British Journal of Mathematics & Computer Science
20(3): 1-29, 2017; Article no.BJMCS.30783
ISSN: 2231-0851

SCIENCEDOMAIN *international*
www.sciencedomain.org

Generation of Checkered Patterns and Their Variations by Making Use of Eulerian Graph Features

Tomio Kurokawa[1*]

[1]*Department of Information Science, Aichi Institute of Technology, 1247 Yachigusa, Yagusa-Cho, Toyota 470-0392, Japan.*

Author's contribution

The sole author designed, analyzed and interpreted and prepared the manuscript.

Article Information

DOI: 10.9734/BJMCS/2017/30783
Editor(s):
(1) Doina Bein, Applied Research Laboratory, The Pennsylvania State University, USA.
Reviewers:
(1) André Luiz Pires Guedes, Federal University of Paraná, Brazil.
(2) C. D. Guruprakash, Visvesvaraya Techhnological University, Karnataka, India.
(3) Liancui Zuo, Tianjin Normal University, Tianjin, China.
(4) Dipa Dixit, Fr. C. Rodrigues Institute of Technology(FrCRIT), Mumbai University , India.
Complete Peer review History: http://www.sciencedomain.org/review-history/17639

Original Research Article

Received: 30th November 2016
Accepted: 16th January 2017
Published: 27th January 2017

Abstract

A variety of pictures of checkered pattern graphics appear on the Internet. However, rarely seen is how those graphics were produced. This paper shows various checkered patterns and their variations generated automatically by a computer program, making use of features of Eulerian Graph. Given a line drawing of binary images, the procedure extracts contours of the line figures. The contours in turn form a set of contour cycles of the regions surrounded by the original lines of the figures, which are supposed to form an Eulerian Circuit. The drawing can be various combinations of closed lines -- circles, rectangles, ellipses, any of closed drawings. Bi-partitioning the contours into partitions (X, Y), filling inside the regions of X and Y, the procedure outputs the checkered patterns or their likes. Various patterns were successfully produced in a number of experiments as well as demonstrating that the algorithm is very robust and that some of the patterns are aesthetic. Since the checkered patterns are used in the Emblem of Tokyo Olympic 2020, it is timely to do the demonstration about the design of checkered patterns and their relations with computer programs and mathematics.

Keywords: Checkered patterns; closed one stroke curve; Eulerian graph; contour cycles; region adjacency graph; dual; bipartite graph.

**Corresponding author: E-mail: kurokawa@aitech.ac.jp;*

1 Introduction

Checkered patterns (Ichimatsu Moyo in Japanese) are to appear, or it might be better saying to have appeared on the Emblem of Tokyo Olympic Games 2020 [1]. Checkered patterns are recognized as traditional arts and widely known in Japan as well as in the world.

Images of various checkered patterns [2] can be seen on the Internet. But we could rarely see articles written about how they are drawn, especially the methods to automatically draw the patterns by a computer program. This paper is intended to demonstrate various checkered patterns and their variations automatically generated by a computer program using the method [3], which was previously published but without demonstrations of generating checkered patterns.

The generation method of checkered patterns in this study first concerned with contour extraction of binary images, especially binary line drawing. This contour extraction is realized by using the OpenCV library [4]. The specific function of contour extraction of the library is based on the study by Suzuki and Abe [5]. In the study, they provide a method to obtain the topological structure and the contours of the binary image at the same time. In the software [4], the topological structure is realized in a tree and the contours are by sequences of coordinates. They call the contour extraction as "border following". It provides two types of borders -- "outer border" and "hole border." These two are important for the present study. Researches concerning "border following" also appear in [6-8].

Suzuki and Abe reported that their study [5] was extended from [6]. This contour extraction using the library [4] has been successfully used in [9-11].

Another matter about this study after extraction of contours is bi-partitioning of the contours. The construction of bipartite graph usually appears in a textbook, such as [12]. But in the present study, the extracted contours are bi-partitioned into two sets, just separating the contours into two sets. The final step is the actual region painting, to which the present study does not pay too much attention.

2 Outline of the Study and Contour Extraction of Line Figures

2.1 Outline of overall procedure

The procedure of this study first extract contours from an original binary line figure which is supposed to constitute Eulerian Circuit [12]. Intersections of the lines of the figure are considered as vertices and the lines between the intersections are edges. The extracted contours are the cycles which surround each of the regions surrounded by the originally drawn curves. If two contours surrounded regions are adjacent, then the corresponding two contours are adjacent, that means the two contours are very near or very close. Based on this nearness between contours, the contours can be bi-partitioned. Bi-partitioning algorithm [3] just separates them into X-partition and Y-partition. The regions of one partition are painted with a color and those of another partition are with another color -- 2-coloring, which produces the final checkered patterns or their variations.

2.2 Contour extraction of line figures

The contour extracting procedure of this study, which appears in [10], works as follows:

1. Prepare a line figure binary image.
2. Execute the contour extraction function of the library [4] for the image with obtaining a tree, which holds the result.
3. Visit each node in the tree and obtain a set of series of coordinates of the contours.

The extracted contours are organized as a tree [5,10] represent the topological features of the contours and series of two dimensional coordinates for each contour. The root node of the tree represents the outermost contour of the line figure, the so called "outer border" [5]. All the nodes other than the root are just one level under the root, representing the inside contours, or "hole borders" [5] for the case of the figures of the present study. All points of contours, regardless outer or inside, are on the lines of the original figures. The outer contour points are represented counter-clockwise and the inside contour clockwise.

The obtained contours are the set of contour cycles which surround each of the regions surrounded by the originally drawn figure curves. The processing of this set of contours is explained in the next section.

3 Rephrasing and Modifying Algorithm of Previous Studies

The previous study [3] provides a method that generates a maze, where the solution path is the given one-stroke curve itself. In the study, an algorithm was introduced to make the bi-partitioning of the contours extracted from the one-stroke curve. The curve need not be literally one-stroke. Any curve having the crosses (or vertices) with an even number of lines (edges) can be handled. This includes a diagram made from a number of closed curves. That is, the line figure such as Fig. 1, grids, or a combination of a number of rectangles or closed curves are accepted. For such a figure, all crosses (vertices) have even number of edges, where extracted contours can be partitioned into sets [3].

Fig. 1 is an original drawing of two rectangles, the line width of which is 1, from which checkered patterns were produced. These rectangles were drawn by using OpenCV library with line thickness of 1. The program with VC++ and OpenCV for this study first extracts the contours of the lines from the figure. They are shown in Fig. 2. The contours are on the lines of the original figure. However, they are also the contours of regions surrounded by the drawn curves (lines). Actually, they are closed, hence cycles. They are numbered from 1 to 6 by the OpenCV program library as shown in Fig. 2. No.1 is for the contour of the outer region, or the outer region itself. They are double lines everywhere on the curves. However, those curves of Fig. 2 do not look double. It is because the contours overlap everywhere on the curves. Hence, Fig. 2 looks exactly the same as the original curves, Fig. 1. This means that all the contour lines are on the original image lines.

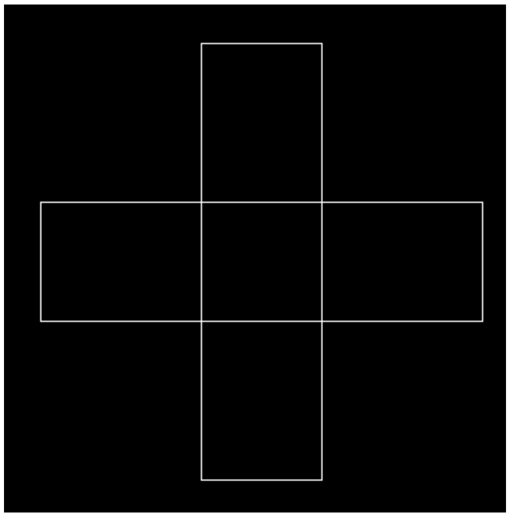

Fig. 1. Original drawing of two rectangles overlapped with line width of 1

Since a figure made of closed lines (foreground white line) constitutes an outer region, a connected figure lines and a number of hole regions, the extracted contours of the figure lines should be made of an "outer" contour and a number of "hole" contours, both on the original figure lines. Since the figure is made of lines of one-pixel width, all contour line parts are on the original one-pixel figure lines. That's why contours look like consisting of single lines, but actually completely overlapped. Accordingly, every line part is made of two borders –- either an "outer border" and a "hole border", or two "hole borders". Despite extracting contours of line figure are considered a kind of unusual processing, results were successful and reasonable. The contour extraction function of OpenCV [4] and its theoretical base [5] should be highly evaluated.

Since the dual [13] of an Eulerian planar graph is a bipartite graph, the region adjacency graph (RAG), which can be obtained by complete checking of contour adjacency, shown as yellow in Fig. 3, where the regions are considered as vertices and the edges exist between the adjacent regions, constitutes the bipartite graph. This process of obtaining RAG is not in the bi-partitioning algorithm [3] in subsection 3.1.

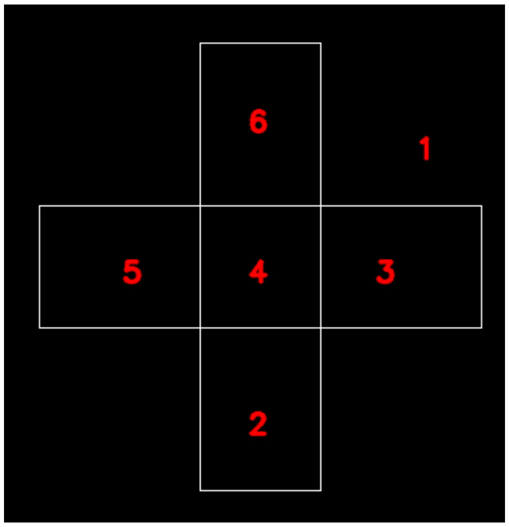

Fig. 2. Extracted contours of two rectangles overlapped, 6 contours

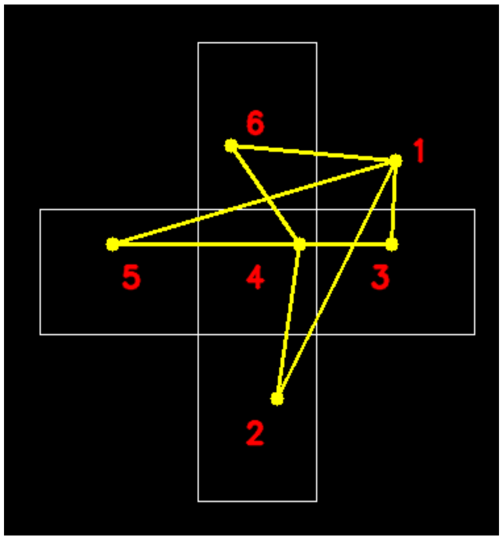

Fig. 3. Extracted contours constitute a bipartite graph and the regions are numbered

The algorithm introduced in [3] can be applied to make the partitions (X, Y) and is rephrased with more clarity:

3.1 Bi-partitioning algorithm

Input: A set, CONTOURS, of all the extracted n contour cycles as in Fig. 2 and is expressed as

$$CONTOURS = \{ cycle_j \mid 1 \le j \le n\}. \quad (1)$$

With n_j the length of $cycle_j$, each $cycle_j$ is expressed as

$$cycle_j = \{ \big(x(j,i), y(j,i)\big) \mid 1 \le i \le n_j\}. \quad (2)$$

Output: Two partitions called *X-partition* and *Y-partition,* obtained from the bipartite graph, are disjoint and the union of the two is $CONTOURS$ of (1).

Definitions: The distance from $cycle_j$ to $cycle_k$ is defined as:

$$d(j,k) = min_{1\le p\le n_j \,\&\, 1\le q\le n_k} \sqrt{\big(x(j,p)-x(k,q)\big)^2 + \big(y(j,p)-y(k,q)\big)^2}. \quad (3)$$

Method:
Assumption: n: initial number of contour cycles in *CONTOURS*
X, *Y*: partition sets of the bipartite graph, both sets are empty at the beginning; they will have the resulted vertices of bi-partition.

```
1. Remove an arbitrary $cycle_j$ from CONTOURS and put it in X;
2. Repeat begin
3.     For each $j$ in X do begin
4.         For each $i$ in CONTOURS do begin
5.             If $j$ and $i$ are adjacent, do begin
6.                 Remove $i$ from CONTOURS and put it in Y;
7.                 If no cycle remains in CONTOURS, do begin
8.                     If the outer cycle is in Y, exchange X and Y;
8.                     Paint inside all CONTOURS with different colors for X and Y and finish;
10.                end;
11.            end;
12.        end;
13.    end;
14.    Exchange X and Y;
15. end;
```

Contour adjacency: The adjacency checking is based on distance computation between two cycles. The basic criteria for the adjacency is based on the overall average of the shortest distance (d_{AS}) between two cycles $cycle_j$ and $cycle_i$, where d_{AS} is defined as:

$$d_{AS} = \frac{1}{\sum_{j=1}^{c} n_j} \sum_{j=1}^{c} \sum_{l=1}^{n_j} \min_{1\le i\le c} \min_{1\le k\le n_i} d(j,l,i,k), \quad (4)$$

$where\ d(j,l,i,k)$ is defined as

$$d(j,l,i,k) = \sqrt{(x(i,k)-x(j,l))^2 + (y(i,k)-(y(j,l))^2}, \quad (5)$$

$where\ j \ne\ \ i\ and\ n_j$ is the number of points on $cycle_j$.

Let's define two cycles $cycle_j$ and $cycle_i$ are adjacent if there are more than a certain number, β, of points on $cycle_j$ such that the shortest distance, $d_S(j,p_l,i)$, from a point p_l on $cycle_j$ to $cycle_i$ is less than d_{AS} multiplied by a certain number, α, where $d_S(j,p_l,i)$ is expressed as:

$$d_S(j, p_l, i) = \min_{1 \le k \le n_i} d(j, l, i, k). \tag{6}$$

So if more than β such different points on $cycle_j$ exist, $cycle_j$ and $cycle_i$ are considered adjacent. That is, $cycle_j$ and $cycle_i$ are adjacent if the size of S is greater than β in the following expression:

$$S = \{p_l \mid d_S(j, p_l, i) \le \alpha * d_{AS} \text{ and } 1 \le l \le n_j \text{ where } a \text{ is a small positive value}\}. \tag{7}$$

3.2 Experiment with Bi-partitioning and region filling

By applying the above algorithm to Fig. 2, the contour cycles are bi-partitioned to X-partition (Fig. 4) and Y-partition (Fig. 5). Although they are interchangeable, the partition with the most outer contour is designated as X-partition in this study. X-partition includes the contours of 1, and 4. While Y-partition holds 2, 3, 5, and 6. Fig. 6 is obtained by examining all the adjacencies between the contours. The graph (yellow) in Fig. 3 and the one in Fig. 6 represent the same graph. Filling the regions of X-partition with yellow, Fig. 7 is obtained, so called checkered patterns.

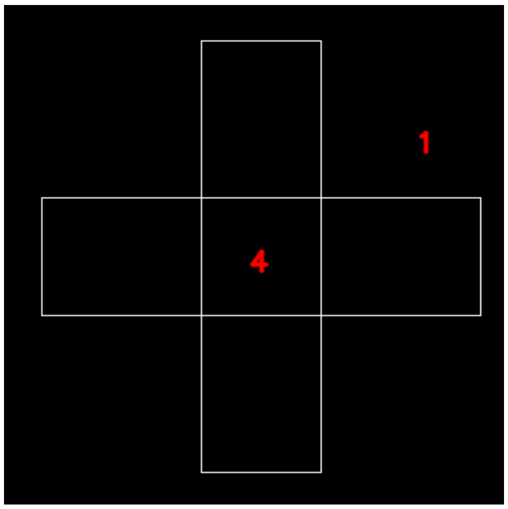

Fig. 4. X-partition, 2 contours

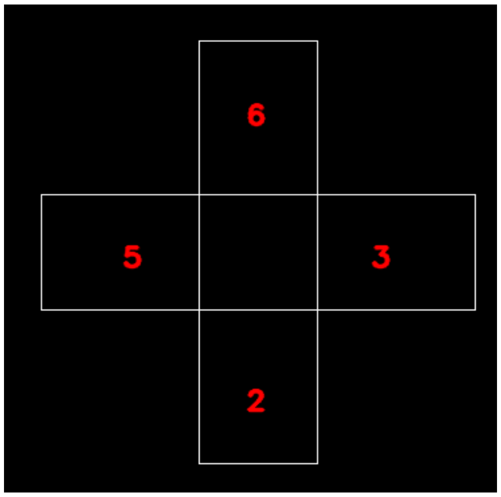

Fig. 5. Y-partition, 4 contours

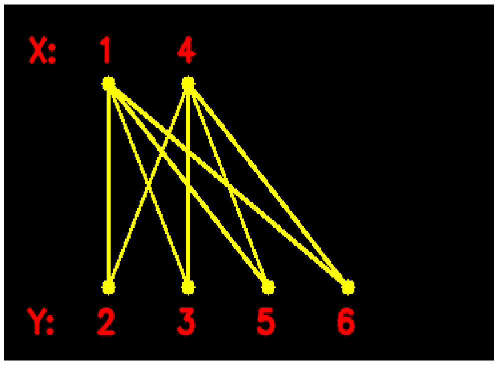

Fig. 6. Bi-partitioned graph

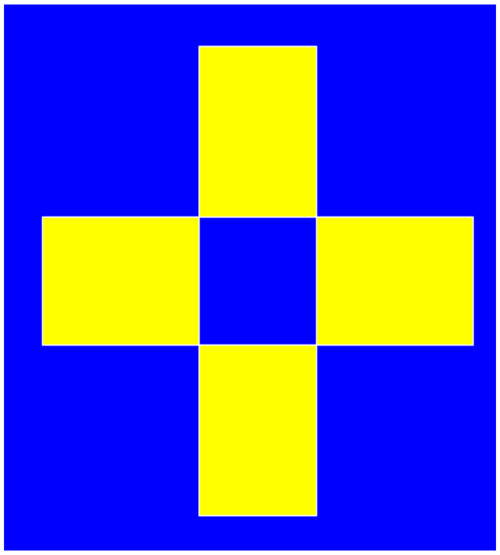

Fig. 7. Checkered patterns: X-partition is filled with blue, and Y-partition and boundary

4 More Experiments with Complex Figures and Discussion

4.1 Two rectangles overlapped but with line width 5

In these experiments, the algorithm was applied to the diagram of Fig. 8, which was drawn using OpenCV library with line thickness of 5, the same shape figure as Fig. 1, but with different line width. Contours were extracted as in Fig. 9. All the contours clearly appeared and no overlap occurred this time. By comparing Figs. 8 and 9, we notice that contours are for the drawn lines as well as for the regions surrounded by the drawn lines. And the figure clearly shows how each of the contours is related to its corresponding region. The contours were bi-partitioned into X-partition (Fig. 10) and Y-partition (Fig. 11). These figures clearly illustrate which contour belongs to which partition. Although the two partitions are interchangeable, the partition with the most outer contour is designated as X-partition in this study. Hence, X-partition includes the contours of 1, and 4, and Y-partition holds 2, 3, 5, and 6. Fig.12 (RAG) and Fig. 13 were obtained by checking all the adjacencies between the contours. The graph (yellow) in Fig. 12 is exactly the same as the bipartite graph (yellow) of Fig. 3. And the graph of Fig. 13 is the same as that of Fig. 6. This means the extracted contours constitute the same bipartite graphs as well as the same RAGs. This suggests that the width difference of the original line drawings causes no significant difference for the RAG. Filling the regions of X-partition with blue color and Y-partition with red color and the boundaries with white color, Fig. 14 was obtained, the checkered patterns with wider boundaries. The patterns give different impressions with wider boundaries. The successful experiments in section 3 and section 4.1 suggest that the algorithm is effective even for different line widths.

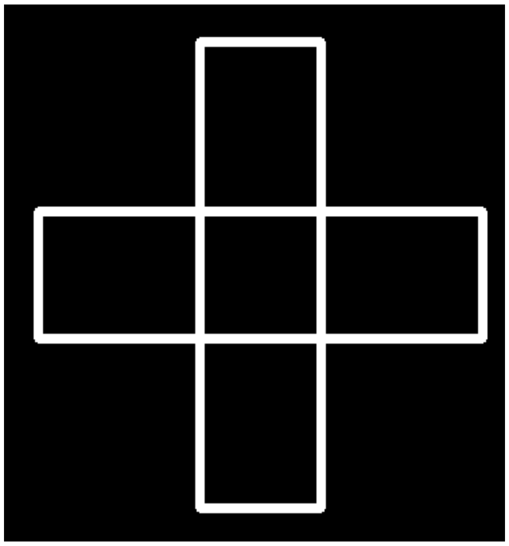

Fig. 8. Original drawing of two rectangles overlapped with line width of 5

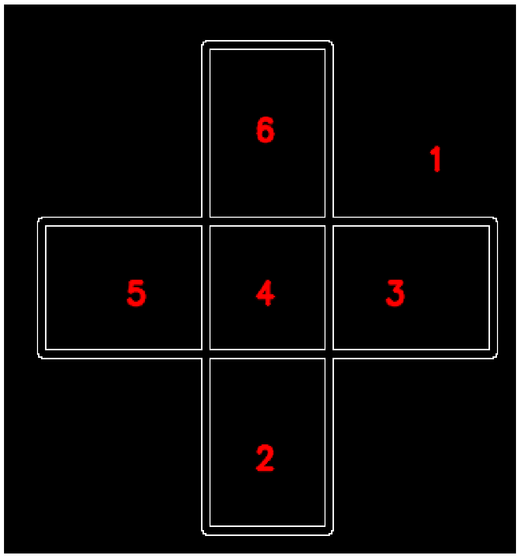

Fig. 9. Extracted contours, 6 contours and regions are numbered

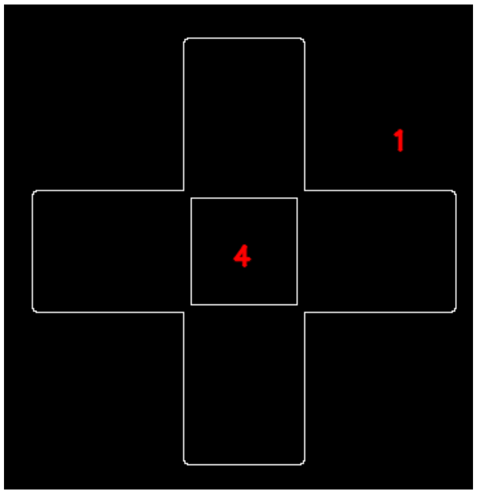

Fig. 10. X-partition, 2 contours

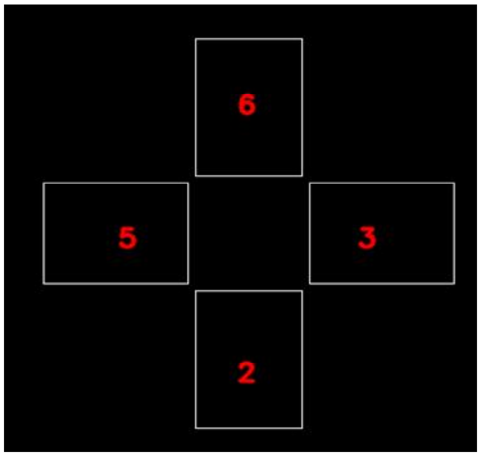

Fig. 11. Y-partition, 4 contours

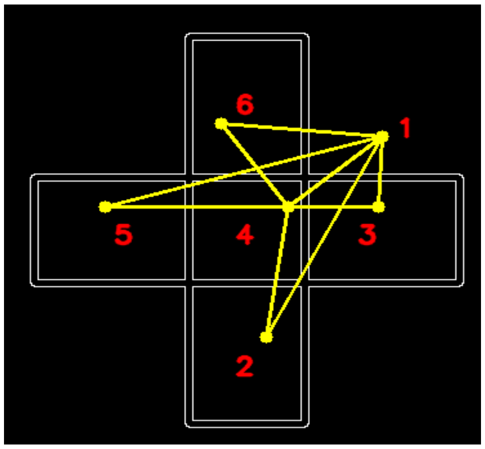

Fig. 12. Extracted contours constitute a region adjacency graph (RAG) as well as a bipartite graph

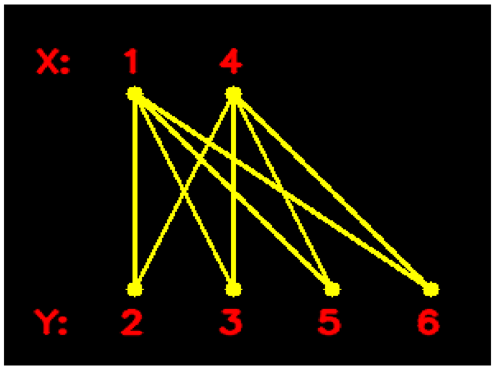

Fig. 13. Bi-partitioned graph

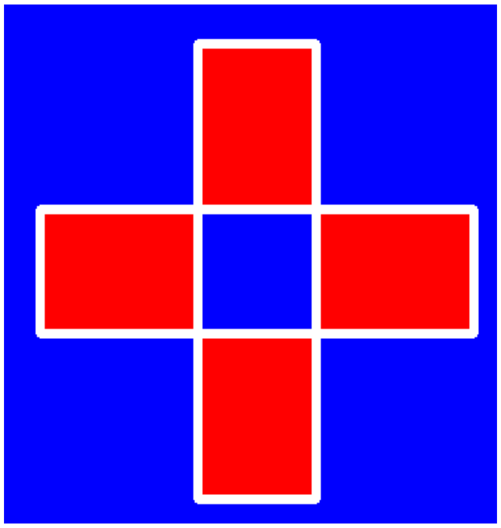

Fig. 14. Checkered patterns: X-partition blue, Y-partition red, and boundaries

4.2 A large number of grids with line width 1

Fig. 15 is the figure of 10 rectangles crossed drawn using OpenCV library with line thickness of 1. Contours were extracted as in Fig. 16. There are 102 contours. Figs. 15 and 16 look exactly the same. The lines in Fig. 16 look single everywhere on the lines. Hence, it means that the contours should be overlapped everywhere.

The bi-partitioning algorithm was applied to Fig. 16. Fig. 17 is the resulted Y-partition with 60 contours; X-partition is not shown because it's supposed to look the same as Y-partition. Fig. 18 is the checkered patterns obtained by filling the inside of all the contours of X-partition blue, Y-partition red, and the boundary white. Fig. 19 is with X-partition and boundary blue and Y-partition red, literally 2-colored. Some subtle difference seems to exist between Figs. 18 and 19.

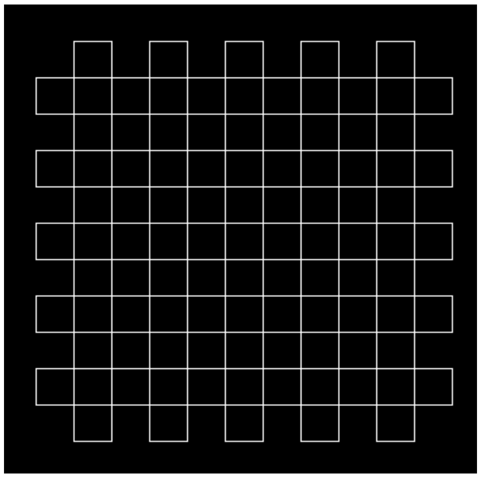

Fig. 15. Original drawing of grids: 10 overlapped rectangles with line width of 1

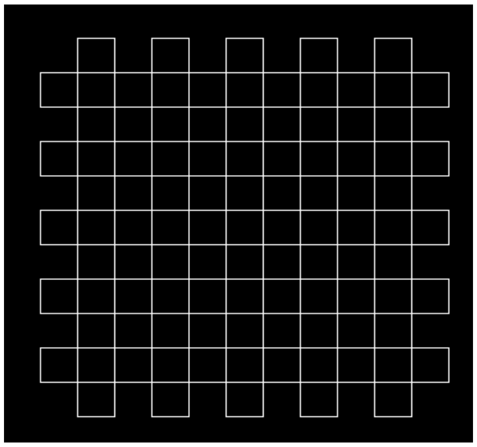

Fig. 16. Contours extracted, 102 contours

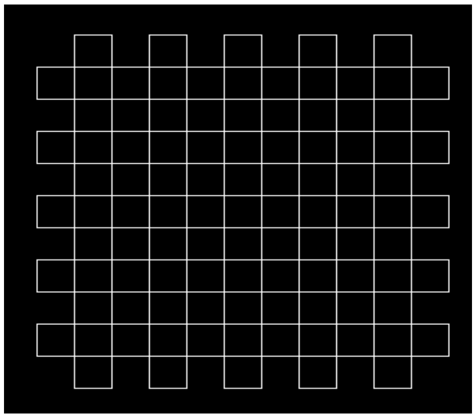

Fig. 17. Y-partition, 60 contours

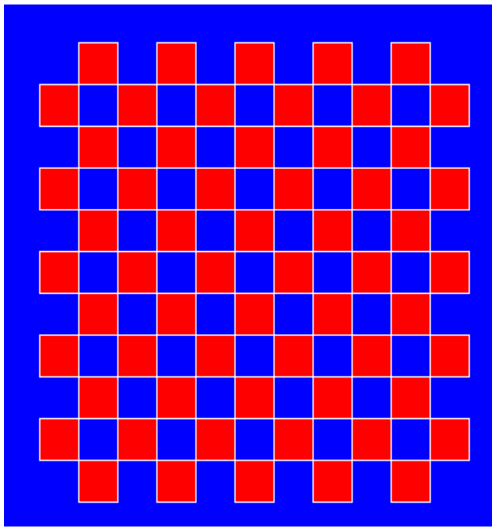

Fig. 18. Checkered patterns: X-partition with blue, Y-partition with red, and boundary white

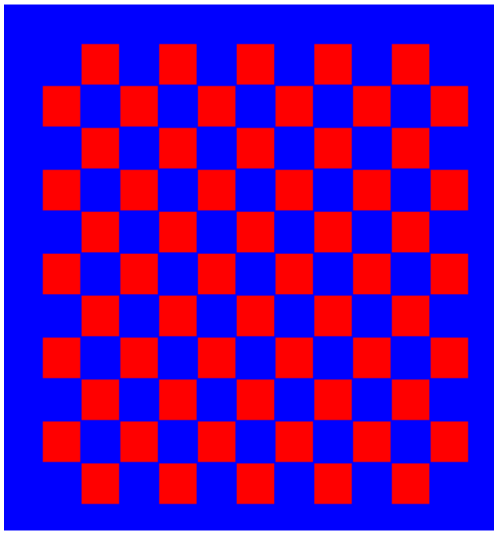

Fig. 19. Checkered patterns: X-partition and boundary blue, and Y-partition red

4.3 A large number of grids with line width 5

Fig. 20 is the figure of 10 rectangles crossed drawn using OpenCV library, but with line thickness of 5. Contours were extracted and the result is in Fig. 21. There are 102 contours shown as in Fig. 16. It seems that no contours overlap. The contours in Fig. 21 are quite different from those in Fig.16. The bi-partitioning algorithm was applied to Fig. 21. Fig. 22 is X-partition with 42 contours, one of which is the outer contour, very long, and Fig. 23 is Y-partition with 60 contours. Fig. 24 shows the checkered patterns with X-partition filled with blue color and Y-partition with red and the boundaries with white. It seems that the algorithm worked quite well, even with the different line width, this time too. The original picture of Fig. 21 is different from that Fig. 15 just for the line width. But the resulted patterns give different impressions.

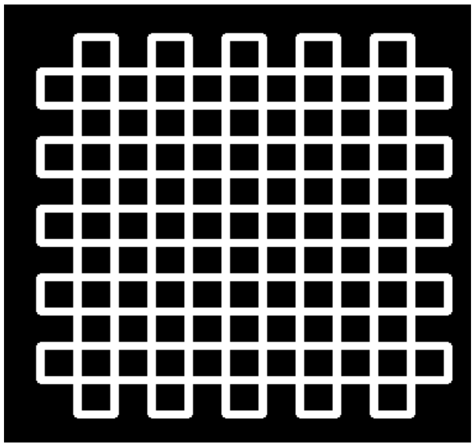

Fig. 20. Original drawing of grids: 10 overlapped rectangles with line width of 5

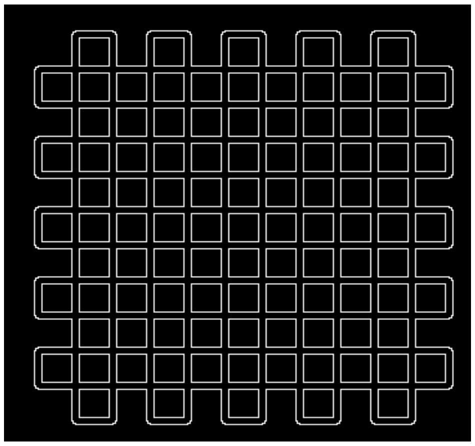

Fig. 21. Contours extracted from Fig. 20, 102 contours

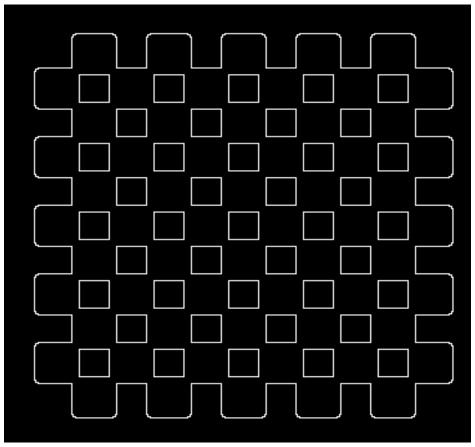

Fig. 22. X-partition, 42 contours

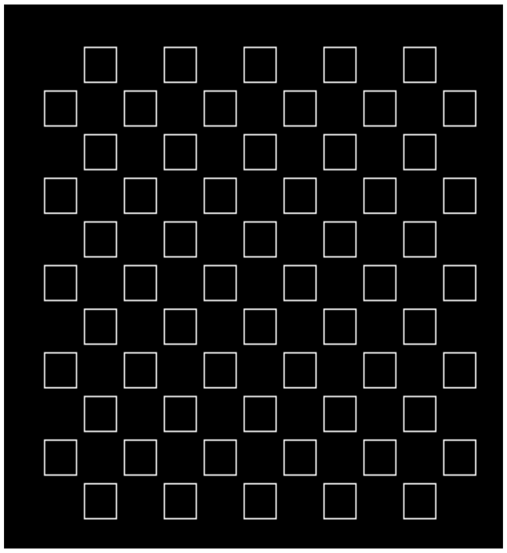

Fig. 23. Y-partition, 60 contours

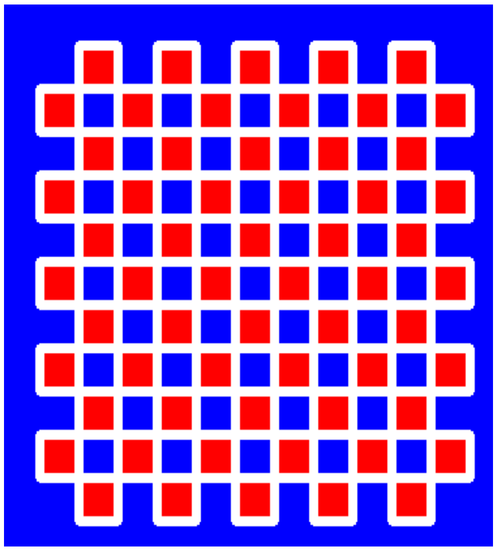

Fig. 24. Checkered patterns: X-partition with blue, Y-partition with red and boundary white

4.4 A large number of circles overlapped with line width 1

Fig. 25 is the figure of 30 circles crossed. These circles were drawn using OpenCV library with line thickness of 1. Since the drawing of circle curves is different from that of straight lines like rectangles, contours could be different from those of rectangle figure. Fig. 26 shows the contours extracted. But the lines look somewhat bolder. There are 140 contours counted by the program. The bi-partitioning algorithm was applied to Fig. 26. Fig. 27 is the resulted X-partition and Fig. 28 is Y-partition, both holding 70 contours. Figs. 26, 27 and 28 look alike. Lines of Fig. 26 look a little bolder. But those of Figs. 27 and 28 look thin. This suggests that the contour lines in Fig. 26 overlap partially, but those in Figs. 27 and 28 are not overlapped and just look single. As a matter of facts, the regions surrounded by the lines of Fig. 27 and those of Fig. 28 are completely different, exclusive each other because they are bi-partitioned regions. Fig. 29 is the output patterns (checkered-like patterns) with X-partition and boundaries painted with blue and Y-partition by red.

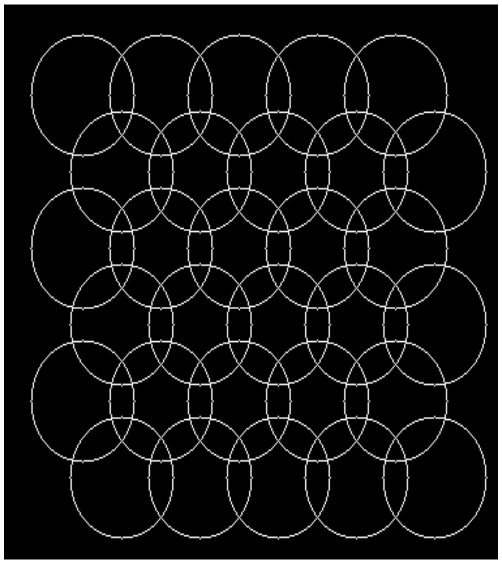

Fig. 25. Original drawing of 30 circles with line width of 1

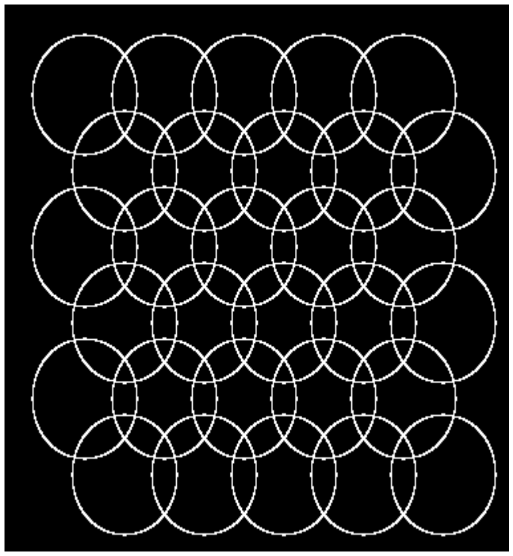

Fig. 26. Contours extracted, 140 contours

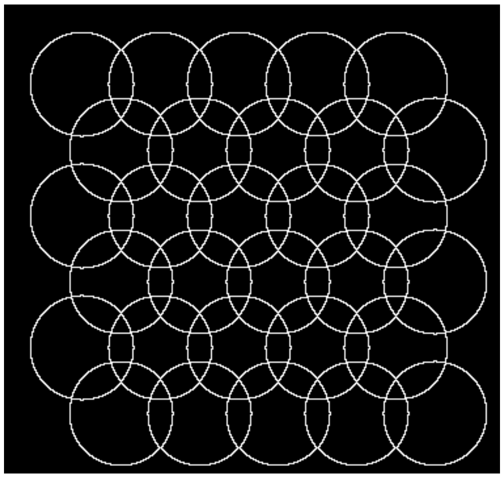

Fig. 27. X-partition, 70 contours

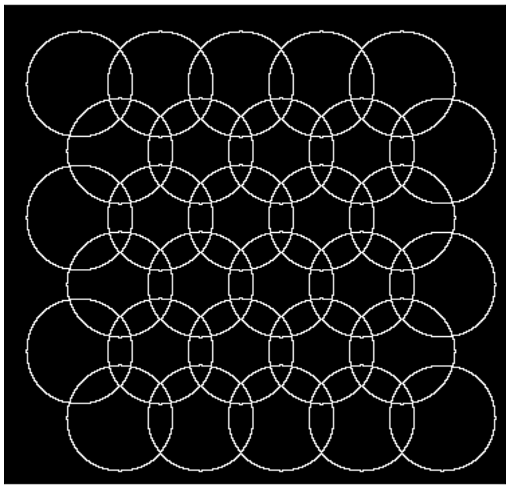

Fig. 28. Y-partition, 70 contours

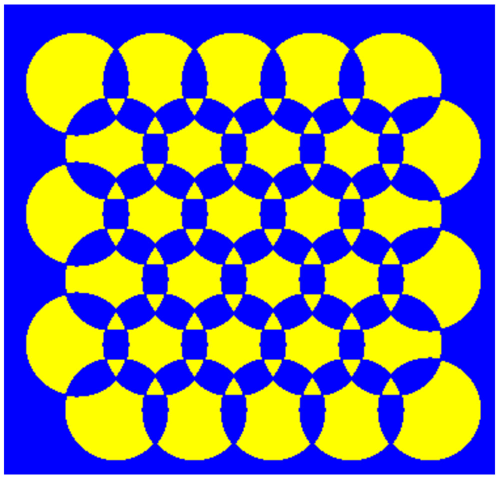

Fig. 29. Checkered-like patterns: with X-partition and boundary blue, Y-partition yellow

4.5 A large number of circles overlapped with line width 3

Fig. 30 is the figure of 30 circles crossed, but drawn using OpenCV library with line thickness of 3. Fig. 31 shows the contours extracted. It seems that all the contour lines are completely separate and look thin. There are 140 contours. The bi-partitioning algorithm was applied to Fig. 31. Fig. 32 is the resulted X-partition and Fig. 33 is Y-partition, both holding 70 contours. Figs. 32 and 33 look complexly different and all the contours are completely separate. This means that regions of the partitions are completely separate and exclusive, but they occupy all the regions. Fig. 34 is the output patterns (checkered-like patterns) with X-partition painted by blue color, but the most outer region with green, Y-partition by red, and boundaries white.

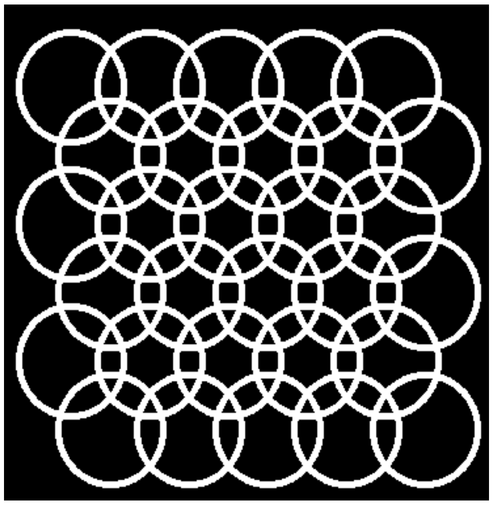

Fig. 30. Original drawing of 30 circles with line width of 3

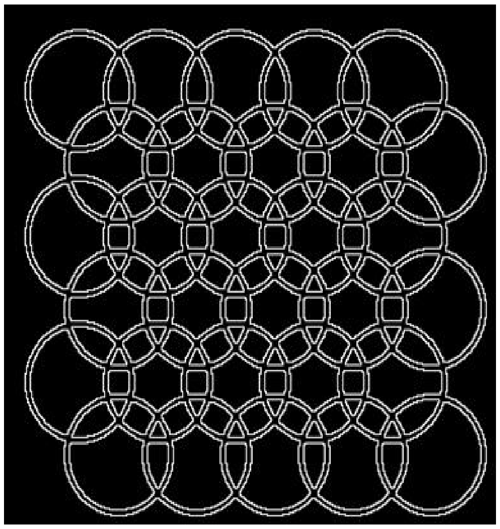

Fig. 31. Contours extracted, 140 contours

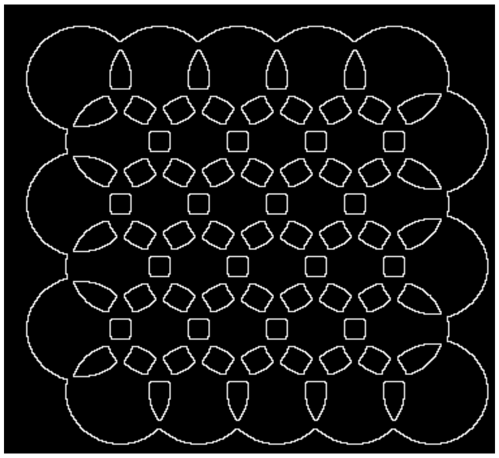

Fig. 32. X-partition, 70 contours

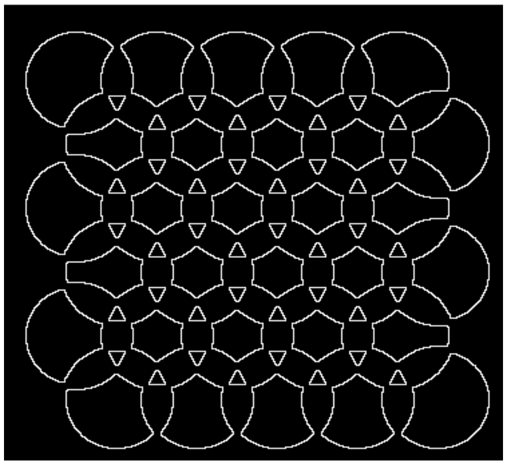

Fig. 33. Y-partition, 70 contours

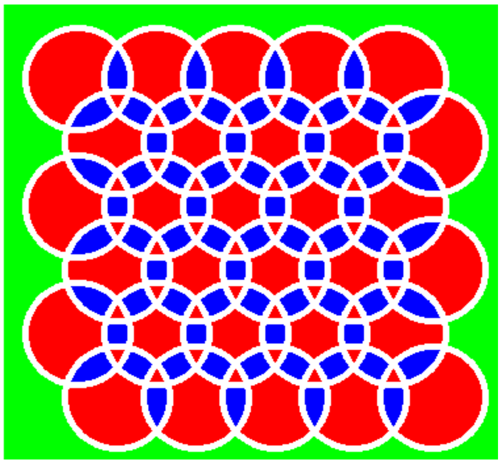

Fig. 34. Checkered-like patterns: with X-partition blue, most outer region green, Y-partition red and boundary white

4.6 Freehand drawing of 10 ovals overlapped with line width 1

Fig. 35 is the diagram of 10 freehand drawn ovals with line width of 1 overlapped. Somehow line widths vary at places -- originally drawn by MS WORD. Fig. 36 shows 102 contours extracted. Notice that some curve parts are double lines. The double lines seem 2 pixels apart. That is, the distance between the two are two pixels. Accordingly, the distance of the rests should be one pixel. As will be shown in the subsequent section, the average shortest distance (4) for Fig. 35 is 1.01. This suggests that only a very limited line parts are double. The bi-partitioning algorithm was applied to Fig. 36. Fig. 37 is the resulted X-partition with 42 contours and Fig. 38 is Y-partition with 60 contours. As seen, those contour curves are very thin, one-pixel width, thinner than those of Fig. 35. Fig. 39 is the checkered patterns obtained from Figs. 37 and 38 -- X-partition is blue, Y-partition and boundary are yellow.

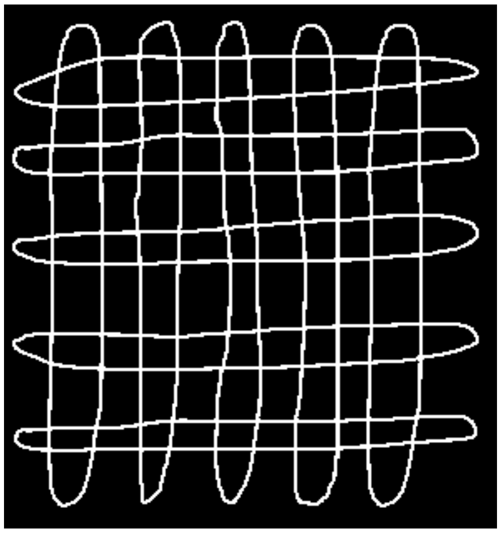
Fig. 35. Original drawing of 10 freehand ovals with line width of 1

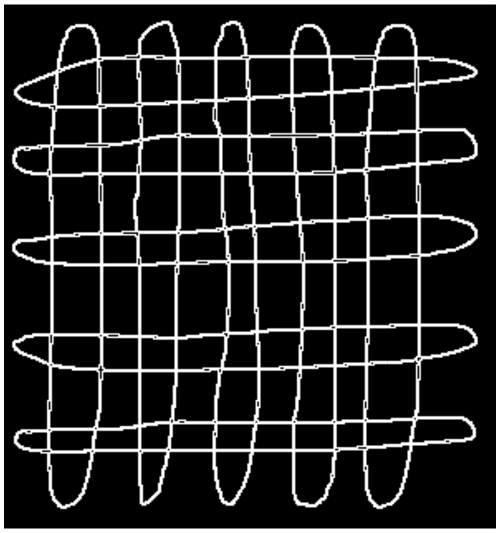
Fig. 36. 102 contours extracted

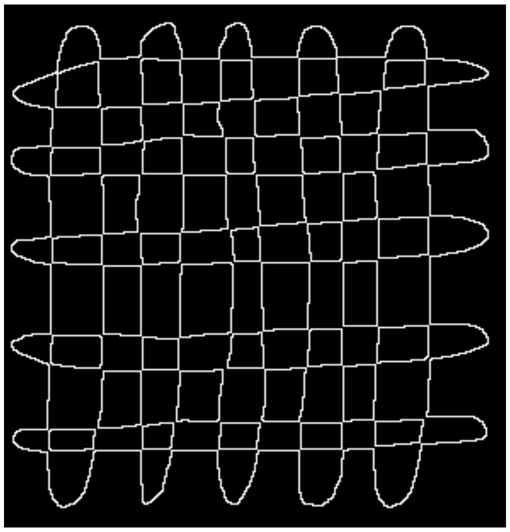
Fig. 37. X-partition, 42 contours

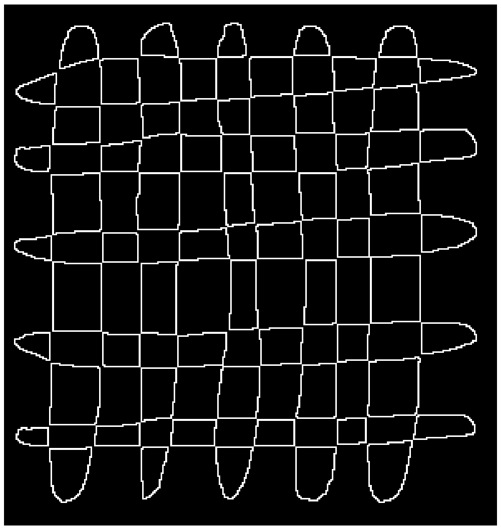
Fig. 38. Y-partition, 60 contours

Fig. 40 is the same diagram as Fig. 35, but with the outer rectangle frame line. The oval curves and the outer frame line cross for longer or wider area at the two touching places. Since these curves are not mathematically ideal curves, but realized in an actual image plane, occupying fussy crossing areas, it is expected to be more difficult to recognize the adjacency between contours. Actually, the parameter $\beta = 10$, adjacent point count, was used instead of $\beta = 4$ which were applied to most of the cases. This means that more severe criterion for adjacency was applied in this case. Fig. 41 shows 122 contours extracted. Those seem successfully extracted -- no significant difference from the Fig. 36. The bi-partitioning algorithm was applied and the results are as in Fig. 42 (X-partition) and Fig. 43 (Y-partition). All contours were successfully partitioned into two parts. Since the intersecting parts are longer between freehand ovals and rectangle frame lines, the adjacent parts between the two regions which are supposed to be in a same partition set are inclined to be longer. This length, sometimes causes the confusion. But this was solved by setting β to 10, longer adjacent point count, of (7). The final checkered patterns are shown in Fig. 44. Notice that colors of blue and yellow are interchanged between Figs. 39 and 44. It is by the interchangeability of X-partition and Y-partition.

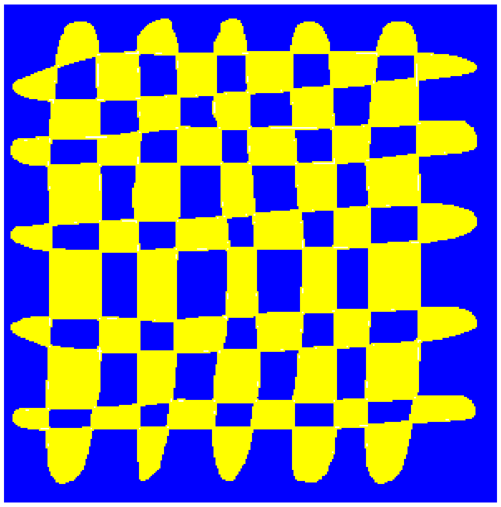
Fig. 39. Checkered-like patterns: X-partition blue, Y-partition and boundary yellow

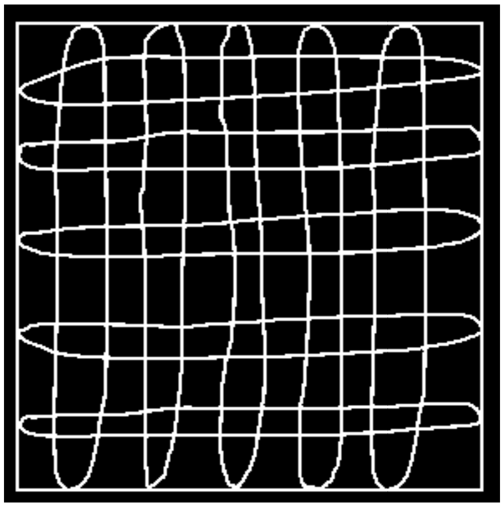
Fig. 40. Original drawing of 10 freehand ovals with line width of 1 and the outer frame

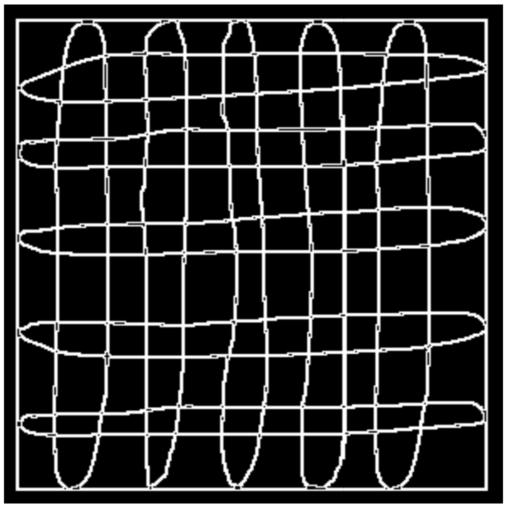
Fig. 41. 122 contours extracted

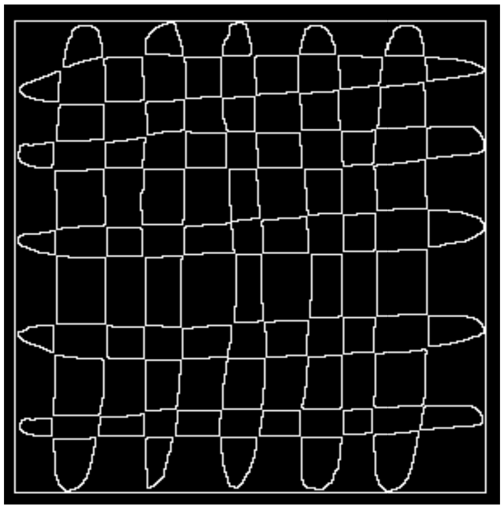
Fig. 42. X-partition, 61 contours

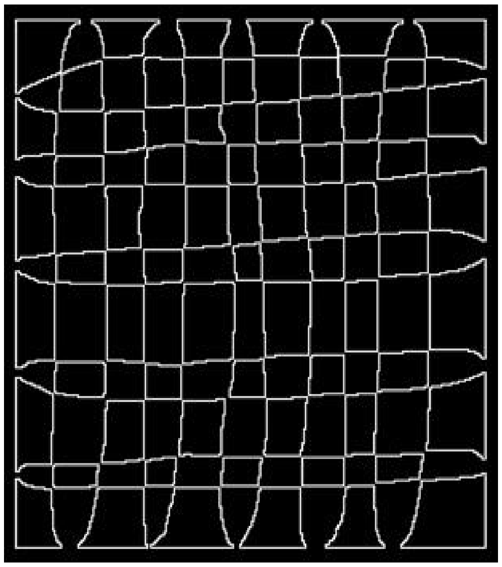

Fig. 43. Y-partition, 61 contours

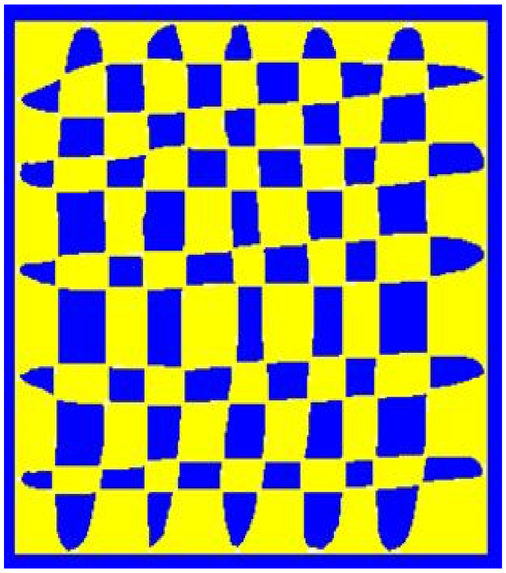

Fig. 44. Checkered-like patterns: X-partition blue, Y-partition and boundary yellow

4.7 Freehand drawing of 10 ovals crossed with curve width 5

Fig. 45 is the figure of 10 freehand drawn ovals overlapped, but with curve width of 5. Fig. 46 shows 102 extracted contours. All of them are successfully extracted. Notice that the most outer contour is very long. All the contours are completely separate. The bi-partitioning algorithm was applied to Fig. 46. Fig. 47 is the resulted X-partition with 42 contours and Fig. 48 is Y-partition with 60 contours. Despite the wide and fussy crossing areas of curves, the bi-partitioning was successfully performed for all the regions using the parameters $\alpha = 1.1$ and $\beta = 4$. Fig. 49 is the final checkered-like patterns with X-partition blue and Y-partition red and boundaries white. These give another different impression from the ordinary checkered-like patterns of Fig. 39 or Fig. 44.

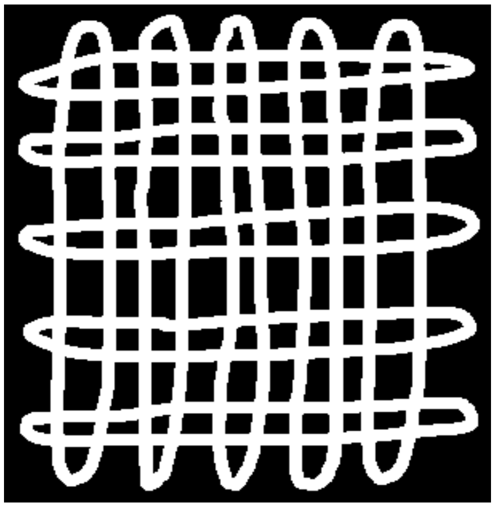

Fig. 45 Original drawing of 10 freehand ovals with line width of 5

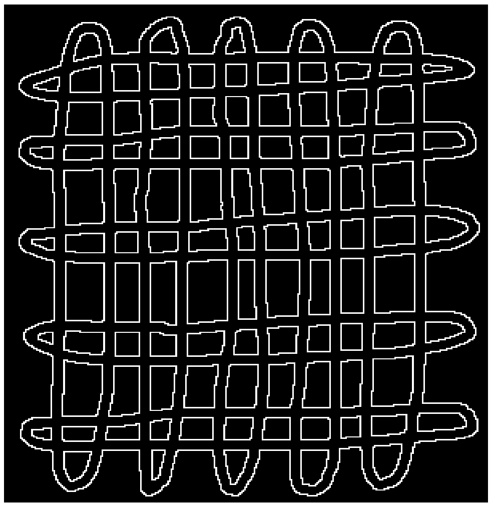

Fig. 46. 102 contours extracted

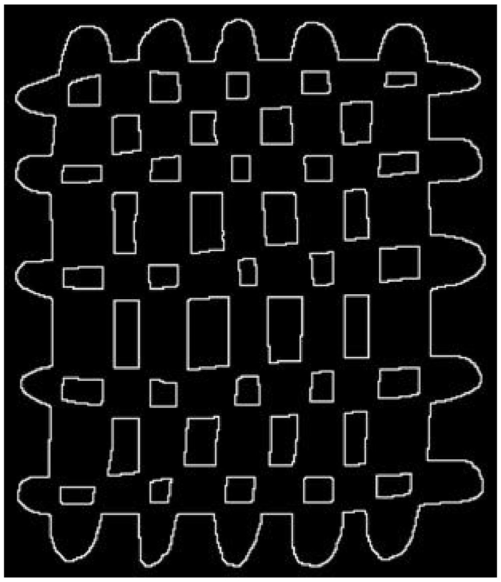

Fig. 47. X-partition, 42 contours

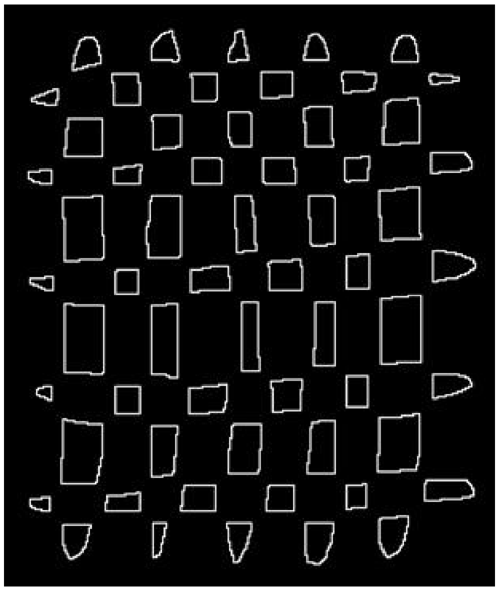

Fig. 48. Y-partition, 60 contours

Fig. 50 is supposedly the same diagram as Fig. 45, but with the outer rectangle frame line of width 5. Notice that curves and lines cross at wider and fussy areas. Fig. 51 shows successfully extracted122 contours. The bi-partitioning algorithm was applied. The results are as in Fig. 52 (X-partition with 61 contours) and Fig. 53 (Y-partition with 61 contours). The parameters $\alpha = 1.1 \; and \; \beta = 4$ were used as before. The final checkered-like patterns are shown in Fig. 54. Notice that colors of blue and red are seemingly interchanged between Figs. 49 and 54 despite regions of X-partition were painted with blue and those of Y-partition red. It is due to the interchangeable feature of X-partition and Y-partition.

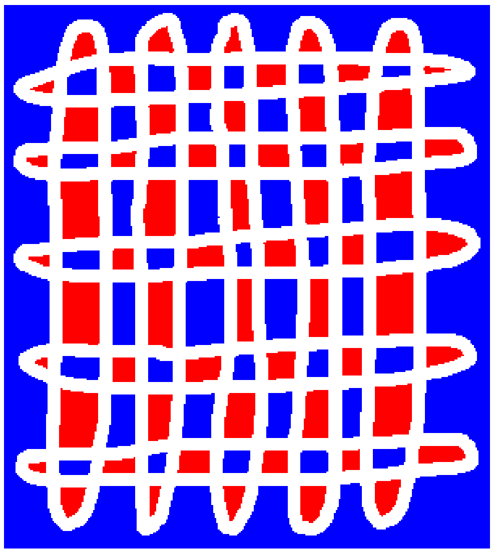

Fig. 49. Checkered-like patterns: X-partition blue, Y-partition red,

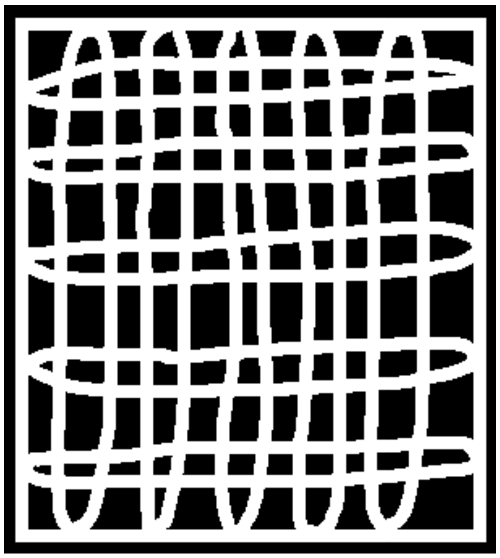

Fig. 50. Original drawing of 10 freehand ovals with line width of 5 and the outer rectangle frame

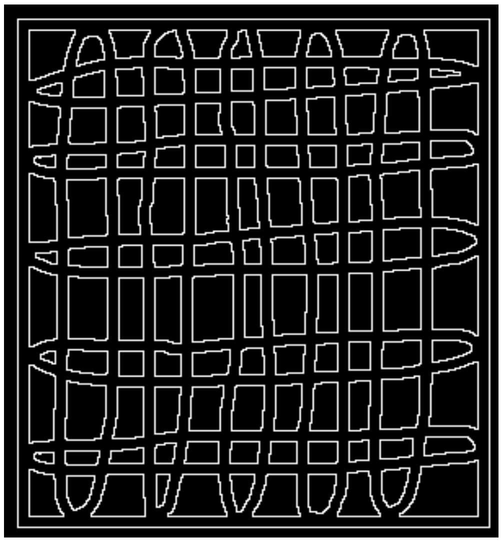
Fig. 51. 122 contours extracted

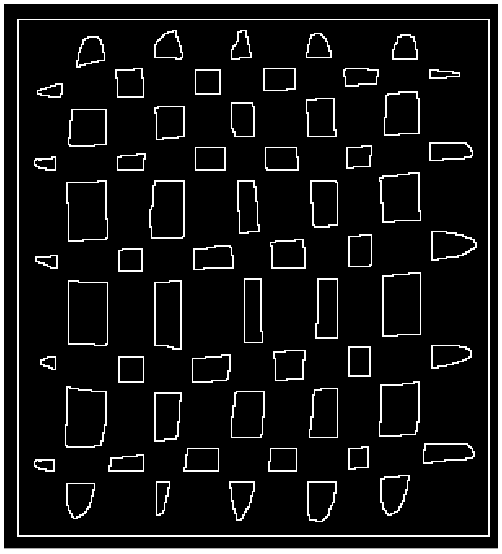
Fig. 52. X-partition, 61 contours

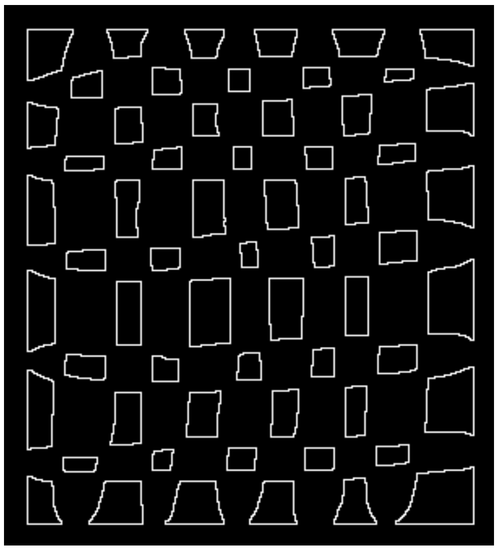
Fig. 53. Y-partition, 61 contours

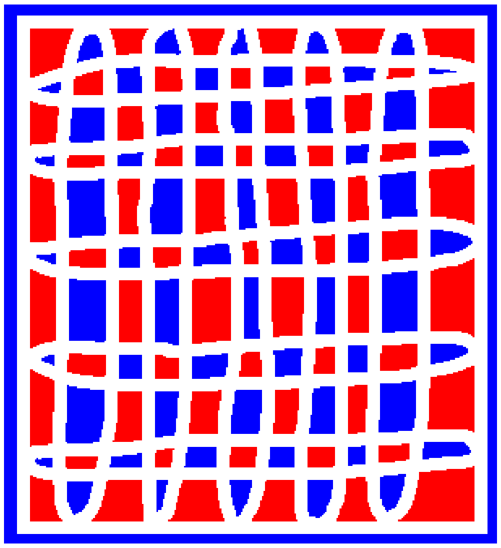
Fig. 54. Checkered-like patterns: X-partition blue, Y-partition red, boundary white

4.8 Triangles with six degree vertices with line width 8

Fig. 55 is the figure of triangles with 6 degree vertices drawn using OpenCV library with line thickness of 8. Contours were extracted as in Fig. 56. There are 35 contours, all separate. The bi-partitioning algorithm was applied to Fig. 56 and contours were partitioned into two parts. Fig. 57 is the resulted X-partition with15 contours and Fig. 58 is Y-partition with 20 contours. The vertices of 4 degrees as well as 6 were handled successfully. By the way, the parameters, $\alpha = 1.1$ *and* $\beta = 4$, were used as before. Fig. 59 is the colored checkered-like patterns with X-partition blue including the most outer region, Y-partition red and boundaries white.

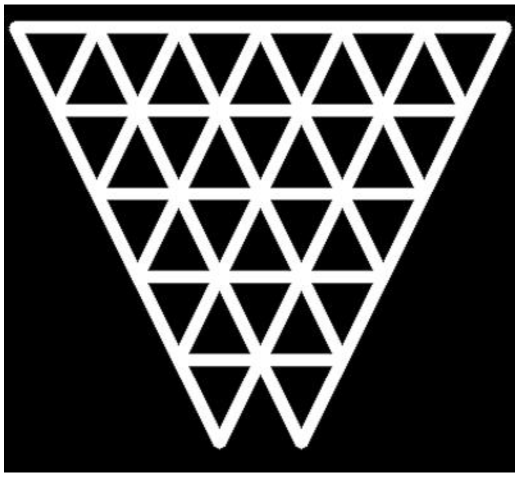

Fig. 55. Original drawing of triangles with 6 degree vertices and line width of 8

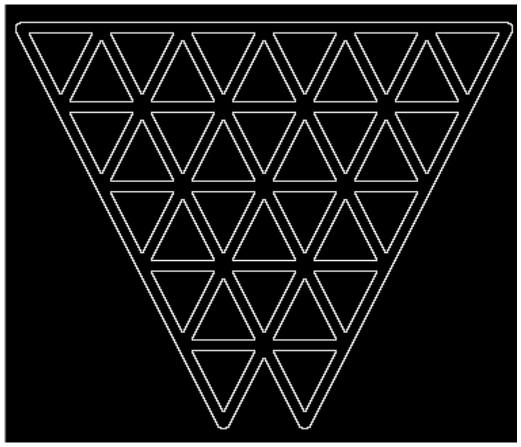

Fig. 56. 35 contours extracted

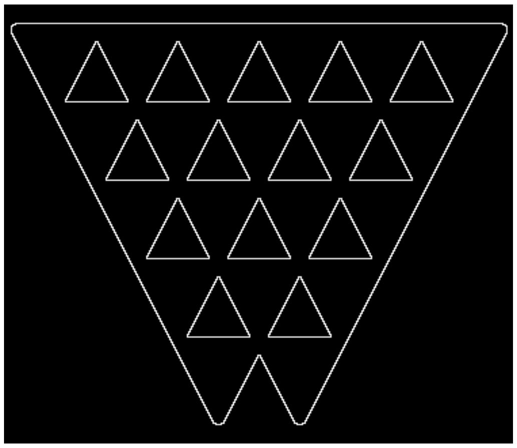

Fig. 57. X-partition, 15 contours

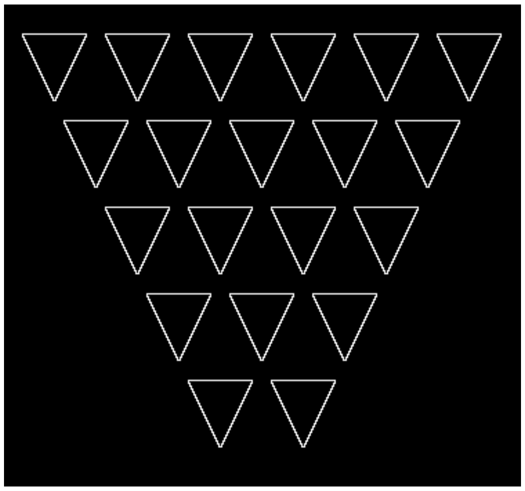

Fig. 58. Y-partition, 20 contours

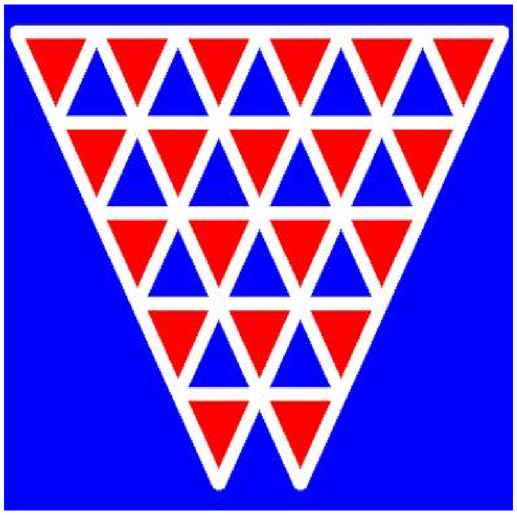

Fig. 59. Checkered-like patterns: X-partition blue, Y-partition red and boundary white

4.9 Triangles with six degree vertices and line width 1

Fig. 60 is the figure of triangles with 6 degree vertices. It was drawn using OpenCV library with line thickness of 1. Fig. 61 is the colored checkered-like patterns with X-partition blue, but the most outer region green and Y-partition and boundary yellow. This example also successfully demonstrates the bi-partitioning algorithm worked well with 6 degree vertices. The parameters, $\alpha= 1.1$ $and\ \beta = 4$, were used as before.

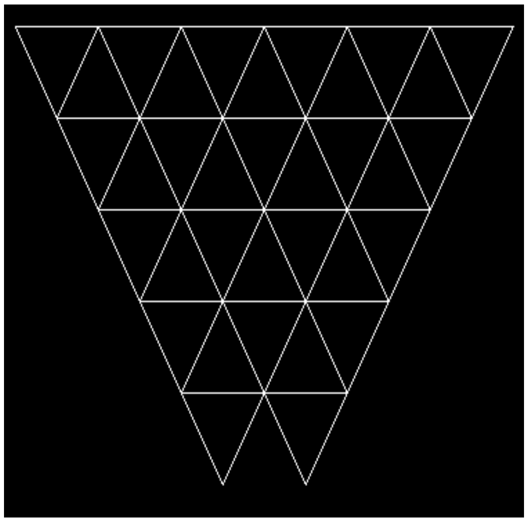

Fig. 60. Original drawing of triangles with 6 degree vertices and line width of 1

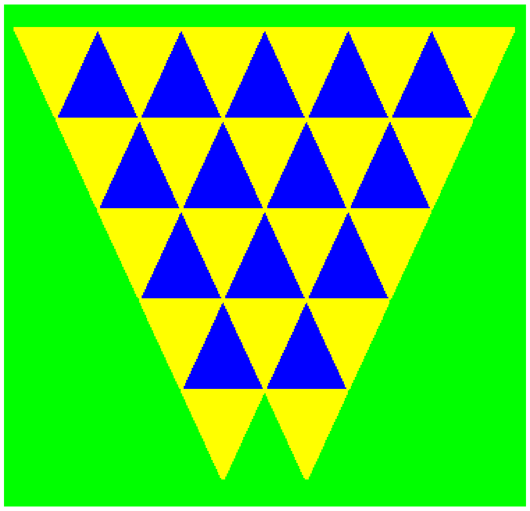

Fig. 61. Checkered-like patterns: X-partition blue, the most outer region green and Y-partition yellow

5 Line Width and Adjacency between Contours

A number of figures with different drawing line widths were tried for generating checkered patterns and their likes in the experiments. This is to see the effects of the different line widths. It turned out that different line widths cause no problem in algorithm execution and give successful outputs with different impressions -- Figs. 7 and 14, Figs. 18 and 24, Figs. 29 and 34, Figs. 39 and 49, and Figs. 44 and 54, and Figs. 59 and 61 are the examples. These different line widths were specified by the parameters, but once the figures were drawn by hand or by the program, all the checkered-like patterns were automatically generated by the program. Specifications such as line width for different original figures are given in Table 1.

The bi-partitioning was done based on the adjacency between two contours. The adjacency criterion is the distance between two contours. In the experiments, the overall average of the shortest distance from a point on a contour to other contours as (4) was first computed. The values vary depending on the drawn lines, especially on the line widths. This distance was used as the criteria for deciding if two contours are adjacent. The averages (d_{AS}) for Figs. 15, 20, 25, 30, 35, 45, 40, 50, 55, and 60, respectively, are shown in Table 1. This table also includes the shape type of figures and the line width originally given.

The adjacency is based on how many adjacent points exist between two contours. The minimal count was given by β of the expression (7) with the parameter $\alpha = 1.1$. $\beta = 4$ was used for all the experiments except for Fig. 40 ($\beta = 10$ was used). This figure of Fig.40 is with the outer rectangle frame. Many of the line crossing parts are relatively long for Fig. 40 than other figures. The adjacency checking had to be more severe for this kind of figures. That is, at those places where ovals touch the outer frame, the crossing parts are inclined to be longer and misleading. With proper parameters manually given at the program start, the program produces automatically the checkered-like patterns at once. All the experiments successfully demonstrated that the bi-partitioning algorithm is very robust.

The suitable parameters (α and β) are dependent on the figure drawn situations. If the line width changes while the figure is drawn, the overall average distance (d_{AS}) might not work properly. But, as the experiments show, $\alpha = 1.1$ seems to be proper under the condition of no line width change. But smaller α might be better, more severely distinguishing the real adjacency and the false one. The parameter β, together with α has to work to distinguish between the real adjacency and the fake one. Fake adjacency comes often from the long intersecting or touching part between two contours which are not adjacent. This fake adjacency was caused, when $\beta = 4$, in the experiment of Fig.40, the free hand drawn ovals touch the straight frame line, which could be considered as an unusual type of figure. For most cases, β can be arbitrary within a roughly estimated reasonably range. But β cannot be too large for the case where contours are not long enough. If proper parameters are computed automatically, it's better.

The contour cycles of Fig. 2 were all on the lines of rectangles with line thickness of 1 (as of OpenCV). This means that the distance between any two adjacent regions was zero. The computed average distance was also zero. Rectangles of Fig. 15, Circles of Fig. 25 and triangles of Fig. 60, all with the line width 1, give the distance of 0.0, 0.28 and 0.25, respectively. It seems that horizontal or vertical lines give the shorter distance. Rectangles of Figs. 20 and triangles of Fig. 55 are with line width of 5 and 8, respectively, and the computed average distance was 6.0 and 7.8, respectively. The distance does not seem to be proportional to the width. Freehand ovals of Figs. 35 and 40 were drawn with line width 1. But the d_{AS} is slightly smaller for Fig. 35 than Fig. 40. It should be due to the crossing parts between ovals and outer frame.

Table 1. Figure specifications and average of shortest distance (d_{AS})
Unit shape symbol explanation: r means rectangle, c circle, f freehand drawing, t triangle.
Line thickness of Figs. 35 and 44 are by MSWORD (point), others are by OpenCV library

Fig. no	**15**	**20**	**25**	**30**	**35**	**45**	**40**	**50**	**55**	**60**
Unit shape	r	r	c	c	f	f	f	f	t	t
Line thickness	1	5	1	3	1pt	5pt	1pt	5pt	8	1
d_{AS}	0.0	6.0	0.28	3.4	1.01	7.0	1.03	7.03	7.8	0.25

6 Conclusion

The method of this study demonstrated its ability to generate varieties of checkered patterns and their variations with various features including:

1) Wide range of patterns can be produced.
2) It can be applied to drawings of various closed line shapes as -- circles, rectangle, triangles,

freehand drawing, etc.
3) Width of the line or curve can vary.
4) It is automatic and very robust as demonstrated.

Competing Interests

Author has declared that no competing interests exist.

References

[1] Available:https://www.google.co.jp/search?q=tokyo+olympic+games+2020+emblem&espv=2&biw=1920&bih=1060&tbm=isch&tbo=u&source=univ&sa=X&ved=0ahUKEwiG-72Jjb7PAhVFnJQKHZnJCJAQsAQIKA, (accessed 2016/10/3)

[2] Available:https://www.google.co.jp/search?q=checkered+pattern&espv=2&biw=1920&bih=1060&tbm=isch&tbo=u&source=univ&sa=X&ved=0ahUKEwj7vPCQ9b3PAhVCVZQKHT8KAUYQsAQIHg&dpr=1, accessed 2016/10/3

[3] Kurokawa T. Maze construction by using characteristics of Eulerian graphs. British Journal of Mathematics & Computer Science. 2015;9(6):453-472. SCIENCE DOMAIN international, UK, ISSN: 2231-0851.
Available: http://sciencedomain.org/issue/1147

[4] Open CV Library.
Available: https://msdn.microsoft.com/ja-jp/library/dn188695.aspx (Accessed 2016/12/30)

[5] Suzuki S, Abe K. Topological structural analysis of digitized binary images by border following. CVGIP. 1985;32-46.

[6] Rosenfeld A, Kak AC. Digital picture processing, 2nd ed. Academic Press, New York. 1982;2.

[7] Yokoi S, Toriwaki J, Fukumura T. An analysis of topological properties of digitized binary pictures using local features. Comput. Graphics Image Process. 1975;4:63-73.

[8] Agoston MK. Computer graphics and geometric modeling, Springer; 2005.

[9] Kurokawa T, Kazumasa Mori (Shizuoka Information Processing Center Co., Ltd.), and Teruaki Mizuno (Daiichi System Engineering Co., Ltd.). Automatic construction of picture maze by repeated contour connection. Thailand-Japan Joint Conference on Computational Geometry and Graphics 2012;5-6. Bangkok, Thailand.

[10] Kurokawa T. Picture maze generation by repeated contour connection and graph structure of maze. Computer Science and Engineering. 2013;3(3):76-83. Scientific & Academic Publishing, USA, ISSN:2163-1484.
Available: http://article.sapub.org/10.5923.j.computer.20130303.04.html

[11] Kurokawa T. Construction of picture maze along set of image dot vertices. British Journal of Mathematics & Computer Science. 2016;1(1):1-14. SCIENCEDOMAIN international, UK, ISSN: 2231-0851.
Available: http://sciencedomain.org/issue/1786

[12] Meng KK, Fengming D, Guan TE. Introduction to graph theory. World Scientific; 2007.

[13] Matousek J, Nesetril J. Invitation to discrete mathematics. Second Edition, Oxford University Press, New York; 2009.

Peer-review history:
The peer review history for this paper can be accessed here (Please copy paste the total link in your browser address bar)
http://sciencedomain.org/review-history/17639