Erosion, dilation and related operators

Mariusz Jankowski

Department of Electrical Engineering
University of Southern Maine
Portland, Maine, USA

mjankowski@usm.maine.edu

This paper will present implementation details of an important set of numerical operators in the *Digital Image Processing* application package. These include the erode and dilate operators and two related recursive operators that are central to morphological reconstruction algorithms which are currently being developed for a future release candidate.

Erode and dilate are the fundamental operators of mathematical morphology, a theory for the analysis of spatial structure. The methods of mathematical morphology make possible a large number of very powerful image analysis techniques and therefore these operators and their implementations are of great theoretical and practical interest to many involved in image processing and analysis.

■ Introduction

Mathematical morphology is a powerful tool for geometrical shape analysis and description. It was originaly developed by G. Matheron and J. Serra for the purpose of analysing binary images, thus a set formalism was used with simple operations such as intersection, union, or translation. It was subsequently extended to integer and real−valued signals and images and has evolved into a powerful image analysis tool.

This paper presents in some detail two fundamental morphological operators: dilation and erosion. Most morphological algorithms are based on these two primitive operators, in particular all the additional operators discussed in this paper, including open, close, top−hat, and reconstruction by dilation. It is of considerable interest therefore to explore effective implementations of these operators.

The paper is organized as follows. First, the basic definitions of binary erosion and dilation are given. These are based on the concepts of sets and their translations. The *Mathematica* implementations follow immediately, but due to their computational inefficiency other algorithms will be developed and presented. The section concludes with a comparison of timing data for several different versions of the operators. In the next section, grayscale erosion and dilation operators are presented and several realizations are compared. Finally, in the last section a brief introduction to morphological reconstruction is given with details of reconstruction by geodesic dilation.

■ Binary erosion and dilation

□ Set−theoretic formulation

The binary operators of erosion and dilation are typically defined using the concept of a set description of a binary image. A set description of a binary image is simply a list of integer pairs representing the positions of the 1−valued samples. Thus we have the following correspondence between a binary image (in matrix format) and a set where a coordinate system centered on the upper−left sample was used.

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \rightarrow \{\{1, 0\}, \{1, 1\}, \{1, 2\}, \{2, 2\}\} \quad (1)$$

When displayed graphically the origin shifts to the lower−left position as shown in Figure 1.

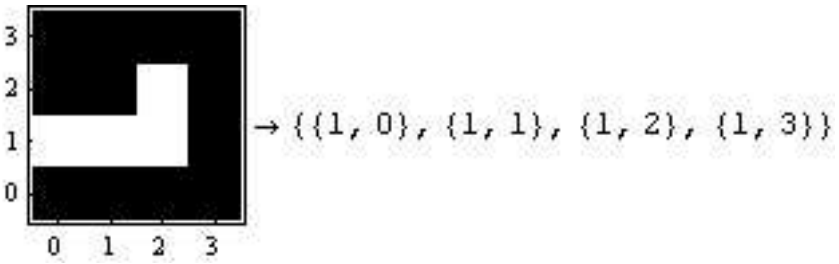


Figure 1. Binary image and its set description.

The definitions of dilation and erosion are typically formulated using the concept of a set translation and a set reflection. The translation of a set α by a point (or vector) x, denoted α_x, is defined by

$$\alpha_x = \{a + x : a \in \alpha\} \quad (2)$$

The reflection of a set α, denoted $\overline{\alpha}$, is defined

$$\check{\alpha} = \{-a : a \in \alpha\} \quad (3)$$

Binary dilation of set α by set β, denoted here $\alpha \oplus \beta$, is the set union of all translations of set α by elements of set β or equivalently the set of all positions of the reflected set $\check{\beta}$ for which it intersects with, or "hits", set α. The set β is commonly called the structuring element and plays a similar role to a finite impulse response filter in linear signals and systems theory.

$$\alpha \oplus \beta = \bigcup_{b \in \beta} \alpha_b = \left\{x : \check{\beta}_x \cap \alpha \neq 0\right\} \quad (4)$$

Here are two example sets.

```
(α = {{1, 0}, {1, 1}, {1, 2}, {2, 2}};
 β = {{0, 0}, {0, 1}, {1, 1}};)
```

This gives the dilation of set α by the structuring element β.

```
Union @@ Table[α[[j]] + β[[i]], {i, Length[β]}, {j, Length[α]}]
```

```
{{1, 0}, {1, 1}, {1, 2}, {1, 3}, {2, 1}, {2, 2}, {2, 3}, {3, 3}}
```

The erosion of set α by set β, denoted here $\alpha \ominus \beta$, is the set intersection of all negative translations of set α by elements of set β or equivalently the set of all positions for which set β is a subset of α (i.e., it "fits" inside α).

$$\alpha \ominus \beta = \bigcap_{b \in \beta} \alpha_{-b} = \{x : \beta_x \subset \alpha\} \tag{5}$$

```
Intersection @@
 Table[α[[j]] - β[[i]], {i, Length[β]}, {j, Length[α]}]
```

```
{{1, 1}}
```

The set descriptions of binary erosion and dilation have straightforward implementations in *Mathematica* but unfortunately they are terribly inefficient.

□ Other implementations

For purposes of signal and image processing it is preferable to consider formulations of erosion and dilation in terms of arrays instead of sets. Interestingly this leads to realizations of the operators that are strikingly similar to convolution and correlation. Recall that linear convolution of a one–dimensional (1D) signal f with filter h is defined.

$$(f * h)(x) = \sum_{y \in \mathcal{D}_h} f(x - y)\, h(y) \tag{6}$$

where $\mathcal{D}_h$ denotes the region of support of filter h. The dilation of a signal f with structuring element s may likewise be written in a convolution–like form.

$$(f \oplus s)(x) = \bigcup_{y \in \mathcal{D}_s} f(x - y) \bigcap s(y) \tag{7}$$

where symbols $\bigcup$ and $\bigcap$ are the logical OR (union) and logical AND (intersection) operators, respectively. This process of scanning a small kernel over a large data array and computing a sort of inner product of the corresponding samples suggests an implementation using `PartitionMap`. Here matrices `A` and `B` represent the sets α and β, respectively and `rB` is the reflection of `B`.

```
A = {{0, 0, 0, 0}, {1, 1, 1, 0}, {0, 0, 1, 0}, {0, 0, 0, 0}};
B = {{0, 0, 0}, {0, 1, 1}, {0, 0, 1}};
rB = {{1, 0, 0}, {1, 1, 0}, {0, 0, 0}};

PartitionMap[BitOr @@ BitAnd[Flatten[#], Flatten[rB]] &,
  A, {3, 3}, {1, 1}, 2, 0] // MatrixForm
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Alternatively, this convolution–like computation may be implemented with function `ListConvolve` by replacing the default operators `Times` and `Plus` by `BitAnd` and `BitOr`. This leads immediately to the following realization of binary dilation.

```
ListConvolve[B, A, 2, 0, BitAnd, BitOr] // MatrixForm
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Unfortunately, neither one of these two implementations is sufficiently fast for typical applications. There are two more possible solutions. The first is based on the equivalence of logical “and” (`BitAnd`) and binary multiplication (`Times`). It is easy to see that for

binary−valued signal f and structuring element s the following relation between linear convolution, denoted by $*$ and binary dilation holds.

$$(f \oplus s)(x) = T(\,(f * s)(x)\,), \text{ where } T(x) = \begin{cases} 0, & 0 \le x < 1 \\ 1, & \text{else} \end{cases} \tag{8}$$

Thus we get

```
Threshold[ListConvolve[B, A, 2, 0], 1] // MatrixForm
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

This indeed gives the best possible realization with *Mathematica*'s built−in functions. The last remaining alternative is to code the dilation and erosion algorithms in a compiled language such as Java or C++ and use *Mathematica*'s linking technologies to execute such code. This has been demonstrated to provide clear benefits under certain conditions [1]. Here is a somewhat simplified code fragment of a Java versions of the dilation algorithm.

```
    // for all x, y in img
                dst[x][y] = img[x][y] ;
                for (int i = 0; i < kerX; i++)
                    for (int j = 0; j < kerY; j++) {
                        if (img[x − i][y − j]==1 && ker[i][j]==1)
{
                            dst[x][y] = 1;
                            break ;
                        }
                }
```

This executes the Java code via *JLink* and returns the desired result.

```
obj = JavaNew["Morphology"];
Take[obj@dilate[ZeroPad[A, {1, 1}, {1, 1}], B],
  {2, −2}, {2, −2}] // MatrixForm
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Here is an example of the effect of erosion and dilation on a binary image. Note how dilation increases and erosion decreases the number of white pixels in the image. The difference between the two returns the border of the white objects.

```
bin = Threshold[books, 140];
se = Table[1, {3}, {3}];
DisplayTogetherArray[{Graphics[BinaryDilate[bin, se]],
   Graphics[BinaryErode[bin, se]],
   Graphics[BinaryDilate[bin, se] - BinaryErode[bin, se]]},
  ImageSize → {420, 100}];
```

□ Timing

In this section I measure and compare timing results for several realizations of binary dilation and three different image dimensions. All results can be compared to the speed of linear convolution which sets the processing standard for linear filtering of signals and images. Here I define all the realizations.

```
(method1 :=
  PartitionMap[BitOr @@ BitAnd[Flatten[#], Flatten[B]] &,
   A, {3, 3}, {1, 1}, 2, 0] ;
method2 := ListConvolve[B, A, 2, 0, BitAnd, BitOr] ;
method3 := UnitStep[ListConvolve[B, A, 2, 0] - 1] ;
method4 := obj@dilate[A, B] ;
 convolve := ListConvolve[B, A, 2, 0] ;)
```

This executes the code, collects the timing data and present the results in a table.

```
Block[{A, B}, (B = Table[1, {3}, {3}];
  A = Table[Random[Integer], {256}, {256}];
  t256 = {time[method1], time[method2],
    time[method3], time[method4], time[convolve]};
  A = Table[Random[Integer], {512}, {512}];
  t512 = {time[method1], time[method2],
    time[method3], time[method4], time[convolve]};
  A = Table[Random[Integer], {1024}, {1024}];
  t1K = {"n/a", time[method2],
    time[method3], time[method4], time[convolve]};
  TableForm[Join[{{" ", "256×256", "512×512", "1024×1024"}},
    Transpose[{{"method 1", "method 2", "method 3",
       "method 4", "convolve"}, t256, t512, t1K}]])]
```

	256×256	512×512	1024×1024
method 1	1.0156250	4.0781250	n/a
method 2	0.7031250	2.8125000	11.1250000
method 3	0.017089844	0.07714844	0.30468750
method 4	0.019042969	0.08886719	0.20898438
convolve	0.016357422	0.07031250	0.27734375

Note that the timing results for method 4 include the time to transfer the data over the Java link, which at an average rate of approximately 10 MB/sec requires about 0.1 sec for a 1024^2 image. These results imply that while method 3 is a suitable algorithm for evaluating binary erosion and dilation, a kernel version of method 4 (in C/C++) has the potential of increasing speed by roughly a factor of 2 by eliminating the data transfer time.

■ Grayscale erosion and dilation

Grayscale morphology extends the morphological operators to the domain of integer or real−valued signals defined on a Cartesian grid. The definitions of binary morphology extend naturally to the domain of digital grayscale signals with translation, reflection, and inversion defined as in linear processing while intersection and union become point−wise minimum and maximum operators, respectively. Therefore we have the following definition of grayscale dilation.

$$f \oplus s = \bigvee_{x \in \mathcal{D}_s} f_x \tag{9}$$

where the symbol $\vee$ denotes a point−wise maximum. Thus the dilation of a grayscale signal or image is a point−wise maximum of a series of translations defined by the shape of the structuring element. This defintion implies a flat structuring element. In the more general case of a nonflat structuring element and again, for sake of simplicity assuming 1D signals, we have

$$(f \oplus s)(x) = \bigvee_{y \in \mathcal{D}_s} (f(x-y) + s(y)) \tag{10}$$

This yields the following *Mathematica* realizations.

```
ListConvolve[B, A, 2, Min[A], Plus, Max] // MatrixForm
```

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 1 & 2 & 2 & 2 \\ 1 & 1 & 1 & 2 \end{pmatrix}$$

Here is the equivalent `PartitionMap` implemenentation.

```
PartitionMap[Max[rB + #] &, A, {3, 3}, {1, 1}, 2, 0] // MatrixForm
```

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 1 & 2 & 2 & 2 \\ 1 & 1 & 1 & 2 \end{pmatrix}$$

For a flat structuring element the dilation operator may be simplified as follows. This scans the image with a rectangular window and returns the maximum of the samples within the window. Note the difference in the result due to a change to the structuring element.

```
PartitionMap[Max, A, {3, 3}, {1, 1}, 2, 0] // MatrixForm
```

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

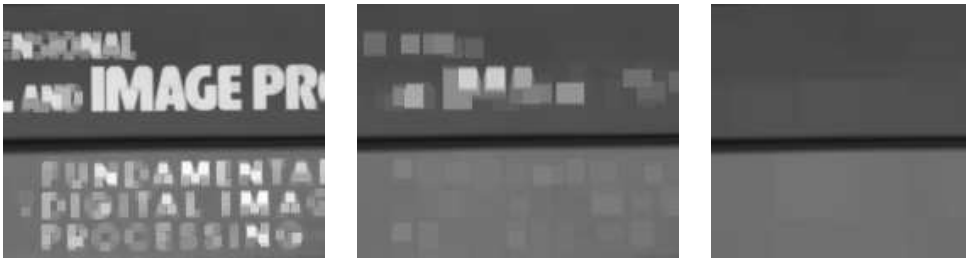
Erosion and dilation operators are seldom used by themselves. Two well−known combinations of the operators result in the so−called open ($\circ$) and close ($\bullet$) operators.

$$f \circ s = ((f \ominus s) \oplus s) \tag{11}$$

$$f \bullet s = ((f \oplus s) \ominus s) \tag{12}$$

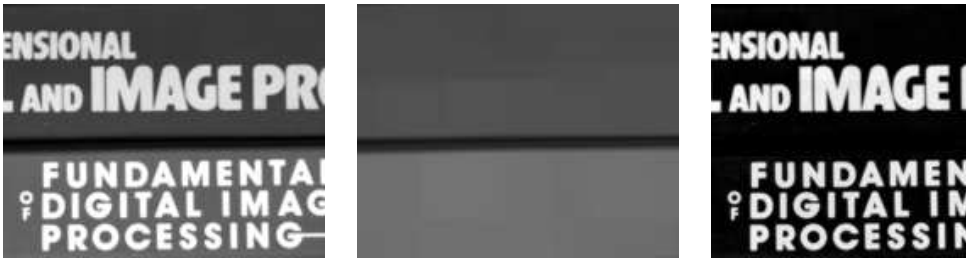
Openings and closings are typically used to suppress structures that cannot contain the structuring element, peaks in the case of openings and valleys in the case of closings. Here is an example of opening the books image with flat structuring elements of increasing dimensions.

```
DisplayTogetherArray[
  Graphics[GrayscaleOpen[books, #], PlotRange → {0, 255}] & /@
   {Table[1, {5}, {5}], Table[1, {10}, {10}],
    Table[1, {25}, {25}]}, ImageSize → {420, 100}];
```



This useful property quickly leads to the morphological top–hat transformation which is an arithmetic difference between an image and it's closing or opening. The top–hat by opening operation, defined $f - f \circ g$, is a very useful background normalization operator. Here is an example, which shows the original image, the result of opening with a large structuring element, and the tophat transformation.

```
se = Table[0, {25}, {25}];
DisplayTogetherArray[{Graphics[books],
   Graphics[GrayscaleOpen[books, se], PlotRange → {0, 255}],
   Graphics[books - GrayscaleOpen[books, se]]},
  ImageSize → {420, 100}];
```



□ Timing

In this section I measure the timing data for four different evaluations of grayscale dilation and several different image dimensions.

```
(Clear[method1, method2, method3, method4];
method1 := ListConvolve[B, A, 2, Min[A], Plus, Max] ;
method2 := PartitionMap[Max[rB + #] &, A, {3, 3}, {1, 1}, 2, 0] ;
method3 := PartitionMap[Max, A, {3, 3}, {1, 1}, 2, 0] ;
method4 := GrayscaleDilate[A, B] ;)
```

This evaluates the code and presents the results in a table. For comparison, the timing of linear convolution is also shown.

```
Block[{A, B}, (B = Table[1, {3}, {3}];
  A = Table[Random[Integer], {256}, {256}];
  t256 = {time[method1], time[method2],
    time[method3], time[method4], time[convolve]};
  A = Table[Random[Integer], {512}, {512}];
  t512 = {time[method1], time[method2],
    time[method3], time[method4], time[convolve]};
  A = Table[Random[Integer], {1024}, {1024}];
  t1K = {"n/a", "n/a",
    time[method3], time[method4], time[convolve]};
  TableForm[Join[{{" ", "256×256", "512×512", "1024×1024"}},
    Transpose[{{"method 1", "method 2", "method 3",
       "method 4", "convolution"}, t256, t512, t1K}]])]
```

	256×256	512×512	1024×1024
method 1	0.6328125	2.5000000	n/a
method 2	0.5000000	2.0468750	n/a
method 3	0.06005859	0.24218750	1.0156250
method 4	0.09179688	0.35937500	0.8828125
convolution	0.016601563	0.07031250	0.28515625

It is clear from the timing data presented here that grayscale morphological operators present processing challenges. Fortunately, this issue has been addressed in a number of articles in the image processing and mathematical computing literature. In two important articles van Herk [2] and independently Gil and Werman [3] proposed a one−dimensional algorithm with computational complexity which is largely independent of the length of the structuring element. The algorithm breaks the signal into segments whose length matches the size of the structuring element, and computes minima forwards and backwards. A second step combines the results to produce the final result with a fixed cost of three comparisons per sample. Other authors [4, 5] have proposed additional improvements that hold the potential of an additional speedup by a factor of 2. It is clearly of interest to investigate the *Mathematica* and Java/C++ realizations of these algorithms.

■ Morphological reconstruction

Reconstruction from markers is a very important category of morphological operators as many image processing tasks have a natural formulation in terms of these operators. Unlike the fundamental operators presented thus far which take as inputs an image and a structuring element, reconstruction requires two input images. During reconstruction some fundamental operator (for example, dilation or erosion) is applied repeatedly to a so−called marker image, while the result of each operation is constrained by the other of the two images called the mask.

Here I present an example of reconstruction by geodesic dilation for binary images. A sequence of n conditional dilations is known as a size−n geodesic dilation where a conditional dilation is defined as the intersection of the mask image g with a dilation of the marker image f[6].

$$(f \oplus_g s) = (f \oplus s) \cap g \tag{13}$$

Reconstruction of image g from the marker image f is accomplished by repeating the conditional dilations to stability, until the result no longer changes.

$$R_g^{\oplus}(f) = (f \oplus_g s)^n = (((f \oplus_g s) \oplus_g s) \ldots \oplus_g s) \tag{14}$$

where n is such that $(f \oplus_g s)^n = (f \oplus_g s)^{n+1}$.

Here are a marker image, $f = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ and a mask image,

$g = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$.

This shows the results of three consecutive conditional dilations of the marker image with a flat structuring element of dimensions 3×3 at which point further conditional dilations do not change the result.

$(f \oplus_g s)^1 = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$, $(f \oplus_g s)^2 = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$,

$(f \oplus_g s)^3 = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$.

The *Mathematica* implementation of reconstruction by geodesic dilation is straightforward. Here are two image arrays.

```
A = {{0, 0, 1, 0, 0}, {0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0}, {0, 0, 0, 0, 0}, {0, 0, 0, 0, 0}};
B = {{0, 1, 1, 0, 0}, {0, 1, 1, 0, 0}, {1, 1, 1, 0, 1},
    {0, 1, 0, 0, 1}, {0, 0, 0, 1, 1}};
```

The following one−liner computes the reconstruction of B from marker image A.

```
FixedPoint[B BinaryDilate[#, Table[1, {3}, {3}]] &, A] //
 MatrixForm
```

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Clearly, evaluating the reconstruction could be time−consuming as the number of iterations is image dependent. Fortunately, efficient algorithms requiring a limited number of scans of the marker image are known [7]. *Mathematica* implementations of these are currently under development. Here is an example of region filling using reconstruction by geodesic dilation.

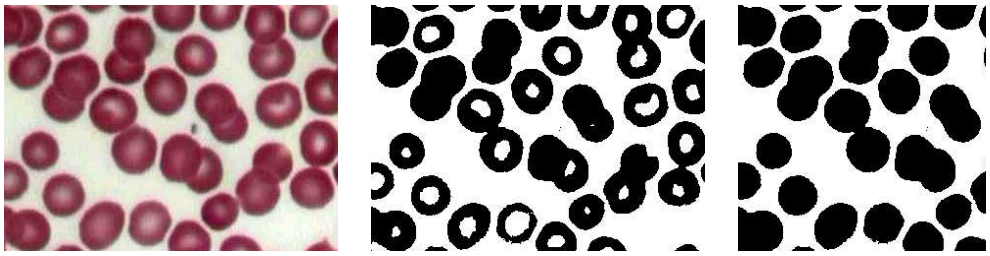
```
mask = Threshold[Round[ToGrayLevel[cells]], 173];
se = Table[1, {3}, {3}];
```

This defines a marker image, an all−zero image with a 1−valued seed placed anywhere within the background region of the original image. This seed defines the starting location of a flooding process that fills the background region.

```
mark = (0 mask)[[1]];
mark[[230, 194]] = 1;
tmp =
  ToGrayLevel[FixedPoint[mask[[1]] BinaryDilate[#, se] &, mark]];
```

This shows the result.

```
DisplayTogetherArray[
  {Graphics[cells], Graphics[mask], Graphics[tmp]},
  ImageSize → {420, 100}];
```



■ References

[1] M. Jankowski and J. P. Kuska, "Connected components labeling − algorithms in *Mathematica*, Java, C++, and C#," Proceeding Sixth International *Mathematica* Symposium," Banff, Canada, August 2004.

[2] M. van Herk, "A fast algorithm for local minimum and maximum filters on rectangular and octogonal kernels," Pattern Recognition Letters, vol. 13, no. 7, pp. 517−521, July 1992.

[3] J. Gil and M. Werman, "Computing 2−D min, median, and max filters," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 15, no. 5, pp. 504−507, May 1993.

[4] J. Gil and R. Kimmel, "Efficient dilation, erosion, opening, and closing algorithms," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 12, pp. 1606−1617, December 2002.

[5] M. Van Droogenbroeck and M. Buckley, "Morphological erosions and openings: fast algorithms based on anchors," Journal of Mathematical Imaging and Vision, vol. 22, pp. 121−142, May 2005.

[6] P. Soille, Morphological Image Analysis, Springer−Verlag, Berlin, 2004.

[7] L. Vincent, "Morphological grayscale reconstruction in image analysis: applications and efficient algorithms," IEEE Transactions on Image Processing, vol. 2, no. 2, pp. 176−201, April 1993.

■ Initializations