## Efficient Implementation of Morphological and Local Neighbourhood Operations

Attila Tanács and Kálmán Palágyi

Mathematical morphology offers a powerful approach to numerous image processing problems. It is useful in the representation and description of region shape (such as boundaries, skeletons, and the convex hull) and in pre- or post-processing (such as filtering, thinning and pruning).

Morphological operations can be extended to gray–scale images, but our attention is focussed on binary images whose components are elements of $\mathbb{Z}^2$ or $\mathbb{Z}^3$. A digital binary image containing finitely many object points can be stored in a binary array in which 1's represent object points and 0's correspond to the background and the holes of the image.

A complex image analysis problem often involves the concatenation of several low–level morphological operations including dilation, erosion, hit–or–miss transformation, and point–wise logical operations. It is important to implement such low–level operations efficiently and to solve their concatenation as well.

There is a more general class of image operations called *local neighbourhood binary operations*. An image operation belongs to this class if and only if the value of every point in the destination image is a function of a small (say $3 \times 3$ in 2D and $3 \times 3 \times 3$ in 3D) neighbourhood of that point in the source image.

Local neighbourhood operations can be described by Boole–functions: a point is to be inverted if and only if the corresponding Boole–function is true for its neighbourhood. Any Boole–function can be replaced by a set of single-template predicates. A template is a small binary array (corresponding to the investigated neighbourhood) for defining a predicate: where a point satisfies the predicate if and only if the template matches its neighbourhood, where each 1 template element matches 1 point and each 0 template element matches 0 point. Note that no reflection or rotation is allowed in matching the template to the neighbourhood of the given point. In order to reduce the number of templates, templates can contain "don't care" elements, too. They match either 0's or 1's. It is easy to see that a template containing $k$ "don't care" elements can be replaced by $2^k$ binary templates.

We introduce a new type of description that is called *generalized template matching*. It is a combination of Boole–formulas and templates. We have shown that any local neighbourhood operation can be described by this method.

The local dependence must be evaluated for every points of the image which is rather slow for large images. A general way for speeding up that kind of processes is using a lookup–tree: for all the possible neighbourhood configurations is evaluated only once and the the given values are stored in a binary tree. For $n$ variables the tree has $2^n$ leaves. A common 3D operation can require more than 20 variables and in this case it is not efficient to store the whole tree thus tree–compression techniques are necessary. In order to reduce the size of trees, two methods are applied: *identical–subtree–compression* and *identical–leaves–compression*.

A special *programming language interpreter* is also proposed and developed for defining complex local neighbourhood operations. The graphical user interface provides easy and uniform way for giving and executing these operations.

There are some systems (e.g., KHOROS, AVS) that allow users to dynamically connect low–level operations or software modules to create data flow graphs for scientific computation. There are, however, some deficiencies of those systems which limit their usefulness (data flow graphs must be special, e.g. circle–free). Our programming language interpreter differs from the above systems in two regards: creating low–level operations does not need any programming; the structure of the complex operation is not limited (e.g., loops can be applied).

An example is to illustrate the simplicity and the efficiency of our approach. A 6–subiteration 3D thinning algorithm is given by the proposed language. The result of this morphological operation can be seen in Fig. 4.

```
rem 6-phase 3D thinning

picture A, B;
operation t1, t2, t3, t4, t5, t6;
int i;

assign(t1, "6thu.for" );
assign(t2, "6thd.for" );
assign(t3, "6thn.for" );
assign(t4, "6ths.for" );
assign(t5, "6the.for" );
assign(t6, "6thw.for" );

A=GetActivePicture();
B=CreateCompatiblePicture(A);
CopyPicture(B,A);
DisplayPicture(B);

InfobarText("6-subiteration 3D Thinning") ;
i=0;
while ( exec(B,B,t1 ) or exec(B,B,t2 ) or exec(B,B,t3 ) or
        exec(B,B,t4 ) or exec(B,B,t5 ) or exec(B,B,t6 )    )
{
   i=i+1;
   InfobarText("Number of itarations: %d", i);
}

DisplayPicture(B);
end;
```

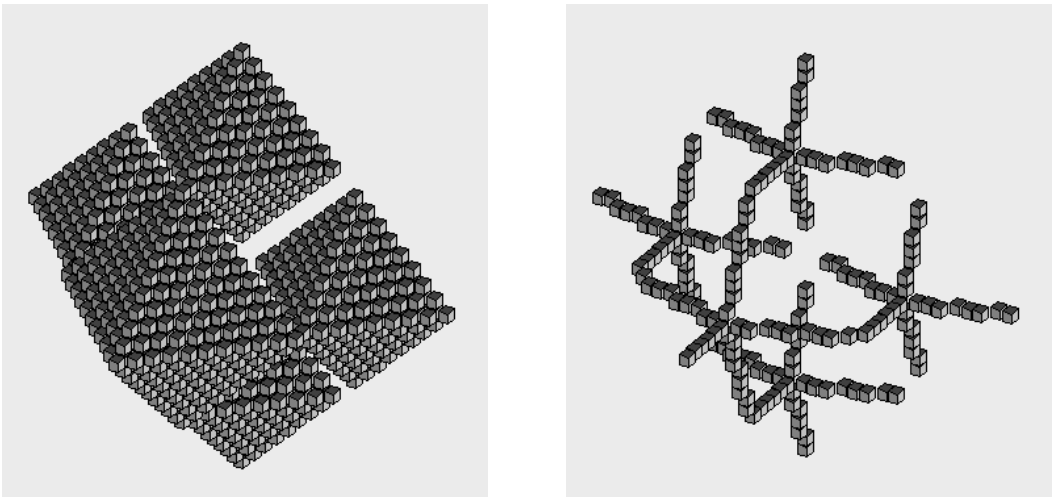

Figure 4: Example of 3D thinning. The original synthetic object (left); its medial lines (right). (Small cubes represent object points.)