D. Computational Issues

To compare the time performance, the algorithm has been implemented and tested on the SIMD parallel machines Connection Machine CM-200 with 4096 one-bit processing elements (PE's) [17] and MasPar MPP 12 000 with 4 096 four-bit PE's [18], the implementation being as intuitively efficient as possible. The comparisons were made against the best parallel implementations of the Zhang and Suen, Wang and Zhang, Chin *et al.*, and Holt and Stewart [7] which were available to us at the moment we wrote this paper. The thinning algorithms were tested on the 64×64 digital patterns "X," "hideogram," "A," and "H." The times, expressed in seconds, are shown in Table II. The analysis of the tables makes evident that the proposed two-subcycle algorithm and the Wang algorithm outperform the remaining. Nevertheless, the average computation time (AT) of the proposed algorithm is lower than the Wang algorithm (≈60% on CM-200 and ≈85% on MasPar). In particular, we observe in our experiments that the number of iterations required by the proposed algorithm is equal to about the half of the maximum width of the input picture; a characteristic of the proposed parallel algorithm which makes it near optimal.

V. Concluding Remarks

In this paper, we have reported a new parallel thinning algorithm with two subcycles, characterized by templates of dimension 3×4 and 4×3 for the first subcycle, while a 3×3 template is used in the second. The algorithm has been tested on different patterns and the results compared with those obtained by applying other algorithms of analogous nature. We achieve better results according to the degree of 8-connectedness (perfect skeleton), accuracy, degree of erosion, stability under pattern rotation, and boundary noise sensitivity. Timings taken on the CM-200 and MasPar MPP-12 000 also show the time complexity to be very low for the proposed algorithm.

References

[1] R. T. Chin, H.-K. Wan, D. L. Stover, and R. D. Iverson, "A one-pass thinning algorithm and its parallel implementation," *Comput., Vis., Graph., Image Process.*, vol. 40, pp. 30–40, 1987.

[2] Y.-S. Chen and W.-H. Hsu, "A modified fast parallel algorithm for thinning digital patterns," *Pattern Recognit. Lett.*, vol. 7, no. 2, pp. 99–106, 1988.

[3] Y.-S. Chen and Y.-T. Yu, "Thinning approach for noisy digital patterns," *Pattern Recognit.*, vol. 29, no. 11, pp. 1847–1862, 1996.

[4] E. S. Deutsch, "Thinning algorithms on rectangular, hexagonal and triangular arrays," *Commun. ACM*, vol. 15, pp. 827–837, 1972.

[5] Z. Guo and R. W. Hall, "Parallel thinning with two-subiteration algorithms," *J. ACM*, vol. 32, no. 3, pp. 359–373, 1989.

[6] W. L. M. Hoeks, "Performance evaluation for thinning algorithms with respect to boundary noise," in *Proc. Int. Conf. Image Processing Applications*, Maastricht, NE, 1992, pp. 250–253.

[7] C. Holt and A. Stewart, "A parallel thinning algorithm with fine grain subtasking," *Parallel Comput.*, vol. 10, pp. 329–334, 1989.

[8] B. K. Jang and R. T. Chin, "One-pass parallel thinning: Analysis, properties, and qualitative evaluation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 14, no. 11, 1992.

[9] R. Krishnapuram and L.-F. Chen, "Implementation of parallel thinning algorithms using recurrent neural networks," *IEEE Trans. Neural Networks*, vol. 4, no. 1, 1993.

[10] L. Lam, S.-W. Lee, and C. Y. Suen, "Thinning methodologies—A comprehensive survey," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 14, pp. 869–885, Sept. 1992.

[11] L. Lam and C. Y. Suen, "An evaluation of parallel thinning algorithms for character recognition," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 17, Sept. 1995.

[12] P. K. Saha, B. B. Chaudhuri, and D. D. Majumder, "A new shape preserving parallel thinning algorithm for 3D digital images," *Pattern Recognit.*, vol. 30, no. 12, pp. 1939–1955, 1997.

[13] F. Y. Shih and W.-T. Wong, "Fully parallel thinning with tolerance to boundary noise," *Pattern Recognit.*, vol. 27, no. 12, pp. 1677–1695, 1994.

[14] R. Stefanelli and A. Rosenfeld, "Some parallel thinning algorithms for digital pictures," *J. ACM*, vol. 18, no. 2, pp. 255–264, 1971.

[15] S. Suzuki and K. Abe, "Binary picture thinning by an iterative parallel two-subcycle operation," *Pattern Recognit.*, vol. 10, no. 3, pp. 297–307, 1987.

[16] H. Tamura, "A comparison of line thinning algorithms from a digital geometry viewpoint," in *Proc. 4th Int. Conf. Pattern Recognition*, 1978, pp. 715–719.

[17] "Technical Summary, Connection Machine Model CM-200," Thinking Machines Corporation, Cambridge, MA, Ver. 6.1, Oct. 1991.

[18] "Technical Summary, MasPar Machine Model MPP-12 000," Digital Equipment Corporation, Maynard, MA, Ver. 1.0, Jan. 1992.

[19] P. S. P. Wang and Y. Y. Zhang, "A fast and flexible thinning algorithm," *IEEE Trans. Comput.*, vol. 38, pp. 741–745, May 1989.

[20] T. Y. Zhang and C. Y. Suen, "A fast parallel algorithm for thinning digital patterns," *Commun. ACM*, vol. 27, no. 3, pp. 236–239, 1984.

A Fast Implementation of 3-D Binary Morphological Transformations

Nikos Nikopoulos and Ioannis Pitas

***Abstract*—This paper proposes a fast algorithm for implementing the basic operation of Minkowski addition for the special case of binary three-dimensional (3-D) images, using 3-D 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 analyzed and some experimental results of its application are presented. As shown, the efficiency of the algorithm increases with the size of the structuring element.**

***Index Terms*—Fast algorithm, image processing, mathematical morphology, 3-D binary morphological transformations.**

I. 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 *dilation* and *erosion* [1]. This problem has hindered the application of MM in three-dimensional (3-D) image processing and analysis, which is very promising, since the basic theory of MM is based on set theory and can be easily extended to three and higher 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 restricted to two-dimensional (2-D) images [4], [5]. Also, most techniques are applicable only for

Manuscript received October 23, 1997; revised March 10, 1998. The associate editor coordinating the review of this paper and approving it for publication was Prof. Scott T. Acton.

The authors are with the Department of Informatics, University of Thessaloniki, 54006 Thessaloniki, Greece.

Publisher Item Identifier S 1057-7149(00)01154-4.

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 2-D images is presented in [4]. However, that algorithm cannot be extended directly to 3-D images, because chain coding has not an equivalent in three dimensions.

The present paper proposes a fast algorithm for implementing the basic operation of Minkowski addition for the special case of binary 3-D images (volumes), using 3-D structuring elements of arbitrary shape and size. Basically, it introduces a suitable modification of the approach [4] that enables its extension in the 3-D 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 presented first and the algorithm itself is described subsequently. Its computational complexity is analyzed and some experimental results of its application are given.

II. 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 3-D images. A digital binary 3-D image I is a mapping defined on a certain domain $D_I \subset \mathbf{Z}^3$ and taking its values in $\{0;1\}$:

$$I \begin{cases} D_I & \rightarrow & \{0;1\} \\ p & \rightarrow & I(p) \end{cases} \tag{1}$$

where $\mathbf{Z}^3$ denotes the *digital 3D space* and p denotes an arbitrary point *(voxel)* belonging to D_I. The definition domain D_I of I is generally an orthogonal parallilepid. In the framework of MM, we are interested in the set of feature voxels (volume elements) of a binary 3-D 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]. A 3-D structuring element B can similarly be represented by a 3-D 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 $O = (0,0)$:

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

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

$$A \oplus B = \{x \in \mathbf{Z}^3\colon \exists b \in B, x - b \in A\} \tag{5}$$

$$= \{a + b\colon a \in A, b \in B\} \tag{6}$$

$$A \ominus B = \{x \in \mathbf{Z}^3\colon \forall b \in B, x - b \in A\} \tag{7}$$

$$= (A^C \oplus B)^C. \tag{8}$$

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.

1) 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 $B_1, B_2, \cdots, B_n$ 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 × 3 × 3 neighborhood, the set $Surf(X)$ practically includes all the voxels of X that have at least one nonfeature voxel (0-voxel) in their 26-neighborhood. The above theorem is an extension in the 3-D case of the corresponding theorem for the 2-D case that is proved in [4]. This extension is mathematically trivial since the proof is based on set theory. Specifically, it suffices to change $\mathbf{Z}^2$ with $\mathbf{Z}^3$ and the notion of contour with that of surface. Therefore, the proof of the theorem was deliberately omitted.

III. Algorithm Description

In this section, we introduce an algorithm for calculating Minkowski addition of a 3-D object X with a 3-D structuring element B, based on (9). We assume that X is stored in a 3-D image I defined in a domain D_I (3-D array), B is stored in a 3-D image I_B defined in a domain D_{I_B} and the output $X \oplus B$ is written in a 3-D image I' defined in a domain $D_{I'}$. The algorithm includes three steps: surface tracking and encoding, structuring element encoding, and output calculation. Each step is described in detail subsequently.

A. 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 $S_1, \cdots, S_{n(X)}$. 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, $i \in [1; n(X)]$ is coded as a list of N_{S_i} structures, where N_{S_i} is the number of voxels belonging to S_i. Each structure corresponding to S_i contains the position (coordinates) 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. The index j of the voxel $p_{S_i,j}$ denotes the order in which it is detected to belong to S_i during the tracking of the connected surface S_i, which is explained

below in detail. $l(p_{S_i,j})$ denotes the number of links corresponding to the voxel $p_{S_i,j}$. 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 are achieved efficiently in one scanning of D_I, by using a "burning" procedure to track each connected surface and determine the links between its voxels and by utilizing proper labeling 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 labeled with a label 3, whereas, if it is a surface voxel it is the first voxel of a new connected surface, which is subsequenly tracked using a "burning" procedure before continuing the global scanning from the first voxel of the tracked connected surface. During the "burning" procedure all the voxels of the current connected surface are labeled with a label 2 to ensure that the next surface voxel with value 1 that will be encountered during global scanning will belong to a new connected surface.

The "burning" procedure that tracks a connected surface is achieved via a FIFO stack filled with voxels belonging to it. The first voxel of the connected surface is labeled with a label 2 and initializes the FIFO stack. We proceed by extracting voxels from the stack until it is empty (by the time we try to extract a voxel). For each voxel extracted from the stack, we examine the 1-voxels in its 26-neighborhood: If we encounter a surface voxel with value 1, it is a voxel belonging to the same connected surface that has not already been linked. Thus, we add a link to it from the current voxel extracted from the stack, we label it with a label 2 and we put it in the FIFO stack; if we encounter an internal voxel with value 1, we label it with a label 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.

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

B. 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

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

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 size 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.

C. Output Calculation

Output calculation 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 $\cup_{i\in[1;n(B)]}\ X_{p_i(B)}$ by assigning the value 1 to the voxels of $D_{I'}$ belonging to the set $\cup_{p\in X}\ \cup_{i\in[1;n(B)]}\ (p+p_i(B))$. 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 $\cup_{i\in[1;n(X)]}\ [\cup_{p\in A(B)}\ (p_{S_i,1}+p) + \cup_{j\in[1;N_{S_i}]}\ \cup_{l\in[1;l(p_{S_i,j})]}\ \cup_{p\in A_{d_l(p_{S_i,j})}(B)}(p_{S_i,j} + \vec{u}_{d_l(p_{S_i,j})} + p)]$. 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.

IV. Algorithm Analysis

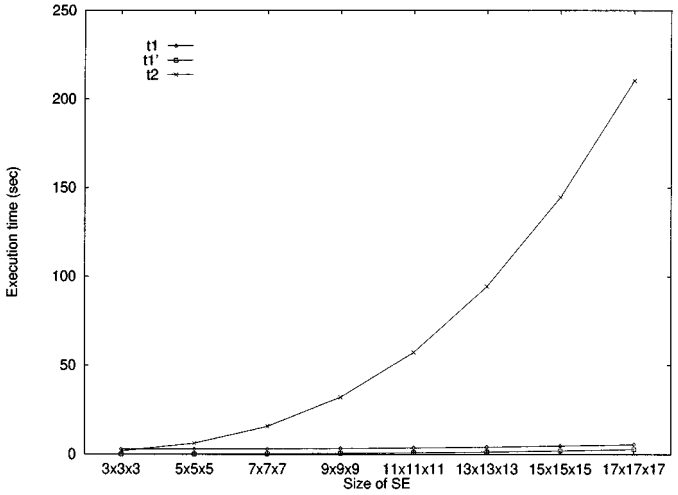
The efficiency of the above algorithm is the result of processing as few voxels as possible during each step of the algorithm, especially in the output calculation step, as previously explained. Although the step of surface tracking and encoding and the step of structuring element encoding (in case of 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 3-D 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 3-D 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 labeled 3-D images to their initial values)

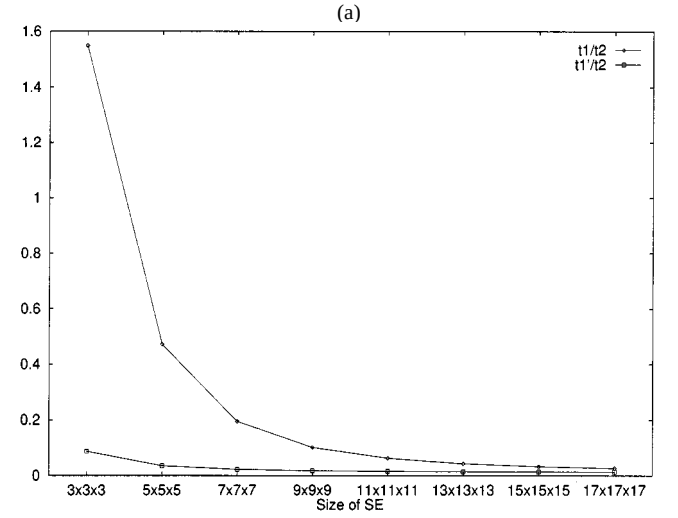
$$\begin{aligned} N_{BO} = {} & N_I + (27 + n(B)) \times N_I^1 + (26 + s'(B)) \times N_S + N_{I_B} \\ & + (27 + n(X)) \times s(B) - n(X) \times s'(B) \end{aligned} \tag{11}$$

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

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



(a)



(b)

Fig. 1. (a) Execution times of the dilation of a 128 × 128 × 128 binary 3-D image with structuring elements (SE) of different grid size. For every grid size, the SE is the union of three rhombuses having their center on the center of the grid and their corners on the edges of the grid and being parallel to the xy, yz, zx planes respectively. 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 200 MHz workstation running IRIX 5.3) and (b) comparison between the above execution times.

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 × 5 × 5, it is $27 + n(B) \ll N_{I_B} + s(B)$, which explains the efficiency of the presented algorithm.

In Fig. 1, we graphically present some experimental results of the application of the presented algorithm in the calculation of the dilation of a 128 × 128 × 128 binary 3-D image with a structuring element having nontrivial shape, for different structuring element grid sizes. We compare the execution time of the proposed algorithm and of that using the definition of Minkowski addition (6). As derived from the experimental results, for a small structuring element of size 3 × 3 × 3 the overall execution time is comparable for the two cases. The efficiency of the algorithm is obvious for structuring elements of size 5 × 5 × 5 or larger. Undoubtedly, the larger the structuring element, the greater the gain if the present algorithm is used. The execution time of the output calculation step only is much smaller than that of the case when the definition is used, even for a small structuring element of size 3 × 3 × 3. This fact reveals also the gain obtained by using the proposed algorithm in an application where, e.g., a 3-D 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 at the beginning of the algorithm.

V. Conclusion

In this paper, we presented a very efficient algorithm for the implementation of Minkowski addition for the special case of 3-D sets represented by binary 3-D images, using 3-D 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 analyzed and some results of its application were presented. As it was made obvious, the efficiency of the algorithm is significant for 3-D structuring elements larger than 5 × 5 × 5.

It is stated in the introduction that the presented algorithm is a suitable extension in the 3-D case of the algorithm presented in [4]. This extension was not trivial and required a new approach (novel surface and structuring element encoding scheme and respective new structuring element propagation rules). This approach is the contribution of this paper.

References

[1] J. Serra, *Image Analysis and Mathematical Morphology*. London, U.K.: Academic, 1982.
[2] ——, *Image Analysis and Mathematical Morphology, Part II: Theoretical Advances*. London, U.K.: Academic, 1988.
[3] I. Pitas and A. N. Venetsanopoulos, *Nonlinear Digital Filters: Principles and Applications*. Norwell, MA: Kluwer, 1990.
[4] L. Vincent, "Morphological transformations of binary images with arbitrary structuring elements," *Image Process.*, vol. 22, pp. 3–23, Jan. 1991.
[5] L. J. Piper and J.-Y. Tang, "Erosion and dilation of binary imaes by arbitrary structuring elements using interval coding," *Pattern Recognit. Lett.*, pp. 201–209, Apr. 1989.
[6] H. Park and R. T. Chin, "Decomposition of arbitrarily shaped morphological structuring elements," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 17, pp. 2–15, Jan. 1995.