LETTER

A Further Improvement on Bit-Quad-Based Euler Number Computing Algorithm

Bin YAO[†], ***Nonmember***, **Lifeng HE**[†,††a)], ***Member***, **Shiying KANG**[†††], **Xiao ZHAO**[†], ***and*** **Yuyan CHAO**[††††], ***Nonmembers***

SUMMARY The Euler number is an important topological property in a binary image, and it can be computed by counting certain bit-quads in the binary image. This paper proposes a further improved bit-quad-based algorithm for computing the Euler number. By scanning image rows two by two and utilizing the information obtained while processing the previous pixels, the number of pixels to be checked for processing a bit-quad can be decreased from 2 to 1.5. Experimental results demonstrated that our proposed algorithm significantly outperforms conventional Euler number computing algorithms.

key words: *Euler number, topological property, object feature, computer vision, pattern recognition*

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 band. Therefore, the Euler number is a robust feature of a binary image, and it 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 [7]–[11]. Among others, there are (1) bit-quad-based algorithm proposed by Gray [12], which calculates the Euler number by counting certain 2×2 pixel patterns called bit-quads and is adopted by the famous commercial image processing tools MATLAB [13]; (2) run-based algorithm [14], which calculates the Euler number by use of the numbers of runs and the neighboring runs in the image; (3) labeling-based algorithm proposed by He, Chao and Suzuki [15], which calculates the Euler number by labeling connected components and holes in the image; (4) an improved bit-quad-based algorithm proposed [16], which reduces the number of pixels to be checked for processing a bit-quad from 4 to 2; and (5) graph-based algorithm [17], which calculates the Euler number by use of graph theory, and only needs to check 1.875 pixels for processing a bit-quad on average. For convenience, we refer the algorithms proposed in Refs. [12], [14]–[17] as to *GRAY* algorithm, *RUN* algorithm, *HCS* algorithm, *I-GRAY* algorithm, and *GT* algorithm, respectively.

This paper presents a further improved bit-quad-based algorithm for computing the Euler number in a given binary image. By scanning image rows two by two and utilizing the information obtained during processing the previous pixels, the number of pixels to be checked for processing a bit-quad can be reduced from 2 to 1.5, which leads to a more efficient processing. Experimental results showed that our proposed algorithm is more efficient than conventional Euler number computing algorithms.

2. Reviews of Conventional Bit-Quad-Based Euler Number Computing Algorithms

For an $N \times M$-size binary image, we assume that the object (foreground) pixels and non-object (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 for object pixels in this paper.

2.1 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 Q_1, Q_2, and Q_3. Let N_1, N_2, and N_3 be the numbers of patterns Q_1, Q_2, and Q_3 in a binary image, respectively. Then, the Euler number of the image, namely E, can be calculated by the following formula.

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

For each bit-quad, the GRAY algorithm checks all of

Manuscript received July 22, 2015.
Manuscript revised September 21, 2015.
Manuscript publicized October 30, 2015.

[†]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 Faculty of Information Science and Technology, Aichi Prefectural University, Nagakute-shi, 480–1198 Japan.
[†††]The author is with School of Information Engineering, Xianyang Normal University, Xianyang, Shaanxi 712000, China.
[††††]The author is with Faculty of Environment, Information and Business, Nagoya Sangyo University, Owariasahi-shi, 488–8711 Japan.
a) E-mail: helifeng@ist.aichi-pu.ac.jp (Corresponding author)

DOI: 10.1587/transinf.2015EDL8159

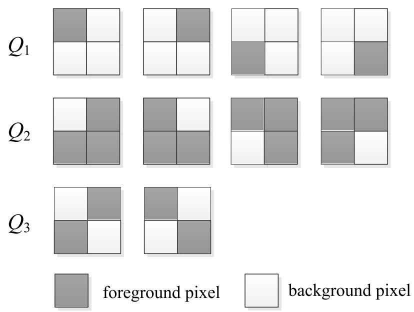

Fig. 1 Bit-quads for calculating the Euler number in the GRAY algorithm.

the four pixels in the bit-quad. Thus, the number of pixels to be checked for processing a bit-quad is 4.

2.2 I-GRAY Algorithm

The I-GRAY algorithm proposed in Ref. [16] is an improvement on the GRAY algorithm. For processing a bit-quad $\begin{bmatrix} a & c \\ b & d \end{bmatrix}$, by use of the information about the two pixels a and b, which can be obtained during processing the previous bit-quad, the I-GRAY algorithm only needs to check the pixels c and d. Thus, for the I-GRAY algorithm to process a bit-quad, the number of pixels to be checked is 2.

3. Our Improvement

As mentioned above, by utilizing the information obtained during processing the previous pixels, for processing a bit-quad, the I-GRAY algorithm can avoid checking the two pixels that have been checked during processing the previous bit-quad. However, some pixels will still be checked repeatedly in the I-GRAY algorithm. For example, for processing the first row in Fig. 2, we need to process the three bit-quads $\begin{bmatrix} 0 & a \\ 0 & b \end{bmatrix}$, $\begin{bmatrix} a & d \\ b & e \end{bmatrix}$ and $\begin{bmatrix} d & 0 \\ e & 0 \end{bmatrix}$, where we need to check the pixels a, b, d, and e. Then, for processing the second row, we need to process the three bit-quads $\begin{bmatrix} 0 & b \\ 0 & c \end{bmatrix}$, $\begin{bmatrix} b & e \\ c & f \end{bmatrix}$ and $\begin{bmatrix} e & 0 \\ f & 0 \end{bmatrix}$, where we need to check the pixels b, c, e, and f. Thus, the pixels b and e are checked repeatedly.

The number of pixels to be repeatedly checked as mentioned above can be reduced by scanning image rows two by two. For each pixel x in the scan, we check the related six pixels, i.e., $\begin{bmatrix} x & X \\ y & Y \\ z & Z \end{bmatrix}$, to decide whether the two bit quads $\begin{bmatrix} x & X \\ y & Y \end{bmatrix}$ and $\begin{bmatrix} y & Y \\ z & Z \end{bmatrix}$ are the patterns to be counted or not simultaneously. For convenience, we denote the two bit-quads as to BQ_1 and BQ_2, respectively.

When processing $\begin{bmatrix} x & X \\ y & Y \\ z & Z \end{bmatrix}$, similar as in the I-GRAY algorithm, because the pixels x, y, and z have been checked during processing the previous pixel in the scan, we only need to check the pixels X, Y and Z. Obviously, there are eight states, as shown in Fig. 3, to be considered.

0	a	d	0
0	b	e	0
0	c	f	0

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

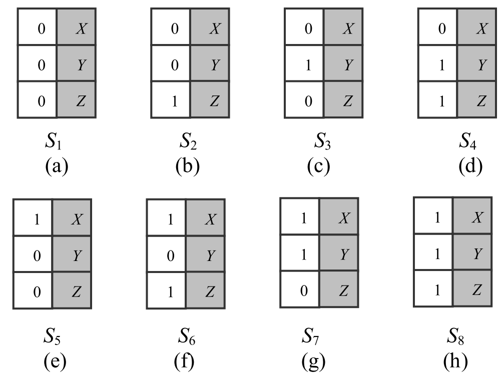

Fig. 3 Eight states defined in our improved algorithm.

Let we consider, for example, how to process in the case S_3 (Fig. 3 (c)).

(1) If the values of all pixels X, Y and Z are 0, then BQ_1 is $\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$ and BQ_2 is $\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$, both are Q_1. Moreover, the next state to be processed will be S_1 (Fig. 3 (a));

(2) If the values of the pixels X, Y and Z are (0, 0, 1), then BQ_1 is $\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$, i.e., Q_1, and BQ_2 is $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, i.e., Q_3. Moreover, the next state to be processed will be S_2 (Fig. 3 (b));

(3) If the values of the pixels X, Y and Z are (0, 1, 0), then BQ_1 is $\begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$ and BQ_2 is $\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$, both are not patterns to be counted. Moreover, the next state to be processed will be S_3 again;

(4) If the values of the pixels X, Y and Z are (0, 1, 1), then BQ_1 is $\begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$ and BQ_2 is $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$. Thus, BQ_1 is not a pattern to be counted, and BQ_2 is Q_2. Moreover, the next state to be processed will be S_4 (Fig. 3 (d));

(5) If the values of the pixels X, Y and Z are (1, 0, 0), then BQ_1 is $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, i.e., Q_3 and BQ_2 is $\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$, i.e., Q_1. Moreover, the next state to be processed will be S_5 (Fig. 3 (e));

(6) If the values of the pixels X, Y and Z are (1, 0, 1), then BQ_1 is $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ and BQ_2 is $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, both are Q_3. More-

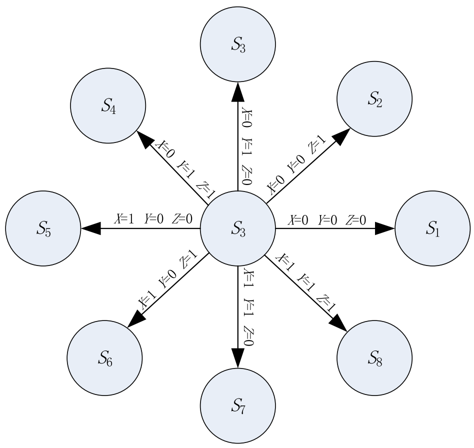

Fig. 4 State transition diagram.

over, the next state to be processed will be S_6 (Fig. 3 (f));

(7) If the values of the pixels X, Y and Z are (1, 1, 0), then BQ_1 is $\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$ and BQ_2 is $\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$. We know that BQ_1 is Q_2 and BQ_2 is not a pattern to be counted. The next state to be processed will be S_7 (Fig. 3 (g));

(8) Lastly, if the values of the pixels X, Y and Z are (1, 1, 1), then BQ_1 is $\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$ and BQ_2 is $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$, both are Q_2. Moreover, the next state to be processed will be S_8 (Fig. 3 (h)).

The state transition for case S_3 is shown in Fig. 4. Other cases can be analyzed in a similar way.

Thus, after processing the image, we can obtain the numbers of the patterns Q_1, Q_2 and Q_3, i.e., N_1, N_2, and N_3, respectively. Then, the Euler number can be calculated by the formula (1) easily.

In our implementation, for an $N \times M$-sized binary image, the pseudo codes for processing cases, for example, S_1, S_2 and S_3 in our algorithm for processing row y $(1 \leq y < M)$ can be described as follows, where x is initialized to 1. Because all pixels in the border are background pixels, we will begin our processing from state S_1.

S_1:
x increases 1;
if $x \geq N$, go to process the $(y + 2)th$ row if any;
if $X = 1$
if $Y = 1$
if $Z = 1$, go to case S_8;
else N_1 increases 1, go to case S_7;
else
N_1 increases 1;
if $Z = 1$, N_1 increases 1, go to case S_6;
else go to case S_5;
else
if $Y = 1$
N_1 increases 1;
if $Z = 1$, go to case S_4;
else N_1 increases 1, go to case S_3;
else
if $Z = 1$, N_1 increases 1, go to case S_2;
else go to case S_1;
end of if

S_2:
x increases 1;
if $x \geq N$, go to process the $(y + 2)th$ row if any;
if $X = 1$
if $Y = 1$
if $Z = 1$, N_2 increases 1, go to case S_8;
else N_3 increases 1, go to case S_7;
else
N_1 increases 1;
if $Z = 1$, go to case S_6;
else N_1 increases 1, go to case S_5;
else
if $Y = 1$
N_1 increases 1;
if $Z = 1$, N_2 increases 1, go to case S_4;
else N_3 increases 1, go to case S_3;
else
if $Z = 1$, go to case S_2;
else N_1 increases 1, go to case S_1;
end of if

S_3:
x increases 1;
if $x \geq N$, go to process the $(y + 2)th$ row if any;
if $X = 1$
if $Y = 1$
N_2 increases 1;
if $Z = 1$, N_2 increases 1, go to case S_8;
else go to case S_7;
else
N_3 increases 1;
if $Z = 1$, N_3 increases 1, go to case S_6;
else N_1 increases 1, go to case S_5;
else
if $Y = 1$
if $Z = 1$, N_2 increases 1, go to case S_4;
else go to case S_3;
else
N_1 increases 1;
if $Z = 1$, N_3 increases 1, go to case S_2;
else N_1 increases 1, go to case S_1;
end of if

Other cases can be processed in a similar way.

In our implementation method, instead of state variables, we use states transition to avoid accessing pixels repeatedly, and the information of checked pixels does not need to be stored. Therefore, for processing two bit-quads in a process, we only need to check three pixels.

According to the above analysis, for checking two bit-quads, we only need to check three pixels. In other words, for processing a bit-quad, the number of pixels to be checked in our improvement will be $3/2 = 1.5$ pixels, smaller than that in the I-GRAY algorithm, which is 2.

4. Experimental Results

In the experiments, we compared our proposed algorithm with the RUN algorithm, the HCS algorithm, the I-GRAY algorithm, and the GT algorithm. All algorithms used for our comparison 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 4.6.1) with the option –O3.

Images used for testing were composed of artificial images, natural images, texture images, and medical images.

Artificial images consist of specialized patterns (stair-like, spiral-like, saw-tooth-like, checker-board-like, and honeycomb-like connected components) and noise images. Forty-one noise images of each of five sizes (128×128, 256×256, 512×512, 1024×1024, and 2048×2048 pixels) were used for testing (a total of 205 images). For each size, the 41 noise images were generated by thresholding of the images containing uniform random noise with 41 different threshold values from 0 to 1000 in steps of 25. Because connected components in such noise images have complicated geometric shapes and complex connectivity, severe evaluations of algorithms can be performed with these images.

Natural images were obtained from the Standard Image Database (SIDBA) developed by the University of Tokyo†, and the image database of the University of Southern California††. The textural images were downloaded from the Columbia-Utrecht Reflectance and Texture Database†††, and the medical images were obtained from a medical image database of the University of Chicago.

All experimental results presented in this section were obtained by averaging of the execution time for 5000 runs.

4.1 Execution Times versus Image Sizes

We used all noise images to test the linearity of the execution time versus image sizes. The results are shown in Fig. 5. We can find that both the maximum execution times and the average execution times of the four algorithms have the ideal linear characteristics versus image sizes. For either the maximum execution time or the average execution time, that of our algorithm is much smaller than that of any of the other four algorithms.

4.2 Execution Times versus Densities of Images

Noise images with a size of 1024×1024 pixels were used for testing the execution time versus the density of the foreground pixels in an image. The results are shown in Fig. 6. We can find that our proposed algorithm is much more efficient than any of other conventional algorithms.

† http://sampl.ece.ohiostate.edu/data/stills/sidba/index.htm (June 2010)

†† http://sipi.usc.edu/database/ (September 2012)

††† http://www1.cs.columbia.edu/CAVE/software/curet/ (September 2012)

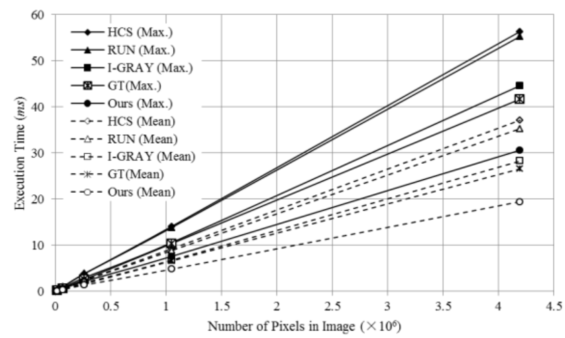

Fig. 5 Execution time (*ms*) versus the size of an image.

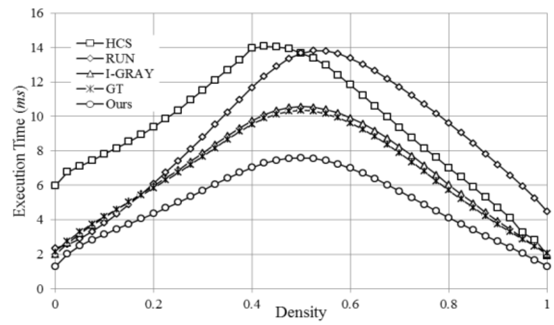

Fig. 6 Execution time (*ms*) versus the density of the foreground pixels in an image.

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

Image Type		HCS	RUN	I-GRAY	GT	Ours
Natural	Max.	1.97	1.69	1.34	1.19	0.98
	Mean	1.40	1.07	0.86	0.79	0.66
	Min.	0.87	0.61	0.55	0.54	0.43
Medical	Max.	1.50	1.07	0.89	0.82	0.69
	Mean	1.25	0.92	0.72	0.68	0.57
	Min.	0.91	0.75	0.63	0.63	0.48
Textural	Max.	1.60	1.66	1.16	1.03	0.87
	Mean	1.10	1.35	0.83	0.77	0.62
	Min.	0.51	1.04	0.49	0.53	0.36
Artificial	Max.	1.35	1.03	0.56	0.55	0.46
	Mean	0.70	0.67	0.35	0.34	0.29
	Min.	0.32	0.24	0.16	0.13	0.12

4.3 Comparisons in Terms of the Maximum, Mean, and Minimum Execution Times on Various Types of Images

Natural images, medical images, texture images, and artificial images with specialized shape patterns (stair-like, spiral-like, saw-tooth-like, checker-board-like, and honey comb-like connected components) were used for this test. The results of the comparisons are shown in Table 1. From Table 1, we can find that for all types of images, our proposed algorithm is much more efficient than any of other algorithms for the maximum execution time, the average execution time, and the minimum execution time.

5. Conclusion

In this paper, we presented a further improvement on the bit-quad-based algorithm for calculating Euler number in binary images. In our proposed algorithm, by scanning image rows two by two and utilizing the information obtained during processing the previous pixel, our proposed algorithm can further reduce the number of pixels to be checked for processing a bit-quad. Experimental results on various types of images demonstrated that our proposed algorithm outperformed conventional Euler number computing algorithms.

In principle, if we process more rows simultaneously, the number of pixels to be checked for processing a bit-quad can be further reduced. However, to do that, we need to consider more states; thus, the implementations will be much more complicated, and the efficiency will be reduced. For future work, we will extend our method to process more rows simultaneously, and find the optimal number of rows for processing.

Acknowledgments

We thank the anonymous reviewer for his/her valuable comments that improved this paper greatly. We are grateful to the associate editor, Dr. Hiroaki Kawashima, for his kind cooperation and a lot of valuable advices. This work was supported in part by the National Natural Science Foundation of China under Grant No. 61471227, the Grant-in-Aid for Scientific Research (C) of the Ministry of Education, Science, Sports and Culture of Japan under Grant No. 26330200, and the Key Science and Technology Program for Social Development of Shaanxi Province, China (No. 2014K11-02-01-13).

References

[1] R.C. Gonzalez and R.E. Woods, "Digital Image Processing," third ed., Pearson Prentice Hall, Upper Saddle River, NJ 07458, 2008.

[2] A. Hashizume, R. Suzuki, H. Yokouchi, et al., "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," International Journal of Computer Vision, vol.17, no.3, pp.219–240, 1996.

[6] B.P.K. Horn, Robot Vision, pp.73–77, McGraw-Hill, New York, 1986.

[7] H. Beri and W. Nef, "Algorithms for the Euler characteristic and related additive functionals of digital objects," Comput. Vision, Graphics Image Process, vol.28, no.2, pp.166–175, 1984.

[8] H. Beri, "Computing the Euler characteristic and related additive functionals of digital objects from their beentree representation," Computer Vision, Graphics Image Process, vol.40, no.1, pp.115–126, 1987.

[9] M.-H. Chen and P.-F. Yan, "A fast algorithm to calculate the Euler number for binary images," Pattern Recognition Letters, vol.8, no.12, pp.295–297, 1988.

[10] F. Chiavetta and V. Di Gesú, "Parallel computation of the Euler number via connectivity graph," Pattern Recognition Letters, vol.14, no.11, pp.849–859, 1993.

[11] J.L. Díaz de León S. and J.H. Sossa-Azuela, "On the computation of the Euler number of a binary object," Pattern Recognition, vol.29, no.3, pp.471–476, 1996.

[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, March 2012.

[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," Journal of Systems Architecture, vol.51, no.8, pp.470–487, 2005.

[15] L.-F. He, Y.-Y. Chao, and K. Suzuki, "An Algorithm for Connected-Component Labeling, Hole Labeling and Euler Number Computing," Journal of Computer Science and Technology, vol.28, no.3, pp.469–479, 2013.

[16] B. Yao, H. Wu, Y. Yang, Y. Chao, A. Ohta, H. Kawanaka, and L. He, "An efficient strategy for bit-quad-based Euler number computing algorithm," IEICE Trans. Inf. & Syst., vol.E97-D, no.5, pp.1374–1378, 2014.

[17] L. He, B. Yao, X. Zhao, Y. Yang, Y. Chao, and A. Ohta, "A graph-theory-based algorithm for Euler number computing," IEICE Trans. Inf. & Syst., vol.E98-D, no.2, pp.457–461, 2015.