FILTERING OF COLOR MAP IMAGES BY CONTEXT TREE MODELING

Pavel Kopylov and Pasi Fränti

Department of Computer Science, University of Joensuu

Box 111, FIN-80101 Joensuu, Finland.

ABSTRACT

We propose a method for filtering raster map images by context tree modeling. This is a two-pass method. At the first pass filter utilizes statistical information about image spatial structure and stores the statistics in a tree structure. At the second pass, these statistics are used in actual filtering to calculate the probability of current pixel in its neighborhood. We test this method on a set of map images. We use different noise models to evaluate the performance of the proposed filter. Finally we compare the performance results for our method and Vector Median filter. The proposed method does not destroy the object borders and outperforms the Vector Median filter for moderate level of noise.

1. INTRODUCTION

Digital maps can be converted to *raster images* for data transmission, distribution via Internet, or simply because of incompatibility of the vector representations of different systems.

Several techniques for noise removal are well known in color image processing [1][2][3]. However, map image processing needs additional constraints to be set. First of all, edges should be preserved, which means that various information types in the map like roads, signs and text information should not be corrupted by smoothing and remain untouchable if possible.

We have found out that among variety of methods Vector Median filter shows the best performance in sense of preserving the edge information and removing impulsive noise [3]. Map images, however, contain detailed information such as thin lines, which can be destroyed by median filter.

Here we introduce statistical Context-Tree based Filter, which proceeds the filtering based on estimation of the conditional probability of filtered pixel according to its neighborhood. In contradiction to Median and Vector Median filters, this is a two-phase filter, when at the first phase we collect conditional probability statistics for every pixel, and at the second phase we proceed with actual filtering. Pixels that have smaller estimated probability than given threshold are considered as noise and subject to filter.

2. OVERVIEW OF FILTERING METHODS

2.1. Vector Median Filter

The standard median algorithm is a non-linear operation of choosing the middle value inside a moving window of fixed size along the image. In map images the information usually consists of composite information of several correlated components, like color or spectral planes. Thus, if we try to separate the original color image into components, and to proceed with median filter each component separately, the output can easily have large error level near signal edges, since edge information could be different in each component. According to this, the multi component information is considered as vector, and processed by multidimensional extension of median filter, called Vector Median Filter. Vector median x_{vm} for a set of M-dimensional vectors, x_i, i =1,…,N, such that $x_{vm} \in \{x_i | i = 1,...,N\}$ and $\sum_{i=1}^{N} |x_{vm} - x_i|$ the sum is minimized [3]. If we consider map images as RGB-images, then we will have the set of three dimensional vectors with R, G and B axis's.

2.2. Context-Based Filter

Context-based filtering uses context template to calculate probability of the pixel being filtered according to its local neighborhood [4]. The principle of context-based filtering corresponds to the adaptive arithmetic coders [5], which use context model to accurately estimate the probability of the upcoming symbol. The combination of colors within local neighborhood template uniquely defines a *context*.

The number of possible contexts depends on size of the spatial template, and the size of alphabet. This number can

0-7803-8554-3/04/$20.00 ©2004 IEEE.

also be very large – especially when non-binary alphabets are used. In fact it is defined by formula: $N=K^M$, where K is the size of alphabet, and M is the size of template.

The context template, which used in arithmetic coding, consists only of symbols that have already been coded, see Figure 1, and thus visible by both coder and decoder. However context filtering does not have such restriction and template symbols can be chosen also from the area that has not yet been processed. Moreover, use of just upper part of the clairvoyant template will not provide us enough accurate statistics for filtering.

The context filter takes two passes over the image. Beforehand, the amount of memory needed for storing conditional pixel probability information is allocated. For every context, defined as combination of neighborhood colors $(C_1,C_2,\ldots,C_n)$, where n is the size of template, we also reserve array of counters for every color appear in this context.

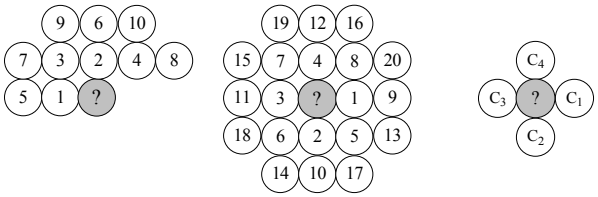

Figure 1: Context templates: 10 pixel template used in JBIG encoder, 20 pixel, and 4 pixel clairvoyant templates.

We proceed with the first pass of filtering by counting the number of times each color appears in every context. Then on the basis of these counters, conditional probabilities are estimated. As a result, we have an array of the size equal to the number of all possible contexts filled with information of how many times different colors appear in each context. At the second pass, for every image pixel we set its context, and if the conditional probability of the pixel is less than a predefined threshold level, then its color value is changed to the one in that particular context, which delivers the maximum conditional probability.

3. CONTEXT TREE FILTER

One of disadvantages of context filter is that we have to allocate memory for storing conditional pixel probability information for all possible contexts. But it can often happen, that the particular context could just not happen in the image because of image spatial properties.

Consider an example of 16 color image with dimensions 1024×1024 pixels, and using 20 pixel clairvoyant template of Figure 1, we have to spend $16^{20}\times16\times4\approx7.74\cdot10^{25}$ bytes to store all color counters for all contexts. On the other hand, the physical dimensions of the image limit the number of available contexts so that we can have at most up to 1024×1024×16×4=67108864 bytes to store all color counters for all contexts.

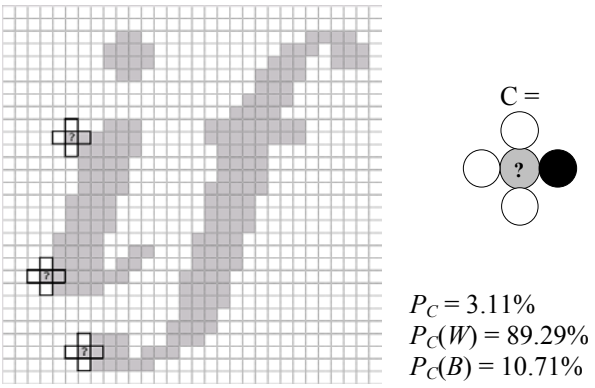

Figure 2: Context statistics in selected positions of black and white image. The selected positions are marked with "?". With the threshold level of 10% the selected pixels will be filtered – turned to white.

We propose to use context tree [6] for storing filtering statistics only for the contexts that appear in the image. The tree structure is as follows: leaves of the tree are used for storing information about contexts. Every tree node has as many branches as there are colors in the image in that particular context. The children of a node correspond to their parent by adding one more pixel at the position defined by context template, see Figure 1. The context selection is made by traversing the context tree from root to leaf, each time selecting the branch according to the value of the pixel in the corresponding position within context template.

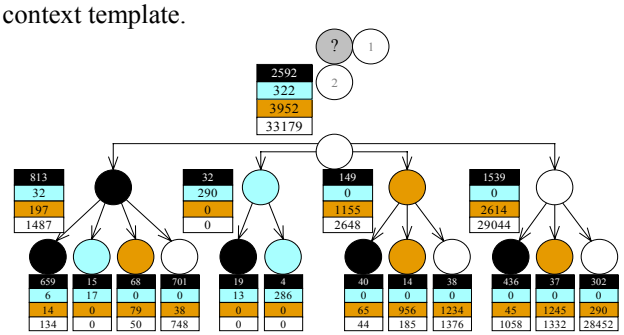

Figure 3: An example of context tree constructed up to level two, for a sample image containing 4 colors.

The filtering process has two phases. In the first phase, we create the tree and populate it with statistics. At the second phase, we proceed with actual filtering on the basis of the statistics of the context in the same way as done in context filter with one difference, that the context

selection is made by traversing the context tree, instead of straight–forward context mapping.
The tree is constructed up to a depth defined by context template while traversing the image in raster scan order. For every pixel in the image we select corresponding context and update its color counters, if during the selection process appropriate branch in the tree was not found, we create it.

4. EXPERIMENTS

We evaluate the proposed method by filtering randomly chosen map images from Finnish National Land Survey map database [7], see Figure 4. Originally, scanning and dithering noise exists on the images, which can be observed as countering along dark sides of objects. Additionally, to check the efficiency of the filters, we corrupt the original images with noise. For this purposes we degrade images with impulsive noise with different ratios, and also with content-dependent noise for distorting the contours of the objects making them ragged.

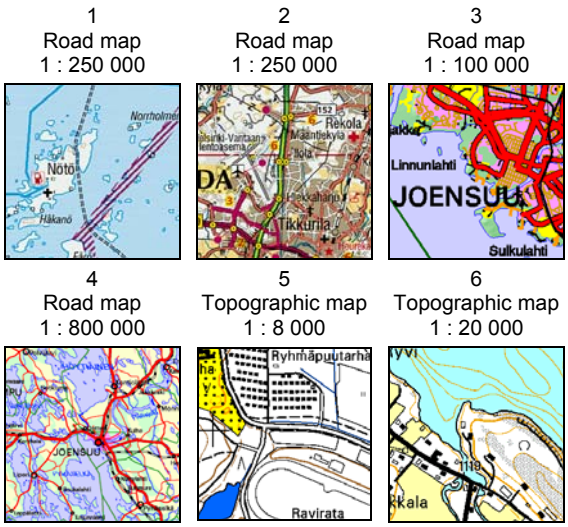

Figure 4: Sample fragments of the test images.

The comparison of the filters is shown in Figure 5 and Figure 6. The dot line in both graphs serves for the reference of the attenuated noise.
To evaluate the efficiency of filtering methods we calculate mean color difference between original and filtered images using formula:

$$\Delta E = \frac{1}{N}\sum \Delta E^*_{ab}$$

where ΔE^*_{ab} is the Euclidean distance between two color samples in uniform $L^*a^*b^*$ (CIELAB) space [8], and it is measured as:

$$\Delta E^*_{ab} = \sqrt{(\Delta L^*)^2 + (\Delta a^*)^2 + (\Delta b^*)^2}\ .$$

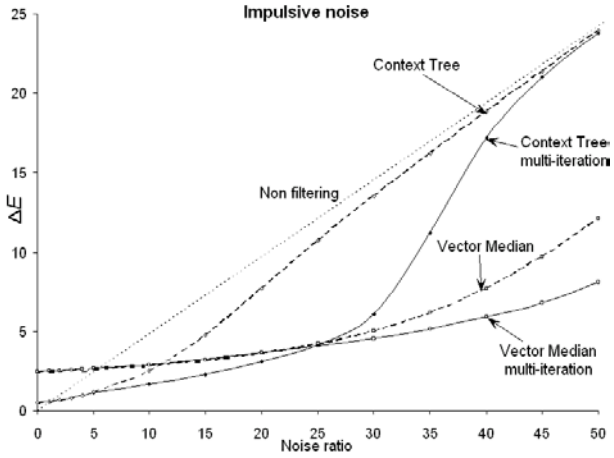

Figure 5: The comparison of multi-iteration and single-iteration variants of Context Tree and Vector Median filters for image #1. Single-iteration variants are given with dashed line

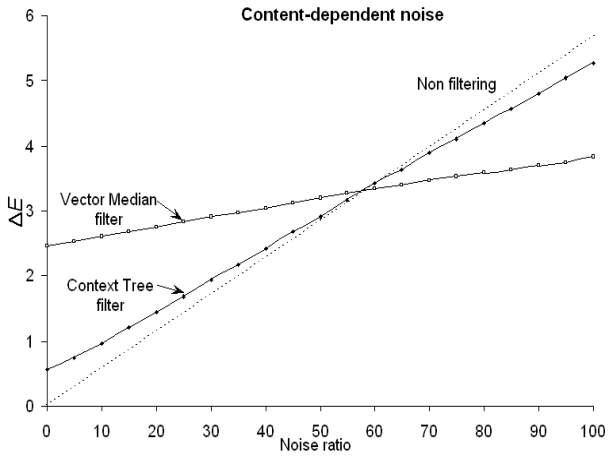

Figure 6: The comparison of Vector Median and Context Tree filters on images distorted with content-dependent noise for image #1.

As an extension to both filters we perform the same filter iteratively to improve the filtering quality while the filtering error is decreasing. This multi-iterative variant of the filter was applied to the same filtering sequence. We can see from the Figure 5, that with multi-iteration modification of Context Tree filter outperform single iteration variant already when we have 5% pixels distorted by impulsive noise. At the same time multi-iteration version of Vector Median filter starts improving filtering quality after 20% or more pixels have been distorted by impulsive noise. On the other hand, using multi-iteration filter modifications, does not provide any difference for filtering image sequence distorted by content-dependent noise, see Figure 6.

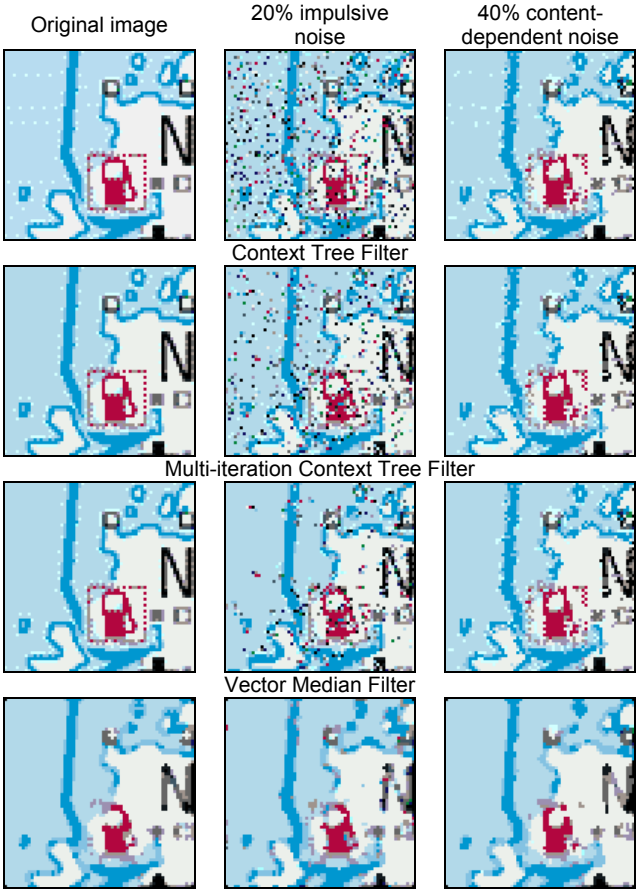

Figure 7: Visual comparison of Context Tree and Vector Median Filters on 64×64 pixel fragment of image #1

5. CONCLUSIONS

We have proposed a method for filtering color map images. The main advantage of the filter is that it does not destroy the borders of the object. It outperforms Vector Median filter until level of 25% corrupted pixels, which can be considered as severe noise. As main disadvantage of proposed method, we can outline the extensive memory consumption, which is not noticeable during procession of small images, but can lead to problems when filtering huge map images. Currently we do not use any strategy to reduce the size of tree structure, and as future development of the filter we could use tree-pruning methods and pre-quantizing step could be applied to reduce amount of children for non-leave nodes.

REFERENCES

[1] D. Vernon, *Machine Vision*, Prentice-Hall, 1991.

[2] E. Davies, *Machine Vision: Theory, Algorithms and Practicalities*, Academic Press, 1990

[3] J. Astola, P. Haavisto, Y. Neuvo, "Vector median filters", *IEEE Trans. Image Proceeding* 78 (4): 678-689, 1990.

[4] E.R. Dougherty, J. Astola (eds) "Nonlinear Filters for Image Processing", SPIE Optical Engineering Press, 1997.

[5] G.G. Langdon, J. Rissanen, "Compression of black-white images with arithmetic coding", *IEEE Trans. Communications* 29 (6): 858-867, 1981.

[6] B. Martins, S. Forchhammer, "Bi-level image compression with tree coding", *IEEE Trans. Image Proceeding* 7 (4):517-528, 1998.

[7] National Land Survey of Finland, Opastinsilta 12 C, P.O. Box 84, 00521 Helsinki, Finland. (http://www.nls.fi/index_e.html)

[8] CIE, *Colorimetry*, CIE Pub. No. 15.2, Centr. Bureau CIE, Vienna, Austria, 1986

Table 1: The mean color difference value of the Context Tree (CT) and Vector Median (VM) filters for set of map images. Multi-iteration filter modifications are given with CT MI and VM MI accordingly. Two different noise models are presented: 20% impulsive noise (case A) and 40% content-dependent noise (case B).

	Image #1		Image #2		Image #3		Image #4		Image #5		Image #6	
	A	B	A	B	A	B	A	B	A	B	A	B
CT	7,74	2,45	11,41	8,66	14,69	8,94	12,49	8,64	12,22	5,81	7,94	4,49
CT MI	3,12	2,45	7,82	8,66	5,38	8,94	8,04	8,64	5,39	5,81	4,34	4,49
VM	3,67	3,07	15,45	12,24	15,27	11,35	3,67	13,00	10,93	6,23	8,12	5,90
VM MI	3,67	3,07	15,45	12,24	15,27	11,35	3,67	13,00	10,93	6,23	8,12	5,90