SPEEDING UP THE KÖHLER'S METHOD OF CONTRAST THRESHOLDING

Guillaume Noyel, Member, IEEE

International Prevention Research Institute
95 cours Lafayette
69006 Lyon, France

ABSTRACT

Köhler's method is a useful multi-thresholding technique based on boundary contrast. However, the direct algorithm has a too high complexity - $\mathcal{O}(N^2)$ i.e. quadratic with the pixel numbers N - to process images at a sufficient speed for practical applications. In this paper, a new algorithm to speed up Köhler's method is introduced with a complexity in $\mathcal{O}(NM)$, M is the number of grey levels. The proposed algorithm is designed for parallelisation and vector processing, which are available in current processors, using OpenMP (Open Multi-Processing) and SIMD instructions (Single Instruction on Multiple Data). A fast implementation allows a gain factor of 405 in an image of 18 million pixels and a video processing in real time (gain factor of 96).

Index Terms— Köhler multi-thresholding, boundary contrast, fast image segmentation, parallelisation, pattern recognition

1. INTRODUCTION

Adaptive thresholding is one of the most used technique in many applications because it is fast to compute and when combined with previous filters, it gives robust decision rules for pattern recognition. Among many other techniques of thresholding [1]; Köhler's method computes a curve of contrast of the region boundaries in an image [2]. The contrast steps correspond to the local maxima of the curve and can be extracted for (multi-)thresholding of the image. This is useful for many applications: industrial, biomedical, video, etc [3, 4, 5]. However, computing Köhler's method is time consuming; almost 1 minute using a C++ implementation on a current computer with an image acquired by a recent camera (18 million pixels, fig. 1). The purpose of this paper, is to introduce and implement a new algorithm for Köhler's thresholding method faster than the existing algorithms and making it useful for applications requiring fast or real-time processing (e.g. video thresholding, large datasets) [6].

Previously, two attempts were made to speed up the computation of Kölher's method. Zeboudj [7, 8] used mathematical morphology operations to give a similar version of Köhler's method. However, his approach was efficient on specific devices which are not available any more. The other one, from Hautière [9, 3], consists of making the computation on a reduced part of the neighbourhood and to pre-calculate some intermediate images of minimum and maximum between the image translated horizontally and vertically in order to compute the contrast. However, this algorithm does not introduce any parallelisation.

In this paper, after a reminder on Köhler's method and on parallel computing in Mathematical Morphology [10, 11, 12]; we will introduce a parallel algorithm for Köhler's method using line translations of the image. We will also propose to compute the contrast on a reduced neighbourhood as in [9]. Eventually, we will compare an implementation of our algorithm, using vectorisation with SIMD instructions [13] and multi-core (i.e. parallel) processing with OpenMP [14], to other implementations of Köhler's method.

2. PREREQUISITES

Let us remind Köhler's method and the acceleration of Mathematical Morphology operations. An image is a function f defined on a discrete domain $D \subset \mathbb{Z}^n$ with values in $[0, M[$, $M \in \mathbb{R}$ and $M = 256$ for 8 bits images. We denote x the location of a pixel and f_x its value. In the sequel, we will use the 4-neighbourhood N_4 of pixels. For bi-dimensional images, we can also use the 8 or the 6-neighbourhood [12] with insignificant differences [3].

2.1. Köhler's method

Let us remind Köhler's method [2, 4]. Given a grey-level image f and a threshold $t \in [0, M[$, two classes are generated by t: $C_0^t(f) = \{x \in D, f_x \leq t\}$ and $C_1^t(f) = \{x \in D, f_x > t\}$. A boundary $B(t)$ is also generated:

$$B(t) = \left\{(x_0, x_1) \in D^2, x_0 \in C_0^t(f), x_1 \in C_1^t(f) \text{ and } x_1 \in N_4(x_0)\right\}. \quad (1)$$

For each couple of pixels (x_0, x_1) of $B(t)$, Köhler associates a contrast $C_K^t(x_0, x_1)$ defined as:

$$C_K^t(x_0, x_1) = \min\left(f_{x_1} - t, t - f_{x_0}\right) \quad (2)$$

which is the minimum of the two steps (in terms of contrast) generated by the threshold t between f_{x_0} and f_{x_1}. The average contrast of the boundary $B(t)$ is defined as:

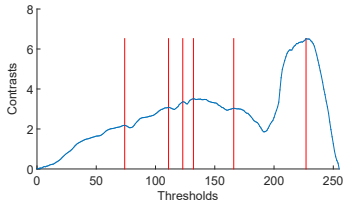
$$C_K(B(t)) = \frac{1}{\#B(t)} \times \sum_{(x_0,x_1)\in B(t)} C_K^t(x_0, x_1). \quad (3)$$

$\#B(t)$ is the cardinality (number of elements) of $B(t)$ and the summation is made on the couples of pixels (x_0, x_1) belonging to $B(t)$. This generates a curve of contrasts $C_K(B(t))$ for all the possible thresholds $t \in [0, M[$. The optimal threshold t_0 is selected as:

$$C_K(B(t_0)) = \max_{t\in[0,M[} (C_K(B(t)) . \quad (4)$$

In figure 1, we have extracted the 6 most significant thresholds (i.e. the local maxima) from the contrast curve. These multiple thresholds give an efficient simplification (i.e. compression) of the image grey levels: passing from 256 to 7 grey levels. The 7 grey levels corresponds to the mean value of the pixels for each class of the segmentation.

(a) Initial image (256 classes) (c) Segmented image (7 classes)



(c) Contrast curve

Fig. 1. Multiple thresholding by Köhler's method of the (a) original image into a (b) segmented image (the class value is the mean values of the class pixels) by (c) the seven thresholds selected on the contrast curve.

A direct C++ implementation consists of computing for each threshold $t \in [0, M[$, the contrast $C_K^t(B(t))$ of the boundary $B(t)$. It has a duration of 53 s using an image of size 3672×4096 pixels and a processor Intel® Core™ i7 CPU 4702HQ, 2.20 GHz, 4 cores, 8 threads. As the algorithm is not parallel, a single thread is used. For real-time applications, or big datasets, a faster algorithm is needed.

2.2. Accelerating operations on a neighbourhood

In Mathematical Morphology, for operations on a neighbourhood some acceleration methods exists. With a symmetric structuring element A (such as the one associated to the 4-neighbours), the morphological dilation corresponds to the Minkowski addition [15, 10, 11, 12, 16]:

$$X \oplus A = \bigcup_{a\in A} X_a = \{x + a : x \in X, a \in A\}. \quad (5)$$

$X_a = \{x + a : x \in X\}$ is the set $X \subset D$ translated by the vector a. A direct implementation of a dilation, by computing the union on the neighbourhood of each pixel (fig. 2 (a)), will lead to an algorithm of complexity of $\mathcal{O}(N \times |A|)$ [17]. N is the number of pixels in the image and $|A|$ is the cardinality of $|A|$. However, the Minkowksi addition (i.e. the dilation) has the property to distribute the union [18][11]: $X \oplus (A \cup A') = (X \oplus A) \cup (X \oplus A')$. This property has important technological consequences as a dilation can be computed elements by elements of the structuring element A, before combining the intermediate results by union. Therefore,

$$\begin{aligned} \mathrm{X} \oplus A &= \{\, \mathrm{x} \in D \mid \exists a \in A, x - a \in X\} \\ &= \textstyle\bigcup_{a\in A}\{x \in D \mid x - a \in X\}. \end{aligned} \quad (6)$$

An implementation by translating all pixel x by the vectors $a \in A$ and checking if they belong to the set X will have a complexity of $\mathcal{O}(|X \oplus A|)$ [17]. As images are stored in computer memory as unidimensional arrays, an efficient implementation [19] consists of translating the lines instead of the pixels (fig. 2 (b)). As the operations in each line are independent from the other lines (eq. 6), a parallel programming method is used, namely OpenMP [14], which is designed for multi-processor/core, shared memory computers. Each thread is computed by a single core and all cores share a common memory. In addition, vectorised data are used with SIMD instructions [13] in each thread. Vector operations using SIMD instructions allow multiple data to be processed with a single instruction of the processor while scalar operations use one instruction to process each individual data. The size of the SIMD registers being of 128 bits, 16 operations on integers of 8 bits are performed at the same time instead of a single one. In recent compilers, the vectorisation is performed automatically after activation of the right option. In [19], a dilation with a square structuring element of size 3 pixels in a 8 bit image (of size 1024×1024 pixels) is accelerated by a factor

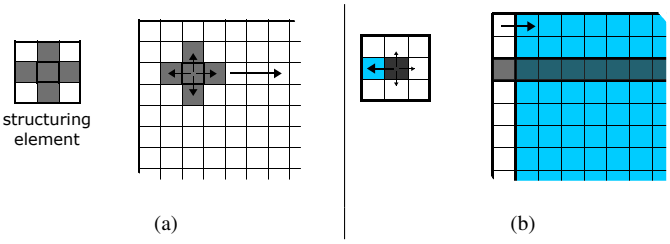


Fig. 2. Computation of a dilation (a) using a neighbourhood iterator or (b) by line translation.

136 with a neighbourhood implementation by comparison to a line implementation with parallelisation and vectorisation (45 ms versus 0.33 ms, Intel® Core™ i3 CPU M330, 2.13 GHz, 2 cores, 4 threads). Let us introduce an efficient method to compute Köhler's method.

3. A FAST ALGORITHM FOR KÖHLER'S METHOD

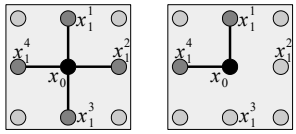
3.1. Accelerating the computation of Köhler's contrast

A direct implementation of Köhler's approach (section 2.1) is not designed for parallel and vector processing. Kölher's contrast (eq. 2) is summed on boundaries $B(t)$ which are computed by the set difference between a morphological dilation of the set $C_1^t(f)$ and this same set: $B(t) = (C_1^t(f) \oplus A) \setminus C_1^t(f)$. The structuring element A corresponds to the 4-neighbours. The direct implementation has a complexity of $\mathcal{O}(N^2)$ (i.e. the complexity of a dilation, $\mathcal{O}(N)$, multiplied by the complexity of scanning the pixel pairs of the boundary $B(t)$, $\mathcal{O}(M \times \#B(t)) = \mathcal{O}(N)$). A direct acceleration would consist of computing the boundary with an accelerated morphological dilation, as presented above. However, such approach does not reduce the complexity of the algorithm. For this purpose, we propose first to perform the translation of the image lines, which is suited for parallel processing, and then to compute the contribution to the contrast of each pixel pairs, for each threshold. The complexity decreases to $\mathcal{O}(NM)$, i.e. the product of the number of pixels by the number of grey levels. The algorithm 1 presents our approach.

Algorithm 1 Fast algorithm for Köhler's method

1: I, J ▷ number of lines and columns
2: $inLine(i,:)$ ▷ Line i of the input image
3: a ▷ $a = (a.x, a.y)$ translation of the neighb. $A = N_4^*$
4: $curLine$, $nLine_1$ ... $nLine_a$ ▷ arrays of length J
5: C, $card$ ▷ arrays of length M initially set to zero
6: **for all** $i \in [1, I]$ **do**
7: $curLine \leftarrow inLine(i,:)$
8: **for all** $a \in A$ **do**
9: $nLine_a \leftarrow translate(inLine(i + a.y, :), a.x)$
10: **for all** $j \in [1, J]$ **do**
11: $mini \leftarrow \min(curLine(j), nLine_a(j))$
12: $maxi \leftarrow \max(curLine(j), nLine_a(j))$
13: **for all** $t \in [mini, maxi - 1[$ **do**
14: $C(t) \leftarrow C(t) + \min(maxi - t, t - mini)$
15: $card(t) \leftarrow card(t) + 1$
16: **end for**
17: **end for**
18: **end for**
19: **end for**
20: **for all** $t \in [0, M[$ **do**
21: $C(t) \leftarrow C(t)/card(t)$
22: **end for**

3.2. Reduction of the neighbourhood size



(a) N_4 (b) N_4^*

Fig. 3. (a) The 4-neighbourhood N_4 and its (b) "half" N_4^*

In order to gain an additional factor 2, let us show that it is the same to make the computation on a "half" neighbourhood N_4* as on the 4-neighbourhood N_4 (fig. 3). The idea has been introduced in [9, 3]. Here, the equality of the two approaches is demonstrated.

When scanning the boundary B(t) with the 4-neighbourhood N_4, only the pixels x_1, such as $f_{x_0} < f_{x_1}$, are contributing to the boundary contrast. However, both pixels x_0 and x_1 are neighbours: $x_0 \in N_4(x_1) \Leftrightarrow x_1 \in N_4(x_0)$. Therefore, both (ordered) pairs (x_0, x_1) and (x_1, x_0) are scanned and only one pair is contributing to the contrast between the (unordered) set of points $\{x_0, x_1\}$

$$\begin{aligned}\Gamma_K^t(\{x_0,x_1\}_{N_4}) &= \begin{cases}\min\left(f_{x_0}-t, t-f_{x_1}\right), \text{if } f_{x_1} \le t < f_{x_0} \\ \min\left(f_{x_1}-t, t-f_{x_0}\right), \text{else } f_{x_0} \le t < f_{x_1}\end{cases} \\ &= \min\left(|f_{x_0}-t|, |t-f_{x_1}|\right).\end{aligned} \tag{7}$$

Let us remove the order condition between the grey levels, $f_{x_0} \le t < f_{x_1}$ and define the absolute pair contrast $\overline{C}_K^t(x_0, x_1) = \min\left(|f_{x_0} - t|, \ |t - f_{x_1}|\right)$. When scanning both pairs without the grey level order, the contrast between the set of points $\{x_0, x_1\}$, is:

$$\begin{aligned}\overline{\Gamma}_K^t(\{x_0,x_1\}_{N_4}) &= \overline{C}_K^t(x_0,x_1) + \overline{C}_K^t(x_1,x_0) \\ &= 2\min\left(|f_{x_0}-t|, |t-f_{x_1}|\right) \\ &= 2\,\Gamma_K^t(\{x_0,x_1\}_{N_4}), \quad \text{(eq. 7)}\end{aligned} \tag{8}$$

Using the "half"-neighbourhood N_4^* allows to scan only one pair of pixels. Therefore, we obtain our result:

$$\overline{\Gamma}_K^t(\{x_0,x_1\}_{N_4^*}) = \Gamma_K^t(\{x_0,x_1\}_{N_4}). \tag{9}$$

3.3. Implementation: parallelisation and vectorisation

In order to make parallel the algorithm, the computation of the contrast can be performed independently line by line. For each parallel thread k, two arrays of length M, C_k (contrast) and $Card_k$ (counter), need to be created at the beginning of the parallel process (line 6, alg. 1). At the end of the parallel process (line 19), they are grouped by summation in two arrays C (contrast) and $Card$ (counter). The parallel programming language used is OpenMP in C++. Instead of being performed between single numbers, several operations can be performed using arrays, allowing the vectorisation of the data.

The following operations are vectorised and processed using SIMD instructions: i) the line translation (line 9), ii) the computation of the minimum (line 11) and the maximum (line 12) between the arrays $curLine$ and $nLine_a$, iii) the computation of the contrast C_k (line 14) and of the counter $card_k$ (line 15) and iv) the normalisation of the contrast (line 20).

4. RESULTS

We now compare the duration of the direct implementation to the fast algorithm. We have used a processor Intel® Core™ i7 CPU 4702HQ, 2.20 GHz, 4 cores, 8 threads with 16Gb RAM. Using the image "Tulips" (fig. 1) with a current camera resolution of 3672×4896 pixels and the fast algorithm with parallelisation, the computation of Köhler's method is made in 0.13 s (tab. 2) instead of 53 s, with a gain factor of 405. With images of former resolution (512×512 pixels), such as Lenna image; the direct method takes 0.69s and the fast method 0.005s with a gain factor of 126. Therefore, the necessity of using a faster algorithm instead of direct implementation becomes essential to process images with current resolution. Other experiments have confirmed this result.

Name	Lenna	Tulips
Size (pixels)	512×512	3672×4896
Direct (D)	6.90e-01 s	5.29e+01 s
Fast (B)	1.62e-02 s	4.86e-01 s
Fast (A)	5.46e-03 s	1.30e-01 s
Gain (B vs. D)	42	109
Gain (A vs. D)	126	405

Table 1. Comparison of the duration of the different implementations for images of different sizes: direct (D), fast without parallelisation (B) and fast with parallelisation (A). The gain factors are computed between the implementations B and D and between the implementations A and D.

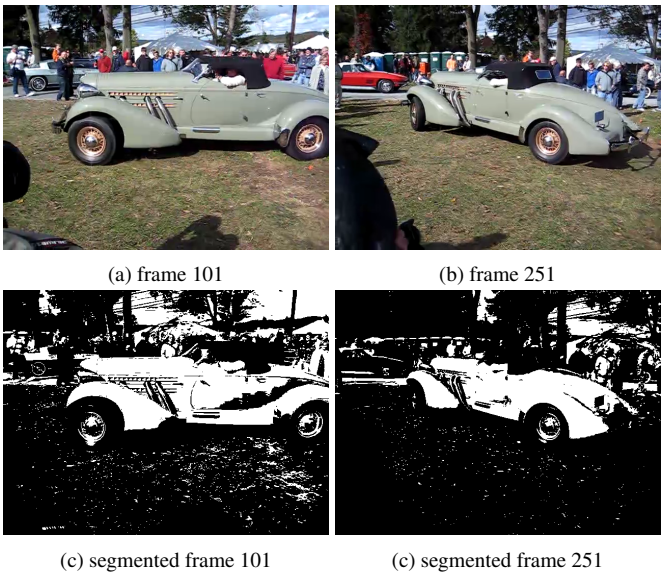
Let us try Köhler's method with a video of a car from the dataset YFCC100M (Yahoo Flickr Creative Commons 100M) [6, 20]. In figure 4, two frames and their segmentations in two classes are shown. The direct implementation segments the video at a rate of 1 frame per second while the fast implementation (with parallelisation) processes 97 frames per second, which is faster than the 25 frames/s of the video. Therefore, the fast algorithm for Köhler's method is suited for real time video processing.

5. CONCLUSION AND PERSPECTIVES

A faster algorithm for Köhler's thresholding has been introduced with a lower complexity, $\mathcal{O}(NM)$, than the direct approach, $\mathcal{O}(N^2)$. It is designed to benefit from the capacities of processors: multi-core processing with OpenMP and vector processing using SIMD instructions. Results show that with an image of 18 million pixels the duration is reduced by a factor 405 (from 53 s to 0.13 s) and that a video can be processed at a rate of 97 frames/s instead of 1 frame/s. Importantly, this algorithm is suited for applications requiring real-time or fast processing: video, industrial, large databases, etc. Its practical interest is to be combined with previous transforms: a low-pass filter, a mathematical morphology transform [11, 21, 17, 22] or a map of colour distances [23, 24]. In future works, the influence on the method of different contrasts will be presented (already studied), such as the contrasts defined in the Logarithmic Image Processing framework [4, 5].

Name	YFCC100M (car)
Size (pixels)	502×480
Number of frames	640
Direct (D)	0.99 frames/s
Fast (A)	96.96 frames/s
Gain (A vs. D)	98

Table 2. Frame per seconds segmented by Köhler's method with different implementations applied on a video: direct (D), and fast with parallelisation (A). The gain factors between the implementations A and D have been computed.



(a) frame 101 (b) frame 251

(c) segmented frame 101 (c) segmented frame 251

Fig. 4. (a), (b) Two frames of a video from the dataset YFCC100M and (c), (d) their segmentations in two classes by Köhler's method.

6. REFERENCES

[1] H. Cai, Z. Yang, X. Cao, W. Xia, and X. Xu, "A new iterative triclass thresholding technique in image segmentation," *IEEE Transactions on Image Processing*, vol. 23, no. 3, pp. 1038–1046, March 2014.

[2] Ralf Kohler, "A segmentation system based on thresholding," *Computer Graphics and Image Processing*, vol. 15, no. 4, pp. 319 – 338, 1981.

[3] N. Hautière, R. Labayrade, and D. Aubert, "Real-time disparity contrast combination for onboard estimation of the visibility distance," *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, no. 2, pp. 201–212, June 2006.

[4] M. Jourlin, M. Carré, J. Breugnot, and M. Bouabdellah, "Chapter 7 - Logarithmic image processing: Additive contrast, multiplicative contrast, and associated metrics," in *Advances in Imaging and Electron Physics*, Peter W. Hawkes, Ed., vol. 171, pp. 357 – 406. Elsevier, 2012.

[5] M. Jourlin, "Chapter three - metrics based on logarithmic laws," in *Logarithmic Image Processing: Theory and Applications*, Michel Jourlin, Ed., vol. 195 of *Advances in Imaging and Electron Physics*, pp. 61 – 113. Elsevier, 2016.

[6] Bart Thomee, David A. Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li, "YFCC100M: The new data in multimedia research," *Commun. ACM*, vol. 59, no. 2, pp. 64–73, Jan. 2016.

[7] Rachid Zéboudj and Michel Jourlin, *Filtrage, seuillage automatique, contraste et contours du pré-traitement à l'analyse d'image*, Ph.D. thesis, Université de Saint-Etienne, France, 1988, Thèse de doctorat Sciences. Informatique.

[8] M. Coster and J.L. Chermant, *Précis d'analyse d'images*, CNRS plus. Presses du CNRS, 1989.

[9] Nicolas Hautière and Michel Jourlin, "Measurement of local contrast in images, application to the measurement of visibility distance through use of an onboard camera," *Traitement du Signal*, vol. 23, no. 2, pp. 145–158, 2006.

[10] G. Matheron, *Eléments pour une théorie des milieux poreux*, Masson, Paris, 1967.

[11] J. Serra and N.A.C. Cressie, *Image analysis and mathematical morphology: Vol. 1*, Academic Press, London, 1982.

[12] Pierre Soille, *Morphological Image Analysis: Principles and Applications*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2 edition, 2003.

[13] Paul Cockshott and Kenneth Renfrew, *SIMD Programming Manual for Linux and Windows*, Springer Publishing Company, Incorporated, 1st edition, 2010.

[14] B. Chapman, G. Jost, and R. van der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming*, Number vol. 10 in Scientific Computation Series. MIT Press, 2008.

[15] Hermann Minkowski, "Volumen und oberfläche," *Mathematische Annalen*, vol. 57, pp. 447–495, 1903.

[16] L. Najman and H. Talbot, *Mathematical Morphology*, ISTE. Wiley, 2013.

[17] Thierry Géraud, Hugues Talbot, and Marc Van Droogenbroeck, *Algorithms for Mathematical Morphology*, pp. 323–353, John Wiley Sons, Inc., 2013.

[18] H. Hadwiger, *Vorlesungen über Inhalt, Oberfläche und Isoperimetrie*, Grundlehren der mathematischen Wissenschaften. Springer, 1957.

[19] Matthieu Faessel and Michel Bilodeau, "SMIL: simple morphological image library," Séminaire Performance et Généricité, LRDE, Mar. 2013.

[20] "Video of a car from the YFCC100M dataset," www.flickr.com/photos/32109282@N00/4078577031.

[21] J. Serra, *Image analysis and mathematical morphology: Theoretical advances*, vol. 2, Academic Press, 1988.

[22] G. Noyel, J. Angulo, and D. Jeulin, "A new spatio-spectral morphological segmentation for multi-spectral remote-sensing images," *International Journal of Remote Sensing*, vol. 31, no. 22, pp. 5895–5920, 2010.

[23] G. Noyel and M. Jourlin, "Asplünd's metric defined in the logarithmic image processing (LIP) framework for colour and multivariate images," in *Image Processing (ICIP), 2015 IEEE International Conference on*, Sept 2015, pp. 3921–3925.

[24] Guillaume Noyel and Michel Jourlin, "Spatio-colour Asplünd 's metric and Logarithmic Image Processing for Colour Images (LIPC)," in *CIARP2016 - XXI IberoAmerican Congress on Pattern Recognition*, Lima, Peru, Nov. 2016, International Association for Pattern Recognition (IAPR).