Algorithmic description of erosions and dilations in Mamba

Serge BEUCHER
CMM/ARMINES/MinesParisTech
http://www.mamba-image.org

This document is part of the Mamba algorithmic documentation

This note describes the implementation of the basic structuring elements in the Mamba-Image library [1]. These structuring elements are composed of those which are defined on the elementary neighborhood of a point (hexagon on an hexagonal grid, square on a square one) but also of the octogonal (square grid) and dodecagonal (hexagonal grid) ones.

1. Basic structuring elements on the elementary neighborhood

A class named *structuringElement* allows to define structuring elements on the elementary neighborhood of a point. A structuring element is simply defined by a list of points of the neighborhood and by the grid which is used. The points are coded according to the following scheme (Figure 1):

Figure 1*: Directions coding for the hexagonal and square grids.*

For instance, an hexagonal structuring element is defined by:

HEXAGON = structuringElement([0, 1, 2, 3, 4, 5, 6], mamba.HEXAGONAL)

A square one by:

SQUARE3x3 = structuringElement([0, 1, 2, 3, 4, 5, 6, 7, 8], mamba.SQUARE)

The following definition corresponds to a centered segment of size 2 in the 60° direction on hexagonal grid:

SEGMENT60 = structuringElement([0, 1, 4], mamba.HEXAGONAL)

In any case, the origin of the structuring element is always the center point although it is not compulsory that this center point belongs to the structuring element.

An erosion or a dilation by such a structuring element is obtained by performing the infimum (for the erosion) or the supremum (for the dilation) of the different shifts of the initial image which are imposed by (derived by the position of) the points belonging to the structuring element. These successive operations can be obtained with the *infNeighbor* or *supNeighbor* transformations available in the Mamba library.
Note that, with some structuring elements, it would be possible to achieve erosions and dilations in a faster way by using successive operations with segments. A dilation by a 3x3 square for instance can be obtained with four successive dilations by size 1 segments in the directions 1, 3, 5 and 7 of the square grid which is twice as fast as the implementation based on suprema of shiftings. The same approach could be used with the elementary hexagon (that is, performing three successive dilations in directions 1, 3 and 5 instead of six shifts). However, in this case, it is not a good idea, as it produces errors on the edge of the image. These edge effects are illustrated in Figure 2.

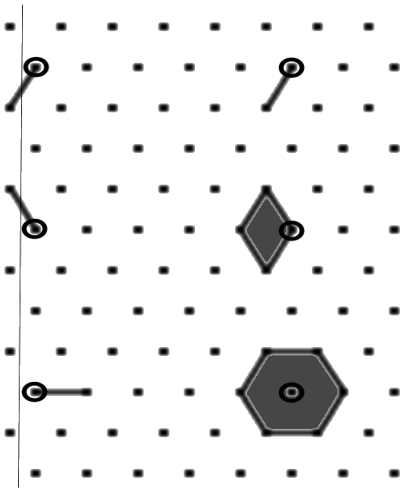

Figure 2*: Edge effects when computing an hexagonal dilation with three successive dilations by segments: on the left, points generated by dilations in directions 1 and 3 fall outside the image window and do not contribute to the next dilations.*

They come from the fact that a point generated by a dilation by a segment may fall outside of the image window which prevents it to be used at the next step to generate another point inside the window. The same phenomenon may occur also with erosions. As this bias does not occur when using suprema or infima of shiftings, this implementation is therefore used systematically with operators (erosion, dilation) using the *structuringElement* class (even on a square grid, for the sake of simplicity, although this bias does not appear).
The structuring elements already defined in the Mamba library are given in Figure 3. Note that these structuring elements can be easily rotated, transposed through specific methods of the *structuringElement* class.
Note also that a specific implementation of large structuring elements exists in the library (module *erodilLarge.py*). This implementation is described in [2]. As this implementation actually uses successive operations with segments (with some computation tricks to cope with edge effects), it is likely that these operators be faster on the square grid even for small size structuring elements.

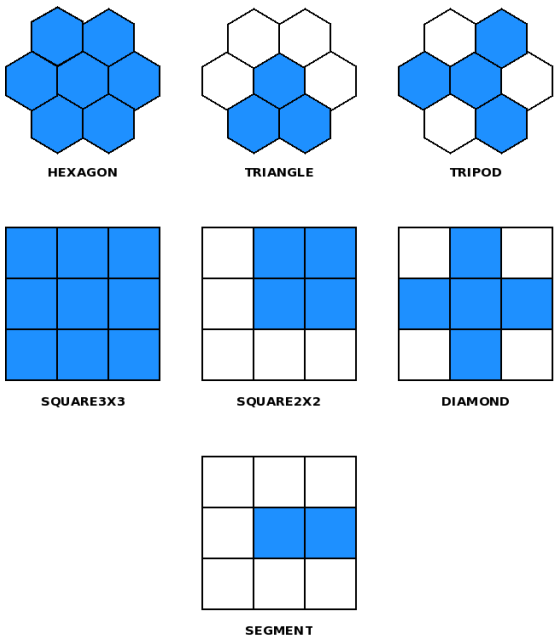

***Figure 3**: Some structuring elements present in the Mamba library.*

2. Octogonal structuring elements

2.1. Octogons design

Octogons can be obtained by concatenating dilations with squares and with conjugate squares (squares turned by $\pi/4$), also called diamonds.

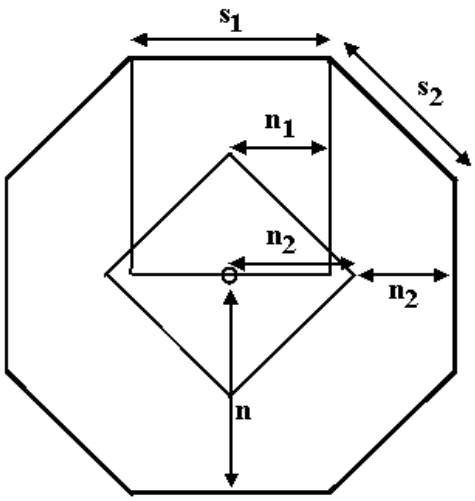

***Figure 4**: size of the octogon compared to the respective sizes of the square and of the diamond used to get it.*

The size n of the octogon is equal to:

$$n = n_1 + n_2$$

where n_1 is the size of the square and n_2 the size of the diamond. These sizes determine the size of

the sides of the octogon, n_1 determining the size s_1 of the horizontal and vertical sides and n_2 the size s_2 of the oblique sides (Figure 4). We have:

$$s_1 = 2n_1$$

$$s_2 = \sqrt{2}\, n_2$$

In order to obtain isotropic octogons, we put:

$$s_1 = s_2$$

which leads to:

$$n_1 = (\sqrt{2} - 1)n = 0.41421\, n$$

$$n_2 = n - n_1 = (2 - \sqrt{2})\, n$$

Finally, as n_1 and n_2 must be integer numbers, we write:

$$n_1 = E(0.41421\, n + 0.5)$$

where E(x) stands for the integer part of x.

$$n_2 = n - n_1$$

The precision of the formula is sufficient for sizes of operations up to 100,000.
The following figure (Figure 5) shows the increasing sequence of octogons generated by these formulas.

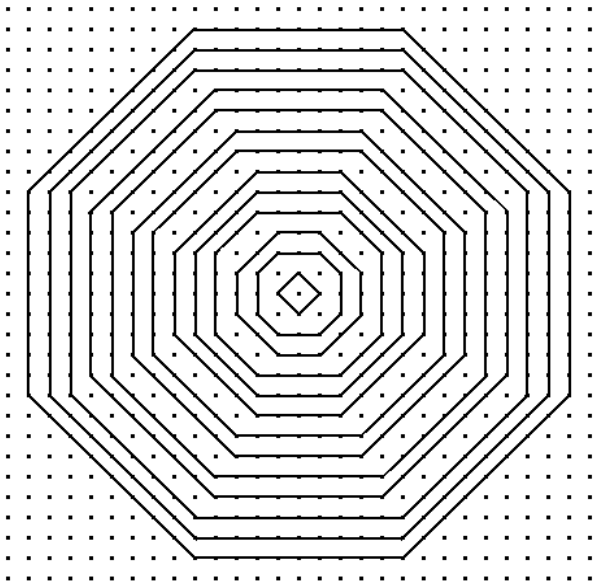

Figure 5*: Successive octogons generated by the above algorithm.*

2.2. Generating a sequence of octogons of increasing sizes

The following table (Table 1) contains the respective sizes of the squares and diamonds for the octogons of sizes 1 to 10. Note that these respective sizes do not increase according to a periodic scheme. We show that, to obtain an octogon of size i+1 from an octogon of size i, either an operation with a square or with a conjugate square is performed.
Many morphological transformations (distance functions, residual transforms) need the elaboration of sequences of basic operations (erosions or dilations). So, it is necessary to exhibit the rule allowing to obtain a size i+1 transformation from the previous size i result. In the case of octogons, this rule is simple:

- calculate $n = E(0.41421\, i + 0.5)$
- calculate $n' = E(0.41421\, (i+1) + 0.5)$
- if n = n', the size i + 1 octogon is obtained by an operation with a diamond. Otherwise, it is

obtained by an operation with a square.

As an example, let us determine how to produce a dilation by an octogon of size 73 of a set X from the dilation of size 72. Let us calculate n and n' (see below). We see that n = n'. Therefore the size 73 octogonal dilation will be obtained by applying a dilation by a diamond to the previous dilated set.

$$n = E(0.41421 \times 72 + 0.5) = 30$$
$$n' = E(0.41421 \times 73 + 0.5) = 30$$

n	n_1	n_2
1	0	1
2	1	1
3	1	2
4	2	2
5	2	3
6	2	4
7	3	4
8	3	5
9	4	5
10	4	5

***Table 1**: Squares and diamonds sizes for the generation of size n octogons.*

3. Dodecagonal structuring elements

3.1. Generating dodecagons

Dodecagons can be obtained on the hexagonal grid by concatenating dilations with hexagons and with conjugate hexagons. These latter ones are produced by two transposed tripod structuring elements (Figure 6).

Note that the size of the conjugate hexagon which corresponds to the number of iterations of both tripod transformations is half the size of the circumscribed hexagon (Figure 6a). As for the octogons, the respective sizes n_1 of the hexagon and n_2 of the conjugate hexagon control the sizes s_1 and s_2 of the 60° and 90° sides of the dodecagon. In order to generate isotropic dodecagons, we must have $s_1 = s_2$. This condition leads to the following values for the size n_1 of the hexagon and n_2 of the conjugate hexagon (Figure 6b) needed to generate a size n dodecagon.

$$s_1 = n_1$$
$$s_2 = \sqrt{3}\, n_2$$
$$s_1 = s_2 \text{ leads to } n_1 = \sqrt{3}\, n_2$$

The size n of the dodecagon (which is also the size of the embedding hexagon) is given by:

$$n = n_1 + 2n_2$$

Then:

$$n_1 = \sqrt{3}(2 - \sqrt{3})\, n \approx 0.46410\, n$$
$$n_2 = (2 - \sqrt{3})\, n \approx 0.26795\, n$$

As for octogons, the values n_1 and n_2 must be integer values. However, contrary to the octogonal case, it is not sufficient to take the closest integer to obtain an appropriate value. Indeed, if n is, for instance, equal to 5, n_1 and n_2 are respectively equal to 2.32 and 1.34 before truncation, which leads to $n_1 = 2$ and $n_2 = 1$ if the nearest integer rule was enforced. But n would be equal to 4 and not to 5...

In order to cope with this problem, a more sophisticated rule must be applied.

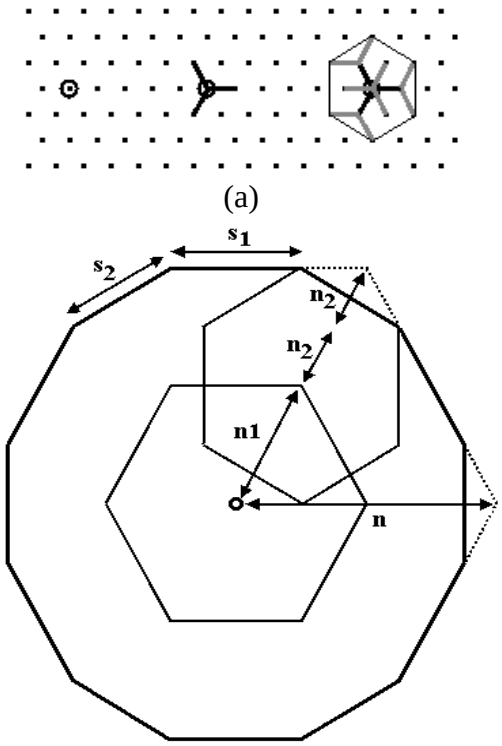

Figure 6*: (a) generation of a conjugate hexagon of size 1 by two successive tripods. Note the size of the conjugate hexagon compared to its embedding hexagon. (b) Relations between the size of the dodecagon and the sizes of the hexagon and conjugate hexagon.*

Firstly, let us calculate n_1. Its real value is bounded by two successive integer values, p and p+1. Now, if the size n of the dodecagon is an even number, as $n = n_1 + 2n_2$, n_1 must be even. Therefore, the approximation of n_1 must be chosen among p and p+1 depending on the parity of these two numbers.
Figure 7 shows successive dodecagons produced by this algorithm and Table 2 gives the corresponding hexagon and conjugate hexagon sizes for the first ten values.

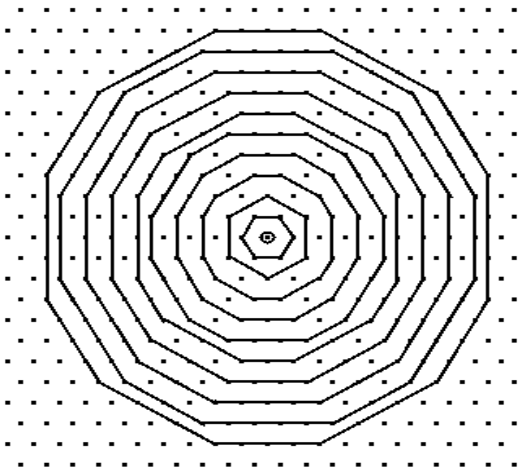

Figure 7*: Successive dodecagons generated by the above procedure.*

n	n_1	n_2
1	1	0
2	0	1
3	1	1
4	2	1
5	3	1
6	2	2
7	3	2
8	4	2
9	5	2
10	4	3

Table 2*: Hexagons and conjugate hexagons sizes for the generation of size n dodecagons.*

.

3.2. Iterating dodecagons

As for octogons, it is interesting to provide a rule for producing dodecagonal operations of size n + 1 from the result of the operation at size n. However, in the dodecagonal case, the rule is not as simple as it was in the octogonal case because, as it can be shown in the previous table, to obtain the result of a dodecagonal operation of size n + 1, the results of size n but also of size n - 1 operations are needed. For instance, if we consider the size 5 dodecagonal operation, we see that it can be performed by concatenating a size 3 hexagonal operation and a size 1 operation with a conjugate hexagon. But the result of size 6 can only be produced by performing an operation with an elementary hexagon on the size 4 result. On the contrary, to get an operation of size 7, the previous result of size 6 is sufficient: we just have to perform an elementary hexagonal dilation to get it.

So, to perform successive dodecagonal operations of increasing sizes, we must preserve the two previous results. Then, from the results of the size i and size i - 1 operations, getting the result of the size i + 1 operation is achieved by the following procedure:

- calculate n_1 for the size i operation, with the parity correction described above (in an iterative procedure, this value is supposed to have been calculated at the previous step).
- calculate n'_1 for the size i + 1 operation.
- if $n'_1 = n_1 + 1$, perform an elementary hexagonal operation on the size i result to obtain the size i + 1 dodecagonal operation.
- else, perform an operation by an elementary conjugate hexagon on the size i - 1 result to obtain the size i + 1 dodecagonal operation.

This implementation has been used to realise the dodecagonal distance function of a set. It can be found in the *mambaComposed.miscellaneous.py* module.

4. Conclusion

Erosions and dilations with basic structuring elements are realised in Mamba in such a way that no edge effects are likely to occur, which would be unacceptable for these elementary transformations. This exactness is obtained after a small reduction of the performances.

Regarding octogonal and dodecagonal operators, their design has been made in order to favour their simplicity. However, it is of the outmost importance to keep in mind some of their characteristics

and, in particular, the fact that the operations are not associative. It is generally not true that $\delta_{m+n} = \delta_m(\delta_n)$ if δ_n is a dodecagonal or an octogonal dilation (the same problem holds for erosions). Therefore, it is wise to handle these transformations cautiously to avoid erroneous results.

5. References

[1] Beucher, Nicolas (2010): Mamba Image Library Python Reference - http://mamba-image.org.
[2] Beucher Serge (2010): Fast implementation of large erosions and dilations in Mamba - http://cmm.ensmp.fr/~beucher/publi.html

(Publication date: February 22, 2011)

*This document is copyrighted under the **Creative Commons "Attribution Non-Commercial No Derivatives"** license. For terms of use, see http://creativecommons.org/about/licenses/.*