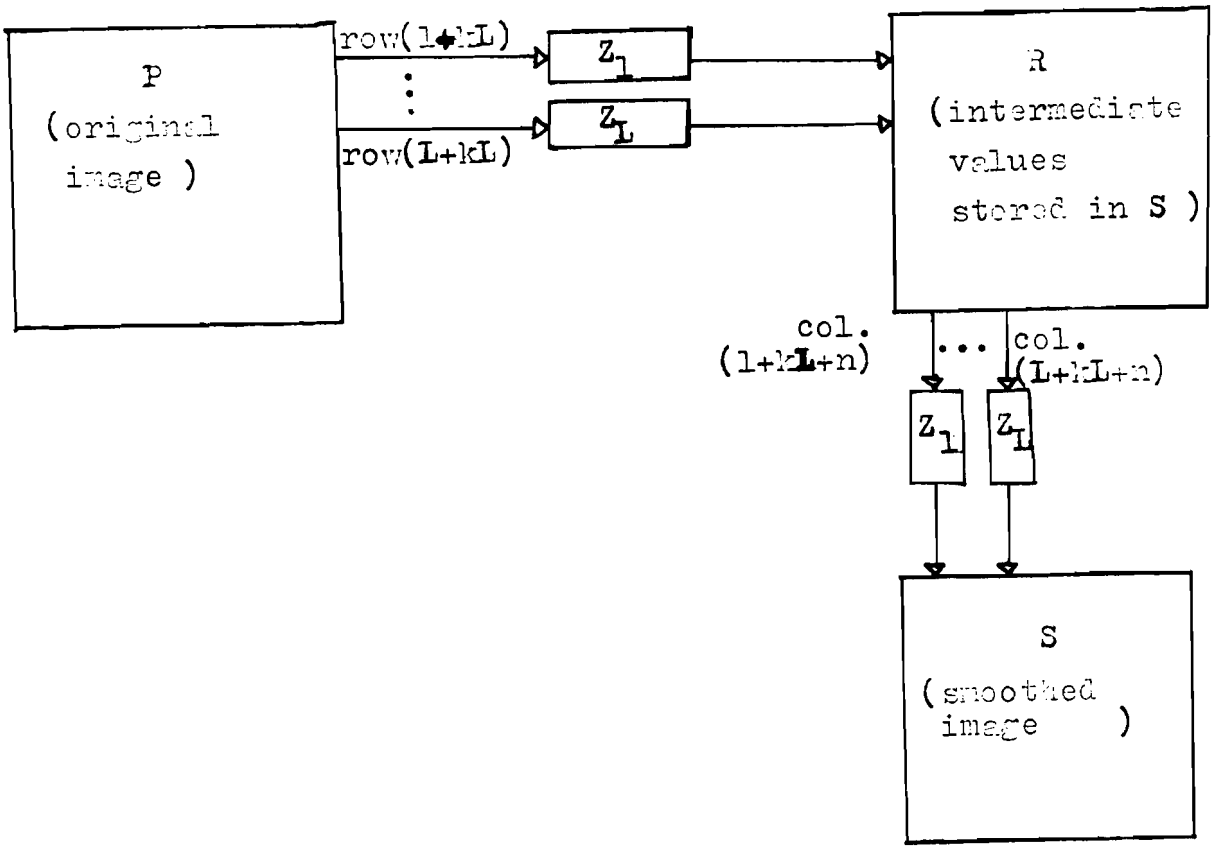

Figure 2: Parallel Algorithm for Mean Filtering
a Digital Picture P of Resolution M x N (M > N)

The above algorithm can be generalized to k dimensions for k > 2. The number of processors need not be less than the dimensions of the picture, as discussed in Section 3.4. Given L parallel processors, the average number of additions required by the above algorithm to find the sum of each n^k nbhd of a k-dimensional array is given by:

1) k/L if n = 2,

2) 1.5k/L if n = 3,

3) 1.8k/L if n = 4, and

4) 2k/L if n $\geq$ 5

under the assumptions of the previous sections.

In particular, for a two-dimensional nbhd, given only five parallel processors, the average number of additions required for obtaining the sum of an n x n square nbhd ($n \geq 5$) is less than or equal to 0.8. Thus, given sufficiently many parallel processors, the average number of additions required for obtaining a two-dimensional mean filter can be made as small as required. This is true for higher dimensions as well.

3. An Efficient Median Type Filtering Algorithm

3.1. Some One-Dimensional Sorting Procedures for a Sequence of Numbers

Related to the addition problem of Section 2.1, is the problem of sorting every n consecutive numbers of a given sequence of numbers $x_1, ..., x_M$, where $n < M$. For $n = 3$, the procedure is as follows:

1) For sorting x_1, x_2, x_3 and x_2, x_3, x_4, first compare x_2, x_3 and store the sorted pair. Then merge the sorted pair with x_1 and x_4 to obtain the sorted triplets (x_1, x_2, x_3) and (x_2, x_3, x_4), respectively.

2) Repeat the procedure in (1) for x_i, x_{i+1}, x_{i+2} and $x_{i+1}, x_{i+2}, x_{i+3}$ for $i = 2, ..., M-3$.

For comparing x_2, x_3, one comparison is needed, and merging x_1 with sorted x_2, x_3 takes at most two comparisons. Thus for sorting (x_1, x_2, x_3) and (x_2, x_3, x_4), at most five comparisons are required, which implies that on the average 2.5 comparisons are sufficient for sorting every triplet in a given sequence of numbers.

In general, to sort every n consecutive numbers ($n \geq 4$) of a sequence $x_1, ..., x_M$ (n considered to be negligibly small compared to M), the procedure is simply to sort $x_1, ..., x_n$ by some sorting scheme and then derive every subsequent sorted n-tuplet by deleting one element and inserting one element into the previous sorted list using binary search. That is, if $\{x_i, ..., x_{i+n-1}\}$ are already sorted, then $\{x_{i+1}, ..., x_{i+n}\}$ can be obtained by deleting x_i and inserting x_{i+n} into the previous sorted list. If n is negligible compared to M, the maximum number of comparisons needed is $2\lceil \log_2(n+1) \rceil - 1$, on the average. The -1 arises because at most $\lceil \log_2(n+1) \rceil - 1$ comparisons are required to delete an element in a sorted list when it is known that the element to be deleted is in the list.

3.2. An Efficient Separable Median Filtering Algorithm

An approximation to the median filter is the separable median filter (SMF) discussed by Narendra [8]. The main purpose of the approximation is to increase the efficiency of median type filters. An algorithm based on histogram updation technique was suggested by Huang et al [5]; both these algorithms have O(n) time complexity. The algorithm suggested here reduces the time complexity to

$O(\log_2 n)$ by maintaining a sorted array of size n and inserting a value and deleting a value from it using binary search.

The SMF algorithm splits up a square nbhd into separable segments, which are nothing but the rows or columns of the nbhd, computes the median of each of the segments, and finally the median of the medians. In order to obtain an efficient implementation of SMF, an algorithm similar to the one discussed in Section 2.2 is considered. The algorithm consists of two passes. In the first pass the medians of all the partial columns (or rows) are computed and stored in the array provided for storing the smoothed image. In the second pass the medians computed in the first pass are used to obtain the medians of all the partial rows (or columns, respectively), giving us the SMF of the original image. Note that usually the results obtained by performing the operations row-wise first and then column-wise vary from the results obtained by reversing the operations of each pass, unlike the invariant performance of the mean filtering algorithm. However, the general properties of the SMF (discussed in Section 4) are retained in either case.

Notationally, the algorithm may be represented as follows:

$$S_{ij} = \text{Median}\{C_{i,j+l};\ l = -n, \ldots, n\} \tag{3.2.1}$$

and

$$C_{i,j+l} = \text{Median}\{x_{i+k,j+l};\ k = -n, \ldots, n\} \tag{3.2.2}$$

where X is the original image, S the smoothed image, and a square nbhd of size (2n+1) is used for filtering. In the above example, the processing is done column-wise first and then row-wise. If the algorithm in Section 3.1 is used to obtain the C_{ij}'s from the X_{ij}'s in the first pass and then again to obtain the S_{ij}'s from the C_{ij}'s in the second pass, by arguing as in Section 2.2, it can be easily verified that the maximum number of comparisons required, on the average, to obtain an S_{ij} is $4\lceil \log_2(n+1) \rceil - 2$.

3.3. A Generalization of the Efficient Separable Median Filtering Algorithm to Higher Dimensions

The separable median filter of a k-dimensional nbhd can be recursively defined as follows:

1) The separable median of a one-dimensional nbhd is the median of the nbhd.

2) The separable median of a k-dimensional nbhd ($k \geq 2$), n^k, is the median of the n separable medians of the n (k-1)-dimensional nbhds which form the n^k nbhd.

[Without loss of generality, it may be assumed that a (k-1)-dimensional nbhd contained in a k-dimensional nbhd is obtained by fixing the last index of the k-dimensional nbhd and varying the rest.]

Symbolically the above definition can be rewritten as:

$$SM\{A_i \; ; \; i = 1, ..., n\} = Median\{A_i \; ; \; i = 1, ..., n\}$$

$$SM\{A_{i_1}, ..., i_k \; ; i_1 = 1, ..., n, ..., i_k = 1, ..., n\}$$

$$= Median\{SM\{A_{i_1}, ..., i_k \; ; \; i_1 = 1, ..., n, ..., i_{k-1}$$

$$= 1, ..., n\}, \; i_k = 1, ..., n\} \qquad (3.3.1)$$

where SM denotes the separable median.

As in Section 2.3, it can be easily verified that the maximum number of comparisons required on the average for obtaining the separable median filter of a k-dimensional array using an n^k nbhd is $2k \lceil \log_2(n+1) \rceil - k$. The k-dimensional SMF can be obtained by repeatedly applying the algorithm in Section 3.1 to obtain the SMs for all one-dimensional nbhds, then the SMs for all two-dimensional nbhds, and so on, until the SMF of the k-dimensional array is obtained.

3.4. A Parallel Algorithm for Efficient SMF

The implementation of the SMF using parallel processing can be viewed quite similarly to the algorithm given for efficient mean filtering using parallel processors. Given L parallel processors, the average maximum number of comparisons required to obtain the two-dimensional SMF is $(4 \lceil \log_2(n+1) \rceil - 2) / L$ (Table I). The number of processors used is not limited by the dimensions of the picture. For example, for an N x N size image, given 2L parallel processors, where $2L > N$ but $2L < 2N$, we can consider L processors smoothing the first N/2 columns and the remaining L processors smoothing the last N/2 columns (assuming N even) in the first pass, and similarly in the second pass.

Table I: Maximum Number of Comparisons Needed by Efficient Separable Median Filter (ESMF) with Parallel Processing

Nbhd. Size	No. Of Comparisons With 1 Processor	No. Of Comparisons With 10 Processors	No. Of Comparisons With 100 Processors
3x3	5	0.5	0.05
7x7	10	1.0	0.10
15x15	14	1.4	0.14
31x31	18	1.8	0.18
63x63	22	2.2	0.22

4. Properties of the Efficient Separable Median Filter (ESMF)

4.1. Noise Reduction by ESMF

For an n x n nbhd the variance of Gaussian and uniform noise, with noise a^2, after being filtered by a two-dimensional median filter, are given by (respectively):

$$\frac{\pi}{2(n^2+2)}\, a^2 \text{ and } \frac{3}{(n^2+2)}\, a^2 \tag{4.1.1}$$

For ESMF the corresponding approximate figures as given in [8] are (respectively):

$$\frac{\pi}{8}\, \frac{2^{2n}[(\frac{n-1}{2})!]^4}{(n!)^2\,(n+2)}\, a^2 \text{ and } \frac{3}{4}\, \frac{2^{2n}[(\frac{n-1}{2})!]^4}{(n!)^2\,(n+2)}\, a^2 \tag{4.1.2}$$

Since the above expression is rather complicated, we derive a simplification of it. (It may be noted that for small n the variances are quite close.) A convenient approximation to factorials is given by Stirling's approximation [9]:

$$n\,! = \sqrt{2\pi n}\; n^n\, e^{-n} \tag{4.1.3}$$

Using the above approximation, we obtain

$$\frac{2^{2n}[(\frac{n-1}{2})!]^4}{(n!)^2\,(n+2)} = 2\pi\frac{e^2}{n^2}\,(1-\frac{1}{n})^{2n}$$

$$\text{but } \lim_{n\to\infty}(1-\frac{1}{n})^n = e^{-1}$$

Thus, asymptotically,

$$\frac{2^{2n}[(\frac{n-1}{2})!]^4}{(n!)^2\,(n+2)} = \frac{2\pi}{n^2} \tag{4.1.4}$$

Thus for large n, the values in (4.1.2) reduce to (respectively):

$$\frac{\pi^2}{4n^2}\,a^2 \text{ and } \frac{3\pi}{2n^2}\,a^2 \tag{4.1.5}$$

From (4.1.1) and (4.1.5), it can be seen that asymptotically the variance of the noise after separable median filtering is only $\pi/2$ times the noise variance after median filtering, and for both the reduction of noise variance is inversely proportional to the number of points in the nbhd.

4.2. Edge Preservation by ESMF

The ESMF retains the detailed structure of many corners or vertices in an image much better than the ordinary median filter. The following example illustrates this. Consider the corner shown in Fig. 3.

20	20	10
20	20	10
10	10	10

Figure 3: A sharp corner in the image

Suppose the corner in Fig. 3 is being smoothed with a 3 x 3 window. The normal median filtering and even threshold median filters [11] replace the central gray level value 20 by the value 10, though this destroys the vertices of the rectangles in the filtered image. Note that the value of the separable median is 20 and not 10. Thus ESMF preserves the vertices of rectangular shapes, which form an important component in many images. It may be observed that median filters can destroy a k x k nbhd around the vertex of a rectangle by filtering with a (2n+1) x (2n+1)

nbhd, where k is the largest integer satisfying:

$$(n+k)^2 < 2n^2 + 2n + 1$$

$$\Rightarrow k = \lfloor -n + \sqrt{2n(n+1)} \rfloor$$

where $\lfloor x \rfloor$ denotes the largest integer $\leq$ x. In fact, points outside the k x k window mentioned above for which the filtering window has more points from outside the rectangle than from inside are also likely to be destroyed by median filtering. Table II below shows the number of points around a rectangular vertex that may be destroyed by median filtering using an n x n window:

Table II

Window Size	No. Of Points Around Rectangular Vertices That May Be Destroyed By Median Filtering
3x3	1
5x5	3
7x7	5
9x9	8
11x11	12
13x13	17

No point is disturbed by ESMF in any of the above cases, provided the length of the smaller side of a rectangle is not less than n/2 for an nxn window. It is assumed of course that that the sides of the rectangular shapes are somewhat parallel to the sides of the image. Thus it is clear that the ESMF discussed here not only reduces the time complexity of the filter to $O(\log_2 n)$ for an n x n nbhd, but also preserves certain significant image information far better than ordinary median filter. Also, the noise reduction by ESMF is of the same order as that of median filter except for a factor of $\pi/2$ as shown in the earlier section. The performance of the ESMF is compared with median filter for some actual images

in Section 5 where a hardware implementation is also briefly outlined.

5. Results and Implementation

5.1. Results

Fig. 4 illustrates the results obtained by applying median and separable median filters with 3 x 3, 5 x 5, 7 x 7, and 9 x 9 nbhds on a generated picture which resembles the side of a building at night illuminated by the lights in the rooms. It is seen that the median filter performs worse as the nbhd size increases, whereas separable median filtering does not affect any details of the picture for n $\leq$ 7. It may be observed that if an M x N rectangle undergoes ESMF, then no details are destroyed as long as n $\leq$ minimum (M, N) when filtering using a (2n+1) x (2n+1) square nbhd. Thus when a 3 x 3 square nbhd is used, the only edges that may be affected by ESMF are lines with different gray level values on either side.

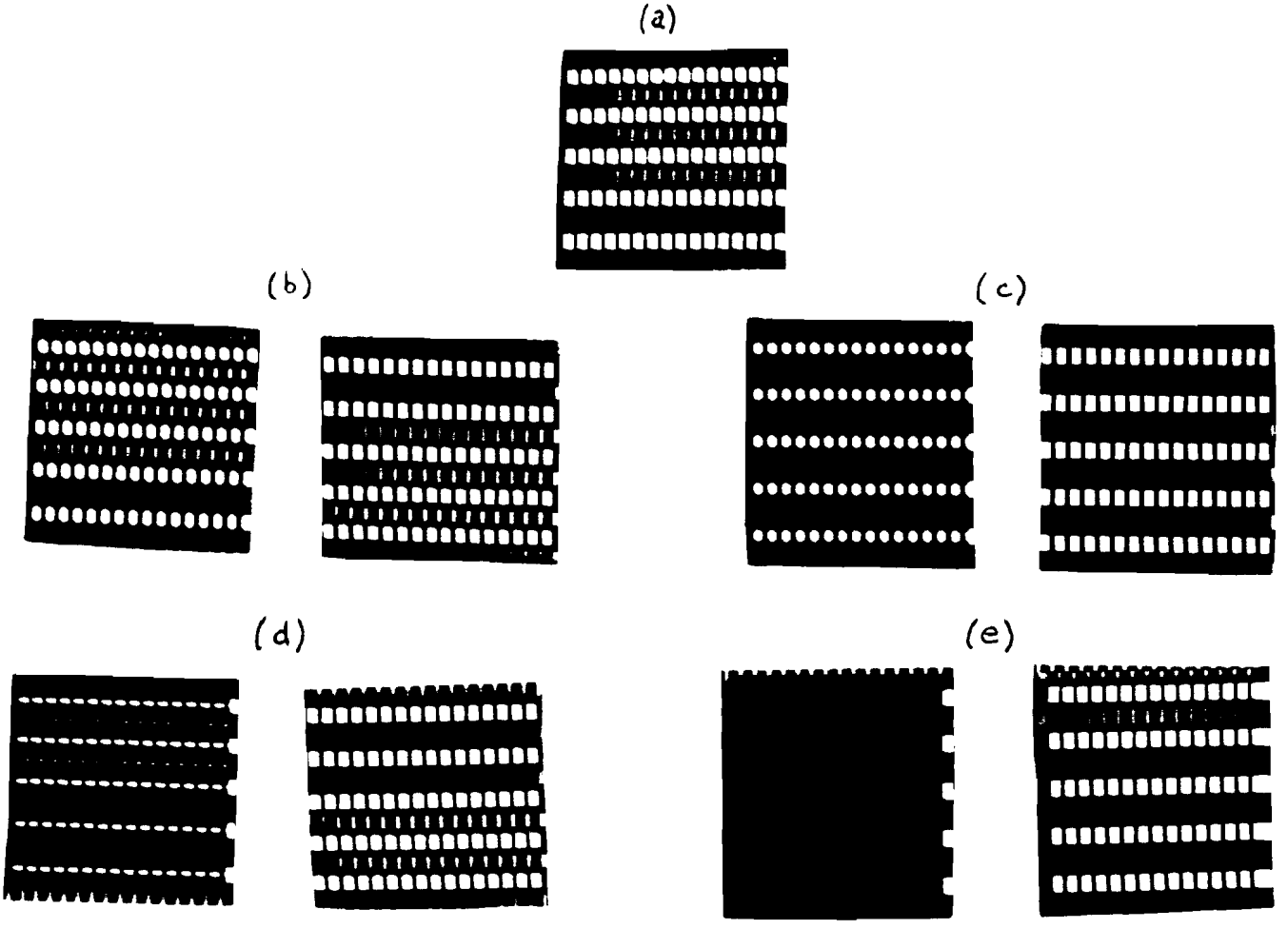

Figure 4: Comparison of Median and ESMF Filters
Original image (a), and pairs after median filtering and ESMF of nbhd. sizes 3 x 3, 5 x 5, 7 x 7, 9 x 9 (b, c, d, e)

5.2. Implementation

A hardware implementation of ESMF for nbhd size $\leq$ N x N can be realized using the following basic units (Fig. 5):

1) A shift register (SR) which can contain up to N pixel values (N words) and has a word-oriented shift operation. This register can be accessed as an array (SR(I), I = 1, ..., N) and the shift operation can be performed both towards left and right on a left truncated part of it. That is, there are basic operations SHIFTLEFT(I) which moves the contents of the $(J+1)^{th}$ word to the J^{th} word for J = I, ..., N-1 and 0 to the N^{th} word, I may vary between 1 and N. SHIFTRIGHT(I) is similarly defined.

2) A circuit BX for performing binary search on an array of n elements sorted in a descending order (first n elements of SR). Given X, BX(X) returns the index where the match occurs or returns I where SR(I-1) $\geq$ X $\geq$ SR(I) (define SR(0) = max value of pixel).

With these components inserting a value X into SR reduces to:

I = BX(X); SHIFTRIGHT(I); SR(I) = X;

And deleting a value is equivalent to:

SHIFTLEFT(BX(X));

where X is known to belong to SR.

To obtain the medians in a particular pass the procedure initializes the register SR by n insert operations and then performs one delete and one insert operation for every successive point.

The actual circuit for BX does not require any binary search, but uses N comparators connected in parallel. The comparator C_i takes the contents of SR(i) as one of the inputs and X (the number whose position is to be determined) as the other input, and outputs 1 if SR(i) > X and 0 otherwise. Unlike the procedure discussed in [8], the circuit shown in the diagram below does not require any sorting network pipeline with comparators; also it needs a smaller number of comparators. Also the complexity of the circuit increases only linearly depending on the number of inputs, unlike sort-based hardware whose complexity increases exponentially.

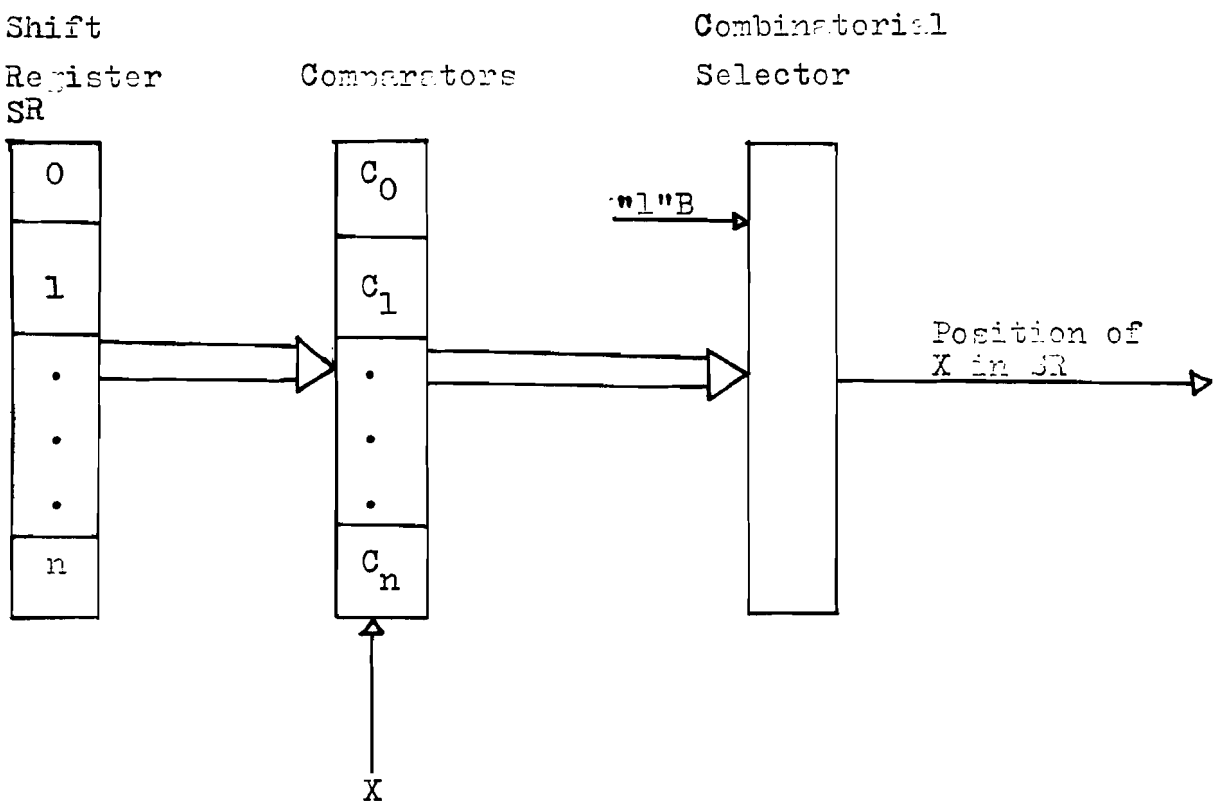

Figure 5: Circuit for Searching a Sorted Array

The Combinatorial Selector first looks for the pattern 10 in the input string, which is a bit string of 1's followed by 0's. Note that an extra bit 1 is input to the selector to detect the first 0 in the input string when all bits are zeros. For the problem considered here the input string can never be all one bits. The occurance of the pattern 10 may be detected by combining every two consecutive inputs, and the output of the selector is determined by the position where this combination occurs. Note that the above combination can occur in one position only.

For digital implementation in real-time hardware (video rates, for example) an intermediate array is required to store the row medians before they are subsequently processed. With additional storage the line scanned serial input and the intermediate row medians can be processed simultaneously (Fig 6). This hardware structure can achieve higher speed compared to sort-based designs. Also the speed is almost constant independent of the number of inputs if the delay due to fan-in limitations of the gates in the selector is ignored. The number of comparators required by the hardware discussed here increases only linearly depending on the number of inputs and is compared with the number of comparators required by sort-based implementation of S.M.F. in Table III. The number of comparators required by sorter networks is obtained from Knuth[13], for n > 16 the asymptotic case [13,p. 229] is used. Thus the algorithm discussed here not only has $O(\log_2 n)$ computational complexity for an n x n nbhd. compared to O(n) algorithms [5,8] discussed earlier, but also has a simpler real time implementation.

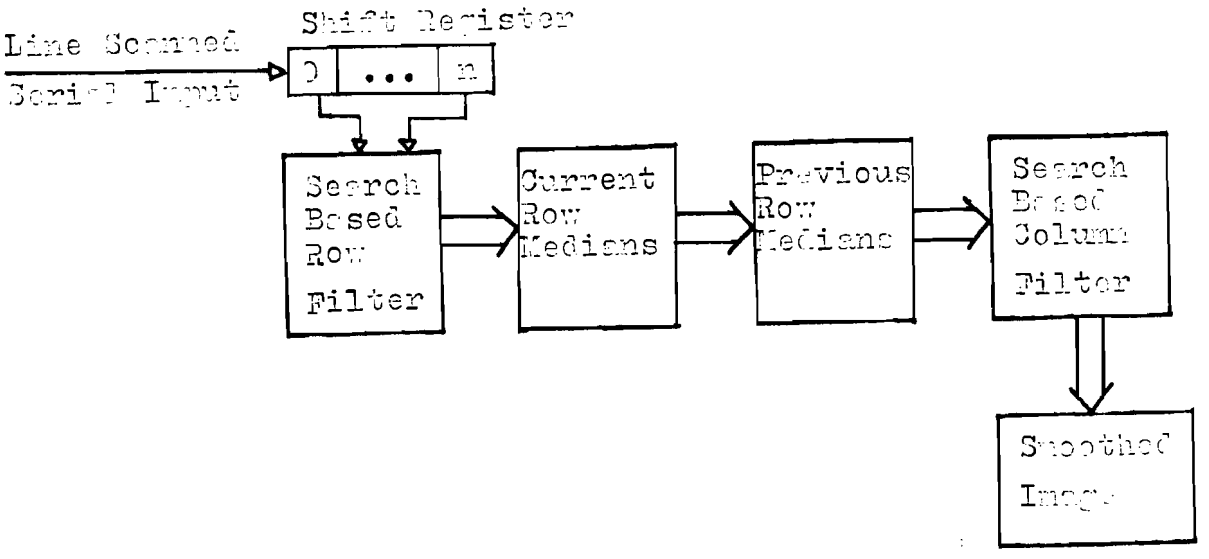

Figure 6: Real Time E.S.M.F.

Table III: Comparison Of No. Of Comparators Required By Real Time Sort-based S.M.F. and E.S.M.F.

Nbhd. Size	No. of comparators used by sort-based S.M.F.	No. of comparators used by real time E.S.M.F.
3X3	6	6
5X5	18	10
9x9	50	18
31x31	270	62
63x63	860	126

6. Acknowledgements

The preparation of this paper was supported in part by the National Science Foundation under Grant MCS-8302038. The authors would also like to acknowledge the support of Peggy Meeker in preparing this document.

7. References

[1] Ataman, E., V.K. Aatre, and K.M. Wong, "A fast method for real time median filtering," *IEEE Transactions ASSP 28*, 415-420, 1980.

[2] Chaudhuri, B.B., "Efficient algorithm for image enhancement," *Proc., IEEE 130*, E, 3, 95-97, May 1983.

[3] Eliason, E.M. and L.A. Soderblom, "An array processing system for lunar geochemical and geophysical data," *Proc., 8th Lunar Science Conference*, 1163-1170, 1977.

[4] Gonzalez, R.C. and P. Wintz. *Digital Image Processing.* Cambridge, MA: Addison-Wesley, 1977.

[5] Huang, T.S. et al., "A fast two-dimensional median filtering algorithm," *IEEE Transactions ASSP-27*, 13-18, February 1979.

[6] Kuck, D.J. *The Structure of Computers and Computations* (Vol. 1, pp. 135-55, 259-63). Wiley, 1978.

[7] McDonnell, M.J., "Box-filtering techniques," *CGIP 17*, 65-70, 1981.

[8] Narendra, P.M., "A separable median filter for image noise smoothing," *IEEE Transactions PAMI 3*, 1, 1981.

[9] Rao, C.R. *Linear Statistical Inference* (2nd edition) (p. 59). Wiley, 1973.

[10] Rosenfeld, A. and A.C. Kak. *Digital Picture Processing.* New York: Academic Press, 1976.

[11] Scollar, I. et al., "Image enhancement using the median and interquartile distance," *CVGIP 25*, 2, 236-251, February 1984.

[12] Tukey, J.W. *Exploratory Data Analysis* (Ch. 7, pp. 205-236). Reading, MA: Addison-Wesley, 1976.

[13] Knuth, D.E. *The Art Of Computer Programming.* vol. 3.(pp. 220-246). New York:Addison-Wesley ,1973.