

A Fast Two-Dimensional Median Filtering Algorithm

THOMAS S. HUANG, SENIOR MEMBER, IEEE, GEORGE J. YANG, STUDENT MEMBER, IEEE, AND
GREGORY Y. TANG, STUDENT MEMBER, IEEE

Abstract—We present a fast algorithm for two-dimensional median filtering. It is based on storing and updating the gray level histogram of the picture elements in the window. The algorithm is much faster than conventional sorting methods. For a window size of $m \times n$, the computer time required is $O(n)$.

I. INTRODUCTION

TUKEY [1], [2] was among the first who suggested the use of median filters for signal smoothing. More recently, Rabiner, Samgur, and Schmidt [3] and Jayant [4] applied median filters to speech processing; Pratt [5] and Frieden [6] applied them to image processing.

In terms of image processing, median filtering is defined as follows. Let $[x_{ij}]$ be the matrix representing a digitized image. Then the result of the median filtering with an $m \times n$ (where $m, n = \text{odd integers}$) window is an image $[y_{ij}]$ where y_{ij} is equal to the median of the gray levels of the picture elements lying in an $m \times n$ window centered at the picture element x_{ij} in the input image. Throughout this paper, by an $m \times n$ window, we mean a window which has m pels in the horizontal direction and n pels in the vertical direction. In computer calculation, the output image is usually computed in the conventional scanning order: $y_{11}, y_{12}, y_{13}, \dots, y_{21}, y_{22}, y_{23}, \dots$.

Although it is well known that median filtering is useful for reducing random noise (especially when the noise amplitude probability density has large tails) and periodic patterns, theoretical results on its behavior are nonexistent in the open literature. Recently, Tyan [7] and Justusson [8] have obtained some interesting analytical results. Justusson studied the statistical behavior of median filters, while Tyan studied the fixed points of median filtering among other things. We hope their results will be published soon.

In this paper, we are concerned with only the computational aspects of two-dimensional median filtering. Conventional sorting methods, such as Quicksort [9], are time-consuming. To obtain the median of $m \times n$ numbers, the average number of comparisons is typically proportional to mn assuming the original ordering is random. The fast algorithm we report here requires only approximately $(2n + 10)$ comparisons.

Manuscript received February 14, 1978; revised August 31, 1978. This work was supported by the Defense Advanced Research Projects Agency under Contract MDA 903-77-G-1.

The authors are with the School of Electrical Engineering, Purdue University, West Lafayette, IN 47907.

II. A FAST TWO-DIMENSIONAL MEDIAN FILTERING ALGORITHM

In doing median filtering, we are computing *running* medians. From one output picture element to the next, the $m \times n$ window moves only one column. To get the numbers in the new window from those in the preceding window, we throw away n points and add in n new points. The remaining $mn - 2n$ numbers are unchanged. To take advantage of this, a fast median filtering algorithm is developed which is based on storing the gray level histogram of the mn picture elements (pels) in the window, and updating it as the window moves.

The algorithm consists of the following steps:

Step 1: Set up the gray level histogram of the first window and find the median. Also, make the count, 1tmdn of the number of pels with gray level less than the median.

Step 2: Move to the next window by deleting the leftmost column of the (previous) window and adding one column to the right. The histogram is updated. So is the count 1tmdn . Now 1tmdn stores the number of pels in the current window having gray levels less than the median of the previous window.

Step 3: Starting from the median of the previous window, we move *up/down* the histogram bins one at a time if the count 1tmdn is *not greater/greater* than [number of pels in a window divided by 2] and update the count 1tmdn until the median bin is reached.

Step 4: Stop if the end of the line is reached. Otherwise go to Step 2.

Note: In Step 3, if the count 1tmdn is *not greater than* [number of pels in a window divided by 2], then we may have the same median as before and, therefore, do not need to move.

The exact procedure is described below in pidgin Algol: Algorithm Fast Median Filtering

begin

comment This algorithm is doing median filtering of an image;

comment $\text{hist}[0:255]$: histogram array;

comment mdn : median value in a window;

comment 1tmdn : number of pels having gray levels less than mdn in a window;

comment $\text{left.column}[0:\text{window.y size} - 1]$: the leftmost column of the previous window;

comment $\text{right.column}[0:\text{window.y size} - 1]$: the rightmost column of the current window;

comment All the above variables and arrays are global;

```

comment This algorithm ignores border processing problem;
th ← [window.xsize * window.ysize/2]; /*global; a param-
                                     eter to help de-
                                     cide median */
for i ← [window.ysize/2] + 1 until picture.ysize - [win-
                                     dow.ysize/2] do
                                     /* i indicates pic-
                                     ture line number*/
begin
  Set up array hist for the first window and find mdn
  by moving through the histogram bins and updating
  count 1tmdn;
  for j ← [window.xsize/2] + 2 until picture.xsize -
  [window.xsize/2] do
  /* j indicates picture column number */
  begin
    put the leftmost column of the previous win-
    dow in array left.column; put the rightmost
    column of the current window in array right.
    column; runningmdn
  end
end
end
procedure runningmdn:
begin
  for k ← 0 until window.ysize - 1 do /* moving to the
                                     next (i.e. the
                                     current) win-
                                     dow */
begin
  g1 ← left.column[k]; /* delete the leftmost col-
                                     umn of the previous
                                     window */
  hist[g1] ← hist[g1] - 1;
  if g1 < mdn then 1tmdn ← 1tmdn - 1; /* check
                                     to up-
                                     date the
                                     counter
                                     1tmdn */
  g1 ← right.column[k]; /* add the rightmost col-
                                     umn of the current
                                     window */
  hist[g1] ← hist[g1] + 1;
  if g1 < mdn then 1tmdn ← 1tmdn + 1 /* check to
                                     update
                                     the
                                     counter
                                     1tmdn */
end;
comment find the median;
if 1tmdn > th /* the median in the current window is
               smaller than the one in the previous
               window */
then repeat
begin
  mdn ← mdn - 1; /* move down one
                  histogram bin */

```

```

1tmdn ← 1tmdn - hist[mdn] /* update
                           counter
                           1tmdn */
end
until 1tmdn ≤ th /* mdn is the desired
                 median */
else while 1tmdn + hist[mdn] ≤ th /* the desired
                                   median is
                                   (still)
                                   greater than
                                   mdn */
do begin
  1tmdn ← 1tmdn + hist[mdn]; /* update
                              counter
                              1tmdn
                              */
  mdn ← mdn + 1 /* move up one histo-
                 gram bin */
end;
comment At this point, mdn is the desired median in all
cases.

```

end

A Fortran version of the algorithm was implemented and tested on a PDP 11/45 computer under the UNIX operating system. A listing of the Fortran program is available on request.

We note that the storage requirement of the algorithm is minimal. Mainly, we have to store the histogram which occupies 256 locations if the image amplitude is quantized to 8 bits.

III. EXPERIMENTAL RESULTS ON COMPUTING TIME REQUIREMENTS

The Fortran version of the fast median filtering algorithm was applied (on the PDP 11/45 computer) to six images.

Images

- 1) Girl, original;
- 2) Girl, with BSC noise;
- 3) Girl, with Gaussian noise;
- 4) Airport, original;
- 5) Airport, with BSC noise;
- 6) Airport, with Gaussian noise.

Each image contains 256×256 picture elements and 8 bits (256 gray levels) per picture element. Images 2) and 5) were obtained by passing images 1) and 4), respectively, bit by bit through a binary symmetrical channel with a bit error probability of 0.035. Images 3) and 6) were obtained by adding to images 1) and 4), respectively, zero-mean white Gaussian noise. The signal-to-noise ratios of all four noisy images are approximately 19.5 dB. Images 1)–6) are shown in Fig. 1 (a)–(f), respectively. The median-filtered results (using a 3×3 window) are shown in Fig. 2 (a)–(f).

As a comparison, the Quicksort algorithm was programmed in Fortran and applied to the same six images. In running each of the six images, the average number of comparisons

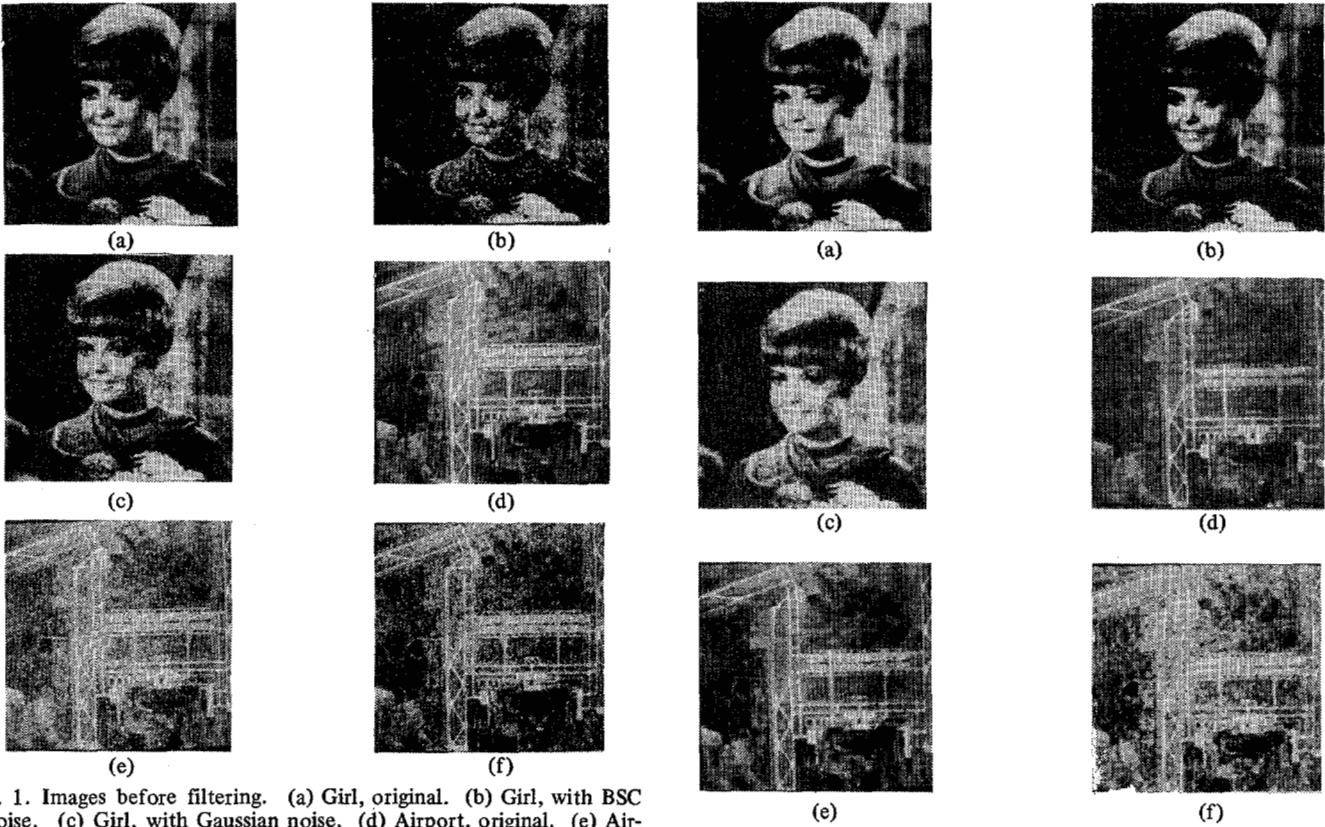


Fig. 1. Images before filtering. (a) Girl, original. (b) Girl, with BSC noise. (c) Girl, with Gaussian noise. (d) Airport, original. (e) Airport, with BSC noise. (f) Airport, with Gaussian noise.

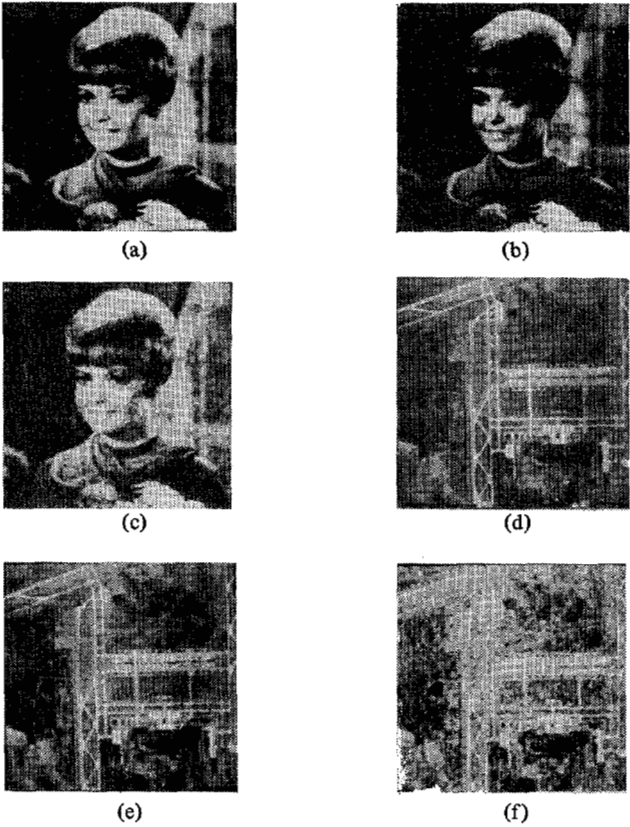


Fig. 2. The six images of Fig. 1 after 3 × 3 median filtering.

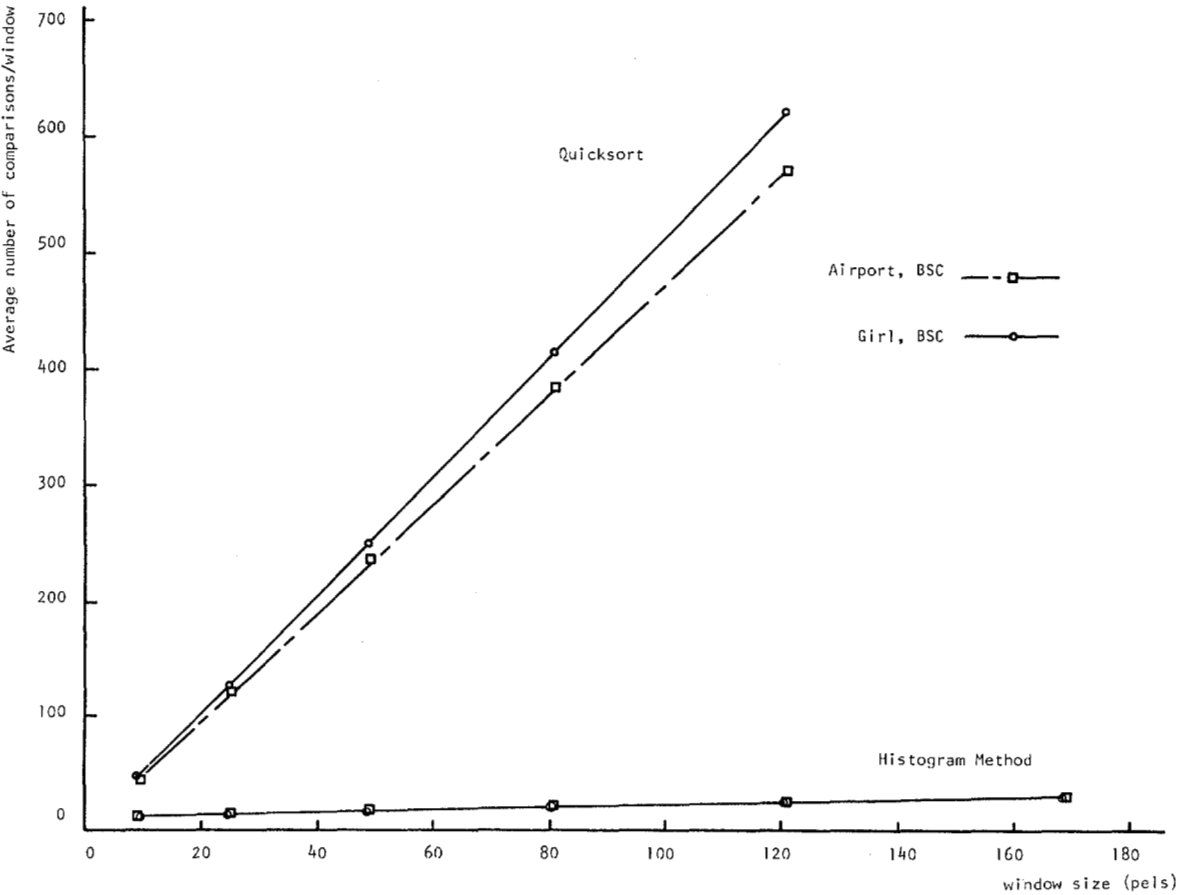


Fig. 3. Number of comparisons per window plotted against window size. The images filtered were girl with BSC noise and airport with BSC noise.

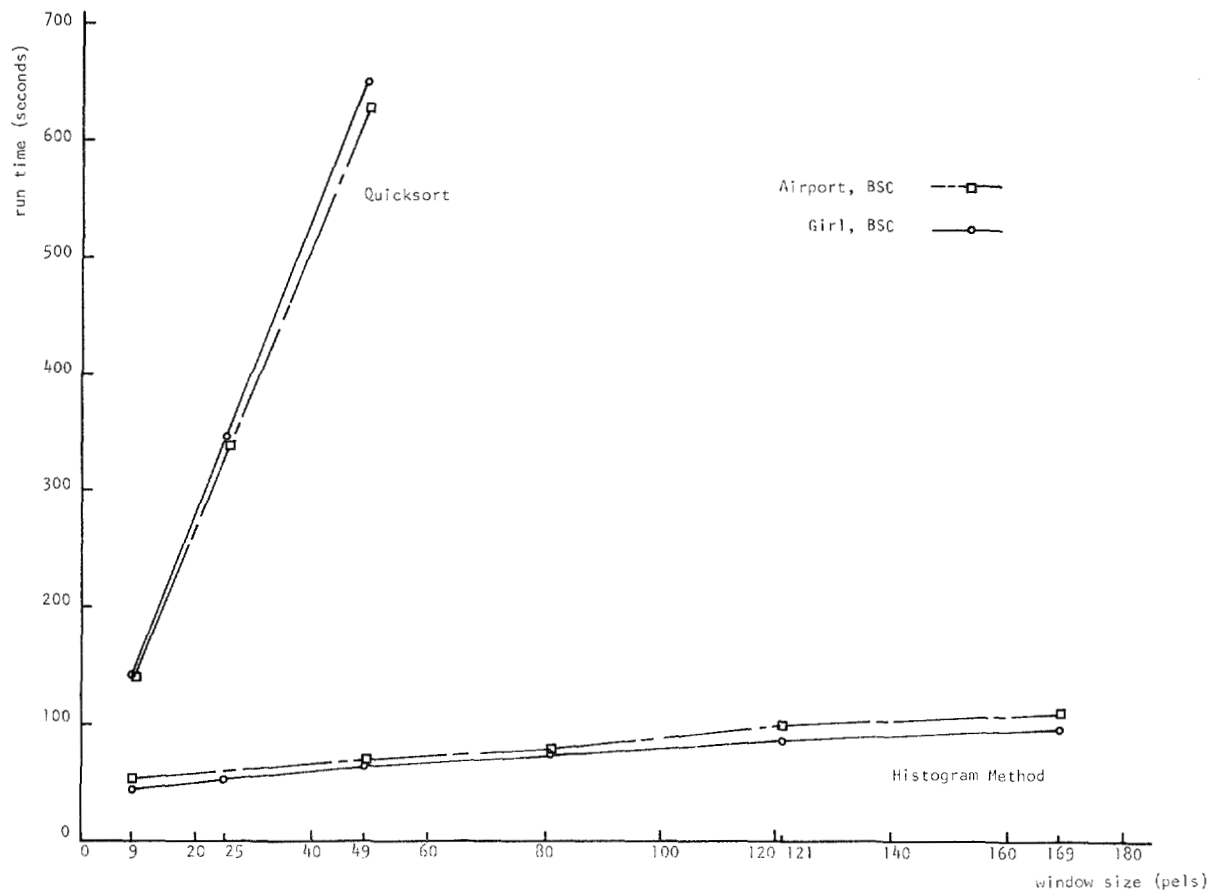


Fig. 4. Computer time per image plotted against window size. The images filtered were girl with BSC noise and airport with BSC noise.

TABLE I
NUMBER OF COMPARISONS PER WINDOW (GIRL—IMAGES 1-3)

Window Size	Histogram Method			Quicksort		
	Girl	Girl BSC	Girl Gaussian	Girl	Girl BSC	Girl Gaussian
3 × 3	11.52	11.70	16.01	47.50	47.03	46.22
5 × 5	14.68	14.74	16.37	130.69	127.70	123.34
7 × 7	18.20	18.22	18.99	256.07	249.27	237.72
9 × 9	21.90	21.89	22.29	430.80	414.51	386.06
11 × 11	25.67	25.65	25.85	643.88	622.72	569.23
13 × 13	29.48	29.46	29.56	923.19	876.56	792.98

TABLE II
COMPUTER TIME IN SECONDS (GIRL—IMAGES 1-3)

Window Size	Histogram Method			Quicksort		
	Girl	Girl BSC	Girl Gaussian	Girl	Girl BSC	Girl Gaussian
3 × 3	41.9	45.4	53.1	180.4	143.7	155.8
5 × 5	52.9	52.9	56.4	443.32	347.5	372.7
7 × 7	64.4	66.4	65.6	833.3	651.0	690.5
9 × 9	74.9	76.2	75.9	1087.4	1053.3	
11 × 11	86.2	86.9	86.5			
13 × 13	97.3	98.4	99.2			

TABLE III
NUMBER OF COMPARISONS PER WINDOW (AIRPORT—IMAGES 4–6)

Window Size	Histogram Method			Quicksort		
	Airport	Airport BSC	Airport Gaussian	Airport	Airport BSC	Airport Gaussian
3 × 3	12.84	13.17	16.7	46.37	46.19	45.54
5 × 3	11.06	11.3				
7 × 3	10.32	10.47				
5 × 5	14.95	15.10	16.56	124.26	123.28	121.15
3 × 5		16.82				
3 × 7		20.50				
7 × 7	18.10	18.15	18.94	239.24	236.83	233.03
9 × 9	21.53	21.57	22.06	388.35	385.03	378.72
11 × 11	25.23	25.24	25.56	574.01	571.09	559.37
13 × 13	28.95	28.96	29.21	808.00	802.86	775.48

TABLE IV
COMPUTER TIME IN SECONDS (AIRPORT—IMAGES 4–6)

Window Size	Histogram Method			Quicksort		
	Airport	Airport BSC	Airport Gaussian	Airport	Airport BSC	Airport Gaussian
3 × 3	50.0	52.8	69.1	156.5	142.1	154.8
5 × 3	44.3	42.0				
7 × 3	42.3	42.8				
5 × 5	56.4	59.5	72.5	378.3	339.1	368.5
3 × 5		63.1				
3 × 7		76.1				
7 × 7	67.5	67.4	83.3	628.8	627.9	680.6
9 × 9	79.7	78.0	96.1		995.5	1083.3
11 × 11	90.8	101.1	109.0			
13 × 13	101.0	112.7	121.8			

per window (i.e., per output point) and the computational time per image are measured. The results are listed in Tables I–IV. The results for the two images with BSC noise are also plotted in Figs. 3 and 4. As we can see, the histogram method is much faster than the Quicksort method. According to these results, for median filtering with an $n \times n$ window, the computer time required for the first is approximately $O(n)$, and for the second $O(n^2)$. Some of the nonsquare rectangular windows are also used. For an $m \times n$ window (moving horizontally), the computer time required is approximately $O(n)$.

IV. ANALYSIS

We show that for the fast median filtering algorithm, the average number of comparisons per window is approximately

$$2n + \overline{|d|} + 1.5 + 0.5p_0$$

where

$n \times n$ = square window size or $m \times n$ = rectangular window size

p_0 = probability of d being zero in one picture

and

d = horizontally adjacent pel gray level difference of the median filtered picture

$\overline{|d|}$ = the average of absolute value of d .

Proof: From the procedure running *mdn* in the *pidgin* Algol algorithm, we see that 1) exactly $2n$ comparisons are made in the *for* statement, and 2) number of comparisons and d are related as shown in the following table for the *if* statement.

d	number of comparisons
⋮	⋮
-3	4
-2	3
-1	2
0	2
1	3
2	4
3	5
⋮	⋮

Assuming d is symmetrically distributed (confirmed in the experiment for the chosen images), we then have the table

$ d $	average number of comparisons
0	2
1	2.5
2	3.5
3	4.5
⋮	⋮

Let \bar{c} be the average number of comparisons and p_i be the probability that $|d| = i, i = 0, 1, \dots, 255$. Then

$$\begin{aligned}
 \bar{c} &= 2p_0 + 2.5p_1 + 3.5p_2 + 4.5p_3 + \dots + 256.5 p_{255} + 2n \\
 &= 0.5p_0 + \sum_{i=0}^{255} (i + 1.5)p_i + 2n \\
 &= 0.5p_0 + 1.5 + \sum_{i=0}^{255} i p_i + 2n \\
 &= 0.5p_0 + 1.5 + \overline{|d|} + 2n.
 \end{aligned}
 \tag{1}$$

Q.E.D.

Therefore, \bar{c} is loosely upper bounded by $2n + \overline{|d|} + 2$ and lower bounded by $2n + \overline{|d|} + 1.5$. Clearly, this algorithm takes advantage of the fact that for typical images the value of $\overline{|d|}$ is small. The value of $\overline{|d|}$ is less than 10 for the images we used. Hence, approximately only $(2n + 10)$ comparisons are needed on the average. Equation (1) agrees well with the experimental results reported in Section III.

Note: This analysis ignores the comparisons made in finding the median of the first window in each line. That, on average, amounts to $\overline{mdn}/(q - m)$ comparisons per window for an $q \times r$ image, an $m \times n$ filtering window, and \overline{mdn} being the average median of the first windows in each line. This additional number of comparisons can be significant compared to (1) for large number of gray levels and small image size in the horizontal direction. This effect can be easily eliminated by applying the proposed algorithm vertically at the end of each line and scanning alternate lines in reverse direction.

REFERENCES

- [1] J. W. Tukey, *Exploratory Data Analysis* (preliminary ed.). Reading, MA: Addison-Wesley, 1971.
- [2] —, *Exploratory Data Analysis*. Reading, MA: Addison-Wesley, 1977.
- [3] L. R. Rabiner, M. R. Sambur, and C. E. Schmidt, "Applications of a nonlinear smoothing algorithm to speech processing," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-23, pp. 552-557, Dec. 1975.
- [4] N. S. Jayant, "Average and median-based smoothing techniques for improving digital speech quality in the presence of transmission errors," *IEEE Trans. Commun.*, vol. COM-24, pp. 1043-1045, Sept. 1976.
- [5] W. K. Pratt, "Median filtering," in *Semiannual Report, Image Processing Institute, Univ. of Southern California*, Sept. 1975, pp. 116-123.
- [6] B. R. Frieden, "A new restoring algorithm for the preferential enhancement of edge gradients," *J. Opt. Soc. Amer.*, vol. 66, pp. 280-283, 1976.
- [7] S. G. Tyan, "Fixed points of running medians" (unpublished report), Dep. Elec. Engr. Electrophysics, Polytechnic Inst. of New York, Brooklyn, NY, 1977.
- [8] B. Justusson, "Statistical properties of median filters in signal and image processing" (unpublished report), Math. Inst., Royal Inst. of Technology, Stockholm, Sweden, Dec. 1977.
- [9] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974, p. 101.