AN EFFICIENT ALGORITHM FOR 3D BINARY MORPHOLOGICAL TRANSFORMATIONS WITH 3D STRUCTURING ELEMENTS OF ARBITRARY SIZE AND SHAPE

Nikos Nikopoulos and Ioannis Pitas

Department of Informatics, University of Thessaloniki
GR-54006 Thessaloniki, GREECE
Tel./Fax: +30-31-996304
e-mail: pitas@zeus.csd.auth.gr

ABSTRACT

This paper proposes a fast algorithm for implementing the basic operation of Minkowski addition for the special case of binary three-dimensional images, using three-dimensional structuring elements of arbitrary size and shape. The application of the proposed algorithm for all the other morphological transformations is straightforward, as they can all be expressed in terms of Minkowski addition. The efficiency of the algorithm is analysed and some experimental results of its application are presented. As shown, the efficiency of the algorithm increases with the size of the structuring element.

1. INTRODUCTION

Over the last two decades, Mathematical Morphology (MM) has proven itself as a powerful image processing and analysis tool [1, 2]. A plethora of successful applications of MM in different fields have been presented in the bibliography. A problem arised from the early stages of MM was the high computational complexity of the basic morphological transformations, that is *dilation* and *erosion* [1]. This problem has put a brake in the application of MM in 3D image processing and analysis, which is also very promising, although the basic theory of MM is based on set theory and is not restricted to two dimensions.

Many different techniques have been proposed for implementing the basic morphological operations more efficiently than by using their definition. Many of them involve the use of parallel computers or specialized hardware. Other techniques are strictly restricted to 2D images [4, 5]. Also, most techniques are applicable only for structuring elements of specific shape or size, although the use of structuring elements of arbitrary size and shape can be very interesting in several applications [4]. For large structuring elements, decomposition in small structuring elements can be applied [3, 6], which is generally computationally intensive. One of the most interesting and efficient algorithms for the fast calculation on conventional computers of the basic morphological operations for 2D images is presented in [4]. However, that algorithm cannot be extended directly in the case of 3D images due to the use of chain coding.

The present paper proposes a fast algorithm for implementing the basic operation of Minkowski addition for the special case of binary 3D images (volumes), using 3D structuring elements of arbitrary shape and size. Basically, it introduces a suitable modification of the approach [4] that enables the extension of its basic idea in the 3D case. The use of this algorithm for all other morphological operations is straightforward, as they can all be expressed in terms of Minkowski addition. On conventional computers, this algorithm can provide a substantial reduction of the execution time in comparison to the corresponding time when the definition is used, even to less than the one twentieth in case of large structuring elements.

In the following, the theoretical background of the proposed algorithm is first presented and then the algorithm itself is described. Also, its computational complexity is analysed and some experimental results of its application are given.

2. THEORETICAL BACKGROUND

In this section, we shall give first some notations and then we shall present a theorem on which the algorithm is based.

In this paper, we are concerned about binary 3D images. A digital binary 3D image I is a mapping defined on a certain domain $D_I \subset \mathbf{Z}^3$ and taking its values in $\{0;1\}$:

$$I \left\{ \begin{array}{ccc} D_I & \rightarrow & \{0;1\} \\ p & \rightarrow & I(p) \end{array} \right. \tag{1}$$

$\mathbf{Z}^3$ denotes the *digital 3D space*. The definition domain D_I of I is generally an orthogonal parallilepid. In the framework of Mathematical Morphology, we are interested in the set of feature voxels (volume elements) of a binary 3D image, i.e. the voxels with value 1 (1-voxels), which is usually regarded as a point set in the case of the set being transformed, or as a vector set in the case of the *structuring element* [1] that is used for the morphological transformation (a 3D structuring element B can similarly be represented by a 3D binary image I_B defined in a domain $D_{I_B} \subset \mathbf{Z}^3$).

Let B be a subset of $\mathbf{Z}^3$, considered as a vector set. We denote $\check{B}$ the transposed set of B, that is its symmetric set with respect to the origin $O = (0,0)$:

$$\check{B} = \{-b : b \in B\} \tag{2}$$

We denote B_x the translated of set B with respect to the vector $x \in \mathbf{Z}^3$:

$$B_x = \{b + x : b \in B\} \tag{3}$$

We also denote B^C the complement of set B:

$$B^C = \{b \in \mathbf{Z}^3 : b \notin B\} \tag{4}$$

Let A and B be two subsets of $\mathbf{Z}^3$. Their *Minkowski addition*, denoted $A \oplus B$, and their *Minkowski subtraction*, denoted $A \ominus B$, are given by:

$$\begin{aligned} A \oplus B &= \{x \in \mathbf{Z}^3 : \exists b \in B, x - b \in A\} && (5) \\ &= \{a + b : a \in A, b \in B\} && (6) \\ A \ominus B &= \{x \in \mathbf{Z}^3 : \forall b \in B, x - b \in A\} && (7) \\ &= (A^C \oplus B)^C && (8) \end{aligned}$$

It is well known that all morphological transformations, from the simplest (dilation, erosion, opening, closing) to the more complex ones, are based on Minkowski addition and Minkowski subtraction [1]. Moreover, as derived from (8), Minkowski subtraction can be reduced to Minkowski addition. Therefore, in order to implement any morphological transformation, it suffices to implement the Minkowski addition.

The algorithm presented in the next section is based on the following easily proven theorem [4], which introduces an alternative way for calculating Minkowski addition:

THEOREM: Let X be a subset of $\mathbf{Z}^3$ and $Surf(X) \subseteq X$ the set of all surface points of X. Also, let $B \subset \mathbf{Z}^3$ be an arbitrary structuring element made of n connected components $B_1, B_2, \ldots, B_n$ and for each $i \in [1;n]$ let b_i be an arbitrary point included in B_i. Then, the following relation holds:

$$X \oplus B = \left(\bigcup_{i \in [1;n]} X_{b_i} \right) \cup (Surf(X) \oplus B) \tag{9}$$

Assuming 26-connectivity in a $3 \times 3 \times 3$ neighborhood, the set $Surf(X)$ practically includes all the voxels of X that have at least one non-feature voxel (0-voxel) in their 26-neighborhood.

3. ALGORITHM DESCRIPTION

In this section, we introduce an algorithm for calculating Minkowski addition of a 3D object X with a 3D structuring element B, based on (9). We assume that X is stored in a 3D image I defined in a domain D_I (3D array), B is stored in a 3D image I_B defined in a domain D_{I_B} and the output $X \oplus B$ is written in a 3D image I' defined in a domain $D_{I'}$ (with sufficient dimensions).

The algorithm includes three steps: surface tracking and encoding, structuring element encoding, and output calculation. Each step is described in detail below.

3.1. Surface tracking and encoding

This step aims at finding the set $Surf(X)$, that is the set of surface voxels of X, and coding it in a way suitable for the output calculation step. The proposed object surface coding is a novel one, specialized for this algorithm. The object surface is represented by voxel lists. We assume that $Surf(X)$ consists of $n(X)$ different connected surfaces $S_1, \ldots, S_{n(X)}$. The number of connected surfaces can be equal or greater than the number of connected components of X, depending on whether there are connected components with internal "holes" or not. Each S_i is coded as a list of N_{S_i} structures, where N_{S_i} is the number of voxels of S_i. Each structure contains the position of the corresponding voxel $p_{S_i,j}$, $j \in [1; N_{S_i}]$ of S_i and an array of links $d_l(p_{S_i,j}) \in [1;26]$, $l \in [1; l(p_{S_i,j})]$ to other voxels of S_i in its 26-neighborhood. A link is in fact the direction $d \in [1;26]$ of movement from the current voxel to the voxel being linked. These links are a key point in achieving the efficiency of the algorithm. The following rules are employed:

- The first voxel $p_{S_i,1}$ of each S_i is not linked from another voxel.
- Each of the other voxels $p_{S_i,j}$, $j \in [2; N_{S_i}]$ is linked from only one other voxel of S_i.
- Each voxel can have links to more than one other voxels, or to none.

Surface tracking and encoding is achieved efficiently in one scanning of D_I, by using a "burning" procedure and by utilizing proper labelling of 1-voxels to avoid repetitions in value checking. During the global scanning, if a 1-voxel is reached, then, if it is an internal voxel it is labelled with a value 3, whereas, if it is a surface voxel it is considered as the first voxel of a connected surface, which is subsequenly tracked in a burn-like manner: The first voxel is labelled with a value 2 and is put in a FIFO stack. For each voxel extracted from the stack, we examine the voxels in its 26-neighborhood; we put links to the surface voxels that have not already been linked, we label them with a value 2 and we put them on the stack, whereas the internal voxels are labelled with a value 3. When the stack is empty, all the voxels of the current connected surface have been tracked and coded and the global scanning is continued from the first voxel, so that all other connected surfaces are tracked and coded in the same way.

At the end of this step, one optional further simple scanning of D_I may be necessary in case the input 3D image I should be left unchanged. That is, all 1-voxels, which have been labelled during the first scanning, are restored to value 1.

3.2. Structuring element encoding

In order to achieve efficient output calculation, an appropriate encoding of the structuring element B is also required. As it will be seen in the third step, it is important to find and keep the sets $Surf_d(B)$, of the surface voxels of B in each direction $d \in [1; 26]$, given by:

$$Surf_d(B) = \{p \in B : p + \vec{u}_d \notin B\} \tag{10}$$

where $\vec{u}_d$ is the vector from a voxel to the voxel in its 26-neighborhood in direction d. The encoding includes the following elements:

- $n(B)$: the number of connected components of B.
- $s(B)$: the size of B, i.e. the number of 1-voxels of B.
- $\{s_d(B)\}_{d\in[1;26]}$: the sizes of the sets $Surf_d(B)$.
- $A(B)$: an array of size $s(B)$ of all voxels (vectors) of B.
- $\{A_d(B)\}_{d\in[1;26]}$: arrays of respective size $s_d(B)$ of the voxels (vectors) of the sets $Surf_d(B)$.
- $flag_B$: a variable whose value is 0 if B does not contain its center, or, otherwise, the label (i.e. the number) of the connected component of B holding the center.
- $\{p_i(B)\}_{i\in[1;n(B)]}$: array of size $n(B)$ of arbitrary voxels (vectors), such that $p_i(B) \in B_i$, $\forall i \in [1; n(B)]$, where B_i is the respective connected component of B.

The encoding of the structuring element is achieved with a similar procedure as that of tracking and encoding the set $Surf(X)$. The difference is that we do not discriminate between surface and internal voxels and that, instead of forming the encoding of $Surf(X)$, we put each 1-voxel encountered in array $A(B)$ and, if needed, in one of the arrays $A_d(B)$, $d \in [1; 26]$. Also, we easily update the other elements of the encoding during the scanning.

3.3. Output calculation

Output calculation, in fact, implements (9). We assume that $D_{I'}$ is initialized with zero values. Thus, we need only to set the 1-voxels of $D_{I'}$.

First, we form the set $\bigcup_{i\in[1;n(B)]} X_{p_i(B)}$ by assigning the value 1 to the voxels of $D_{I'}$ belonging to the set $\bigcup_{p\in X} \bigcup_{i\in[1;n(B)]} (p + p_i(B))$ Next, we form the set $Surf(X) \oplus B$ by propagating B along the voxels of $Surf(X)$, which, as it is easily proven, is equivalent to assigning the value 1 to the voxels of $D_{I'}$ belonging to the set $\bigcup_{i\in[1;n(X)]} [\bigcup_{p\in A(B)} (p_{S_i,1} + p) + \bigcup_{j\in[1;N_{S_i}]} \bigcup_{l\in[1;l(p_{S_i,j})]} \bigcup_{p\in A_{d_l(p_{S_i,j})}(B)]} (p_{S_i,j} + \vec{u}_{d_l(p_{S_i,j})} + p)]$. As it is obvious from the last expression, we make use of the fact that when propagating B from a surface voxel to another surface voxel in its neighborhood, we need only to add the voxels of the set $Surf_d(B)$, where d is the direction (the link) from the first voxel to the second. This leads to the extremely fast calculation of $Surf(X) \oplus B$. Considering also the fact that only the set $Surf(X)$ is used, instead of the entire X, we can have an idea of the efficiency of the presented algorithm.

4. ALGORITHM ANALYSIS

The efficiency of the above algorithm is the result of processing as few voxels as possible during each step of the algorithm, especially in the output calculation step as explained above. Although the step of surface tracking and encoding and the step of structuring element encoding (in case of structuring elements with large size) can require a significant percentage of the overall operations, the output calculation step is very efficiently performed, in comparison to the number of operations needed when implementing the Minkowski addition by using its definition. The same stands for the case when we use the above mentioned algorithm for implementing the dilation or erosion. Also, since the time needed for the surface tracking and encoding step for a specific 3D image is constant, it is expected that the overall time of all three steps becomes comparatively much smaller as the size of the structuring element increases.

In the following, the computational complexity is measured with the number of accesses to a voxel of a 3D image, either for examining its value, or for assigning a new value (basic operations). From the above description, we can easily show that an upper bound for the number of basic operations N_{BO} performed by the algorithm is (not including the steps of restoring the labelled 3D images to their initial values):

$$\begin{aligned} N_{BO} = N_I + (27 + n(B)) \times N_I^1 \\ +(26 + s'(B)) \times N_S + N_{I_B} \\ +(27 + n(X)) \times s(B) - n(X) \times s'(B) \end{aligned} \qquad (11)$$

where N_I is the number of voxels of I, N_I^1 is the number of 1-voxels of I, $N_S = \sum_{i=1}^{n(X)} N_{S_i}$ is the number of voxels of $Surf(X)$, N_{I_B} is the number of voxels of I_B, and $s'(B) = \max_{i \in [1;26]} s_i(B)$.

The number of basic operations N'_{BO} performed by a trivial implementation of Minkowski addition using its definition (6) is:

$$N'_{BO} = N_I + (N_{I_B} + s(B)) \times N_I^1 \qquad (12)$$

since we need to perform an entire scanning of I and, for every 1-voxel of I, to scan I_B and assign 1 at the appropriate voxel of the output I' for each 1-voxel of I_B. Usually, in (11) and (12) the terms related with N_I^1 are the most significant. For structuring elements larger than $5 \times 5 \times 5$, it is $27 + n(B) \ll N_{I_B} + s(B)$, which explains the efficiency of the presented algorithm.

In Fig. 1, we give some experimental results of the application of the presented algorithm in the calculation of the dilation of a test binary 3D image with structuring elements of different size. We compare the execution time of the proposed algorithm and of that using the definition of Minkowski addition (6). The data of Fig. 1 are also illustrated in Fig. 2. As derived from the experimental results, for a small structuring element of size $3 \times 3 \times 3$ the overall execution time is comparable for the two cases. The efficiency of the algorithm is obvious for structuring elements of size $5 \times 5 \times 5$ or larger. Undoubtedly, the larger the structuring element, the greater the gain if the present algorithm is used. Considering only the execution time of the output calculation step, this is much smaller than that of the case when the definition is used, even for a small structuring element of size $3 \times 3 \times 3$. This fact reveals also the gain of using the proposed algorithm in an application where e.g. a 3D image needs to be dilated successively by different small structuring elements; in such an application, the surface tracking and encoding step, whose execution time dominates over that of the output calculation step for small structuring elements, needs to be performed only once in the beginning.

SE size	t_1	t'_1	t_2
$3 \times 3 \times 3$	2.83s	0.23s	2.13s
$5 \times 5 \times 5$	3.05s	0.44s	7.71s
$7 \times 7 \times 7$	3.46s	0.86s	20.03s
$9 \times 9 \times 9$	4.13s	1.52s	41.83s
$11 \times 11 \times 11$	5.12s	2.74s	76.20s
$13 \times 13 \times 13$	6.42s	4.05s	127.44s
$15 \times 15 \times 15$	8.17s	5.84s	195.36s
$17 \times 17 \times 17$	11.22s	8.44s	291.92s

(a)

SE size	$\frac{t_1}{t_2} \cdot 100\%$	$\frac{t'_1}{t_2} \cdot 100\%$
$3 \times 3 \times 3$	132.86%	10.80%
$5 \times 5 \times 5$	39.56%	5.70%
$7 \times 7 \times 7$	17.27%	4.30%
$9 \times 9 \times 9$	9.87%	3.63%
$11 \times 11 \times 11$	6.72%	3.60%
$13 \times 13 \times 13$	5.04%	3.18%
$15 \times 15 \times 15$	4.18%	2.99%
$17 \times 17 \times 17$	3.84%	2.89%

(b)

Figure 1: (a) Execution times of the dilation of a $128 \times 128 \times 128$ binary 3D image with cubic structuring elements (SE) of different size. t_1: execution time using the presented algorithm. t'_1: part of t_1 corresponding to the output calculation step only. t_2: execution time using the definition of Minkowski addition. (on a Silicon Graphics Indy MIPS R4400 200MHz workstation running IRIX 5.3), (b) Comparison between the above execution times.

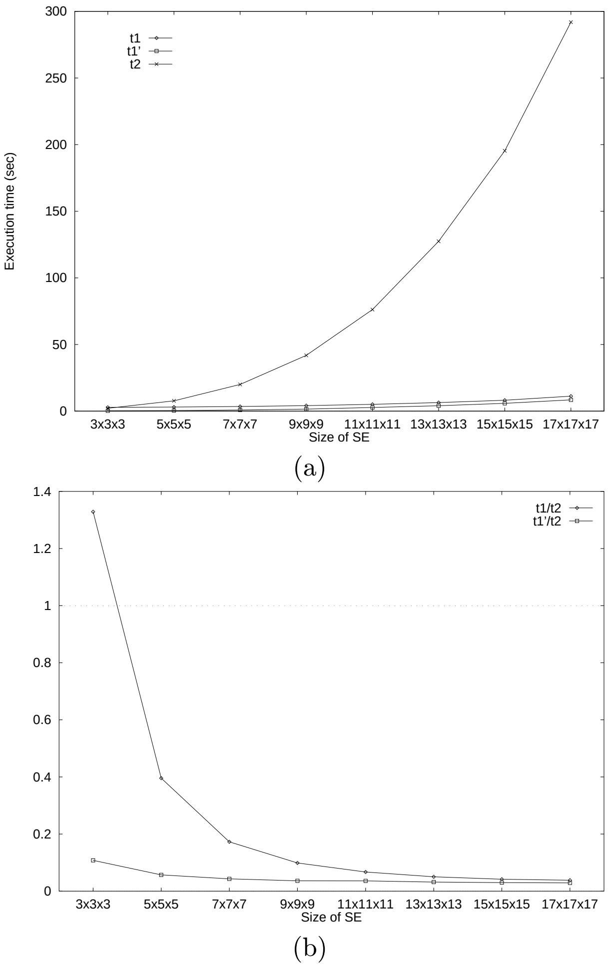

Figure 2: (a) Diagram illustrating the values of Fig. 1a. (b) Diagram illustrating the values of Fig. 1b.

5. CONCLUSION

In this paper, we presented a very efficient algorithm for the implementation of Minkowski addition for the special case of 3D sets represented by binary 3D images, using 3D structuring elements. It can easily be modified for the implementation of all the other morphological transformations. The algorithm does not pose any restrictions on the shape or the size of the structuring elements. The efficiency of the algorithm was analysed and some results of its application were presented. As it was made obvious, the efficiency of the algorithm is significant for 3D structuring elements larger than $5 \times 5 \times 5$.

6. REFERENCES

[1] J. Serra, *Image Analysis and Mathematical Morphology*, Academic Press, London, 1982.

[2] J. Serra, ed., *Image Analysis and Mathematical Morphology, Part II: Theoretical Advances*, Academic Press, London, 1988.

[3] I. Pitas, A.N. Venetsanopoulos, *Nonlinear Digital Filters: Principles and Applications*, Kluwer Academic Publishers, 1990.

[4] L. Vincent, “Morphological transformations of binary images with arbitrary structuring elements”, *Image Procesing*, vol. 22, no. 1, pp. 3–23, January 1991.

[5] L.J. Piper and J.-Y. Tang, “Erosion and dilation of binary images by arbitrary structuring elements using interval coding”, *Pattern Recognition Letters*, pp. 201–209, April 1989.

[6] H. Park, R.T. Chin, “Decomposition of Arbitrarily Shaped Morphological Structuring Elements”, *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 17, no. 1, pp. 2–15, January 1995.