Comparison Between Immersion-Based and Toboggan-Based Watershed Image Segmentation

Yung-Chieh Lin, Yu-Pao Tsai, Yi-Ping Hung, and Zen-Chung Shih

***Abstract*—Watershed segmentation has recently become a popular tool for image segmentation. There are two approaches to implementing watershed segmentation: immersion approach and toboggan simulation. Conceptually, the immersion approach can be viewed as an approach that starts from low altitude to high altitude and the toboggan approach as an approach that starts from high altitude to low altitude. The former seemed to be more popular recently (e.g., Vincent and Soille), but the latter had its own supporters (e.g., Mortensen and Barrett). It was not clear whether the two approaches could lead to exactly the same segmentation result and which approach was more efficient. In this paper, we present two "order-invariant" algorithms for watershed segmentation, one based on the immersion approach and the other on the toboggan approach. By introducing a special RIDGE label to achieve the property of order-invariance, we find that the two conceptually opposite approaches can indeed obtain the same segmentation result. When running on a Pentium-III PC, both of our algorithms require only less than 1/30 s for a 256 $\times$ 256 image and 1/5 s for a 512 $\times$ 512 image, on average. What is more surprising is that the toboggan algorithm, which is less well known in the computer vision community, turns out to run faster than the immersion algorithm for almost all the test images we have used, especially when the image is large, say, 512 $\times$ 512 or larger. This paper also gives some explanation as to why the toboggan algorithm can be more efficient in most cases.**

***Index Terms*—Immersion approach, order-invariance, toboggan approach, watershed image segmentation.**

I. Introduction

IMAGE segmentation plays a very important role in computer vision. Many innovative methods have been proposed in the last few decades, but automatic image segmentation for general applications still remains to be an open problem. Recently, watershed segmentation became a popular tool for different applications that require image segmentation, such as machine inspection, aerial image understanding, medical image analysis, and video object segmentation [5], [6], [8], [12]–[15]. In fact, the watershed transform has been studied for a long time and began to be used for image segmentation about two decades ago, e.g., [11]. A brief historical review can be found in [6].

The basic idea of watershed segmentation is to consider the regions to be extracted as catchment basins in topography. The watershed lines are then the boundaries of catchment basins. If one applies the watershed segmentation method directly to the intensity image, he may not obtain meaning meaningful segmentation results for most images. However, it will make more sense if the watershed segmentation method is applied to gradient magnitude images. That is, we can view the gradient magnitude image as a simple topographic surface by treating the gradient magnitude value as the height (or the altitude). Given a grayscale image, the computation of gradient magnitude is quite straightforward, though many variations exist (e.g., Sobel, Gaussian, or Morphological gradient operator) [9]. As for color images, an example of computing gradient magnitude for watershed segmentation can be found in [5].

An interesting thought closely related to watershed segmentation was the one presented by Fairfield in 1990, where an image is segmented by first applying the toboggan contrast enhancement and then a simple contrast segmentation [1]. The concept of the toboggan contrast enhancement is very similar to watershed segmentation. An illustrative example of the toboggan contrast enhancement is given below for comparison with the watershed image segmentation.

Consider Fig. 1, where $\mathrm{f(x)}$ shown in Fig. 1(a) is a one-dimensional (1-D) signal. Let $\mathrm{G(x)} = |\mathrm{f'(x)}|$ be the absolute value of the first derivative of $\mathrm{f(x)}$, as shown in Fig. 1(c). When extending $f(\cdot)$ to a two-dimensional (2-D) intensity image, $G(\cdot)$ will be used to denote the magnitude of the image gradient. In Fig. 1(c), we can find two local maxima, P_1 and P_2, at the positions of the inflective points of the original signal $\mathrm{f(x)}$. Notice that these two peaks (or crest lines in 2-D gradient magnitude images) are exactly the watershed points (or lines) for region segmentation. Hence, the original signal can be partitioned into three segments while enhanced to be the piecewise constant signal illustrated in Fig. 1(d).

In Section II, we first review two typical approaches to implementing toboggan image segmentation, one using immersion approach and the other using toboggan simulation. Based on the concept of immersion simulation, we introduce in Section III an order-invariant version of the famous Vincent and Soille algorithm. Then, in Section IV, we propose a new order-invariant algorithm based on toboggan simulation. This toboggan algorithm can achieve exactly the same segmentation result as what

Manuscript received December 19, 2003; revised January 17, 2005. This work was supported in part by the National Science Council, Taiwan, R.O.C., under Grant NSC 93-2752-E-002-007-PAE. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Eli Saber.

Y.-C. Lin is with the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, R.O.C. (e-mail: cole@iadea.com).

Y.-P. Tsai is with the Institute of Information Science, Academia Sinica, Taipei, Taiwan, R.O.C., and also with the Department of Computer and Information Science, National Chiao-Tung University, Hsinchu 300, Taiwan, R.O.C. (e-mail: gis90806@cis.nctu.edu.tw).

Y.-P. Hung is with the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, R.O.C., and also with the Institute of Information Science, Academia Sinica, Taipei, Taiwan, R.O.C., and the Graduate Institute of Networking and Multimedia, National Taiwan University, Taipei, Taiwan, R.O.C. (e-mail: hung@csie.ntu.edu.tw).

Z.-C. Shih is with the Department of Computer and Information Science, National Chiao-Tung University, Hsinchu 300, Taiwan, R.O.C. (e-mail: zcshih@cis.nctu.edu.tw).

Digital Object Identifier 10.1109/TIP.2005.860996

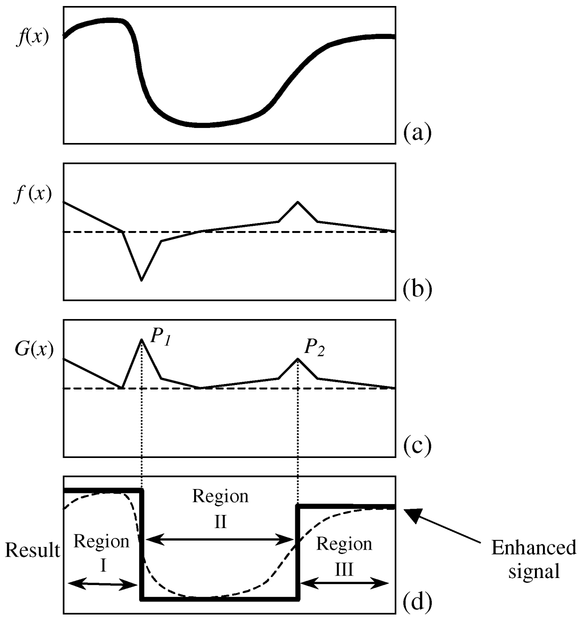


Fig. 1. Illustrative example of toboggan contrast enhancement and segmentation.

is achieved by the immersion algorithm presented in Section IV. We show in Section V that, surprisingly, this toboggan algorithm is in general more efficient than the immersion algorithm. Section VI gives some concluding remarks.

II. Overview of Two Algorithmic Approaches

While the concept of the watershed image segmentation is quite simple, its implementation turns out to be more involved, especially when one wants to deal with the plateau and ambiguous pixels in digital spaces in an appropriate way (more details will be discussed in Sections III and IV). Typically, there are two approaches to implementing the watershed method: the immersion approach and the toboggan simulation.

The immersion approach [6] is also referred to as the flooding analogy. In the immersion simulation, we first pierce a hole in every local minimum of the topographic surface formed by the gradient magnitude image. Then, we slowly immerse the topographic surface in water. Starting from the minima of lowest altitude, the water will progressively fill up all the different catchment basins. At some point, the rising water in any one specific basin will start to merge with water coming from its neighboring basins. Suppose that this merging can be prevented by constructing dams at the merging sites all the way up to the highest surface altitude (or until the immersion procedure ends). At the end of this immersion simulation, each basin will be completely surrounded by dams and the location of dams corresponds to watershed line. A good depiction of the 1-D example and its extension to 2-D can be found in [10].

Another approach to implementing watershed segmentation is the toboggan approach [5]. This approach is called "tobogganing" because of its similarity to riding a sled downhill to the bottom of a basin. The toboggan approach is also referred to as the drainage analogy (i.e., any two points are in the same region if they drain to the same minimum point). This approach tries to find a downstream path (along the steepest descent) from each pixel of the gradient magnitude image to a local minimum of the topographic surface. Pixels that slide into the same local minimum can be efficiently grouped together by assigning them a unique label. A catchment basin (or watershed region) is then defined to be the set of pixels having the same label. Here, adjacent watershed regions are divided by a boundary path where a drop of water has equal chance of flowing into adjacent watershed regions. This boundary path is the watershed line, which usually occurs along the peaks of ridge-like structure.

Conceptually, the immersion approach can be viewed as an approach that starts from low altitude to high altitude, and the toboggan approach an approach that starts from high altitude to low altitude (even though the latter also requires an upward backtracking after the downward tobogganing). The question is can these two opposite simulations lead to exactly the same image segmentation result? Vincent and Soille mentioned in their classic work [6] that the toboggan approach (which they called "steepest slope line") was an intuitive approach but was not well suited to practical implementations in digital spaces. We agree with their first point but not the second one. It might be true that their toboggan algorithm could yield biased results in some cases (due to the ambiguity problem caused by the digital spaces, which will be explained in the next section), but it would not be the case if they had used the order-invariant toboggan algorithm proposed in Section IV. In fact, it can be demonstrated that our simple toboggan algorithm proposed in Section IV can always obtain exactly the same segmentation results as that is obtained by the immersion algorithm presented in Section III. The C++ source code can be downloaded from http://ippr.csie.ntu.edu.tw.

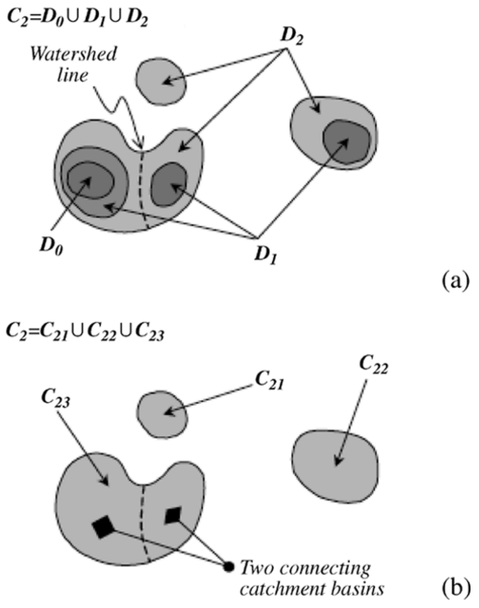
III. Order-Invariant Immersion Algorithm

This section presents the details of our order-invariant immersion algorithm for image segmentation. Similar to the one proposed by Vincent and Soille in [6], this algorithm first requires a sorting of the pixels in the increasing order of their gradient magnitude values before running the level-by-level flooding step. It is this sorting step that has made the level-by-level flooding step efficient enough so that the Vincent and Soille algorithm can surpass its predecessors in computational efficiency. However, this sorting step is also one of the reasons that have made the immersion algorithm slightly less efficient than the toboggan algorithm presented in the next section.

Let $\mathrm{G} : \mathrm{D} \rightarrow \mathrm{R}^+$ be a gradient magnitude image, where D is the indexing domain of the image (e.g., $\mathrm{D} = \mathrm{Z}^2$). Let min(G) and max(G) be the minimum value and the maximum value of G, respectively. By sorting the pixels of G in the increasing order of their gradient magnitude values, we can easily decompose D into a finite number of disjoint level sets, each denoted by

$$\mathrm{D_h} = \{\mathbf{p} \in \mathrm{D} | \mathrm{G}(\mathbf{p}) = \mathrm{h}\}.$$

That is, we have $D = \bigcup_{\forall h} D_h$, $\min(\mathrm{G}) \leqq \mathrm{h} \leqq \max(\mathrm{G})$, and $D_k \cap D_l = \phi$ if $\mathrm{k} \neq l$. Different sorting techniques can be used here. For better efficiency, our algorithm uses counting sort if the data type of the gradient magnitude is fixed point, while it uses quick sort if the data type is floating point.



$C_2=D_0 \cup D_1 \cup D_2$
Watershed line
D_2
D_0
D_1
(a)
$C_2=C_{21} \cup C_{22} \cup C_{23}$
C_{21}
C_{22}
C_{23}
Two connecting catchment basins
(b)

Fig. 2. Example of decomposing a topographic surface into disjoint level sets. Here, darker regions in (a) represent areas of lower altitude, or deeper water after immersion. The pixels in white region have altitudes greater than 2 and need not to be considered in this example.

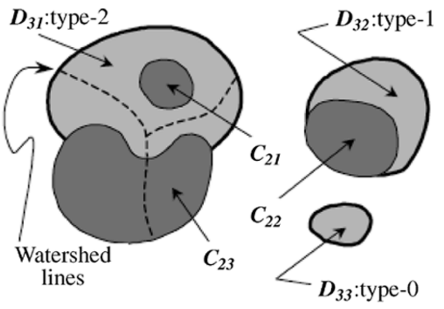


Fig. 3. Example of the relation between D_3 and C_2, where $D_3 = D_{31} \bigcup D_{32} \bigcup D_{33}$ and $C_2 = C_{21} \bigcup C_{22} \bigcup C_{23}$, as shown in Fig. 2.

Once the pixels are sorted, we can simulate the immersion process efficiently with a level-by-level flooding step. At each level h, we have a set of connected components whose union constitutes the level set $\mathrm{D_h}$. Fig. 2(a) shows an example of three level sets, $\mathrm{D_0}$, $\mathrm{D_1}$, and $\mathrm{D_2}$. Without loss of generality, we assume here that $\mathrm{h} = 0, 1, \ldots, 255$, for the simplicity of the following explanation.

Suppose that water has risen to a particular level h-1 during the immersion process. Let $\mathrm{C}_{h-1} \equiv \bigcup_{l=0}^{h-1} D_l$. Notice that $\mathrm{C_{h-1}}$ can also be viewed as a union of connected components, i.e., $C_{h-1} = \bigcup_i C_{(h-1)i}$, where each connected component $\mathrm{C_{(h-1)i}}$ contains one or more than one catchment basin. The catchment basin here can be referred to as a pre-h catchment basin because it is formed right before the water level rises up to level-h. For example, $\mathrm{C_2}$ in Fig. 2(b) contains three connected components, denoted by $\mathrm{C_{21}}$, $\mathrm{C_{22}}$, and $\mathrm{C_{23}}$. Note that $\mathrm{C_{23}}$ contains two pre-2 catchment basins separated by a watershed line, while each of $\mathrm{C_{21}}$ and $\mathrm{C_{22}}$ contains only one pre-2 catchment basin.

Next, by letting the water level goes up to level h, we have a new level set $\mathrm{D_h}$, which can also be viewed as a union of connected components, i.e., $D_h = \bigcup_j D_{hj}$. An example is given in Fig. 3. The relation between $\{D_{hj}, \forall j\}$ and $\{C_{(h-1)i}, \forall i\}$ is quite important for understanding the two order-invariant algorithms proposed in this paper because $\{C_{(h-1)i}, \forall i\}$ contains all the pre-h catchment basins, which are all the areas already beneath the water level right before water floods into the level set $\mathrm{D_h}$ and, hence, are the sources where water floods from. We then classify each connected component in $\{D_{hj}, \forall j\}$ into one of the following three types based on the number of its connecting pre-h catchment basins.

Type-2 Component: More than one pre-h catchment basin is connected to this type of connected component. For example, $\mathrm{D_{31}}$ in Fig. 3 is a type-2 component because it connects to three pre-2 catchment basins. Notice that $\mathrm{C_{23}}$ actually contains two pre-2 catchment basins.

Type-1 Component: Exactly one pre-h catchment basin is connected to this type of connected component. For example, $\mathrm{D_{32}}$ in Fig. 3 is a type-1 component because it has only one pre-2 catchment basin, $\mathrm{C_{22}}$.

Type-0 Component: No pre-h catchment basin is connected to this type of connected component. For example, $\mathrm{D_{33}}$ in Fig. 3 is a type-0 component.

Notice that when the flooding has been completed up to level h-1, every pixel having altitude less than or equal to h-1 will have already been assigned a unique catchment basin label. As the water level goes up from level h-1 to level h, water (or catchment basin labels) will flood into all the three types of components. The direction of flooding motivates and guides the labeling process of the immersion algorithm. For a type-2 component, water emerging from its multiple neighboring catchment basins will meet at watershed lines, as depicted in Fig. 3. For a type-1 component, water can only emerge from one single catchment basin, and, hence, all the pixels in this component will have the same label as its neighboring catchment basin. Finally, for a type-0 component having no neighboring catchment basin, we have to pierce a hole in it to let water flood in. That is, this connected component, in $\{D_{hj}, \forall j\}$, is detected as a new local-minimum, i.e., a newly discovered catchment basin, and will be assigned a new label.

To implement the flooding step, we divide the pixels in $\mathrm{D_h}$ into the following three classes and label the pixels in each class one by one. It is worth mentioning that the classification of the connected components in $\mathrm{D_h}$ is for understanding the algorithm, while the following classification of the individual pixels in $\mathrm{D_h}$ is for implementing the algorithm.

Class-I Pixel: A pixel $\mathbf{p}$ in $\mathrm{D_h}$ is called a class-I pixel if its altitude is strictly greater than the altitude of its lowest neighbor, i.e., if $\mathrm{G}(\mathbf{p}) > \mathrm{h_{MIN}}(\mathbf{p})$, where $\mathrm{h_{MIN}}(\mathbf{p}) = \min\{\mathrm{G}(\mathbf{q}), \mathbf{q} \in \mathrm{Neighbor}(\mathbf{p})\}$. If we consider each connected component in $\mathrm{D_h}$ as a plateau (i.e., a set of connected pixels having constant altitude) in the topographic surface, then class-I pixels are those locating located on the descending boundaries of plateaus. Note that a single pixel is also a plateau (of unit area). As can be seen in Fig. 4, $\mathrm{D_{31}}$ and $\mathrm{D_{32}}$ of Fig. 3 contain some class-I pixels while $\mathrm{D_{33}}$ contains no class-I pixels because $\mathrm{D_{33}}$ has no pre-h catchment basins connecting to it.

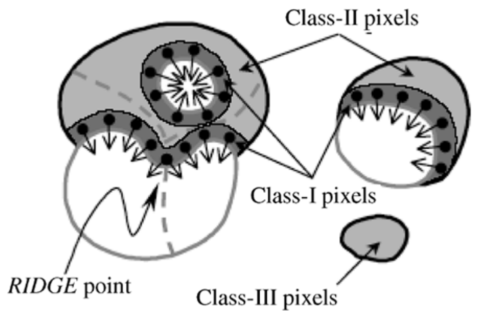


Fig. 4. Illustrative example of three classes of pixels contained in D_3.

Class-II Pixel: A pixel $\mathbf{p}$ in $\mathrm{D_h}$ is called a class-II pixel if it is contained in a type-2 or type-1 component but is not a class-I pixel. This class of pixels can be viewed as interior pixels of nonlocal-minimum plateaus. Fig. 4 shows some examples of class-II pixels.

Class-III Pixel: Pixels in type-0 components of $\mathrm{D_h}$ are class-III pixels. All the pixels in one type-0 component will be assigned a new and unique label. An example is given in Fig. 4.

Based on the above classification of pixels, we can summarize the immersion algorithm for watershed segmentation as follows.

Algorithm 1. Immersion Approach

```
Step 1.   Sorting Step:
            Sort all the pixels in the
          gradient magnitude image G to
          obtain level sets $D_h$ in in-
          creasing h.
Step 2.   Flooding Step:
            For each level set $D_h$, in
          the increasing order of h.
Step 2.1.   Simulate flooding for all
            the class-I pixels in $D_h$
            by labeling each class-I
            pixel with the label of its
            lowest neighbor (care has
            to be taken for ambiguous
            cases, as explained below).
            All these class-I pixels
            are pushed into a FIFO
            queue for region growing in
            Step 2.2.
Step 2.2.   Simulate flooding for all
            class-II pixels in $D_h$ by
            region growing from class-I
            pixels using the FIFO queue
            initialized in Step 2.1.
Step 2.3.   Simulate flooding for
            class-III pixels in $D_h$ by
            assigning a new and unique
            label to each of the type-0
            components in $D_h$.
```

In step 2.1, we first label each class-I pixel in $\mathrm{D_h}$ with the label of its lowest neighbor. If the pixel has more than one lowest neighbor, we then check if these multiple lowest neighbors have the same catchment basin label (note that all the lower neighbors have been labeled at previous iterations). If the labels of the multiple lowest neighbors are consistent, we simply label this pixel with the consistent label. If they are not consistent, we then have an ambiguous situation, which occurs only in the digital spaces. In this situation, one possibility for label assignment is to choose the label of any of the lowest neighbors, either randomly or based on the visiting order of neighbor scanning. However, this kind of arbitrary assignment will result in the biased segmentation results, which Vincent and Soille have criticized the toboggan algorithm for [6].

To solve the above ambiguity problem, we assign a special label, RIDGE, to those ambiguous pixels, and achieve an order-invariant segmentation algorithm, i.e., an algorithm whose segmentation results are independent of the visiting order of neighbor scanning. It is the property of order-invariance that makes both the immersion and toboggan algorithms obtain exactly the same segmentation results.

The following is the pseudocode for this algorithm.

```
1  PROCEDURE Watershed-Algorithm
2  INPUT: Gradient Image $G$
3  OUTPUT: Label Image $L$

        Sorting the whole gradient image
.
4    Sort $G \Rightarrow G_h := \{p|G(p) = h\}$
5    FOR $h := \mathrm{Min}(G)$ UPTO $\mathrm{Max}(G)$ DO
6      Initiate FIFO $Queue$

        Part 1. Simulation of flooding.

7      FOR-EVERY $p \in G_h$ DO
8        $h_{\mathrm{MIN}}$ := the minimal value of $G$ in $\mathrm{Neighbor}(p)$
9        IF $h > h_{\mathrm{MIN}}$ THEN
10         $S := \{q|G(q) = h_{\mathrm{MIN}}$ AND $q \in \mathrm{Neighbor}(p)\}$
11         IF $S$ has a unique label $\alpha$ THEN
12           $L(p) := \alpha$
13         ELSE
14           $L(p) := \mathrm{RIDGE-LABEL}$
15         END-IF
16         $Queue \leftarrow p$
17         $Growing-Dist(p) := 0$
18       END-IF
19     END-FOR

        Part 2. Region growing from lower boundary.

20     WHILE $Queue$ is not empty DO
21       $p \leftarrow Queue$
22       $d := Growing-Dist(p) + 1$
23       FOR-EVERY $q \in \mathrm{Neighbor}(p)$ AND $G(q) = h$ DO
```

```
24          IF $L(q)$ is not assigned THEN
25            $L(q) := L(p)$
26            $Growing-Dist(q) := d$
27            $Queue \leftarrow q$
28          ELSE-IF $L(p) \neq L(q)$ AND $Growing-Dist(q) = d$ THEN
29            $L(q) := \text{RIDGE}-\text{LABEL}$
30          END-IF
31        END-FOR
32      END-WHILE
```

$\boxed{\text{Part 3. Labeling the local}-\text{minimum flat regions.}}$

```
33      FOR-EVERY $p_0 \in G_h$ AND $L(p_0)$ is not assigned DO
34        $L(p_0) := \textbf{NEW}$ LABEL
35        $Queue \leftarrow p_0$
36        WHILE $Queue$ is not empty DO
37          $p \leftarrow Queue$
38          FOR-EVERY $q \in \text{Neighbor}(p)$ AND $G(q) = h$ DO
39            IF $L(q)$ is not assigned THEN
40              $L(q) := L(p_0)$
41              $Queue \leftarrow q$
42            END-IF
43          END-FOR
44        END-WHILE
45      END-FOR
46    END-FOR
47  END-PROCEDURE.
```

IV. Order-Invariant Toboggan Algorithm

This section introduces a new order-invariant toboggan algorithm for watershed segmentation. Our toboggan algorithm originates from the toboggan contrast enhancement proposed by Fairfield [1]. The basic idea of toboggan contrast enhancement is very simple, and it took only seven lines for Fairlield to write down its pseudocode. One drawback of his algorithm is that too many small regions may be generated due to the existence of nonsingle-pixel plateaus in the topographic surface. A simple keep-sliding technique can be used to solve this problem [3]. However, the algorithms presented in [1] and [3] require contrast segmentation as a post processing in order to acquire image segmentation results. Hence, they cannot obtain the same segmentation result as the immersion algorithm.

Recently, Mortensen and Barrett proposed a toboggan segmentation algorithm that did not require contrast segmentation as a post processing, and successfully integrate it with intelligent scissors [5]. The major problem with their algorithm is that it did not deal with the ambiguity problem mentioned in the previous section, and, thus, is not order-invariant.

Based on the understanding of the immersion algorithm provided in the last section, it is not hard to comprehend our toboggan algorithm given below.

Algorithm 2. Toboggan Approach

```
Step 1.   Simulation of Sliding:
            This step records the sliding
          directions for all the class-I
          and class-II pixels in D (not
          just in $D_h$).
  Step 1.1.  Simulate sliding for each of
             the class-I pixels in D by
             recording its lowest neigh-
             bors in a sliding list. All
             these class-I pixels are
             pushed into a FIFO queue as
             the seeds for region growing
             in Step 2.2.
  Step 1.2.  Simulate keep-sliding for all
             class-II pixels in D by re-
             gion growing from class-I
             pixels (using the FIFO queue
             initialized in Step 2.1).
Step 2.   Label assignment for all the
          class-III pixels:
            Label all the local-minimum
          plateaus (i.e. all the bottom
          levels of catchment basins).
Step 3.   Tobogganing Step:
            Assign label to each unla-
          beled pixel (i.e., class-I and
          class-II pixels) by first to-
          bogganing and then backtracking,
          using best first search.
```

It is interesting to observe the similarity between Algorithm 1 and Algorithm 2. In fact, the functions of Steps 1.1, 1.2, and 2 in Algorithm 2 are very much like those of Steps 2.1, 2.2, and 2.3 in Algorithm 1. They both deal with class-I, class-II, and class-III pixels in the same order, except that Steps 1.1 and 1.2 in Algorithm 2 only record the sliding directions of class-I and class-II pixels without actual labeling, while Steps 2.1 and 2.2 immediately assign label to class-I and class-II pixels and are perform level-by-level. Because the toboggan algorithm does not require level-by-level processing, it does not need the expensive sorting at the beginning. Instead, it needs to resolve the labeling problem at the end by tobogganing and backtracking, but this can be done by efficient depth first search and has been implemented by utilizing transition table in our C++ code.

The details of the above algorithm can be better known with the following pseudocode.

```
48  PROCEDURE Toboggan-Algorithm
49  INPUT: a gradient magnitude image, $G$
50  OUTPUT: a label image, $L$
51    Initialize FIFO $Queue = \Phi$
```

$\boxed{\text{Simulation of sliding for all class}-\text{I pixels}}$

```
52    FOR-EVERY $\mathbf{p} \in D$ DO
```

```
    $h := G(\mathbf{p})$
    $h_{\mathrm{MIN}} := \min\{G(\mathbf{q}), \mathbf{q} \in Neighbor(\mathbf{p})$
    IF $h > h_{\mathrm{MIN}}$ THEN
      $S := \{\mathbf{q} | G(\mathbf{q}) = h_{\mathrm{MIN}}$ AND $\mathbf{q} \in Neighbor(\mathbf{p})\}$
      $Sliding - List(\mathbf{p}) := S$
      $Queue \leftarrow \mathbf{p}$ /*queue for class – I pixels*/
      $Growing - Dist(\mathbf{p}) := 0$
    ELSE
      $Sliding - List(\mathbf{p}) := \varnothing$
    END-IF
  END-FOR

$\boxed{\text{Simulation of keep – sliding for all class – II pixels}}$

  WHILE $Queue$ is not empty DO
    $\mathbf{p} \leftarrow Queue$
    $d := Growing - Dist(\mathbf{p}) + 1$
    $h := G(\mathbf{p})$
    FOR-EVERY $\mathbf{q} \in Neighbor(\mathbf{p})$ AND $G(q) = h$ DO
      IF $Sliding - List(\mathbf{q}) = \varnothing$ THEN
        Append $\mathbf{p}$ to $Sliding - List(\mathbf{q})$
        $Growing - Dist(\mathbf{q}) := d$
        $Queue \leftarrow \mathbf{q}$
      ELSE-IF $Growing - Dist(\mathbf{q}) = d$ THEN
        Append $\mathbf{p}$ to $Sliding - List(\mathbf{q})$
      END-IF
  END-WHILE

$\boxed{\text{Labeling for all class – III pixels}}$

  FOR-EVERY $\mathbf{p}_0 \in D$ AND $Sliding - List(\mathbf{p}_0) = \varnothing$ DO
    IF $L(\mathbf{p}_0)$ is not assigned THEN
      $L(\mathbf{p}_0) :=$ NEW LABEL
      $Queue \leftarrow \mathbf{p}_0$
      $h := G(\mathbf{p}_0)$
      WHILE $Queue$ is not empty DO
        $\mathbf{p} \leftarrow Queue$
        FOR-EVERY $\mathbf{q} \in Neighbor(\mathbf{p})$ AND $G(\mathbf{q}) = h$ DO
          IF $L(\mathbf{q})$ is not assigned THEN
            $L(\mathbf{q}) := L(\mathbf{p}_0)$
            $Queue \leftarrow \mathbf{q}$
          END-IF
        END-FOR
      END-WHILE
    END-IF
  END-FOR

$\boxed{\text{Tobogganing by depth – first search}}$

  FOR-EVERY $\mathbf{p} \in D$ DO
    Resolve($\mathbf{p}$)
  END-FOR

  PROCEDURE Resolve
  INPUT: Pixel site $\mathbf{p}$
    IF $L(\mathbf{p})$ is not assigned THEN
      $S := Sliding - List(\mathbf{p})$
        FOR-EVERY $\mathbf{q} \in S$ DO
          Resolve($\mathbf{q}$)
        END-FOR
        IF $S$ has a unique label $\alpha$ THEN
          $L(\mathbf{p}) := \alpha$
        ELSE
          $L(\mathbf{p}) :=$ RIDGE – LABEL
        END-IF
      END-IF
   END-PROCEDURE
```

V. Experimental Results

Four different watershed algorithms are mentioned in this paper, which are order-variant immersion (Vincent's algorithm [6]), order-variant toboggan, order-invariant immersion, and order-invariant toboggan. In our experiments, we focus on comparing the segmentation result and the computational time of the proposed toboggan and immersion algorithms. We have tried our best in implementation both algorithms by carefully designing the data structure and memory management while making 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 6.0.

In the following experiments, the gradient image is computed by using the multiscale morphological gradient operation described in [8]. Sobel or other operations can be used as well, but the segmentation result may be a little different. Fig. 5(a) shows an example of the image segmentation results obtained by our order-invariant watershed algorithms. Since our immersion algorithm and toboggan algorithm always obtain exactly the same segmentation result, only one result needs to be shown here. It should be noted that without any preprocessing, the results obtained by the watershed method are usually severely over-segmented, as shown in Fig. 5(a). In order to solve the over-segmentation problem, one popular method is to apply some preprocessing, such as geodesic reconstruction, to the gradient image before using the watershed algorithm. For example, the gradient image is first reconstructed by erosion in [8]. In this paper, we use a modified version of reconstruction by closing introduced in [7]. Fig. 5(b) shows that the over-segmentation problem has been largely overcome by the geodesic reconstruction.

Fig. 6 shows some results on an artificial image to show the differences between these algorithms. In 4-connected cases, the Vincent's immersion algorithm generates straight watershed.

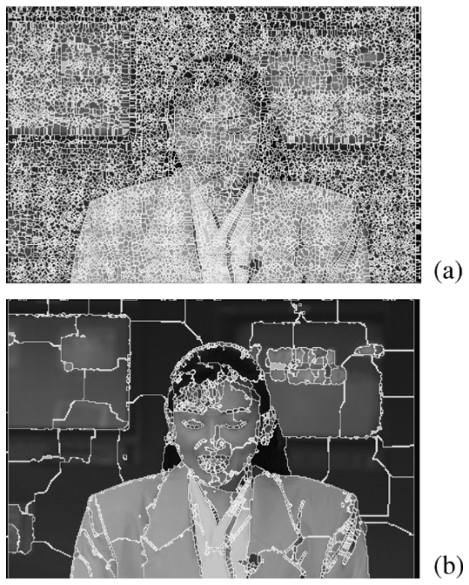


Fig. 5. Watershed image segmentation without and with the preprocessing of geodesic reconstruction. (a) Without geodesic reconstruction. (b) With geodesic reconstruction.

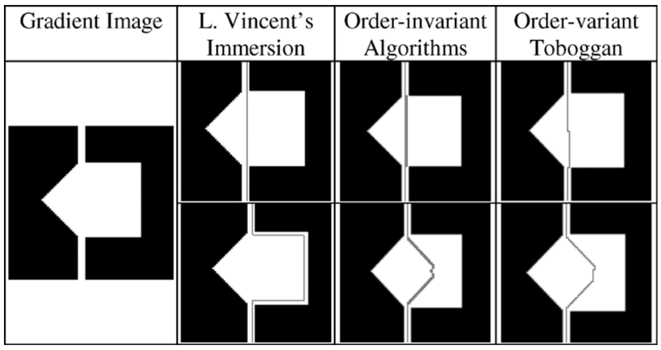


Fig. 6. Test results of the four difference watershed algorithms (order-invariant toboggan and order-invariant immersion produce the same results). The upper row is 4-connected results and the lower row is 8-connected results.

However, the order-invariant algorithm yields an additional ridge due to the difficulties of counting distances on grid. The order-variant algorithm produces nonstraight watershed, which is considered as the worst result. In 8-connected cases, no algorithms produce straight watershed.

It is not unexpected that both of our algorithms, immersion-based and toboggan-based, can obtain exactly the same image segmentation result since we have purposely introduced a special RIDGE label to achieve the property of order-invariance. What is more surprising is that the toboggan algorithm, which is less well-known in the computer vision community, turns out to run faster than the immersion algorithm for most of the test images we have tried. The efficiency gain of the toboggan algorithm will become even larger when the size of the image increases. This phenomenon can be easily observed from the following experiments, which use 77 images arbitrarily chosen from the USC-SIPI image database, or more precisely, from the image groups of "aerials" and "miscellaneous" (can be download from http://sipi.usc.edu/services/database/Database.html).

Fig. 7(d) shows six typical images of the 77 images mentioned above. Fig. 7(a) and (b) shows the execution time for the

TABLE I
WITH RECONSTRUCTION, 4-CONNECTIVITY

	256x256	512x512	1024x1024	2048x2048
ρ	75.32%	94.81%	100.00%	100.00%
T_1	25.78ms	149.97ms	830.15ms	3624.56ms
T_2	24.23ms	107.28ms	439.62ms	1785.71ms

ρ: The percentage of the cases where toboggan is faster
T_1: Average execution time of the immersion algorithm
T_2: Average execution time of the toboggan algorithm

TABLE II
WITH RECONSTRUCTION, 8-CONNECTIVITY

	256x256	512x512	1024x1024	2048x2048
ρ	53.25%	93.51%	100.00%	100.00%
T_1	32.09ms	186.14ms	965.13ms	4147.12ms
T_2	31.86ms	135.57ms	554.59ms	2245.66ms

TABLE III
WITHOUT RECONSTRUCTION, 4-CONNECTIVITY

	256x256	512x512	1024x1024	2048x2048
ρ	92.21%	100.00%	100.00%	100.00%
T_1	27.35ms	173.89ms	916.66ms	3916.28ms
T_2	24.31ms	101.60ms	408.52ms	1647.00ms

six images of size 512×512 and 2048×2048, respectively. Notice that the USC-SIPI images have different sizes, ranging from 256×256 to 1024×1024. To perform the experiments, we apply subsampling to generate a smaller image form a large one and apply repeating to generate a larger image from a small one. Each execution time shown is the average of ten experimental results, using a Pentium-III PC running at 1.0 GHz with 256-MB RAM running at 133 MHz. This execution time does not include the time used for the generation and reconstruction of the gradient image. From Fig. 7(a), we can see that the toboggan algorithm is in general faster than the immersion algorithm, no matter 4-connected or 8-connected neighborhood is used. For images of size 2048×2048, the computational gain of the toboggan algorithm becomes even larger, as can be seen in Fig. 7(b).

In Fig. 7(c), we show that the computational difference is less significant when the image size is relatively small. However, when the image size becomes larger, the execution time for the toboggan algorithm can be two or three times faster than the immersion algorithm. Notice that Fig. 7(c) is shown in logarithmic scale to accommodate the large span of the execution time.

Tables I–III show the average time of all the 77 test images with different sizes, where T_1 and T_2 are the average execution time for the immersion algorithm and the toboggan algorithm, respectively. To disclose more information, we have also shown the percentage of the images for which the toboggan algorithm is faster than the immersion algorithm. We can see that, as the image size is large enough, say 1024×1024, the toboggan algorithm is always faster than the immersion algorithm.

As mentioned in Section III, the immersion algorithm needs an initial sorting operation, which makes its overhead slightly larger than the toboggan algorithm. However, the main reason that the immersion algorithm is less efficient in terms of execution time, especially for large images, is probably due to

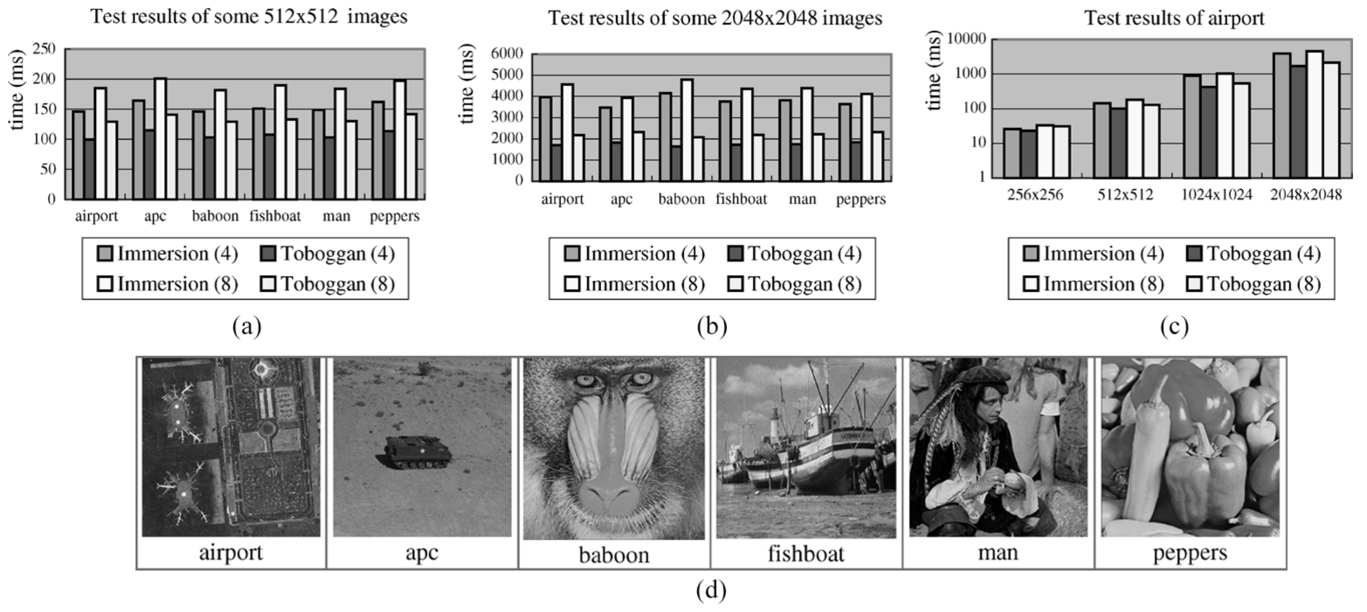


Fig. 7. Some experimental results of the order-invariant watershed segmentation. The test platform is a PC based on Intel Pentium III 1.0-GHz CPU with 256-MB RAM. The execution time is in milliseconds (ms). The test images are from the USC-SIPI image database.

the fact that the memory access pattern of the immersion algorithm is more distributed (or less localized) since it considers one level set per iteration. On the other hand, the toboggan algorithm tends to focus its operations within a nearby region at any time instant. This implies that less memory swapping is required for the toboggan algorithm in general. With the current CPU design, cache memory is much faster than the main memory, and, hence, the memory access pattern plays a very important role for the efficiency of the algorithm.

The results in Table II show that, when 8-connected neighborhood is used instead of 4-connected neighborhood, both algorithms become slower. The reason is the computation for each pixel is doubled. The efficiency differences between the two algorithms do not change much when the neighborhood system changes from 4-connectivity to 8-connectivity.

An interesting observation can be found from Table III. In this experiment, we do not apply the reconstruction preprocessing to the gradient images prior to the watershed segmentation. This makes the segmentation results contain more smaller fragments. In these cases, T_1 (the average execution time of immersion algorithm) of Table III is smaller slower than T_1 of Table I, but T_2 (the average execution time of toboggan algorithm) of Table III is faster greater than T_2 of Table I. This should be because when the fragments are smaller, the memory access pattern of the toboggan algorithm becomes even more localized. Meanwhile, more fragments bring more overhead to the immersion algorithm. Thus, the difference between the efficiencies of the two algorithms becomes larger.

VI. Conclusion

In this paper, we have presented two order-invariant watershed segmentation algorithms, one based on the immersion approach and the other on the toboggan approach. By introducing a special RIDGE label to achieve the property of order-invariance, we find that the two conceptually opposite approaches can indeed obtain the same segmentation result. The similarity and difference between the two algorithms are thoroughly analyzed and discussed in this paper. Four different watershed algorithms are compared by experiment, which are order-variant immersion (Vincent's algorithm [6]), order-variant toboggan [5], order-invariant immersion, and order-invariant toboggan. According to our experiments, the order-variant immersion algorithm is the slowest one, and order-variant toboggan algorithm is fastest. The two order-invariant algorithms produce the same results, but the order-invariant toboggan algorithm is faster in our experiments. Moreover, the toboggan algorithm requires less memory than the watershed algorithm, because the sliding list and the label image used in the toboggan algorithm can actually share memory (because they are used in different steps), while the immersion algorithm requires additional memory for storing the sorting results.

The source codes of the two algorithms are available in our website, http://ippr.csie.ntu.edu.tw, and it is recommended that the toboggan algorithm be used for most applications.

References

[1] J. Fairfield, "Toboggan contrast enhancement for contrast segmentation," in *Proc. IEEE 10th Int. Conf. Pattern Recognition*, vol. 1, 1990, pp. 712–716.
[2] J. M. Gauch, "Image segmentation and analysis via multiscale gradient watershed hierarchies," *IEEE Trans. Image Process.*, vol. 8, no. 1, pp. 69–79, Jan. 1999.
[3] X. Yao and Y.-P. Hung, "Fast image segmentation by sliding in the derivative terrain," in *Proc. SPIE Intelligent Robots and Computer Vision X: Algorithms and Techniques*, vol. 1607, 1991, pp. 369–379.
[4] A. Moga, B. Cramariuc, and M. Gabbouj, "An efficient watershed segmentation algorithm suitable for parallel implementation," in *Proc. IEEE Int. Conf. Image Processing*, vol. 2, 1995, pp. 101–104.
[5] E. N. Mortensen and W. A. Barrett, "Toboggan-based intelligent scissors with a four-parameter edge model," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 1999, pp. 452–458.
[6] L. Vincent and P. Soille, "Watersheds in digital spaces: an efficient algorithm based on immersion simulations," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, no. 6, pp. 583–598, Jun. 1991.

[7] L. Vincent, "Morphological grayscale reconstruction in image analysis: applications and efficient algorithms," *IEEE Trans. Image Process.*, vol. 2, no. 2, pp. 176–201, Feb. 1993.

[8] D. Wang, "Unsupervised video segmentation based on watersheds and temporal tracking," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 8, no. 5, pp. 539–546, May 1998.

[9] K. Haris, S. N. Efstraiadis, N. Maglaveras, and A. K. Katsaggelos, "Hybrid image segmentation using watersheds and fast region merging," *IEEE Trans. Image Process.*, vol. 7, no. 12, pp. 1684–1699, Dec. 1998.

[10] L. Najman and M. Schmitt, "Geodesic saliency of watershed contours and hierarchical segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 18, no. 12, pp. 1163–1173, Dec. 1996.

[11] S. Beucher, "Watersheds of functions and picture segmentation," in *Proc. IEEE Int. Conf. Acoustic, Speech, Signal Processing*, 1982, pp. 1928–1931.

[12] M. Couprie and G. Bertrand, "Topological grayscale watershed transformation," in *Proc. SPIE Vision Geometry V*, vol. 3168, 1997, pp. 136–146.

[13] C. Riddell, P. Brigger, R. E. Carson, and S. L. Bacharach, "The watershed algorithm: a method to segment noisy PET transmission images," *IEEE Trans. Nucl. Sci.*, vol. 46, no. 3, pp. 713–719, Mar. 1999.

[14] S. Ghosh, O. Beuf, M. Ries, N. E. Lane, L. S. Steinbach, T. M. Link, and S. Majumdar, "Watershed segmentation of high resolution magnetic resonance images of articular cartilage of the knee," in *Proc. IEEE Conf. EMBS*, vol. 4, 2000, pp. 3174–3176.

[15] M. W. Hansen and W. E. Higgins, "Watershed-based maximum-homogeneity filtering," *IEEE Trans. Image Process.*, vol. 8, no. 7, pp. 982–988, Jul. 1999.

Yung-Chieh Lin received the B.S. and M.S. degrees in computer science and information engineering from the National Taiwan University, Taipei, Taiwan, R.O.C., in 1998 and 2000, respectively.

From 1998 to 2000, he was with the Intelligent System Laboratory, Institute of Information Science, Academia Sinica, Taipei, where he did research on video object segmentation algorithms. Since 2000, he has been a Cofounder and Chief Technology Officer of IAdea Corporation, Taipei, a software company specializing in embedded system development. His research interest is in the area of computer vision and image processing.

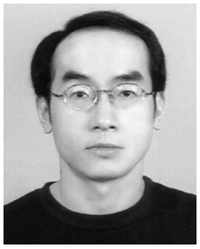
Yu-Pao Tsai received the B.Sc. degree in computer science from the National Chengchi University, Taiwan, R.O.C., in 1993, and the M.Sc. degree in computer and information Science from the National Chiao-Tung University, Hsinchu, Taiwan, in 1999, where he is currently pursuing the Ph.D. degree in computer and information science.

His current research interests include image-based rendering, video object segmentation, and virtual reality.

Yi-Ping Hung received the B.Sc. degree in electrical engineering from the National Taiwan University, Taipei, Taiwan, R.O.C., in 1982, the M.Sc. degree from the Division of Applied Mathematics, Brown University, Providence, RI, and the M.Sc. and the Ph.D. degrees from the Division of Engineering, Brown University, in 1987, 1988, and 1990, respectively.

He is currently a Professor with the Graduate Institute of Networking and Multimedia and the Department of Computer Science and Information Engineering, National Taiwan University. From 1990 to 2002, he was with the Institute of Information Science, Academia Sinica, Taipei, first as an Associate Research Fellow, promoted to Tenured Research Fellow in 1997. He served as a Deputy Director of the Institute of Information Science from 1996 to 1997. His current research interests include computer vision, pattern recognition, image processing, virtual reality, multimedia, and human-computer interface.

Dr. Hung received the Young Researcher Publication Award from Academia Sinica in 1997.



Zen-Chung Shih was born on February 10, 1959, in Taipei, Taiwan, R.O.C. He received the B.S. degree in computer science from Chung-Yuan Christian University, Taiwan, in 1980, and the M.S. and Ph.D. degrees in computer science from the National Tsing Hua University, Taiwan, in 1982 and 1985, respectively.

Currently, he is a Professor with the Department of Computer and Information Science, National Chiao-Tung University, Hsinchu, Taiwan. His current research interests include procedural texture synthesis, nonphotorealistic rendering, global illumination, and virtual reality.