IEICE TRANS. INF. & SYST., VOL.E97–D, NO.5 MAY 2014

LETTER

An Efficient Strategy for Bit-Quad-Based Euler Number Computing Algorithm

Bin YAO[†], **Hua WU**[†], **Yun YANG**[†], **Yuyan CHAO**[††], ***Nonmembers***, **Atsushi OHTA**[†††], **Haruki KAWANAKA**[†††], ***and*** **Lifeng HE**[†††a)], ***Members***

SUMMARY The Euler number of a binary image is an important topological property for pattern recognition, and can be calculated by counting certain bit-quads in the image. This paper proposes an efficient strategy for improving the bit-quad-based Euler number computing algorithm. By use of the information obtained when processing the previous bit quad, the number of times that pixels must be checked in processing a bit quad decreases from 4 to 2. Experiments demonstrate that an algorithm with our strategy significantly outperforms conventional Euler number computing algorithms.

key words: *Euler number, topological property, computer vision, pattern recognition, binary image*

1. Introduction

The Euler number of a binary image, which is defined as the difference between the number of connected components and that of holes in the image, is one of the most important topological properties in a binary image [1]. The Euler number of a binary image will not change when the image is stretched or flexed like an elastic rubber. Therefore, the Euler number has been used in many applications: processing cell images in medical diagnosis [2], document image processing [3], shadow detection [4], reflectance-based object recognition [5], and robot vision [6].

Many algorithms have been proposed for calculating the Euler number of a binary image. For example, there are parallel algorithm for bit-plane computers [7],* graph-based algorithm [8], hardware algorithm [9], and algorithms for images with special formats [10], [11]. For the ordinary computer architecture and pixel-based format images, the algorithm proposed by Gray [12], which was adopted by the famous commercial image processing tools MATLAB [13], is one of the most efficient algorithms. This algorithm, denoted as *GRAY* algorithm in this paper, is based on counting certain patterns. There is also run-based algorithm [14], which calculates the Euler number by use of the number of runs and the number of neighboring runs, and skeleton-based algorithm [15], which calculates the Euler number by use of the number of terminal points and the number of three edge points. Recently, a new algorithm was proposed by He, Chao and Suzuki [16]. This algorithm, denoted as *HCS* algorithm, calculates the Euler number by labeling connected components and holes, and it is more efficient than the GRAY algorithm in many cases.

This paper presents an efficient strategy for improving the GRAY algorithm. By use of the information obtained during processing the previous bit-quad, the number of the times of checking the neighbor pixels for processing a bit quad decreases from 4 to 2, which leads to more efficient processing. Experimental results showed that our algorithm is more efficient than the GRAY algorithm for all images and other conventional Euler number computing algorithms for almost all images.

2. Reviews of the GRAY Algorithm and the HCS Algorithm

For an $N \times M$-size binary image, we assume that the object (foreground) pixels and background pixels in a given binary image are represented by 1 and 0, respectively. As in most image processing algorithms, we assume that all pixels on the border of an image are background pixels. Moreover, we only consider 8-connectivity in this paper.

2.1 The GRAY Algorithm

The GRAY algorithm for calculating the Euler number of a binary image is based on counting certain 2×2 pixel patterns called bit-quads, which are shown in Fig. 1, in the image. It checks whether the corresponding bit-quad is one of patterns P_1, P_2, and P_3. Let N_1, N_2, and N_3 be the numbers of patterns P_1, P_2, and P_3 in a binary image, respectively. Then, the Euler number of the image, namely E, can be calculated by the following formula.

$$E = (N_1 - N_2 - 2N_3)/4 \quad (1)$$

2.2 The HCS Algorithm

The HCS algorithm calculates the Euler number of a binary

Manuscript received November 1, 2013.
Manuscript revised January 11, 2014.

[†]The authors are with Artificial Intelligence Institute, College of Electrical and Information Engineering, Shaanxi University of Science and Technology, Xi'an, Shaanxi 710021, China.
[††]The author is with the Graduate School of Environment Management, Nagoya Sangyo University, Owariasahi-shi, 488–8711 Japan.
[†††]The authors are with the Graduate School of Information Science and Technology, Aichi Prefectural University, Nagakute-shi, 480–1198 Japan.
a) E-mail: helifeng@ist.aichi-pu.ac.jp (Corresponding author)
DOI: 10.1587/transinf.E97.D.1374

*This algorithm was proposed for cases where 4-connectivity is considered.

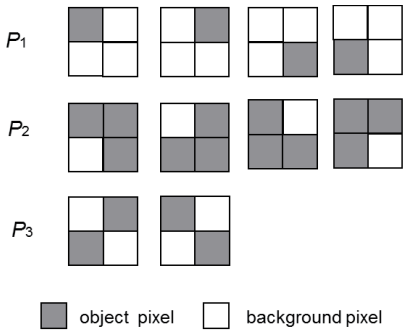

Fig. 1 Bit-quads for calculating the Euler number in the GRAY algorithm.

image according to its definition:

$$E = C - H \tag{2}$$

where C is the number of the connected components, and H is that of the holes in the image, respectively.

For calculating C and H, this algorithm extended the labeling algorithm proposed in Ref. [17] for looking for connected components and holes in the given binary image simultaneously. At any moment in the raster scan, all provisional labels assigned to an 8-connected component or a 4-connected hole in the processed area of the image are combined in an equivalent label set (the corresponding operation is called *label-equivalence resolving*), respectively. Thus, after the raster scan, all provisional labels assigned to a connected component or a hole in the image will be combined in an equivalent label set, respectively. Then, by counting the number of the equivalent label sets corresponding to connected components and that for holes, we can obtain the number of connected components, i.e., C, and that of holes, i.e., H, respectively.

3. Proposed Algorithm

In fact, some pixels are checked repeatedly in the GRAY algorithm. For example, in Fig. 2, when processing pixel a, it needs to check the four pixels in the corresponding bit-quad $\{a, b, c, d\}$. After doing that, it goes to process pixel c, and the four pixels in the corresponding bit-quad $\{c, d, e, f\}$ will be checked, where pixels c and d have just been checked during processing the previous pixel a. These repeated checking can avoided if we can use the information related to the pixels c and d obtained during processing pixel a.

Based on the above consideration, we define four states, namely S_1, S_2, S_3, and S_4, as shown in Fig. 3. Obviously, to confirm whether each corresponding bit-quad is a pattern to be counted in the GRAY algorithm, we only need to check the other two pixels, i.e., pixel X and pixel Y.

For each row in the given image, because all pixels in the border are background pixels, we will begin our processing from state S_1 (Fig. 3 (a)), and check the values of pixels X and Y: (1) if the values of both pixels X and Y are 1, we know that none of the patterns that should be counted in the

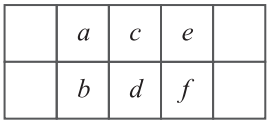

Fig. 2 An example for explaining the problem in the GRAY algorithm.

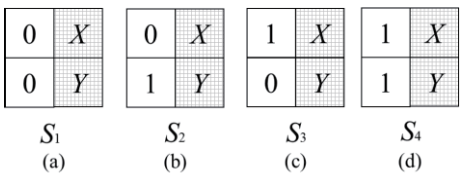

Fig. 3 Four states in our algorithm.

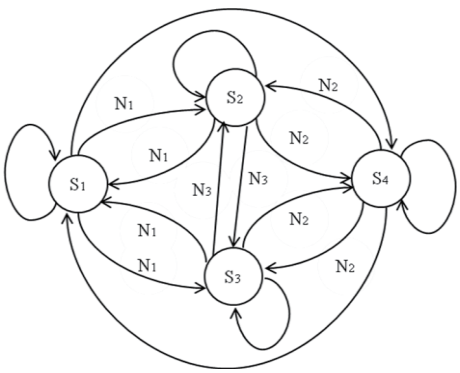

Fig. 4 State transition diagram.

GRAY algorithm, and then we go to process the next bit-quad, which obviously will be state (Fig. 3 (d)); (2) if the value of pixel X is 1 and that of pixel Y is 0, the corresponding bit-quad is pattern $P1$, $N1$ increases by 1, then we go to process the next bit-quad, which will be state S_3 (Fig. 3 (c)); (3) if the value of pixel X is 0 and that of pixel Y is 1, the corresponding bit-quad is also pattern P_1, and N_1 increase by 1, then we go to process the next bit-quad, which is state S_2 (Fig. 3 (b)); (4) if the values of both pixels X and Y are 0, the corresponding bit-quad is none of patterns P_1, P_2, and P_3, we go to process the next bit-quad, which will come back to state S_1. Other states can be processed in a similar way. The state transition is shown in Fig. 4.

After processing all pixels in the image, we obtained the numbers of the patterns P_1, P_2 and P_3, i.e., N_1, N_2, and N_3, then, we can calculate the Euler number by use of the formula (1) easily.

4. Time Complexity

As indicated above, for processing a bit-quad, the GRAY algorithm needs to check 4 pixels, i.e., it takes 4 pixel accesses. For an $N \times M$-size binary image, the total number of pixel accesses will be about $4N \times M$. According to the analysis results shown in Ref. [14], the skeleton-based algorithm [15] will take about $8N \times M$ pixel accesses, and for the run-based algorithm [14], it will take about $4N \times M$ pixel ac-

cesses in the worst case, and about $3N \times M$ pixel accesses in average case.

The algorithm proposed in Ref. [7] is a graph-based method. It computes the Euler number of a binary image for 4-connectivity by use of graph theory, where a binary image is taken as a square graph, which is constructed by taking all object pixels as vertices and adding all edges e_{ij} such that object pixels p_i and p_j is 4-neighbored. Let v, s, e be the number of vertices, basic square faces[†] and 4-connected edges, respectively, then the Euler number E can be calculated by $E = v + s - e$. This algorithm can be extended to compute the Euler number in a binary image for 8-connectivity as follows [18]: Let v, s, t, e be the number of vertices, basic square faces, basic right-angled triangle faces[††] and 8-neighbored edges except those hypotenuse edges inside basic square faces, respectively, then the Euler number E can be calculated by $E = v + s + t - e$. Because it needs to check all 8-connectivity edges, it will take at least $4N \times M$ pixel accesses for computing the Euler number.

Accordingly, the run-based algorithm is more efficient than the GRAY algorithm, the skeleton-based algorithm, and the graph-based algorithm.

By our strategy, for processing a bit-quad, by use of the information obtained during processing the previous bit-quad, we only need to check 2 pixels, i.e., for an $N \times M$-size binary image, the total number of pixel accesses is about $2N \times M$. Therefore, our algorithm is better than the run-based algorithm, thus, better than the GRAY algorithm, the skeleton-based algorithm, and graph-based algorithm.

On the other hand, as introduced above, the HCS algorithm calculates the Euler number of a binary image by labeling foreground pixels and background pixels respectively. Because for processing a pixel in labeling operation, we do not only need to check some of its neighbor pixels, but also need to do label-equivalence resolving, it is hard to analyze its time complexity accurately[†††]. We will compare this algorithm with our algorithm by testing them on various practical images in the next section.

5. Experimental Results

Images used for testing were composed of artificial images (including noise images and specialized patterns), natural images obtained from the Standard Image Database (SIDBA) developed by the University of Tokyo[††††] and the image database of the University of Southern California[†††††], texture images downloaded from the Columbia-Utrecht Reflectance and Texture Database[††††††], and medical images obtained from a medical image database of the University of Chicago.

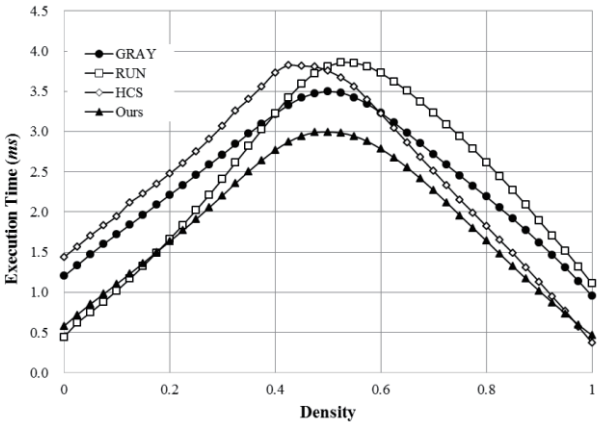

Fig. 5 Execution time versus density of an image.

Because our algorithm is an improvement of the GRAY algorithm, the run-based algorithm, denoted as *RUN* algorithm in this section, is the most efficient conventional Euler-number computing algorithm, and the HCS algorithm is the newest one, with the GRAY algorithm, the RUN algorithm and the HCS algorithm. All algorithms were implemented in the C language on a PC-based workstation (Intel Core i5-3470 CPU, 3.20 GHz, 4 GB Memory, Ubuntu Linux OS), and compiled by the GNU C compiler (version 4.2.3) with the option -O. All experimental results presented in this section were obtained by averaging of the execution time for 5000 runs.

5.1 Execution Time versus the Density of an Image

41 noise images with a size of 512×512 pixels, which were generated by thresholding of the images containing uniform random noise with 41 different threshold values from 0 to 1000 in steps of 25, were used for testing the execution time versus the density of the foreground pixels in an image. The results are shown in Fig. 5. We can find that our algorithm is much better than the GRAY algorithm for all images, is better than the HCS algorithm for all images except for the images whose densities are over 95%, and is also better than the RUN algorithm for all images whose densities are over 20%.

5.2 Comparisons in Terms of the Maximum, Mean, and Minimum Execution Times on Various Kinds of Real Images

50 Natural images, 25 medical images, 7 texture images, and 5 artificial images with specialized shape patterns were used for this test. The resolution of all of these images is 512×512 pixels. The results of the comparisons are shown

[†]A basic square is a pattern $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ in a binary image.

[††]A basic right-angled triangle is one of the patterns $\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$, $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$, $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$, and $\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$ in a binary image.

[†††]The authors of the HCS algorithm could not give the time complexity of the algorithm as well.

[††††]http://sampl.ece.ohiostate.edu/data/stills/sidba/index.htm

[†††††]http://sipi.usc.edu/database/

[††††††]http://www1.cs.columbia.edu/CAVE/software/curet/

Table 1 Maximum, mean, and minimum execution times (*ms*) on various types of images.

Image Type		GRAY	RUN	HCS	Ours
Natural	Max.	1.86	1.69	1.97	1.34
	Mean	1.42	1.07	1.40	0.86
	Min.	1.10	0.61	0.87	0.55
Medical	Max.	1.47	1.07	1.50	0.89
	Mean	1.29	0.92	1.25	0.72
	Min.	1.17	0.75	0.91	0.63
Textural	Max.	1.73	1.66	1.60	1.16
	Mean	1.38	1.35	1.10	0.83
	Min.	1.00	1.04	0.51	0.49
Artificial	Max.	1.11	1.03	1.35	0.56
	Mean	0.70	0.67	0.70	0.35
	Min.	0.28	0.24	0.32	0.16

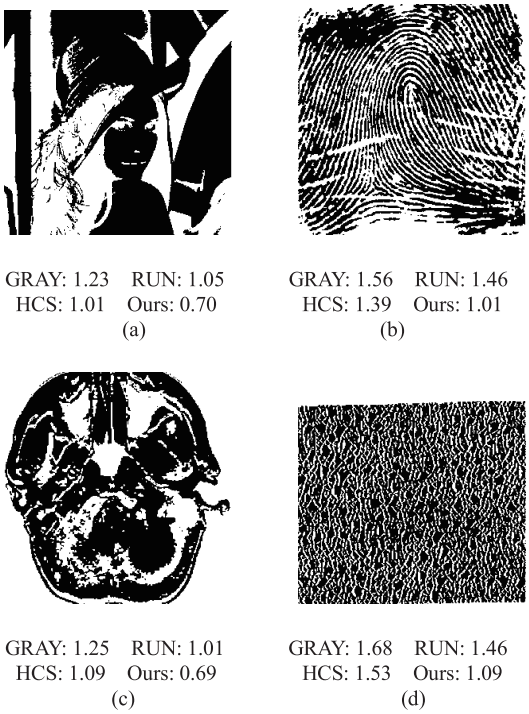

Fig. 6 The execution times (*ms*) for the selected 4 images: (a) a portrait image; (b) a fingerprint image; (c) a medical image; (d) a texture image.

in Table 1. From Table 1, for all types of images, our algorithm is much more efficient than GRAY algorithm, the RUN algorithm and the HCS algorithm for the minimum time, the average time and the maximum time. In fact, for each image used in this test, our algorithm is better than any of the other three algorithms. The execution times (in *ms*) for the selected 4 images are illustrated in Fig. 6, where the foreground pixels are displayed in black.

6. Conclusion

In this paper, we presented an efficient strategy for improving the bit-quad-based Euler number computing algorithm. By use of the information obtained during processing the previous bit quad, our algorithm can avoid checking pixels repeatedly, unlike the GRAY algorithm. Experimental results on various types of images demonstrated that our algorithm outperformed the GRAY algorithm and other conventional Euler number calculating algorithms. For future work, we will consider hardware implementation and parallel implementation of our algorithm.

Acknowledgments

We thank the anonymous referee for his/her valuable comments that improved this paper greatly. We are grateful to the associate editor, Dr. Yoshihisa Ijiri, for his kind cooperation and a lot of valuable advices. This work was supported in part by the Grant-in-Aid for Scientific Research (C) of the Ministry of Education, Science, Sports and Culture of Japan under Grant No. 23500222, and the Foundation of Xi'an Science and Technology Bureau under Grant No.CXY1343.

References

[1] R.C. Gonzalez and R.E.Woods, Digital Image Processing, Third ed., Pearson Prentice Hall, Upper Saddle River, 2008.
[2] A. Hashizume, R. Suzuki, H. Yokouchi, H. Horiuchi, and S. Yamamoto, "An algorithm of automated RBC classification and its evaluation," Bio Medical Engineering, vol.28, no.1, pp.25–32, 1990.
[3] S.N. Srihari, "Document image understanding," Proc. ACM/IEEE Joint Fall Computer Conference, pp.87–95, Dallas, TX, Nov. 1986.
[4] P.L. Rosin and T. Ellis, "Image difference threshold strategies and shadow detection," Proc. British Machine Vision Conference, pp.347–356, Sept. 1995.
[5] S.K. Nayar and R.M. Bolle, "Reflectance-based object recognition," Int. J. Comput. Vis., vol.17, no.3, pp.219–240, 1996.
[6] B. Horn, Robot Vision, pp.73–77, New York, McGraw-Hill, 1986.
[7] M.H. Chen and P.F. Yan, "A fast algorithm to calculate the Euler number for binary images," Pattern Recognit. Lett., vol.8, no.5, pp.295–297, 1988.
[8] F. Chiavetta and V. Gesu, "Parallel computation of the Euler number via connectivity graph," Pattern Recognit. Lett., vol.14, pp.849–859, 1993.
[9] Dey, S., Bhattacharya, B.B., Kundu, M.K., and Acharya, T. (2000). A fast algorithm for computing the Euler number of an image and its VLSI implementation. Thirteenth International Conference on VLSI Design, 2000. pp.330–335, 2000.
[10] C.R. Dyer, "Computing the Euler number of an image from its quadtree," Computer Graphics and Image Processing, vol.13, no.3, pp.270–276, 1980.
[11] H. Samet and H. Tamminen, "Computing geometric properties of images represented by linear quadtrees," IEEE Trans. Pattern Anal. Mach. Intell., vol.7, no.2, pp.229–240, 1985.
[12] S.B. Gray, "Local properties of binary images in two dimensions," IEEE Trans. Comput., vol.C-20, pp.551–561, 1971.
[13] C.M. Thompson and L. Shure, Image Processing Toolbox, The Math Works, Incorporated, 1993.
[14] A. Bishnu, B.B., Bhattacharya, M.K. Kundu, C.A. Murthy, and T. Acharya, "A pipeline architecture for computing the Euler number

of a binary image," J. Systems Architecture, vol.51, no.8, pp.470–487, 2005.
[15] S.J.L. Diaz-de-Leon and J.H. Sossa-Azuela, "On the computation of the Euler number of a binary object," Pattern Recognit., vol.29, no.3, pp.471–476, 1996.
[16] L. He, Y. Chao, and K. Suzuki, "An algorithm for connected-component labeling, hole labeling and euler number computing," J. Computer Science and Technology, vol.28, no.3, pp.469–479, 2013.
[17] L. He, Y. Chao, and K. Suzuki. "An efficient first-scan method for label-equivalence-based labeling algorithms," Pattern Recognit. Lett., vol.31, no.1, pp.28–35, 2010.
[18] A. Ohta, "On the derivation of the euler number," Technical report, Aichi Prefectural University, Oct. 2013. (in Japanese)