FAST RANK ALGORITHMS WITH MULTISCALE HISTOGRAMS LAZY UPDATING[1]

M. Storozhilova[2], D. Yurin [3]

[2,3] Laboratory of Mathematical Methods of Image Processing
Faculty of Computational Mathematics and Cybernetics
Lomonosov Moscow State University
Leninskie Gory, Moscow 119991, Russia.
mariastorozhilova@gmail.com[2], yurin@cs.msu.su[3]

Rank algorithms propose good solutions for image smoothing and impulse noise removing. In our previous work on the subject we have proposed methods for fast computations of ε_V and KNV neighborhood average based on multiscale histograms. In this paper a new method for multiscale histograms fast updating is described and analyzed. Performance comparisons of the combined methods with fastest known median filtering algorithm are given. We have achieved the processing speed for ε_V and KNV neighborhood average algorithms only a few times slowly than the fastest known median filtering algorithm.

Introduction

Rank algorithms [5] do not blur the edges of objects and are well suitable for the impulse noise suppression. But currently only median filtering is frequently used, as the most quickly and easily implemented algorithm. Methods for calculating of ε_V or KNV neighborhood average were used seldom due to their computational complexity. In [3] new approaches for fast rank algorithms based on multiscale histograms were proposed. But the multiscale histograms updating and summation algorithms were based on the classical [1] Huang method. In this paper a new approach for histogram construction and updating is proposed.

Our new method is based on column histograms which were considered in [2]. We adopted this approach for multiscale histograms maintaining. Thus using [3] the resultant algorithm is applicable not only for median filtering, but also for fast ε_V and KNV neighborhood average calculating. The columns summation is performed only for those histogram's detail levels and entries which are required by the rank algorithm and only immediately when required. We remember when each histogram entry was updated and when need to update it for the next time we start from the saved position. In [2] such approach was applied to the whole coarse level histogram (16 entries) and required to update 16-entries fine level block. I.e. updating is performed totally for 32 histogram entries. In our approach we update no more than 8 entries for median, 16 for ε_V and 32 (not the same as for [2]) for KNV algorithms from all detail levels.

KNV and ε_V neighborhood definitions

Let us remind some useful definitions. Let the current or central pixel is v_0 and let there is a neighbourhood of pixel v_0 containing N pixels. The rank series $\{v(r)\}$ is a one-dimensional sequence of N pixels of such neighborhood whose elements are sorted in an ascending order with the respect to their values: $\{v(r): v(r) \le v(r+1), r=0,1..N-1\}$. Pixel v rank R is the position of the element in the rank series.

Definition 1. ε_V neighborhood is a subset of pixels $\{v(r)\}$ whose values deviate from the value of the central pixel v_0 at most by predetermined quantities $-\varepsilon$ and $+\varepsilon$, r = 0..N-1:

[1] The research is done under support of Federal Target Program "Scientific and Scientific Pedagogical Personnel of Innovative Russia" in 2009-2013 and the Russian Foundation for Basic Research, project no. 09-07-92000-HHC_a.

$$\varepsilon_V(v_0) = \{v(r) : v_0 - \varepsilon \le v(r) \le v_0 + \varepsilon\} \quad (1)$$

It should be noted that averaging by ε_V neighborhood can be treated as a simplified bilateral filter [4] where bilateral filter parts depending both on distance and on pixel brightness are represented by rectangular functions instead of Gaussians.

Definition 2. *KNV*-neighborhood is a subset of a specified number K of pixels $\{v(r)\}$ whose values are nearest to the value of the central pixel v_0:

$$KNV(v_0) = \left\{ v(r) : \sum_{r=p}^{p+K-1} |v_0 - v(r)| = \min_p \right\} \quad (2)$$

Multiscale histograms

Multiscale histogram (Fig. 1) is a set of usual histograms, which are connected with each other in a special way. The number of levels is $L_{max}=\log_2(I_{max}+1)$, where I_{max} is the image's maximum intensity.

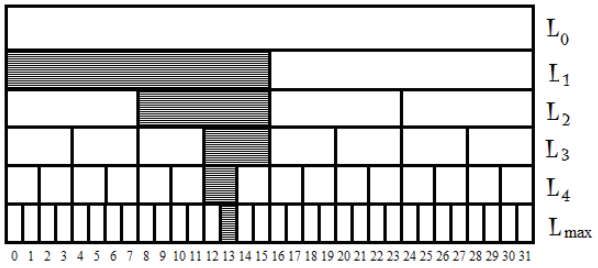

Fig. 1. Multiscale histogram.

The most detailed level (L_{max}) is the usual histogram where each element corresponds to one intensity value. Each upper (L_i) level's capacity is two times decreased and every element is the sum and the total amount of two corresponding elements on level L_{i+1}. In other words every element of higher level includes two corresponding elements of lower level. The L_0 level contains the total sum of intensities and the total number of elements in the neighborhood.

Histogram lazy updating

The classical Huang [1] approach for sliding window histogram updating includes $2r+1$ operations of addition and of subtraction per pixel (where r is the sliding window radius). In terms of multiscale histograms that means $(2r+1)\cdot\log_2(I_{max}+1)$ operations per pixel where I_{max} is number of image colors. For an image of 256 grades of gray color it will be $8\cdot(2r+1)$ operations per pixel. The main problem of this method is that the complexity of updating a histogram increases with the radius of the neighborhood and equals $O(2r+1)$.

The main idea of our lazy algorithm is based on histogram updating approach from [2]. Let w is the image width. In [2] it is proposed to maintain w histograms simultaneously – one per each column of the image and one additional kernel histogram for a sliding window. At the beginning first $r+1$ elements of the rows are added to the corresponding histogram. Then during the sliding window movement through the image, the kernel histogram updating process includes $I_{max}+1$ operations of addition and $I_{max}+1$ operations of subtraction per pixel. Column histograms are updated only with 1 addition and 1 subtraction per pixel (see Figure 2). From 2nd to the last pixel in a row we need to add one pixel below and to subtract the top pixel of the nearest right to the sliding window column histogram. For the first pixel we need to update all the first $r+1$ column histograms simultaneously.

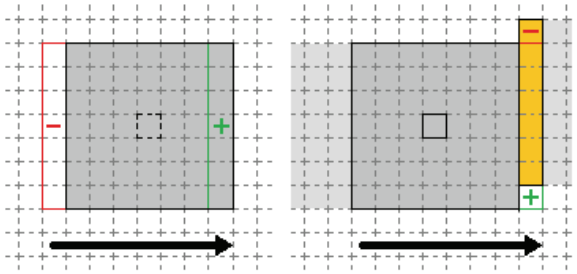

Fig. 2. a) – Huang's algorithm, n pixels must be added to and subtracted from the kernel's histogram on each step; b) – ctmf's column histogram updating method, 1 pixel must be added to and subtracted from the corresponding column histogram on each step.

It should be noted that our [3] algorithm for fast calculation of the ε_V and KNV neighborhood average requires access to no more than $2\cdot\log_2(I_{max}+1)$ and $4\cdot\log_2(I_{max}+1)$ elements of the kernel histogram per pixel. And for finding an arbitrary element in a rank series this number equals to $\log_2(I_{max}+1)$. The

[1] The research is done under support of Federal Target Program "Scientific and Scientific Pedagogical Personnel of Innovative Russia" in 2009-2013 and the Russian Foundation for Basic Research, project no. 09-07-92000-HHC_a.

new lazy histogram processing approach takes into account the fact that not all elements of the kernel multiscale histogram are involved in current algorithm's calculations. So we propose to sum up only those elements of histograms, whose values are required for processing the current pixel.

Most images consist from areas with insignificantly varying colors. That means that elements of coarse levels of multiscale histogram will not differ very much while monotonous area processing. For each row it is proposed to save the number of the column, in which this histogram element was updated for the last time. If the area is rather monotonous, this approach can noticeably increase the processing time making only 1 addition and 1 subtraction on the upper coarse levels of multiscale histogram because these levels change insignificantly on homogeneous areas. In case when the difference between the number of current column and the number of column with last update is grater then r, the algorithm performs the summation of all $2r+1$ histograms elements. The first element of each row is also handled that way. So the total number of operations is less than $2\cdot(I_{max}+1)$ histogram entries summations and subtractions per pixel. We perform the same operations for each pixel of the image, but only for the values that are needed at the moment. The computations whose results will not be used are never fulfilled. Thus, the computation time can not be larger than $O(I_{max}+1)$ which is a constant relative to window radius, but in practice it is much faster.

The following algorithm is proposed:

Algorithm. Lazy kernel histogram updating: computation of the element that is required at the moment.

Input: L – number of the histogram level for the required element;
h – multiscale histograms vector;
h[i].col – last updated column number;
r – the sliding window radius;
val – the intensity of the required element;
elem – the required histogram element;
elem.col – the current column number;
Output: *element(elem,h,val,L,r);*

1: **if** ((*elem*.col - h_L[*val*].col) > r) **then**
2: **for**(i := *val*-r; i < *val*+r; i++) **do**
3: *elem* := *elem* +h_L[i];
4: **end for**;
5: h_L[*val*].col := *elem*.col;
6: h_L[*val*] := *elem*;
7: **else**
8: *elem* := h_L[*val*];
9: **for**(i := 1; i < (*elem*.col - h_L[*val*].col); i++) **do**
10: *elem* := *elem* + h_L[*val*+r-*i*];
11: *elem* := *elem* - h_L[*val*-r-*i*];
12: **end for**;
13: **end if**;
14: The required element sum and number of pixels are calculated.

The process can be useful not only for rank algorithms implementation, but for any application that involves histogram maintaining and updating process.

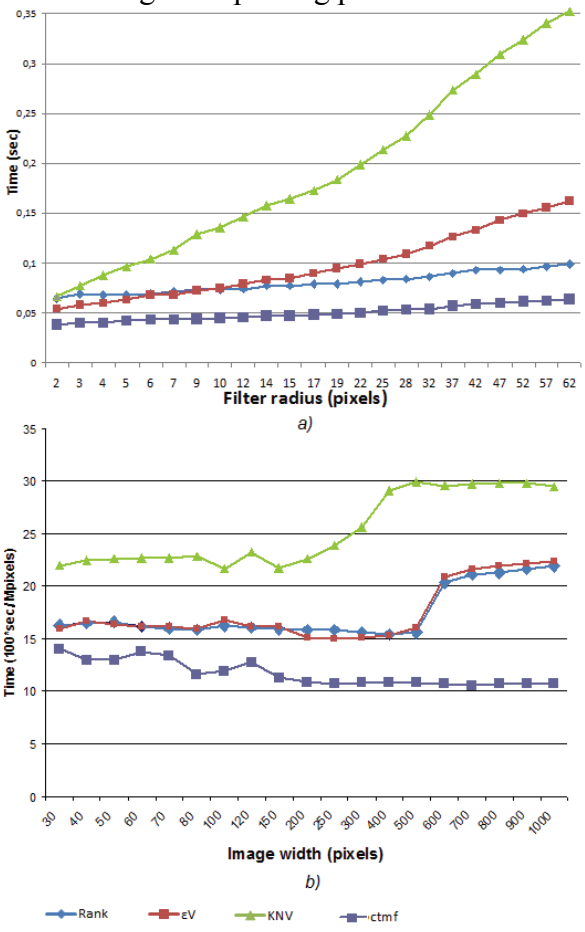

Fig. 3. a) – timings in seconds for a 375x486 pixels image; b) - execution time per megapixel for images of 1000 pixels height and different width, filter radius = 7.

[1] The research is done under support of Federal Target Program "Scientific and Scientific Pedagogical Personnel of Innovative Russia" in 2009-2013 and the Russian Foundation for Basic Research, project no. 09-07-92000-HHC_a.

Radius and image size dependences

New algorithms were compared with constant time median filtering (ctmf) algorithm [2], which implements time-saving column histogram updating method. Timing was conducted on Core i-7 2600K, 3.40GHz, Windows 7.

Fig. 3a shows the dependency between the calculation time and the radius of the sliding window. The proposed algorithm for search of an arbitrary element in a rank series has the same radius dependency as ctmf, ε_V and KNV neighborhood average with lazy histogram updating methods' times grow 2 and 4 times faster, but the new approach still performs better than updating the histograms by Huang's method.

Fig. 3b shows execution time per megapixel in dependency of image width. The plot shows that the greater width of the image is the more efficient proposed algorithms behave themselves. This result is explained by the huge amount of already counted and stored multiscale histogram elements while moving the sliding window on the row of the image. The step on the graph is a consequence of the fact that from a certain image width column histogram set do not fit into processor's cache. However, it should be noted that for large image width this value is constant too. The plot for ctmf [2] algorithm don't have step because this algorithm use smaller histogram structure and step occurs at larger image width. On the other hand KNV algorithm [3] requires triple sized histogram that's why step appears 3 times earlier in the plot. It is interesting to note, that the step for KNV algorithm is more gently sloping. Probably this is due to the fact that two additional side histograms for KNV algorithm are used not so intensive as main center histogram and in average are not used in whole. This results that side parts of histograms are cashed partially and with low priority and effect of cash overflow appears later then it can be expected.

Time statistics

Numerical comparison of proposed algorithms with [3] and [2] is presented in Table 1.

Table 1. Comparison of the proposed, fast rank algorithms [3] and ctmf [2] methods' efficiency. Image size 375x486

radius	r = 2	r = 12	r =37	r = 62
Fast rank algorithms [3] + Huang's histogram updating [1]				
ε_V	0.579	2.555	7.236	11.625
KNV	0.581	2.605	7.318	11.713
Fast rank algorithms [3] + proposed lazy histogram updating method				
ε_V	0.357	0.492	0.674	0.781
KNV	0.314	0.63	0.541	0.565
Constant time median filtering [2]				
ctmf	0.196	0.198	0.198	0.201

Conclusion

New fast method for multiscale histogram updating and summation is proposed. The method is based on lazy histogram updating. Computation times in dependence of image width and sliding window radius were analyzed during numerical experiments. It was shown that the processing speeds for εV and KNV neighborhood average algorithms are only a few times slower than times for the fastest known median filtering algorithm. The proposed approach is based on multiscale histograms and its lazy updating allows fast implementation of wide variety of rank algorithms.

References

1. T. Huang, G. Yang, G. Tang. A Fast Two-Dimensional Median Filtering Algorithm // IEEE Trans. Acoust., Speech, Signal Processing, - 1979, - Vol. 27, No. 1, - P. 13–18.
2. S. Perreault, P. Hebert. Median Filtering in Constant Time // IEEE Transactions on Image Processing, - 2007, - Vol. 16, - P. 2389-2394.
3. M.V. Storozhilova, D.V. Yurin. Fast Rank Algorithms Based on Multiscale Histograms // In: 21-th International Conference on Computer Graphics GraphiCon'2011. Moscow, Russia, - 2011, - P. 132 -135.
4. C. Tomasi, R. Manduchi. Bilateral Filtering for Gray and Color Images // Proceedings of the IEEE Sixth International Conference on Computer Vision (ICCV'98), January 1998, - P. 839-846.
5. L.P. Yaroslavsky, V. Kim. Rank Algorithms for Picture Processing, Computer Vision // Graphics and Image Processing, - 1986, - Vol. 35, - P. 234-258.

[1] The research is done under support of Federal Target Program "Scientific and Scientific Pedagogical Personnel of Innovative Russia" in 2009-2013 and the Russian Foundation for Basic Research, project no. 09-07-92000-HHC_a.