Image Processing

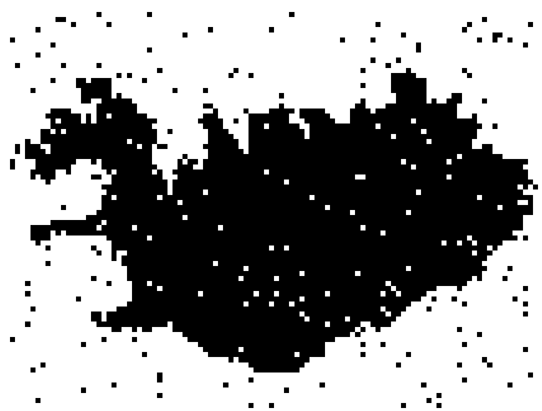
Morphological Operations

Dilation und Erosion

First, for binary images : $D \to \{0,1\}$

AND (Erosion): $y_r = \bigwedge_{r' \in W(r)} x_{r'}$ $\quad y = x \ominus W$

OR (Dilation): $y_r = \bigvee_{r' \in W(r)} x_{r'}$ $\quad y = x \oplus W$

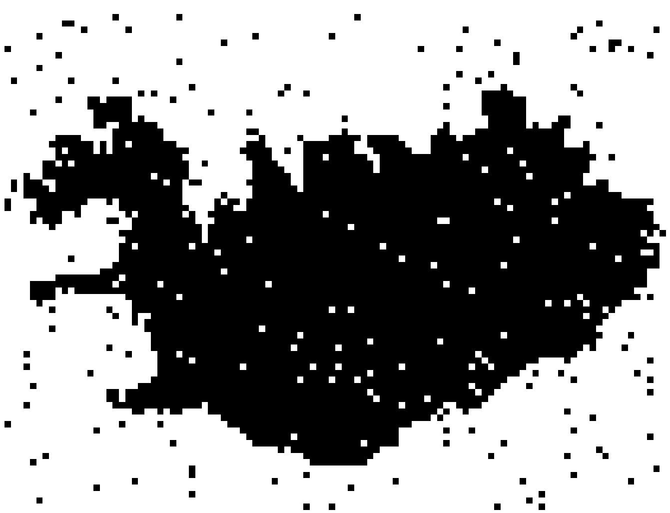


W is called **Structuring Element** (square, circle, etc.)

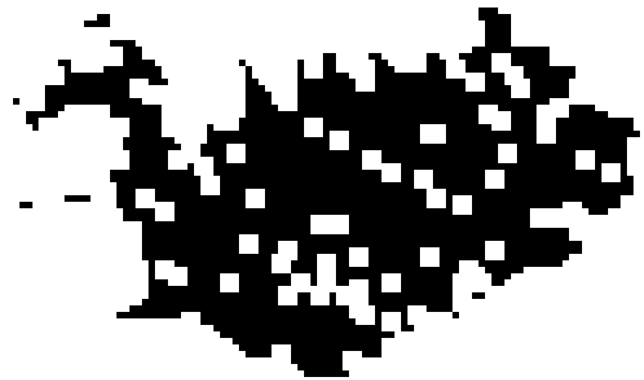
Dilation und Erosion

Original

Erosion



Salt and Pepper

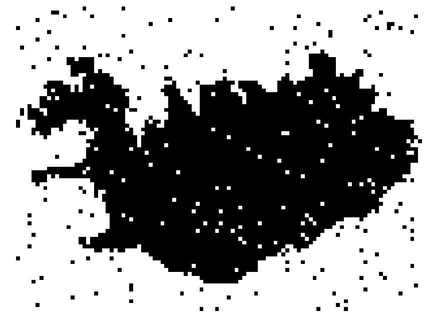


Dilation

Opening and closing

Opening: $x \circ W = \big((x \ominus W) \oplus W\big)$

Closing: $x \bullet W = \big((x \oplus W) \ominus W\big)$



Note: nothing is **commutative**

Closing → than opening

Opening → than closing

Real-Valued Images

Erosion: $y_r = \min_{r' \in W(r)} x_{r'}$

Dilation: $y_r = \max_{r' \in W(r)} x_{r'}$


Closing → than opening




(it is rather senseless for real images but very useful for other types of information, e.g. for Harris-Detector)

Fast minimum

The task in 1D: $y_r = \min_{r'=r-W}^{r+W} x_{r'}$

A naïve algorithm (according to the formula):
for each output enumerate all inputs and take the minimal one.

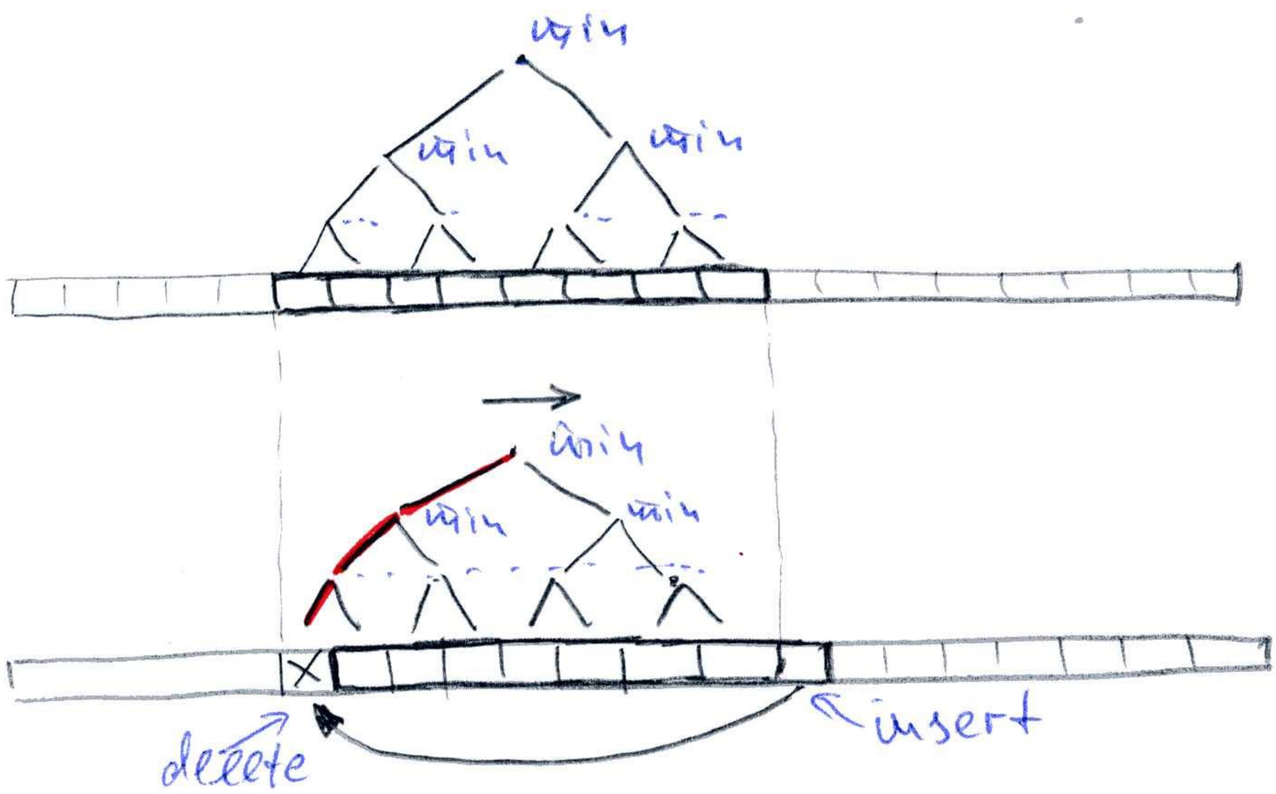
Time complexity: $O(nW)$

The idea:

1. Keep the ordered set of all values
2. For each output one elements should be inserted and one should me removed
3. Use a data structure that allows to do it fast, i.e. in $O(\ln W)$

The overall time complexity is $O(n \ln W)$

Fast minimum



Min-Filter is separable → the time complexity in 2D is $O(n \ln W)$ too !

A generalization

Structuring Element $W \subset \mathbb{R}^2$ becomes
a **Structuring Function** $w : \mathbb{R}^2 \to \mathbb{R}$

Erosion becomes

$$y_r = \min_{r' \in \Omega} \Big(x_{r'} + w(r - r') \Big)$$

"Usual" erosion is a special case:

$$w(x) = \begin{cases} 0 & \text{If} \quad x \in W \\ \infty & \text{otherwise} \end{cases}$$

Compare to the linear filtering

$$y_r = \sum_{r'} x_{r'} \cdot g(r - r')$$

Morphological filtering is a "linear filtering in another semiring":

$$(\mathbb{R}, +, \times) \quad \Rightarrow \quad (\mathbb{R}, \min, +)$$

Distance Transform

Let the foreground pixels are marked by ∞ and the background pixels are marked by 0

The distance transform is

$$y_r = \min_{r' \in D} (x_{r'} + d(r, r'))$$

i.e. for each foreground pixel the distance to the closest background one. For example Euclidian distance, block-distance etc.

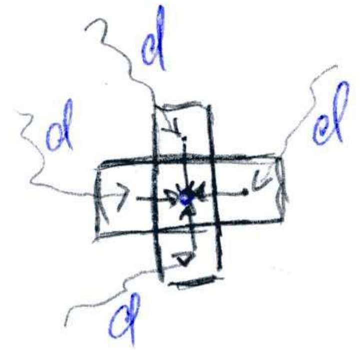
Distance Transform, Algorithms

(for the Block-distance)

A parallel version:

Initialize all foreground pixels with ∞ and the background pixels with 0 . Repeat for each pixel:

```
d(i,j)=min(d(i,j),
      d(i-1,j)+1, d(i,j-1)+1,
      d(i+1,j)+1, d(i,j+1)+1)
```



as long as something is changed.

Assume that all pixels can be processed in parallel,
then the time complexity is proportional to the longest distance.

Distance Transform, Algorithms

Sequential algorithm:

If for a pixel the closest one is located e.g. “left-top”, the distance can be found by considering only the “left-top” pixels.

```
for i=1…n, for j=1…m
        d(i,j)=min(d(i,j), d(i-1,j)+1, d(i,j-1)+1)

for i=1…n, for j=m…1
        d(i,j)=min(d(i,j), d(i-1,j)+1, d(i,j+1)+1)

for i=n…1, for j=1…m
        d(i,j)=min(d(i,j), d(i+1,j)+1, d(i,j-1)+1)

for i=n…1, for j=m…1
        d(i,j)=min(d(i,j), d(i+1,j)+1, d(i,j+1)+1)
```

Linear time complexity.

Some interesting topics:

1. Euclidean Distance Transform can be done In linear time as well (the dimensionality does not matter)

2. A bit of topology – connected components, skeletonization

3. A bit more topology and digital geometry – digital straight lines, arcs, polygons, manifolds etc.