An Object-Oriented Implementation of Morphological Operations Using Element Vector Representation

Zoran Đukić[1] and Sven Lončarić[2]

[1] Institute "Ruđer Bošković", Zagreb, Croatia
[2] Department of Electronic Systems and Information Processing, Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia

A novel implementation of morphological operations is proposed in this paper. Major benefit of the implementation is the independence of computation algorithms to space dimension. This is achieved using a sorted linked list of elements for the set representation instead of conventional raster representation. A set compression method is used for smaller memory requirements. Since the compressed elements act as subsets, the set operations are performed on the subsets instead of the single elements resulting in the computation speed up. The implemented morphological set operations include erosion, dilation, opening, and closing. The software is developed in GNU C++, using object-oriented paradigm that enables easy reuse of the designed software components.

Keywords: Mathematical morphology, object-oriented, class architecture, linked list, compression.

1. Introduction

The mathematical morphology provides a set of operations that are useful for processing and decomposition of shapes in arbitrary dimensions. Morphological operations are defined in terms of set-theoretic operations like union and intersection. Such a system of operations and their compositions are useful for shape decomposition, description, and matching [2]. The primary morphological operations are dilation and erosion. The morphological operations of opening and closing are defined in terms of dilation and erosion. The definitions of the morphological operations can be found in [3,6]. A novel implementation for the computation of the morphological operations is proposed in this paper. The proposed implementation uses explicit element representation in the vector form and sorted and compressed linked list for set representation, as opposed to conventional raster implementations [9].

2. Element Vector Representation for Mathematical Morphology

The conventional representation of images by two-dimensional (2–D) array storage structure, well known as a raster representation, brings two major limitations:

- Algorithms for calculating morphological operations and program code are not invariant to space dimension;
- Loss of information if the resulting image size exceeds 2–D array boundaries.

The definitions of morphological operations, based on the operations on the sets of vectors, are invariant to space dimension and do not introduce problems with image boundaries. Since these limitations originate from the raster image representation and not from mathematical morphology, the main goal of this work was to overcome these limitations. Operations on images, or especially objects (3–D space), are computationally intensive and require enough memory space for storage and manipulation. For that reason special care should be taken that new implementation does not essentially decrease speed of computation and/or increase memory requirements. Since one of the benefits of the proposed implementation is the invariance to space dimension, the term element is used instead of the foreground pixel of binary images or foreground voxel of binary solids. The element vector components are element coordinates in n-dimensional space. In order to overcome the above two limitations elements are represented explicitly, i.e. the element vector components are stored. This is opposed to the raster representation where element vector components are determined implicitly by element position in the array and only the element value is stored. To avoid the significant increase of the memory requirements of element vector representation a compression of the set of elements is used. Elements are represented by $((a_1, \ldots, a_n), elcount)$ where $(a_1, \ldots, a_n)$ denotes element coordinates, and elcount is the element run length. The element run length is used for the run-length compression of the set.

The algorithm for merging of two element run-lengths is as follows:

Let $A \subseteq Z^n$ be the set of elements, where Z is the set of integers.

Let $((a_1, \ldots, a_n), elc_a)$ represent $(a_1, \ldots, a_n)$, $(a_1, \ldots, a_n+1), \ldots, (a_1, \ldots, a_n+elc_a-1) \in A$. Let $((b_1, \ldots, b_n), elc_b)$ represents $(b_1, \ldots, b_n)$, $(b_1, \ldots, b_n+1), \ldots, (b_1, \ldots, b_n+elc_b-1) \in A$.

```
**if** $(a_i = b_i, i < n)$
    **if** $(a_n <= b_n$ and $a_n + elc_a >= b_n)$
        elements are merged end represented by $((a_1, \ldots, a_n), \mathbf{max}(elc_a, b_n + elc_b - a_n))$
    **if** $(b_n <= a_n$ and $b_n + elc_b >= a_n)$
        elements are merged end represented by $((b1, \ldots, b_n), \mathbf{max}(a_n + elc_a - b_n, elc_b))$
```

An example of a set compression is given in Table. 1.

The set is a collection of elements. In order to achieve simple and fast element merging the set is sorted. It is very important that the usage of the compressed and sorted set, which is needed to decrease memory requirements, does not decrease a speed of computation. On the contrary, compressed element run lengths act as subsets, so that set operations are performed on the subsets, instead on the single elements, resulting in more efficient computation.

3. Object-Oriented Implementation of Morphological Operations

We developed an object-oriented implementation for computation of binary morphological operations in GNU C++. Since object-oriented paradigm [1] is used for implementation, behavior of morphological sets is implemented as a class **MorphSet**. The class architecture of the proposed implementation is given in Fig. 1, where rectangles denote classes, rectangles with rounded corners template classes, double arrows denote inherit relationship, pointing from subclass to superclass, and single arrows denote client-supplier relationship, pointing from client to supplier class.

Class **MorphElement** defines behavior of elements. Storage attributes, defining a state of objects, of the **MorphElement** are:

- *pVec*, pointer to the dynamically allocated space for vector representation, that is inherited from the class **shortVec**;
- *VecDim*, a variable that holds vector dimension, inherited from the class **shortVec**;
- *ElemCount*, a variable used for the compression.

Most important **MorphElement** operations are:

Tab. 1. An example of a set compression for $n = 3$

Set	Compressed Set
(1,5,7), (1,5,8), (1,5,9)	((1,5,7), 3)
(1,5,11), (1,5,12)	((1,5,11), 2)
(2,3,4)	((2,3,4), 1)
(2,5,6), (2,5,7), (2,5,8), (2,5,9), (2,5,10)	((2,5,6), 5)

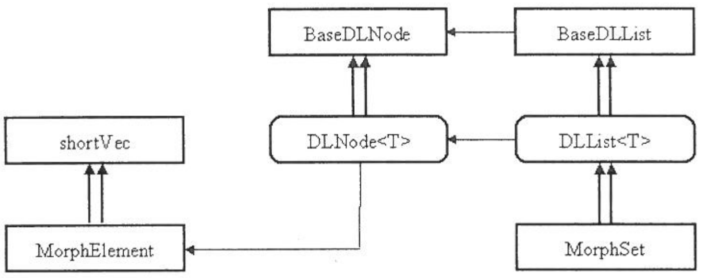


Fig. 1. The class architecture for the computation of the morphological operations

- Relational operators ($==$, $!=$, $<$, $>$, $<=$, $>=$) used for searching for an element position in a sorted set as well as to find out whether the merging condition is met;
- Arithmetic operators ($+$, $-$, $+=$, $-=$) used for the element translation in space;
- Functions *merge* and *intersect*, which perform merging and intersecting of elements, respectively.

Double linked list is used as an underlying storage structure, because it enables easy insertion of an item between two items in the list, which is needed to keep the list sorted. The number of items stored in the list is limited by the memory space only.

Class **MorphSet** defines behavior of the morphological sets. Storage attributes of the **MorphSet** are:

- *h*, pointer to the head of the set, inherited from **baseDLList**;
- *TemPos*, pointer to the temporary position in the set that essentially speeds up searching for the correct position;
- *TransVec*, variable of **shortVec** type that stores components of the translation vector;
- *IsComplement*, variable of **int** type that enables selecting of the correct operation if the operation on complement set is needed.

The most important functions of class **MorphSet** are: *AddElement*, which adds new element to the correct position in the sorted set, and is responsible for compression (by calling *merge* function if merging condition is met) and *RemoveElement*, which removes element from the sorted set.

Since a class in C++ represents a new data type, the following operators are defined for the **MorphSet** class:

- $+$ and $+=$, the set union and unary set union, respectively;
- $*$ and $*=$, the set intersection and unary set intersection, respectively;
- $-$ and $-=$, the set difference and unary set difference, respectively.

The implementations of the above listed set operations are based on *AddElement* and *RemoveElement* functions. In addition to the above operators, the functions *Dilation, Erosion, Opening*, and *Closing* are defined for the class **MorphSet**. Implementations of the morphological operations are based on the above set operations and *Translate* function that is also defined for the **MorphSet**. The pseudo code for dilation operation is shown bellow:

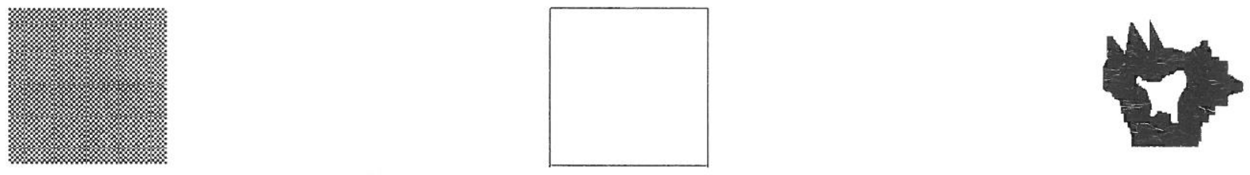
```
for each element of B
    A.Translate(B.Element)
    ResultSet += A
return ResultSet
```

The following example illustrates the ease of use of developed classes for edge detection:

```
MorphSet A, B, EdgeSet
fill A from image file
fill B with ((−1, −1), 3), ((0, −1), 3), ((1, −1), 3)
EdgeSet = A − Erosion(A, B)
```

4. Results and Comparison to Raster Approach

The proposed implementation of morphological operations has two major advantages in comparison to the implementation based on raster representation:



a) the worst case
1. 16,384 byte
2. 163,862 byte

b) the best case
1. 16,384 bytes
2. 22 bytes

c) a general case
1. 16,384 byte
2. 3,182 bytes

Fig. 2. Sample images of size 128 × 128
(1. shows memory requirement for the raster representation, and
2. shows memory requirements for the proposed representation.)

- Generality — The algorithms for the computation of the morphological operations are independent of space dimension;
- The proposed implementation does not involve problems with the image boundaries in the way implementations based on raster representation do. The boundaries are, in the proposed implementation, determined by a range of values for a variable of **short** type ($-32,768$ to $32,767$), and considering expected shape sizes cannot be exceeded.

4.1. Memory Requirements

Each object of **DLLNode<MorphElement>** type requires 20 bytes (for 3–D case 22 bytes), while 22 bytes are needed for the object of **MorphSet** type. In order to avoid increase of memory requirements the set compression is used. A set compression ratio depends on image contents and Fig. 2 illustrates two extreme and one practical case. There are cases where the proposed representation requires more memory space, but for the most practical cases it requires less memory space then raster representation.

4.2. The Speed of Computation

The following features of the proposed representation are very useful for increasing the speed of computation:

- Set compression — A compressed element run lengths act as subsets, so that set operations are performed on subsets, instead on single elements, resulting in more efficient computation;
- Set sorting — Speeds up the searching of the linked list;
- Usage of *TemPos* variable — Significantly speeds up the searching. Set operations and morphological operations take two sets as input, and perform an operation between elements from both sets (for example, union adds each element of one set to the other). Since both sets are sorted, the probability that *TemPos* is pointing "very near" to the correct position in the set is high.

The computation speed for both the raster and the proposed approach depends on image size and image contents and it is impossible to generally state which one is faster. There is a proportional relation between the compression ratio and the speed of the proposed implementation. It is demonstrated in the previous section that the compression ratio is proportional to the size of binary image regions. If compression ratio is less then one (e.g. chess board shape in Fig. 2), the proposed implementation computes few times slower. However, considering that most practical cases are shapes without "too many" holes, the proposed implementation even increases the speed of computation.

5. Discussion and Conclusion

The motivation for this work was a desire to utilize the proposed approach for implementation of the Morphological Signature Transform [5] for 2–D and 3–D shape descriptions. We developed a new implementation for computation of binary morphological operations, that is invariant to space dimension, so that the same code and computation algorithm can be used in arbitrary dimensions. The proposed implementation uses sorted double-linked lists for the representation of morphological sets instead of 2–D arrays for images or 3–D arrays for objects.

Since the number of items that can be stored in the list is limited only by memory space, the proposed implementation does not have problems with the image/object boundaries. In order to avoid the increase of memory requirements the set compression is used. Due to the usage of compressed and sorted linked list for the most practical cases the proposed implementation requires less memory and computes morphological operations as fast as implementations based on raster representation.

The proposed implementation can also be used for gray scale morphology, because 2–D gray scale morphology can be, through the use of umbra concept, represented in terms of 3–D binary morphology [6]. However, the computation speed, in that case, is not as good as if a dedicated code for the computation of gray scale morphology operation was used. Further work will include an optimization of the proposed implementation for the gray scale morphology.

References

[1] L. ATKINSON AND M. ATKINSON, Using C, Prentice Hall, 1990.

[2] R. M. HARALICK AND L. G. SHAPIRO, Computer and Robot Vision, Vol. 1, Addison–Wesley, 1992.

[3] R. M. HARALICK, S. STERNBERG AND X. ZHUANG, "Image Analysis Using Mathematical Morphology", IEEE Transactions on PAMI, Vol. 9, pp. 532–550, 1987.

[4] S. LONČARIĆ, Morphological Signature Transform for Shape Representation and Matching, PhD thesis, University of Cincinnati, 1994.

[5] S. LONČARIĆ AND A. DHAWAN, "A Morphological Signature Transform for Shape Description", Pattern Recognition, No. 26, pp. 1029–1037, 1993.

[6] J. SERRA, "Introduction to Mathematical Morphology", Computer Vision, Graphics, and Image Processing, No. 35, pp. 283–305, 1986.

[7] J. SERRA, Image Analysis and Mathematical Morphology, Academic Press, 1982.

[8] S. STERNBERG, "Grayscale Morphology", Computer Vision, Graphics, and Image Processing, No. 35, pp. 333–355, 1986.

[9] The Khoros Group, Khoros Image Processing Package, University of New Mexico, 1992.

Received: November, 1996
Accepted: March, 1997

Contact address:
Zoran Đukić
Institute "Ruđer Bošković"
Bijenička 54, 10 000 Zagreb
Croatia
e-mail: dukic@olimp.irb.hr

Sven Lončarić
Department of Electronic Systems
and Information Processing
Faculty of Electrical Engineering and Computing
University of Zagreb
Unska 3, 10 000 Zagreb
Croatia
e-mail: sven@zems.fer.hr

ZORAN ĐUKIĆ (1969) received the B.Sc. degree in Electrical Engineering and the M.Sc. degree in computer science from the Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia, in 1993 and 1996, respectively. He is a research engineer at the Institute Ruđer Boškovic in Zagreb. His research interests include object-oriented programming and image processing.

SVEN LONČARIĆ (1961) received the B.Sc. and M.Sc. degrees in Electrical Engineering from the Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia, in 1985 and 1989, respectively. He received the Ph.D. degree in Electrical Engineering from the University of Cincinnati, USA, in 1994, as Fulbright Fellow. Dr. Lončarić is currently Assistant Professor at the Department of Electronic Systems and Information Processing, Faculty of Electrical Engineering and Computing, University of Zagreb. His research interests include image processing and analysis, pattern recognition and volume visualization.