LETTER

A Graph-Theory-Based Algorithm for Euler Number Computing

Lifeng HE[†,††a)], ***Member***, **Bin YAO**[†], **Xiao ZHAO**[†], **Yun YANG**[†*], **Yuyan CHAO**[†††], ***Nonmembers***, ***and*** **Atsushi OHTA**[††], ***Member***

SUMMARY This paper proposes a graph-theory-based Euler number computing algorithm. According to the graph theory and the analysis of a mask's configuration, the Euler number of a binary image in our algorithm is calculated by counting four patterns of the mask. Unlike most conventional Euler number computing algorithms, we do not need to do any processing of the background pixels. Experimental results demonstrated that our algorithm is much more efficient than conventional Euler number computing algorithms.

key words: *Euler number, topological property, graph theory, computer vision, pattern recognition, image analysis*

1. Introduction

The Euler number of a binary image that is defined as the difference between the number of connected components and the number of holes in the image is a basic topologic property of the binary image, which does not change when the image is stretched or flexed like an elastic rubber. It can describe the structure of objects without relation to their geometric shapes [1]. The Euler number has been used in many applications [2]–[5]. Therefore, Euler number computing is one of essential processing tasks for extracting objects' features from a binary image for pattern recognition, image analysis, and computer (robot) vision [1], [6].

Many algorithms have been proposed for calculating the Euler number of a binary image [7]–[10]. One of the most famous algorithms is based on counting certain 2×2 pixel patterns called bit-quads proposed by Gray [11], and it is used in the MATLAB image-processing tool box[**]. For convenience, we denote this algorithm as *GRAY algorithm*. Recently, an improvement of the GRAY algorithm was proposed in Ref. [12], which reduces the number of pixels to be checked for processing a bit-quad from 4 to 2 by use of already-known information obtained while processing the previous pixel. For convenience, we denote this algorithm as *I-GRAY algorithm*. According to the complex analysis and the experimental results, the I-GRAY algorithm is the most efficient Euler number computing algorithm up to now.

This paper proposes a graph-theory-based Euler number computing algorithm. It is well known that a binary image can be transmitted to a graph, and by graph theory, the Euler number of a graph can be calculated according to the numbers of vertices, edges and spaces. We show that for processing a foreground pixel, among the 16 patterns of the mask, we only need to consider four patterns. Experimental results on various types of images showed that our algorithm is much more efficient than conventional Euler number computing algorithms.

2. Proposed Algorithm

A square graph corresponding to a binary image for 8-connectivity can be constructed as follows: (1) each foreground pixel in the image is transformed to a vertex in the graph; (2) adding an edge between p and q if pixel p and pixel q are 8-connected neighbor unless it crosses with another edge. For example, the binary image shown in Fig. 1 (a) can be transformed to the graph in Fig. 1 (b), where basic right-angle triangles, each of which consists of two right-angle sides of the length 1, are called *faces*. Euler's theorem in graph theory can be described as follows [13].

Euler's Theorem *If G is a square graph, v, e, r and c are the number of the vertices, the edges, the squares and the connected components in G, respectively. Then, $v - e + r = c + 1$.*

In Euler's theorem, squares include holes, basic faces and an infinite square outside of the graph. Let h and s be the number of holes and basic faces in G, respectively, then, $r = h + s + 1$. Thus, by Euler's theorem, the Euler number E can be represented as:

$$E = c - h = v - e + s$$

Therefore, the Euler number of a binary image for 8-connectivity can also be calculated by the numbers of the vertices, spaces and edges in a graph corresponding to the image. For example, there are one connected component and two holes in Fig. 1 (a), therefore the Euler number of

Manuscript received July 30, 2014.
Manuscript revised October 2, 2014.
Manuscript publicized November 10, 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 authors are with the Graduate School of Information Science and Technology, Aichi Prefectural University, Nagakute-shi, 480–1198 Japan.
[†††]The author is with the Graduate School of Environment Management, Nagoya Sangyo University, Owariasa-shi, 488–8711 Japan.
[*]Corresponding author
a) E-mail: helifeng@ist.aichi-pu.ac.jp
DOI: 10.1587/transinf.2014EDL8155

[**]http://www.mathworks.com/access/helpdesk/help/toolbox/images/bweuler.html

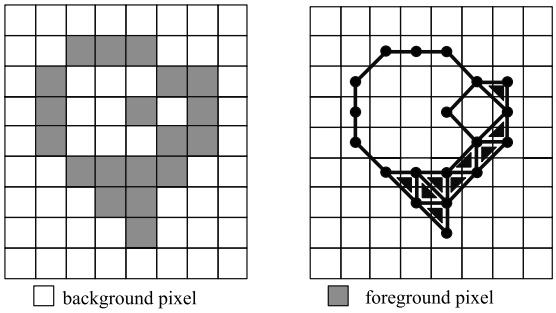

Fig. 1 A binary image (a) and a corresponding graph (b).

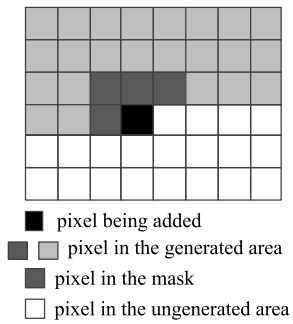

Fig. 2 Calculating the number of vertices, edges and spaces by adding a foreground pixel.

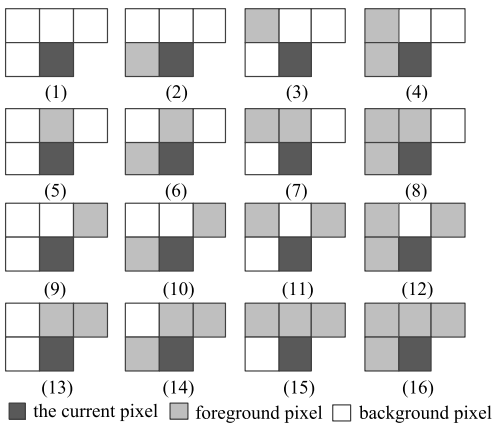

Fig. 3 The 16 patterns of the mask.

Table 1 The increments of numbers of v, e and s, and the Euler number ΔE for a pattern of the mask.

Pattern	Configuration	Δv	Δe	Δs	ΔE
$P1$		1	0	0	1
$P2$		1	1	0	0
$P3$		1	1	0	0
$P4$		1	2	1	0
$P5$		1	1	0	0
$P6$		1	2	1	0
$P7$		1	2	1	0
$P8$		1	2	1	0
$P9$		1	1	0	0
$P10$		1	2	0	-1
$P11$		1	2	0	-1
$P12$		1	3	1	-1
$P13$		1	2	1	0
$P14$		1	3	2	0
$P15$		1	3	2	0
$P16$		1	3	2	0

the image is $E = c - h = 1 - 2 = -1$; on the other hand, the numbers of c, h, v, e and s in the graph shown in Fig. 1 (b) are 1, 2, 19, 29 and 9, respectively, the Euler number of the graph is $E = v - e + s = 19 - 29 + 9 = -1$.

In practice, for calculating the Euler number of a binary image, we can count the numbers of vertices, spaces and edges without constructing a corresponding graph actually. We imagine that a binary image is constructed by adding pixels one by one in the raster scan. For each pixel $b(x, y)$ being added, we calculate the increments of the numbers of vertices, edges, and faces generated by adding the pixel. If $b(x, y)$ is a background pixel, no new vertex, edge or face will generated, thus, nothing needs to be done. Otherwise, i.e., if $b(x, y)$ is a foreground pixel, the number of vertices should increase by 1. Moreover, because edges and faces can be newly generated only between $b(x, y)$ and its four 8-neighbors in the constructed area (Fig. 2), i.e., $b(x-1, y)$, $b(x-1, y-1)$, $b(x, y-1)$ and, $b(x+1, y-1)$, the combination of which is usually called *the mask* of $b(x, y)$ [1], we can calculate the number of the edges and faces newly generated according to the configuration of the mask, which has the 16 patterns as shown in Fig. 3.

However, counting the numbers of vertices, edges, and faces generated by adding a foreground pixel one by one directly will be inefficient. In our algorithm, we consider the increment over the Euler number for each pattern of the

mask. The analysis results are shown in Table 1, where Δv, Δe, Δs, and ΔE denote the increments for the numbers of vertices, edges, spaces, and the Euler number, respectively, and $\Delta E = \Delta v - \Delta e + \Delta s$.

From the Table 1, we can find that for calculating the Euler number, among the 16 patterns of the mask, only four patterns, i.e., patterns 1, 10, 11, and 12, need to be considered.

Because we process pixels in the raster scan, when processing a pixel, we will know whether the previous pixel was a background pixel or an object pixel. Thus, as shown in Fig. 4 (a), for processing an object pixel R, we only need to check the three pixels A, B, and C in the mask.

From Table 1, for the case where R follows a background pixel, the patterns should be considered are $P1$ and $P11$. On the other hand, for the case where R is an object pixel, the patterns should be considered are $P10$ and $P12$. For convenience, we denote the two cases as $S1$ and $S2$, respectively.

In order to enhance the efficiency, for processing an object pixel R, we should check as less pixels in the mask as possible. To do that, the Karnaught map [16] can be used. The Karnaught map is a method to simplify boolean algebra expressions. In our case, the value of background pixels is considered as 0 and that of object pixels is considered as 1. Thus, for example, the values of A, B, and C for $P11$ are 1, 0, and 1, respectively.

For the cases $S1$ and $S2$, the conditions for checking pixels in the mask are shown in Fig. 4 (b) and (c), respectively. By Fig. 4 (b), the condition for checking pixels in the mask is $\neg A \wedge \neg B \wedge \neg C \vee A \wedge \neg B \wedge C = \neg B(\neg A \wedge \neg C \vee A \wedge C)$, where $\neg$, $\wedge$, and $\vee$ are for logic NOT, AND and OR, respectively, therefore, we should first check pixel B. On the other hand, by Fig. 4 (c), the conditions for checking pixels in the mask is $\neg B \wedge C$, thus, we do not need to check pixel A. The pseudo codes of our algorithm are shown in Fig. 5.

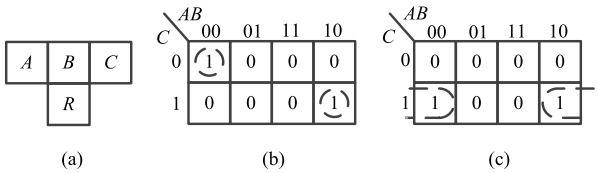

Fig. 4 Karnaught maps for $S1$ and $S2$.

```
procedure ComputingEulerNumer(b(N, M))
    n1←0
    n2←0
    for y = 1 → M−1 do
        for x = 1 → N−1 do
            if b(x, y)>0 then          // S1
                if b(x, y−1)<1 then    // check B
                    if b(x−1, y−1)>0 then // check A
                        if b(x+1, y−1)>0 then // check C
                            n2←n2+1    // patterns 11
                        end if
                    else if b(x+1, y−1)<1 then // check C
                        n1←n1+1    // patterns 1
                    end if
                end if
                x←x+1
                while b(x, y)>0        // S2
                    if b(x, y−1)<1 and b(x+1, y−1)>0 then
                        // check B and C
                        n2←n2+1 // patterns 10 or 12
                    end if
                    x←x+1
                end while
            end if
            x←x+1
        end for
        y←y+1
    end for
    no_of_euler←n1−n2
end procedure
```

Fig. 5 The pseudo codes of our algorithm.

3. Experimental Results

In this section, we compared our algorithm with the I-GRAY algorithm, which is the fastest conventional Euler number computing algorithm [12]. Both the two algorithms used were implemented in the C language on a PC-based workstation (Intel Core i5-3470 CPU, 3.20GHz, 4GB Memory, Ubuntu Linux OS), and compiled by the GNU C compiler (version 4.2.3) with the option $-O3$. All experimental results presented in this section were obtained by averaging of the execution time for 5000 runs.

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 on the noise images are shown in Fig. 6. We can find that our algorithm is much more efficient than I-GRAY algorithm.

On the other hand, six specialized-pattern artificial images (spiral-like, saw-tooth-like, checker-board-like, upward-stair-like, downward-stair-like, and honey comb-like connected components), 50 natural images (including landscape, aerial, fingerprint, portrait, still-life, snapshot, and text images, obtained from the Standard Image

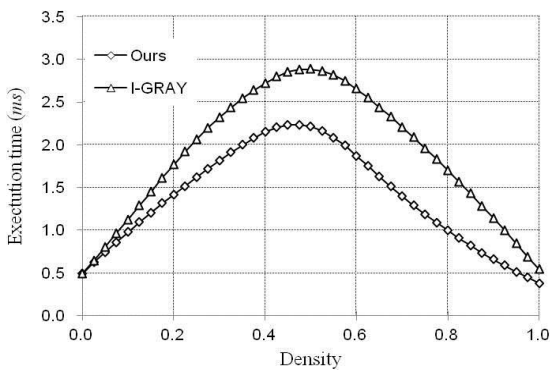

Fig. 6 Execution times versus the density of an image.

Table 2 Maximum, mean, and minimum execution times (*ms*) on various types of images.

Image type		I-GRAY	Ours
Natural	Max.	1.34	1.02
	Mean	0.86	0.71
	Min.	0.55	0.49
Medical	Max.	0.89	0.73
	Mean	0.72	0.62
	Min.	0.63	0.54
Textural	Max.	1.16	0.92
	Mean	0.83	0.68
	Min.	0.49	0.41
Artificial	Max.	0.56	0.49
	Mean	0.35	0.28
	Min.	0.16	0.11

Database (SIDBA)[†] and the image database of the University of Southern California[††]), 7 texture images (downloaded from the Columbia-Utrecht Reflectance and Texture Databas[†††]), and 25 medical images (obtained from database of The University of Chicago) are used for testing maximum, mean, and minimum execution times (*ms*), where all of these images were 512×512 pixels in size, and were transformed into binary images by means of Otsu's threshold selection method [15].

The results of the comparisons are shown in Table 2. From Table 2, we can find that our algorithm is much more efficient than the I-GRAY algorithm. In fact, for any image used in this test, our algorithm is more efficient than the I-GRAY algorithm.

4. Discussion

As introduced in above, for processing a pixel, the I-GRAY algorithm needs to check two pixels. By our algorithm, we should first check whether the current pixel is a foreground pixel or a background pixel. In the case where the current pixel is a background pixel, nothing else needs to be done. Otherwise, if the current pixel $b(x, y)$ is a foreground pixel, by the pseudo codes given in Sect. 3, we will check pixels in the mask in the order $b(x, y-1) \rightarrow b(x+1, y-1) \rightarrow b(x-1, y-1)$.

We first consider the cases where $b(x, y)$ follows a background pixel (one of the patterns 1, 3, 5, 7, 9, 11, 13, and 15 in Table 1). If $b(x, y-1)$ is a foreground pixel (one of the patterns 5, 7, 13 and 15), no other pixel would be checked, otherwise, $b(x+1, y-1)$ and $b(x-1, y-1)$ will be also checked. For the cases where $b(x, y)$ follows another foreground pixel (one of the patterns 2, 4, 6, 8, 10, 12, 14, and 16), if $b(x, y-1)$ is a foreground pixel (one of the patterns 6, 8, 14 and 16), we do not need to check other pixels, otherwise, we also need to check $b(x+1, y-1)$. Thus, when $b(x, y)$ is a foreground pixel, the average number of times for checking pixels in the mask is $(4\times1+4\times3+4\times1+4\times2)/16 = 1.75$. Therefore, the average number of times for checking pixels for processing a pixel is $(1 + (1 + 1.75))/2 = 1.875$, which is smaller than the I-GRAY algorithm and any of other conventional algorithms.

It is worth to mention that, in our algorithm, only four special patterns need to be counted, while in the I-GRAY algorithm, 10 special patterns need to be counted. Moreover, our algorithm does not need to do anything for background pixels, but the I-GRAY algorithm does. These should be the main reasons that our algorithm is more efficient than the I-GRAY algorithm.

Moreover, it is worth to mention that, although we introduced our algorithm by use of raster-scan access, the same as in the GRAY algorithm and I-GRAY algorithm, in our algorithm, different rows of the given image can be processed simultaneously. Therefore, our algorithm can be also parallelized easily.

5. Concluding Remarks

This paper proposed a graph-theory-based Euler number computing algorithm. Through analyzing the patterns of the mask based on graph theory, we only need to consider four patterns of the mask. By our algorithm, the average number of pixels necessary to check for processing a pixel is only 1.875. Experimental results on various kinds of images demonstrated that our algorithm is much more efficient than conventional Euler number computing algorithms.

Acknowledgments

We thank the anonymous referee for their valuable comments that improved this paper greatly. We are grateful to the associate editor Prof. Hitoshi Habe for his kind cooperation. This work was supported in part by the National Natural Science Foundation of China under Grant No.61471227 and the Grant-in-Aid for Scientific Research (C) of the Ministry of Education, Science, Sports and Culture of Japan under Grant No.26330200.

[†]http://sampl.ece.ohio-state.edu/data/stills/sidba/index.htm
[††]http://sipi.usc.edu/database/
[†††]http://www1.cs.columbia.edu/CAVE/software/curet/

References

[1] R.C. Gonzalez, 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. of Comput. Vis, vol.17, no.3, pp.219–240, 1996.
[6] B. Horn, Robot Vision, pp.73–77, McGraw-Hill, New York, 1986.

[7] M.H. Chen and P.F. Yan, "A fast algorithm to calculate the Euler number for binary image," Pattern Recognit. Lett, vol.8, no.5, pp.295–297, 1988.
[8] L. Juan and H. Juan, "On the computation of the Euler number of a binary object," Pattern Recognit, vol.29, no.3, pp.471–476, 1996.
[9] S. Zenzo, L. Cinque, and S. Levialdi, "Run-based algorithms for binary image analysis and processing," IEEE Trans. Pattern Anal. Mach. Intell., vol.18, no.1, pp.83–89, 1996.
[10] 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.468–478, 2013.
[11] S.B. Gray, "Local properties of binary images in two dimensions," IEEE Trans. Comput., vol.C-20, pp.551–561, 1971.
[12] 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, May 2014.
[13] B.M. Douglas, WIntroduction to Graph Theory — Second edition, Prentice Hall, 2001.
[14] 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.
[15] N.A. Otsu, "Threshold selection method from gray-level histograms," IEEE Trans. Syst. Man Cybern., vol.SMC-9, pp.62–66, 1979.
[16] M. Karnaugh, "The map method for synthesis of combinational logic circuits," Trans. AIEE. pt I, vol.72, no.9, pp.593–599, 1953.