アルゴリズム 58－9
（1997. 9. 19）

Contour Representation of an Image with Applications

浅野哲夫[1]、木村宗市[2]

[1]北陸先端科学技術大学院大学（石川県辰口町）
[2]大日本スクリーン製造（株）（京都市）

画像は各画素の明るさの情報を表す行列の形で表現されるのが一般的であるが、ここでは、画素の明るさをその位置における標高と見なして、画像を等高線で表現する表現方法について扱う。最初に、この表現の効率の良い求め方とデータの蓄え方を提案し、次に、この表現を用いた画像処理法を幾つか紹介する。

Contour Representation of an Image with Applications

Tetsuo Asano[1]、Souichi Kimura[2]

[1]School of Computer Science, JAIST (Tatsunokuchi, Ishikawa, Japan)
[2] Dainippon Screen MFG. (Kyoto)

A common representation of an image is due to a matrix with intensity levels as its elements. This paper deals with another representation using contour lines by regarding intensity levels of pixels as height at corresponding locations. We first propose an efficient algorithm for finding such a representation and a data structure for the representation, and then survey several applications of the contour representation.

1 Introduction

There have been a nubmer of attempts to apply algorithmic techniques in computational geometry to image processing or computer vision. Their purposes are often to have asymptotically faster algorithms. One of typical such examples is the linear-time algorithm for computing distance transformation by Breu, Gil, Kirkpatrick and Werman [4] and Hirata and Katoh [5] which is to compute the Euclidean distance from each black pixel to its nearest white pixel based on the geometric notions of Voronoi diagram [4, 6] and lower envelope of parabolas [5], respectively.

An alternative way of applications of computational geometry is found, for example, in the study on detecting fundamental curve components in a binary edge image by the author [2]. Although a great number of methods have been proposed to detect planar curves in a binary image, most of them are based on heuristic methods and a digital curve component is often undefined or ambiguous. In this sense the author's is the first theoretical algorithm with theoretically guaranteed performance.

It is common to represent a quantized image as a set of pixels with information of brightness levels in three colors (Red, Blue, and Green) in a matrix form. An alternative way of representation is the one by contour lines. For each gray level i we compute connected regions of pixels with gray levels greater than or equal to i. It is easy to reconstruct an original image from those boundaries. Representation of an image as a collection of those boundary lines associated with gray levels (contour lines) is the contour representation of an image. This representation is not new but it has been used only for data compression of an image [8].

If we keeping the whole collection of those contour lines, we would need at least several

times more space for image matrix. Thus, we need a compact way for the representation. Since it is rather easy to follow any contour line once a starting edge is specified, we keep a set of those starting edges (called "seed edges") such that for each contour line at least one edge is included. We give an efficient algorithm for finding a reasonable size of a seed set.

We describe three different applications of the representation: restoration of an image with flaw based on dynamic programming, improvement of image quality (resolution), and enlargement of an image with smooth edges.

2 Preliminaries

Let $G = (g_{ij})$, $i, = 1, 2, \ldots, v-2$, $j = 1, 2, \ldots, h-2$ be an image such that each g_{ij} is an integer between 0 and $L-1$. For convenience we embed the image into a two-dimensional $v \times h$ array so that the boundary elements are all -1. For further convention, we represent the two-dimensional array by a one-dimensional array in a raster-scan fashion. Formally, an ij-element of the array is mapped to the $k = i * h + j$-th element of the one-dimensional array. Therefore, the four neighboring pixels of an internal pixel numbered k are those numbered $k-h$ (above), $k+h$ (below), $k-1$ (left) and $k+1$ (right), respectively. Throughout the paper we denote the pixel numbered k by p_k.

(Lattice) edges are also sequentially numbered. The horizontal edge below a pixel p_k and vertical edge to the right of p_k are numbered $2k$ and $2k+1$, respectively. Now a set of edges is: $\{e_0, e_1, \ldots, e_E\}$, where $E = 2hv - 1$. Each edge is directed so that the interior of a region lies to its right.

A set of pixels with gray levels $\geq i$, $0 \leq i \leq L-1$ may be divided into several connected regions, where connectivity is defined by horizontal or vertical adjacency as usual. Such a set is denoted by R_i. By the definition, $R_{i+1} \subseteq R_i$ holds for every i, $0 \leq i < L-1$. Each connected region may have more than one boundary due to holes. Throughout the paper each boundary is represented as a sequence of edges directed so that the interior lies to their right. Thus, an external boundary is directed clockwisely while an interior one counterclockwisely.

3 Computing a Seed Set

The contour representation of an image is to represent an image as a collection of the boundaries (contours) for the regions $R_0, R_1, \ldots, R_{L-1}$ for all intensity levels. Note that there are as many contours as the number of connected components of each R_i with levels greater than or equal to i. The authors proposed an algorithm [1] for computing all those contours in an output-sensitive manner, that is, in time $O(n+K)$ where n is the number of pixels and K is the total length of the contours.

One serious disadvantage of the representation is its high space complexity. The length of a contour could be as large as $O(n)$ in the worst case. In fact, if an image is like a checkerboard in which white and black pixels alternate, the total length of contours is $O(nL)$. The average length of a contour is expected to be $O(\sqrt{n})$. Therefore, keeping all the contours may be too expensive in practice.

Our experiences from experiments on real image data also suggest not to store all of them. Our approach here is to keep some additional information in addition to an image matrix so that given an intensity level as a query we can trace its corresponding set of contours in an efficient manner.

Without preprocessing, computing all the contours of a region R_i requires $\Omega(n)$ time. Our approach here is to devise how to output all the necessary contours in an output-sensitive manner, that is, in time $O(K_q + \log n)$ where K_i is the length of the output for a query level q with a small space of additional information as possible.

A compact set of edges such that every contour contains at least one of them is enough to find contours of a given level. The problem is how to find a minimum set of those starting

edges called *seed edges*. We can borrow the algorithm in [7] which finds a minimum seed set for a triangulated information network.

Given an image matrix, we first construct a contour tree. For the time being we assume that no two pixels have the same intensity level. This constraint is resolved later. Nodes of the tree are contours and two nodes are connected by edges if there is a region bounded by their corresponding contours. This tree can reflect the global structure of a given image.

The contour tree can be constructed as follows [7]. First of all we sort all the pixels in the decreasing order of their intensity levels. Then, we maintain two sets, HIGH and LOW, where HIGH (LOW, resp.) keeps all the pixels with levels higher (lower, resp.) than or equal to the current level as connected components. Important difference is that 4-connectivity (connectivity only by horizontal and vertical adjacency) is used in HIGH wheras 8-connectivity (allowing 45-degree and 135-degree adjacency) in LOW. We keep the two data structures HIGH and LOW while decreasing the current level from the maximum level to the minimum.

At the beginning, HIGH is empty and LOW contains all the pixels as one component. We assume dummy pixels of level -1 around a given image and all those dummy pixels are put into LOW. At a level k, if there is a pixel p of the level, we insert it into HIGH and delete it from LOW.

When a pixel is inserted into HIGH, two different events occur. If it is not connected to any pixel which has been added to HIGH so far, it creates a new component and we create a new corresponding node in the contour tree. Such a new component is caused by a local maximum. On the other hand, if it had some horizontal or vertical neighbors in HIGH which belong to different components, then those two components are merged into one. In this case we create a new corresponding node and connect it with the two nodes corresponding to those components to be merged.

In LOW, deletion of a pixel may split one component into two or three components. In this case a new node is created and connected with those corresponding nodes.

When there is a tie in intensity levels, node creation is postponed until all the pixels of one level is processed. Then, if two such pixels belong to the same compoent in HIGH, they are merged into one node. See Figure 1 which described how a contour tree is constructed.

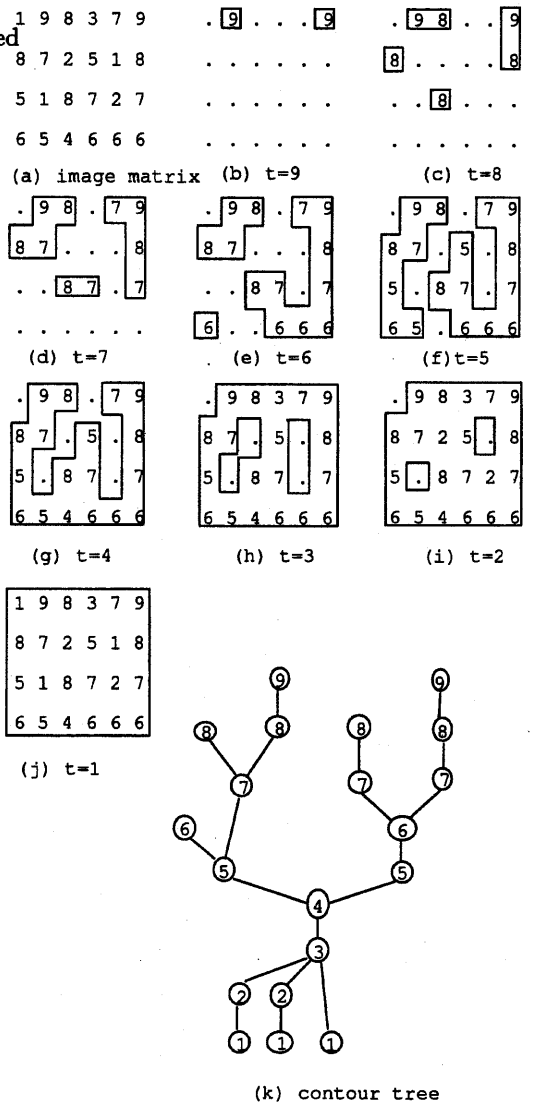

Figure 1: A image matrix and contour tree.

For the example shown in Fiure 1, HIGH and LOW change as follows:

(1) Level 9
HIGH= {{9},{9}},

LOW = one component.

(2) Level 8
HIGH= {{98}, {98}, {8}, {8}},
LOW = one component.

(3) Level 7
HIGH= {{9887}, {7987}, {87}},
LOW = one component.

(4) Level 6
HIGH= {{9887}, {798766678}, {6}},
LOW = one component.

(5) Level 5
HIGH= {{9887565}, {7987666785}},
LOW = one component.

(6) Level 4
HIGH= one component,
LOW = one component.

(7) Level 3
HIGH= one component,
LOW = {{1}, {12}, {12}}.

(8) Level 2
HIGH= one component,
LOW = {{1}, {1}, {1}}.

(9) Level 1
HIGH= the whole image,
LOW = one component consisting of dummy pixels.

According to the algorithm in [7], a contour tree can be constructed in $O(n \log n)$ time. A difficulty is how to implement split operations. A trick for efficient implementation is to trace the associated contours to find shorter ones based on a so-called tandem search. The detail should be found in [7].

Then, an edge corresponds to a path connecting two nodes in the contour tree. A minimum seed set is a minimum set of those paths that cover all the tree edges. The problem of finding a minimum seed set is formulated as a minimum cover in a bipartite graph which finds to be strongly chordal [7]. Therefore, it is solved in polynomial time.

4 Applications

We describe three different applications of the representation. One is the problem of restoring an image with flaw. Suppose that we have a commercial poster of a model and want to erase crow's feet of the model. If we specify the crow's feet as flaw regions to be erased, then contour lines intersecting them become disconnected. Thus, the problem is how to connect those disconnected contours naturally (see Fig. 2). It looks like a problem of interconnecting terminal pairs on a printed circuit board. One major difference is that in our case there are two types of terminals "source" and "target" and any source can be interconnected to any target as far as one source is connected to exactly one target and vice versa. Dynamic programming is a natural selection for this problem. In fact, we coded a dynamic-programming-based algorithm and made experiments on several different images, which will be described in another opportunity.

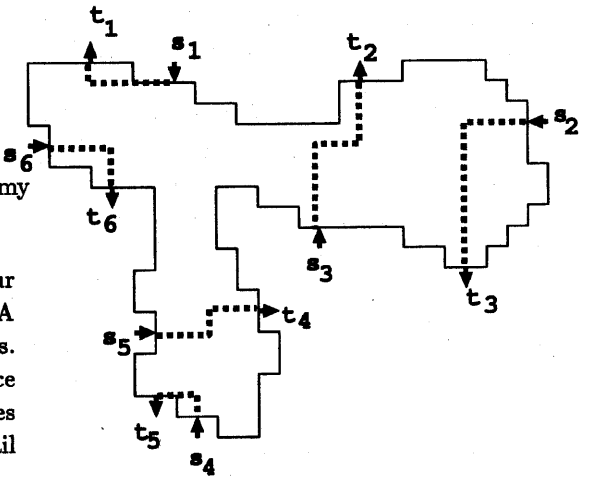

Figure 2: Interconnecting contour lines disconnected by a flaw region.

Another problem is to increase resolution. To describe the problem, suppose we have an image scanner with 6-bits resolution for each of red, blue and green. The goal is to increase the resolution, say from 6-bits to 8-bits. One physical way is to take four pictures of the same object and to define the sum of the gray levels at each pixel to be the output gray level at the corresponding pixel. Of course, this approach has several difficulties and is not what

we want. A number of algorithms for this purpose are known in computer vision. A serious disadvantage of those algorithms is that the resulting image tends to be blurred. The goal here, therefore, is to devise a method by which we can increase the number of brightness levles into arbitrarily many without introducing any blur (see Fig. 3). Our task is to partition a region bounded by two contour lines of consecutive levels by a closed curve which is equidistant from these two contour lines. If we repeat such interpolation twice, then the region is partitioned into several regions of four different intensity levels. This is a geometric problem which is similar to the one arising in geographic information system.

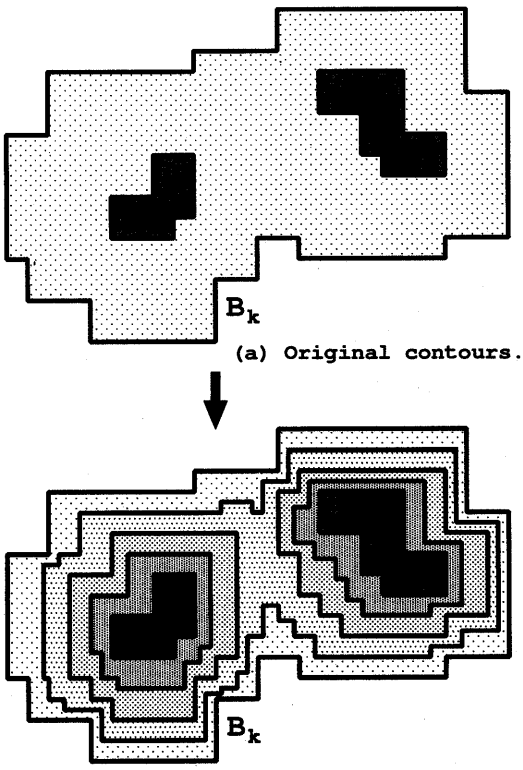

Figure 3: Contour lines of two consecutive levels and their interpolation.

The contour representation has yet another useful application. When an image is enlarged by two times in each side, one pixel in the original image corresponds to four pixels in the resulting image. As is easily imagined, if we replace each pixel with four identical pixels, the resulting image looks very ugly because of large squares. One way to have better-looking image without introducing blur is the following: First, we approximate each contour line by smooth curve. Then, each contour line is expanded in each direction to define an output image. This method does not blur the image and the resulting contours are smooth.

5 Conclusion

The contour representation of an image presented in this paper is totally different from the conventional representation as an image matrix. The most noteworthy advantage of this representation is that it can reflect global properties of an image. Notice that such an advantage is difficult to attain if we insist on the conventional matrix representation of an image. Although we have presented only three applications, we believe that it could be a rich source of many other applications in image processing.

6 Acknowledgements

This is a joint work between Asano Laboratory and Dainippon Screen MFG Inc. in Kyoto. The authors would like to express their thanks to S. Shimazu and K. Nakai of Dainippon Screen MFG and T. Hirama of Osaka Electro-Communication University, for their efforts on the experiments and also their several stimulating suggestions and discussions.

References

[1] T. Asano and S. Kimura: "Contour Representaiton of an Image with Applications," Proc. SPIE's International Symposium on Vision Geometry IV, pp.14-22, San Diego, July 1995.

[2] T. Asano, N. Katoh, and T. Tokuyama: "A unified scheme for detecting *Proc. 2nd European Symp. on Algorithms,* pp. - , Utrecht, 1994.

[3] C. Burnikel, K. Mehlhorn, S. Schirra: "How to compute the Voronoi diagram of line segments," *Proc. 2nd European Symp. on Algorithms,* pp. - , Utrecht, 1994.

[4] H. Breu, J. Gil, D. Kirkpatrick and M. Werman: "Linear time Euclidean Distance transform algorithms," to appear in *IEEE Trans. on Pattern Analysis and Machine Intell.*

[5] T. Hirata and T. Katoh: "An algorithm for Euclidean distance transformation," *SIGAL Technical Report of IPS of Japan,* 94-AL-41-4, pp.25 - 31, Sept. 1994.

[6] F. P. Preparata and M. I. Shamos: "Comoutational Geometry: An Introduction," *Springer-Verlag,* New York, Ny, 1985.

[7] M. van Kreveld, R. van Oostrum, C. Bajaj, V. Pascucci, and D. Shikore: "Contour Trees and Small Seed Sets for Isosurface Traversal," Proc. ACM Symp. on Computational Geometry, pp, 212-219, Nice, 1997.

[8] L.C. Wilkins and P.A. Wintz: "A contour tracing algorithm for data compression of two dimensional data," Purdue University, School of Engineering, Report TR-EE-69-3, January 1969.