Available online at www.sciencedirect.com
SCIENCE DIRECT®
Signal Processing: *Image Communication* 21 (2006) 100–112
SIGNAL PROCESSING: IMAGE COMMUNICATION
www.elsevier.com/locate/image

Marker-based image segmentation relying on disjoint set union

Hai Gao[a], Weisi Lin[b,*], Ping Xue[a], Wan-Chi Siu[c]

[a]*School of EEE, Nanyang Technological University, Singapore 639798, Singapore*
[b]*Institute for Infocomm Research, 21 Heng Mui Keng Terrace, Singapore 119613, Singapore*
[c]*Centre for Multimedia Signal Processing, Department of EIE, Hong Kong Polytechnic University, Hong Kong*

Received 7 February 2005; received in revised form 22 June 2005; accepted 29 June 2005

Abstract

Marker-based image segmentation has been widely used in image analysis and understanding. The well-known Meyer's marker-based watershed algorithm by immersion is realized using the hierarchical circular queues. A new marker-based segmentation algorithm relying on disjoint set union is proposed in this paper. It consists of three steps, namely: pixel sorting, set union, and pixel resolving. The memory requirement for the proposed algorithm is fixed as $2 \times N$ integers (N is the image size), whereas the memory requirement for Meyer's algorithm is image dependent. The advantage of the proposed algorithm lies at its regularity and simplicity in software/firmware/hardware implementation.

Keywords: Marker-based image segmentation; Disjoint set union; Union find

1. Introduction

Image segmentation is an important task in many image-processing applications, such as content-based retrieval, content-based compression, watermarking, remote sensing, and medical image analysis. Generally speaking, *image segmentation* is the process of isolating objects in the image from the background, i.e., partitioning the image into disjoint regions, such that each region is homogeneous with respect to some property, such as gray-level value or texture [8].

The watershed is one of the major tools for image segmentation in the field of mathematical morphology [2,3,10,11,13,14,19,21,26]. The watershed algorithms can be mainly classified into two categories [19]: watershed by immersion and watershed by topographical distance. The most well-known watershed algorithm by immersion was presented by [26], whereas the most well-known algorithm by topographical distance was presented by Meyer [11]; [13]. Apart from Meyer's algorithm by topographical distance, there were also some other algorithms by topographical

*Corresponding author.
E-mail address: weisi@ieee.org (W. Lin).

0923-5965/$ - see front matter © 2005 Published by Elsevier B.V.
doi:10.1016/j.image.2005.06.008

distance presented [2,3,10,14,19]. Disjoint set union has been used in watershed by topographic distance [19]. Note that in the algorithms by topographical distance special attention should be paid to the plateaus [19], i.e. regions of constant gray value. The above-mentioned two categories of algorithms are based on different definitions and are different approaches. Generally speaking [20], algorithms by immersion are more computationally efficient than algorithms by topographical distance, and are therefore more widely used [6,7,12,15–18,20,25,26].

Usually, the watershed algorithms (either by immersion or by topographic distance) are not applied to the original image, but to its gradient image [11]. The inherent problem with the watershed is that it often results in severe over-segmentation, even if appropriate filters are used for the original image or for the gradient image before the watershed operation is performed. This is due to the fact that the gradient image exhibits too many minima. To avoid over-segmentation, a very powerful method was introduced by Meyer [11]. Meyer assumes markers are available (either determined through user interactions or extracted automatically). The idea is to modify the gradient image according to the markers (to filter out the undesired minima of the gradient image) [22], before the watershed operation is performed.

Meyer further proposed an efficient marker-based watershed algorithm by immersion [12] relying on the hierarchical queues. Instead of filtering out the undesired minima of the gradient image, this algorithm suppresses unwanted minima (consequently, unwanted regions) during the algorithm itself. Meyer's algorithm [12] was further investigated by Salembier et al. [6,20,21], and Dong et al. [18]. Many robust methods for automatic marker extraction have been proposed for different kinds of images, such as textured images [25], infrared images [17], and color images [6,7], and made the marker-based watershed by immersion more widely adopted for image segmentation in recent years [6,7,15–18,20,25].

However, Meyer's marker-based watershed algorithm by immersion uses the data structure of hierarchical queues[1], and it is difficult to reserve an appropriate number of places for the hierarchical queues. Insufficient memory reservation for the hierarchical queues results in overflow, and too large memory reservation affects the performance and applications of the algorithm.

In this paper, we have proposed a new algorithm that relies on disjoint set union. The memory requirement for the proposed algorithm is fixed as $2 \times N$ integers, whereas the memory requirement for Meyer's algorithm is image dependent. The proposed algorithm is attractive in software/firmware/hardware implementation, due to its regularity and simplicity for memory allocation.

The rest of this paper is organized as follows: Section 2 gives a brief introduction on the general principles of the watershed algorithms by immersion and the Meyer's algorithm which depends on the hierarchical queues. Section 3 proposes a new marker-based watershed algorithm by immersion, based on disjoint set union. Section 4 analyzes the memory requirement, while Section 5 presents the experimental results and compares the proposed algorithm with Meyer's algorithm. Section 6 concludes the paper.

2. Watershed by immersion

2.1. Notions and definitions

Minimum [26]: a minimum refers to a connected component in which the pixels are with the same value and the value of this component is strictly lower than the values of its neighboring pixels.

Markers [22]*:* markers are connected sets of pixels included in region, and each marker indicates the presence of an object, either a foreground or a background object.

Gradient image: a gradient image is of the same size of the original image, while its pixels are gradients represented by integer gray-level values. For an 8-bit representation, there are 256 gray-level values (i.e., 0,1,2,…,255) in a gradient image.

[1] *The hierarchical queues* [6,12,18,20] are a set of queues with different priorities, and each queue is a first-in-first-out (FIFO) data structure.

0 means black,255 means white

Marker image: a marker image [22] is a two-level image with the object interiors (markers) being set to a constant c strictly less than any gray-level value of the gradient image and non-marker pixels being set to another constant a not less than any gray-level value of the gradient image, i.e., c is a negative value and a is 255 if the gradient image is of 8-bit representation.

2.2. Principles

A gradient image g can be considered as a topographic surface (Fig. 1(a)). Imagine that we pierce each minimum of the topographic surface and then we plunge this surface into a lake at a constant vertical speed. The water entering through the holes floods the surface, and two or more floods coming from different minima may merge. In order to avoid this, dams are built up at potentially merging points of the surface. At the end, only dams emerge, and the resultant dams correspond to the *watersheds* of the gradient image, or in other words, correspond to the object boundaries (see Fig. 1(b)).

2.3. Avoidance of over-segmentation

Fig. 2(a) illustrates an example of the gradient image g, the marker image m (with markers being set to a negative value c). Meyer's ideal [11,22] to avoid over-segmentation is to modify the gradient image g according to the marker image m and get the final modified gradient image s as shown in Fig. 2(b) and Fig. 2(c). Any watershed algorithms (either by immersion or by topographic distance) performed on the final modified gradient image s will get a desired segmentation.

2.4. Meyer's algorithm

Meyer proposed a direct marker-based watershed algorithm by immersion [12], depending on the hierarchical queues. In this algorithm, the flooding process is performed directly on the marker-modified gradient image i instead of the final modified gradient image s as shown in Fig. 2, and unwanted minima (consequently, unwanted regions) are suppressed during the algorithm itself.

The priority of a pixel of the modified gradient image is defined as the reciprocal of its gray-level value. This implies that a high (low) priority is assigned to a pixel with low (high) gray-level value. With such a convention, Meyer's algorithm can be summarized as follows:

Step 1: labeling of markers and initialization of the hierarchical circular queues.

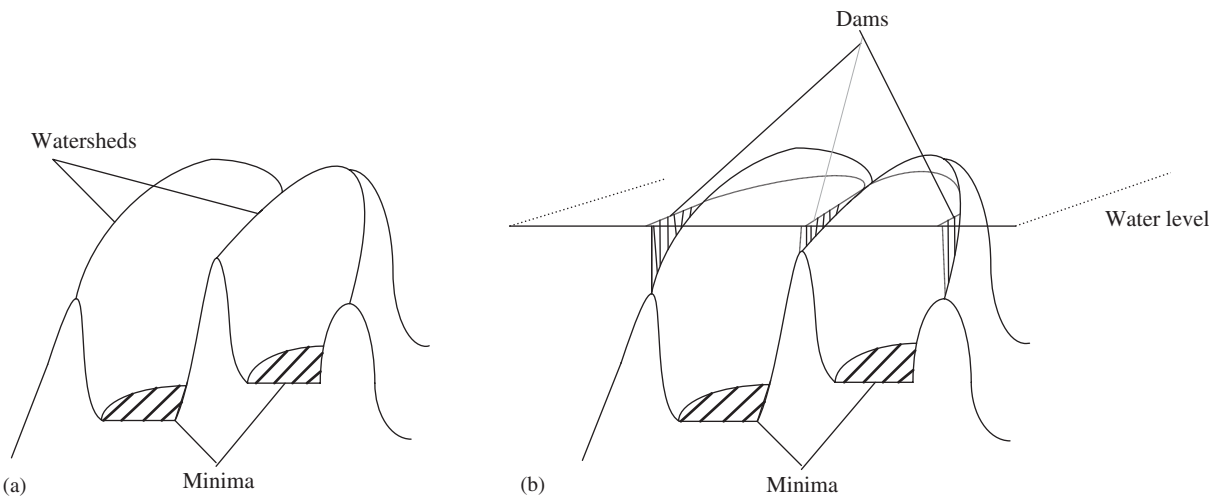

Fig. 1. Principles of the watershed algorithm by immersion: (a) considering a gradient image g as a topographic surface; (b) building dams at the places where the water coming from two different minima would merge.

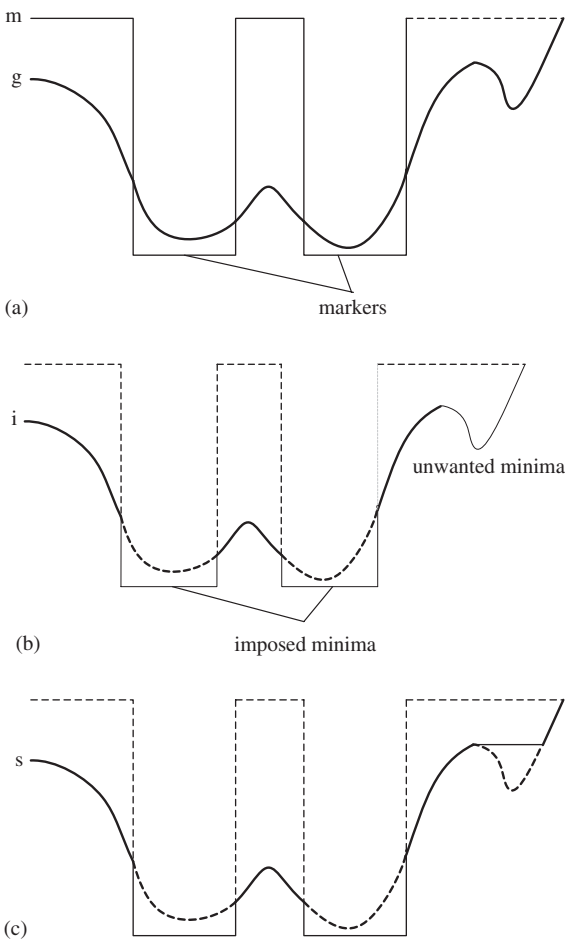

Fig. 2. One-dimensional illustration of marker-based gradient modification: (a) the gradient image g and the marker image m; (b) impose markers on the gradient image g as minima and get the *marker-modified gradient image* i; (c) suppress all unwanted minima and get the *final modified gradient image* s.

Step 2: flooding (also called region growing). The flooding step repeats the following steps until the hierarchical circular queues are empty: (a) extracts a pixel p from the highest priority end of the hierarchical circular queues and give p a label; (b) the neighbor pixels of p that have not been labeled and are still outside the hierarchical queues are put into the hierarchical queues with pixel priorities no higher than the pixel priority of p.

As described in step (b), if the gray-level value of an unlabeled neighbor is lower than the gray-level value of p, the pixel priority of this neighbor is decreased to the pixel priority of pixel p. Thus unwanted regions are suppressed. This technique is further clarified in Dong's paper [18], and is called the filling operation [18], as shown in Fig. 3.

3. Proposed algorithm with disjoint set union

3.1. Disjoint sets

Disjoint sets [5,24] refer to a collection of mutually exclusive sets, and can be represented as rooted trees [9]. Each tree corresponds to a set, and has one and only one root to uniquely identify the set. Nodes of a tree correspond to elements of the associated set, and each node except the root of the tree has a pointer to its parent node.

3.2. Overview of the proposed algorithm

The disjoint set union can be applied to the marker-based watershed by immersion. In this paper, the elements (nodes) of the disjoint sets are pixels which are presented using their location indices p ($p = \text{image\_width} * y + x$), where x and y are pixel coordinates. For an image with N pixels, the disjoint sets can be implemented in an integer array, named *parent*, with the same size as the image, and *parent*[p] stores the parent pixel of pixel p. In the case that p has no parent (i.e., p being the root of a tree), *parent*[p] is set to a special negative value.

Like Meyer's algorithm relying on the hierarchical queues, the proposed algorithm is performed directly on the marker-modified gradient image i (as exemplified in Fig. 2(b)) instead of on the final modified gradient image s (as exemplified in Fig. 2(c)), and it suppresses unwanted minima (consequently, unwanted regions) during the algorithm itself. The pixels in image i are processed in an increasing order of gray-level value. When a pixel p is being processed, a new singleton set containing pixel p is first created. Since pixel p has no parent (i.e., p being the root of the singleton set), *parent*[p] is originally set to a special negative integer (*parent*[p] is originally set to -2 if p is a pixel belonging to a marker, otherwise *parent*[p] is

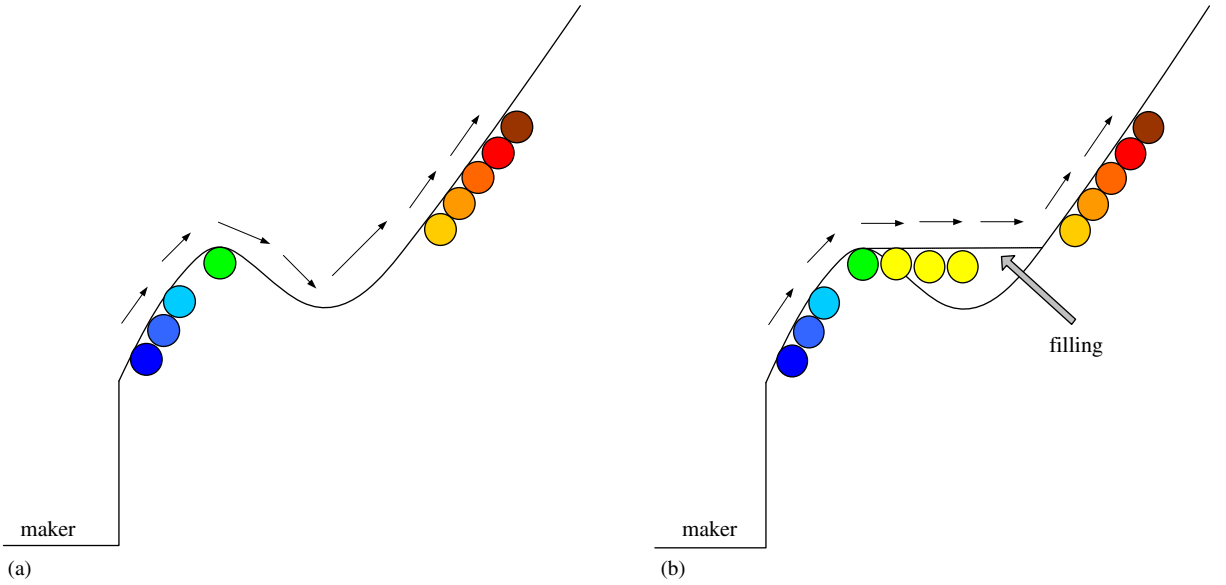

Fig. 3. Illustration of the filling operation: (a) the *marker-modified gradient image i* with unwanted minima; (b) the filling operation during flooding.

originally set to -1). After that, we check the neighbors of pixel p to see if there are some already processed neighbors (i.e., if there are some neighboring sets). If yes, the singleton set will be combined recursively with its neighboring sets if a certain set union criterion is met.

Considering two sets with roots pixel r and pixel p are to be combined, suppose pixel p is made as the root of the newly combined set. This combination is achieved in two steps. In the first step, check whether *parent*[p] needs to be changed or not, and change *parent*[p] if necessary. As pixel p and pixel r are the roots of the two original sets, *parent*[p] and *parent*[r] are originally all negative integers, more specifically, *parent*[p] and *parent*[r] are originally -2 or -1 (-2 implies the associated original set is grown from a marker or a marker pixel, -1 implies the associated original set is not grown from a marker). In the only case that the original set containing pixel p is not a set grown from a marker (or a marker pixel) but the original set containing pixel r is a set grown from a marker (or a marker pixel), the newly combined set rooted with pixel p will become a set grown from a marker or a marker pixel, *parent*[p] should be changed. More specifically, in the only case that originally *parent*[p] is -1 and *parent*[r] is -2, *parent*[p] should be changed from -1 to -2; otherwise, *parent*[p] is kept unchanged. At the end of this step, *parent*[p] is -2 or -1 (-2 denotes that the newly combined set is a set grown from a marker or a marker pixel, -1 denotes that the newly combined set is a set not grown from a marker or a marker pixel). In the second step, the two sets are finally combined by making r to be a child of pixel p (by setting *parent*[r] to p), in other words, pixel p is formally made as the root of the newly combined set. The new *parent*[r] becomes a positive integer, signaling the new status of pixel r (a non-root pixel) and its associated parent pixel.

After all the pixels have been processed, for every node p, *parent*[p] will have a value with practical meaning. For a non-root node p (pixel p), *parent*[p] stores the parent pixel location index of p; for a root node p, *parent*[p] is a negative integer. By resolving the *parent* array, the specific set that a node (pixel) belongs to can easily be identified.

```
void MakeSet (int p)
{
   if pixel p is a pixel belonging to a marker:
         parent[p]=-2;
   else
         parent[p]=-1;
}
```

Fig. 4. The pseudo-codes for the *MakeSet* operation that is used in the proposed algorithm.

```
int Find(int p)
{
    if (parent[p]>=0)
    {   parent[p]=Find(parent[p]);
        return parent[p];
    }
    else return p;
}
```

Fig. 5. The pseudo-codes for the *Find* operation that is used in the proposed algorithm.

3.3. Basic operations

The basic operations in disjoint set union for the proposed segmentation algorithm are listed here:

(1) *MakeSet*(p): to create a new set containing a single element (pixel), p, which does not exist in any set previously, by setting *parent*[p] to a special negative value as illustrated in Fig. 4. In this paper, *parent*[p] is originally set to -2 if p is a pixel belonging to a marker, otherwise *parent*[p] is originally set to -1.
(2) *Find*(p): to return the root of the tree containing pixel p. This operation starts from node p and follows the pointer to the parent until the tree root is reached, as illustrated in Fig. 5. The *Find* is a recursive operation and it makes every encountered node pointing directly to the root. This technique is called the path compression [5,9,24].

```
void Union (int p, int r)
{
   if (parent[p]==-1 && parent[r]==-2)parent[p]=-2;
   parent[r]=p;
}
```

Fig. 6. The pseudo-codes for the *Union* operation that is used in the proposed algorithm.

(3) *Union*(p, r): to combine the two sets whose roots are pixel p and pixel r, respectively into a new set whose root is p as illustrated in Fig. 6, the resultant *parent*[p] is a negative integer signaling whether the newly combined set is grown from a marker or not.

3.4. Description of the proposed algorithm

There are three steps in the proposed method, namely, pixel sorting, set union, and pixel resolving, as will be explained in the rest of this section.

3.4.1. The first step: pixel sorting

Pixels of the marker-modified gradient image i are firstly sorted according to the increasing order of their gray-level values. An extra integer array S of length N (N is the image size) is used to store the sorting results. For pixel sorting, the gray-level histogram of marker-modified gradient image[2] i is firstly obtained, and space for each gray level is reserved in S accordingly. Each pixel of the image is put into S with respect to its gray-level value[3], in a raster scanning (from left to right and from top to bottom).

3.4.2. The second step: set union

Pixels are then processed one by one according to the order in the array S. Note that all the pixels belonging to markers in i are put into the front of S and they will be processed before all the non-marker pixels.

[2]There are totally 257 gray-levels, i.e., c, 0, 1, 2, …, 255 (where c is a negative value), in i.

[3]Note that all the pixels belonging to markers in i have a negative gray-level value c and therefore they are put into the front of S.

For each pixel p extracted from array S, do the following:

(1) Create a new singleton set containing pixel p by calling *MakeSet(p)* function to set *parent*[p] to –2 if p is a pixel belonging to a marker or -1 if p is a non-marker pixel.
(2) Check all the neighboring pixels of pixel p to see whether they are processed before pixel p (to see whether they appears earlier in array S than p). For each neighboring pixel p_0 already processed, do the following:
   (a) Call *Find*(p_0) to find the root r of the set containing pixel p_0 .
   (b) If the set containing pixel p_0 and the set containing the current pixel p are not the same one[4], combine these two sets into one set if the following *set union criterion* is met.

Set union criterion: As mentioned above, the marker pixels are processed before all the non-marker pixels are being processed. During the processing of the marker pixels, the two sets are combined without any constrain; whilst during the processing of the non-marker pixels, the two sets are combined if and only if at least one of them are not grown from a marker.

Under such set union criterion, at the end of the processing of all the marker pixels, each set corresponds to a marker; and after all the pixels are processed, each resultant set is grown from a marker and corresponds to a region in the image. Those sets grown from unwanted minima during the processing of the non-marker pixels are combined with some set grown from a marker sooner or later, thus the proposed algorithm suppresses unwanted minima (consequently, unwanted regions).

[4]Each tree (set) has one and only one root to uniquely identify the set. If the set containing pixel p_0 and the set containing the current pixel p are the same set, *Find*(p_0) and *Find*(p) returns the same root pixel. Since the current pixel p is the root of the set containing pixel p, to see whether the set containing pixel p_0 and the set containing the current pixel p are the same set, we only need to see whether pixel r and pixel p are the same pixel.

As shown in Fig. 6, when the set containing current pixel p is combined with a neighboring set containing pixel p_0 (as aforementioned, p_0 is an already processed neighbor of p), the pixel p is always made as the root of the newly combined set. No matter how many already processed neighbors p_0 there are, (i.e., no matter how many set combinations are performed as the current pixel p is being processed), the root of the set containing pixel p must be p. Hence when the set containing current pixel p is combined with a neighboring set containing pixel p_0, we only need to find out the root r of the set containing pixel p_0, and then make r be a child of pixel p. And furthermore, *parent*[p] is reset to -2 to if *parent*[p] or *parent*[r] is -2 (i.e., if one of the two sets is grown from a marker), thus we can always identify whether a set is grown from a marker or not.

Since pixels are processed according to the order as they are stored in the array S, the current pixel p appears later than any other already processed pixel p_0 in the array S, so the *Union* operation defined in Fig. 6 also implies that any parent will appear later than all its children in the array S after processing. This simplifies the pixel-resolving step.

3.4.3. The third step: pixel resolving

A resolving step is carried out after the set union step to obtain the segmentation result (in a form of object membership map). The pixels are visited in the reverse processing order of the set union step, thus parents are always visited before their children. A pixel to be visited is either a root with no parent or a pixel whose parent has already been resolved. For each pixel p, if *parent*[p] is negative (p has no parent), a new region is encountered; if *parent*[p] is non-negative (pixel p has a parent, i.e., pixel *parent*[p], which must have already been resolved), pixel p is assigned to the same region as pixel *parent*[p]. No extra memory is needed because the array *parent* can also be used to store the segmentation results (more exactly, for each pixel p, p is assigned to the mth region by setting *parent*[p] to m).

The pseudo-codes for the proposed marker-driven segmentation algorithm are listed in Fig. 7.

```
/* Array S (of size N) stores the sorted pixels (pixel location indices) of the
marker-modified gradient image i. */

//the set union step
for (k=0; k<N; k++)
{  p=S[k];
   MakeSet (p);
   if(parent[p]==-2)
   {   for each processed p0, p0 is a neighbor of p
       {  r=Find(p0);
          if (r!=p) Union(p,r);
       }
   }
   else
   {
      for each processed p0, p0 is a neighbor of p
      {
          r=Find(p0);
          if (r!=p)
          {
               if ( parent[r]==-1 || parent[p]==-1 ) Union(p,r);
          }
      }
   }
}

//The resolving step
int m=1;
for (k=N-1; k>=0; k--)
{  p=S[k];
   if (parent[p]<0)
   {   parent[p]=m;
       m=m+1;
   }
   else parent[p]=parent[parent[p]];
}
```

Fig. 7. The pseudo-codes for the proposed marker-based segmentation algorithm with disjoint set union.

4. Memory requirement

The total memory requirement for the proposed algorithm based on disjoint set union is only $2 \times N$ integers since the algorithm just needs two integer arrays (S and *parent*) of size N each.

In Meyer's algorithm [12], the hierarchical queues are created with as many levels of priority as there are gray-level values in the marker-modified gradient image i, i.e., a set of 257 queues is used for an 8-bit representation of the gradient image. To allocate memory for each queue, one possible way is dynamic allocation, more specifically, allocating memory for each queue according to the histogram of the marker-modified gradient image i. However, as explained in Section 2.4, in order to suppress unwanted regions, the gray level values of some pixels in the marker-modified gradient image i are adjusted during the flooding process. Thus it is not possible to foresee the largest number of pixels in a queue for a specific image. Dynamic allocation works in many applications because usually all the pixels of the image are not in the queue at the same time. However, it is quite possible that in some applications some queue (or queues) will become full during the flooding process with dynamic allocation. So additional measures must be taken to avoid any queue being full. More specifically, if a queue is full, an additional queue must be created and the priority of the additional queue is set as the same as the full queue. Just because dynamic allocation is complicated, in Meyer's marker-based watershed algorithm by immersion [12], including some further investigations on this algorithm [6,18,20], a fixed number of h places was reserved for each queue. Since h is an image-dependent parameter, the total memory requirement for Meyer's marker-based watershed algorithm based on the hierarchical queues is image dependent.

5. Experimental results

In the proposed algorithm, each pixel is assigned to a region and no watershed lines are explicitly constructed. To have a better visual view of the segmentation results, we highlight the object boundaries in this section according to a simple procedure as shown in Fig. 8. Note that the boundary highlighting is not an indispensable step for segmentation tasks.

```
/* Array parent stores the segmentation results in a form of object
membership map. */

for(p=0;p<N;p++)
{
   for every p0, p0 is a neighbor of p
   {
      if (parent[p]>parent[p0]), highlight p;
   }
}
```

Fig. 8. The pseudo-codes for bounday highlighting.

5.1. Marker-based image segmentation

Figs. 9 and 10 give two examples of marker-based watershed segmentation for medical images (images *Cancer 1 and Cancer 2*) in which markers were determined through user interactions (see Fig. 9(b) and Fig. 10(b), red rectangles indicating marker pixels). As can be seen from Fig. 9(c) and Fig. 10(c), the regions with cancer are segmented (identified) successfully. This segmentation provides helpful information to surgical treatments.

Figs. 11 and 12 give more segmentation examples with boundaries being highlighted. In Fig. 11(b) for the *Akiyo* image and Fig. 12(b) for the *Goldfish* image, there are three markers. In Fig. 11(c), we successfully segmented the foreground from the background; and in Fig. 12(c), we successfully segmented the object of interest (the dominant fish in the image) from the background.

In the above examples, each marker is a hollow rectangle (one-pixel wide). In fact, according to the definition of marker, a marker can be of any shape and of any size, as long as it is a connected set of pixels. Markers can also be extracted automatically [6,7,16,17,20,25], thus marker-based segmentation algorithm can be used for unsupervised segmentation tasks.

The final segmentation is mainly decided by the markers (the marker image) for marker-based segmentation. Different number of markers or different shape of markers or different size of markers will lead to different segmentation. There might be some slight difference in the object

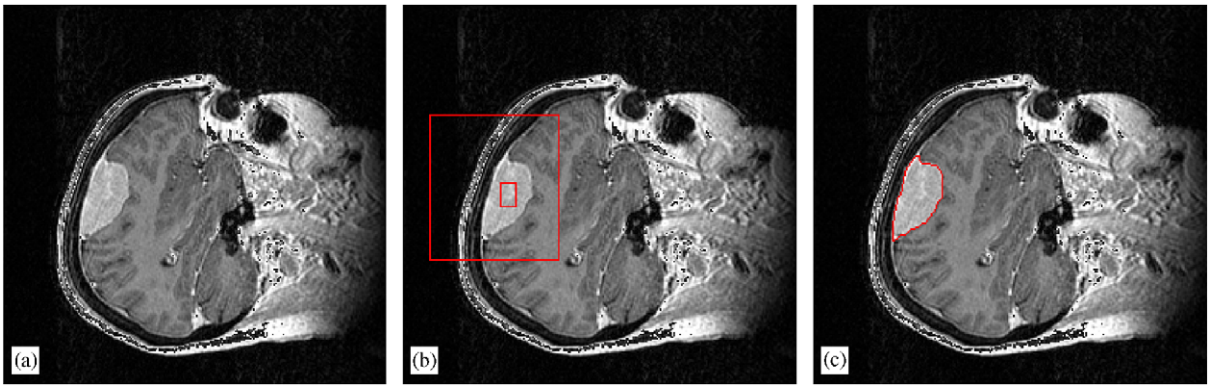

Fig. 9. Marker-based watershed segmentation of *Cancer 1 image*: (a) original cancer image 1; (b) markers; (c) segmentation result with boundaries being highlighted.

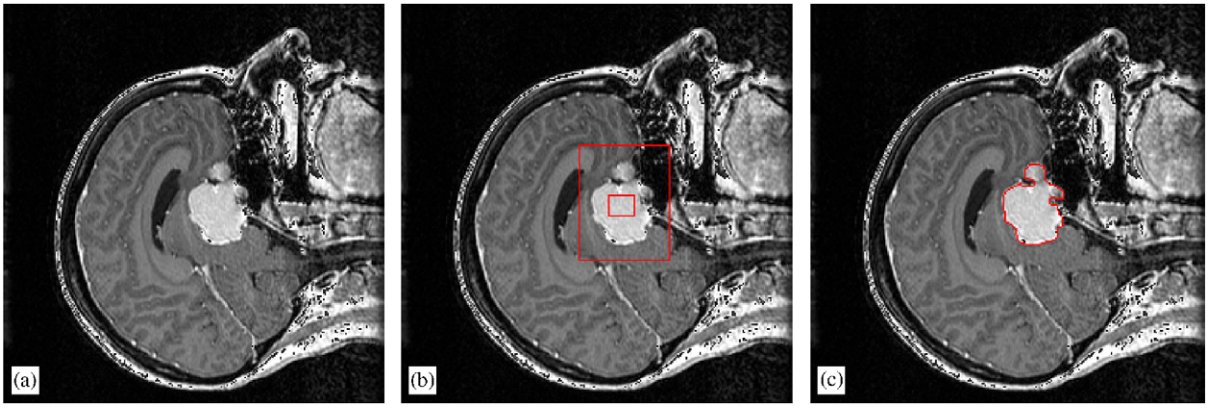

Fig. 10. Marker-based watershed segmentation of *Cancer 2 image*: (a) original cancer image 2; (b) markers; (c) segmentation result with boundaries being highlighted.

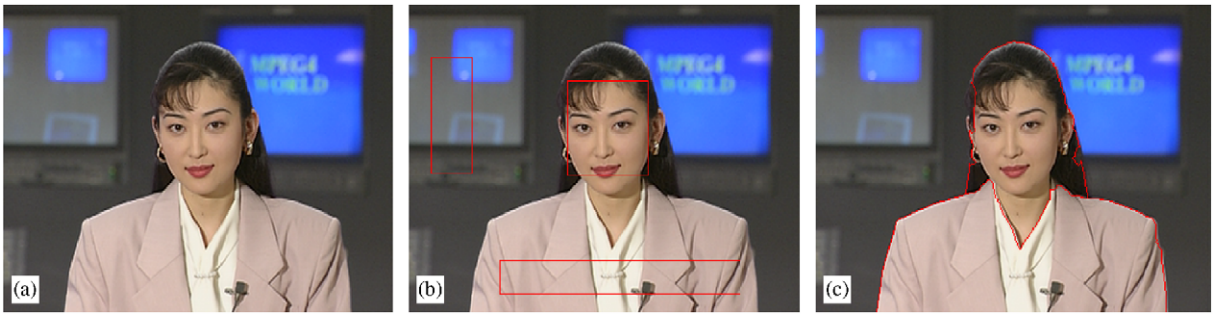

Fig. 11. Marker-based watershed segmentation of the *Akiyo*: (a) the *Akiyo* image; (b) markers; (c) segmentation result with boundaries being highlighted.

Fig. 12. Marker-based watershed segmentation of the *Goldfish* image: (a) the *Goldfish* image; (b) markers; (c) segmentation result with boundaries being highlighted.

Table 1
Timing results for the proposed algorithm

Image name (and image size N)	$T(\times 10^{-3}\,\mathrm{s})$
Cancer 1 (256×256)	42
Cancer 2 (256×256)	43
Akiyo (352×288)	57
Goldfish (352×288)	59

boundaries when comparing the segmentation results of two algorithms. Here we give an explanation. In the case that there exist a block of pixels which have the same gray level value in the marker-modified gradient image, these pixels have no difference in terms of pixel priorities in both algorithms. However, these pixels might be processed in different order in the two algorithms. In the proposed algorithm, the pixels are processed in scanning order because they are stored in that order. On the contrary, in Meyer's algorithm, pixels are put into the queue by depth-first search and are processed in the order as they are in the queue. The difference in processing order makes no difference in segmentation results if the block of pixels is the interior of an object, because the whole block will become a part of a set sooner or later. But, if the block of pixels is object boundaries, there might be some slight difference in the object boundaries when comparing the segmentation results of the two algorithms.

5.2. Timing results

The processing time for the proposed algorithm is mainly related to the image size N. Table 1 lists the timing results for the proposed algorithm based on disjoint set union. The tests have been conducted with C-programming in a Pentium M 725 processor (1.6 GHz) with 512 Mb of RAM.

However, the processing time for Meyer's algorithm is not only related to the image size N, but also related to the memory used for the hierarchical queues. Table 2 lists the timing results for Meyer's algorithm based on the hierarchical queues when different sizes of queues are used.

Allocating more memory for each queue leads to a slower algorithm, because memory allocation on the heap involves additional overheads, both in time and in space [4]. For instance, to allocate memory for a queue, we call *malloc( )* which requests a block of memory from the pool. The pool is searched for a block of memory large enough to satisfy the request. This is done by checking a map or directory of some sort that shows which blocks are currently in use and which are available. It may take several tries so it might not be deterministic—that is, we can't count on *malloc( )* always taking exactly the same amount of time. Before a pointer to that block is returned, the size and location of the block must be recorded, so further calls to *malloc( )* would not use it and when we call *free( )* the system knows how much memory to release. Under such mechanism, generally speaking, allocating more memory for each queue means more tries to find an appropriate block of memory for that queue, and more time. In Table 2, we give a range of processing time because each run takes different time as explained above.

Table 2
Timing results for Meyer's algorithm

Image name (and image size N)	$T(\times 10^{-3}$ s)						
	($h = N/8$)	($h = N/16$)	($h = N/32$)	($h = N/48$)	($h = N/64$)	($h = N/96$)	($h = N/128$)
Cancer 1 (256×256)	60–70	50–60	40–50	40	Queue full		
Cancer 2 (256×256)	60–70	50–60	40–50	40	Queue full		
Akiyo (352×288)	70–80	60–70	50–60	50–60	50	50	Queue full
Goldfish (352×288)	70–80	60–70	50–60	50–60	50	50	Queue full

The minimum sizes of each queue for the above image are $N/58$, $N/46$, $N/125$, and $N/121$, respectively.

As we have discussed in Section 4, a fixed number of places are usually reserved for each queue in Meyer's algorithm. With the comparison of Tables 1 and 2, Meyer's algorithm may perform slightly better, if *just-sufficient* memory places are reserved for the hierarchical queues. However, if the memory allocation is not *just sufficient*, Meyer's algorithm either takes longer processing time (when a queue is too big) or runs into the *full queue* situation (when a queue is too small). With real-world images, it is difficult to decide the *just-sufficient* memory allocation. As can be seen in Table 1, the proposed algorithm overcomes this disadvantage since its memory requirement is deterministic, regardless of image contents, and this image-independent nature can bring about convenience in programming, and more importantly, cost reduction (in terms of architecture complexity, device size, power dissipation, etc.) for hardware implementation.

6. Conclusions

We have proposed a new marker-driven watershed algorithm based on disjoint set union in this paper. Due to the suppression of unwanted regions, appropriate memory allocation cannot be guaranteed for Meyer's algorithm even dynamic memory allocation is employed, because its memory requirement is image dependent. The advantage of the proposed algorithm is its regularity and deterministic nature in memory requirement (with a fixed allocation of $2 \times N$ integers, where N is the image size); therefore the proposed algorithm is simpler in software/firmware/hardware implementations.

References

[2] A. Bieniek, A.N. Moga, A connected component approach to the watershed segmentation, in: J. Gautsias, L.M. Vincent, D.S. Bloomberg (Eds.), Mathematical Morphology and Its Applications to Image and Signal Processing, Springer, 2000, pp. 215–222.
[3] A. Bieniek, A.N. Moga, An efficient watershed algorithm based on connected components, Pattern Recognition 33 (2000) 907–916.
[4] B. Eckel, Thinking in C++, second ed., vol. 1, Prentice-Hall, Englewood Cliffs, NJ, 2000, pp. 547–580 (Chapter 1).
[5] Z. Galil, G.F. Italiano, Data structure and algorithms for disjoint set union problems, ACM Comput. Surveys 23 (3) (September 1991) 319–344.
[6] H. Gao, W.-C. Siu, C.-H. Hou, Improved techniques for automatic image segmentation, IEEE Trans. Circuits Systems Video Technol. 11 (12) (December 2001) 1273–1280.
[7] D. Gatica-Perez, C. Gu, M.-T. Sun, Semantic video object extraction using four-band watershed and partition lattice operators, IEEE Trans. Circuits Systems Video Technol. 11 (5) (May 2001) 603–618.
[8] R.M. Haralick, L.G. Shapiro, Survey: image segmentation techniques, Comput. Vision Graph. Image Process. 29 (1985) 100–132.
[9] J. Hopcroft, J.D. Ullman, Set-merging algorithms, SIAM J. Comput. 2 (December 1973) 294–303.
[10] A. Meijster, J.B.T.M. Roerdink, A disjoint set algorithm for the watershed transform, in: The 1998 European Signal Processing Conference (EUSIPCO' 98), vol. iii, 1998, pp. 1665–1668.
[11] F. Meyer, S. Beucher, Morphological segmentation, J. Visual Commun. Image Represent. 1 (1) (September 1990) 21–46.
[12] F. Meyer, Color image segmentation, in: IEE International Conference on Image Processing and its Applications, Netherlands, 1992, pp. 303–306.
[13] F. Meyer, Topographical distance and watershed lines, Signal Processing 38 (1994) 113–125.

[14] A.N. Moga, M. Gabbouj, Parallel image component labeling with watershed algorithm, IEEE Trans. Pattern Anal. Machine Intell. 19 (5) (1997) 441–450.

[15] A.N. Moga, M. Gabbouj, Parallel marker based image segmentation with watershed transformation, J. Parallel. Distributed Comput. 51 (1) (1998) 27–45.

[16] M. Naemura, A. Fukuda, Y. Mizutani, Y. Izumi, Y. Tanaka, K. Enami, Morphological segmentation of sport scenes using color information, IEEE Trans. Broadcasting 46 (3) (2000) 181–188.

[17] S.R. Neves, E.A.B. da Silva, G.V. Mendonca, Wavelet-watershed automatic infrared image segmentation method, Electron. Lett. 39 (12) (June 2003) 903–904.

[18] D.K. Park, H.S. Yoon, C.S. Won, Fast object tracking in digital video, IEEE Trans. Consumer Electron. 46 (3) (August 2000) 785–790.

[19] J.B.T.M. Roerdink, A. Meijster, The watershed transform: definitions, algorithms and parallelization strategies, Fundamenta Informaticae 41 (2001) 187–228.

[20] P. Salembier, M. Pardas, Hierarchical morphological segmentation for image sequence coding, IEEE Trans. Image Process. 3 (5) (September 1994) 639–651.

[21] P. Salembier, A. Oliveras, L. Garrido, Anti-extensive connected operators for image and sequences processing, IEEE Trans. Image Process. 7 (4) (April 1998) 555–570.

[22] J. Serra, L. Vincent, An overview of morphological filtering, Circuits Systems Signal Process. 11 (1) (1992) 47–108.

[24] R.E. Tarjan, Data Structures and Network Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983 (Chapter 2).

[25] A. Tremeau, P.R. Hill, C.N. Canagarajah, D.R. Bull, Image segmentation using a texture gradient based watershed algorithm, IEEE Trans. Image Process. 12 (12) (December 2003) 1618–1633.

[26] L. Vincent, P. Soille, Watersheds in digital spaces: an efficient algorithm based on immersion simulations, IEEE Trans. Pattern Anal. Machine Intell. 13 (6) (June 1991) 583–598.