An Alternate Algorithm for (3x3) Median Filtering of Digital Images

Satinderjit Singh
HoD, Computer Sc.deptt.
GGNIMT, Ludhina, Punjab (India)
satinderjits@in.com

ABSTRACT

Median filtering is a commonly used technique in image processing. The main problem of the median filter is its high computational cost (for sorting N pixels, the temporal complexity is O(N·log N), even with the most efficient sorting algorithms). When the median filter must be carried out in real time, the software implementation in general-purpose processors does not usually give good results. This Paper presents an efficient algorithm for median filtering with a 3x3 filter kernel with only about 9 comparisons per pixel using spatial coherence between neighboring filter computations. The basic algorithm calculates two medians in one step and reuses sorted slices of three vertical neighboring pixels. An extension of this algorithm for 2D spatial coherence is also examined, which calculates four medians per step.

Keywords

Linear filtering, nonlinear filter, Median filter, bubble sort, Quick sort, computational cost, Algorithm complexity.

1. INTRODUCTION

Image processing is a very important field within industrial automation, and more concretely, in the automated visual inspection. In these applications, the main challenge normally is the requirement of real-time results. Image processing is a very important field within industrial automation, and more concretely, in the automated visual inspection. For example automatically analyzing predetermined features of manufactured parts on an assembly line to look for defects and process variations. In these applications, the main challenge normally is the requirement of real-time results . On the other hand, in many of these applications, the acquired images must pass through a stage of image preprocessing in order to remove distracting and useless information from the images. For example, the existence of impulsive noise in the images is one of the most habitual problems

Median filter is the nonlinear filter more used to remove the impulsive noise-from an image. Furthermore, it is a more robust method than the traditional linear filtering, because it preserves the sharp edges. Furthermore, it is a more robust method than the traditional linear filtering, because it preserves the sharp edges. Typically used on signals that may contain outliers skewing the usual statistical estimators, it is usually considered too expensive to be implemented in real-time or CPU-intensive applications. The median value is determined by placing the brightnesses in ascending order and selecting the centre value.

The obtained median value will be the value for that pixel in the output image. Figure shows an example of the median filter application, as in this case, habitually a 3x3 median filter is used.

Median filter is a spatial filtering operation, so it uses a 2-D mask that is applied to each pixel in the input image. To apply the mask means to centre it in a pixel, evaluating the covered pixel brightness and determining which brightness value is the median value. The median filter is normally used to reduce noise in an image, somewhat like the mean filter. However, it often does a better job than the mean filter of preserving useful detail in the image.

2. BASICS OF MEDIAN FILTERING

Like the mean filter, the median filter considers each pixel in the image in turn and looks at its nearby neighbors to decide whether or not it is representative of its surroundings. Instead of simply replacing the pixel value with the mean of neighboring pixel values, it replaces it with the median of those values. The median is calculated by first sorting all the pixel values from the surrounding neighborhood into numerical order and then replacing the pixel being considered with the middle pixel value. (If the neighborhood under consideration contains an even number of pixels, the average of the two middle pixel values is used.) Figure 1 illustrates an example calculation.

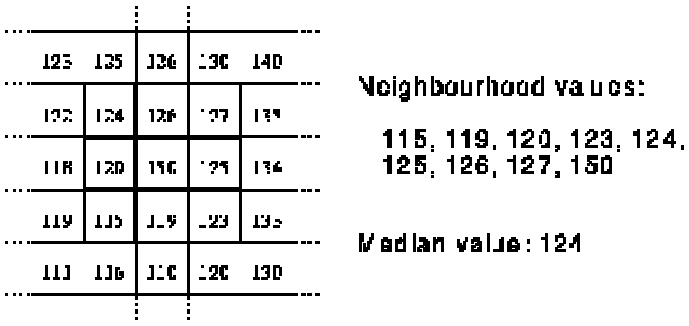
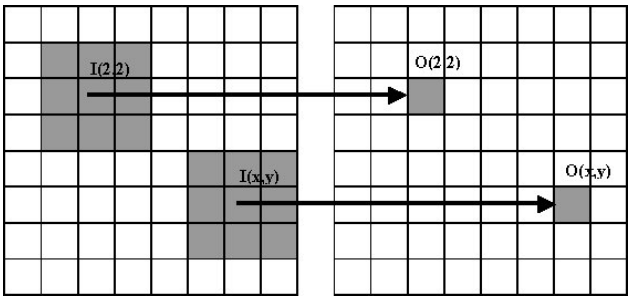


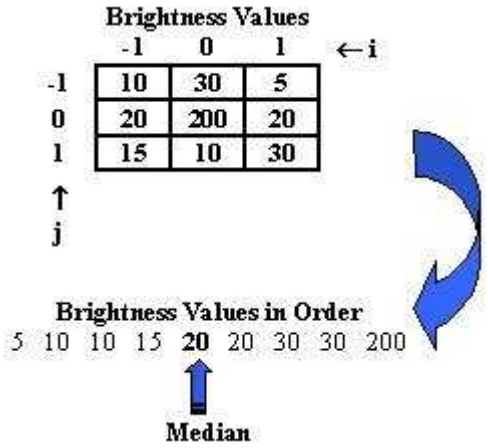
Figure 1 Calculating the median value of a pixel neighborhood.

As can be seen the central pixel value of 150 is rather unrepresentative of the surrounding pixels and is replaced with the median value: 124. A 3×3 square neighborhood is used here --- larger neighborhoods' will produce more severe smoothing.

Figure below presents the concept of spatial filtering based on a 3x3 mask, where I is the input image and O is the output image.



The median value is determined by placing the bright nesses in ascending order and selecting the centre value. The obtained median value will be the value for that pixel in the output image. Figure shows an example of the median filter application, as in this case, habitually a 3x3 median filter is used.



3. ADVANTAGES OVER MEAN FILTER

By calculating the median value of a neighbourhood rather than the mean filter, the median filter has two main advantages over the mean filter:

I. The median is a more robust average than the mean and so a single very unrepresentative pixel in a neighbourhood will not affect the median value significantly.

II. Since the median value must actually be the value of one of the pixels in the neighbourhood, the median filter does not create new unrealistic pixel values when the filter straddles an edge. For this reason the median filter is much better at preserving sharp edges than the mean filter

In general, the median filter allows a great deal of high spatial frequency detail to pass while remaining very effective at removing noise on images where less than half of the pixels in a smoothing neighborhood have been effected.

4. THE OPTIMIZATION PROBLEM

The main problem of the median filter is its high computational cost (for sorting N pixels, the temporal complexity is O(N·log N), even with the most efficient sorting algorithms). When the median filter must be carried out in real time, the software implementation in general-purpose processors does not usually give good results. It is usually considered too expensive to be implemented in real-time or CPU-intensive applications. One of the major problems with the median filter is that it is relatively expensive and complex to compute. To find the median it is necessary to sort all the values in the neighbourhood into numerical order and this is relatively slow, even with fast sorting algorithms such as quicksort. The basic algorithm can however be enhanced somewhat for speed. A common technique is to notice that when the neighbourhood window is slid across the image, many of the pixels in the window are the same from one step to the next, and the relative ordering of these with each other will obviously not have changed. Clever algorithms make use of this to improve performance.

4.1 Background work

The median filter is often used to remove "shot" noise, pixel dropouts and other spurious features of single pixel extent while preserving overall image quality [Huang 1981] [Paeth 1986a] [Paeth 1986b]. In contrast, low pass filters would only blurr the noise instead of removing it. An efficient algorithm to determine the median is desired, because this operation often has to be repeated millions of times for filtering large images.

One simple approach, which is often found in image processing textbooks, is to calculate the 3x3 median using a simple sorting algorithm, like bubble sort or quicksort, and pick the 5^{th} element after the sorting. An improvement to this simple technique is only to sort until the 5th element is determined. For example a modified bubble sort can be used to sort until the 5th element. This approach yields 30 comparisons for one median calculation.

A better approach is published in the first Volume of the Graphics Gems series by Paeth [Paeth 1990]. This approach is based on a successive minmax-elimination: the minimum and the maximum of the first six elements are determined and eliminated. Then the 7th element is added to the remaining four of the first pass and the minimum and the maximum of the five elements are determined and eliminated. This scheme is repeated until the 9th element is added to the remaining two and the minmax-elimination results in the median of all nine elements. This algorithm needs 20 comparisons per median. The drawback, that the algorithm does not use spatial coherence, can easily be remedied: Simply calculate two neighbouring medians in one step, where the first minmax-elimination is computed from the common six elements and can be used for both medians. This improvement would result in a better performance using only 16.5 comparisons per median.

A comparison of other median filtering algorithms can be found in [Juhola et al. 1991], but these techniques are not optimized for the common 3x3 kernel.

The algorithm proposed here uses coherence information between neighbouring median calculations more efficiently and therefore needs only a maximum of 9.5 comparisons per median. The average number of comparisons is even a little smaller.

5.ALGORITHMIC CONCEPT

The proposed algorithm computes two neighbouring medians in one step. Let us assume that the neighbouring medians we want to calculate are horizontally adjacent to each other. This means,

if the first median is at position (x,y), the second is at (x+1,y). Therefore we have to look at the 4x3 pixels within the rectangle (x-1,y-1)-(x+2,y+1). Let us subdivide these points into four vertical slices each containing three pixels.

The **first step** of the algorithm sorts the pixels within the slices. Only the last two slices have to be sorted, because the first two were already sorted during the calculation of the medians calculated before. Only when the first two medians in a row are computed, the first two slices also have to be determined as well. Therefore this step consumes a maximum of 6 comparisons for a median calculation in the non border case.

The **second step** sorts the second and third slice according the merge sort algorithm. Because of time considerations this should be done with nested IF statments instead of a conventional loop. This adds up to a maximum of 5 comparisons for this step.

The **third step** computes the first median with a modified merge sort of the first slice and the sorted middle six elements and the second median from the sorted middle six elements and the fourth slice. Since we are not interested in the sorting of the elements, but only in the median, the merge sort is modified so that it does not store the elements in the sorted order, but only remembers which rank it is now processing and which are the two possible elements , which could have the next rank. Also the first and the last element of the sorted six elements can not be a 3x3 median: The median of nine element has rank 5 therefore it has four elements, that are lower or equal the median and four that are higher or equal. Since the first of the six elements has only possibly three elements - the elements from the compared slice - which are lower or equal, this element can not be the median. The proof for the last element is analog. Instead of computing the median via the determination of the rank five element of a sorted slice and a sorted list of six elements, it can be computed via the rank four element of a sorted slice and the middle four elements of the sorted six elements. For efficiency reasons the modified merge sort should be computed with nested IF statements rather than with loops. This step needs **maximal** two times 4 comparisons. Summing up the maximal comparisons of the three steps gives 19 comparisons per two medians or 9.5 comparisons per median in the worst case. The average of an efficient implementation is about 9.0 comparisons.

5.1 Extension using 2D coherence

An extension to the proposed algorithm uses 2D coherence through the computation of four medians arranged in a 2x2 grid, instead of the computation of only two neighbouring medians per step. The extended algorithm handles these four medians in two times two medians using our proposed 1D coherence algorithm. The only difference lies in the computation of the sorted slices (step one). Instead of computing them independently for the upper two medians and the lower two, coherence is used: For each slice for the upper part we have an overlap of two elements with one slice of the lower part. The idea is to sort these two elements first and use it for the sorting of both slices. This improvement saves an additional 1/2 comparison yielding 9.0 comparisons per median in the worst case.

6. CONCLUSION

This paper presented an algorithm how a 3x3 kernel median filtering of a raster image can efficiently be implemented using spatial coherence between neighbouring median calculations. The 2D extension to the algorithm showed better theoretical but depending on the hardware little better to little worse practical results.

7. REFERENCES

[1] Paeth 1990 A. W. Paeth. Median finding on a 3x3 Grid. In Andrew Glassner, editor,Graphic Gems, pages 171-175. Academic Press, Boston, 1990.

[2] [Paeth 1990] A. W. Paeth. Median finding on a 3x3 Grid. In Andrew Glassner, editor,Graphic Gems, pages 171-175. Academic Press, Boston, 1990.

[3] L. Yin, R. Yang, M. Gabbouj, Y. Neuvo. “Weighted Median Filters: A Tutorial”, IEEE Trans. on Circuits and Systems, 43(3), pp. 157-192 (March 1996).

[4] T.S.Huang, G.J.Yang, G.Y.Tang, A fast two dimensional median filter algorithm, IEEE transactions on acoustics, speech and signal processing, Vol ASSP 27 No 1, Feb 1979.

[5] Aho, Hopcroft, Ullman, The design and analysis of computer algorithms (p 102).

[6] The Image Processing Handbook, John C. Russ, CRC Press (second edition 1995)