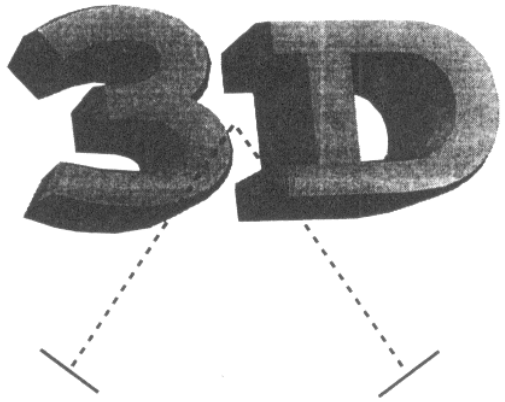
PROCEEDINGS

INTERNATIONAL WORKSHOP
ON STEREOSCOPIC AND
THREE DIMENSIONAL IMAGING

IWS3DI'95



September 6–8, 1995
Conference Centre "Petros M. Nomikos"
Santorini, Greece

Sponsors:
European Commission/Directorate General XIII
Greek General Secretariat of Research and Technology
Siemens AG (Germany) and Intracom S.A. (Greece)

LOGIC OPERATIONS ON 3-DIMENSIONAL VOLUMES USING BLOCK REPRESENTATION

Iraklis M. Spiliotis and Basil G. Mertzios

Automatic Control Systems Laboratory
Electrical and Computer Engineering Dept.
Democritus University of Thrace
Xanthi 67100
HELLAS

tel: +30-541-79559, 79511
fax: +30-541-26473
e-mail: spiliot@demokritos.cc.duth.gr
mertzios@demokritos.cc.duth.gr

ABSTRACT

A method for the execution of logic operations in 3-D volumes is presented. The 3-D volumes are represented by rectangular parallelepipeds, with edges parallel to the axes and will be called blocks in the terminology of this paper. The block representation has been used for the fast execution of image analysis and pattern recognition algorithms. It has been successfully used for the fast computation of moments feature extraction and skeletonization of 2-D images.

I. BLOCK REPRESENTATION

The 3-D volumes are represented by rectangular parallelepipeds, with edges parallel to the axes and will be called *blocks* in the terminology of this paper. Figure 1, shows a block. At the extreme case one point is the minimum parallelepiped. Consider a set that contains as members all the nonoverlapping blocks of a specific volume, in such a way that no other block can be extracted from the volume (or equivalently each point of the volume belongs to one and only one block). This set represents the 3-D volume without loss of information. It is always feasible to represent a 3-D volume as a set of all the nonoverlapping blocks. We call this representation of the 3-D volumes, *block representation* [1]-[4].

Consider the volume V which is represented by k blocks. Then the following notation is used:

$$V(x,y,z) = \{b_i : i = 0,1,\ldots,k-1\} \tag{1}$$

For the representation of each block, it is adequate to store the coordinates of two points in the 3-D space. These points should be two corner points P_{1,b_i}, P_{2,b_i} of the block that are symmetrically located according to the center of the block, therefore

$$b_i = (x_{1,b_i}, y_{1,b_i}, z_{1,b_i}, x_{2,b_i}, y_{2,b_i}, z_{2,b_i}) \tag{2}$$

For the simplicity of the relative algorithms that may be implemented for block represented volumes and without loss of any generality the following requirement should be satisfied:

$$x_{1,b_i} \le x_{2,b_i}, \quad y_{1,b_i} \le y_{2,b_i}, \quad z_{1,b_i} \le z_{2,b_i} \tag{3}$$

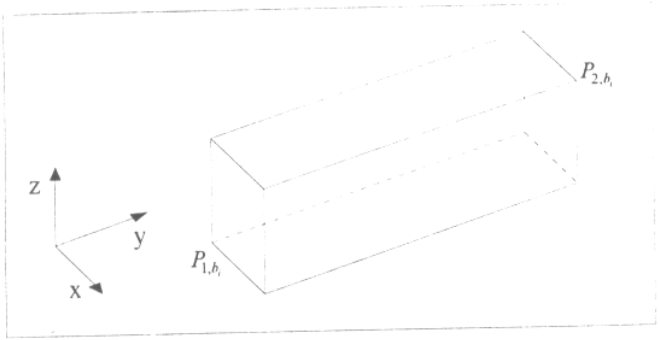


Figure 1. A 3-D block.

II. LOGIC OPERATIONS

Logic operations are performed in block represented volumes. Consider the volume V_1 with k_1 blocks and the volume V_2 with k_2 blocks. Since each volume is represented by a set of nonoverlapping blocks, binary operations among the points of the volumes may be substituted by binary operations on the blocks. The results are always block represented volumes. Therefore, the relevant algorithms can be directly applied in cases where repetitive logic operations should be executed. A new logic function, the COMAND function (COMplementary AND) has also been defined, which is used as an auxiliary operator for the execution of the OR, XOR and NOT operations in block represented volumes.

The COMAND function

Definition 1: COMAND logic function

Consider the ordered pair of binary or logic operands (a,b). The COMAND logic function is defined by the formula:

$$\text{COMAND}(a,b) = a \text{ XOR } (a \text{ AND } b) \tag{4}$$

■

Definition 2: COMAND operation among binary arrays

Consider the 1-D arrays of binary operands $A(i) = [a_0, a_1, \ldots, a_{n-1}]$ and $B(i) = [b_0, b_1, \ldots, b_{n-1}]$. The COMAND operation is defined by:

$$C(i) = \text{COMAND}(A(i), B(i)) = [c_0, c_1, \ldots, c_{n-1}] \tag{5}$$

where

$$c_i = \text{COMAND}(a_i, b_i) = a_i \text{ XOR } (a_i \text{ AND } b_i) \tag{6}$$

■

The COMAND$(A(i),B(i))$ operation, among the 1-D arrays $A(i)$, $B(i)$ produces an equidimensional array $C(i)$, whose logical one elements are those of $A(i)$ that do not belong to $B(i)$. For example, let the 1-D arrays $A(i)$ = [1, 0, 1, 0], $B(i)$ = [1, 1, 0, 0]. Then C(i)=COMAND$(A(i),B(i))$=[0, 0, 1, 0].

The COMAND operation among multidimensional arrays is similarly defined.

According to Definition 2, it is clear that the commutative property in the COMAND operation is not preserved, i.e.:

$$\text{COMAND}(a,b) \neq \text{COMAND}(b,a)$$
$$\text{COMAND}(A(i), B(i)) \neq \text{COMAND}(B(i), A(i)) \tag{7}$$

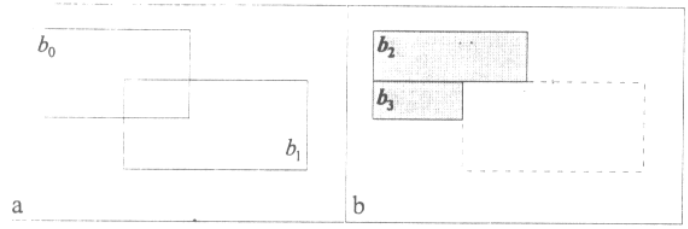


Figure 2. (a). The relative positions of the blocks b_0, b_1 according to a rectangular grid and (b). The execution of the function COMAND(b_0, b_1), produces the blocks b_2, b_3, where the result is visualized with gray tone.

The execution of the COMAND function is easily implemented with blocks that represent 2-D arrays, as it is illustrated in Fig. 2. For convenience and for better visualization results, in all the Figures of this Section, the blocks are displayed as 2-D structures, without the loss of the generality for 3-D or greater dimensions.

Consider the implementation of the COMAND(f_1, f_2) operation between two block represented volumes V_1 and V_2. Due to the nature of the COMAND operation, the following point has to be taken into account: A number of sub-blocks is derived when the execution of the operation COMAND is applied between a block of the first volume and the first block of the second volume. The execution of the COMAND operation between these sub-blocks and the next block of the second volume results to new sub-blocks e.t.c. Therefore, it is appropriate to formulate temporary blocks in order to store these sub-blocks until the execution of the COMAND operation with all the blocks of the second volume is terminated. This procedure is repreated for each block of the first volume.

In the sequel, the algorithm for the implementation of the COMAND operation between block represented volumes is given.

Algorithm: Implementation of the COMAND(f_1, f_2) operation

Step 1: For each block of the volume V_1, b_{i,V_1}, $i \in [0, k_1 - 1]$.

Step 2: Label the block b_{i,V_1} of the volume V_1 in the temporary blocks: $\hat{b}_{i,0} = b_{i,V_1}$.

Step 3: Execute the COMAND operation between each one of the temporary blocks $\hat{b}_{i,j}$ and the blocks of the volume V_2. The resulted sub-blocks are the updated temporary blocks of $\hat{b}_{i,j}$.

Step 4: Place the temporary blocks to the resulted blocks.

Step 5: Clear the temporary blocks. Go to Step 1.

■

The above Algorithm is illustrated by the following example:

Example 1.

Consider the block represented binary volumes $V_1 = \{b_{0,V_1}\}$ and $V_2 = \{b_{0,V_2}, b_{1,V_2}\}$ of Figure 3. For the execution of the function COMAND(V_1, V_2), the block b_{0,V_1} is copied in the temporary blocks, $\hat{b}_{0,0} = b_{0,V_1}$. The execution of the function COMAND($\hat{b}_{0,0}$, b_{0,V_2}) results to the temporary blocks $\hat{b}_{0,1}$, $\hat{b}_{0,2}$ and to the elimination of the block $\hat{b}_{0,0}$. The execution of the COMAND($\hat{b}_{0,2}$, b_{1,V_2}) results to the temporary blocks $\hat{b}_{0,2}$, $\hat{b}_{0,2}$, $\hat{b}_{0,5}$ and to the elimination of the block $\hat{b}_{0,2}$. Finally, the temporary blocks are placed in the resulted blocks and the resulted volume V_r is comprised of the blocks $\{b_{0,r}, b_{1,r}, b_{2,r}, b_{3,r}\}$, where $b_{0,r} = \hat{b}_{0,1}$, $b_{1,r} = \hat{b}_{0,3}$, $b_{2,r} = \hat{b}_{0,4}$ and $b_{3,r} = \hat{b}_{0,5}$.

In the sequel we describe the implementation of the OR, AND, XOR and NOT operations among the blocks in block represented volumes.

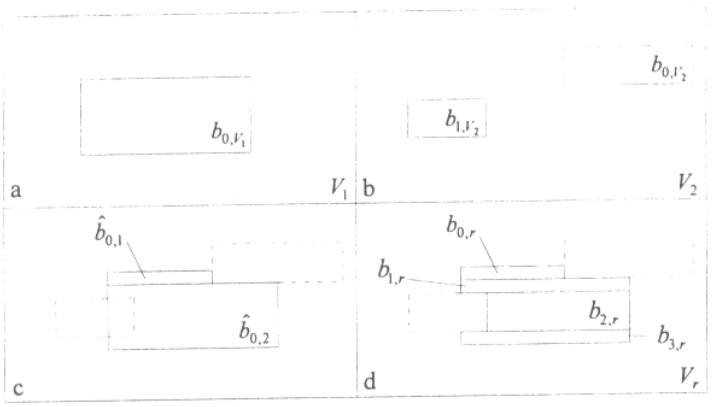


Figure 3. Implementation of the operation COMAND(V_1, V_2). (a). The volume V_1. (b). The volume V_2. (c). The execution of the COMAND($\hat{b}_{0,0}$, b_{0,V_2}). (d). The execution of the COMAND($\hat{b}_{0,2}$, b_{1,V_2}) and the final result.

The OR operation

The OR operation is a union region operation and is easily implemented in block represented volumes. Care has to be taken to avoid the existence of overlapping blocks in the results. The use of the COMAND operation guarantees this goal.

Consider the Figure 4. The application of the OR operation between the blocks b_0 and b_1, produces the blocks b_2, b_3, b_4 and is executed according to the following formula:

$$b_0 \text{ OR } b_1 = \text{COMAND}(b_0, b_1) \bigcup b_1 \tag{8}$$

where the symbol of the set union $\bigcup$ means that the block b_1 is contained in the results.

The OR operation between block represented volumes, may be implemented as follows: Initially the output volume is comprised of the blocks of the first volume V_1. Only the points of V_2 that do not exist in the volume V_1 should be included in the final output volume, in order to avoid the existence of overlapping blocks. This means that new blocks have to be defined, from the blocks of the second volume V_2, which include all the points of the second volume V_2, that do not correspond to points of the first volume V_1. The use of the COMAND operation achieves the above part of the OR operation.

Thus, the OR operation between the block represented volumes V_1 and V_2, is the extension of the OR operation between two blocks and is implemented according to the formula:

$$V_1 \text{ OR } V_2 = \text{COMAND}(V_1, V_2) \bigcup V_2 \tag{9}$$

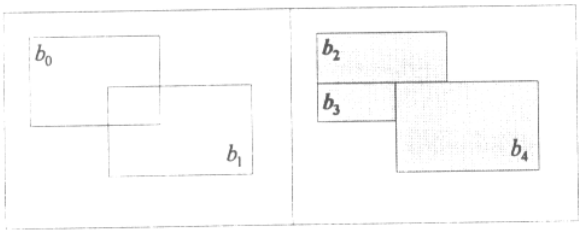


Figure 4. The execution of the OR operation between the blocks b_0 and b_1.

The AND operation

The AND operation between the block represented volumes V_1 and V_2 is implemented according to the formula:

$$V_1 \text{ AND } V_2 = \bigcup_{i\in[0,k_1-1]} \bigcup_{j\in[0,k_2-1]} \left(b_{i,V_1} \text{ AND } b_{j,V_2}\right), \tag{10}$$

where b_{m,V_n} denotes the m-th block of the n-th volume. The operation AND between the two blocks results to a new block, defined by its coordinates as follows:

$$b_{i,V_1} \text{ AND } b_{j,V_2} = \left(\max(x_{1,b_{i,V_1}}, x_{1,b_{j,V_2}}), \min(x_{2,b_{i,V_1}}, x_{2,b_{j,V_2}}), \max(y_{1,b_{i,V_1}}, y_{1,b_{j,V_2}}), \min(y_{2,b_{i,V_1}}, y_{2,b_{j,V_2}})\right) \tag{11}$$

However, in some cases the blocks resulted from the AND operation among different pairs of blocks, may be connected. To this end, each one of the resulted blocks should be checked with all the others, in order to determine the pairs of blocks that can be handled as one block.

The XOR operation

The COMAND operation is used for the execution of the XOR operation in block represented volumes. The XOR operation produces the non common points of the two blocks, or equivalently of the two volumes.

Consider the two blocks b_0 and b_1 of Figure 5. The XOR operation produces the blocks b_2, b_3, b_4, b_5 and is implemented according to the formula:

$$b_0 \text{ XOR } b_1 = \text{COMAND}(b_0, b_1) \cup \text{COMAND}(b_1, b_0) \tag{12}$$

The execution of the XOR operation between the volumes V_1 and V_2, is the extension of the execution of the XOR operation between two blocks and is implemented according to the formula:

$$V_1 \text{ XOR } V_2 = \text{COMAND}(V_1, V_2) \cup \text{COMAND}(V_2, V_1) \tag{13}$$

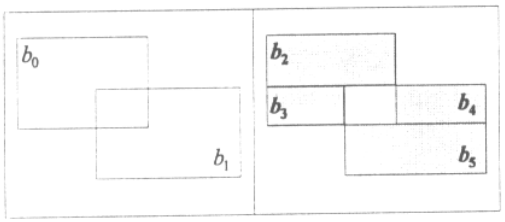


Figure 5. The implementation of the XOR operation between the blocks b_0, b_1.

The NOT operation

The NOT operation cannot be executed directly, since the block representation carries only information concerning volume regions and not empty regions. However, the NOT operation in block represented volumes is implemented by the definition of an volume V_Θ with the same length, width and height as the input volume, containing only one block Θ that covers all the volume. The execution of the operation COMAND(Θ,V) results to the inverse of the volume V, as it is described:

$$\text{NOT}(V) = \text{COMAND}(V_\Theta, V) \tag{14}$$

Consider the block b_0 in Figure 6. The NOT operation of the block b_0 produces the blocks b_1, b_2, b_3 and b_4.

Figure 6. The execution of the NOT operation for the block b_0.

III. CONCLUSIONS

The block representation may be seen as a physical model for the representation of 2-D, 3-D or volumes of greater dimensions. Each block is represented with $2N$ integers, where N is the dimension of the volume, the coordinates of two corners lying symmetricaly according to the center of the volume. The use of block representation allows the implementation of novel and fast algorithms which are based or may be expressed on the execution of the logic operations of 3-D volumes. These algorithms are directly extensible to N-D volumes.

REFERENCES

[1] I.M. Spiliotis and B.G. Mertzios, "Real-Time computation of statistical moments on binary images using block representation", *4th International Workshop on Time-Varying Image Processing and Moving Object Recognition*, pp. 27-34, Florence, Italy, June 10-11, 1993.

[2] I.M. Spiliotis, D.A. Mitzias and B.G.Mertzios, "A skeleton-based hierarchical system for learning and recognition", *Proceedings of MTNS 93, International Symposium on the Mathematical Theory of Networks and Systems*, pp. 873-878, Regensburg, Germany,

[3] I.M. Spiliotis and B.G. Mertzios, "Real-time computation of two-dimensional moments on binary images using image block representation", *IEEE Trans. on Image Processing*. Submitted for publication.

[4] I.M. Spiliotis and B.G. Mertzios, "Pattern analysis on binary images using image block representation", *Pattern Recognition*. Submitted for publication.