Summer term 2024 – Cyrill Stachniss

Photogrammetry & Robotics Lab

Binary Images and Commonly Used Operations

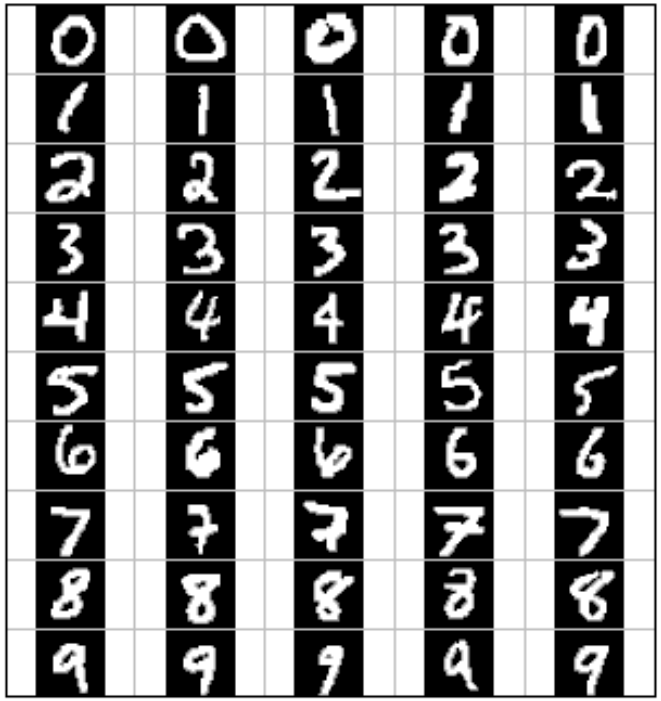
(connected components; distance transform; morphological operators)

Cyrill Stachniss

The slides have been created by Cyrill Stachniss.

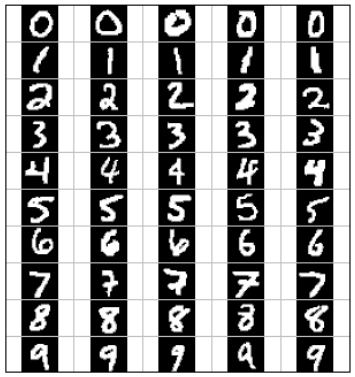
Binary Image

- So far, we considered grayscale images where each pixel typically takes values between 0 and 255
- A **binary image** is an image with a **1 bit color depth**
- Each pixel is **either black or white**
- Either 0 or 1 (0 or 255)



Binary Image Examples

Scanned documents



Handwritten digits

Image Courtesy: Leibe

Background Subtraction

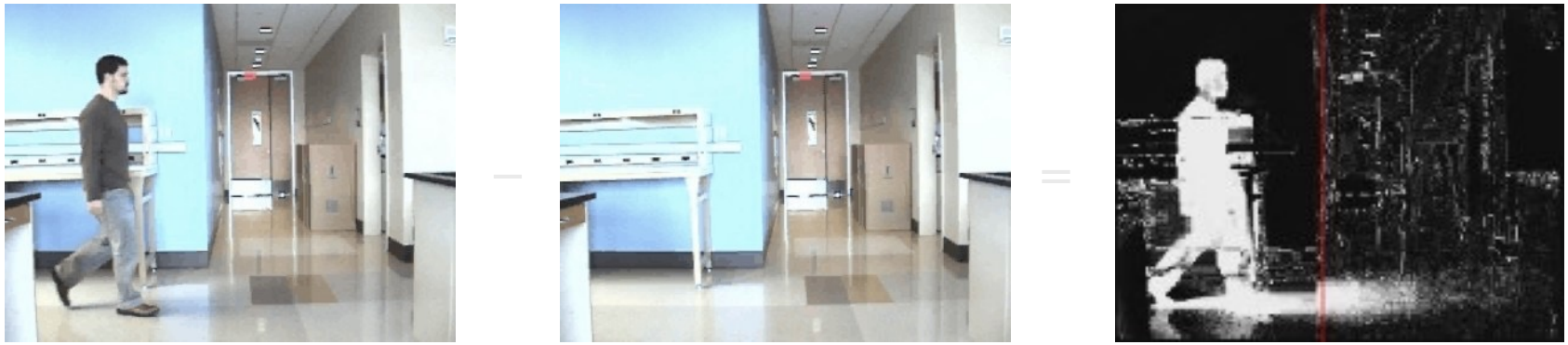
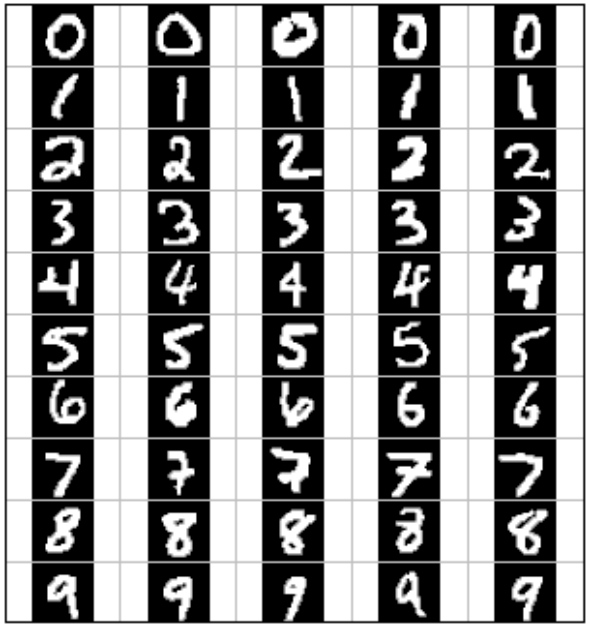


Image Courtesy: Jäggli

Connected Components

Connected Components

- For several applications, we need to identify which components are connected
- Example: Separation of characters



Connected Components

Two points A, B are connected if there exists a path from A to B that goes only through the same component

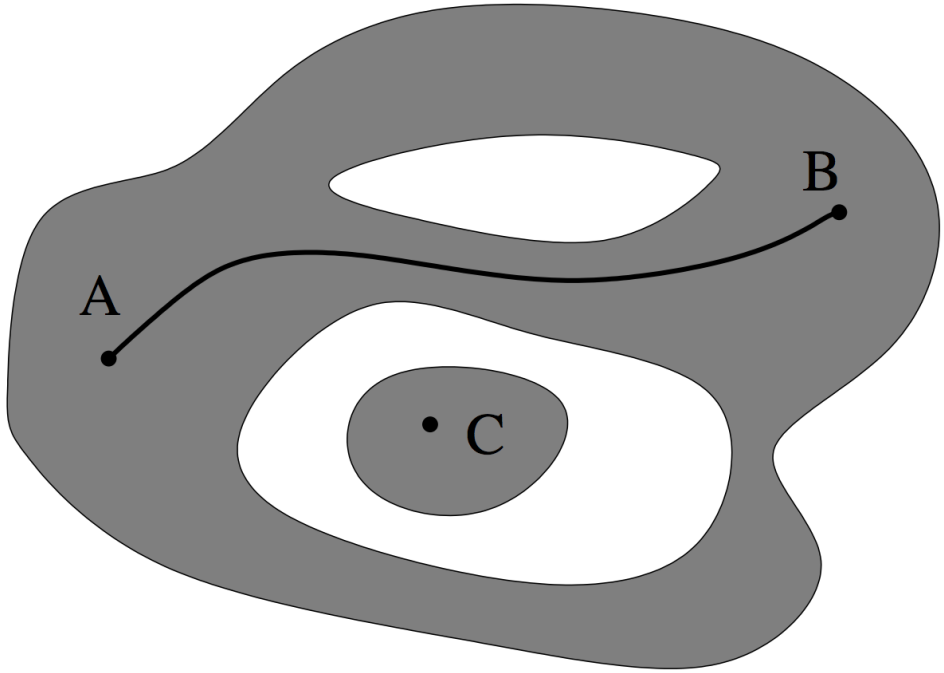
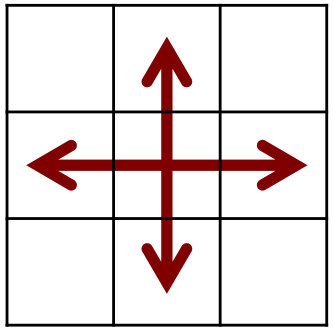


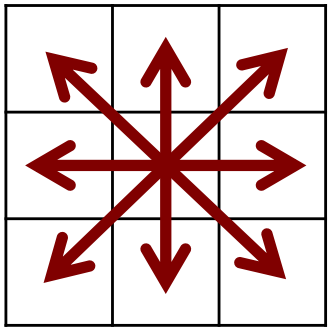
Image courtesy: Förstner

Neighborhoods on Grids

- We need to define a neighborhood relationship for pixels
- Two popular neighborhood definitions



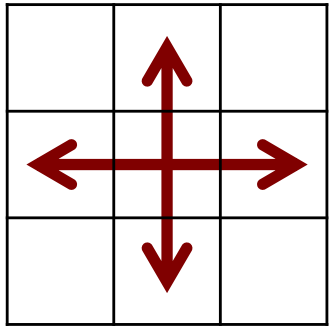
N4 neighborhood



N8 neighborhood

N4 Neighborhood

- The left, right, up, down pixels are connected

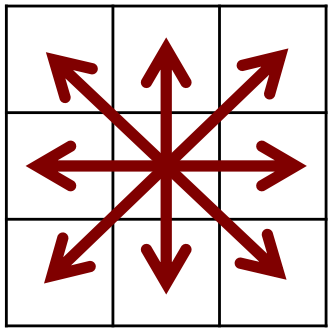


$$\mathcal{N}_4(i,j) = \{(i,j-1),(i-1,j),(i,j+1),(i+1,j)\}$$

- Also called "city-block" neighborhood or Manhattan neighborhood

N8 Neighborhood

- The left, right, up, down and the diagonal pixels are connected



$$\begin{aligned}\mathcal{N}_8(i,j) \quad = \quad & \{(i,j-1),(i-1,j-1),(i-1,j),(i-1,j+1),\\ & (i,j+1),(i+1,j+1),(i+1,j),(i+1,j-1)\}\end{aligned}$$

How to Determine Connected Components?

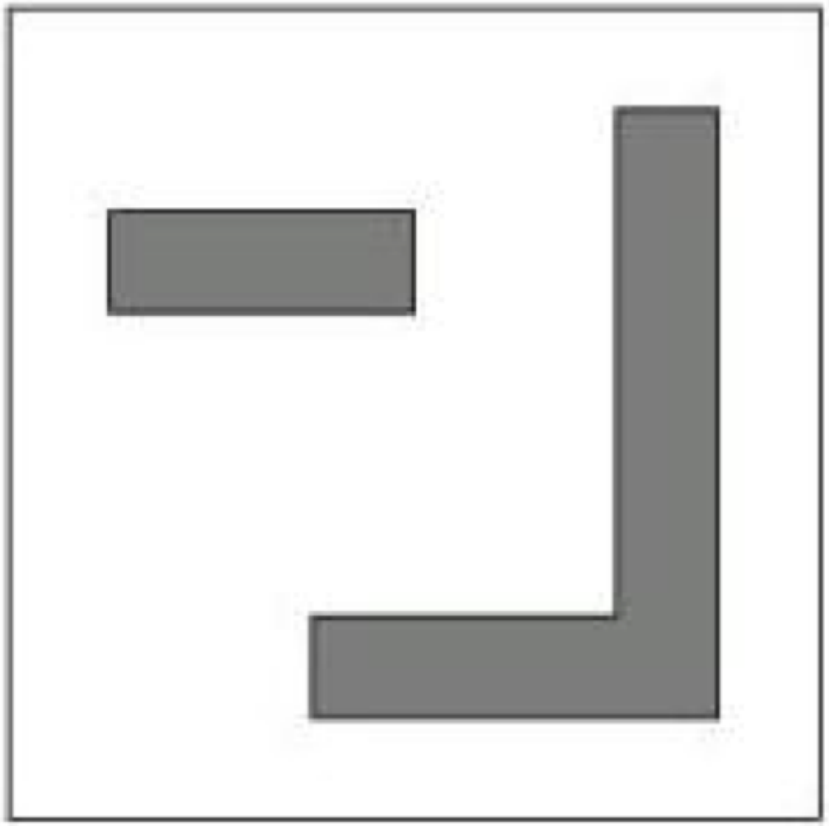
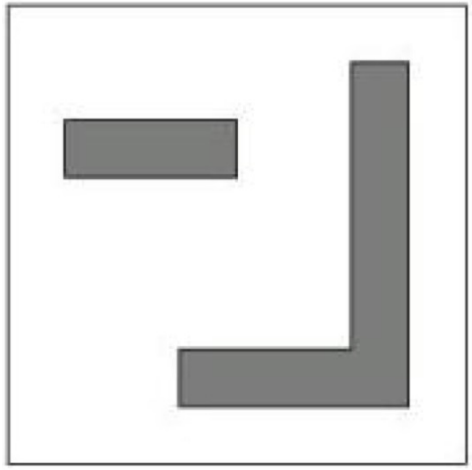
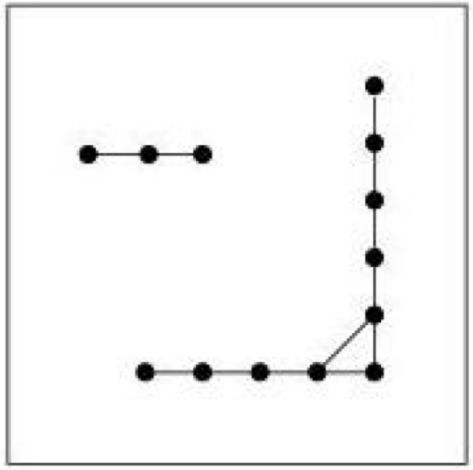


Image courtesy: Förstner

Determining Connected Components via a Graph

- Build graphs of the foreground pixels with edges according to N8 neighbors
- Idea: Label the nodes in the graphs





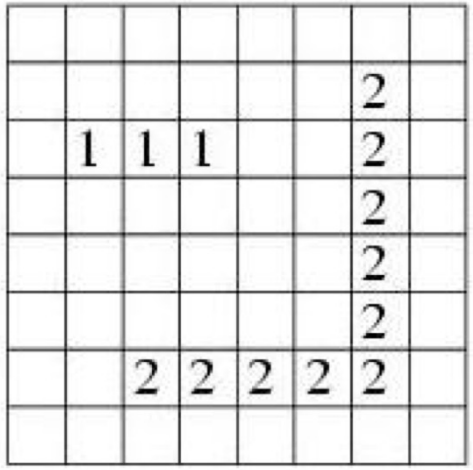


Image courtesy: Förstner

Labeling Approach Informally

1. Select an unlabeled node and assign a new label to it
2. For each unlabeled neighbor of a labeled node, assign the same label
3. Repeat step 2 until all neighbors are labeled
4. Repeat step 1 until all nodes are labeled

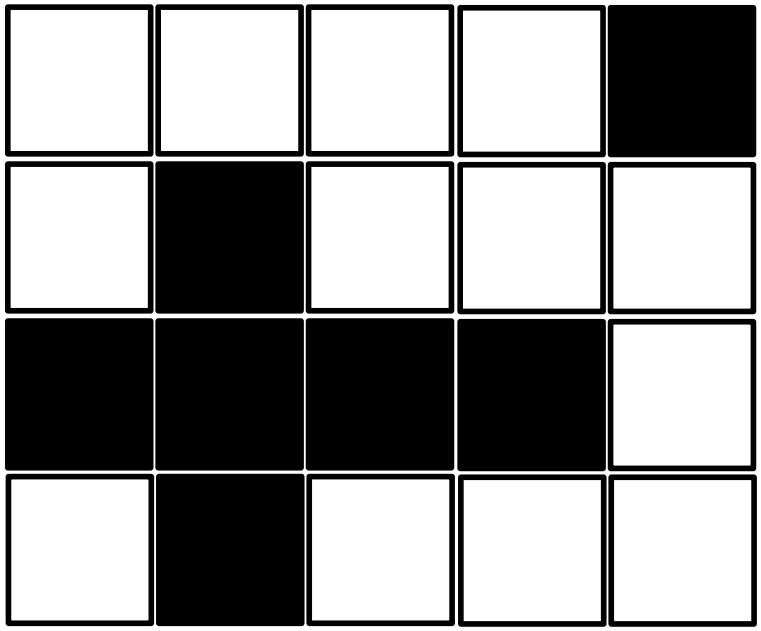
Also called: a “brushfire” approach

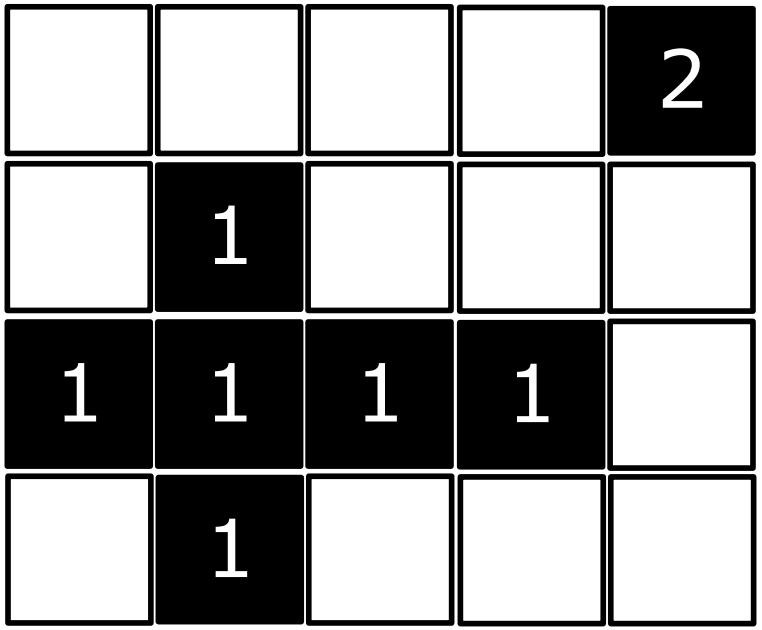
Connected Components

Algorithm 1: connected components
Input: binary image $b(i,j) \in \{0,1\}$
Output: number of components K, component image $k(i,j) \in \{0:K\}$

component number $K = 0$, component image $k(i,j) = 0$;
repeat
  find $(i,j)|\{b(i,j) = 1, k(i,j) = 0\}$;
  $\mathcal{S} := \{(i,j)\}$;
  $K := K + 1$;
  $k(i,j) := K$;
  repeat
    find unlabeled foreground neighbors $\mathcal{N}(\mathcal{S} \mid b(i,j) = 1, k(i,j) = 0)$;
    label all $(i,j) \in \mathcal{N}(\mathcal{S})$ with K: $k(i,j) = K$;
    $\mathcal{S} = \mathcal{S} \cup \mathcal{N}(\mathcal{S})$;
  until no neighbor exists: $\mathcal{N}(\mathcal{S}) = \emptyset$;
until no unlabeled forground pixel exists:
$\nexists(i,j)|\{b(i,j) = 1, k(i,j) = 0\}$;

Example





Steps (outer loop, 1st iteration)

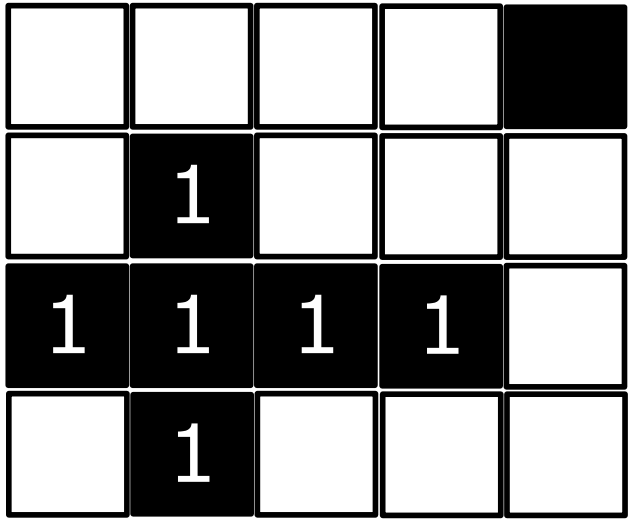
outer loop 1 $\quad \mathcal{S} = \{(3,3)\}$

inner loop 1
$$\mathcal{N}(\mathcal{S}) = \{(3,2),(3,4)\}$$
$$\mathcal{S} = \{(3,3),(3,2),(3,4)\}$$

inner loop 2
$$\mathcal{N}(\mathcal{S}) = \{2,2),(3,1),(4,2)\}$$
$$\mathcal{S} = \{(3,3),(3,2),(3,4),(2,2),(3,1),(4,2)\}$$

inner loop 3 $\quad \mathcal{N}(\mathcal{S}) = \{\}$

```
2  repeat
3  |  find (i,j)|{b(i,j) = 1, k(i,j) = 0};
4  |  S := {(i,j)};
5  |  K := K + 1;
6  |  k(i,j) := K;
7  |  repeat
8  |  |  find unlabeled foreground neighbors N(S | b(i,j) = 1, k(i,j) = 0);
9  |  |  label all (i,j) ∈ N(S) with K: k(i,j) = K;
10 |  |  S = S ∪ N(S);
11 |  until no neighbor exists: N(S) = ∅;
12 until no unlabeled forground pixel exists:
   ∄(i,j)|{b(i,j) = 1, k(i,j) = 0};
```

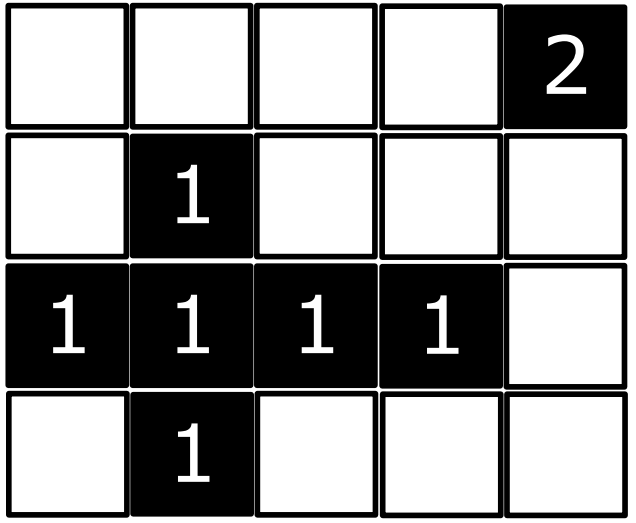


Steps (outer loop, 2nd iteration)

outer loop 2 $\mathcal{S} = \{(1,5)\}$

inner loop 1 $\mathcal{N}(\mathcal{S}) = \{\}$

2 **repeat**
3 | find $(i,j)|\{b(i,j)=1, k(i,j)=0\}$;
4 | $\mathcal{S} := \{(i,j)\}$;
5 | $K := K+1$;
6 | $k(i,j) := K$;
7 | **repeat**
8 | | find unlabeled foreground neighbors $\mathcal{N}(\mathcal{S} \mid b(i,j)=1, k(i,j)=0)$;
9 | | label all $(i,j) \in \mathcal{N}(\mathcal{S})$ with K: $k(i,j)=K$;
10 | | $\mathcal{S} = \mathcal{S} \cup \mathcal{N}(\mathcal{S})$;
11 | **until** no neighbor exists: $\mathcal{N}(\mathcal{S}) = \emptyset$;
12 **until** no unlabeled forground pixel exists: $\nexists(i,j)|\{b(i,j)=1, k(i,j)=0\}$;



Properties of the Algorithm

- Provides the connected components
- Work in general graphs
- Does not exploit the systematic neighborhood of images

Is there any property we can exploit to make the algorithm more efficient?

Labeling by Exploiting the Grid Structure of the Image

Idea

- Process the image in one pass
- Generate a temporary label for a foreground pixel based on the already processed neighbors
- In case of multiple labels for the same component, use an equivalence table

Connected Components on Grids

- Process the image from left to right, top to bottom:

  If the next pixel to process is 1

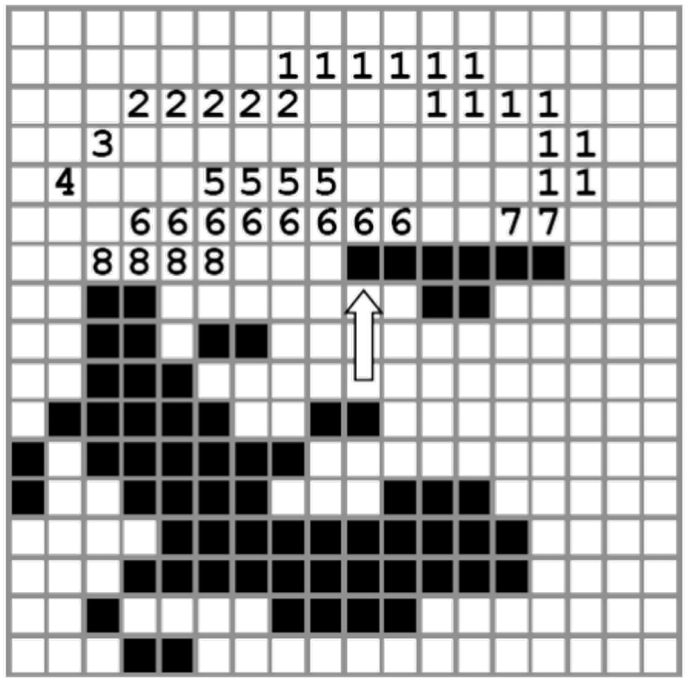
N4 

A) If none of its (top or left) neighbor is 1, assign new label.

B) If all neighboring (top or left) labels are identical, copy the label

C) If the labels differ, copy the one label (e.g., min or max) and update the equivalence table.

NOTE: Example uses max to select the label



Connected Components on Grids

- Process the image from left to right, top to bottom:

  If the next pixel to process is 1

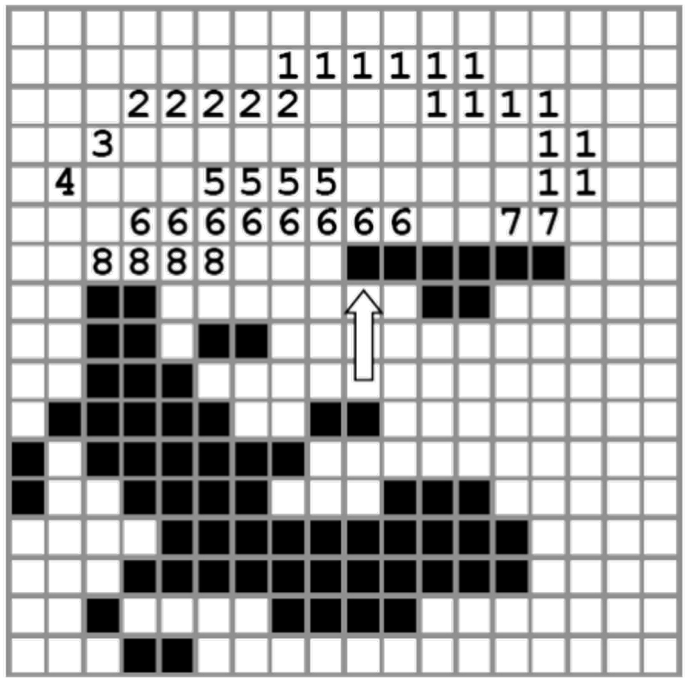
N4 

A) If none of its (top or left) neighbor is 1, assign new label.

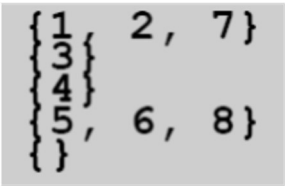
B) If all neighboring (top or left) labels are identical, copy the label

C) If the labels differ, copy the one label (e.g., min or max) and update the equivalence table.

NOTE: Example uses max to select the label



- Re-label with the smallest of equivalent labels
- Extends trivially for N8 neighborhoods



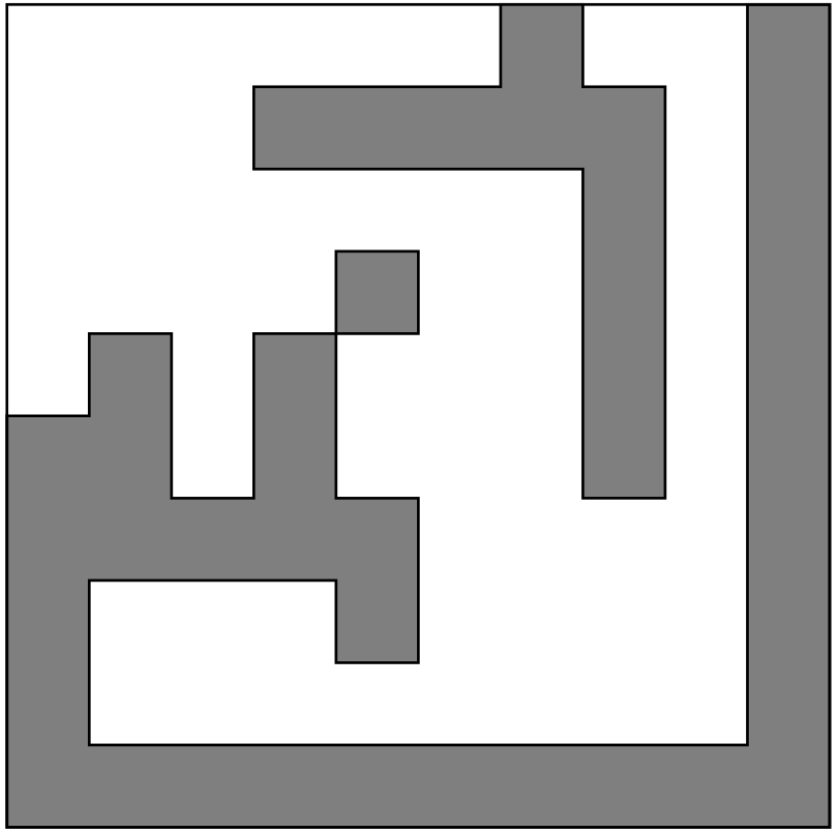
Slide courtesy: J. Neira

Connected Components for Grids/Binary Images

Algorithm 1: connected component in binary image
Input: binary image $b(i,j) \in \{0,1\}$
Output: component number K, component image $k(i,j) \in \{0:K\}$

```
1  component number K = 0, equivalence table E = ∅ ;
2  for i = 0 : I − 1 do all rows
3  |  for j = 0 : J − 1 do all columns
4  |  |  if b(i,j) = 1 then
5  |  |  |  A = N(i,j) all labeled neighbors;
6  |  |  |  if |A| = 0 then
7  |  |  |  |  K := K + 1;
8  |  |  |  |  k(i,j) := K;
9  |  |  |  end
10 |  |  |  if |A| ≥ 1 then
11 |  |  |  |  k(i,j) := min(k(A)); // alternative: max instead of min;
12 |  |  |  |  forall the x in A with k(x) ≠ k(i,j) do
13 |  |  |  |  |  extend equivalence table: E := E ∪ {k(i,j), k(x)};
14 |  |  |  |  end
15 |  |  |  end
16 |  |  end
17 |  end
18 end
19 compute connected components of E using previous Algorithm;
20 replace k(i,j), i ∈ I, j ∈ J with smallest number of the component in
   equivalence graph;
```

Example (N8 Neighborhood)



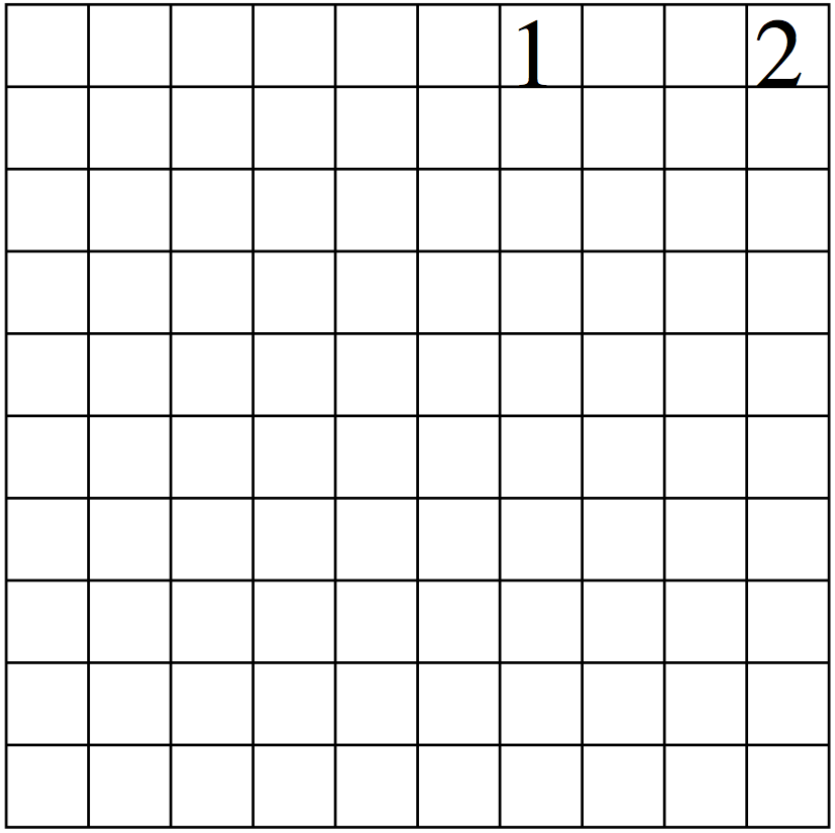
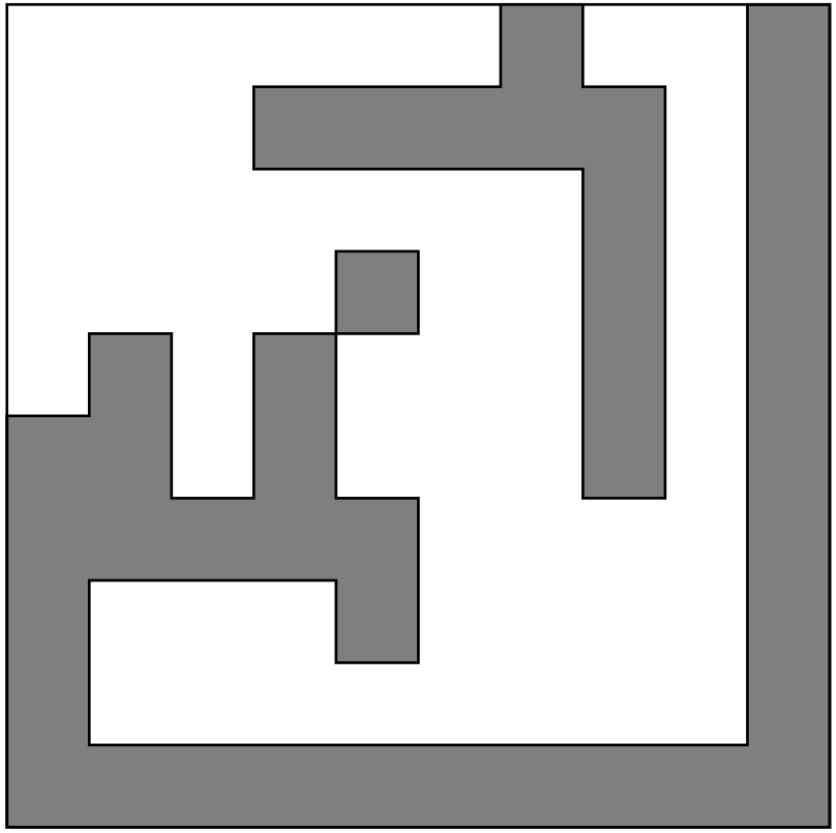


Image courtesy: Förstner

Example (N8 Neighborhood)



						1			2
			3	3	1	1	1		2
							1		2
				4			1		2
	5		4				1		2
5	5		4				1		2
5	5	4	4	4					2
5				4					2
5									2
5	5	5	5	5	5	5	5	2	2

Equivalence table: {1=3, 2=4=5}

Image courtesy: Förstner

Connected Components for Grids/Binary Images

- Exploits the grid neighborhood
- Requires only one pass through the image for labeling
- Second pass to eliminate duplicate labels
- Linear complexity in the number of foreground pixels

Distance Transform

Distance Transformation

Several problems require to compute the distance from any pixel to the border of the components

Examples

- Nearest neighbor problems
- Sensor models for range sensor
- Map visualization
- User interfaces

Distance Transformation

- Distance transform on the foreground pixels is defined as

$$d(r,c) = \begin{cases} \min_{(u,v)\in\partial R} [D_x((r,c),(u,v))], & \text{if } b(r,c)=1 \\ 0, \text{otherwise} \end{cases}$$

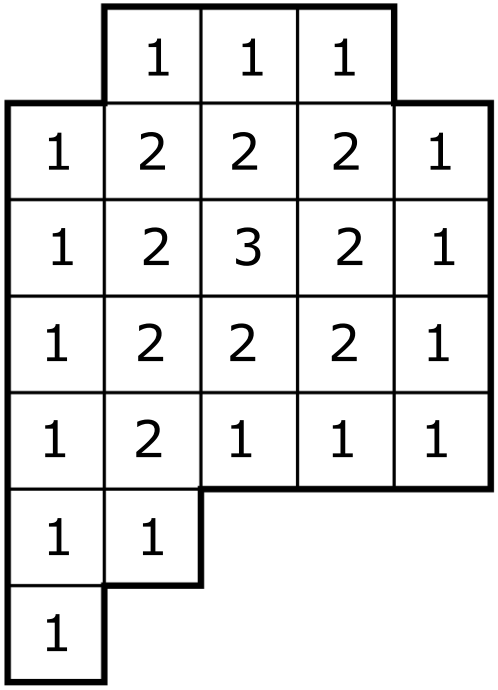
border

- with different distance functions

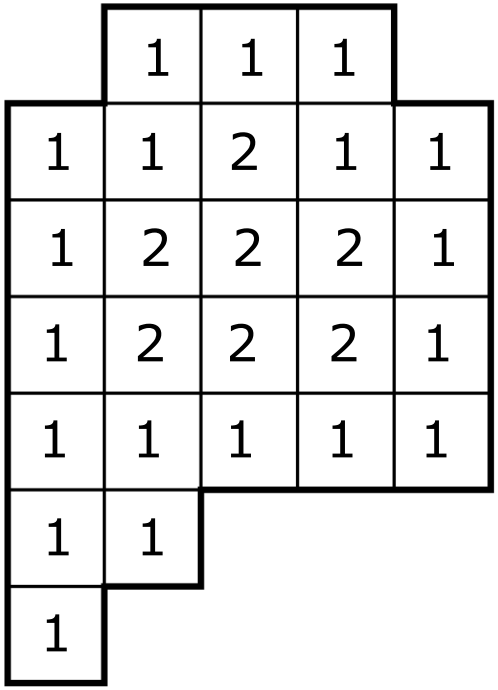
$$D_4((r,c),(u,v)) = |u-r| + |v-c|$$
$$D_8((r,c),(u,v)) = \max(|u-r|, |v-c|)$$

Examples

N4



N8

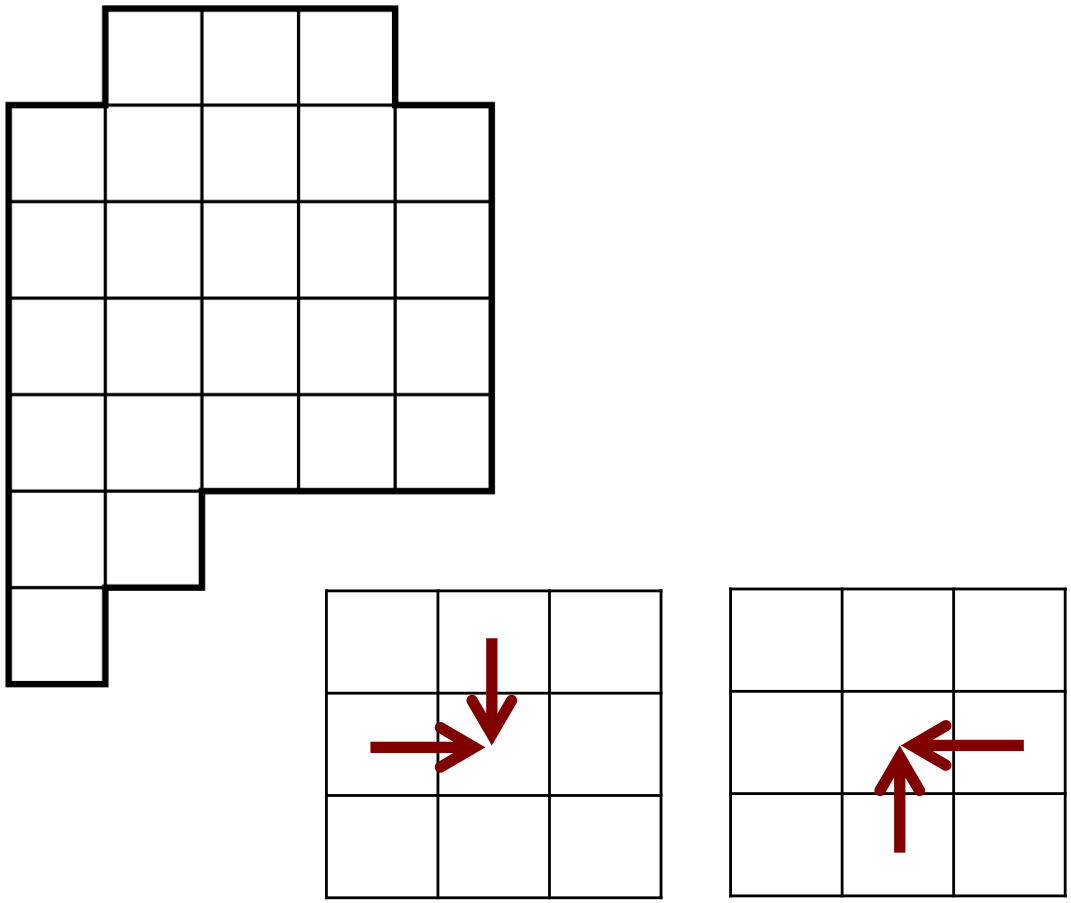


Distance Transformation

- Distance transform can be computed similar to the connected components
- Two passes over the image
  - 1st: top-down, left-right
  - 2nd: down-up, right-left
- Always store the minimum distance

First and Second Pass (N4)

First pass



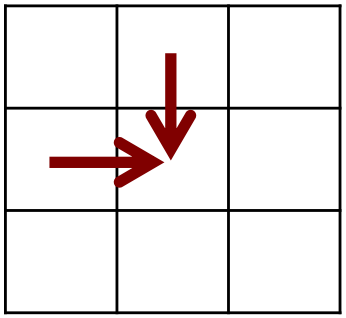
1st pass dist. 2nd pass dist.

First and Second Pass (N4)

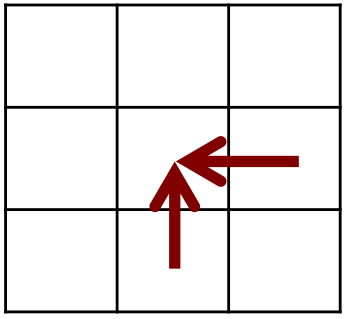
First pass

Second pass

	1	1	1	
1	2	2	2	1
1	2	3	3	2
1	2	3	4	3
1	2	3	4	4
1	2			
1				



1st pass dist.

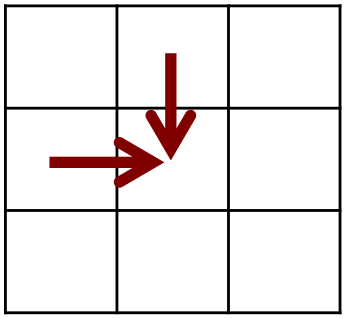


2nd pass dist.

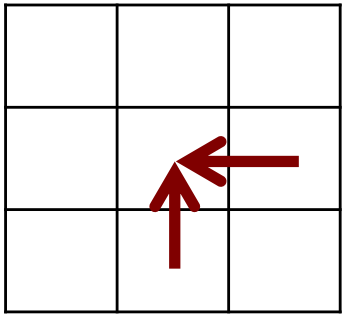
First and Second Pass (N4)

First pass

	1	1	1	
1	2	2	2	1
1	2	3	3	2
1	2	3	4	3
1	2	3	4	4
1	2			
1				



1st pass dist.



2nd pass dist.

Second pass

	1	1	1	
1	2	2	2	1
1	2	3	2	1
1	2	2	2	1
1	2	1	1	1
1	1			
1				

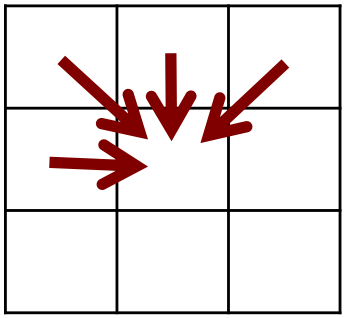
Analogous for N8 Neighborhood

First pass

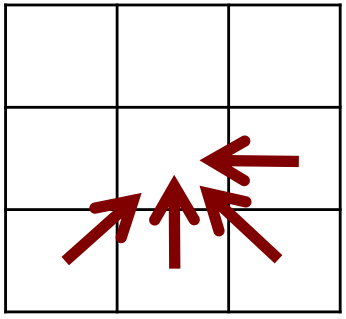
Second pass

	1	1	1	
1	1	2	1	1
1	2	2	2	1
1	2	3	2	1
1	2	3	2	1
1	2			
1				

	1	1	1	
1		2	1	1
1	2	2	2	1
1	2	2	2	1
1	1	1	1	1
1	1			
1				



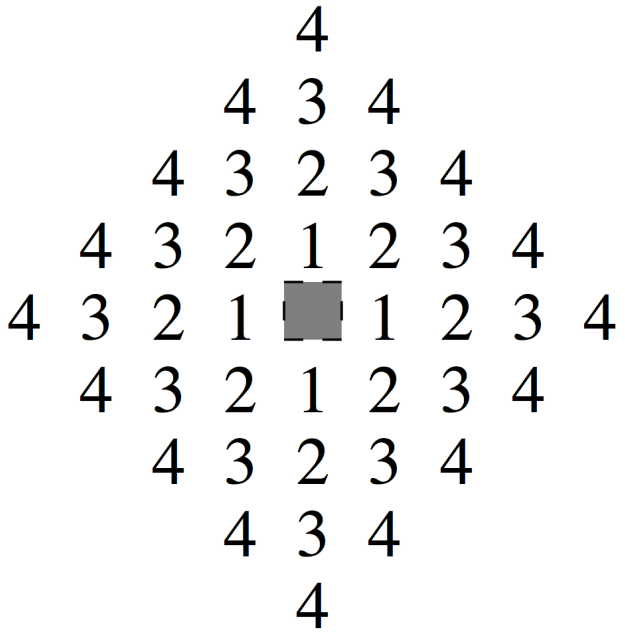
1st pass dist.



2nd pass dist.

Examples for N4 and N8 Distances to the Center

N4



N8

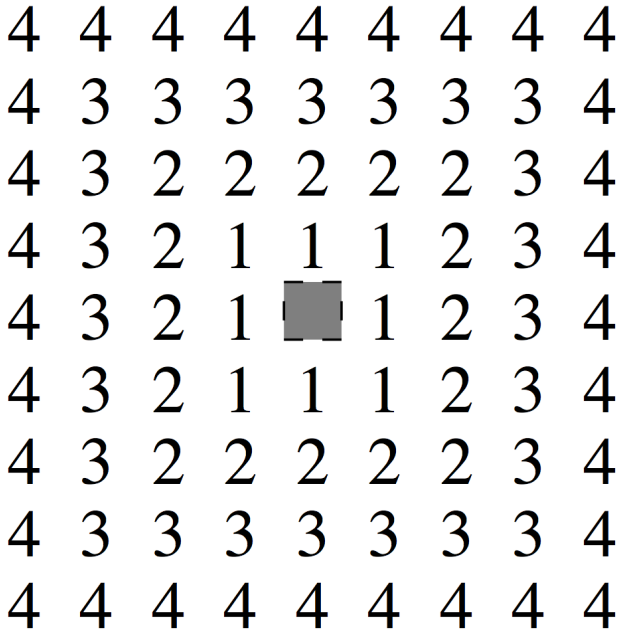


Image courtesy: Förstner

N4 Distance Transformation

1. Initialization

$$\forall(r,c) \quad d(r,c) = \begin{cases} \max, & \text{if } b(r,c) = 1 \\ 0, & \text{otherwise} \end{cases}$$

2. 1st pass (top-left to bottom-right)

$$d(r,c) = \min[d(r,c), d(r,c-1)+1, d(r-1,c)+1]$$

3. 2nd pass (bottom-right to top-left)

$$d(r,c) = \min[d(r,c), d(r,c+1)+1, d(r+1,c)+1]$$

N8 Distance Transformation

1. Initialization

$$\forall(r,c) \quad d(r,c) = \begin{cases} \max, & \text{if } b(r,c) = 1 \\ 0, & \text{otherwise} \end{cases}$$

2. 1st pass (top-left to bottom-right)

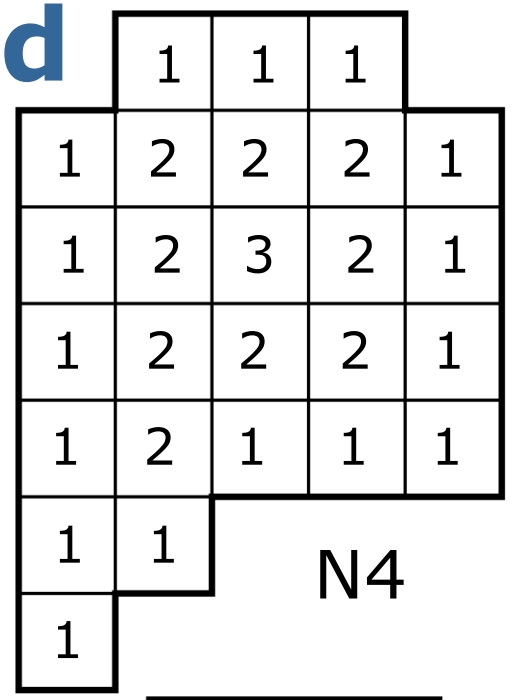
$$\begin{aligned} d(r,c) \quad = \quad & \min[d(r,c), d(r,c-1)+1, d(r-1,c)+1, \\ & d(r-1,c-1)+1, d(r-1,c+1)+1] \end{aligned}$$

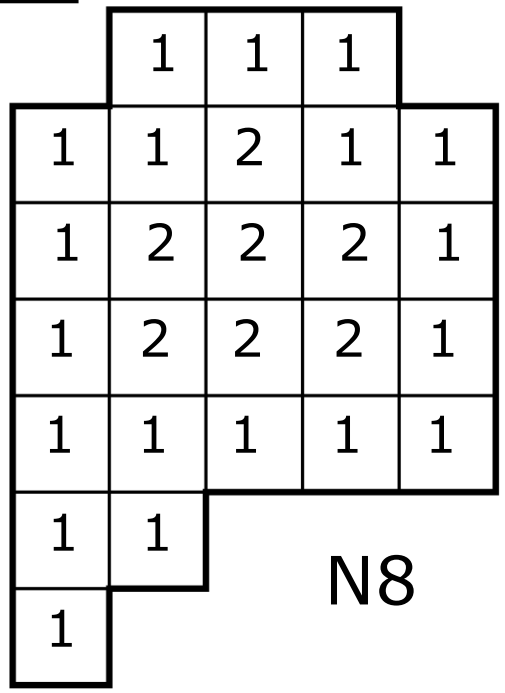
3. 2nd pass (bottom-right to top-left)

$$\begin{aligned} d(r,c) \quad = \quad & \min[d(r,c), d(r,c+1)+1, d(r+1,c)+1, \\ & d(r+1,c+1)+1, d(r+1,c-1)+1] \end{aligned}$$

N4 vs. N8 Neighborhood

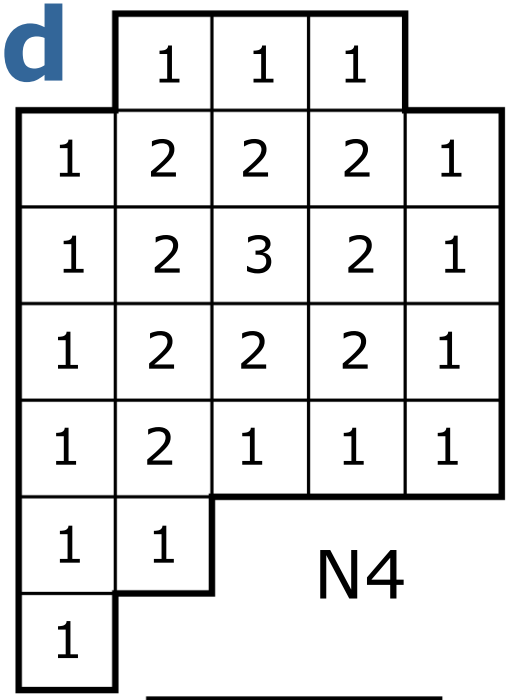
- The N4 neighborhood overestimates the Euclidean distance
- The N8 neighborhood underestimates the Euclidean distance

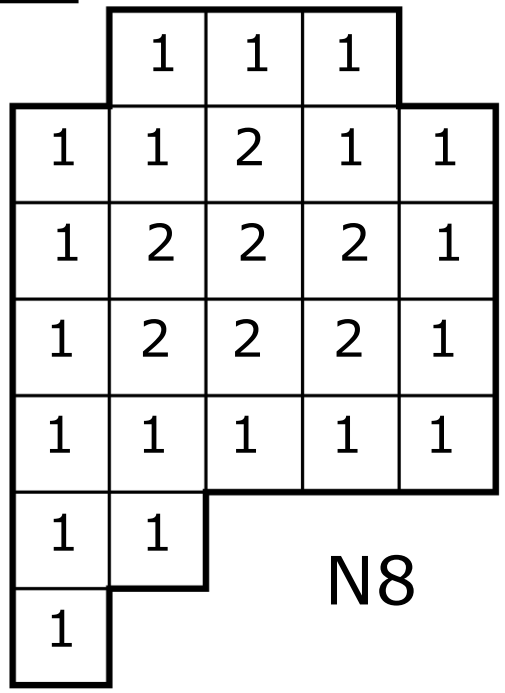




N4 vs. N8 Neighborhood

- The N4 neighborhood overestimates the Euclidean distance
- The N8 neighborhood underestimates the Euclidean distance
- Can we efficiently combine both?





Combined Distance

- The real cost of the diagonal is $\sqrt{2}$
- If we approximate $\sqrt{2} \approx 3/2$
- The sum $D_4 + D_8$ provides a better approximation for twice the distance
- Thus we can use the average distance

$$D_o = \frac{1}{2}(D_4 + D_8)$$

- By using $D_o = (D_4 + D_8)$ we can exploit integer computations

Euclidian Distance Transform

- Computing the real Euclidian distance for every cell to the closest border is more difficult
- EDT in Python (scipy) available as ndimage.morphology.distance_transform_edt
- EDT in Matlab available as bwdist()
- See, for example,: Breu et al., 1995

Morphological Operators: Erosion and Dilation

Filtering

- Binary images are often obtained through thresholding (point operator)
- Operator: $b(a) = \begin{cases} 0 \text{ if } a < T \\ 1 \text{ otherwise} \end{cases}$

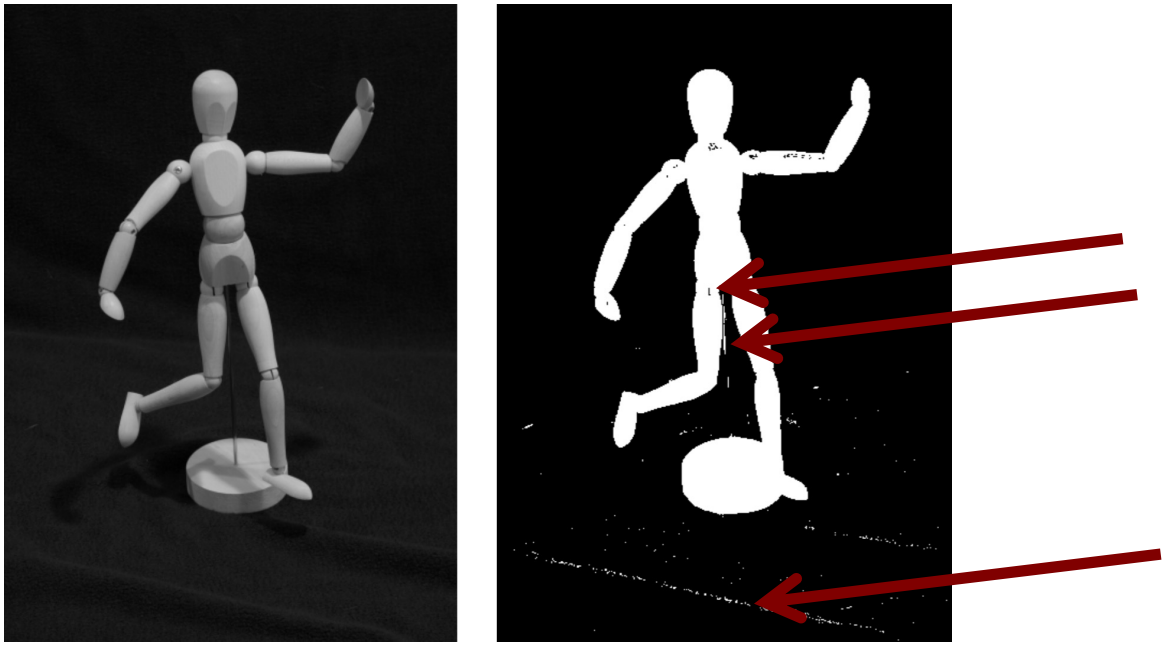
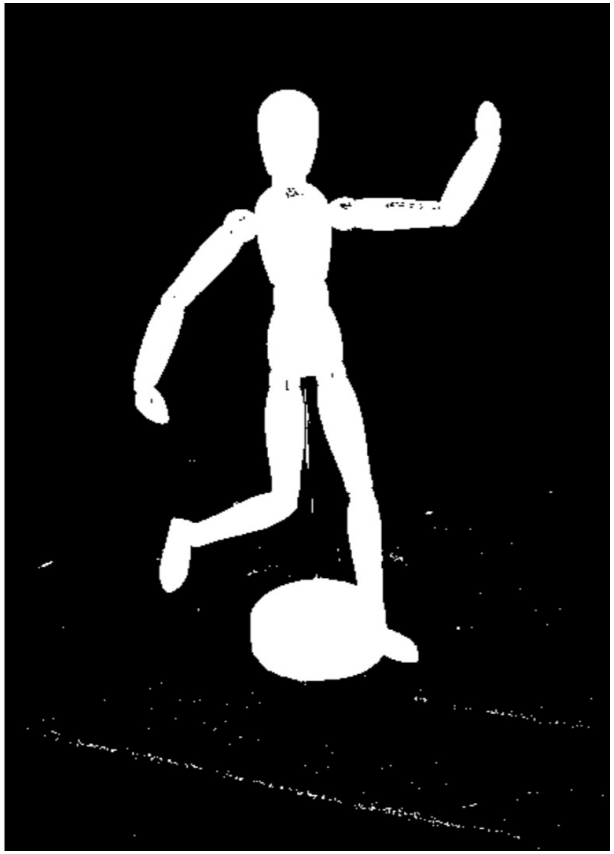


Image Courtesy: Whitaker

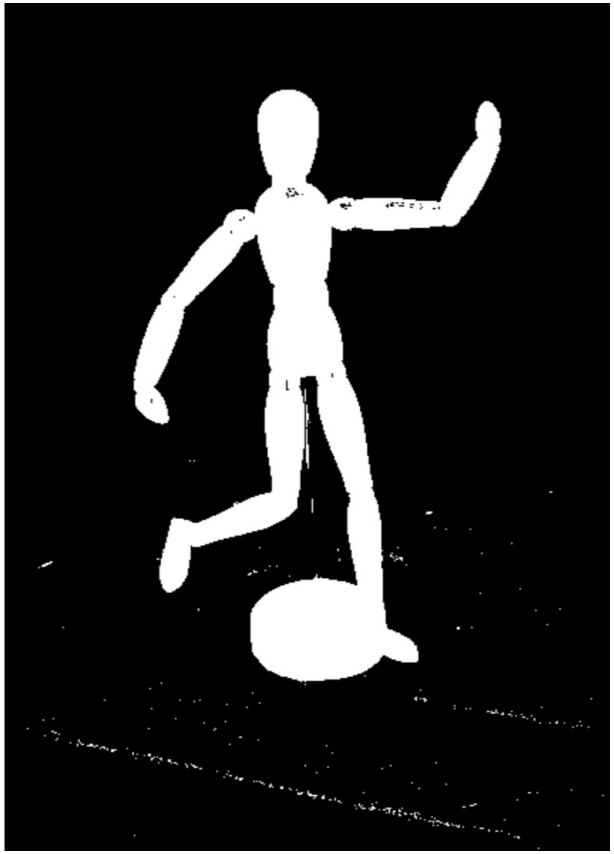
What We Have…



How to Fix This?

Image Courtesy: Whitaker

What We Want…



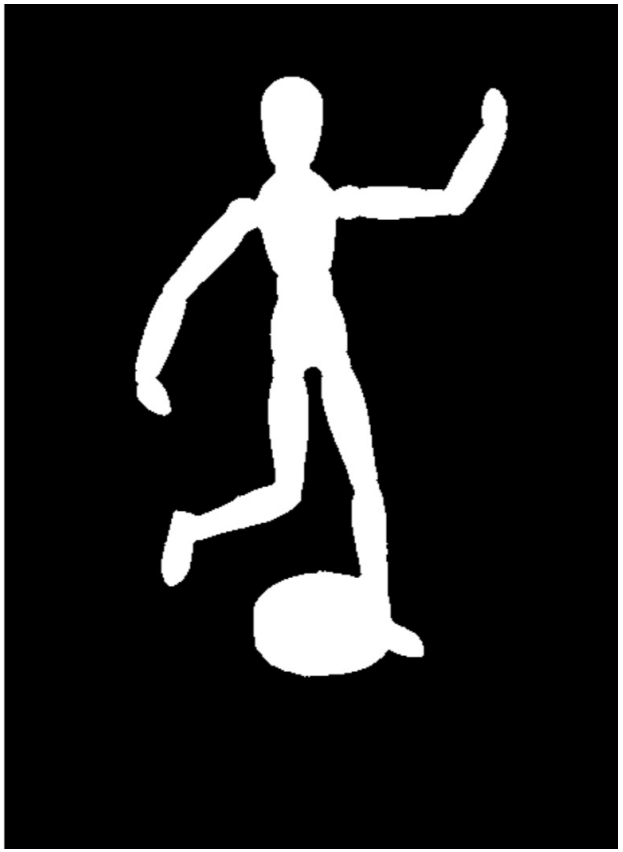


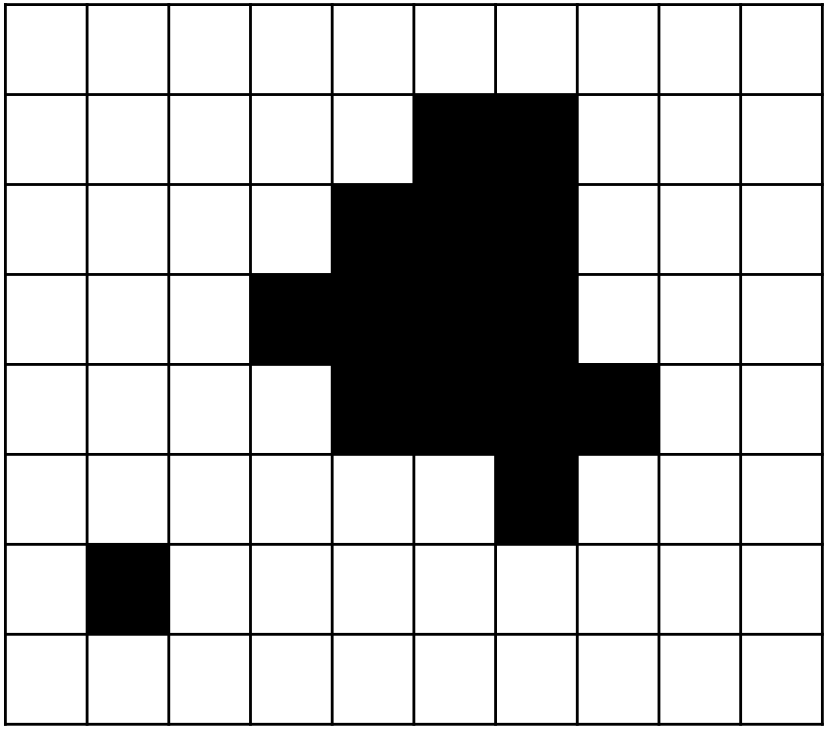
Image Courtesy: Whitaker

Morphological Operators

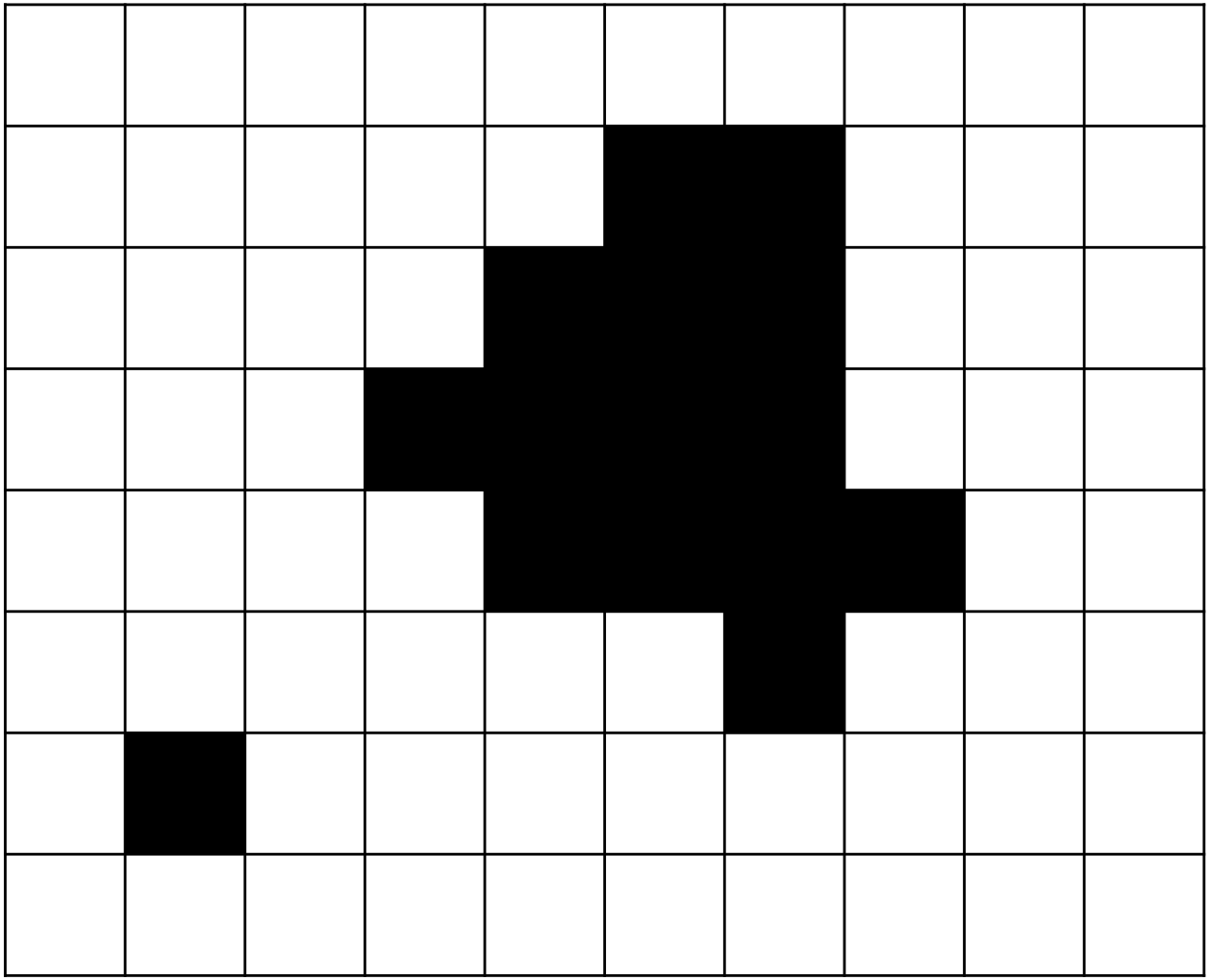
- Shrinking the foreground (“erosion”)
- Expanding the foreground (“dilation”)
- Removing holes in the foreground (“closing”)
- Removing stray foreground pixels in background (“opening”)
- …

Erosion

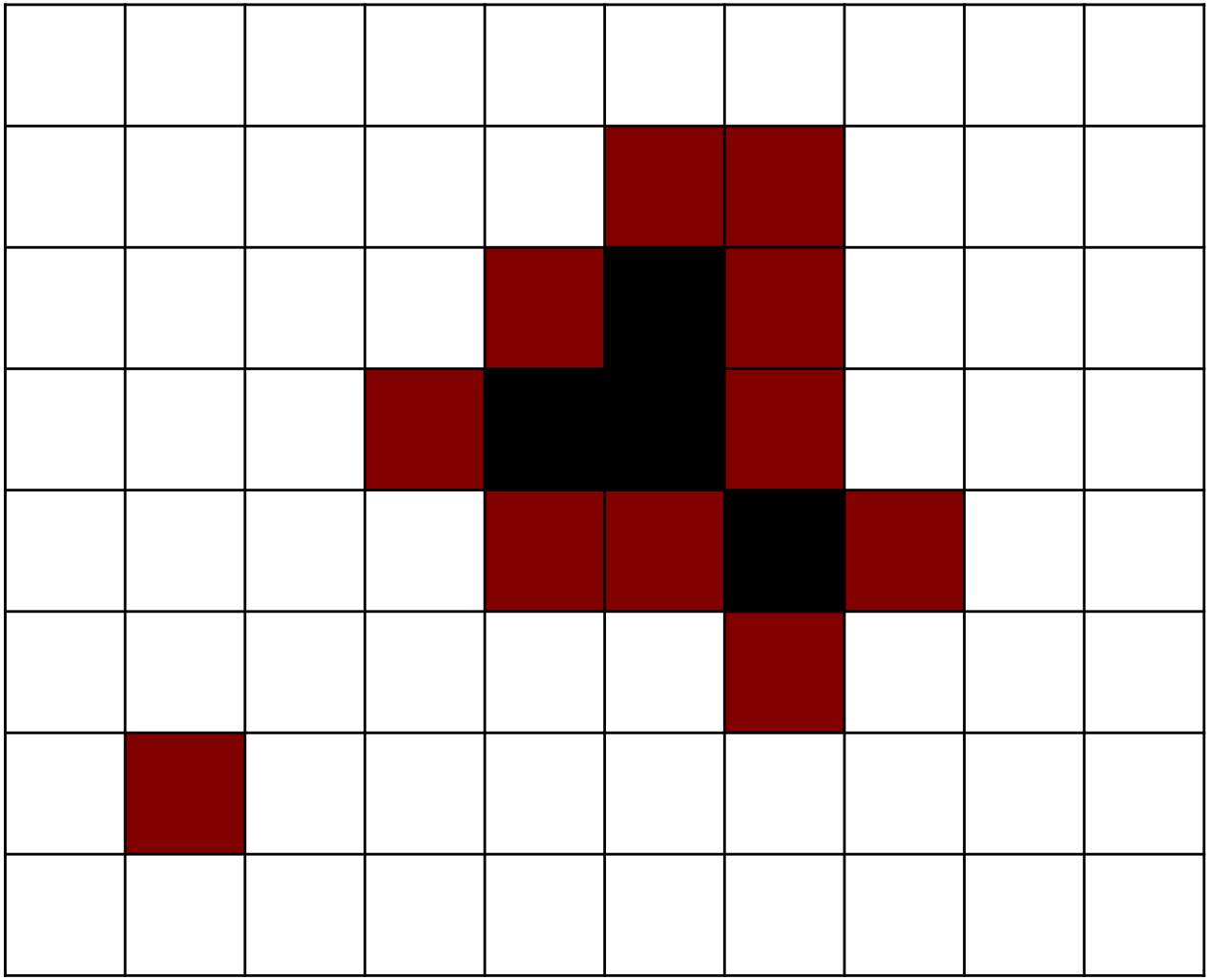
Change each foreground (*here* black) pixel to a background (*here* white) pixel if it has a background pixel as its N4 neighbor



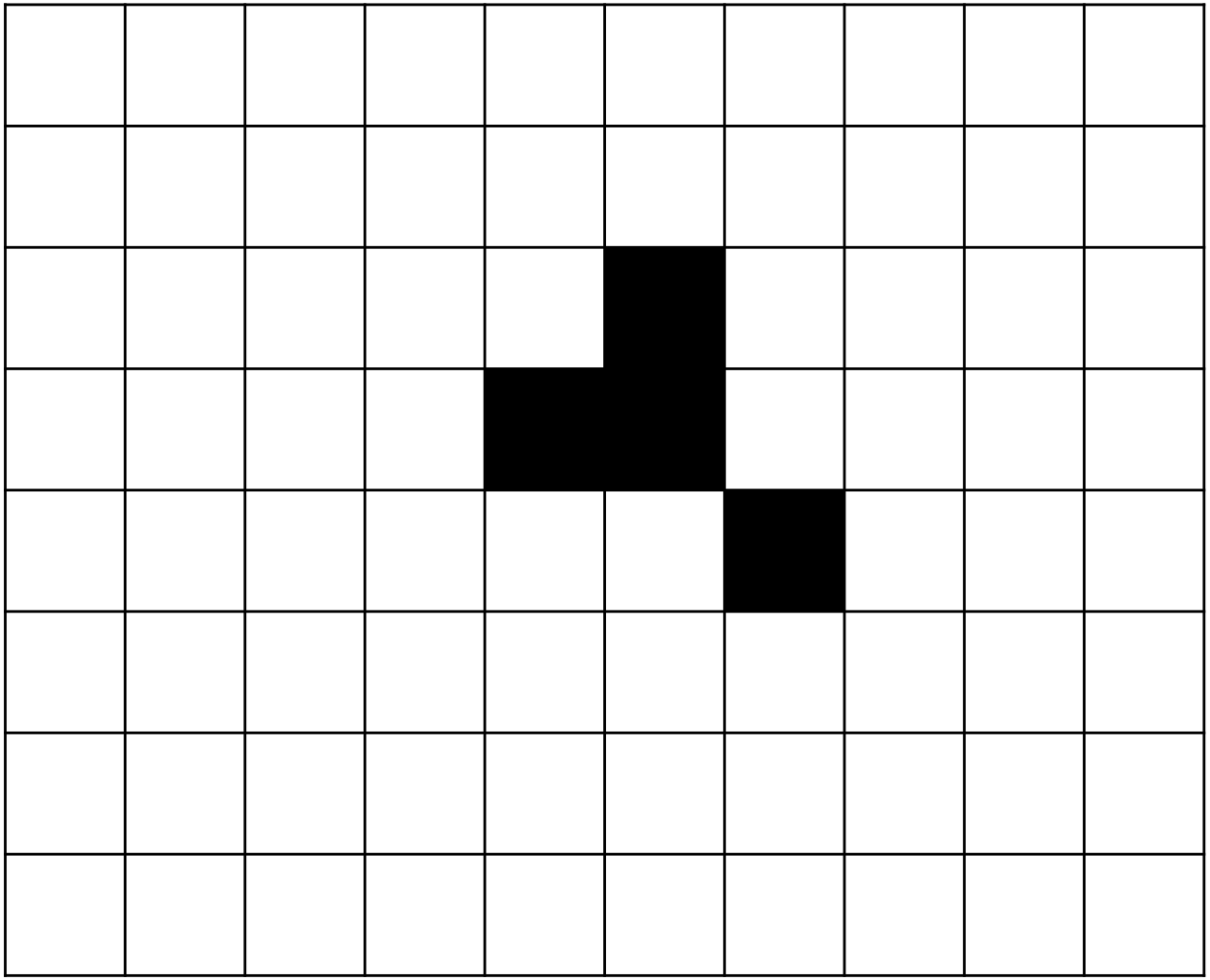
Erosion Example



Erosion Example

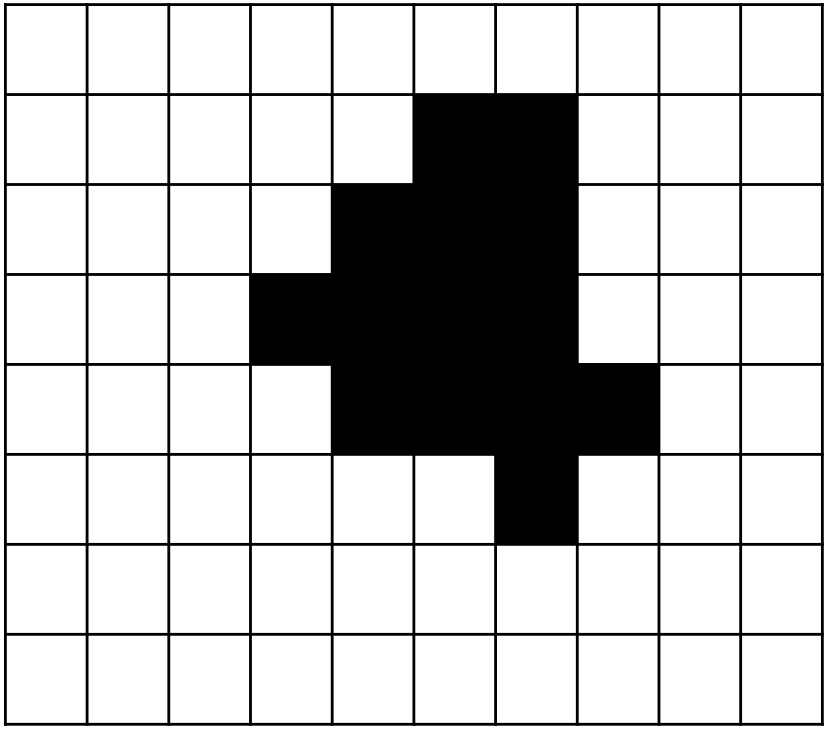


Erosion Example

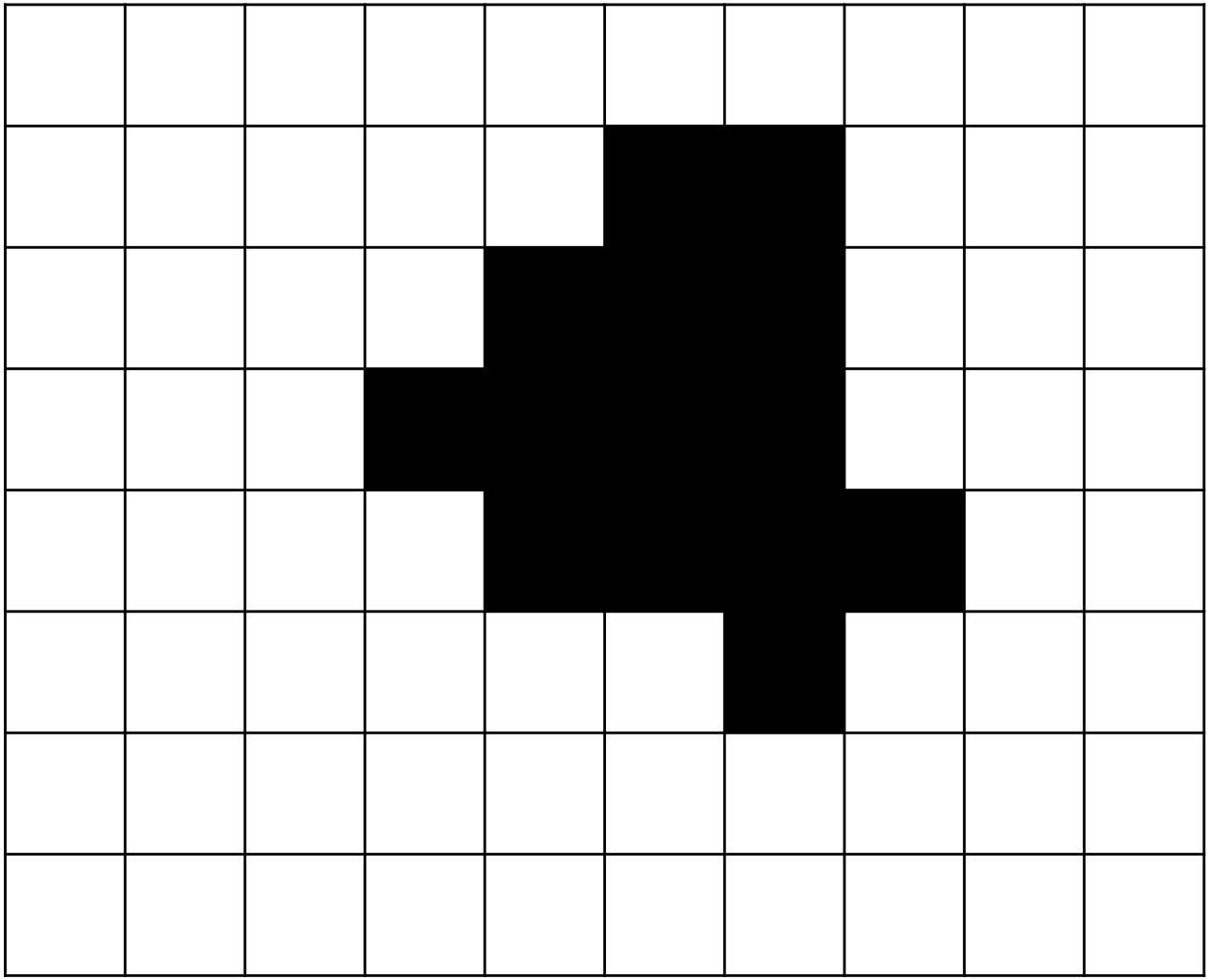


Dilation

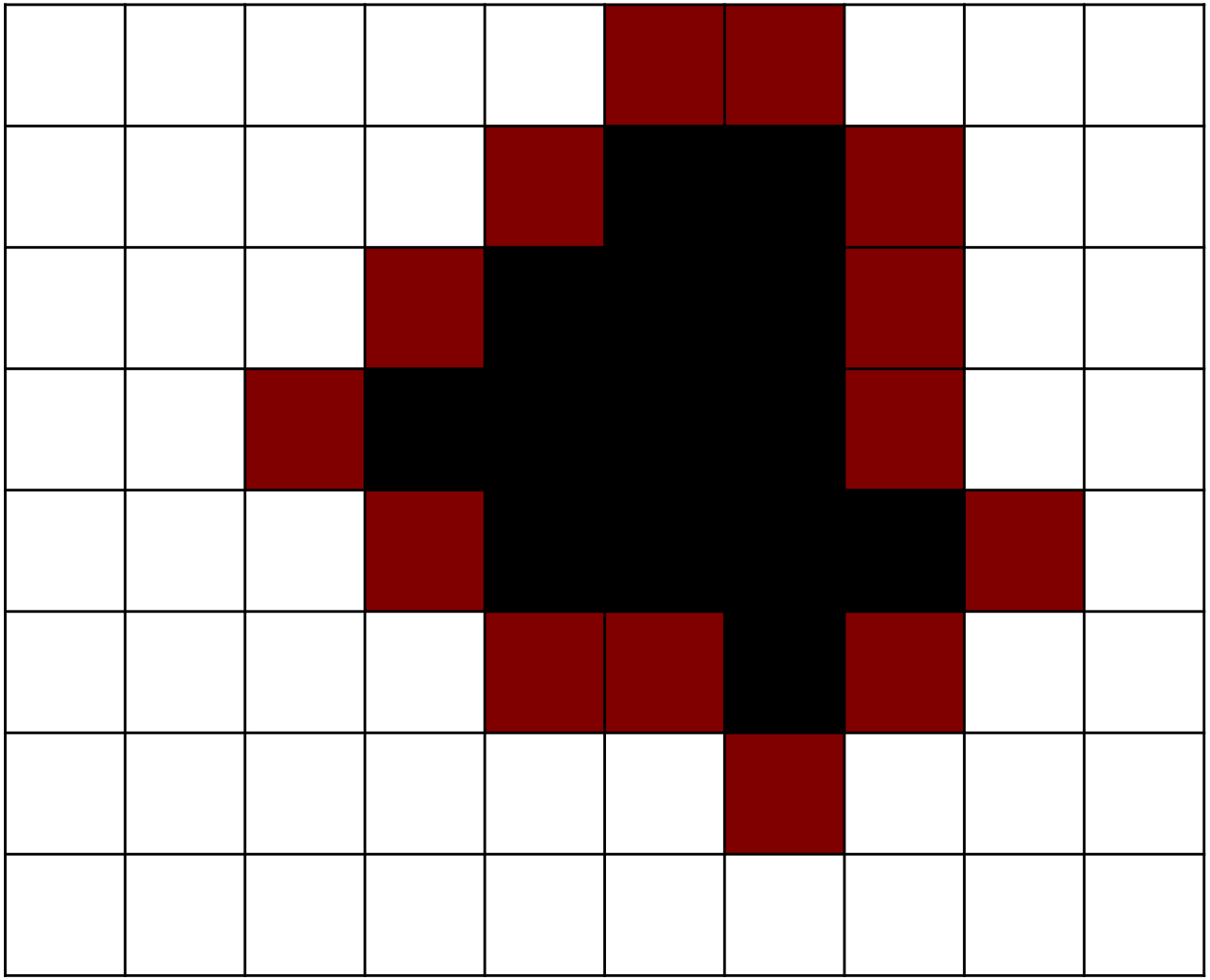
Change each background (*here* while) pixel to a foreground (*here* black) pixel if it has a foreground pixel as its N4 neighbor



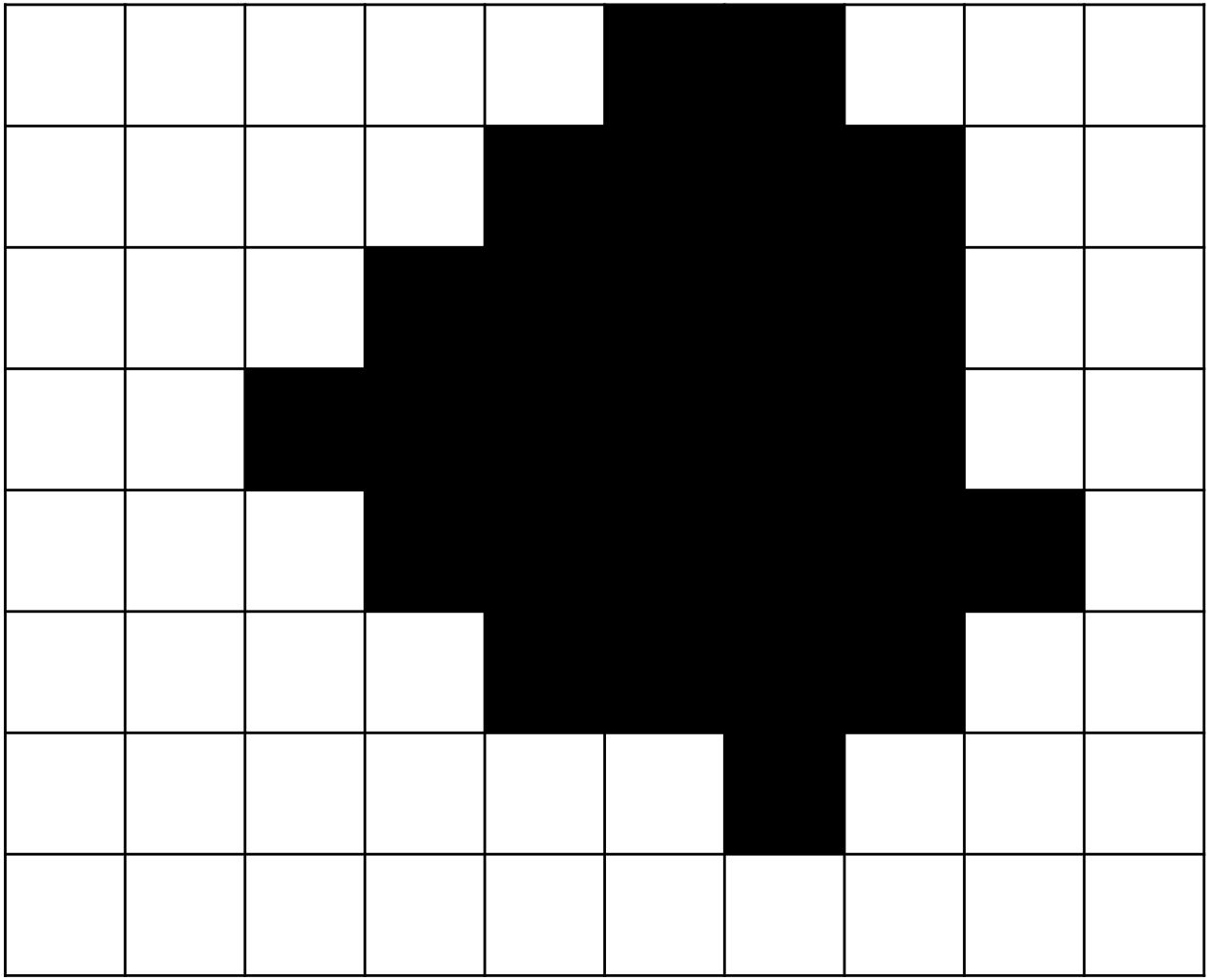
Dilation Example



Dilation Example



Dilation Example



Erosion and Dilation

- Erosion shrinks the foreground
- Erosion removes foreground outliers
- Dilation expands the foreground
- Dilation fills holes

Can we combine both to improve segmentation and masking?

Opening and Closing

Opening

- Removing stray foreground pixels in background
- Step 1: erosion; Step 2: dilation

Closing

- Remove holes in the foreground
- Step 1: dilation; Step 2: erosion

Original Image

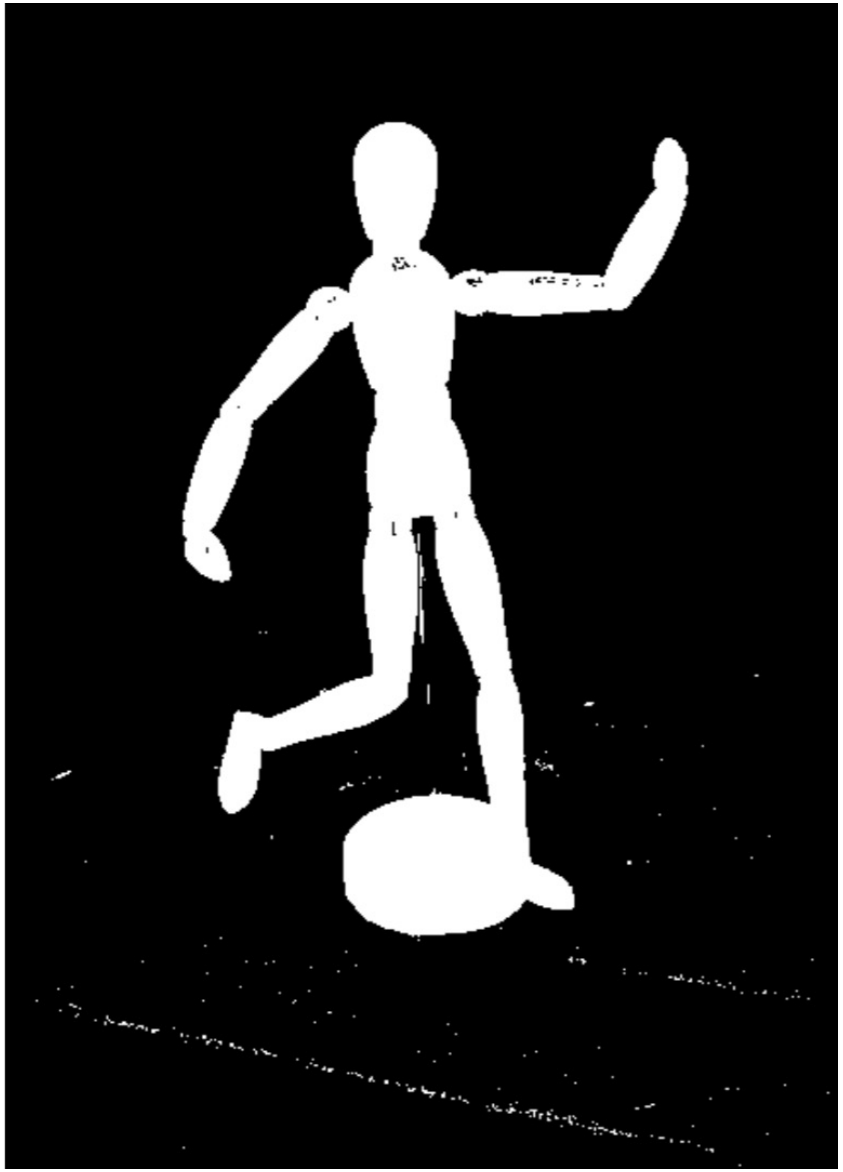


Image Courtesy: Whitaker

Opening

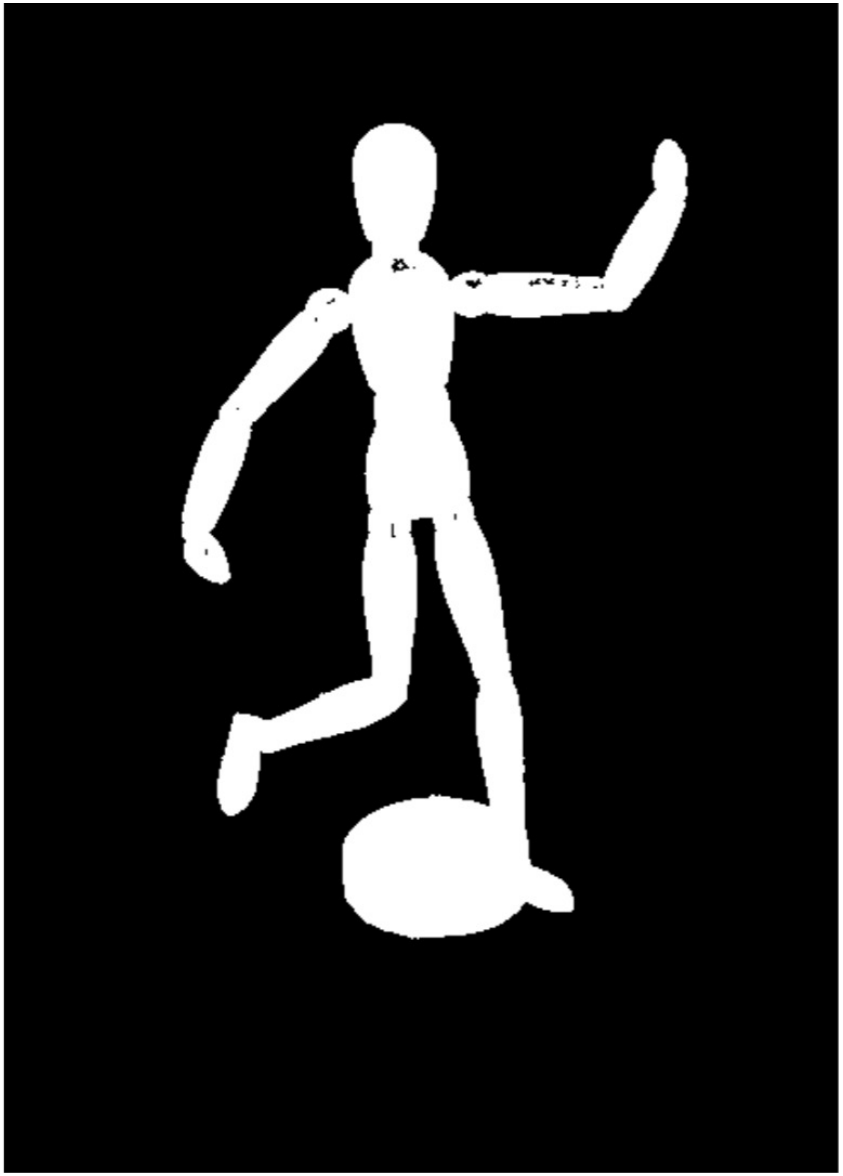


Image Courtesy: Whitaker

Closing (After Opening)

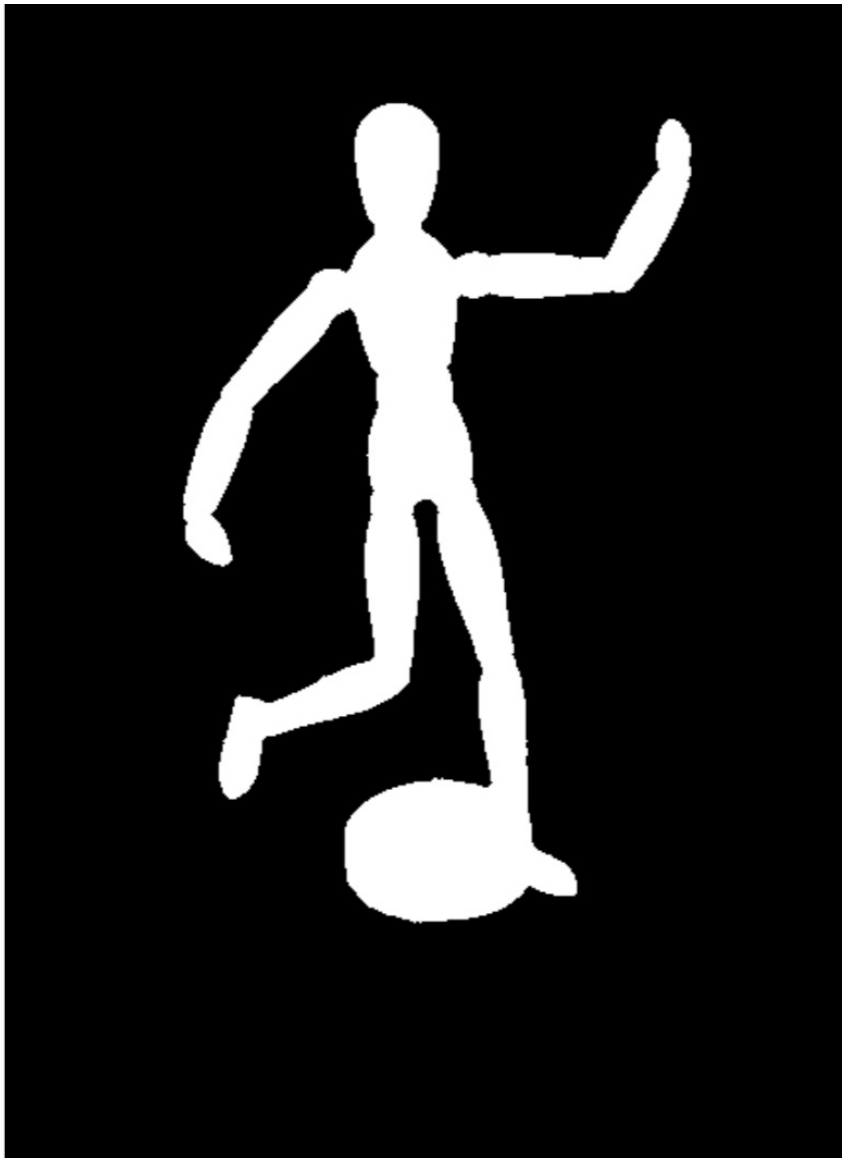


Image Courtesy: Whitaker

Opening + Closing Result

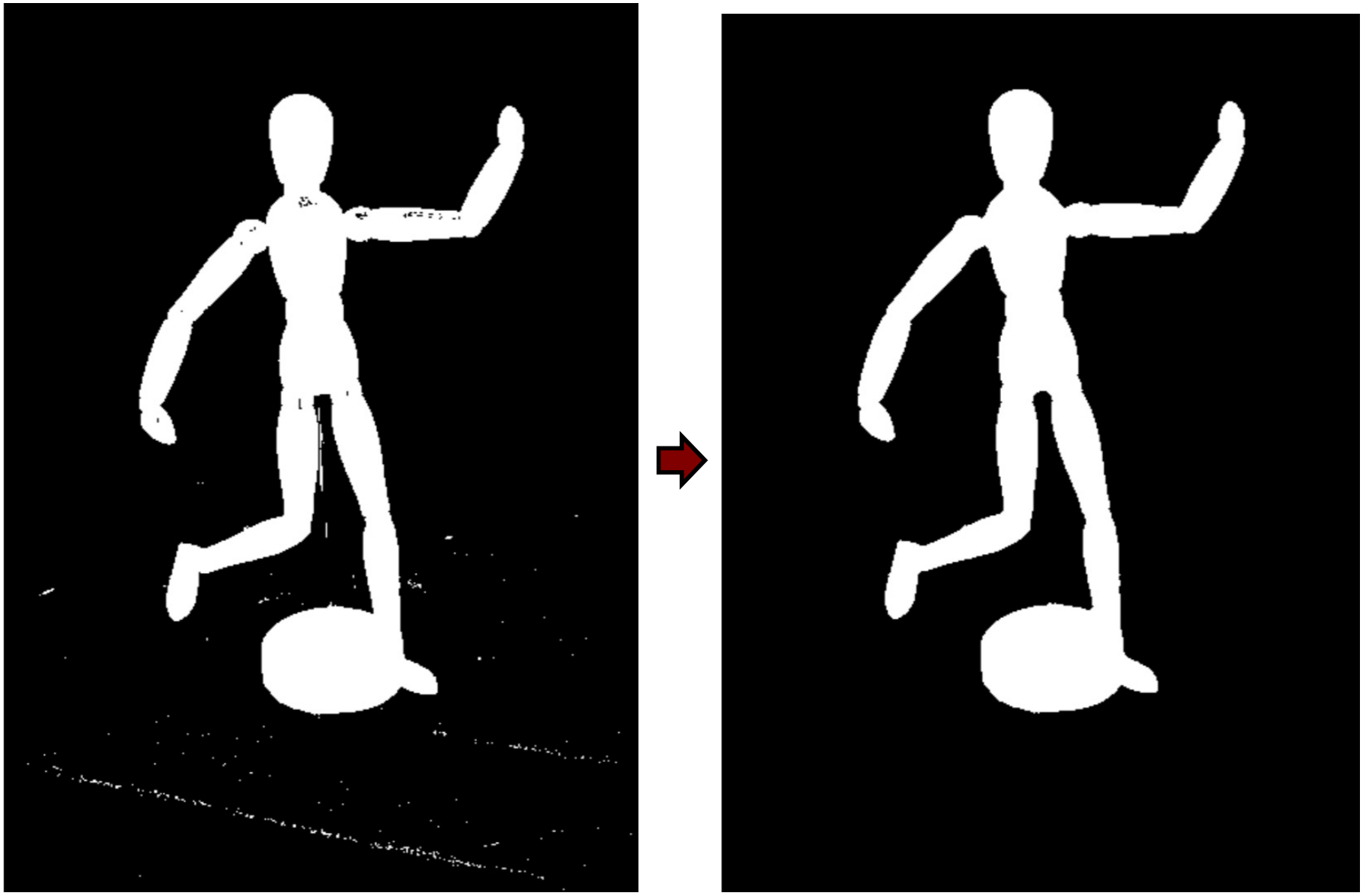


Image Courtesy: Whitaker

Morphological Operators

- Morphological Operators can be used to eliminate noisy masks
- Combination of Opening and Closing removes stray pixels and fills holes
- Erosion and dilation are local operators

Summary

- Binary images are relevant tools in several image processing applications
- Connected components
- Distance transforms
- Erosion, dilation, opening, and closing

Literature

- Szeliski, Computer Vision: Algorithms and Applications, Chapter 3.3
- Förstner, Scriptum Photogrammetrie I, Chapter 6

Slide Information

- The slides have been created by Cyrill Stachniss as part of the photogrammetry and robotics courses.
- **I tried to acknowledge all people from whom I used images or videos. In case I made a mistake or missed someone, please let me know.**
- The photogrammetry material heavily relies on the very well written lecture notes by Wolfgang Förstner and the Photogrammetric Computer Vision book by Förstner & Wrobel.
- Parts of the robotics material stems from the great Probabilistic Robotics book by Thrun, Burgard and Fox.
- If you are a university lecturer, feel free to use the course material. If you adapt the course material, please make sure that you keep the acknowledgements to others and please acknowledge me as well. To satisfy my own curiosity, please send me email notice if you use my slides.

Cyrill Stachniss, cyrill.stachniss@igg.uni-bonn.de