Efficient 3x3 Median Filter Computations

Manfred Kopp
Institute of Computer Graphics
Technical University of Vienna
Karlsplatz 13/186-2
A-1040 Wien, Austria
m.kopp@ieee.org

Werner Purgathofer
Institute of Computer Graphics
Technical University of Vienna
Karlsplatz 13/186-2
A-1040 Wien, Austria
purgathofer@cg.tuwien.ac.at

Abstract

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 neighbouring filter computations. The basic algorithm calculates two medians in one step and reuses sorted slices of three vertical neighbouring pixels.

An extension of this algorithm for 2D spatial coherence is also examined, which calculates four medians per step. Even though theoretical results would yield 5% performance increase compared to the basic algorithm, experimental results showed less significant increase or even worse performance dependent on the hardware.

Keywords: image processing, filtering, 3x3 median kernel, spatial coherence

1 Introduction

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 5th 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.

2 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.

3 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.

4 Experimental Results

Table 1 shows time comparisons of median filter algorithms for a 1024x768 raster image on different hardware. *Basic* is the basic algorithm described in chapter 2, *2D-Coherence* uses the extension described in chapter 3 and *Minmax* refers to the minmax elimination algorithm described in [Paeth 1990]. The tests were computed with optimizing compilers on three computers with different hardware: *R3000* refers to a Silicon Graphics Personal Iris 4D/35 with a MIPS R3000 processor, *R4000* denotes a Silicon Graphics Indigo with a MIPS R4000 processor and *486DX2-50* is a PC with a Intel 486DX2-50 processor running the Linux operating system. The relative timings are based on the *Basic* algorithm.

	R3000		R4000		486DX2-50	
Basic	1335 ms	0	688 ms	0	1465 ms	0
2D-Coherence	1454 ms	+8.9%	691 ms	+0.4%	1447 ms	-1.2%
Minmax	2510 ms	+88.0%	1281 ms	+86.2%	5852 ms	+299.4%

Table 1: Calculation times of median filter computations for a 1024x768 raster image

Even though the *2D-Coherence* algorithm has a 5.3% better theoretical performance than the *Basic* algorithm, the actual performance was worse on the MIPS processor based machines and only 1.2% better on a Intel processor based machine. Reasons for this results may be greater storage reqirements for register variables and for the code caching. Since the results are machine and compiler dependent, the *Basic* and the *2D-Coherence* algorithm should both be considered for a verry fast 3x3 median filter.

5 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.

Bibliography

[Huang 1980] T. S. Huang, ed. Two-Dimensional Digital Signal Processing II, Transforms and Median Filters, in Topics in Applied Physics (43). Springer-Verlag Berlin.

[Juhola et al. 1991] Martti Juhola, Jyrki Katajainen, and Timo Raita. Comparison of Algorithms for Standard Median Filtering. IEEE Transactions on Signal Processing, 39(1):204-208

[Paeth 1986a] A. W. Paeth. Design and Experience with a Generalized Raster Toolkit, Proceedings Graphics Interface '86, Canadian Information Processing Society, Vancouver.

[Paeth 1986b] A. W. Paeth. The IM Raster Toolkit - Design, Implementation and Use, University of Waterloo Technical Report CS-86-65

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