Pattern Recognition Letters 4 (1986) 269-272
North-Holland

September 1986

Fast percentile filtering

R.P.W. DUIN, H. HARINGA and R. ZEELEN
Department of Applied Physics, Delft University of Technology, Delft, Netherlands

Received September 1985
Revised 13 October 1985

Abstract: Four algorithms for percentile filtering are compared. The ways they can be implemented in hardware and software, their flexibility and their speed are taken into account.

Key words: Image processing, non-linear filtering, percentile filtering, rank order filtering, image processing hardware.

1. Introduction

Percentile filtering is a well known non-linear technique for filtering images (and other signals). Each pixel in the image is replaced by the value of that point in the histogram of its neighbour pixels that corresponds with a preset percentile. Percentile filtering includes minimum and maximum filtering (grey-value erosion and dilation) and median filtering. It is very suitable for background estimation and removal of outlyers without degrading the image. A review of properties and applications has been given by Hodgson et al. (1985).

In the discussions we will assume that the neighbourhood of a pixel is defined by a square window of size N. However, results for other shapes may be derived directly. Pixels are processed in a rasterscan way, resulting in an overlap of successive windows. If the pixels are ordered with respect to their value then a preset percentile (0–100%) corresponds with a certain rank number, the 'percentile rank number' with a range from 1 to N. (For this reason, percentile filters are sometimes called rank order filters.) We will call the pixel corresponding to the percentile the 'percentile pixel'. Finally, we will assume that pixels are given by K-bit data words.

In this paper we will discuss some algorithms for percentile filtering of images. As images tend to contain a lot of data, algorithms have to be fast. Special purpose hardware is often investigated for speeding up image processing. An important feature of algorithms is therefore the suitability for implementation in parallel hardware. Flexibility in window size should not be sacrified as different applications may demand different sizes.

2. Window search

The most direct way of finding the percentile pixel value is to sort all pixels in the window on their value. Hereafter, the percentile pixel value follows immediately from the percentile rank number. The complexity of this procedure is determined by the sorting algorithm. This is for the worst case of order $N*N$, but for random input orders, $N \log N$ can be expected.

As we are not interested in a complete ranking, but only in the value of the pixel with the percentile rank, the above number may be further reduced as follows:

0. Put the pixels under investigation (initially all) in a bucket A. Let their number be M (initially

$M=N$). Determine the percentile rank number $M_p: 1 \leq M_p \leq M$.

1. Select an arbitrary pixel from A.
2. Compare the selected pixel with all pixels in A. If a pixel is smaller or equal to the selected one, put it in a temporary bucket A_s. Let M_s be the number of pixels in $A_s: 1 \leq M_s \leq M$.
3. If $M_s \geq M_p$ then proceed with $A=A_s$ and $M=M_s$, otherwise with $A=A-A_s$, $M=M-M_s$ and $M_p=M_p-M_s$. Empty A_s.
4. If $M>1$ then go to 1, otherwise stop.

The pixel in A is now the percentile pixel. This procedure comes never to an end if two or more pixels have the percentile pixel value. This can be detected if M does not change while $M>1$. In that case an arbitrary pixel out of A can be used as the percentile pixel. The number of comparisons needed for this procedure is on the average $N+N/2+N/4+\cdots \simeq 2N$.

The following two observations can be made on the WINDOW SEARCH algorithm:

(a) The ranking procedure may be further simplified by using only the pixel bits that are really needed.

(b) Succesive windows overlap considerably. By treating each window separately a lot of ordering work is repeated several times.

Of the following three algorithms two ones try to take into account one of these aspects and the third one combines both.

3. Histogram search

A widely used algorithm for percentile filtering is given by Huang et al. (1979). It consists of an initialisation in which the histogram is computed of the window defining the neighbourhood of the first pixel. This is followed by an updating procedure for the histograms of the next windows. As windows corresponding with successive pixels differ only on the window boundary, this part of the algorithm is of order $\sqrt{N}$.

The point in the histogram that corresponds with the desired percentile value has for the first window to be found by a search starting from one side of the histogram and can thereafter be updated for successive window histograms. This has to be prepared by updating the rank of the previous percentile pixel value simultaneously with the histogram updating. The number of steps needed for this procedure is data dependent, somewhere between 0 and 2^K-1. Huang reports an experiment in which for a number of 8-bit images this takes, on the average, less than 10 steps for each update.

The total procedure is at most of order $\sqrt{N}+2^K$. The algorithm as formulated by Huang can not easily be implemented by parallel hardware: it is almost completely sequential.

4. Binary window search

Danielsson (1981) proposed a completely different approach directed to hardware parallellisation. In his algorithm each window is treated separately (no updating) as in the WINDOW SEARCH, but now bitplane for bitplane is analysed in a binary search. The result is the bit combination of the desired percentile pixel value. An algorithm, somewhat modified by us, that performs this task is the following:

0. Put the pixels under investigation (initially all) in a bucket A. Let their number be M (initially $M=N$). Determine the percentile rank number $M_p: 1 \leq M_p \leq N$. Set the bitplane under investigation, j, on the most significant bitplane: $j=1$.
1. Move the pixels in A with bit-j equal to zero to a temporary bucket A_s. Let M_s be the number of pixels in A_s.
2. If $M_s \geq Mp$ then proceed with $A=A_s$ and $M=M_s$, otherwise with $A=A-A_s$, $M=M-M_s$ and $M_p=M_p-M_s$. Empty A_s.
3. $j=j+1$,
4. If $j \neq K$ then go to 1, otherwise take an arbitrary pixel from A and stop.

Like in the WINDOW SEARCH only $2N$ steps (here bit-inspections) are needed on the average.

In the scheme given by Danielsson no buckets are used but masks, so in each iteration the bits of all pixels are inspected, resulting in $K*N$ steps. The

nice feature of it is, however, that it may easily be parallised for all pixels, resulting in a time complexity of K. Also bitplane parallellisation seems to be possible, using a pipeline approach. For that case the time-complexity is just 1: in each step a new percentile value is produced. The hardware complexity, however, may be very large: $N*K$ bit-wide logic. The BINARY WINDOW SEARCH is nicely implementable on bit serial SIMD machines like the CLIP, DAP, MPP, GRID and GAPP.

5. Binary histogram search

Another idea put forward by Danielsson (1981) and independently by Ataman et al. (1980) is to collect a set of histograms

$$H_1, H_2, \ldots, H_i, \ldots, H_k$$

where H_i is based on the i most significant bits of the input data only. H_i has a length of 2^i. This idea can be combined with the algorithm of Huang (1979) into a binary search on histograms with updating. The algorithm can be summarised as follows:

1. Compute for the first window the K histograms
$$H_1, H_2, \ldots, H_i, \ldots, H_k.$$
2. Find the percentile pixel value of a window by a binary search:
Let the percentile rank number be r.
Set $j=0$ and $a=0$.
Do for $i=1$ to K
$j=j*2$;
if $r>a+H(j)$ then $(j=j+1,\ a=a+H_i(j))$;
Enddo;
Now j equals the percentile pixel value.
3. If all windows are treated, stop; else, goto the next window and update the K histograms H_i.
4. Goto step 2.

For this algorithm, K histograms have to be updated. The search part consists of a fixed number of K steps. The total complexity is therefore of the order of $K*\sqrt{N}+K$. This can easily be reduced to $\sqrt{N}+K$ by using parallel hardware for the histogram bookkeeping. This has been done by the authors of this paper (Duin et al., 1985; Zeelen and Haringa, 1985), yielding a fast and flexible hardware set-up.

6. Discussion

We will compare here the four algorithms described above. Finally a short characteristic of the use of each of them will be given.

The computational effort needed for the HISTOGRAM SEARCH is of lower order ($\sqrt{N}$) than the effort needed for the WINDOW SEARCH (N). However, the additional overhead needed for updating the percentile pixel in the histogram may be so large that for small windows the WINDOW SEARCH has to be preferred. Where the break even point lies depends on the number of bits, the characteristics of hardware and software and even on the data. For windows as small as 3*3 ($N=9$) and 8-bit pixels usually the WINDOW SEARCH has to be preferred.

Both algorithms are not very suitable for implementing in parallel hardware. Moreover, for that case the WINDOW SEARCH may be replaced by the BINARY WINDOW SEARCH, using 1-bit logic instead of K-bit logic. For both the number of comparisons that is needed is on the average $2N$. If the scheme proposed by Danielsson (1981) is used for the BINARY WINDOW SEARCH the number of steps is $K*N$, but by parallellisation it can be reduced to K, or even to 1. This solution seems particulary suited for a VLSI-design.

The use of histograms instead of a direct search in the window has for the case of a hardware design, the advantage that it offers flexibility for the window size: the same hardware may be used for windows with all kinds of shapes and sizes. Windows of 15*15, or even 30*30 should be no problem. The disadvantage that for histograms the number of bits has to be fixed is compared to that rather small. In many applications windows of several sizes may be needed, but always combined with the same bit accuracy. However, extreme pixel accuracies ($K>16$ bit) are hardly possible in combination with histograms as they become to large to be stored for the present state of technology.

The BINARY HISTOGRAM SEARCH needs, compared to the HISTOGRAM SEARCH K histograms instead of one. The search parts of the algorithms need a fixed number of K steps, versus a data dependent number of maximum $2^K - 1$ steps. This last figure may for some images be very small, e.q. 10, but will in the presence of noise be of the order $2^{K'}$, in which K' is the number of bits that is effected by the noise. The BINARY HISTOGRAM SEARCH is insensitive for this. It always needs just K steps. Moreover, the histogram updating is more simple as only the histogram has to be updated and not simultaneously the rank of the previous percentile pixel value. These advantages have to be paid by the updating of K histograms instead of one. However, this can easily be parallellised in hardware.

Finally the following conclusions may be stated. The WINDOW SEARCH is suited for small windows and when implemented in hardware, of a fixed size. The HISTOGRAM SEARCH is fast for larger windows and offers flexibility in window size. It is the best algorithme that is available for software solutions. The BINARY WINDOW SEARCH offers the possibility of very fast parallel hardware, especially suited for VLSI-design. It is only feasible for small windows, and it is not flexible. The BINARY HISTOGRAM SEARCH is suited for a flexible, fast parallel hardware design, offering the possibility of large windows, but is restricted to a fixed and limited bit width for the pixel data.

Acknowledgement

The authors thank P.E. Danielsson for his comments on an earlier version of this paper.

References

Ataman, E., V.K. Aatre and K.M. Wong (1980). A fast method for real time median filtering. *IEEE Trans. Acoustics, Speech and Signal Processing*, 28(4), 415–421.

Danielsson, P.E. (1981). Getting the median faster. *Comput. Graphics Image Processing* 17, 71–78.

Duin, R.P.W., H. Haringa and R. Zeelen (1985). A hardware design for fast 2-D percentile filtering. *Proc. Internat. Soc. Opt. Eng.* 596.

Hodgson, R.M., D.G. Bailey, M.J. Naylor, A.L.M. Ng, and S.J. McNeil (1985). Properties, implementations and applications of rank filters. *Image Vision Comput.* 3(1), 4–14.

Huang, T.S., G.J. Yang and G.Y. Tang (1979). A fast two-dimensional median filtering algorithm. *IEEE Trans. Acoustics, Speech Signal Processing* 27(1), 13–18.

Zeelen, R. and H. Haringa (1985). Hardware for percentile filtering (in Dutch). Thesis, Dept. of Applied Physics, Delft University of Technology, Delft, Netherlands.