The watershed transform in ITK - discussion and new developments

Richard Beare[1] and Gaëtan Lehmann[2]

June 9, 2006

[1]Department of Medicine, Monash University, Australia.
[2]Unité de Biologie du Développement et de la Reproduction, Institut National de la Recherche Agronomique, 78350 Jouy-en-Josas, France

Abstract

This report discusses various definitions and implementations of the watershed transform as well as providing an introduction to some well known techniques for applying the watershed transform to practical problems. The discussion will be focused on new and existing ITK classes.

Contents

1 Introduction

The watershed transform is a tool for image segmentation that is fast and flexible and potentially fairly parameter free. It was originally derived from a geophysical model of rain falling on a terrain and a variety of more formal definitions have been devised to allow development of practical algorithms. A brief discussion of some of these definitions and algorithms is provided in Section 3.

Section 4 provides an introduction to a methodology known as *marker based* watershed. Using markers in watershed transforms is a well established technique in the morphological image analysis community and has proved very useful for many applications. The aim of this part of the report is to describe the technique to a wider audience [1].

A brief description of two algorithms used to implement the watershed transform is given in Section 5.

Section 8 proposes some applications of watershed with ITK.

2 Image segmentation

Image segmentation, in the context of this report, may be broadly defined as finding the part(s) of the image that is of interest in order to make measurements. This is a basic definition of image analysis, which is different from image processing and computer vision[2].

It is very rare to achieve a useful segmentation using a single procedure, even under the most favourable conditions. Successful segmentation algorithms typically use a carefully constructed combination of procedures to achieve useful results. Some modern algorithms, especially the family of routines related to geodesic active contours, may seem to be an exception to this rule of thumb because they are often presented as achieving good segmentation results entirely on their own. However close examination will show

[1] The lack of a marker based watershed motivated this work on ITK filter classes.

[2] Image processing produces images, usually for human consumption, while computer vision aims to produce high level semantic descriptions of images.

that a variety of preprocessing steps in addition to careful parameter tuning is necessary to achieve anything useful.

These comments are also relevant to the watershed transform – it is very unlikely that any segmentation problem can be solved using the watershed alone. However it is a very useful and flexible tool that can be applied in a number of ways. This report will emphasise the marker based approach to using the watershed transform, but there are a number of others including watershed trees and region merging algorithms.

Finally, an important point that should be mentioned, is that the output of a low level segmentation procedure such as the watershed transform will typically be a *label image*. A label image, sometimes referred to as a categorical image, has unique values for each region. For example, if a watershed produces 2 regions, all pixels belonging to one region would have value A, and all belonging to the other might have value B. Unassigned pixels, such as watershed lines, might have value of 0.

3 The watershed transform – a brief history

The watershed transform was first proposed by Beucher and Lantuéjoul[4] as a geophysical model of rain falling on a terrain. The idea is that a raindrop falling on a surface will trickle down the path of steepest descent to a minimum. The set of points on the surface that lead to the same minimum is known as a catchment basin and borders between catchment basins are watershed lines[3]. If an image is considered as a terrain and divided into catchment basins then the hope is that each catchment basin would contain an object of interest. The image representing the terrain will be referred to as the *control surface* in this report.

Translating the model of rain falling on a terrain to an algorithm has a number of problems[4] which are discussed in Section **??**. The alternative is to imagine a flooding process, with each regional minima[5] becoming the source of a lake. The terrain is progressively flooded and lakes eventually meet neighboring lakes. Virtual dams are constructed to keep the neighboring lakes separate as the water level rises. When the image surface is completely flooded the virtual dams correspond to the watershed lines[6].

A major breakthrough in the implementation of the watershed was made by Vincent and Soille [18] with the introduction of the first queue based implementation of the watershed transform. This was especially significant because it allowed similar execution times on conventional computer systems as was being achieved with specialized parallel processing systems. The Vincent-Soille algorithm appears to be used in the matlab image processing toolbox watershed implementation (based on the observation that the library function has a `_vs` suffix).

Another major advance made soon after by Meyer and Beucher [5, 15]. Meyer defined the watershed transform in terms of *topographic distances* through *lower complete* images[7]. These definitions rigorously defined the behavior in the presence of plateaus and lead to some new algorithms based on graph theory. Two examples of these algorithms are *hill-climbing* and *image integration* and will be discussed in more detail in this report. Figures 1, 2 and 9 illustrate the possible problems with plateaus.

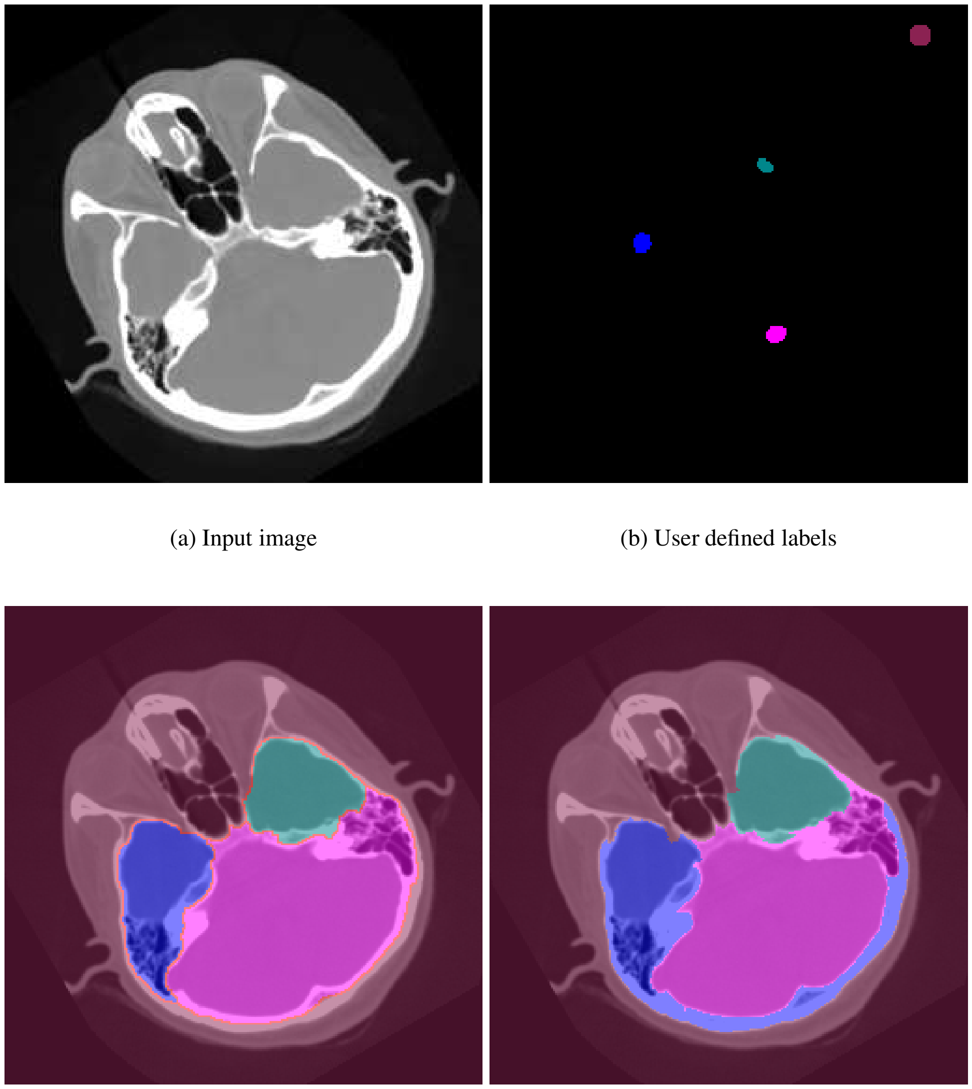
[3]Major geographical features, like rivers or oceans, are often considered as separated by watershed lines, usually along mountain ranges.

[4]Although this is what the first ITK implementation does.

[5]A regional minima is a set of pixels surrounded by pixels of higher value.

[6]In 3D neighboring lakes meet at surfaces.

[7]A lower complete image is an image where the only plateaus are regional minima. A common way to compute the lower completion of an image, is to add the distance from the nearest lower pixel to the pixels of all the plateaus. Lower completion have the property to produce the same image than the input image if the input image is already lower complete. By lower completing an image, we introduce a new order relation in the plateaus pixels which can be used to solve the plateaus problem with watershed.



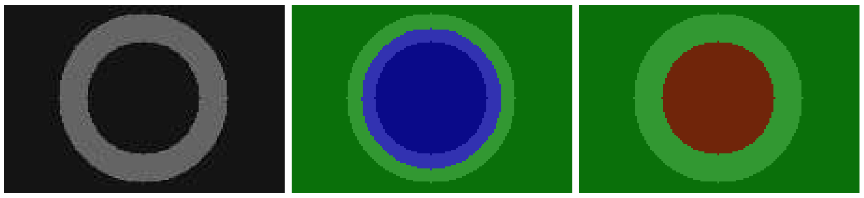
(a) Input image

(b) User defined labels

(c) Morphological watershed

(d) ITK watershed

Figure 1: Example of plateau problem. The blue marker is clearly extended where it should not with the ITK watershed. Labels are colorized and superimposed on the input image.



(a) Input image - bright ring on constant background

(b) Morphological watershed

(c) ITK watershed

Figure 2: Another example of plateau problem. The inside and outside of the ring have the same brightness, but the ring is arbitrarily assigned to the outer region by the ITK watershed. It is split by the morphological watershed.

Readers interested in more technical detail should refer to [17, 13].

An alternative approach, known as the topological watershed, has recently been developed [6]. The topological watershed modifies the topology of the surface so that all pixels that are not watershed lines become regional minima and watershed lines are modified so that they are more suitable for use in measuring saliency of boundaries between regions. Algorithms implementing the topological watershed also exploit priority queues, but don't appear to be able to avoid the need for explicit lower completion or reconstruction by erosion.

The topological watershed appears to have been developed with the aim of using measures of border significance for merging criteria. This work lead to the discovery that it wasn't possible to compute certain measures with the flooding approaches used by many implementations.

The original concept behind the watershed transform was rain falling on a terrain and flowing down paths of steepest descent to local minima. There has been a lot of work on alternatives to this approach because of the problems with it. These problems are basically to do with flat zones in an image which have no paths of steepest descent leading from them, which means that water would just sit in the one spot. A simple minded implementation of this idea would therefore produce a separate catchment basin for each pixel in the flat zone, which is not a very useful result. The ITK watershed uses a number of steps to avoid this problem:

1. Label all minima and flat zones.
2. Trace all unlabelled pixels to a labelled one using gradient descent.
3. Check each flat region and find the lowest neighbor. Change the label of the flat region to that of the lowest neighbor.

It should be evident that a number of arbitrary choices have been made here - for example it is impossible to break a flat region into pieces, which would probably better fit the physical reality of rainfall (consider a flat topped ridge which should perhaps be split down the middle). What happens of there are multiple neighboring regions with equal height - is the labelling decision now somewhat arbitrary?

In general the flooding models avoid these dilemmas and lead to more elegant algorithms.

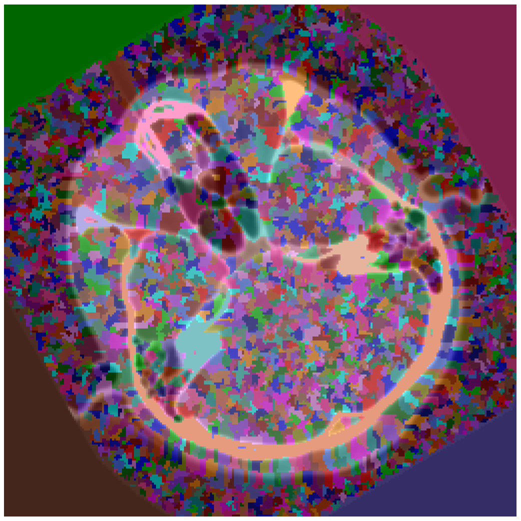


Figure 3: Classical over segmentation.

4 Using the watershed transform

4.1 Classical approach

The simplest way of using the watershed is to preprocess the image we want to segment so that the boundaries of our objects are bright (e.g apply an edge detector) and compute the watershed transform of the edge image. Watershed lines will correspond to the boundaries and our problem will be solved. This is rarely useful in practice because there are always more regional minima than there are objects, either due to noise or natural variations in the object surfaces. Therefore, while many watershed lines do lie on significant boundaries, there are many that don't. Figure 3 illustrates the problem.

A common way to address this problem is to use a ignore the minima with a too small dynamic. Figures 4 and 5 illustrate the effect of this technique. An usage example is shown in Section 8.1. The ITK watershed implementation also allow a threshold to be applied first to reduce the number of local minima.

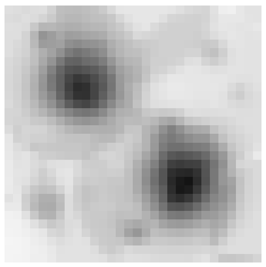
An obvious alternative that is very similar in practice is to smooth the image so that there are fewer regional minima.

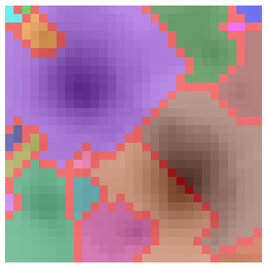
There are some situations where this direct approach can be useful. The best known of these is probably binary object splitting, in which the watershed transform is applied to the distance transform of a mask. This approach is widely used to split touching, approximately circular objects, such as cells.

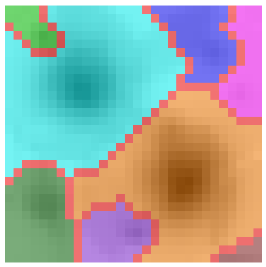
4.2 Watershed from selected minima

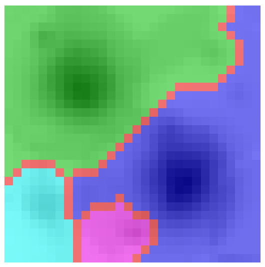
An alternative approach leads us to the concept of markers mentioned earlier. We will also see that the thresholding approach used by ITK may be considered as a specific case of the same approach.

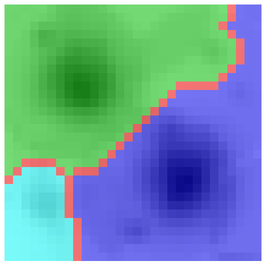
Suppose we have preprocessing steps that let select regional minima that lead to the correct segmentation. How would we modify the topography of the image so that only the desired regional minima are present and the watershed lines that remain are the same as they were before the modification (i.e we only remove the unwanted boundaries)?

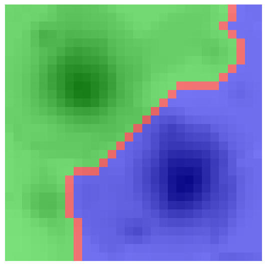

(a) Input image

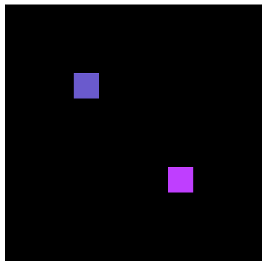

(b) No minima level


(c) Minima level = 10


(d) Minima level = 20


(e) Minima level = 30 and Minima level = 40


(f) Minima level = 50


(g) markers

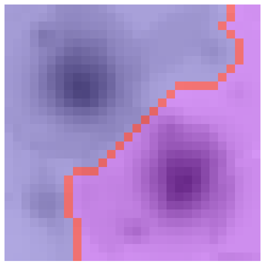
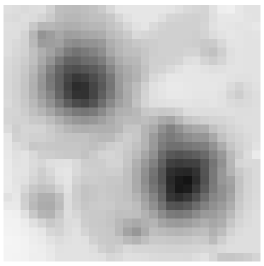
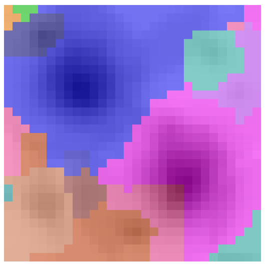
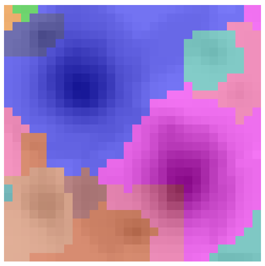
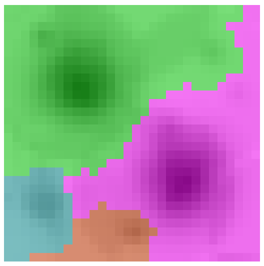

(h) Watershed from markers

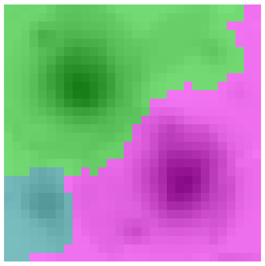
Figure 4: Different results of morphological watershed on an artificial image. The labeled image are overlayed on the input image with *LabelOverlayImageFilter*. From (b) to (f), an example of reduction of the classical watershed over segmentation by minima dynamic filtering. (g) and (h) show a marker defined by a user, and the result over the input image. All the watersheds were performed with $MarkWatershedLine = true$ and $FullyConnected = false$.

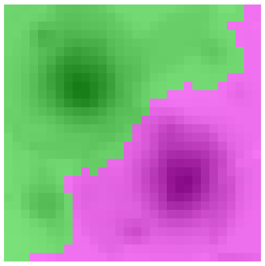

(a) Input image

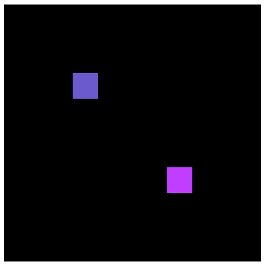

(b) No minima level


(c) Level = 0.1


(d) Level = 0.4


(e) Level = 0.63


(f) Level = 0.65


(g) markers

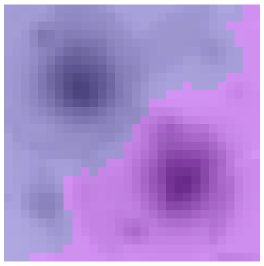

(h) Watershed from markers

Figure 5: Different results of ITK's watershed on an artificial image. The labeled image are overlayed on the input image with *LabelOverlayImageFilter*. From (b) to (f), an example of reduction of the classical watershed over segmentation by minima dynamic filtering. (g) and (h) show a marker defined by a user, and the result over the input image.

This can be achieved by performing a *reconstruction by erosion* or *recursive conditional erosion* from the selected regional minima. This process is also called *minima imposition*. Figure 6 illustrates the result of applying a reconstruction by erosion to a 1D image. The original image is shown in blue and has regional minima A, B, C, D and E. The marker image which is recursively eroded is shown in black, with a single zero at regional minimum B. The red plateaus illustrate the parts of the result image that are above the original. The resulting image has only on regional minima that corresponds to B.

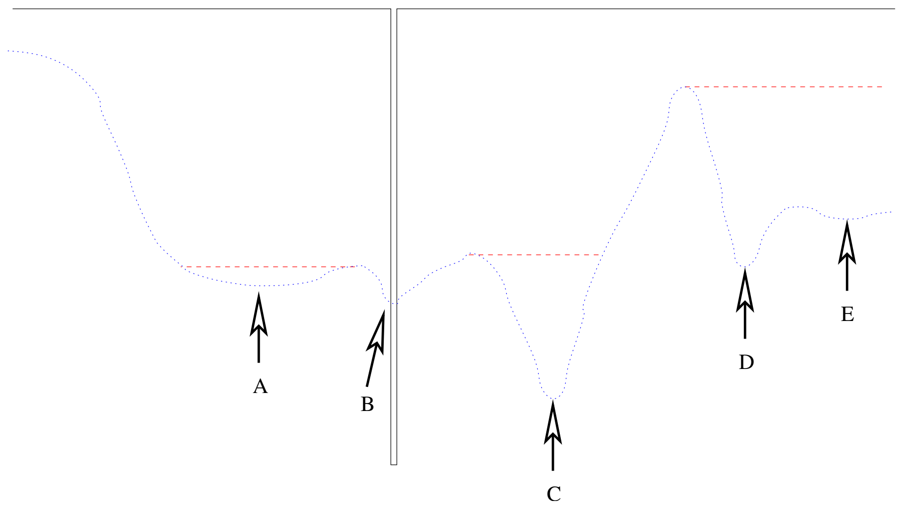


Figure 6: Reconstruction by erosion from a single regional minimum.

A more formal definition of recursive erosion is:

$$h_{n+1} = \max\{f, \varepsilon h_n\} \tag{1}$$

where $h_0(x)$ is the function, with value 0 at the markers and ∞ otherwise and ε denotes the erosion operation.

This procedure may not seem to offer much advantage – after all how are we going to decide which minima are useful to us? The key step to using these ideas in practice is to realize that an arbitrary mask image can be used to create minima in an image in locations we want simply by multiplying the image by the mask. Recursive erosion of the original image using that mask then produces a topology to which we can apply a watershed transform. The mask image is commonly called a marker image.

This is the fundamental concept behind the use of markers in watershed based segmentation.

Let's summarize the steps again:

1. Generate a marker image. The marker image will be used to create one, possibly extended, minima inside each object of interest. A marker image will always need to create at least two minima, and at least one minima will probably be used to segment the background. The marker image may be generated via interaction with an operator or by various preprocessing steps.

2. Impose the minima on the control surface. This might be done by multiplying by the marker image (or an inverted version of it, depending on convention).

3. Recursively erode the control surface to eliminate all minima besides those introduced in the previous step. This produces a new control surface.

4. Apply the watershed transform to the new surface.

4.3 Efficiency issues

The procedure outlined above introduces a lot of additional steps, some of which could be reasonably computationally expensive (i.e. recursive erosion). Other steps have also been mentioned in passing (*lower completion* in Section 3) that may be potentially time consuming. Fortunately we will discover in Section 5 that the appropriate choice of algorithm implementing the watershed transform will carry out these steps implicitly.

5 Algorithms

The two that will be outlined here are known as *image integration*[15] and *hill climbing* [5], both of which are classed as ordered algorithms and are examples of graph flooding approaches. These algorithms implement the watershed transform by topographic distance (or approximate it) and are derived from Dijkstra-Moore minimal path algorithms of graph theory.

5.1 Ordered queues

Both of these implementations depend on ordered queues to avoid the need for explicit computation of lower complete images and recursive erosion. An ordered queue is a hierarchical priority queue with two levels. One level is defined by the pixel gray level, with darker pixels having higher priority, while the second applies to pixels of the same grey level and provides a FIFO ordering. The FIFO ordering leads to a breadth first propagation across plateaus because the ordering becomes related to distance from the regional minima. This is the mechanism that avoids the to explicitly compute lower slope.

Early implementations used a special ordered queue for 8 bit data, which may be where the misconception that a morphological watershed can only be applied to 8 bit data types arose – in fact some of the early work along these lines was covered by a patent[8]. The new ITK watershed algorithms use standard c++ containers (maps and standard queues) to implement the ordered queue[9].

In both algorithms each pixel is regarded as a node in a graph that is connected to neighboring pixels by pairs of edges. Costs of the edges are defined in terms of a cost related to the gradient.

5.2 Image integration

This algorithm maintains a pair of images – a distance image and a label image. The distance image is initialized to initialized to ∞ everywhere, except at the minima where it is initialized to the value of the minima while the label image is initialized to 0 everywhere, except at the minima where it is initialized to the index of the minima (or, more typically, the value of the marker image at those points).

Wavefronts are propagated from each minima, using the ordered queue approach. As the wavefront reaches pixel the distance travelled by the wavefront is compared to the distance stored in the distance image. If

[8] I'm told that the associated paper is [14] and that there may be an English version in [7]

[9] One potential problem with applying this style of algorithm to non integer data is finding the regional minima. Reconstruction approaches are often used to find regional minima of integer type images, by subtracting 1, carrying out a recursive dilation and subtracting the two. The same concept does not extend to floating point data. However, if regional minima are available the algorithms we are discussing can be applied to floating point data. It is possible to use other techniques, such as flooding approaches, to find regional minima of floating point images.

the distance travelled by the new wavefront is less than the stored value then that pixel is given the label associated with the wavefront, otherwise the label is left unchanged.

5.3 Hill climbing

Hill climbing is an approximation that eliminates the need to maintain a distance image. The approximation employed is that distances between all neighbors in a 4, 6, 8, 18 or 26[10] and the center of the neighborhood are equal – i.e the various $\sqrt{2}$ terms are ignored. The position of the watershed lines can have some dependence on raster ordering because of this. In general the approach is very useful and efficient.

There seem to be at least 3 algorithms that fall into the hill-climbing category.

The assumption that the distance between a pixel and all of its neighbors is 1 means that the steepest upper neighbors of a pixel are simply the unlabelled neighbors. This applies to all 3 algorithms mentioned below.

Algorithm 1

- lower complete the image
- initialize the queue with all pixels except interior of minima.
- while (!empty(queue))
  - Take the pixel p from the head of the queue
  - for all pixels q that are steepest upper neighbors of p; do
    - if q is unlabeled, then
      propagate label(p) to label(q);
      else if (label$(q) \neq$ WS) and (label$(q) \neq$ label(p) then
      label(q) = WS;

This is the most basic version of hill climbing, and is presented here for completeness. There are at least two reasons why we shouldn't use this approach. Firstly, the lower completion step is explicit, which we'd like to avoid and secondly *all* non minima pixels are placed on the queue, which means that the queue length is almost the same as the number of image pixels, increasing the overhead of queue management. We shall see that it is only necessary to have the pixels on the borders of the regions in the queue.

In addition to these problems there is no provision for implicit imposition of minima for a marker based segmentation approach.

Algorithm 2 - commonly attributed to Beucher.

This algorithm take a gray scale image I and a marker image M which contains a set of markers with different labels. It can be split into two steps:

1. copy the marker image M to the output image, and put all the marker pixels p with background neighbors in the ordered queue, with the priority $I[p]$.

[10] 4 and 8 connected neighborhoods are used in 2D, 6, 18 and 26 in 3D

2. get a pixel from the ordered queue at the highest priority level, and propagate the label of p to all the unlabeled neighbors of p. The unlabeled neighbors q are added in the ordered queue with priority $I[q]$, if $I[q]$ is greater than the priority level of p, or with the priority of p otherwise.

The detailed algorithm is shown in Figure 7.

```
Algorithm 5.1: BEUCHERWATERSHED($I,M$)

comment: initialization stage
$queue \leftarrow \emptyset$
for each $p \in M$
  do
    comment: copy the marker image to the output image
    $O[p] \leftarrow M[p]$
    comment: put the edges of the marker in the queue
    if $M[p] \neq WSLABEL$
      then
        $bgNeighbor \leftarrow$ false
        for each $q \in \mathrm{N}(p)$
          do
            if $M[q] = WSLABEL$
              then $bgNeighbor \leftarrow$ true
        if $bgNeighbor$                                        (1)
          then $queue[I[p]] \leftarrow queue[I[p]] \cup \{p\}$

comment: flooding stage
while $queue \neq \emptyset$
  do
    $v \leftarrow \mathrm{FIRST}(queue)$
    while $queue[v] \neq \emptyset$
      do
        $p \leftarrow \mathrm{FIRST}(queue[v])$
        for each $q \in \mathrm{N}(p)$
          do
            if $O[q] = WSLABEL$
              then
                $O[q] \leftarrow O[p]$
                $v \leftarrow \mathrm{MAX}(I[q], v)$                  (2)
                $queue[v] \leftarrow queue[v] \cup \{q\}$
```

Figure 7: Simple watershed algorithm. $FIRST(queue)$ extract the highest priority element from the queue. $N(p)$ return the list of p's neighbors. $MAX(a,b)$ returns the maximum of a, and b. $WSLABEL$ is the label used to define background and watershed pixels (usually 0). The lower completion is done by the statement (2). Determine if a marker pixel before adding it in the queue (1) is not required, but helps to decrease memory usage in case of large markers.

It should be noticed that:

- this algorithm use ordered queues,
- this algorithm doesn't mark the watershed lines,
- the ordered queue only contains the borders of the growing regions.

Algorithm 3 - commonly attributed to Meyer.

This algorithm take a gray scale image I and a marker image M which contains a set of markers with different labels. It can be split into two steps:

1. copy the marker image M to the output image, and put all the background pixels p with markers in there neighbors in the ordered queue, with the priority $I[p]$, and mark p as already in the queue. All the marker pixels are marked as already in the queue.
2. get a pixel p from the ordered queue at the highest priority level. If the neighborhood contains only the same label or watershed pixels, mark p with that label. The neighbors q of p which are not already in the queue are added to the queue with priority $I[q]$, if $I[q]$ is greater than the priority level of p, or with the priority of p otherwise.

The detailed algorithm is shown in Figure 8.

Contrary to the previous algorithm, the label in the output image can't be used to determine if the pixel has already been added to the queue, and so we need to maintain this status by another way. It is the role of S in Figure 8. We'll discuss in Section 6 how this status can be implemented.

It should be noticed that:

- this algorithm mark the watershed lines,
- the queue only contains the borders of the growing regions,
- this algorithm never propagates watershed lines, and that the watershed lines it produces have a width of 1 pixel. Also, the watershed lines form a connected object if we consider the opposite connectivity than the one used in the algorithm: with a 2D image the watershed lines are 8-connected if the algorithm is run as 4-connected, and 4-connected if it is run as 8-connected.

6 Implementation

6.1 With ITK watershed

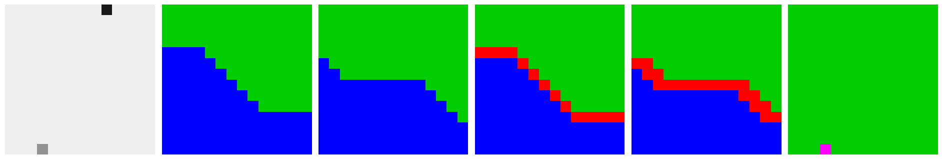
The watershed from markers can be implemented with the ITK watershed, by using the technique of *minima imposition* exposed in Section 4.2. Several steps are required:

- Impose the minima to the input image, with the `MinimaImpositionImageFilter`
- Perform the watershed with the `WatershedImageFilter`
- Restore the labels produced by the `WatershedImageFilter`. `WatershedImageFilter` produce arbitrary labels which doesn't match the labels in the marker image. There is currently no filter able to find the label correspondence in ITK, so this step must be performed by hand with a code like the one before. The `ChangeLabelImageFilter` is well designed for this task:

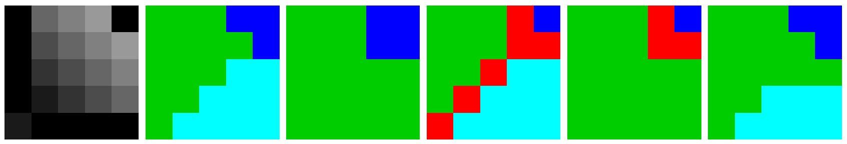
```
filter->Update();

typedef itk::ChangeLabelImageFilter< LIType, IType > ChangeLabelType;
```

```
Algorithm 5.2: MEYERWATERSHED($I$, $M$)

comment: initialization stage
for each $p \in M$
  do $S[p] \leftarrow$ false
$queue \leftarrow \emptyset$
for each $p \in M$
  do
    comment: copy the marker image to the output image
    $O[p] \leftarrow M[p]$
    if $M[p] \neq WSLABEL$
      then
        $S[p] \leftarrow$ true
        comment: add all the neighbors of $p$ with WSLABEL to the queue
        for each $q \in N(p)$
          do
            if $\neg S[q]$ and $M[q] = WSLABEL$
              then
                $S[q] \leftarrow$ true
                $queue[I[q]] \leftarrow queue[I[q]] \cup \{q\}$

comment: flooding stage
while $queue \neq \emptyset$
  do
    $v \leftarrow$ FIRST($queue$)
    while $queue[v] \neq \emptyset$
      do
        $p \leftarrow$ FIRST($queue[v]$)
        comment: found the label of $p$ and determine if it is a watershed pixel
        $label \leftarrow WSLABEL$
        $watershed \leftarrow$ false
        for each $q \in N(p)$
          do
            if $O[q] \neq WSLABEL$ and $\neg watershed$
              then
                if $label \neq WSLABEL$ and $O[q] \neq label$
                  then $watershed \leftarrow$ true
                  else $label \leftarrow O[q]$
        comment: if $p$ is a watershed pixel, don't propagate this pixel
        if $\neg watershed$
          then
            comment: assign the label value to the current pixel
            $O[p] \leftarrow label$
            comment: put all the neighbors of p in the queue
            for each $q \in N(p)$
              do
                if $\neg S[q]$
                  then
                    $S[q] \leftarrow$ true
                    $h \leftarrow$ MAX($I[q]$, $v$)                (1)
                    $queue[h] \leftarrow queue[h] \cup \{q\}$
```

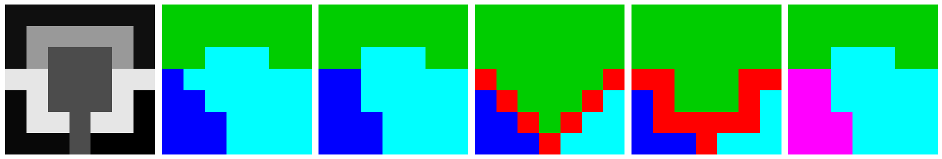
Figure 8: Watershed algorithm with watershed lines. $FIRST(queue)$ extract the highest priority element from the queue. $N(p)$ return the list of p's neighbors. $MAX(a,b)$ returns the maximum of a, and b. $WSLABEL$ is the label used to define background and watershed pixels (usually 0). The lower completion is done by the statement (1)



(a) Plateau (b) $F = false$ and $M = false$ (c) $F = true$ and $M = false$ (d) $F = false$ and $M = true$ (e) $F = true$ and $M = true$ (f) ITK



(g) Thick line (h) $F = false$ and $M = false$ (i) $F = true$ and $M = false$ (j) $F = false$ and $M = true$ (k) $F = true$ and $M = true$ (l) ITK



(m) Button hole (n) $F = false$ and $M = false$ (o) $F = true$ and $M = false$ (p) $F = false$ and $M = true$ (q) $F = true$ and $M = true$ (r) ITK

Figure 9: Three difficult test images. The first image, plateau, illustrates a bad result produced by ITK's watershed in which most of the area is assigned to one minima. This problem is also shown in Figure 1. The morphological watershed give a better result, but not yet perfect: the separation line should not be curved as it is. This problem is also visible in Figure 10(i). The reason for the problem is as follows - the example is attempting to split a plateau into two regions, and should therefore produce a Voronoi tessellation, with the two markers being separated by a straight line. This does not happen because various approximations have been made, as mentioned in Section 5.3, that influence the estimation of the distance function. Even if a more accurate underlying distance function was used the computations are still being carried out on a grid and so are limited by the digital representation of lines. The next image, thick line, again show that ITK watershed can produce strange results with a green label which is where we should have the light blue one. Morphological watershed gives good results. The last image, button hole, is a very difficult case for watershed algorithms which mark watershed lines: the watershed pixel at the bottom of the image stop the label propagation. There are several possible behaviors: produce a large watershed line which covers all the zone (Vincent-Soille algorithm, not shown here), let the label on the top of the image propagate into the zone (Meyer algorithm), split the zone in 2 zones with the 2 labels of the bottom of the image (topological watershed, not shown here). See [16] for more details.

```
ChangeLabelType::Pointer changeLabel = ChangeLabelType::New();
changeLabel->SetInput( filter->GetOutput() );

itk::ImageRegionConstIterator< LIType > wit( filter->GetOutput(),
                filter->GetOutput()->GetLargestPossibleRegion() );
itk::ImageRegionConstIterator< IType > mit( reader2->GetOutput(),
                filter->GetOutput()->GetLargestPossibleRegion() );
for ( mit.GoToBegin(), wit.GoToBegin();
  !mit.IsAtEnd();
  ++wit, ++mit )
  {
  if( mit.Get() != 0 )
    { changeLabel->SetChange( wit.Get(), mit.Get() ); }
  }
```

This approach has several drawbacks:

- it doesn't work if there some markers are not separated by background pixels - they are considered as a single marker
- restoring the labels is not included in a filter and so break the ITK pipeline model
- it's less efficient, as shown in Section 7
- it still gives poor results with plateaus. A lower completion souled be performed before to also solve this problem[11].

A full example of implementation can be found in `iwsm.cxx`.

6.2 MorphologicalWatershedFromMarkersImageFilter

`MorphologicalWatershedFromMarkersImageFilter` implements algorithms 5.3 and 5.3. Setting attribute `MarkWatershedLine` to `true` selects algorithm 5.3 (Meyer) while setting to `false` selects algorithm 5.3 (Beucher). The filter is templated over the input image type, which can be any scalar type, and the label image type which should be an integer. The filter takes 2 inputs:

- a gray scale image which can be set with `SetInput(image)`, `SetInput(0, image)` or `SetInput1(image)`.
- a marker image which can be set with `SetMarkerImage(image)` and `SetInput2(image)`. `SetInput(0, image)` should be avoided because it will fail if the input image and the marker image are not of the same type. The markers are all the non-zero values in the image.

Algorithm 5.3 requires the status of all pixels to be tracked – pixels are either *already added to the queue* or *not yet added to the queue*. This task is often done by using negative values in the output image for the pixels not yet added to the queue. In our implementation, we have chosen to use a binary image to store this, while labels always take values above 0 and watershed lines are marked with value 0. The negative values option have several drawbacks:

[11] Lower completion is not implemented yet in ITK

- it requires to use a different image type than the label type defined by the user in case the user have chosen an unsigned pixel type.
- it requires to use one of the value which can be possibly already used by a label in the marker image, in case of a signed type.

Those problems can be avoided[12] by using a type which contains a larger range of value than the type chosen by the user, but then, we must copy the data to an image of the type chosen by the user at the end of the algorithm.

The connectivity is defined using the usual ITK way: `SetFullConnected(false)` (4-connected in 2D, 6-connected in 3D) or `SetFullConnected(true)` (8-connected in 2D, 26-connected in 3D)[13].

6.3 MorphologicalWatershedImageFilter

`MorphologicalWatershedImageFilter` performs a watershed transformation *without* markers. It encapsulate several filters:

- `HMinimaImageFilter`
- `RegionalMinimaImageFilter`
- `ConnectedComponentImageFilter`
- `MorphologicalWatershedFromMarkersImageFilter`

While `RegionalMinimaImageFilter`, `ConnectedComponentImageFilter`, and `MorphologicalWatershedFromMarkersImageFilter` are the usual steps of the watershed usage, the `HMinimaImageFilter` is a convenient way to avoid the classical watershed's over segmentation by allowing the user to select the dynamic of the minima. The minimum dynamic of minima can be selected with `SetLevel()`.

6.4 WrapITK integration

The new filters have been fully tested with python, with the WrapITK project. An example of python usage is shown in Section 8.1. The sources of this article contain the configuration files needed to build build the new filters with WrapITK, as an external project - see [8] for more information about how to build and use an external project.

7 Timing and performance

7.1 Classical watershed

The relative performance of the new filters are shown in Table 1. The timings were obtained using the `./perf3D images/ESCells.img` on an Athlon 64 Processor 2800+ (1802 MHz) with 512Kb cache, 512 Mb of RAM running Mandriva linux 64 bits with gcc 4.1.1. The input image have a size of $371 \times 371 \times 34$.

[12] It's not true for the unsigned long pixel type

[13] One should regret to not have the 18-connected connectivity in 3D

FC	MWL	Find min	Label min	Ws from markers	Morph ws	ITK ws
0	0	1.39	0.0566	2.74	4.61	7.47
0	1	1.38	0.056	4.34	6.53	-
1	0	2.85	0.0568	5.8	9.54	-
1	1	2.85	0.0568	10.5	14.4	-

Table 1: Execution times. Note - FC and MWL refer to FullyConnected and MarkWatershedLine.

The times for steps involved in a morphological watershed are shown in the first few columns, with the time from a combined filter and the ITK watershed shown last. It should be noted that the ITK watershed does not allow connectivity to be modified by the user - it always uses face connected mode and doesn't have an inbuilt mechanism for boundary selection. The existing ITK watershed is therefore somewhat slower than the new implementation when used to compute a traditional watershed. The new implementations are not easily threadable and therefore may prove to be slower on some systems. Moreover, this results doesn't show the performance in the case where the user modify the `Level` parameters of the ITK filter, case where the time needed to update the filter should be a lot shorter. However, as discussed in earlier sections, the new implementation produces more stable results than the rainfall model and is more dedicated to a marker based approach.

7.2 Marker based watershed

The performance in case of a marker based approach are shown in Table 2, and were obtained on the same system than the ones in Section 7.1, with the command `./mperf3D images/ESCells.img images/ESCells-markers.tif`. The ITK watershed use the implementation presented in Section 6. The extra steps of this implementation are also shown in this table.

FC	MWL	Impose min	ITK ws	Match labels	Change labels	Morph ws	ITK ws
0	0	2.32	6.81	0.0295	0.273	3.12	9.43
0	1	-	-	-	-	5	-
1	0	-	-	-	-	6.57	-
1	1	-	-	-	-	11.6	-

Table 2: Execution times with marker based approach. Note - FC and MWL refer to FullyConnected and MarkWatershedLine.

7.3 Other performance issues/comments

- Preventing boundary checks - not really the ITK way, but if a user is willing to throw away the image border by treating it as a watershed line then there is no need to test for boundary conditions. There isn't even any need to use the face calculator approach because border pixels will be initially tagged as processed, and therefore we can be certain their neighbors are never checked.
- Streaming and threading - queue based algorithms are generally difficult to thread efficiently and generally need to have all of the input available so prevent streaming from working well. The existing ITK watershed does provide a multi threaded implementation and similar techniques may be possible with the new implementation. Other possibilities include using threads to improve performance of queues.

8 Using watersheds in ITK

It should be noticed that all the images shown in this article have been produced with ITK and can be easily reproduced and tested by running *ctest* in the build tree.

8.1 Pronucleus segmentation in bovine embryo

In this example, we use the watershed filter to split two nearly round objects: the male and female pronucleus in a bovine embryo, around 22 hours after fecondation.

The program is in python for conciseness[14]. A C++ version, `splitPronucleus2D.cxx`, is also available.

```
import itk
reader = itk.ImageFileReader.IUC2.New(FileName='embryo.png')
```

We use a median image filter to remove some noise, and threshold the image to get a binary image[15].

```
median = itk.MedianImageFilter.IUC2IUC2.New(reader)
th = itk.BinaryThresholdImageFilter.IUC2IUC2.New(median, LowerThreshold=40)
```

The binary image produced by the `ThresholdImageFilter` contains lot of small objects. To remove them, we use an *opening by reconstruction* which has the advantage to preserve the edges of the objects.

```
kernel = itk.strel(2, 30)
open = itk.OpeningByReconstructionImageFilter.IUC2IUC2.New(th, Kernel=kernel)
```

There is now only one binary object in the image, but it contains lots of holes which will produce an unusable distance map for the watershed. We fill them with a *hole filling* filter. This algorithm, based on a morphological reconstruction, fill the hole in a binary object but preserve the edges of the object. In this case, it is important to keep the two cavities between the two pronucleus unchanged.

```
fill = itk.GrayscaleFillholeImageFilter.IUC2IUC2.New(open)
```

Then we compute the distance map which will be used by the watershed. The binary object must be inverted to compute the distance *inside* the object, and the distance map must be inverted to be used by the watershed.

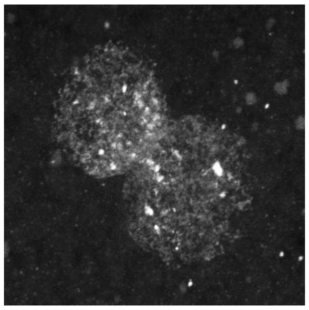
```
invert = itk.InvertIntensityImageFilter.IUC2IUC2.New(fill)
distance = itk.DanielssonDistanceMapImageFilter.IUC2IUC2.New(invert)
idistance = itk.InvertIntensityImageFilter.IUC2IUC2.New(distance)
```

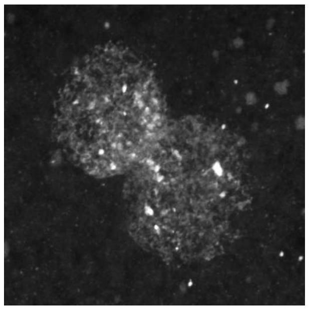
The watershed is then performed to split the two pronucleus. A `Level` is set to avoid the too small minima in the border of the object and thus avoid over segmentation. The label produced by the watershed are then displayed on top of the original image.

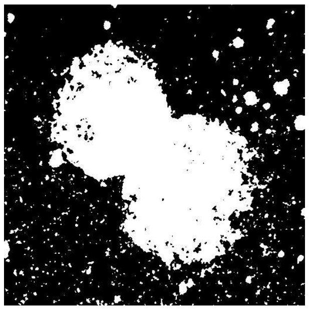
The watershed extends the labels on the entire image. We use a `MaskImageFilter` to keep the labels inside the object. Another commonly used method to keep the labeled zone in the object is to use the regional minima *and* the background as markers, to perform a marker based watershed.

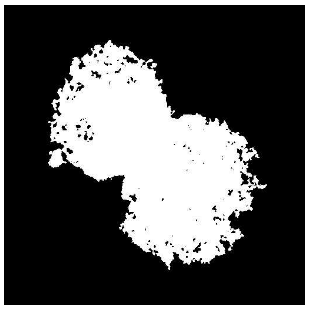
[14]And also to promote python wrappers.

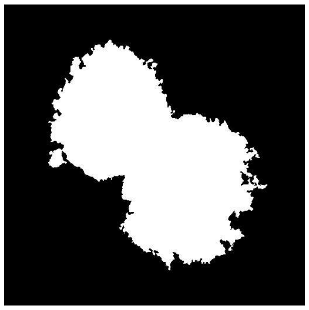
[15]The methods to remove the noise and binarize the image are kept simple for this example

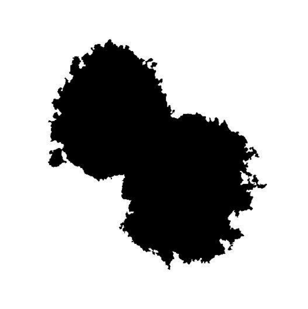

(a) Input

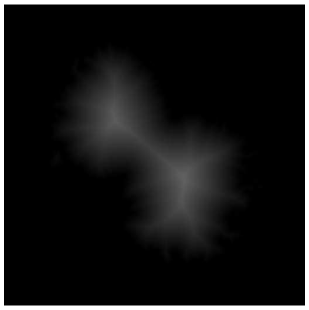

(b) median

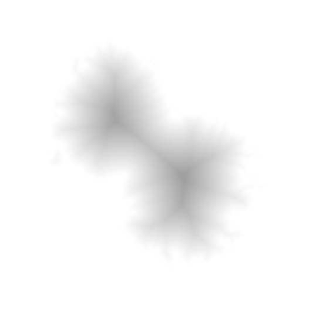

(c) threshold

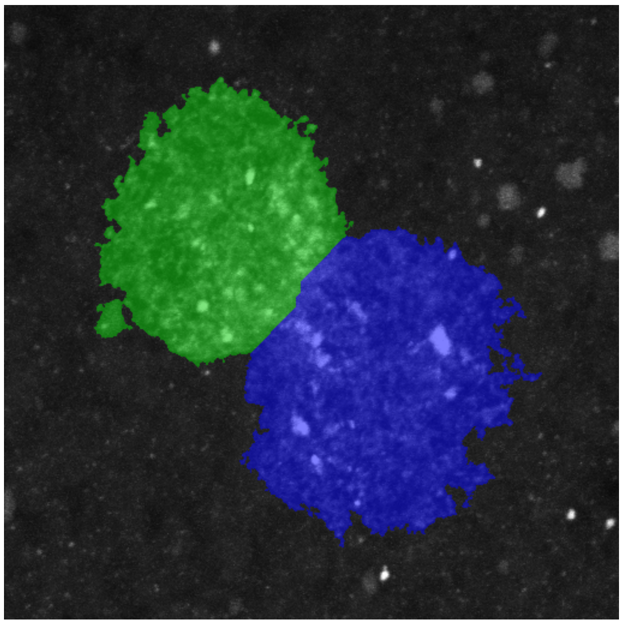

(d) opening by reconstruction


(e) fill hole


(f) invert


(g) distance


(h) inverted distance


(i) morphological watershed

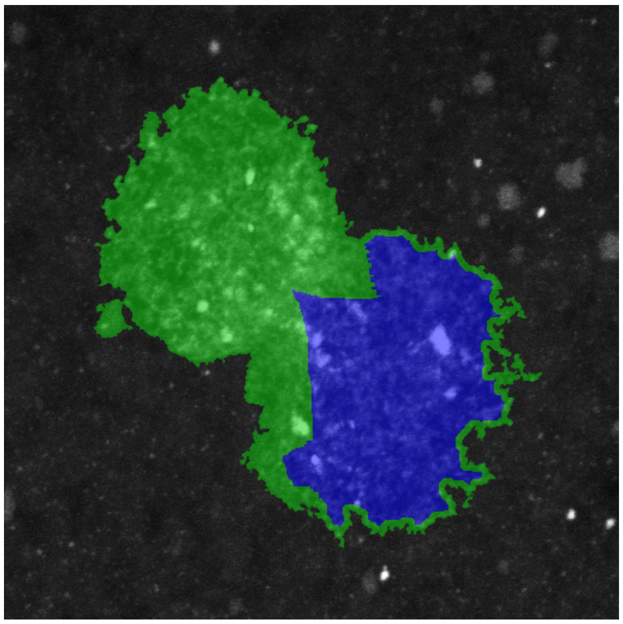

(j) ITK watershed

Figure 10: The results of the different steps of the segmentation of bovine pronucleus.

```
ws = itk.MorphologicalWatershedImageFilter.IUC2IUC2.New(idistance,
                                                        MarkWatershedLine=False,
                                                        Level=10)
mask = itk.MaskImageFilter.IUC2IUC2IUC2.New(ws, fill)
overlay = itk.LabelOverlayImageFilter.IUC2IUC2IRGBUC2.New(reader,
                                                          mask,
                                                          UseBackground=True)
```

The same watershed is done with the ITK watershed filter[16].

```
ws2 = itk.WatershedImageFilter.IUC2.New(idistance, Level=0.5)
relabel = itk.RelabelComponentImageFilter.IUL2IUC2.New(ws2)
mask2 = itk.MaskImageFilter.IUC2IUC2IUC2.New(relabel, fill)
overlay2 = itk.LabelOverlayImageFilter.IUC2IUC2IRGBUC2.New(reader,
                                                           mask2
                                                           UseBackground=True)
```

ITK watershed clearly produce a wrong segmentation in that case. However, using a floating point image to store the distance map would have produce a better result very similar to the one of the morphological watershed. It seems to be another example of plateaus problem, where each level curve is seen as a plateau when the pixel type of the distance image is an integral.

The segmentation of the morphological watershed is quite good, even if not perfect: there are some curves in the separation line of the two labels. This is the result of the distance approximation in the watershed algorithm.

8.2 2D electrophoresis gel segmentation

Protein separation in two dimensions on electrophoresis gels is one of the many situations where image segmentation techniques are commonly used to make measurements of spots which are often touching. This section describes a multi-stage watershed approach and illustrates the use of the watershed transform on gradient images and the use of watershed lines, which are a new capability of the filters introduced in this paper, as markers. This technique almost certainly isn't the best way of segmenting gels, and probably won't work on many gels without tuning. It is simply offered as an example of how watershed transforms may be used. There are many variants of the basic algorithm, some of which are discussed in the text.

The algorithm source code is in `segGel.cxx` and the images used in this example are from `http://www-lecb.ncifcrf.gov/flicker/`.

The original image is shown in Figure 11(a). The spots we are interested in are dark, and frequently touching. In gels often have all sorts of artifacts, ranging from streaks along separation dimensions to an over abundance of a protein that leads to a large dark spot that may saturate the image brightness (i.e be darker than the dark floor of the imaging system). We will not address these sorts of issues here.

The basic steps are

- **Find spot markers.** Here we'll use a combination of smoothing and minima detection. Depending on the noise levels of the image there are likely to be minima in bright regions of the image. It may be possible to discard a number of minima based on brightness, depending on how certain we are about the maximum brightness of a valid spot.

  The results of an unmodified minima detection after median smoothing is shown in Figure 11(b).

[16] A *RelabelComponentImageFilter* is used here because *unsignedlong* is not one of the available types for the *MaskImageFilter* in python

- **Find background marker.** We'll use a watershed on the raw image to generate a background marker. The background marker will be the watershed line. Setting attribute `MarkWatershedLine` to `true` will result in the watershed line being marked with zero values. In this situation we are using the watershed to tessellate the image into regions. The region boundaries will be in the brightest parts of the image between markers. Figure 11(c) illustrates the results of this step.
- **Combine markers.** The spot and background markers are combined by adding them together, after using the binary threshold filter to generate the background marker from the first watershed – see Figure 11(d).
- **Compute gradient.** A gradient of the raw image is then computed. We use the `GradientRecursiveGaussianImageFilter`, but many options are available. The gradient image is shown in Figure 11(e).
- **Compute watershed transform** using the gradient image as the control surface. Figure 11(f).
- **Optional additional watershed steps.** The results of this sort of segmentation, based on an image gradient, often appear to underestimate the spot size. This is because the watershed transform will produce edges along image ridges, which correspond to the locations of maximum image gradient. Unfortunately this tends not to correspond to the location a human observer would select.

  This easy easy to understand if a gaussian spot is considered – the position of maximal gradient is halfway down the edge of the spot, while a human would tend to mark the edge as being near the point where the spot brightness becomes indistinguishable from the background brightness. This location is often closer to the position of the maximum of the second derivative.

  Therefore we can "improve" the segmentation by using the previous results to develop a new set of markers and apply the watershed transform using the second derivative as a control surface. It is sensible to use a different set of spot markers to the original because there are two maxima of second derivative, and the original markers will tend to be inside both, while the results of the first stage will be contain the closer maxima. Figures 11(g), 11(h) and 11(i) illustrate these steps.
- **Postprocessing.** This implementation doesn't do any post processing, but in most cases it would be a useful step. Error situations, such as spots being a crazy size, shape or brightness could be detected brought to the attention of the user or attempts could be made to automatically correct the result.

A cursory examination of the results will show that segmentation is far from perfect. Some touching spots only have one marker so only one spot is found and there are markers placed in what should probably be considered as background. These problems may be addressed in a number of ways, such as performing more careful selection of regional minima and sharpening the original image using deconvolution, as well as various post processing options. We are not going to explore these issues here because the aim is to illustrate a few ways in which the watershed transform can be used.

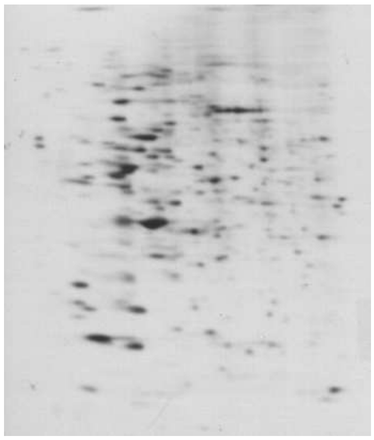
9 Discussion

There is a level of erroneous folklore surrounding the watershed transform and it is hoped that this report helps to clear the air, as well as providing a useful tutorial/introduction to a useful tool.

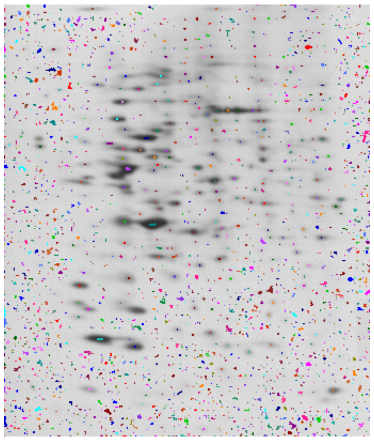
10 Related work

This work has lead us to develop some other filters to:

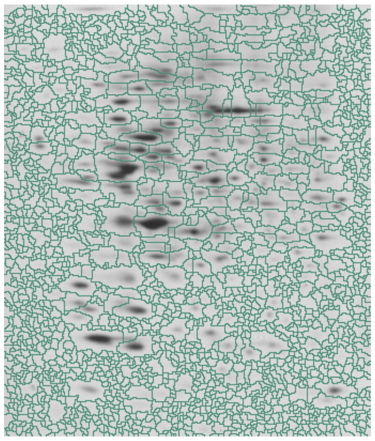
- Find regional extrema [3]
- Superimpose a colored label image onto an image [12]
- Impose the minima [11]



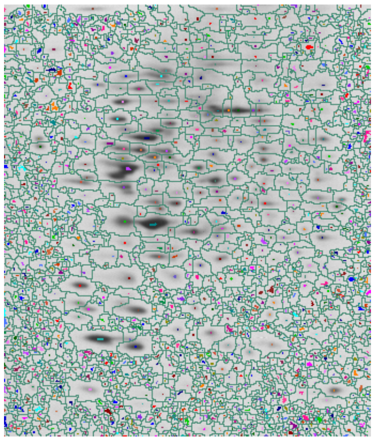
(a) Input



(b) Regional minima

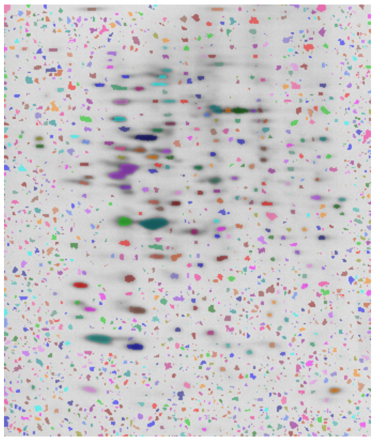


(c) Background marker

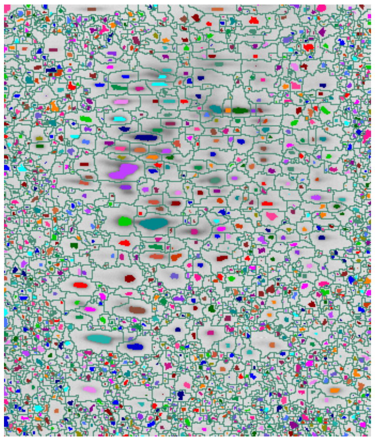


(d) All markers

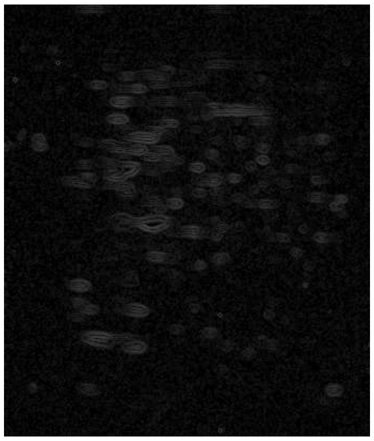
(e) Gradient



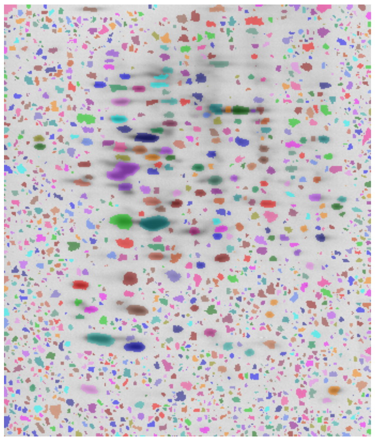
(f) Result (background marker removed)



(g) All markers - stage 2



(h) Second derivative (gamma corrected)



(i) Result stage 2 (background marker removed)

Figure 11: Steps used in segmenting a gel using multiple watershed stages.

- Label the connected component [2]
- Invert the intensity of an image [10]
- Enhance the morphological reconstruction [1]
- Compute some informations about the shape of the labeled objects (not published yet)

11 Acknowledgments

We thank Dr Pierre Adenot and MIMA2 confocal facilities (`http://mima2.jouy.inra.fr`) for providing ESCells image used for timing and for the bovine embryo image.

References

[1] Richard Beare. Improving performance of morphological reconstruction. *Insight Journal*, January - June 2006. 10

[2] Richard Beare. Optimization of connected component labelling. *Insight Journal*, January - June 2006. 10

[3] Richard Beare and Gaëtan Lehmann. Finding regional extrema - methods and performance. *Insight Journal*, August - December 2005. 10

[4] S. Beucher and C. Lantuéjoul. Use of watersheds in contour detection. In *Int. Workshop on Image Processing*, Rennes, France, Sept. 1979. CCETT/IRISA. 3

[5] S. Beucher and F. Meyer. *Mathematical Morphology in Image Processing*, chapter 12: The Morphological Approach to Segmentation: The Watershed Transformation, pages 433–481. Marcel Dekker Inc., 1993. 3, 5

[6] M. Couprie and G. Bertrand. Topological grayscale watershed transformation. volume 3168, pages 136–146, January - June 1997. 3

[7] E. Dougherty, editor. *Mathematical Morphology in Image Processing*. Marcel Dekker Inc., 1993. 8

[8] Zachary Pincus Gaëtan Lehmann and Benoit Regrain. Wrapitk - enhanced languages support for the insight toolkit. *Insight Journal*, January - June 2006. 6.4

[9] L. Ibanez and W. Schroeder. *The ITK Software Guide*. Kitware, Inc. ISBN 1-930934-10-6, http://www.itk.org/ItkSoftwareGuide.pdf, 2003.

[10] Gaëtan Lehmann. Invertintensityimagefilter. *Insight Journal*, August - December 2005. 10

[11] Gaëtan Lehmann. Minimaimpositionimagefilter. *Insight Journal*, August - December 2005. 10

[12] Gaëtan Lehmann. Labeloverlayimagefilter. *Insight Journal*, January - June 2006. 10

[13] A. Meijster. *Efficient Sequential and Parallel Algorithms for Morphological Image Processing*. PhD thesis, Institute for Mathematics and Computing Science, University of Groningen, 2004. 3

[14] F. Meyer. Un algorithme optimal de ligne de partage des eaux. In *Proc. 8ème Congrès Reconnaissance des Formes et Intelligence Artificielle*, pages 847–857. AFCET, November 1991. 8

[15] F. Meyer. Topographic distance and watershed lines. *Signal Processing*, 38:113–125, 1994. 3, 5

[16] L. Najman and M. Couprie. Watershed algorithms and contrast preservation, 2003. 9

[17] J. B. T. M. Roerdink and A. Meijster. The watershed transform: definitions, algorithms, and parallelization strategies. *Fundamenta Informaticae*, 41:187–228, March 2000. ISBN 90-367-1977-1. 3

[18] L. Vincent and P. Soille. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):583–598, June 1991. 3