ORDER-INVARIANT TOBOGGAN ALGORITHM FOR IMAGE SEGMENTATION

Yung-Chieh Lin(林永傑)[†‡], *Yi-Ping Hung*(洪一平)[†‡],
Chiou-Shann Fuh(傅楸善)[‡]

[†] Institute of Information Science, Academia Sinica, Taipei, Taiwan

[‡] Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan

Email: hung@iis.sinica.edu.tw

ABSTRACT

The toboggan and watershed algorithms are two classic algorithms for image segmentation. Both of them are based on the concept that images can be segmented into homogeneous regions by partitioning the gradient images along the ridges. The watershed algorithm has been referred more frequently in the literature than the toboggan algorithm, partially because the watershed algorithm is conceptually easier in classifying the pixels in the flat regions or on the ridges. In this paper, we present an order-invariant toboggan algorithm, which can classify the pixels in the flat regions and on the ridges. Also, a modified morphological grayscale reconstruction is introduced to preprocess the gradient image so that the over-segmentation problem can be alleviated. This paper demonstrates that the toboggan algorithm is faster than the watershed algorithm both in theoretical complexity and in runtime practice.

I. INTRODUCTION

Image segmentation is a fundamental problem to many computer vision applications. The toboggan [2,5,6] and watershed [1,3,8] algorithms are two classic algorithms for image segmentation. Since both methods segment images into homogeneous regions by dividing the gradient images along the ridges, they should generate roughly the same segmentation results when implemented with care. However, the watershed algorithm has been referred to more frequently in the literature than the toboggan algorithm has, partially because the watershed algorithm is conceptually easier in classifying the pixels in the flat regions or on the ridges. In this paper, we present an order-invariant toboggan algorithm, which not only can classify the pixels in the flat regions and on the ridges, but also has the order-invariance property, i.e., the segmentation result is independent of the visiting order of the neighbors.

Figure 1 shows an illustrative example of segmenting a one-dimensional signal. Given a signal function $f(x)$ as shown in Figure 1(a), we first compute the absolute value of its first derivative, i.e. $G(x) = |f'(x)|$, which will be referred to as the gradient magnitude for image data. In Figure 1(c), we can find two local maxima at the positions of the inflective points of the original signal. Hence, the original signal is partitioned into three segments as shown in Figure 1(d).

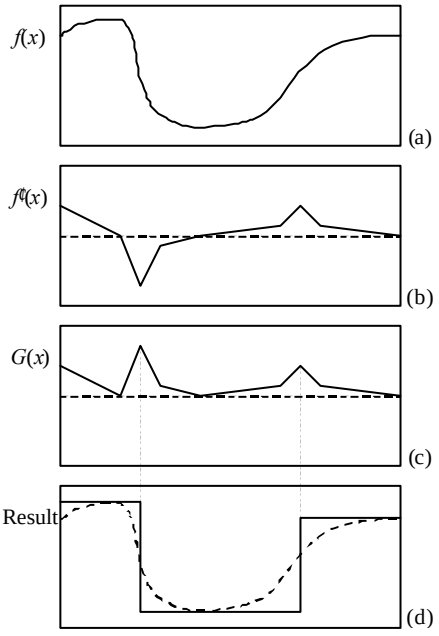


Figure 1. An illustrative example of segmenting a one-dimensional signal based on its gradient magnitude.

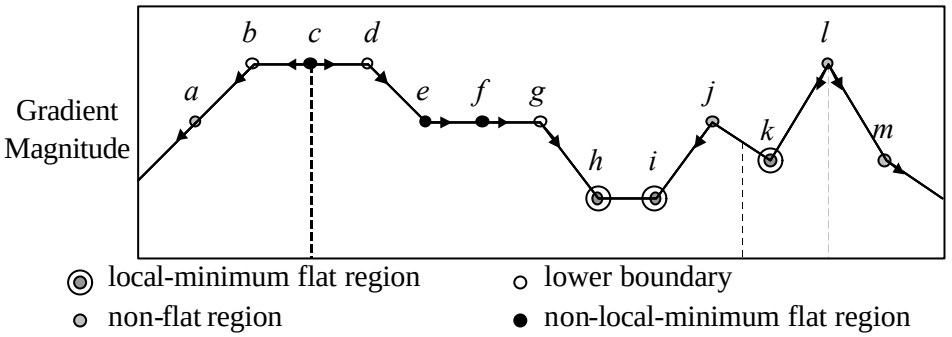


Figure 2. An example for illustrating flat and non-flat regions. In this example, pixels *a*, *j*, *l*, and *m* are on the non-flat regions, pixels *b*, *c*, *d*, *e*, *f*, and *g* are on the non-local-minimum flat regions, and pixels *h*, *i*, and *k* are on the local-minimum flat regions.

Both the toboggan and watershed algorithms are designed to associate each pixel with the minimum of the valley where the pixel is located. The difference between the two algorithms is that the toboggan algorithm uses a top-down approach and the watershed algorithm uses a bottom-up approach. The watershed algorithm initially creates a unique marker at the local minimum of each valley, and then simulates water flooding from the markers level by level. The toboggan algorithm initially makes all pixels slide downward to the local minima, and then gives each connected component a unique label. In some papers, the sliding of pixels is referred to as rain falling [2,4].

For most images, the basic toboggan and watershed algorithms tend to generate over-segmented results, so in many applications, they are usually combined with other procedures. For example, [8] proposed a video object segmentation method that uses the watershed algorithm for initial segmentation. However, before applying the watershed algorithm, they first process the gradient images by erosion in order to eliminate the regions of shallow valleys. In [5], the segmentation results obtained by the toboggan algorithm are used as initial contours for the proposed intelligent scissors. In this paper, we alleviate over-segmentation by adopting a method similar to that used in [7], which will be described in section V.

II. ORDER INVARIANCE

It is not hard to implement either the toboggan algorithm or the watershed algorithm to classify the pixels in non-flat regions, but it may be more complicated in determining which direction a pixel in a flat region should flood (upward) or slide (downward). There are two kinds of flat regions in the gradient image: the local-minimum flat region and the non-local-minimum flat region. A flat region is called local-minimum if the gradient magnitudes of the surrounding pixels are all greater than the gradient magnitude of the flat region. A local-minimum flat region can be uniquely labeled by finding the connected pixels having the same gradient magnitude (e.g., pixels *h* and *i* in Figure 2.). On the other hand, a non-local-minimum flat region can be a thick edge separating several valleys, and should be evenly divided between the valleys (e.g., pixels *b, c,* and *d* in Figure 2.). The toboggan algorithm in [4] solves this problem by associating the pixels in a non-local minimum flat region to the closest lower boundary. Here, the lower boundary means the region boundary whose neighbors have smaller gradient magnitudes than the gradient magnitude of the flat region (e.g., pixels *d* and *g* in Figure 2 are lower boundary points while pixel *e* is not). The watershed algorithm in [6] solves this problem by growing the labels from the lower boundary. It is worth mentioning that the method in [2] avoids the classification problem of the flat regions by working with the Gaussian smoothed floating point images. However, there exist some special cases where this method is not applicable.

If there still exists ambiguity in determining the flooding/sliding direction of a pixel, the toboggan and watershed algorithms can either classify the pixel to any nearby segment or label the pixel a ridge pixel. In this paper, we define the order-invariant algorithms as the algorithms whose segmentation results are independent of the visiting order of the neighbors. Therefore, it is necessary for an order-invariant algorithm to label the

ambiguous pixels as ridge pixels and not to classify such pixels to any regions. The watershed algorithm described in [6] can classify pixels into ridges (which are referred to as watersheds), but it is not order-invariant because a pixel first labeled as a ridge point may be re-classified into another region depending on the visiting order.

Notice that, in the illustrative example shown in Figure 2, pixels *c* and *l* are classified as ridges, because the pixel *c* has equal distance to pixels *b* and *d,* both on lower boundaries, and pixel *l* has equal slope in the both sliding directions. Pixels *e* and *f* are classified into the same region as pixel *g*. Pixels *h* and *i* are given the same label, and pixel *k* is given another label. Pixel *j* is not classified as a ridge because its slope to the pixel *i* is steeper than its slope to the pixel *k*. The vertical dashed lines partition the one-dimensional signal into four regions, and the arrows show the sliding directions.

III. ORDER-INVARIANT TOBOGGAN ALGORITHM

This section presents the pseudo code of the proposed order-invariant toboggan algorithm. The proposed toboggan algorithm maintains a list of sliding direction(s) for each non-flat pixel (including those lower boundary pixels neighboring the flat regions). It is achieved by the following two steps.

First, we examine the neighbors of each pixel and record the steepest downward direction in the sliding list (please refer to lines 5-16 of **Algorithm 1** given below). After the first step, the sliding lists of the non-flat region and of the lower boundary of the non-local-minimum flat regions are non-empty. Second, the region-growing technique (or more precisely, the breath first search) is used to find the shortest path(s) from each inner pixels in the non-local-minimum flat regions to its lower boundary, and the possible directions for the shortest paths are put in the sliding lists of these inner pixels (lines 17-29 of **Algorithm 1**). After the second step, the remaining empty sliding lists belong to the local-minimum flat regions.

Then, the local-minimum flat regions are recognized and labeled by finding connected components (lines 30-45 of **Algorithm 1**). Finally, a topological sort is applied by taking the pixels as the vertices and the corresponding pixels in the sliding list as the targets of the directed edges. The core of the topological sort is a depth first search, which traverses the directed graph in post-order (lines 46-63 of **Algorithm 1**). Hence, we can visit the pixels in the order that the labels of the former pixels are independent of the latter pixels. It guarantees that the whole image can be segmented in linear time. To make the segmentation order-invariant, a pixel is labeled as ridge if the labels of the neighbors directed by the sliding list are not consistent. The following is the pseudo code of the toboggan algorithm:

Algorithm 1.

```
1   PROCEDURE Toboggan-Algorithm
2   INPUT: Gradient Image G
3   OUTPUT: Label Image L
4     Initiate FIFO Queue

Part 1. Simulation of sliding.

5     FOR-EVERY p ∈ DOMAIN(G) DO
6       h := G(p)
7       h_MIN := the minimal value of G in
                 Neighbor(p)
8       IF h > h_MIN THEN
9         S := {q | G(q) = h_MIN AND
                    q ∈ Neighbor(p)}
10        Sliding-List(p) := S
11        Queue  p
12        Growing-Dist(p) := 0
13      ELSE
14        Sliding-List(p) := Ø
15      END-IF
16    END-FOR

Part 2. Region growing from lower boundary.

17    WHILE Queue is not empty DO
18      p  Queue
19      d := Growing-Dist(p) + 1
20      h := G(p)
21      FOR-EVERY q ∈ Neighbor(p) AND
                  G(q) = h DO
22        IF Sliding-List(q) = Ø THEN
23          Append p to Sliding-List(q)
24          Growing-Dist(q) := d
25          Queue  q
26        ELSE-IF Growing-Dist(q) = d THEN
```

```
27          Append p to Sliding-List(q)
28        END-IF
29      END-WHILE
```

Part 3. Labeling the local-minimum flat regions.

```
30      FOR-EVERY p0 ∈ DOMAIN(G) AND
                    Sliding-List(p0) = Ø DO
31        IF L(p0) is not assigned THEN
32          L(p0) := NEW LABEL
33          Queue   p0
34          h := G(p0)
35          WHILE Queue is not empty DO
36            p   Queue
37            FOR-EVERY q ∈ Neighbor(p) AND
                          G(q) = h DO
38              IF L(q) is not assigned THEN
39                L(q) := L(p0)
40                Queue   q
41              END-IF
42            END-FOR
43          END-WHILE
44        END-IF
45      END-FOR
```

Resolving labels based on the sliding lists.

```
46      FOR-EVERY p ∈ DOMAIN(G) DO
47        Resolve(p)
48      END-FOR


49      PROCEDURE Resolve
50      INPUT: Pixel p
51        IF L(p) is not assigned THEN
52          S := Sliding-List(p)
53          FOR-EVERY q ∈ S DO
54            Resolve(q)
55          END-FOR
56          IF S has a unique label α THEN
57            L(p) := α
58          ELSE
59            L(p) := RIDGE-LABEL
60          END-IF
61        END-IF
62      END-PROCEDURE
63    END-PROCEDURE.
```

IV. ORDER-VARIANT VERSION OF THE TOBOGGAN ALGORITHM

Some applications require that every pixel be classified to exactly one segment. This can be achieved efficiently by performing an order-variant version of the above algorithm, but the disadvantage is that the partition of the thick edges can be biased depending on the visiting order. To obtain the order-variant versions of the toboggan algorithm, we can rewrite lines 9 as

S := list of the first pixel in $\{q \mid G(q) = h_{\text{MIN}}$ **AND** $q \in \text{Neighbor}(p)\}$

and remove the following lines: 12, 19, 24, 26, 27, 58, and 59. After the modification, the uniqueness conditions in lines 56 become always true.

V. PREPROCESSING OF GRADIENT IMAGE

Some preprocessing of the gradient image is usually used to reduce the number of segments obtained by the toboggan and watershed algorithms. In [8], the gradient image is reconstructed by erosion. Instead, we use the reconstruction by closing [7] with some modification. The original morphological grayscale reconstruction defined by [7] has to repeat the elementary operation until the stability is reached. Although an efficient hybrid algorithm is proposed to improve the performance, the computation time still can be high in the worst case. In some applications, it is acceptable to preserve shallow valleys having large radiuses. Therefore, the reconstruction we used here is to repeat the elementary operation in a fixed number of iterations. The morphological grayscale reconstruction is defined as

$$\delta_M^{(n)}(K) = \delta_M^{(n-1)} \circ \delta_M^{(1)}(K),\ \forall p,\ K(p) < M(p) \quad (1)$$

$$\varepsilon_M^{(n)}(K) = \varepsilon_M^{(n-1)} \circ \varepsilon_M^{(1)}(K),\ \forall p,\ K(p) > M(p) \quad (2)$$

where $n > 1$, K and M are the kernel and mask images, respectively, and the elementary operations are

$$\delta_M^{(1)}(K) = (K \oplus B_{3\times3}) \wedge M \quad (3)$$

$$\varepsilon_M^{(1)}(K) = (K \ominus B_{3\times3}) \vee M \quad (4)$$

$\oplus$, $\ominus$, $\wedge$, $\vee$, and $B_{3\times3}$ stand for dilation, erosion, point-wise minimum, point-wise maximum, and a 3×3 square structuring element, respectively. Then, our reconstruction is defined to eliminate the valleys which are small (the contour of ridge cannot hold a $(2r+1)\times(2r+1)$ square) and shallow (the height of the surrounding ridge is less than h):

$$G' = \varepsilon_G^{(r)} \delta_{G+h}^{(r)}(G) \qquad (5)$$

where G is the original gradient image, and G' is the reconstructed gradient image. The reason of using closing is that the dilation can fill up the undesired valleys and the erosion can correct the bias dilated ridges.

VI. EXPERIMENTS

The original gradient image in our experiments is obtained by using the Sobel operator, i.e.

$$G(i,j) = \sqrt{G_x^2(i,j) + G_y^2(i,j)} \qquad (6)$$

where

$$G_x = I \otimes \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, G_y = I \otimes \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix},$$

I is the luminance of the input image, and $\otimes$ stands for the convolution. The input image can be 24-bit color or grayscale in 256 levels. The luminance of the color image is obtained by

$$Lu = \begin{pmatrix} 0.212671 \\ 0.715160 \\ 0.072169 \end{pmatrix}^T \begin{pmatrix} R \\ G \\ B \end{pmatrix} \qquad (7)$$

where R, G, and B are the values of red, green, and blue, respectively, in 256 levels, and Lu is rounded to the unit.

In this section, we will compare the performance of the proposed toboggan algorithm with the more popular watershed algorithm. We have tried our best to optimize the watershed algorithm and make sure that the segmentation results obtained by using both algorithms are the same.

In our implementation, the data type of the gradient magnitude can be either short integer or floating point. If the short integer is chosen, the counting sort algorithm is used for the watershed algorithm, and the values of the gradient magnitudes are rounded to the unit. If the floating point is chosen, the quick sort algorithm is used for the watershed algorithm, and the values of the gradient magnitudes are rounded to the single precision. The program of our implementation is written in C++ and compiled by the Microsoft Visual C++ compiler.

Figure 3. The input image used in the experiments.

Figure 4. The gradient image obtained by the Sobel operator without grayscale reconstruction.

Table 1. The computation time in different configurations.

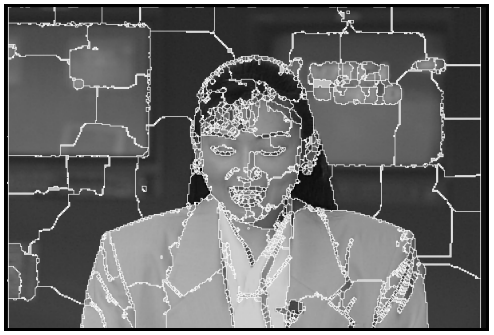
No.	Algorithm	Data Type	Reconstruction	Order-Invariant	Connectivity	Execution Time
1	Toboggan	Integer	×	×	4-neighbor	62 ms
	Watershed					63 ms
2	Toboggan	Floating Point	×	×	4-neighbor	73 ms
	Watershed					144 ms
3	Toboggan	Integer		×	4-neighbor	60 ms
	Watershed					79 ms
4	Toboggan	Integer	×		4-neighbor	40 ms
	Watershed					52 ms
5	Toboggan	Integer	×	×	8-neighbor	86 ms
	Watershed					81 ms

Table 1 lists the parameters and the execution time of the toboggan and watershed algorithms in five different configurations. The parameters of the grayscale reconstruction are set as $h = 32$ and $r = 8$. The input image and its gradient image are shown in Figures 4 and 5, respectively. The size of image is 320(w)×242(h). In each configuration, the toboggan and watershed algorithms generate the same segmentation result. The segmentation results of the five experiments are shown in Figure 6.

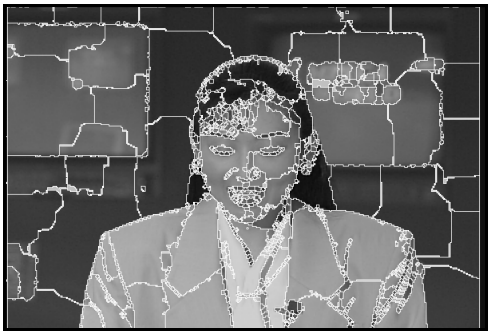
The execution time in Table 1 is the average of 100 experimental results on a Pentium-II PC, excluding the computation time of the generation and reconstruction of the gradient image. We can use Experiment 1 as the standard configuration, where the data type of the gradient magnitude is integer, the gradient image is reconstructed by closing, order-invariant algorithms are used, and the pixels are 4-connected. The experiment shows that the average execution time of the toboggan and watershed algorithms is very close.

In Experiment 2, the floating point is chosen as the data type, so the watershed algorithm must use a comparison-based sorting algorithm and takes more time than the toboggan algorithm. In Experiment 3, the gradient image is not reconstructed, so an over-segmented result is generated (see Figure 6). It is interesting that this change almost has no influence on the toboggan algorithm, but it makes the watershed algorithm significantly slower.

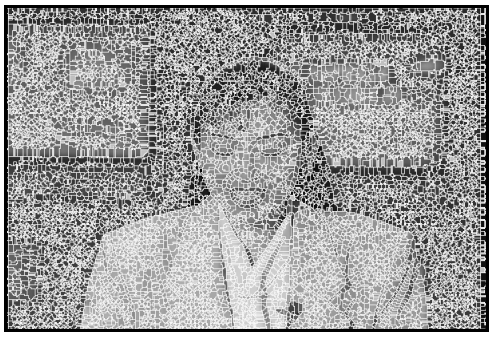
In Experiment 4, order-variant algorithms are used to segment the image. Although both the toboggan and watershed algorithms become faster than the order-invariant versions, the toboggan algorithm is more efficient than the watershed algorithm. In Experiment 5, 4-connected neighbors are used instead. Because the resolving procedure of the toboggan algorithm is relative to the connectivity, the watershed algorithm is slightly faster than the toboggan algorithm.



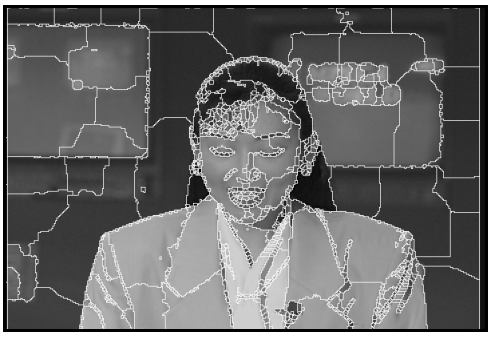
Experiment 1 (standard)



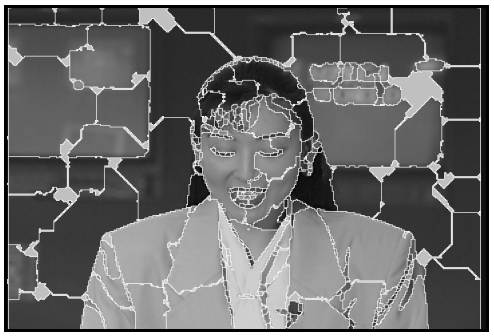
Experiment 2 (floating point)



Experiment 3 (no reconstruction)



Experiment 4 (order-variant)



Experiment 5 (8-neighbor)

Figure 5. The segmentation results of five different configurations.

In our experiments, the toboggan algorithm is faster than the watershed algorithm in most cases. Only that the connectivity becomes large (8-neighbor), the toboggan algorithm is slightly slower than the watershed algorithm. It is worthy to mention that our program assumes the neighbors are in the equal distance, so that we can use the region-growing technique to find the shortest paths between the inner pixels and the lower boundary in the non-local-minimum flat regions. This assumption makes more ambiguous pixels inside the non-local-minimum flat regions. Thus we can see thicker edges in the segmentation result of Experiment 5.

VII. CONCLUSION

This paper has presented a new order-invariant toboggan algorithm for image segmentation. Our experiments show that, with careful implementation, the proposed toboggan algorithms and the more popular watershed algorithm can generate the same segmentation result. However, the proposed toboggan

algorithm is not only more applicable to different data types of the gradient magnitude but also faster than the watershed algorithm in most cases. Practically, the toboggan algorithm requires less memory than the watershed algorithm, because the sliding list and label image of the toboggan algorithm can share memory, but the watershed algorithm requires additional memory for storing the sorting results.

REFERENCES

[1] J. Fairfield, "Toboggan Contrast Enhancement for Contrast Segmentation," in *Proceedings of the 10th IEEE International Conference on Pattern Recognition*, vol. 1, pp. 712–716, 1990.

[2] J. M. Gauch, "Image Segmentation and Analysis via Multiscale Gradient Watershed Hierarchies," *IEEE Transactions on Image Processing*, vol. 8, no. 1, pp. 69–79, January 1999.

[3] Y.-P. Hung and X. Yao, "Keep-Sliding Toboggan Image Segmentation," in *Proceedings of National Computer Symposium*, vol. 2, pp. 392–397, Taiwan, December 1991.

[4] A. Moga, B Cramariuc, and M. Gabbouj, "An Efficient Watershed Segmentation Algorithm Suitable for Parallel implementation," in *Proceedings of IEEE International Conference on Image Processing*, vol. 2, pp. 101–104, Washington, D.C., October 1995.

[5] E. N. Mortensen and W. A. Barrett, "Toboggan-Based Intelligent Scissors with a Four-Parameter Edge Model," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 452–458, 1999.

[6] L. Vincent and P. Soille, "Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 6, pp. 583–598, June 1991.

[7] L. Vincent, "Morphological Grayscale Reconstruction in Image Analysis: Applications and Efficient Algorithms," *IEEE Transactions on Image Processing*, vol. 2, no. 2, pp. 176–201, April 1993.

[8] D. Wang, "Unsupervised Video Segmentation Based on Watersheds and Temporal Tracking," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 5, pp. 539–546, September 1998.