Comparison of Two Approaches to Finding the Median in Image Filtering

A. Bosakova-Ardenska

***Key Words**: Median filtering; partial histograms; bucket sort.*

***Abstract.** This paper discusses two approaches for finding the median when we filter gray-scale images. A partial histogram is used a for finding new value (median) of every pixel (i.e. algorithm proposed by Thomas Huang, George Yang and Gregory Tang). An analytic research is made showing that using the previous median as a „pivot" is a faster approach in comparison with integrate histogram approach when the radius of mask is less than 42. The experiments show that if the type of noise is Gaussian, the bound value of the radius is really 42. But if the type of the noise in the image is different from Gaussian, then the bound could be different.*

Introduction

The image processing is a part of the information processing in every computer system. The algorithms for image processing are divided into three hierarchal groups [1,2]:

- algorithms for primary image processing (low-level image processing);
- algorithms for intermediate image processing;
- high level algorithms for image processing.

Primary image processing could be done either in a spatial domain or in a frequency domain [1,2,3]. Different filters are used for image enhancement. Depending on the chosen domain there are used: the function of the filter (its Fourier transformation) or the mask (kernel, window) that presents filtering function. The masks are square matrices with odd number of size and they have coefficients. The values of these coefficients define the type of the filtering function. If an image is processed in the frequency domain then the image processing is a convolution of Fourier transforms of image and filter function.

$$b(i,j) = a(i,j) * f(i,j)$$

Where a is the a original image, b is the processed image and f is a filter function. When the processing is in spatial domain, the filtering is made with a mask that moves over the image from left to right and from top to bottom. An operation „shift-and-multiply" is made [3], i.e. on every step of the mask movement their coefficients (on the mask) are multiplied with pixel values, and after that these multiples are summed-up and finally the received value is divided to a coefficient of correction. The value received in this way is the new value of the pixel that is under the center (central element) of mask. The processing could be an anisotropic or recursive one [1] depending of what value of pixels is used: original or processed.

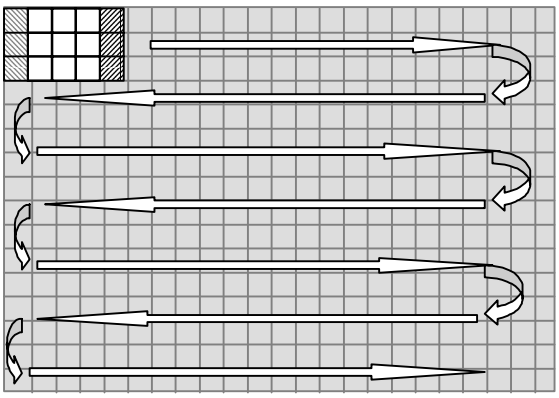

Figure 1

Algorithms for Median Filtering

The median filter is often used for primary images processing. It does not blur the edges (the filter mean or neighbourhood averaging makes this). [1,2,3]. The median filter is non-linear, which means that if two images are summed-up and the result image is median filtered, the processed image is different from the sum of these two images processed with filter median.

$$\text{median}[A(x) + B(x)] \neq \text{median}[A(x)] + \text{median}[B(x)].$$

The new value of the pixel which falls under the center of mask is a median value of all pixels under the mask in sorted order. There are many algorithms for median filtering [4,5,6,7,8,9,10]. The algorithm that is proposed by Thomas Huang, George Yang and Gregory Tang [4] gives significant acceleration in comparison with „brute force" algorithms. They propose the using of partial histograms and removing the old values of pixels and adding new pixel values on the mask movement. Partial histograms are used for finding the median and the previous median is used as a „pivot" (initial point) for finding the current median. This algorithm leads to significant acceleration of the filtering process.

An idea of the movement of the mask being in „zigzag" appears later. In other words in every row the scanning direction will be changed. By this way a partial histogram must be calculated only for the first pixel and for all other pixels this histogram is modified by removing the unnecessary pixels and adding new pixels which fall under the mask (updating partial histogram).

On *figure 1* the movement of mask sized 3x3 is shown. With grey colour are hatch pixels that must be removed from the partial histogram. With black colour are hatch pixels that must be added to partial histogram.

When the mask is moved to the next row it changes its direction. Besides the positions of the pixels that must be removed and those that must be added to the partial histogram will be exchanged.

Obviously, this algorithm could be applied only to the

Array COLOUR

Colour	0	1	2	3	4	5	6	...	127	128	129	130	131	132	...	252	253	254	255
number	0	0	0	0	0	2	0	0	1	0	2	0	1	1	0	0	2	0	0

Array INDEX

element	0	1	2	3	4	5	6	7	8
color with non-zero value	5	127	129	131	132	253	-1	-1	-1

Figure 2

anisotropic image processing. When the processing is a recursive one the direction of mask moving must not change because the order of pixel calculation could not change either. The algorithm could be applied to a recursive image processing only if the mask is moved from left to right and also for every first pixel of a row the partial histogram is cleared and calculated again. Besides this after processing every pixel, the partial histogram could be modified with pixel's new value.

In 2006 Ben Weiss presents an algorithm with time complexity □(log r) where r is the radius of mask [7]. This algorithm is faster than the algorithm of Thomas Huang, George Yang and Gregory Tang and the process is accelerated because multiple columns are processed like one. Ben Weiss's algorithm is implemented on Power Mac G5 2,5 GHz Single Processor and use opportunities for vector operations of this processor. For median finding Ben Weiss uses the previous median as a „pivot" like Thomas Huang, George Yang and Gregory Tang.

In 2007 Simon Perreault and Patrick Hebert present algorithm for median filtering with constant time complexity - □(1) [9]. They use vector operations for acceleration of histograms updating and find median by histogram integrating. Authors show by experiments that their algorithm is faster than Ben Weiss's algorithm when the mask radius is higher than 40. They use Power Mac G5 1,6 GHz for their experiments.

In 2007 David Cline, Kenric B. White and Parris K. Egbert present an algorithm for median filtering of images with constant time complexity [10]. For median finding they use histogram integration. Their algorithm is faster than the algorithm of Thomas Huang, George Yang and Gregory Tang when the mask radius is higher than 7.

In [11] a modification of the algorithm with partial histograms is shown. This modification of the algorithm is named „Fast median filtering based on bucket sort". In this algorithm an additional array (called index) is used for saving the colors that run under the mask. By this way when exploring the array with all colors we can visit only these colors that have non-zero values. (On *figure 2* is shown an example for contents of array COLOUR and array INDEX when using mask 3x3). Additional time is needed for updating INDEX array and sorting it. Despite of this in practice (when using not very large masks) „Fast Median Filtering Based on Bucket Sort" algorithm is faster than the algorithm of Thomas Huang, George Yang and Gregory Tang [4].

The time complexity of „Fast median filtering based on bucket sort" the algorithm is □$(m+l^2)$, where **m** is the number of pixels falling under the mask (i.e. $(2r+1)^2$) and **l** is the number of different colors.

Analysis of Approaches for Finding the Median in Histogram

The existing new fast algorithms for median filtering [7,9,10] use various approaches to accelerate the updating histograms. Most of them integrate histogram to find median value which is slow operation.

Let us explore in details the approach for finding the median by integration of the histogram. We will process an 8-bit gray scale image (there are possible 256 colour values). In this case the histogram array will be sized to 256. Histogram integration means that we will explore the histogram array from the beginning and we will sum its elements. The process will stop when we reach the median. In other words after every sum we will check the value to find the median. Let us get an example:

The radius of the mask is 2, i.e. we will have 25 ($(2r+1)^2$) pixels under the mask (the mask is sized 5x5). The middle value in the array with 25 elements is 13. Let us have a variable - *sum*. We will initialize sum with value of the first element in the histogram array and will use this variable to summing-up elements in the histogram array. Let us name the histogram array - *colour*. In variable *median* we will receive the median value. In this case the integration of the histogram to find the median in C++ code will be as follows:

```
sum = colour[0];
for( i=1; i<256; i++ )
{
        if( sum>=13 )                   // If the sum is >= 13 we write in the
        {
                median = i;             // variable median value of the median.
                break;
        }
        sum += colour[i];               // Summing-up the values of colour
}
```

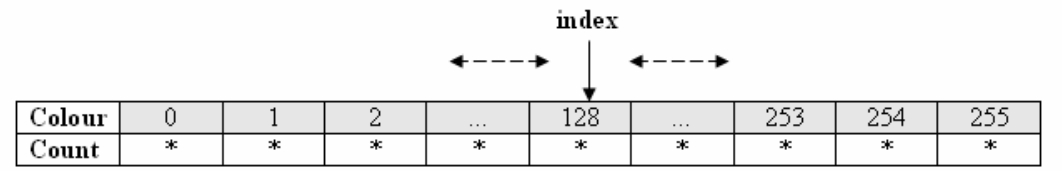

Figure 3

The number of operations (comparisons and additions) depends on the median value. If the median value is in the begging of the histogram array then the number of executed operations will be small. But if the median value is in the end of histogram array then the number of executed operations will be higher. Let us examine 3 variants:

Variant 1. The median value is 0. In this case we will execute 1 operation (1 comparison).

Variant 2. The median value is 127. In this case we will execute 255 operations (127 additions and 128 comparisons).

Variant 3. The median value is 255. In this case we will execute 511 operations (255 additions and 256 comparisons).

Finally we can state that for finding the median by integration when we process an 8-bit gray scale image (there are possible 255 colour values) on the average 255 operations (128 comparisons and 127 additions) are necessary .

This integration process could be avoided and the count of operations for median finding would be reduced. In [4] Thomas Huang, George Yang and Gregory Tang propose using of the previous median as a „pivot" when finding current median. The authors state that this approach is faster than the histogram integration.

Let us explore in details the approach for median finding by using previous median as a pivot. This approach is offered by Thomas Huang, George Yang and Gregory Tang. We will make a partial histogram, i.e. count how many times the colors of pixels under the mask are met. An array with possible colors has 256 elements (we will process 8-bit gray scale images). We will decrement the count of these pixels that do not fall under the mask and will increment the number of these pixels that have already fallen under the mask (*figure1*). We will need two additional variables - *sum* and *index*. The value of the variable index will be the current median. In the variable sum we will save the sum of all colors before the median (including count of median color). The value of sum could not be smaller than the middle value for the selected mask (if the mask is sized 3x3 then the middle value is 5, if mask is sized 5x5 then middle value is 13 and i.e.). We will name „center" the middle value for a selected mask. We can calculate the value of center by the formulae:

center = (mask_size * mask_size)/2 + 1

where the value of the variable mask_size is the size of selected mask (3, 5, 7, 9, i.e.). When the mask is moving the value of every pixel that has not already fallen under the mask must be checked. If the color of the pixel is smaller or equal to index then we decrement the value of the sum. For the new pixels falling under the mask we check the value and if necessary we increment the value of sum. To calculate the new value of the median we must check the value of the sum. It is possible to have 3 cases:

Case I (the median is the same):

sum = **center**, then the median keeps its value (the value of the index is the same).

Case II (we do not achieve the median value yet):

sum < **center**, then while the sum is smaller than the center we sum the value of the sum with the count of the next color in the array with all colors. We increment the value of the index. When the sum is equal or higher than the center, then we will have the median value in variable index.

Case III (we are after median value):

sum > **center**, then while the sum is higher than the center we subtract the value of the previous element in the array of all colors from the sum. If sum=center then go to case I. If sum<center then go to case II.

By this way the variable index is one pointer to the current median value in the array with all colors. When the mask is moving this pointer also moves (*figure 3*). The variable sum is used for finding the new value of the median (moving the pointer index).

Implementation of finding the median (case I, II and III) is as follows:

```
if ( sum < center ) // case II
{
        while ( sum < center )
                sum += colour[++index];
}
else
{
        if ( sum-colour[index] > center ) // case III
        {
                while ( sum - colour[index] > center )
                        sum -= colour[index--];
        }
}
```

During the mask moving only a little part of pixels falling under it are changed. For example if a mask is sized 7x7 we

Before processing

After processing

Figure 4

have 49 pixels but 14 from them are being changed. Therefore the value of the median (variable index) has a little change or it keeps its value.

The average count of the operations for median finding (when we have histogram) is 3.mask_size or 3(2r+1) where r is the radius of the mask. Therefore if we use this approach for the median finding it is necessary next inequality to be satisfied:

$3(2r+1) < 255$, i.e. $r < 42$

In other words when we use a mask with size not higher a 85 this approach is faster than histogram integration. The complexity of the algorithm proposed by Thomas Huang, George Yang and Gregory Tang is O(r).

Experimental Results

On *figure 4* is shown 8 bits gray-scale image with sizes 512x512 before and after processing by classic median filter with mask 5x5 (r=2). The noise in the image is in Gaussian distribution.

In *figure 5* it is shown the time for processing an image in *figure 4* with different sized masks. It is obviously that the approach to finding the median by using the previous median as a „pivot“ is faster than the histogram integration when the radius of the mask is smaller than 42.

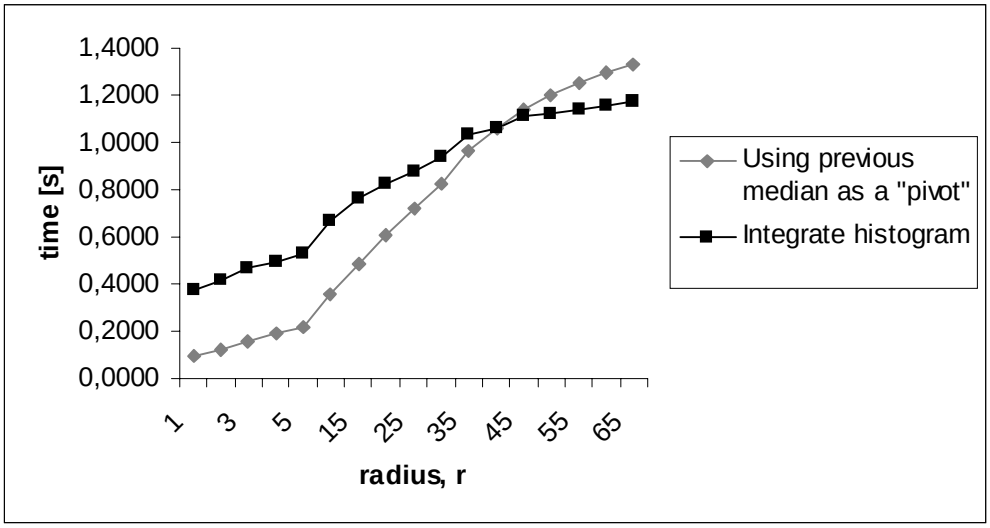

Figure 5

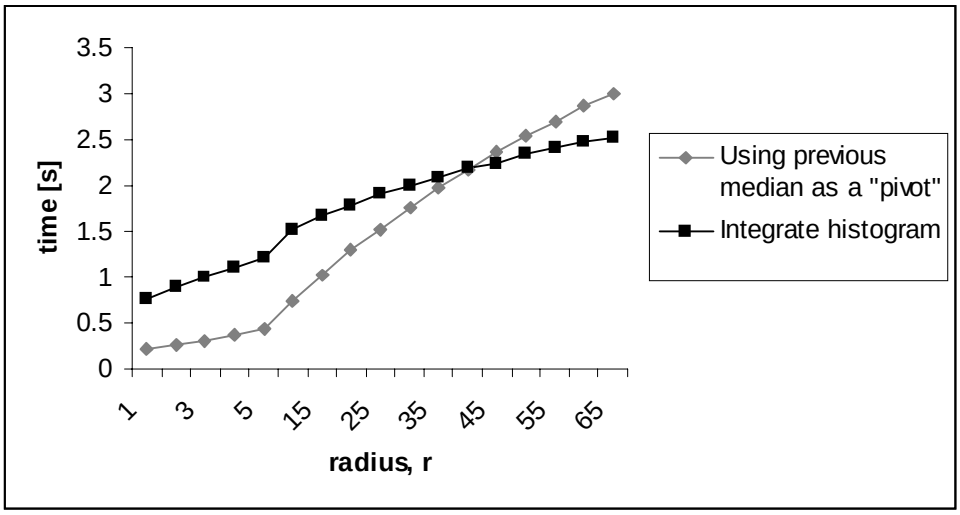

Figure 6

In *figure 6* it is presented the time for processing an 8-bits gray-scale image with 600x877 pixels with different sized masks. The noise in the image is in Gaussian distribution. The approach to find the median by using the previous median as a „pivot" is faster again than histogram integration when radius of the mask is smaller than 42.

Conclusion

Two approaches are known to find median when we have partial histogram. One approach is using the previous median as a „pivot" to find the current median. Another approach is integrating the histogram to find the median value. When integrate

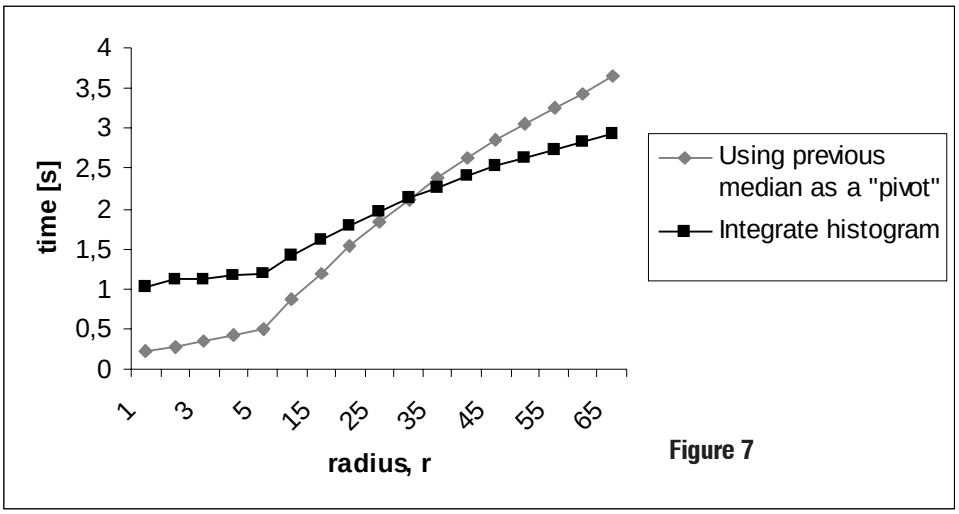

Figure 7

In *figure 7* it is presented the time for processing an 8-bits gray-scale image with 914x676 pixels with different sized masks. The noise in this image is of random nature. The approach to find the median by using the previous median as a „pivot" is faster than the histogram integration when the radius of the mask is smaller than 30.

For the experiments two programs are used. The first program implements the algorithm of Thomas Huang, George Yang and Gregory Tang. The second program implements the algorithm which uses partial histograms and histogram integration. Program implementation is in C++. In all experiments is used Intel Celeron 2,5 GHz processor.

histogram (to process an 8-bits grey-scale image) we have in average 255 operations. When we use a previous median as a „pivot" it is necessary 3(2r+1) operations in average. Therefore when we use no large masks the approach using the previous median as a „pivot" to find the current median is faster than the histogram integrating approach. If the mask radius is less than 42 then the approach which uses the previous median as a „pivot" is faster than the approach which integrates the histogram. Depending on noise nature the border value of r is changing. I.e. the value of the radius by which the first approach is faster than the second one is different when the nature of noise changes.

References

1. Gochev, G. Computer Vision and Neutral Networks. Sofia, Technical University, 1998 (in Bulgarian).

2. Gonzalez R., R. Woods. Digital Image Processing (Second Edition). Prentice Hall, 2002.

3. http://homepages.inf.ed.ac.uk/rbf/HIPR2/wksheets.htm

4. Huang T., G. Yang, G. Tang. A Fast Two-dimensional Median Filtering Algorithm. - *IEEE Transactions on Acoustics, Speech and Signal Processing,* ASSP 27, 1979, No 1.

5. Devillard, N. Fast Median Search: an ANSI C Implementation. http://ndevilla.free.fr/median/median.pdf, July 1998.

6. Ranka, S., S. Sahni. Efficient Serial and Parallel Algorithms for Median Filtering. - *TSP (IEEE Trans. Signal Processing)*, 29, 1991.

7. Weiss, B. Fast Median and Bilateral Filtering, International Conference and Exhibition on Computer Graphics and Interactive Techniques, 2006.

8. Lukin, A. Tips & Tricks: Fast Image Filtering Algorithms, GraphiCon, Russia, 2007.

9. Perreault, S., P. Hebert. Median Filtering in Constant Time. Image Processing, IEEE, 2007.

10. Cline, D. K.B. White, P.K. Egbert. Fast 8-bit Median Filtering Based on Separability. ICIP IEEE International Conference, 2007.

11. Bosakova-Ardenska, A., S. Stoichev, N. Vasilev, E. Stoicheva. Fast Median Filtering Based on Bucket Sort. - *Information Technologies and Control*, 2009, No. 2, 11-17.

Manuscript received on 22.12.2010

Atanaska Bosakova-Ardenska *was born in 1980. She received the M.Sc. degree of Computer Systems and Technologies at Technical University of Sofia, Plovdiv branch 2004. She receives PhD in 2009. From 2010 she is assistant in department of Computer Systems and Technologies in University of Food Technologies. Her research interests include: parallel algorithms, image processing, MPI (Message Passing Interface), C++ programming.*

Contacts:
University of Food Technologies
Plovdiv, blvd. „Maritza" 26
e-mail: abosakova@yahoo.com