**Advances in Science and Technology Research Journal**
Volume 8, No. 23, Sept. 2014, pages 41–44
DOI: 10.12913/22998624.1120321

*Research Article*

# FAST WATERSHED-BASED DILATION

**Jakub Smołka**[1]

[1] Section of Computers Science, Department of Technical Sciences, Pope John Paul II State School of Higher Education in Biała Podlaska, Sidorska 95/97, 21-500 Biała Podlaska, Poland, e-mail: j.smolka@dydaktyka.pswbp.pl

Received: 2014.07.18
Accepted: 2014.08.11
Published: 2014.09.09

**ABSTRACT**
A watershed-based region growing image segmentation algorithm requires a fast watershed-based dilation implementation for effective operation. This paper presents a new way for watershed image representation and uses this representation for effective implementation of dilation. Methods for improving the algorithm speed are discussed. Presented solutions may also be used for solving other problems where fast set summation is required.

**Keywords:** image processing, dilation, watershed transformation.

## INTRODUCTION

Watershed dilation is defined for the purpose of watershed-based region growing [1]. It turns out that a version of the algorithm that utilizes bitmaps is too slow. It is necessary to create a proper data structure for representation of watersheds and image region built by the algorithm. Additionally, the most important part of watershed-based region growing – the dilation – needs to be implemented in an effective manner.

## BASIC CONCEPTS

*Watershed transformation* is performed on images in which high values (that is bright points) correspond to edges [2, 3, 4]. In the case of most natural images this is not the case. That is why it is necessary to apply a gradient filter or another edge detection algorithm before watershed transformation [4]. The transformation algorithm treats pixel values in the gradient image as information about terrain relief. Watershed segmentation consists in simulated pouring water onto the relief. As water accumulates in lower areas, the catchment basins form and divide the image into multiple watersheds. Each watershed corresponds to a local minimum in the gradient image. Implementation of the watershed transformation used for the purpose of this paper produces output images in which all pixels belonging to a certain watershed have the same number. All watersheds have unique numbers.

*Watershed-based region growing* is a segmentation algorithm introduced in [1]. The algorithm begins with a start region that contains only one watershed and then tries to add additional watersheds through dilation. Some watersheds are then removed in order to bring the region's standard deviation below a certain level. The algorithm repeats the sequence of adding and removing selected watersheds until it reaches a point where the region cannot be expanded anymore, and its standard deviation is below a given threshold.

## WATERSHED-BASED DILATION

Dilation is a well-known morphological operation [5, 6]. Due to irregular watershed shapes it is necessary to redefine watershed-based dilation in the following way: *Dilated region D(R) contains all watersheds $w_i$*, which have at least one common point with region R before it is dilated (watersheds and regions are considered to be sets of points):

$$D(R) = R \cup \{a_i \mid a_i \cap R \neq \varnothing\}$$

The above definition assumes that watersheds have common border points.

$$w_i = \sum_{j=0}^{n_i} b_j \cup a_i \Rightarrow \underset{i \neq j}{\exists} a_i \cap a_j \neq \varnothing$$

where: $b_{ij}$ – $j^{th}$ arc that belongs to the border of the watershed $w_i$,
$a_i$ – interior of the watershed $w_i$,
$n_i$ – number or arcs that constitute the border of the watershed $w_i$.

## WATERSHED REPRESENTATION IN THE OPTIMAL ALGORITHM VERSION

In the optimal algorithm version watersheds are not represented as bitmaps. Instead, an array consisting of *N* sets is used. The $i^{th}$ array element ($N_i$) is a set that contains numbers of the $i^{th}$ watershed neighbors. The set $N_i$ also contains the number of the watershed *i* itself. It is important to note that unique numbers for watersheds are assumed. In the practical implementation of the algorithm two class templates are used – namely: *vector* i *set* from *The Standard Template Library* for $C^{++}$ [7].

## WATERSHED-BASED DILATION UTILIZING SETS

Using the way of representing the above-described watersheds, it is possible to define the dilation *D(R)* of region *R* as a sum of all neighbors of all watersheds that constitute region *R*:

$$D(R) = N_1 \cup N_2 \cup \ldots \cup N_k = \bigcup_{i=1}^{k} N_i$$

where: *k* – number of watersheds in region *R,*
$N_i$ – set containing numbers of $i^{th}$ watershed neighbors and the number of $i^{th}$ watershed itself.

A set summation is carried out with a *set_ union()* function template from the *STL library*. It computes the sum of two sets and its worst-case computational complexity is [7]:

$$2(n_i + n_j) - 1$$

for two sets whose sizes are $n_i$ i $n_j$ (comparison is considered to be the dominating operation).

The way in which the sets are summed can significantly influence the region growing speed. Several methods are proposed and tested. The first of them is adding sets in the order in which they are stored in the array. This method is given by the following equation:

$$D(R) = \left(\left[\{N_1 \cup N_2\} \cup N_3\right] \cup \ldots \cup N_k\right)$$

The pessimistic complexity of computing the sum of *k* sets is given by:

$$W(k) = 2(n_1 + n_2) - 1 + 2(n_1 + n_2 + n_3) - 1 + \ldots$$
$$\ldots + 2(n_1 + n_2 + \ldots + n_k) - 1 =$$
$$= 2\sum_{i=1}^{k}(k + 1 - i)n_i - 2\mathrm{n}_1 - k + 1$$

where: *k* – number of watersheds,
$n_i$ – number of $i^{th}$ watershed's neighbors.

The above equation shows that the ordering of sets (with respect to their size) affects the number of comparison operations that need to be performed. The algorithm is tested with different sets ordering: increasing size, decreasing size and unsorted.

A second way of adding sets is adding them in pairs such that the sets being added are similar in size. This method can be expressed in the following equation:

$$D(R) = \left(\left[\{N_1 \cup N_2\} \cup \{N_3 \cup N_4\}\right] \cup \ldots\right.$$
$$\left.\ldots \cup \left[\{N_{k'-3} \cup N_{k-2}\} \cup \{N_{k-1} \cup N_k\}\right]\right)$$

where: *k'* – number of watersheds (lowest power of 2 that satisfies following condition *k'* ≥ *k*).

For simplicity it is assumed that the number of watersheds *k'* is a power of 2. If the real number of sets does not meet this requirement, an appropriate number of empty sets can be added to the set array. The pessimistic computational complexity of this method is given in the following equation:

$$W(k\,) = 2(n_1 + n_2) - 1 + 2(n_3 + n_4) - 1 + \ldots$$
$$\ldots + 2(n_{k'-1} + n_k) - 1 + 2(n_1 + n_2 + n_3 + n_4) - 1 + \ldots$$
$$\ldots + 2(n_{k'-3} + n_{k-2} + n_{k-1} + n_k) - 1 +$$
$$+ \ldots + 2(n_1 + n_2 + \ldots + n_{k'}) - 1 =$$
$$= 2\log_2 k\,(n_1 + n_2 + \ldots + n_k) - \left(1 + 2 + \ldots + \frac{k}{2}\right) =$$
$$= 2\log_2 k \sum_{i=1}^{k} n_i - \frac{1 \cdot \left(1 - 2^{\log_2 k}\right)}{1 - 2} = 2\log_2 k \sum_{i=1}^{k} n_i - k' + 1$$

The above equation shows that, in this case, the order in which the sets are added (with respect to their size) should not influence algorithm's performance.

If one assumes that $k = k = 2^t$ and $t \in Z$ and additionally $n_1 = n_2 = \ldots = n_k$ (with watershed segmentation - in practice - sets' sizes are usually similar), then the complexity of the algorithm adding sets in the order they are given will be in the order $W\left(k^2\right)$ while the algorithm adding sets in pairs will have the complexity of $W\left(k \log_2 k\right)$. As one can see, adding sets in pairs results in a significantly faster algorithm.

## EDGE TRACKING

An additional speedup of dilation computation can be achieved by limiting the number of sets that need to be added. It is important to note that only the watersheds that are neighbors of the region's $R$ boundary $B \subset R$ influence the dilation's result. This fact can be used for limiting the number of added watersheds. Taking this into consideration the dilation can be expressed in the following manner:

$$D(R) = R \cup D(B) = R \cup N_1 \cup N_2 \cup \ldots \cup N_{k_b} = R \cup \bigcup_{i=1}^{k_b} N_i$$

where: $k_b$ – number of watersheds in the boundary $B$.

After each dilation the current boundary $B'$ (necessary for performing consecutive dilations) can be computed using:

$$B = D(B) \square R$$

where: $B \subset R$ – region's boundary before dilation,
$R$ – region before dilation.

The above equation contains a certain simplification that may cause the region's inner watersheds to be included in the boundary. This inconsistency, however, does not affect the region dilation's correctness. The extra watershed can be removed by using a simple equation (but this is unnecessary for the purpose of the edge tracking algorithm).

## COMPARISON OF DIFFERENT DILATION ALGORITHMS

In order to compare the algorithms described above, a test is performed. The test consists in performing a series of dilations. Watershed transformation is applied to a test image. The start region for dilation consists of only one watershed that has a middle number (more precisely: "*watershed count*/2"). Dilations are performed until the dilated region contains all watersheds in the image. Times needed to complete the test are given in Table 1. These are the total times of all dilations performed using a given algorithm. The time required for the old bitmap version of watershed-based dilation is also given for comparison.

Table 2 compares algorithms that add sets one by one and in pairs. Table 3 shows how enabling edge tracking affects the dilation performance. As one can see, adding sets in pairs can increase

**Table 1.** Times needed for dilation test completion

| Set summation algorithm | Sorting (with respect to size) | Edge tracking | $t_{test}$ [s] | Test image |
|---|---|---|---|---|
| Pairs | none | yes | 0.73 | original |
| Pairs | none | no | 1.17 | |
| Pairs | increasing | yes | 0.80 | |
| Pairs | increasing | no | 2.40 | |
| Pairs | decreasing | yes | 0.80 | |
| Pairs | decreasing | no | 2.40 | |
| Consecutive | none | yes | 0.93 | watersheds |
| Consecutive | none | no | 22.00 | |
| Consecutive | increasing | yes | 1.00 | |
| Consecutive | increasing | no | 32.87 | |
| Consecutive | decreasing | yes | 1.13 | |
| Consecutive | decreasing | no | 39.33 | |
| Bitmap | not applicable | | 22.37 | |

**Table 2.** Dilation type comparison

| Sorting | Edge tracking | $t_{consecutive}$ / $t_{pairs}$ |
|---|---|---|
| None | yes | 1.27 |
| None | no | 18.86 |
| Increasing | yes | 1.25 |
| Increasing | no | 13.69 |
| Decreasing | yes | 1.42 |
| Decreasing | no | 16.39 |

**Table 3.** Edge tracking influence

| Summation algorithm | Sorting | $t_{no\ tracking}$ / $t_{with\ tracking}$ |
|---|---|---|
| Pairs | none | 1.59 |
| Pairs | increasing | 3.00 |
| Pairs | decreasing | 3.00 |
| Consecutive | none | 23.57 |
| Consecutive | increasing | 32.87 |
| Consecutive | decreasing | 34.71 |

**Table 4.** Sorting influence

| Set summation algorithm | Edge tracking | $t_{unsorted}$ /$t_{increasing}$ | $t_{unsorted}$ / $t_{decreasing}$ | $t_{decreasing}$ / $t_{increasing}$ |
|---|---|---|---|---|
| Pairs | yes | 1.09 | 1.09 | 1.00 |
| Pairs | no | 2.06 | 2.06 | 1.00 |
| Consecutive | yes | 1.07 | 1.21 | 1.13 |
| Consecutive | no | 1.49 | 1.79 | 1.20 |

the speed dramatically (up to approximately 19 times). Enabling edge tracking can improve algorithm's performance even more – up to 35 times – depending upon the algorithm version.

Table 4 shows the influence of set sorting on summation performance. In general sets ordered with respect to size can be added faster. What is more the algorithm that adds sets in pairs is insensitive to set ordering (whether it is increasing or decreasing).

## SUMMARY

The optimized version of watershed-based dilation is significantly faster than the bitmap-based original. The best execution times are in the order of tenths of a second (on an Intel Core i5 U3317U computer), which means that a single watershed-based dilation is computed in a couple of hundredths of a second. This allows for adding optimization algorithms to watershed-based region growing that would automatically adjust its parameters.

## REFERENCES

1. Smołka J., Watershed Based Region Growing Algorithm. Annales Informatica UMCS AI 3, 2005, 169–178.
2. Beucher S., Lantuejoul C., Use of watersheds in contour detection. International Workshop on Image Processing: Real-time Edge and Motion Detection/Estimation, 1979.
3. Beucher S., The watershed transformation applied to image segmentation. Scanning Microscopy International, 6, 1992, 299–314.
4. Ibanez L., Schroeder W., Ng L., Cates J., The ITK Software Guide. Kitware Inc., 2003.
5. Seul M., O'Gorman L., Sammon M.J., Practical Algorithms for Image Analysis: Description. Examples, and Code, Cambridge Univerisity Press 2000.
6. Gonzalez R.C., Woods R.E., Digital Image Processing. Addison-Wesley Publishing Company 1993.
7. Standard Template Library Programmer's Guide http://www.sgi.com/tech/stl/