Digital Image Processing (CS/ECE 545)
Lecture 6: Detecting Simple Curves

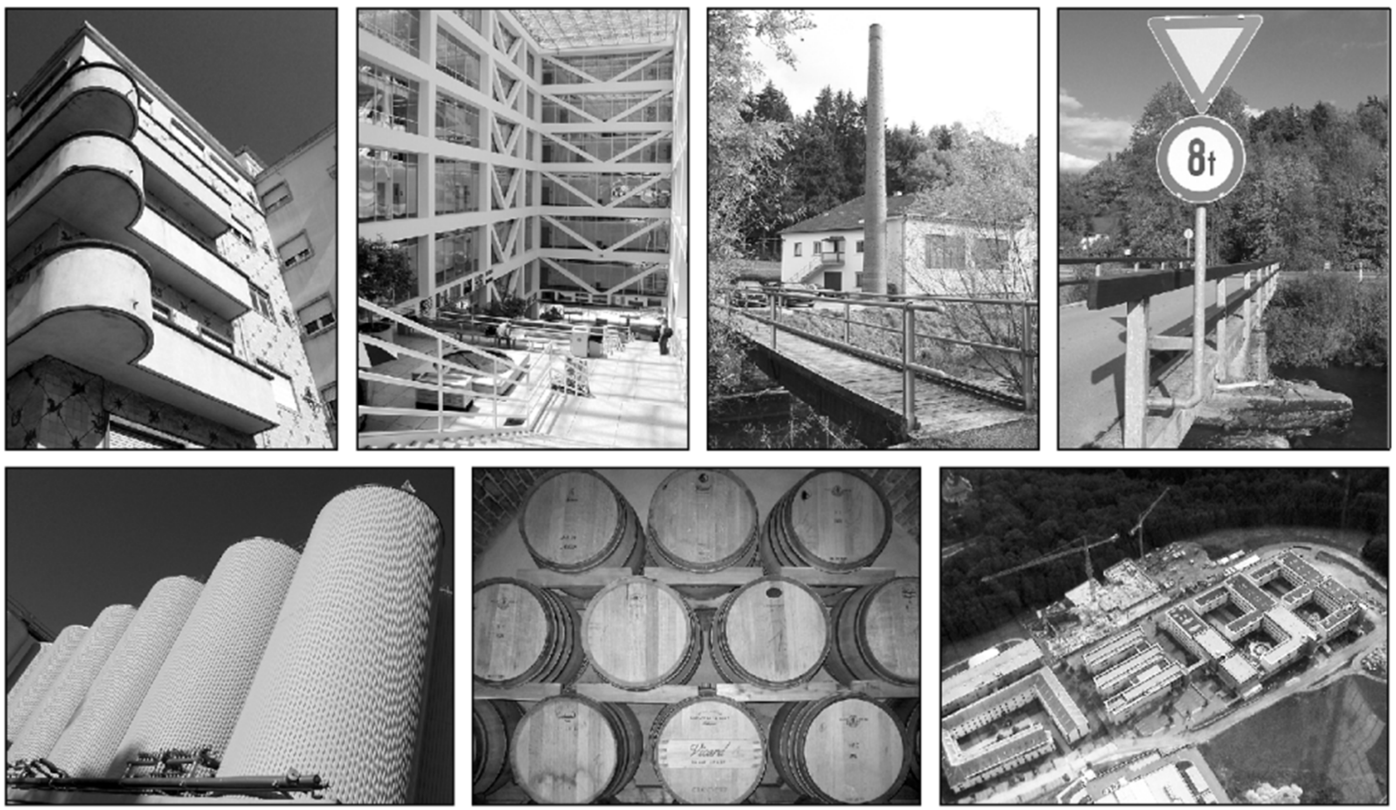
Prof Emmanuel Agu

Computer Science Dept.

Worcester Polytechnic Institute (WPI)

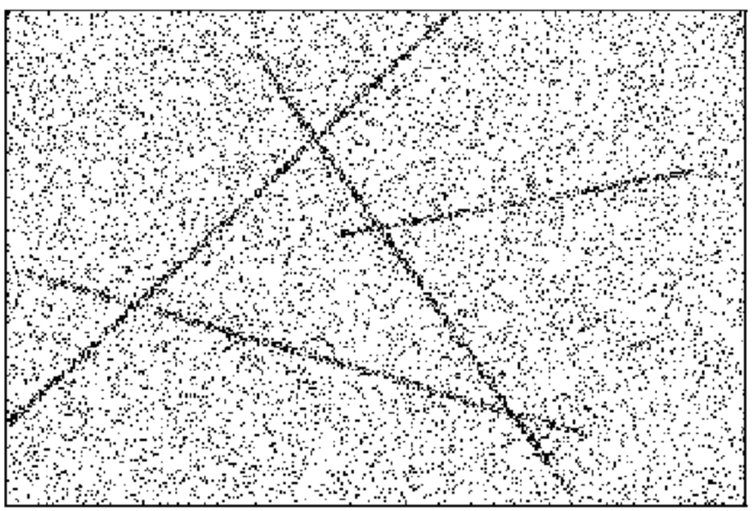
Detecting Lines and Simple Curves

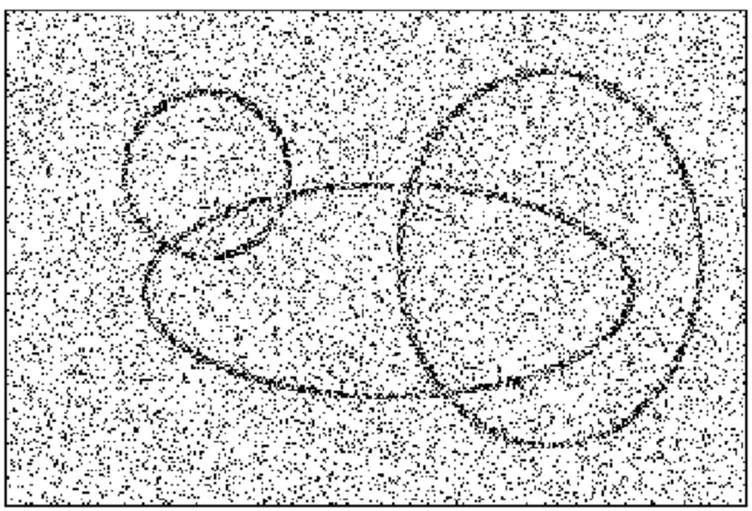
- **Many man-made objects exhibits simple geometric forms:** lines, circles, ellipses



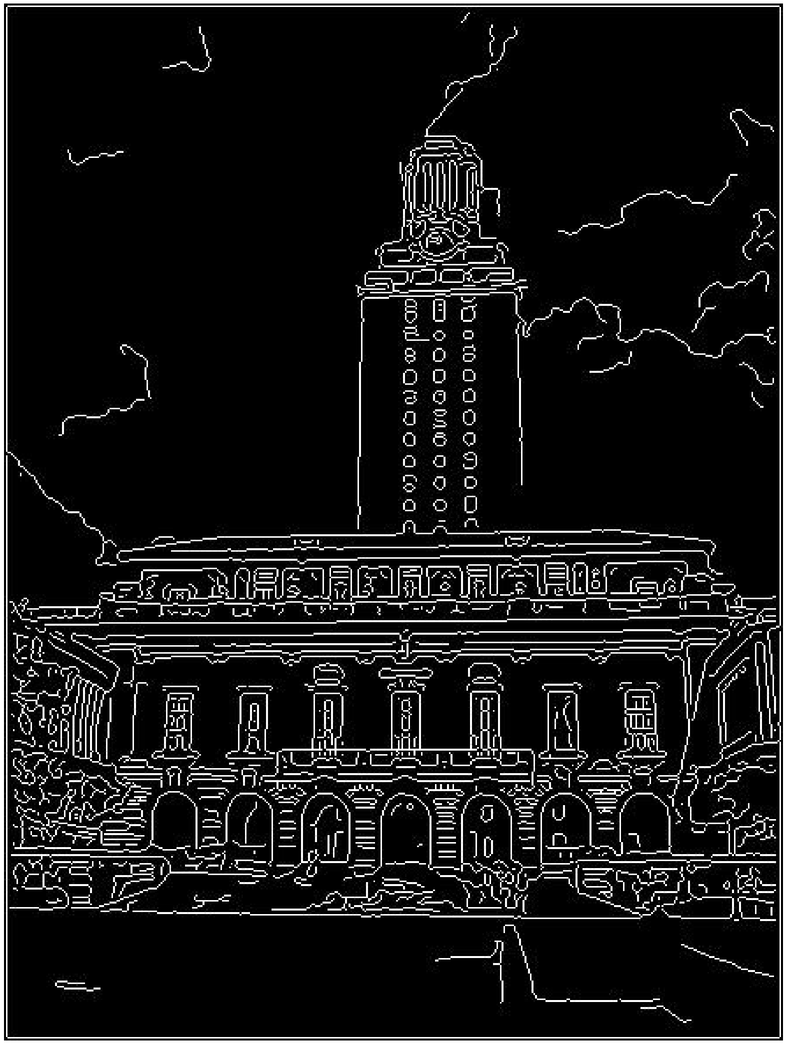
Hough Transform

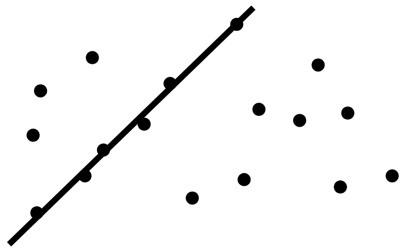
- Hough transform: Find any shape that can be defined parametrically within a distribution of points (Paul Hough)
- **Example:** lines, circles, ellipses.
- Used to find line segments in edge maps
- Why isn’t displaying results of edge detection adequate?





Difficulty of line fitting





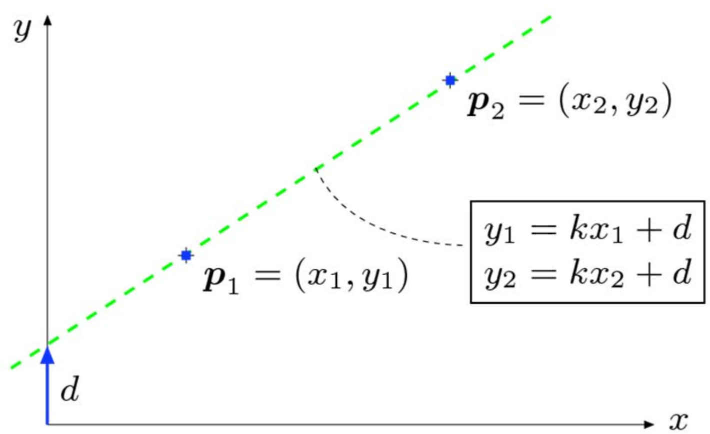
- **Extra** edge points (clutter), multiple models:
  - which points go with which line, if any?
- Only some parts of each line detected, and some parts are **missing:**
  - how to find a line that bridges missing evidence?
- **Noise** in measured edge points, orientations:
  - how to detect true underlying parameters?

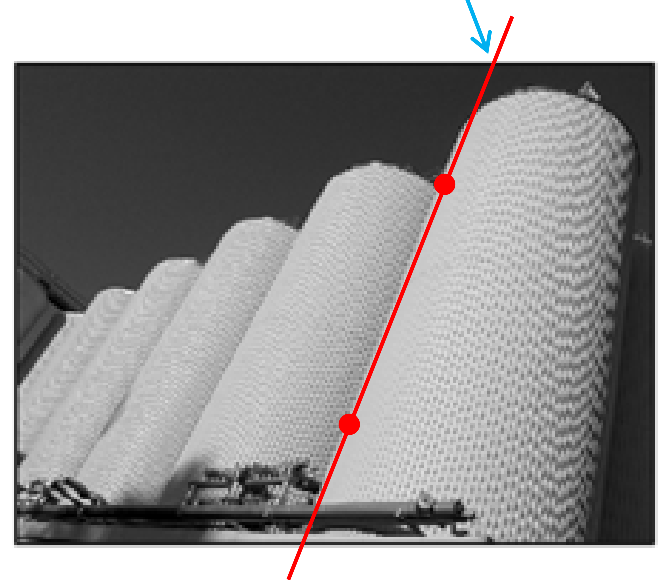
Hough Transform

- Equation of a line

$$y = \overset{\text{slope}}{k}x + \overset{\text{intercept}}{d}$$

- If points $\mathbf{p_1}$ and $\mathbf{p_2}$ on same line, same **k** and **d**





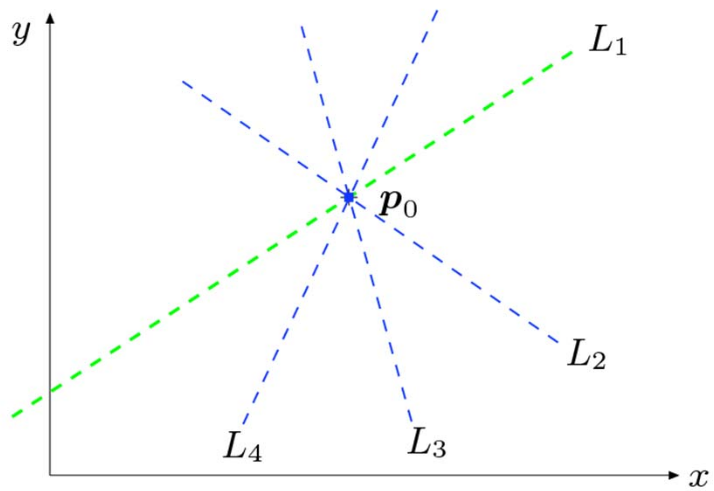
Hough Transform

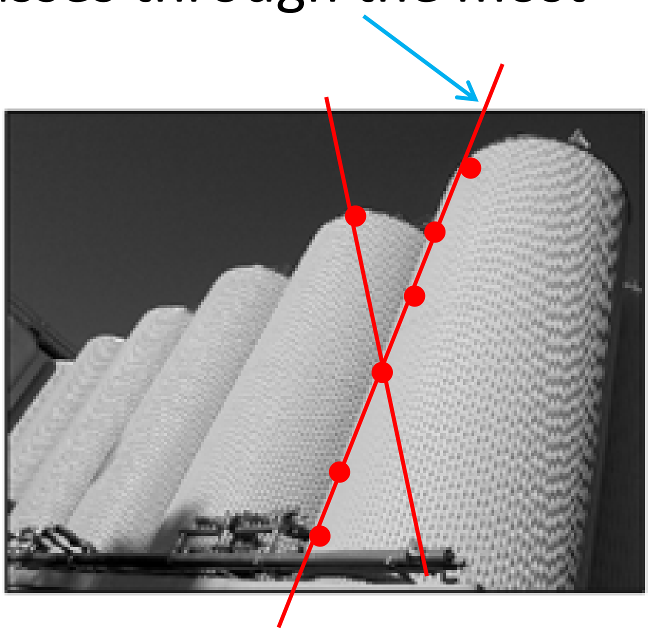
- Equation of a line

slope intercept

$$y = kx + d$$

- Hough transform:
  - start with edge point (x,y)
  - Find (slope k, intercept d) that passes through the most edge points



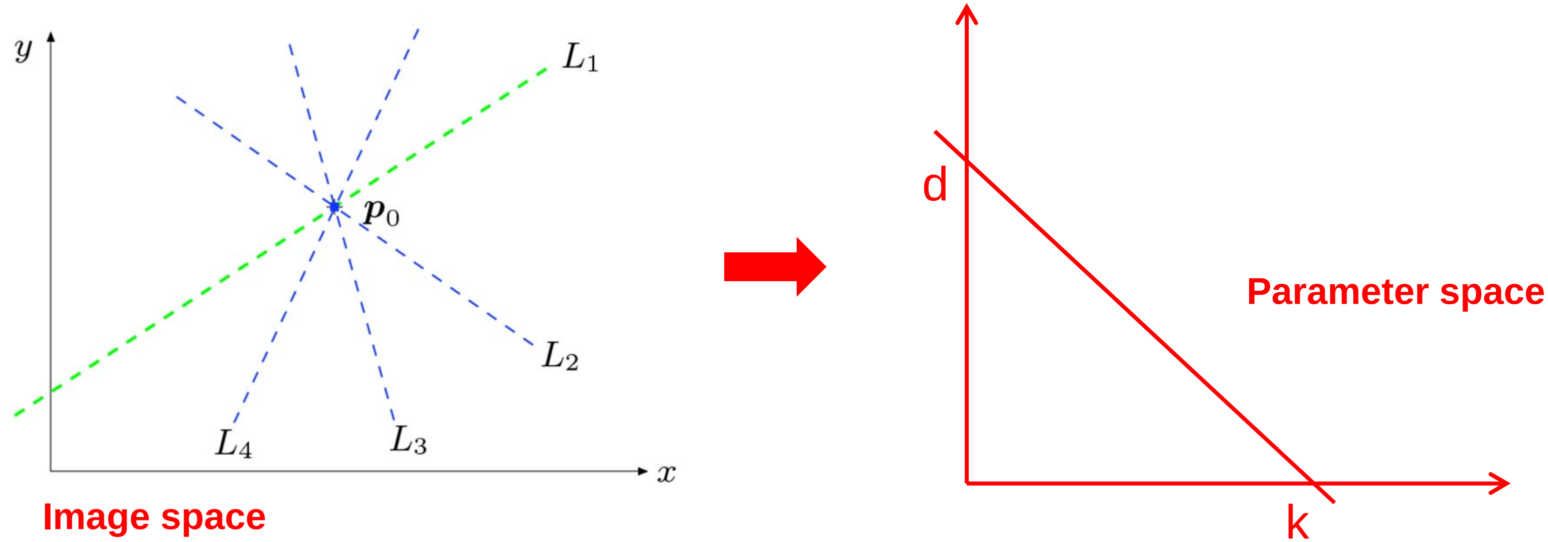


Hough Transform

- Consider the point (x,y) = (1,1)
- Many lines of different slope k and intercept d can pass through (1,1)
- Thus we can rewrite equation of line through (1,1) as

  1 = k.1 + d (image) => d = -k + 1 (parameter space)
- Set of all lines passing through (1,1) (left)

  can be shown as line in parameter space (right)



Hough Transform: Example

- Suppose image has 5 points: (1,0), (1,1), (2,1), (4,1) and (3,2)
- Each of these points corresponds to the following lines that can be plotted

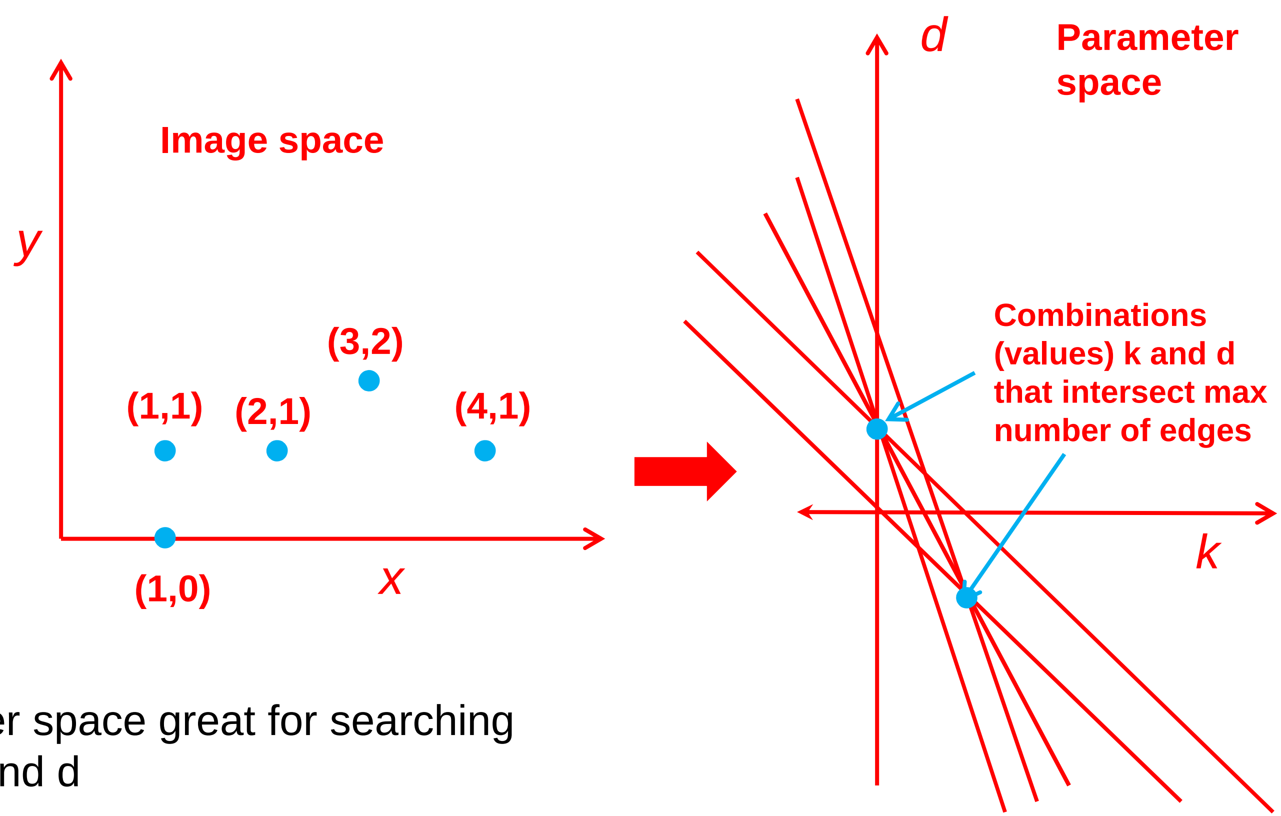
(1,0) -> d = -k

(1,1) -> d = -k + 1

(2,1) -> d = -2k + 1

(4,1) -> d = -4k +1

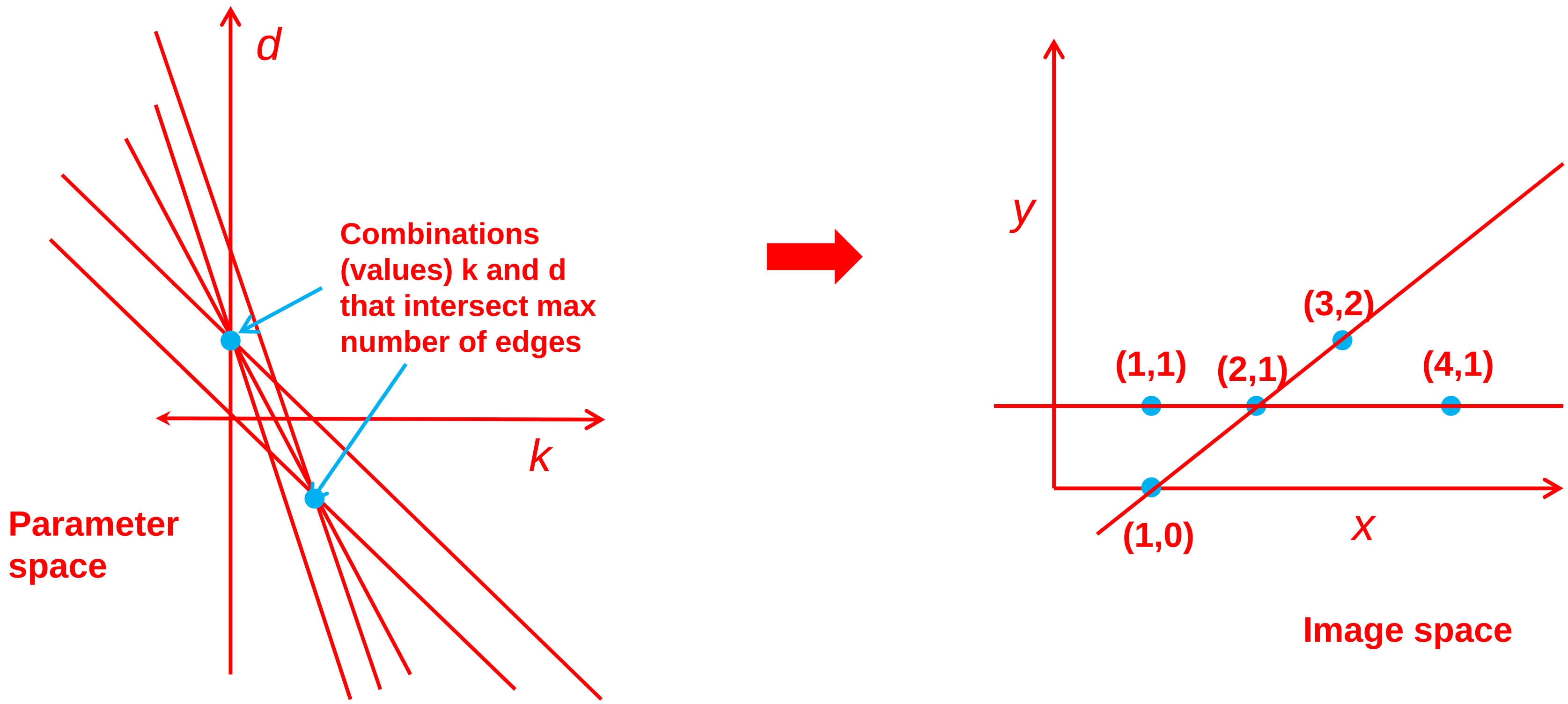
(3,2) -> d = -3k + 2



Note: Lines in parameter space great for searching for optimal values of k and d

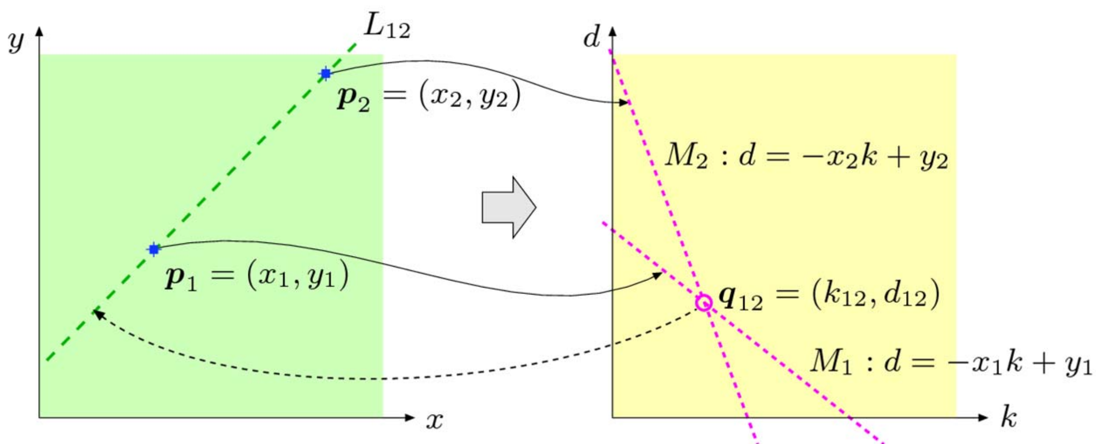
Hough Transform: Example

- After finding optimal values of k and d, we can draw them as lines in image space
- In example: (k,d) = (0,1) and (1,-1) each intersect 3 lines
- Converting to image space: (0,1) -> y = 1, (1,-1) -> y = x - 1



Hough Transform: Image vs Parameter Space

- If *N* lines intersect at position (*k'*, *d'*) in **parameter space**, then *N* image points lie on the corresponding line y = kx + d in **image space**

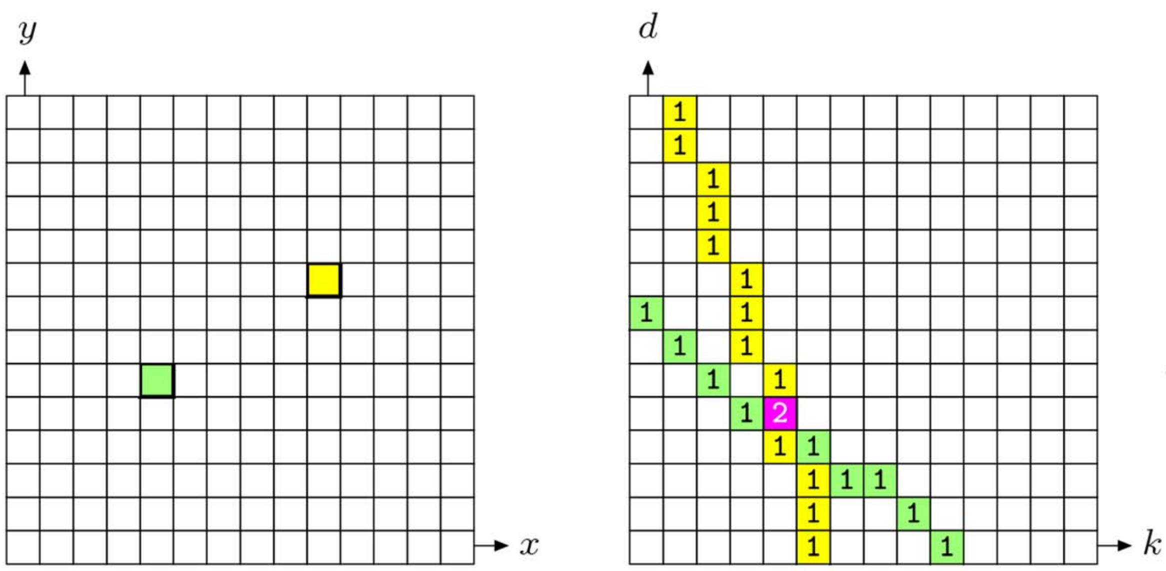


(a) x/y image space (b) k/d parameter space

Image Space (x, y)		*Parameter Space* (k, d)	
Point	$\boldsymbol{p}_i = (x_i, y_i)$	$M_i : d = -x_i k + y_i$	Line
Line	$L_j : y = k_j x + d_j$	$\boldsymbol{q}_j = (k_j, d_j)$	Point

Accumulator Array

- **Accumulator array:** discrete representation of parameter space as 2D array
- Given a point in image, increment all points on it's corresponding line (draw line) in parameter space
- A line in image will be intersection of multiple lines in parameter space.



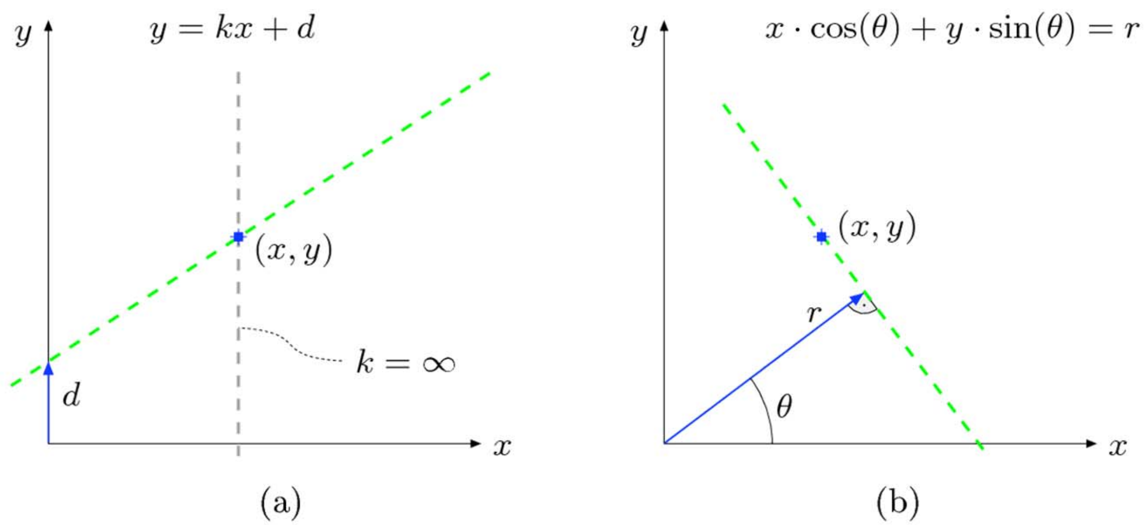
(a) Image Space

(b) Accumulator Array

$$M_i : d = -x_i k + y_i$$

Hough Transform

- **A Problem:** The form of line $y = kx + d$ cannot express vertical line (slope k = infinity)
- Alternate line representation in terms of *r, Θ*:
  - *r,* perpendicular distance of line from origin
  - *Θ,* angle of line's perpendicular to *x* axis
  - If we allow negative *r*, then -90 < *Θ* < 90

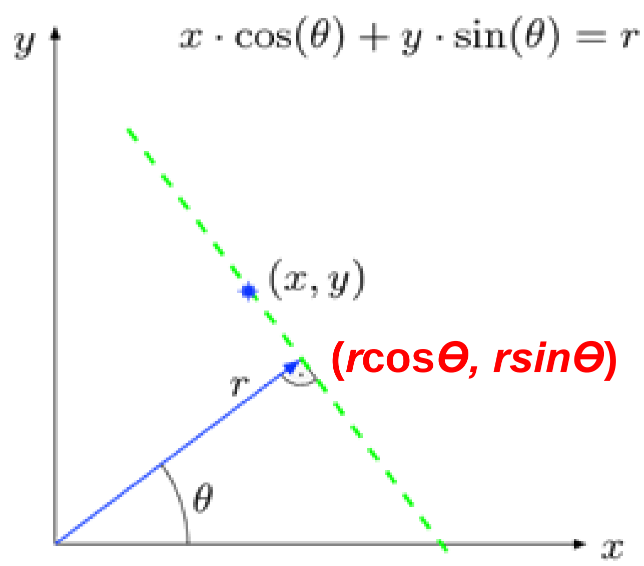


Hough Transform

- Point (p,q) where perpendicular to line meets line

  = (*rcosΘ, rsinΘ*)
- Consider (x,y), another point on line, gradient of line is

$$\frac{\text{rise}}{\text{run}} = \frac{y-q}{x-p} = \frac{y - r\sin\theta}{x - r\cos\theta}.$$



- Gradient of perpendicular is *tanΘ*
- So gradient of line is

$$-\frac{1}{\tan\theta} = -\frac{\cos\theta}{\sin\theta}.$$

- Putting these two expressions together

$$\frac{y - r\sin\theta}{x - r\cos\theta} = -\frac{\cos\theta}{\sin\theta}.$$

Hough Transform

$$\frac{y - r\sin\theta}{x - r\cos\theta} = -\frac{\cos\theta}{\sin\theta}.$$

- If we multiply these fractions, we obtain

$$y\sin\theta - r\sin^2\theta = -x\cos\theta + r\cos^2\theta$$

- And this equation can be rewritten as

$$\begin{aligned} y\sin\theta + x\cos\theta &= r\sin^2\theta + r\cos^2\theta \\ &= r(\sin^2\theta + \cos^2\theta) \\ &= r. \end{aligned}$$

- Below is required equation for the line as

$$x\cos\theta + y\sin\theta = r.$$

called Hessian Normal Form

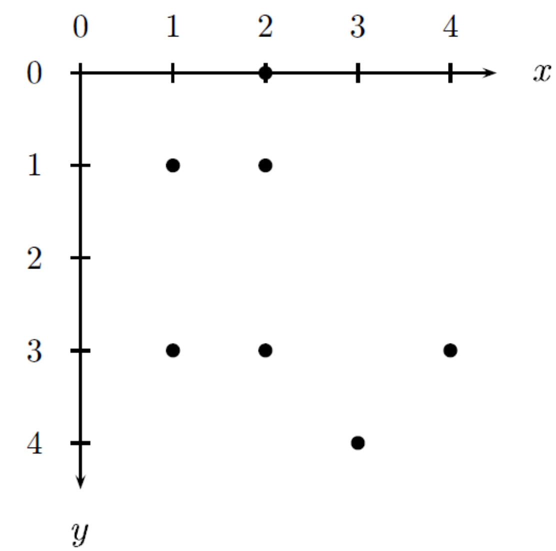
Implementing Hough Transform using Hessian Normal Form of line

- Each point (x,y) that falls on line with (*r*, *Θ*) must satisfy equation below

$$x \cos\theta + y \sin\theta = r.$$

- To implement Hough transform,
  - Choose discrete set of *Θ* values
  - For instance, for image shown consider *Θ* values

$$-45^\circ, \quad 0^\circ, \quad 45^\circ, \quad 90^\circ.$$



  - For each pixel (*x*,*y*) in image, compute *r* as

$$x \cos\theta + y \sin\theta$$

Implementing Hough Transform using Hessian Normal Form of line

- For each pixel (*x,y*) in image compute *r* as

$$x \cos\theta + y \sin\theta$$

- Put results in a table

Θ

(x,y)	$-45°$	$0°$	$45°$	$90°$
$(2,0)$	1.4	2	1.4	0
$(1,1)$	0	1	1.4	1
$(2,1)$	0.7	2	2.1	1
$(1,3)$	-1.4	1	2.8	3
$(2,3)$	-0.7	2	3.5	3
$(4,3)$	0.7	4	4.9	3
$(3,4)$	-0.7	3	4.9	4

r

- Accumulator array (below): contains number of times each value of (*r*, *Θ*) appears in table (above)

	-1.4	-0.7	0	0.7	1	1.4	2	2.1	2.8	3	3.5	4	4.9
$-45°$	1	2	1	2		1							
$0°$					2		3			1		1	
$45°$						2		1	1		1		2
$90°$			1		2					3		2	

Hough Transform

	-1.4	-0.7	0	0.7	1	1.4	2	2.1	2.8	3	3.5	4	4.9
$-45°$	1	2	1	2		1							
$0°$					2		3			1		1	
$45°$						2		1	1		1		2
$90°$			1		2					3		2	

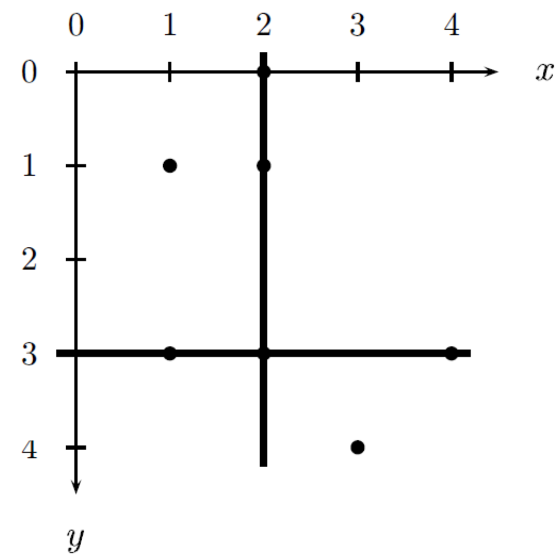
- Two largest values occur at $(r, \theta) = (2, 0°)$ and $(r, \theta) = (3, 90°)$
- The corresponding lines are:

$$x \cos 0 + y \sin 0 = 2$$

or $x = 2$, and

$$x \cos 90 + y \sin 90 = 3$$

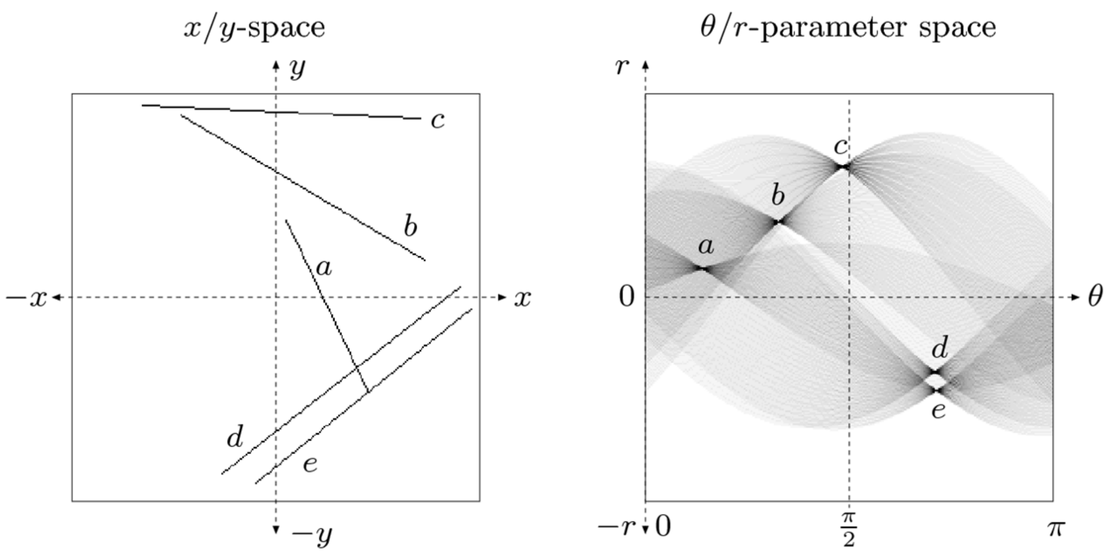
or $y = 3$.



Visualizing Image Space to Parameter Space Representation

- Image Space and parameter space using HNF representation for $0 \leq \theta < \pi$
- If image center is reference, then range of *r* limited to half of diagonal. E.g. for image of width *M* and height *N*

$$-r_{\max} \leq r_{x,y}(\theta) \leq r_{\max}, \quad \text{where} \quad r_{\max} = \frac{1}{2}\sqrt{M^2 + N^2}$$



(a) (b)

$$r_{x_i,y_i}(\theta) = x_i \cdot \cos(\theta) + y_i \cdot \sin(\theta)$$

Hough Transform Algorithm

- 2 main implementation stages

HOUGHLINES(I)

Returns the list of parameters $\langle \theta_i, r_i \rangle$ corresponding to the strongest lines found in the binary image I.

Set up a two-dimensional array $Acc[\theta, r]$ of counters, initialize to 0.
Let (u_c, v_c) be the center coordinates of the image I
for all image coordinates (u, v) **do**
  if $I(u, v)$ is an edge point **then**
    Get coordinate relative to the image center (u_c, v_c):
    $(x, y) \leftarrow (u - u_c, v - v_c)$
    for $\theta_i = 0 \ldots \pi$ **do**
      $r_i = x \cdot \cos(\theta_i) + y \cdot \sin(\theta_i)$
      Increment $Acc[\theta_i, r_i]$
Return the list of parameter pairs $\langle \theta_j, r_j \rangle$ for K strongest lines:
$MaxLines \leftarrow$ FINDMAXLINES(Acc, K)
return $MaxLines$.

1. Create 2-dimensional accumulator array
2. Search accumulator array for maximum values Return parameters for k strongest lines

Filling Accumulator Array

Creating new class example:

```
LinearHT ht = new LinearHT(ip, 256, 256)
```

```
class LinearHT {
  ImageProcessor ip; // reference to the original image I
  int xCtr, yCtr;    // x/y-coordinates of image center (u_c, v_c)
  int nAng;          // N_θ steps for the angle (θ = 0...π)
  int nRad;          // N_r steps for the radius (r = −r_max ... r_max)
  int cRad;          // center of radius axis (r = 0)
  double dAng;       // increment of angle Δ_θ
  double dRad;       // increment of radius Δ_r
  int[][] houghArray; // Hough accumulator Acc[θ_i, r_i]

  //constructor method:
  LinearHT(ImageProcessor ip, int nAng, int nRad) {
    this.ip = ip;
    this.xCtr = ip.getWidth()/2;
    this.yCtr = ip.getHeight()/2;
    this.nAng = nAng;
    this.dAng = Math.PI / nAng;
    this.nRad = nRad;
    this.cRad = nRad / 2;
    double rMax = Math.sqrt(xCtr * xCtr + yCtr * yCtr);
    this.dRad = (2.0 * rMax) / nRad;
    this.houghArray = new int[nAng][nRad];
    fillHoughAccumulator();
  }
```

number of θ increments

number of r increments

$$\Delta_\theta = \frac{\pi}{N_\theta} \quad \text{and} \quad \Delta_r = \frac{2 \cdot r_{\max}}{N_r}$$

Declare array to store results

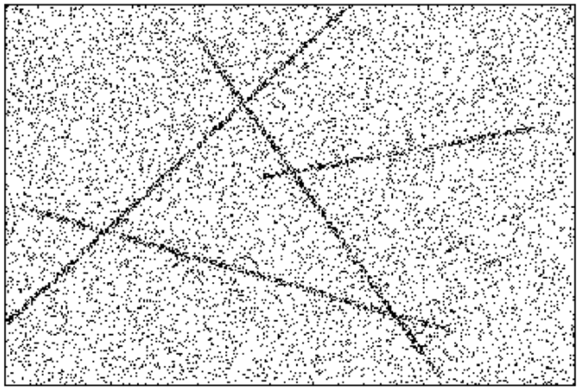
Do Hough transform

Filling Accumulator Array

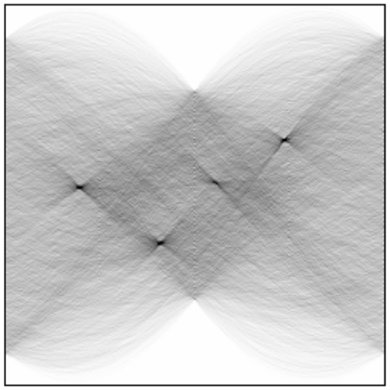
```
void fillHoughAccumulator() {
  int h = ip.getHeight();
  int w = ip.getWidth();
  for (int v = 0; v < h; v++) {
    for (int u = 0; u < w; u++) {
      if (ip.get(u, v) > 0) {
        doPixel(u, v);
      }
    }
  }
}

void doPixel(int u, int v) {
  int x = u - xCtr, y = v - yCtr;
  for (int i = 0; i < nAng; i++) {
    double theta = dAng * i;
    int r = cRad + (int) Math.rint
       ((x*Math.cos(theta) + y*Math.sin(theta)) / dRad);
    if (r >= 0 && r < nRad) {
      houghArray[i][r]++;
    }
  }
}

} // end of class LinearHT
```

Hough Transform: Noise

- 4 lines (left) correspond to 4 darkest points (right)



(a)



(b)

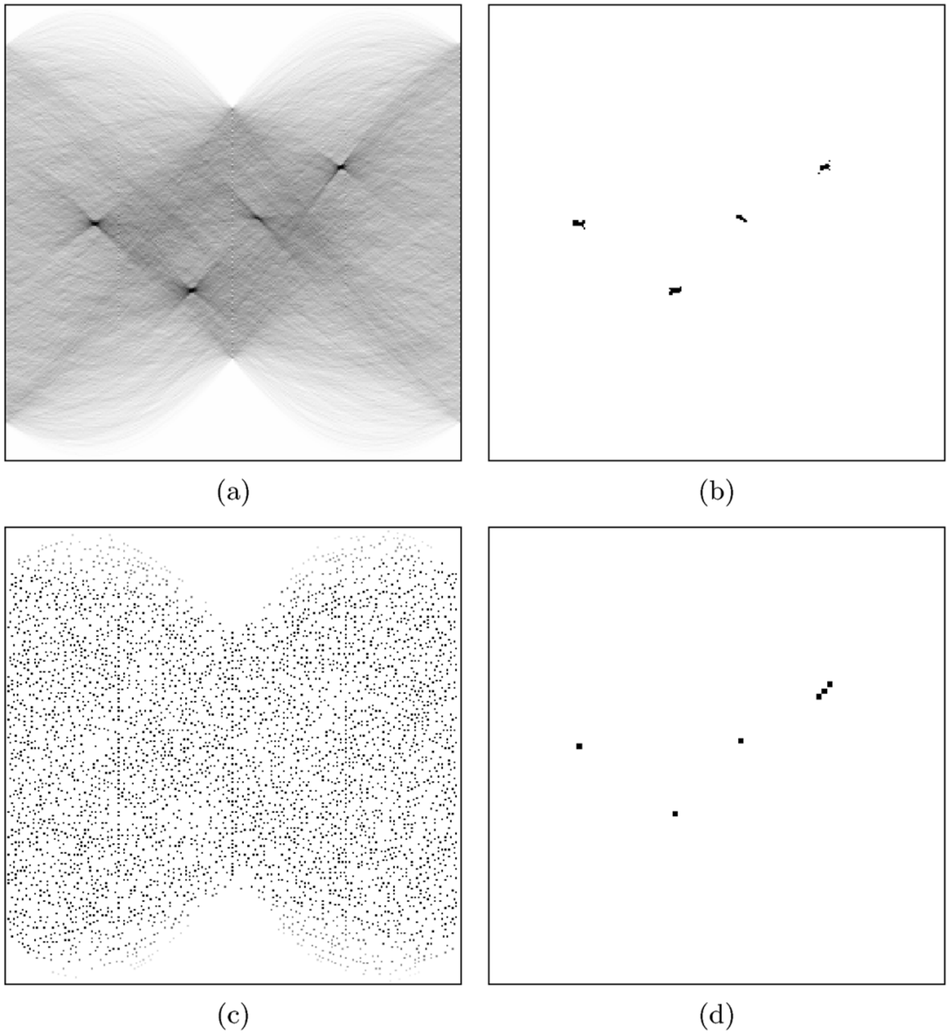
- Results are noisy
  - Max (r, Θ) values do not occur at exactly one point, but in a small area
  - Caused by rounding. Difficult to pinpoint one exact maxima
- Two strategies for dealing with noise:
  - Thresholding
  - Non-maximum suppression

Dealing with Noise: Thresholding

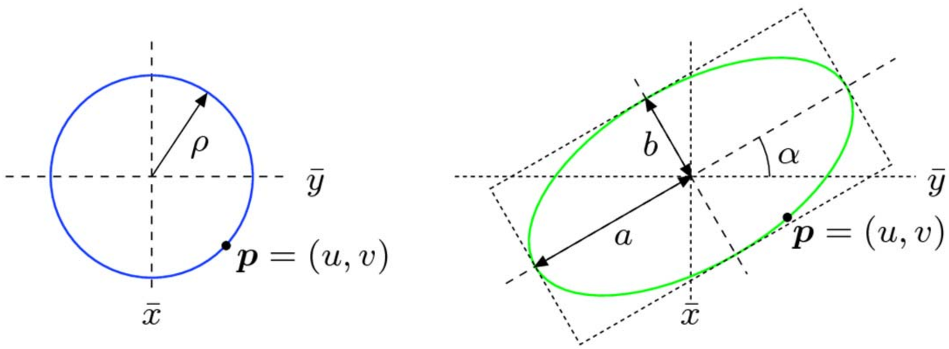
(b) **Thresholding (image b):** threshold using 50% of the maximum value

(c) **Non-maxima suppression (image d):**

- Compare every cell in *Acc*(r, Θ) to its neighbors
  - If greater than all neighbors, keep cell
  - Otherwise set cell to 0
  - Can also apply thresholding afterwards



Hough Transform on Circles

- 

- Equation of circle $Circle = \langle \bar{x}, \bar{y}, \rho \rangle$

where $\bar{x}$, $\bar{y}$ are coordinates of center, ρ is radius of circle

- A point ***p*** = (*u*, v) lies on circle when following relation holds

$$(u - \bar{x})^2 + (v - \bar{y})^2 = \rho^2$$

- Thus, parameter space for circle is 3-dimensional *Acc*[$\bar{x}$, $\overline{y,}$ ρ]
- Hough transform task, find every ($\bar{x}$, $\overline{y,}$ ρ) combination that

Hough Transform on Circles

- Hough transform task, find every $(\bar{x}\ , \overline{y,}\ \rho)$ combination that satisfies equation of circle

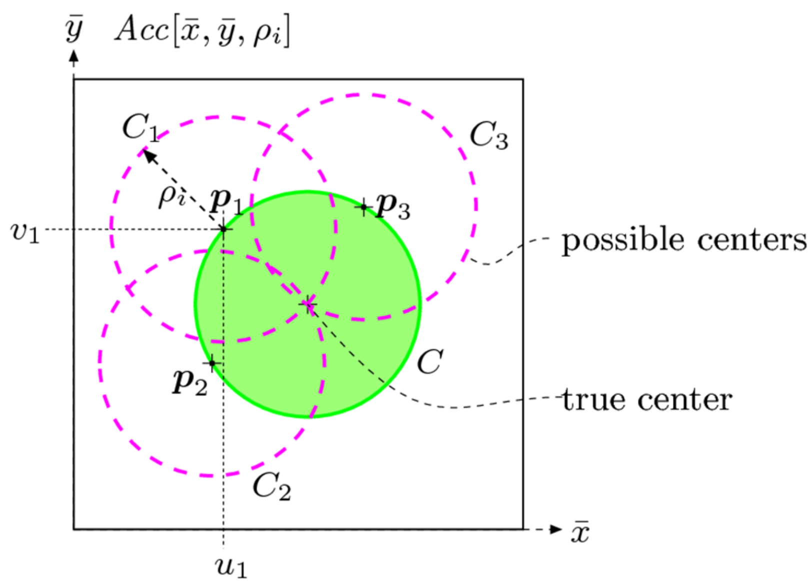
$$(u - \bar{x})^2 + (v - \bar{y})^2 = \rho^2$$

- How? Brute force algorithm?
  - For every point ***p*** = (*u*, *v*), exhaustively test if circle equation holds for every cell in parameter space

```
HoughCircles(I)
    Returns the list of parameters ⟨x̄_i, ȳ_i, ρ_i⟩ corresponding to the
    strongest circles found in the binary image I.
    Set up a three-dimensional array Acc[x̄, ȳ, ρ] and initialize to 0
    for all image coordinates (u, v) do
        if I(u, v) is an edge point then
            for all (x̄_i, ȳ_i, ρ_i) in the accumulator space do
                if (u−x̄_i)^2 + (v−ȳ_i)^2 = ρ_i^2 then
                    Increment Acc[x̄_i, ȳ_i, ρ_i]
    MaxCircles ← FindMaxCircles(Acc)   ▷ a list of tuples ⟨x̄_j, ȳ_j, ρ_j⟩
    return MaxCircles.
```

Improvement of Brute Force Approach

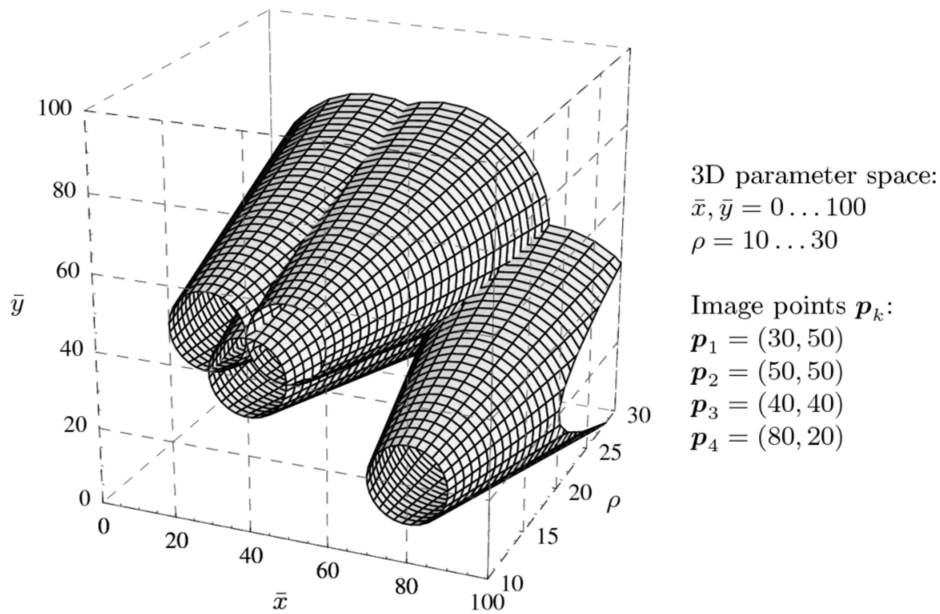
- **Observation:** If we know the radius, the locations of all possible centers lie on a circle



- **Improvement:** For each image point $\boldsymbol{p} = (u, v)$, increment (search) only cell values along edge of appropriate circle on each ρ plane of accumulator array

Hough Transform of Circles

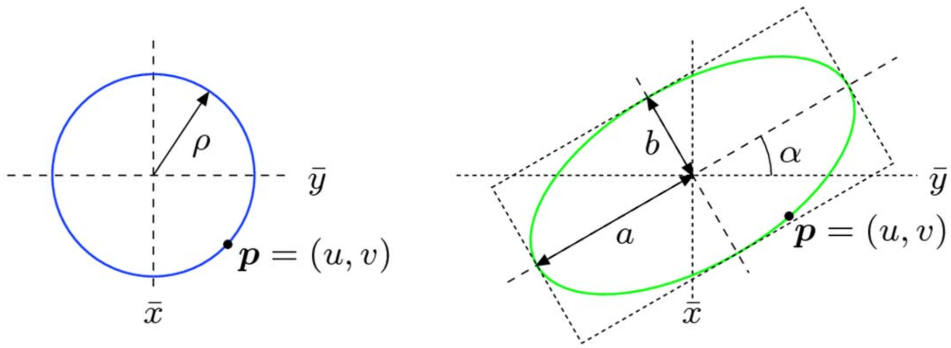
- Spatial structure of 3-dimensional space for circles



- For given image point $\boldsymbol{p_k} = (u_k, v_k)$
  - At each plane along ρ axis, traverse circle centered at (u_k, v_k)

Hough Transform of Ellipses

- Perfect circles seldom appear in images. Why?
- Due to perspective, circles appear as ellipses in images



- Ellipse requires 5 parameters $Ellipse = \langle \bar{x}, \bar{y}, r_a, r_b, \alpha \rangle$

where $(\bar{x}, \bar{y})$ are coordinates of center, (r_a, r_b) are 2 radii and α is orientation of principal axis

Hough Transform on Ellipses

- 5-dimensional parameter space can yield very large data structure
- If $128 = 2^7$ steps used in each dimension
  - requires 2^{35} accumulator cells
  - If 4-byte **int** values, requires 2^{37} bytes (128 gigabytes) memory

- **Generalized Hough Transform** better, requires 4-dimensional space

References

- Wilhelm Burger and Mark J. Burge, Digital Image Processing, Springer, 2008
- University of Utah, CS 4640: Image Processing Basics, Spring 2012
- Rutgers University, CS 334, Introduction to Imaging and Multimedia, Fall 2012
- Gonzales and Woods, Digital Image Processing (3rd edition), Prentice Hall

Digital Image Processing (CS/ECE 545)
Lecture 6: Morphological Filters

Prof Emmanuel Agu

Computer Science Dept.

Worcester Polytechnic Institute (WPI)

Mathematical Morphology

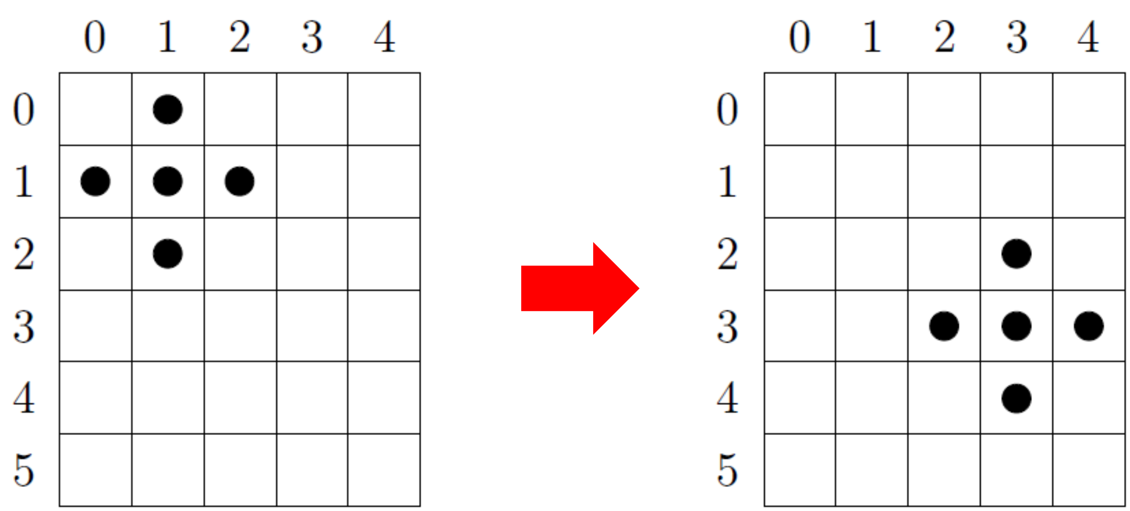
- Originally operated on Binary (black and white) images
- Binary images?
  - Faxes, digitally printed images
  - Obtained from thresholding grayscale images
- Morphological filters alter local structures in an image

Translation

- *A* is set of pixels in binary image
- *w* = (*x*,*y*) is a particular coordinate point
- *A* is set *A* “translated” in direction (*x*,*y*). i.e

$$A_x = \{(a,b) + (x,y) : (a,b) \in A\}.$$

- Example: If *A* is the cross-shaped set, and *w* = (2,2)



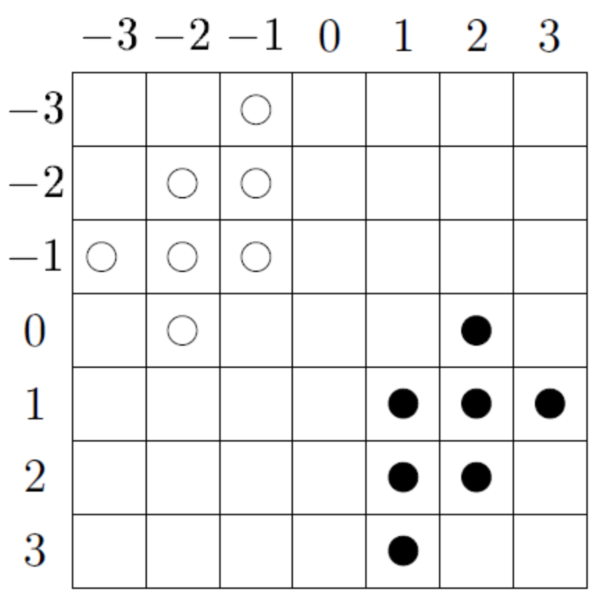
A A_w

Reflection

- *A* is set of pixels
- Reflection of *A* is given by

$$\hat{A} = \{(-x, -y) : (x, y) \in A\}.$$

- An example of a reflection



Mathematical Morphology

- 2 basic mathematical morphology operations, (built from translations and reflections)
  - **Dilation**
  - **Erosion**

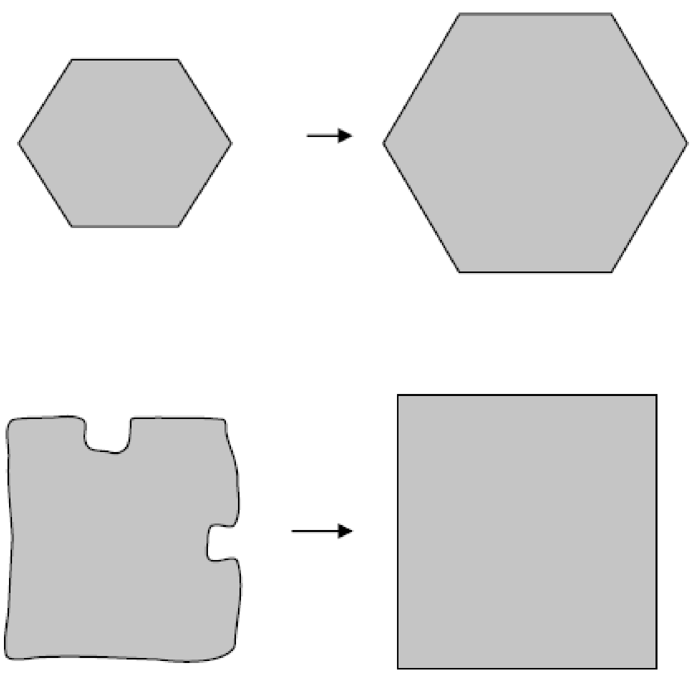
Also several composite relations

- **Closing and Opening**
- **Conditional Dilation**
- **. . .**

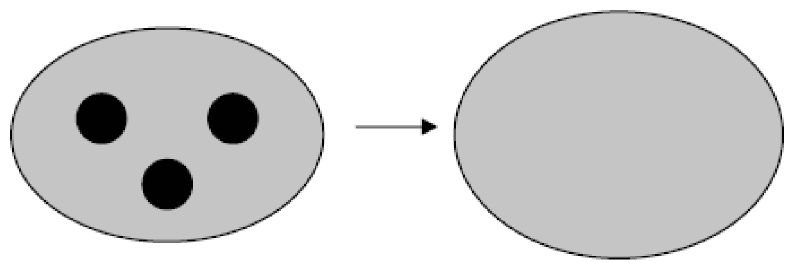
Dilation

- Dilation **expands** connected sets of 1s of a binary image. It can be used for

1. Growing features



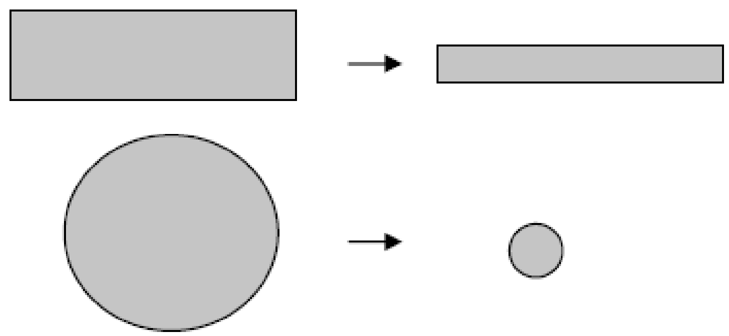
2. Filling holes and gaps



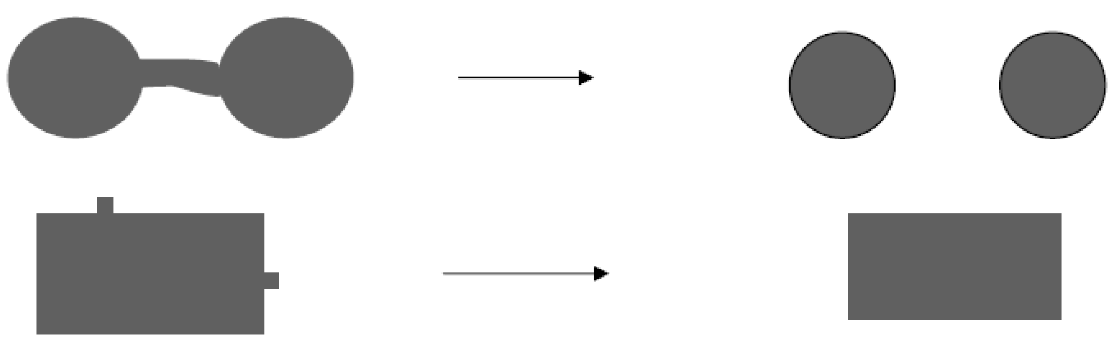
Erosion

- Erosion **shrinks** connected sets of 1s in binary image.
- Can be used for

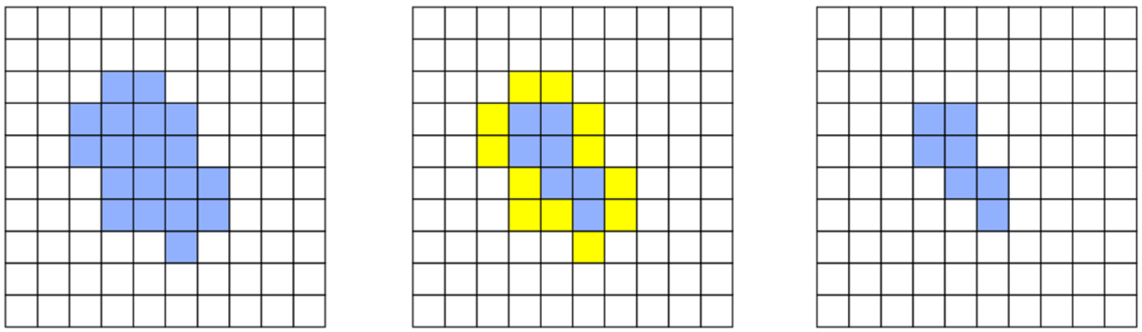
1. shrinking features



2. Removing bridges, branches and small protrusions

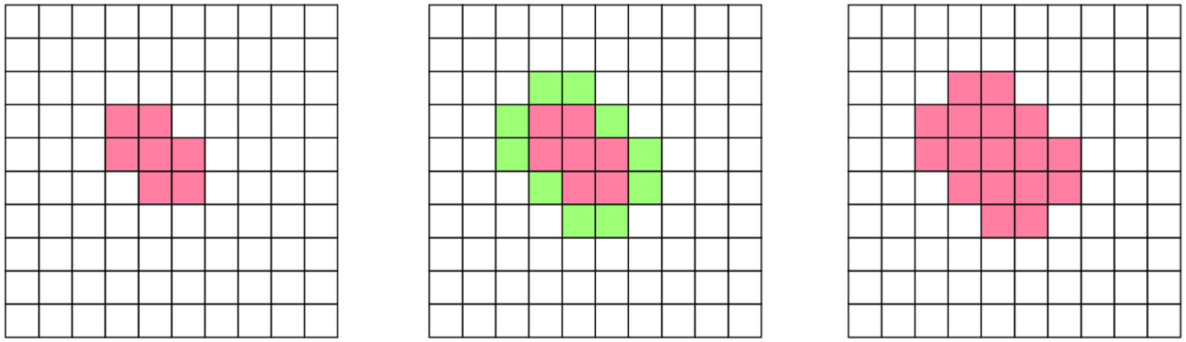


Shrink and Let Grow



(a) (b) (c)

Shrinking: remove border pixels

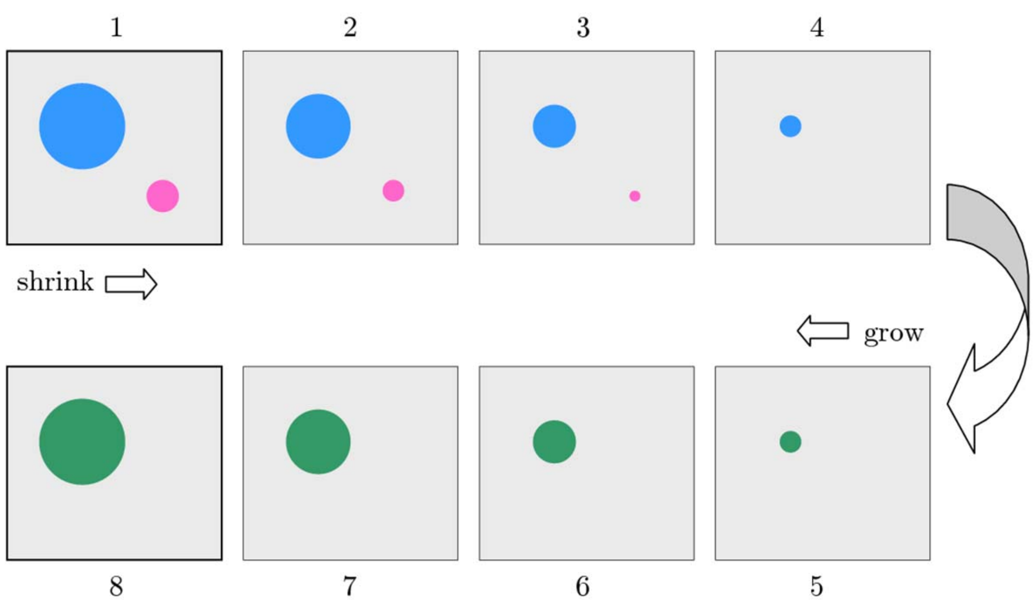


(a) (b) (c)

Growing: add layer of pixels at border

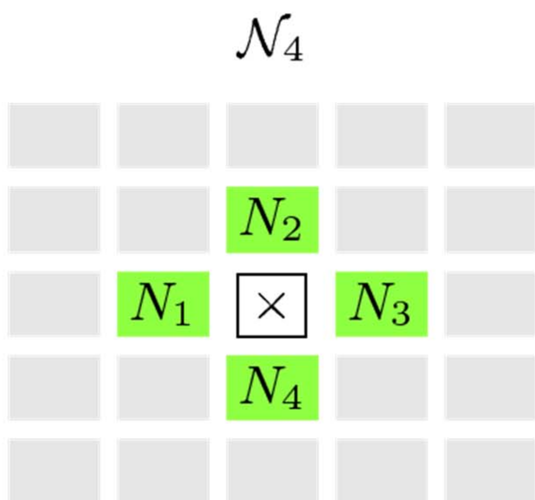
Shrinking and Let Grow

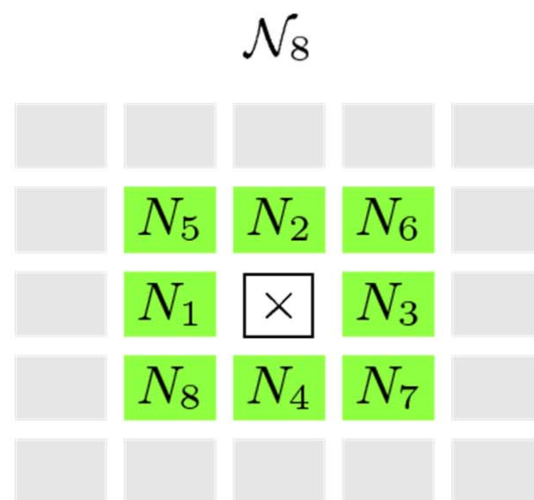
- Image structures are iteratively shrunk by peeling off a layer of thickness (layer of pixel) at boundaries
- Shrinking removes smaller structures, leaving only large structures
- Remaining structures are then grown back by same amount
- Eventually, large structures back to original size while smaller regions have disappeared



Basic Morphological Operations

- Definitions:
  - **4-Neighborhood (N_4):** 4 pixels adjacent to given pixel in horizontal and vertical directions
  - **8-Neighborhood (N_8): :** 4 pixels in N_4 + 4 pixels adjacent along diagonals





Formal Specification as Point Sets

- Morphological operations can be expressed by describing images as **2D point sets**
- For example, for a binary image ($I(u,v) \in \{0,1\}$)

$$\mathcal{Q}_I = \{\boldsymbol{p} \mid I(\boldsymbol{p}) = 1\}$$

- **Example:** OR operation union of individual sets

$$\mathcal{Q}_{I_1 \vee I_2} = \mathcal{Q}_{I_1} \cup \mathcal{Q}_{I_2}$$

Dilation

- Suppose *A* and *B* are sets of pixels, **dilation of A by B**

$$A \oplus B = \bigcup_{x \in B} A_x.$$

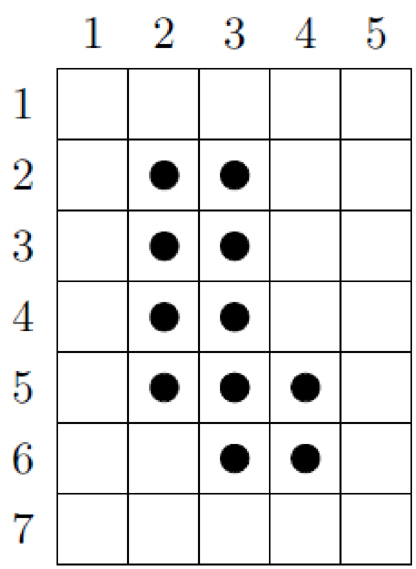
- Also called **Minkowski addition**. **Meaning?**
- Replace every pixel in *A* with copy of *B* (or vice versa)
- For every pixel *x* in *B,*
  - Translate *A* by *x*
  - Take union of all these translations

$$A \oplus B = \{(x,y) + (u,v) : (x,y) \in A, (u,v) \in B\}.$$

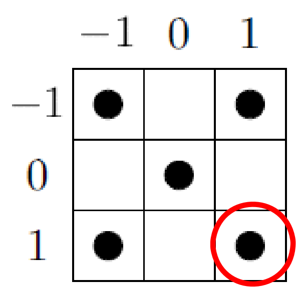
Dilation Example

- For *A* and *B* shown below

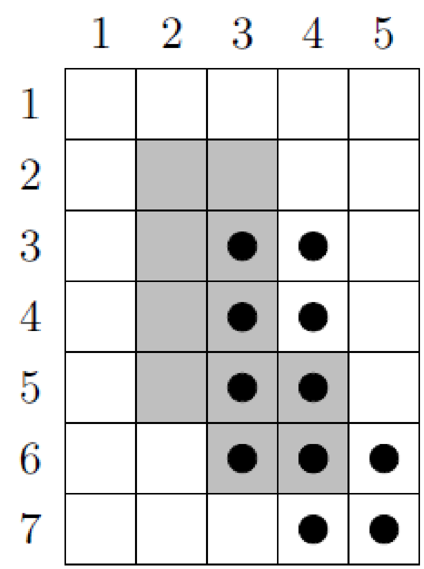
$$B = \{(0,0), (1,1), (-1,1), (1,-1), (-1,-1)\}$$



A



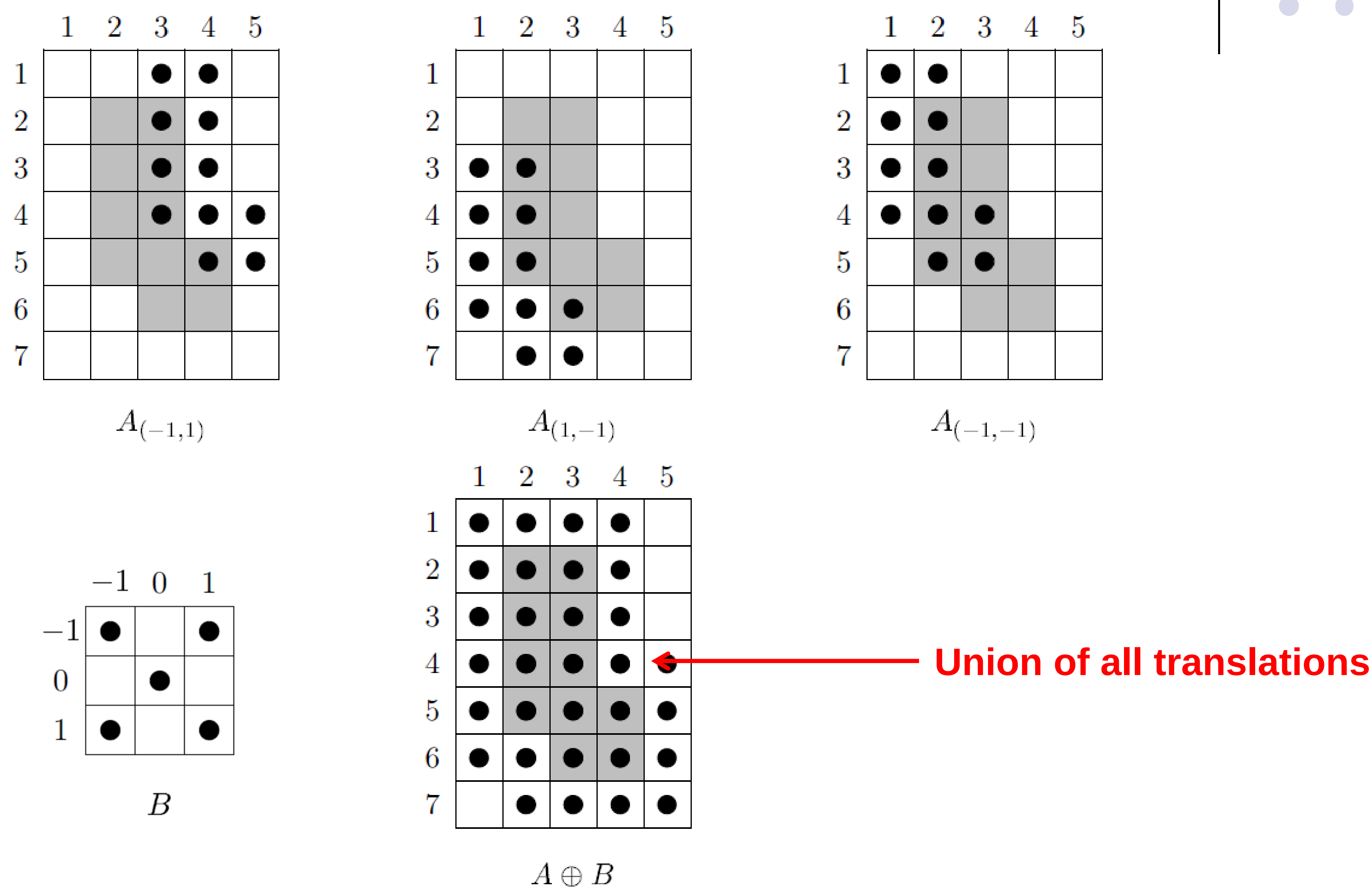
B



$A_{(1,1)}$

Translation of *A* by (1,1)

Dilation Example



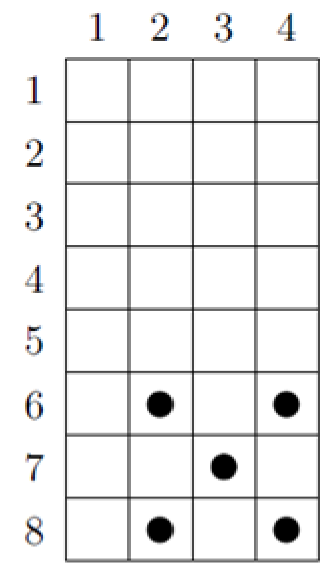
Another Dilation Example

- Dilation increases size of structure
- *A* and *B* do not have to overlap
- **Example:** For the same *A*, if we change *B* to

$$B = \{(7,3), (6,2), (6,4), (8,2), (8,4)\}$$

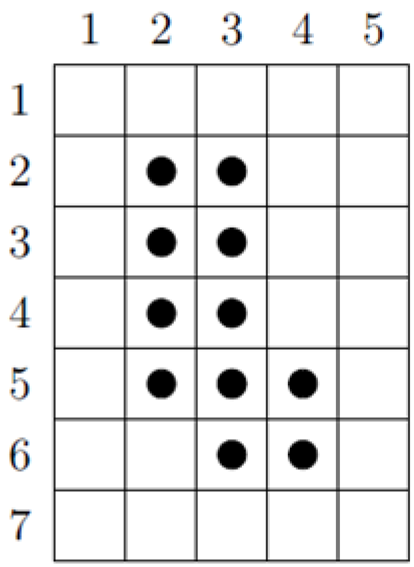
so that

$$A \oplus B = A_{(7,3)} \cup A_{(6,2)} \cup A_{(6,4)} \cup A_{(8,2)} \cup A_{(8,4)}$$

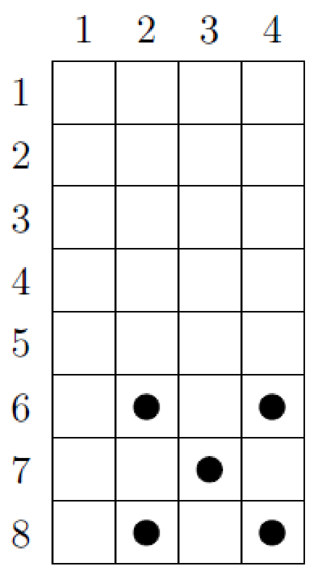


B

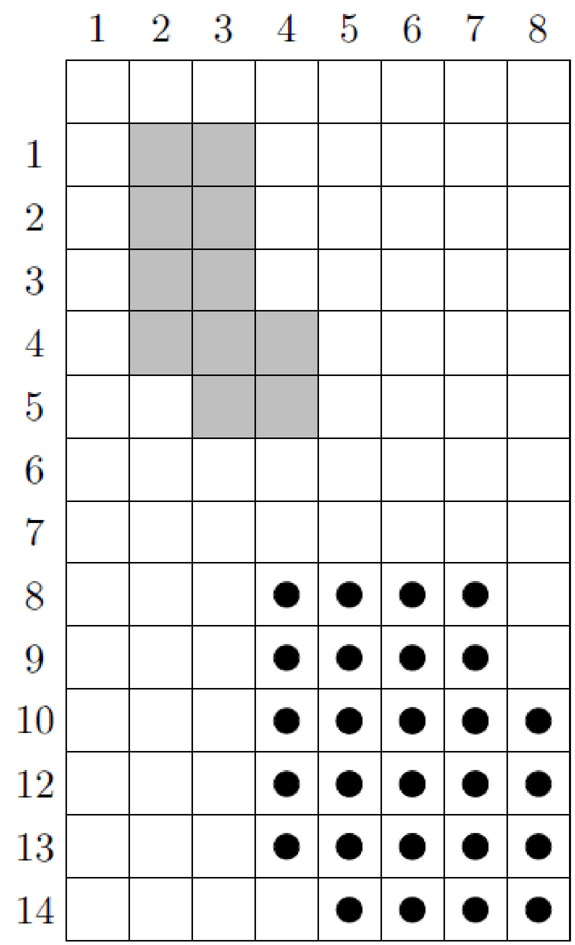
Another Dilation Example



A

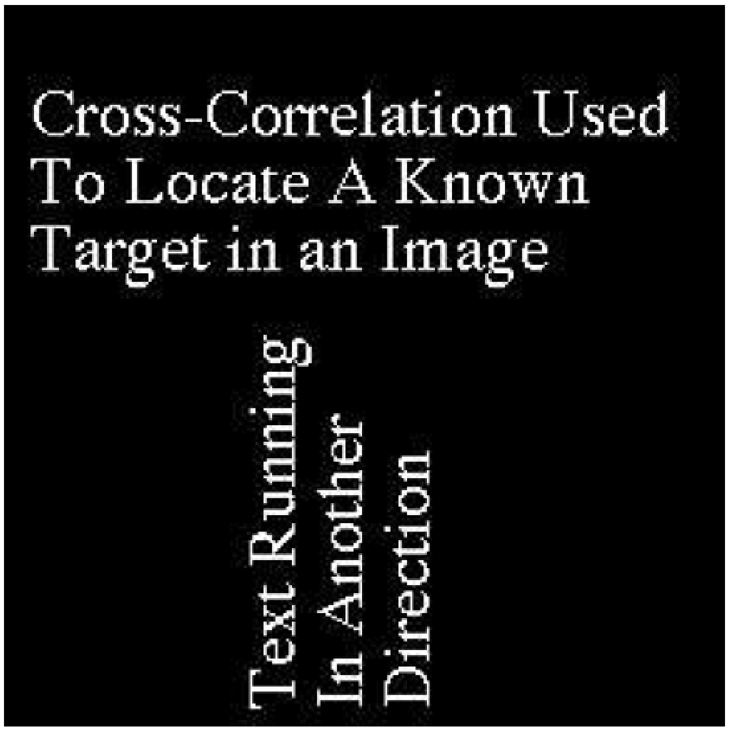


B



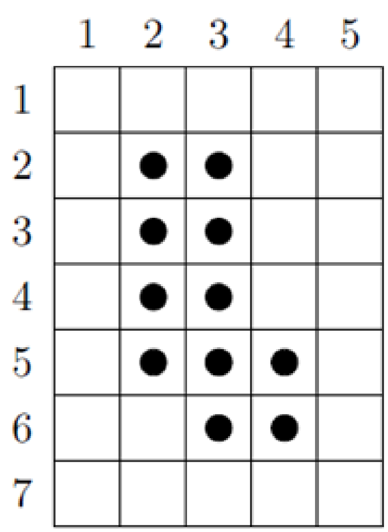
$A \oplus B$

Example: Dilation of a Binary Image

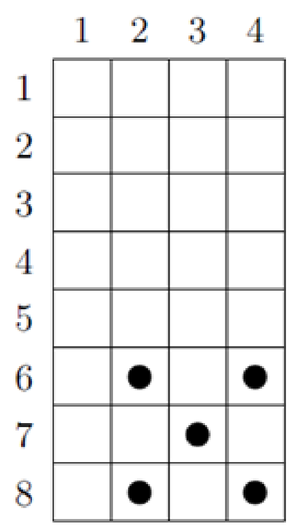


Dilation

- We usually assume
  - *A* is being processed
  - *B* is a smaller set of pixels, called the **structuring element**



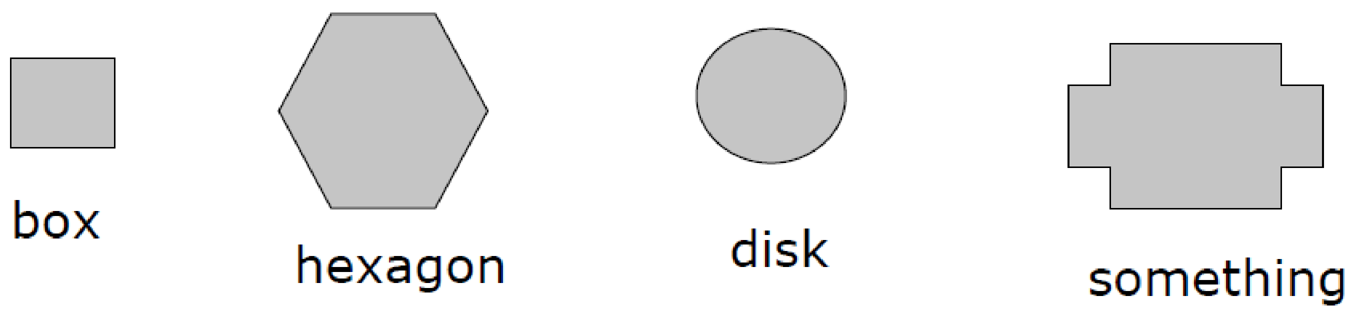
A



B

The Structuring Element

- A structuring element is a shape mask used in the basic morphological operations
- They can be any shape and size that is digitally representable, and each has an origin.



The Structuring Element

- Structuring element somewhat similar to a filter
- Contains only 0 and 1 values
- **Hot spot** marks origin of coordinate system of ***H***
- **Example of structuring element:** 1-elements marked with •, 0-cells are empty

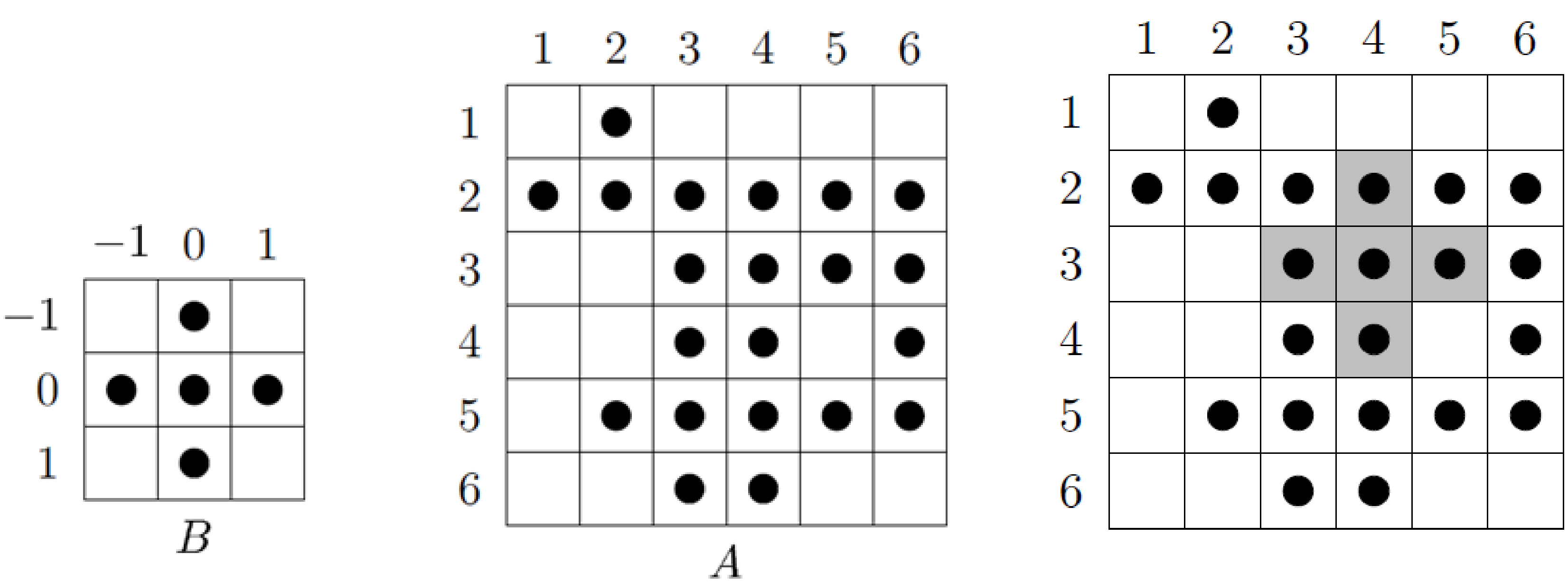
$$H(i,j) \in \{0,1\}$$

Erosion

- Given sets *A* and *B*, the **erosion of *A* by *B***

$$A \ominus B = \{w : B_w \subseteq A\}.$$

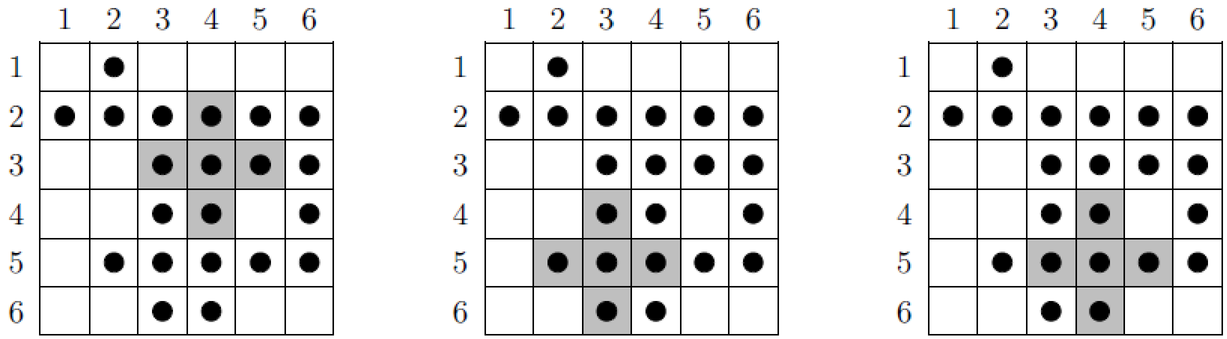
- Find all occurrences of *B* in *A*



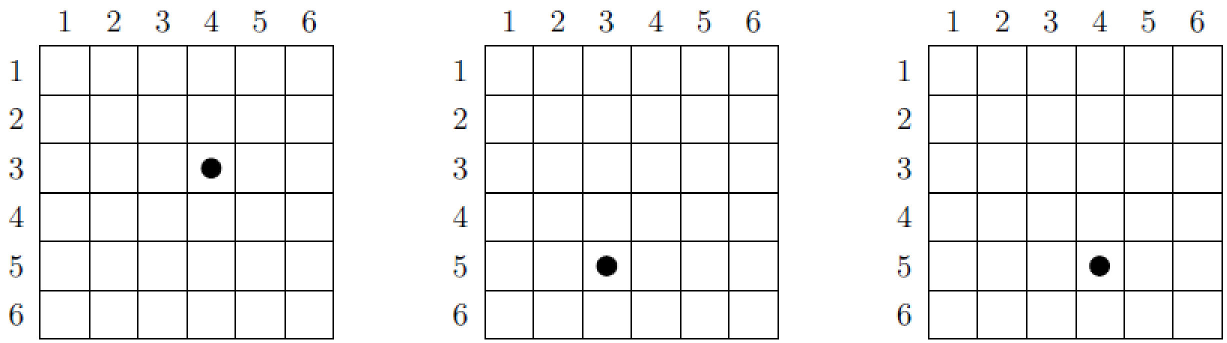
Example: 1 occurrence of *B* in *A*

Erosion

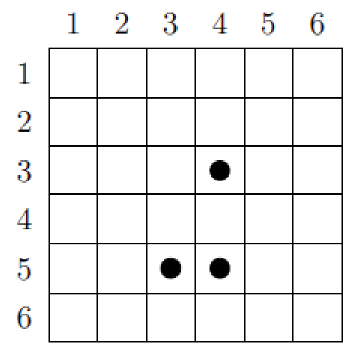
All occurrences of *B* in *A*



For each occurrences Mark center of *B*



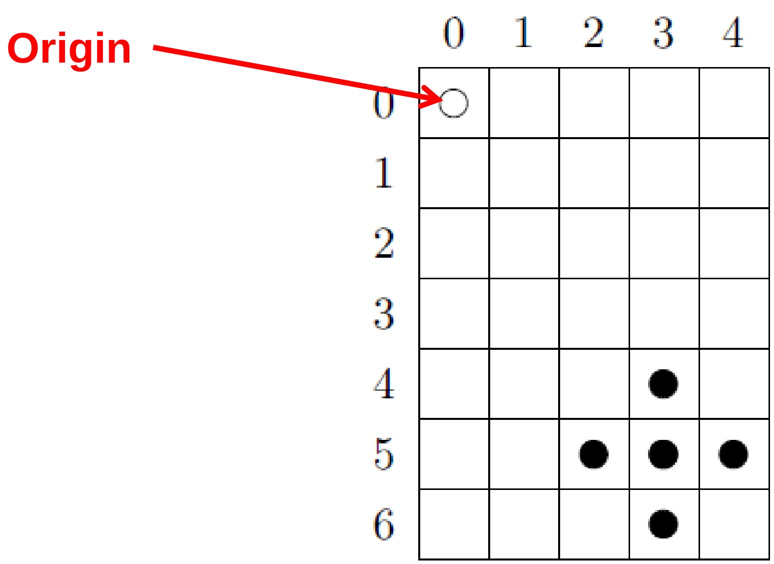
Erosion: union of center of all occurrences of *B* in *A*



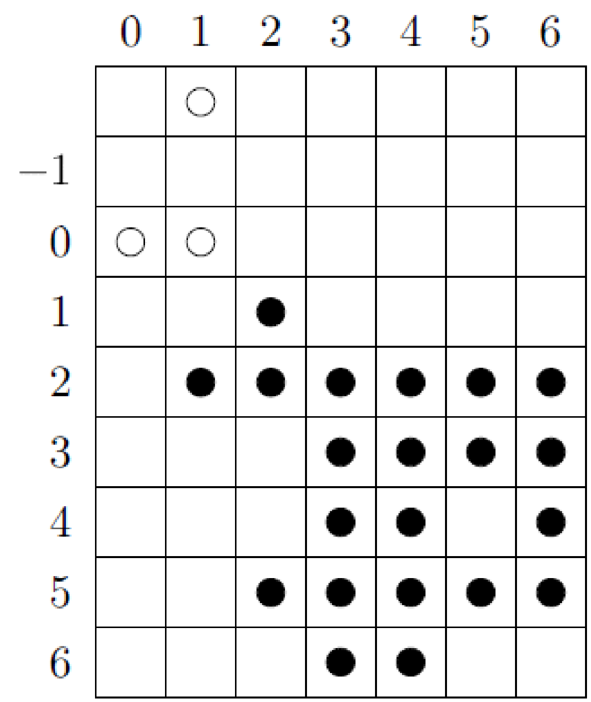
$A \ominus B$

Another Erosion Example

- The structuring element (*B*) does not have to contain the origin
- Another example where ***B*** does not contain the origin

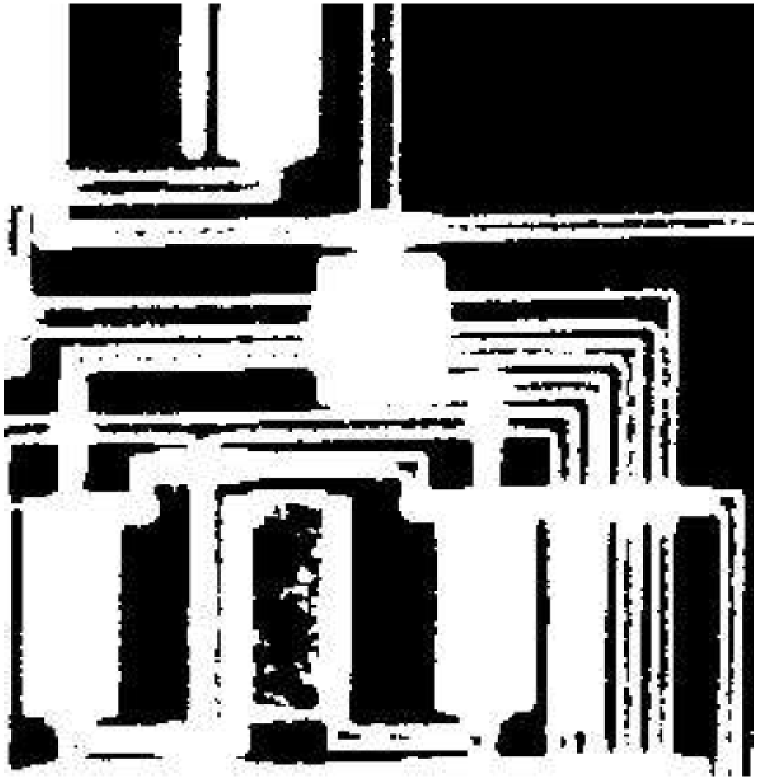


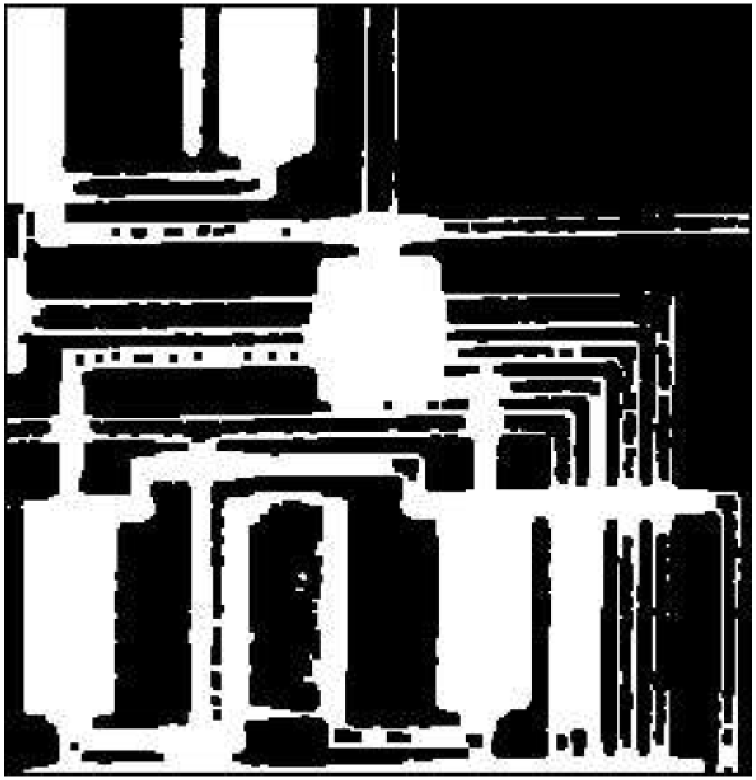
B



$A \ominus B$

Example: Erosion of Binary Image





Erosion

- Erosion related to **minkowski subtraction**

$$A - B = \bigcap_{b \in B} A_b.$$

- Erosion and dilation are **inverses** of each other
- It can be shown that

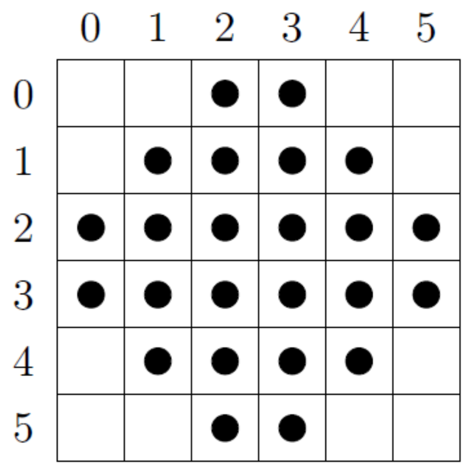
$$\overline{A \ominus B} = \overline{A} \oplus \hat{B}.$$

- And also that

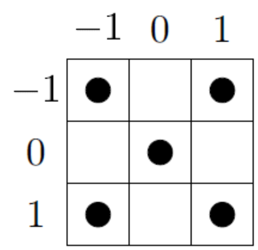
$$\overline{A \oplus B} = \overline{A} \ominus \hat{B}.$$

An Application: Boundary Detection

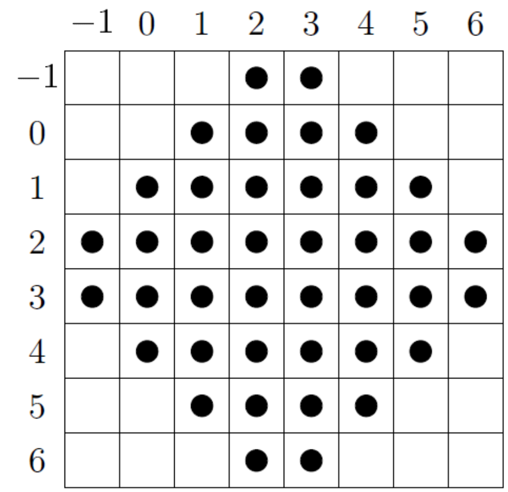
- Given an image *A* and structuring element *B*



A

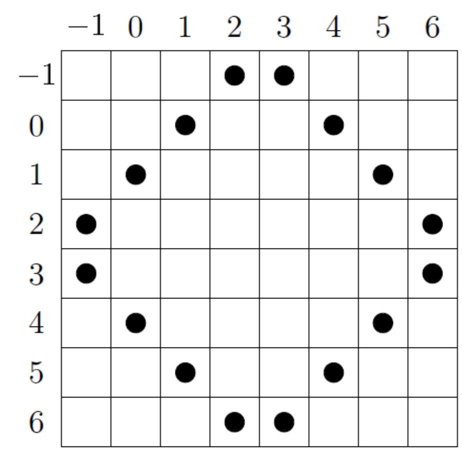


B



$A \oplus B$

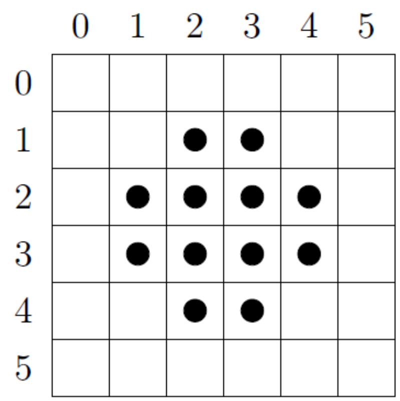
We can define **external boundary**



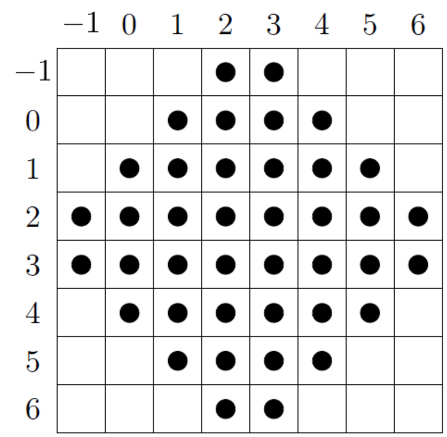
$(A \oplus B) - A)$

An Application: Boundary Detection

- **Dilation of image *A* - erosion image *A*** (by structuring element *B*)



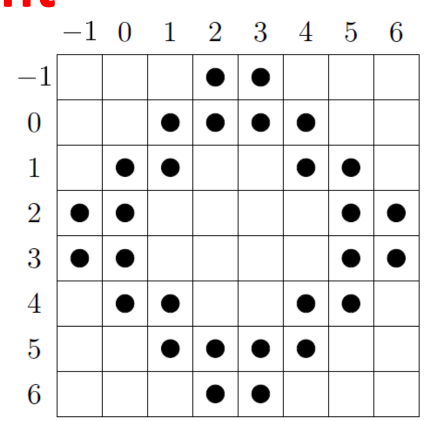
$A \ominus B$



$A \oplus B$

- We can define **morphological gradient**

$$(A \oplus B) - (A \ominus B)$$



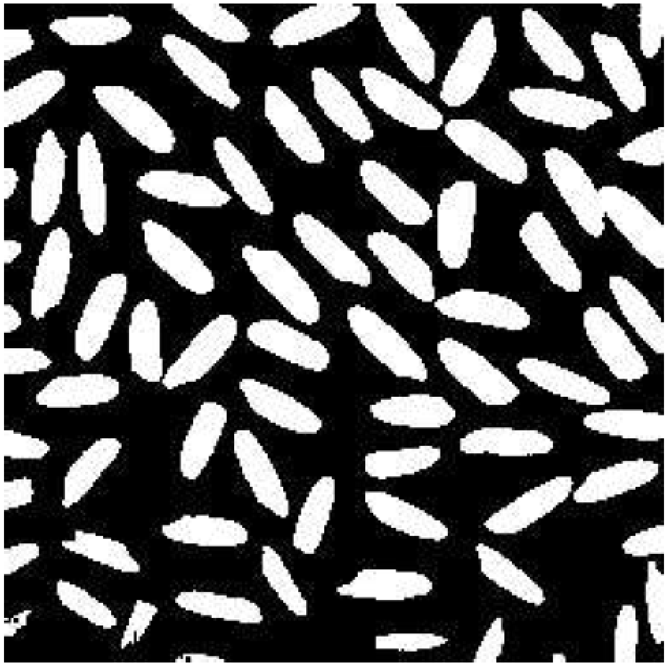
$(A \oplus B) - (A \ominus B)$

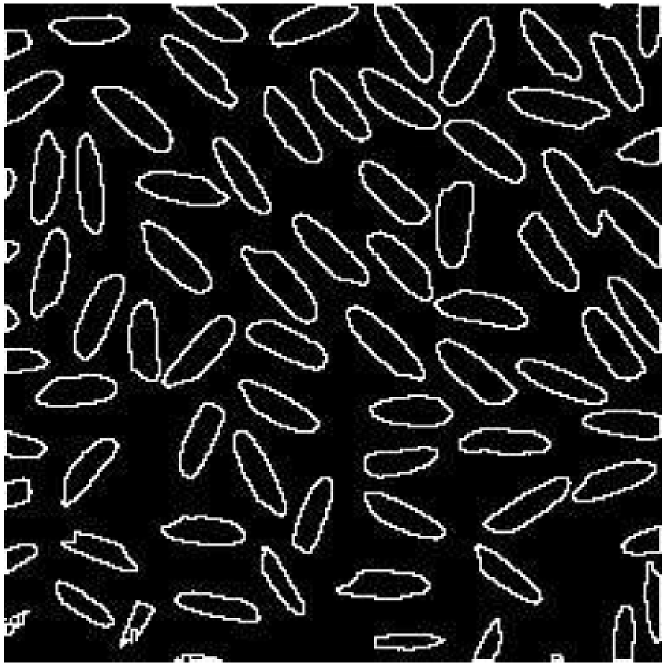
Morphological gradient = Dilation - erosion

Example: Internal Boundary of Binary Image

- We can also define **internal boundary** as

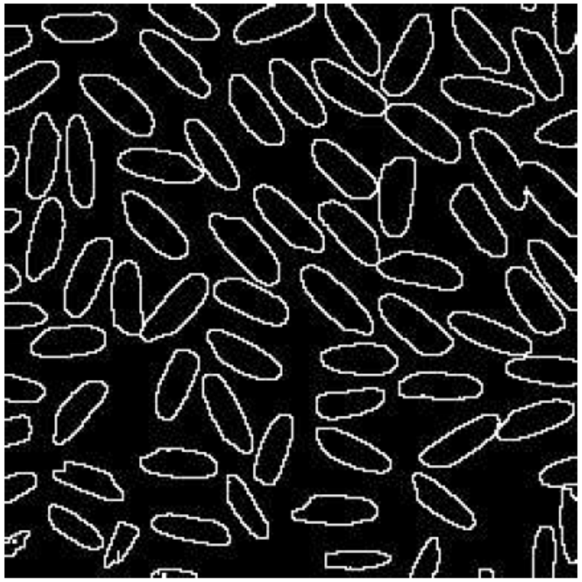
$$A - (A \ominus B)$$





Example: External Boundary and Morphological Gradient

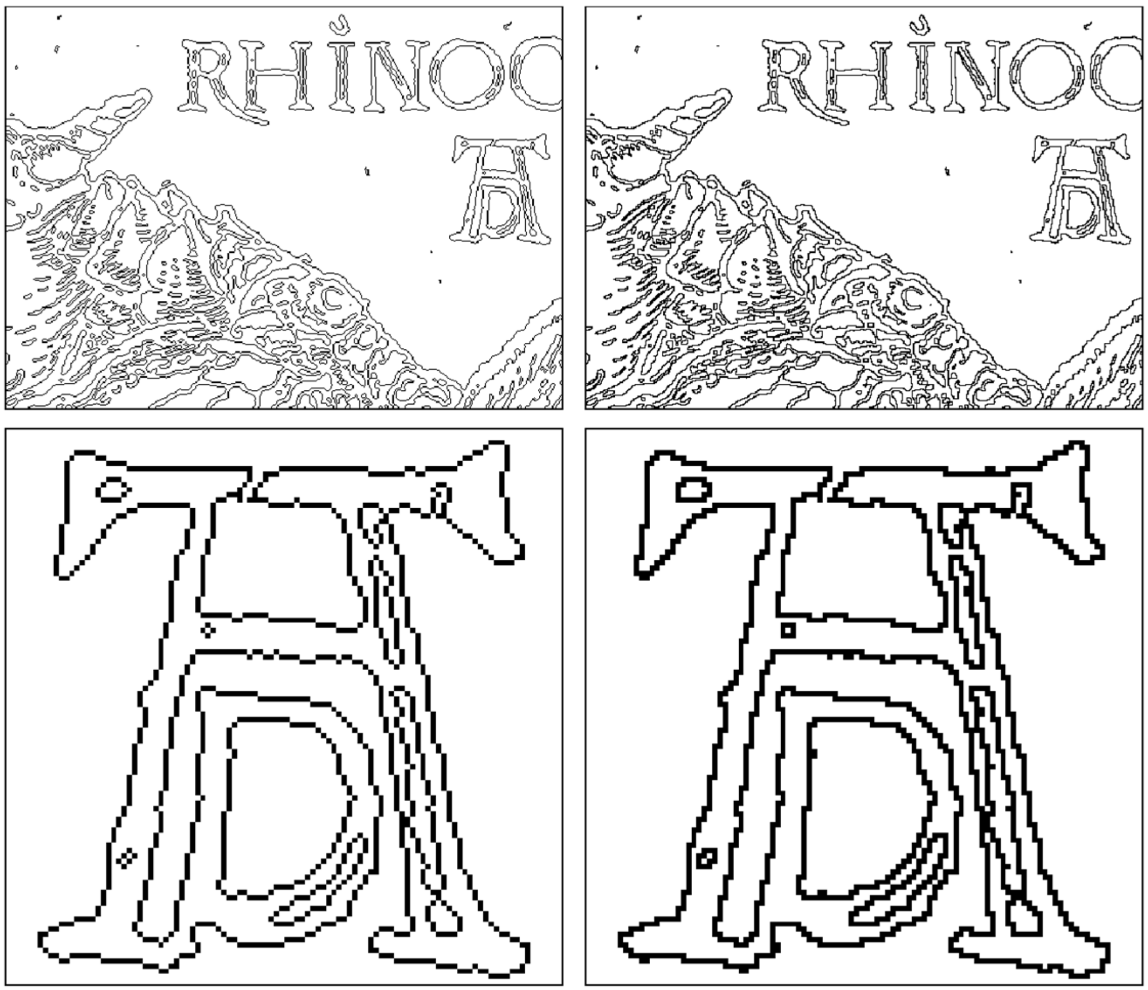
Image



External Boundary

Morphological Gradient

Example: Extraction of Boundary Pixels using Morphological Operations



(a) (b)

Properties of Dilation

- Dilation operation is **commutative**

$$I \oplus H = H \oplus I$$

- Dilation is **associative** (ordering of applying it not important)

$$(I_1 \oplus I_2) \oplus I_3 = I_1 \oplus (I_2 \oplus I_3)$$

- Thus as with separable filters, more efficient to apply large structuring element as sequence of smaller structuring elements

$$I \oplus H_{\text{big}} = (\ldots((I \oplus H_1) \oplus H_2) \oplus \ldots \oplus H_K)$$

Properties of Erosion

- Erosion is **not commutative**

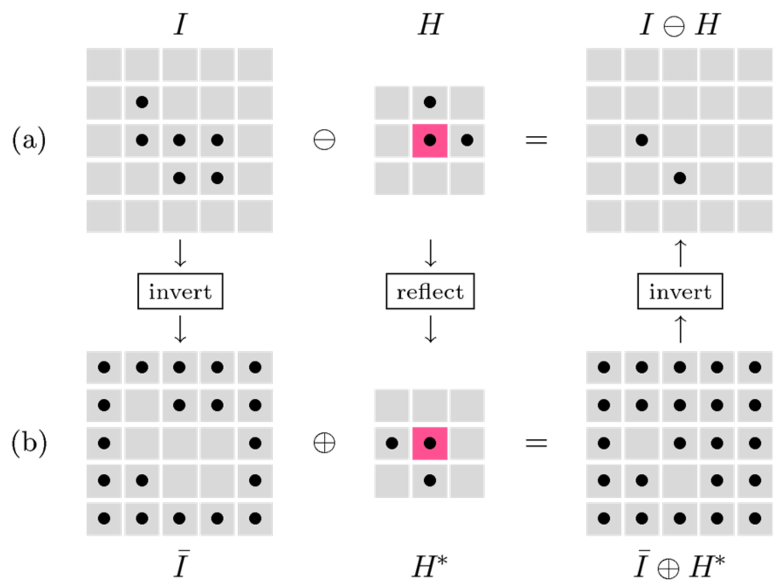
$$I \ominus H \neq H \ominus I$$

- If erosion and dilation are combined, this chain rule holds

$$(I_1 \ominus I_2) \ominus I_3 \;=\; I_1 \ominus (I_2 \oplus I_3)$$

- Dilation of **foreground** = inverting (erosion of **background**)

$$I \oplus H \;\equiv\; \overline{(\bar{I} \ominus H^*)}$$



Dilation and Erosion Algorithm

$\textsc{Dilate}\ (I, H)$

I: binary image of size $w \times h$
H: binary structuring element defined over region $\mathcal{R}_H$
Returns the dilated image $I' = I \oplus H$

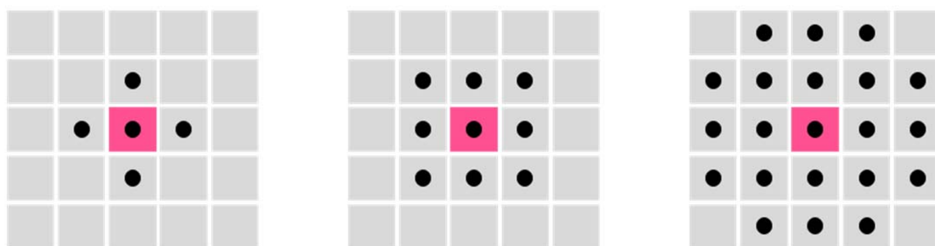
$I' \leftarrow$ new binary image of size $w \times h$
$I'(u, v) \leftarrow 0,$ for all (u, v) $\quad \triangleright\ I' \leftarrow \varnothing$
for all $(i, j) \in \mathcal{R}_H$ **do** $\quad \triangleright\ (i, j) = \boldsymbol{q}$
  if $H(i, j) = 1$ **then** $\quad \triangleright\ \boldsymbol{q} \in H$
    $\textsc{Merge the shifted}\ I_{\boldsymbol{q}}\ \textsc{with}\ I'$: $\quad \triangleright\ I' \leftarrow I' \cup I_{\boldsymbol{q}}$
    for $u \leftarrow 0 \ldots (w-1)$ **do**
      for $v \leftarrow 0 \ldots (h-1)$ **do** $\quad \triangleright\ (u, v) = \boldsymbol{p}$
        if $I(u, v) = 1$ **then** $\quad \triangleright\ \boldsymbol{p} \in I$
          $I'(u+i, v+j) \leftarrow 1$ $\quad \triangleright\ I' \leftarrow I' \cup (\boldsymbol{p}+\boldsymbol{q})$
return I'.

$\textsc{Erode}\ (I, H)$

$\bar{I} \leftarrow \textsc{Invert}(I)$ $\quad \triangleright\ \bar{I} \leftarrow \neg I$
$H^* \leftarrow \textsc{Reflect}(H)$
return $\textsc{Invert}(\textsc{Dilate}(\bar{I}, H^*))$. $\quad \triangleright\ I \ominus H = \overline{(\bar{I} \oplus H^*)}$

Designing Morphological Filters

- A morphological filter is specified by:
  - Type of operation (e.g. dilation, erosion)
  - Contents of structuring element

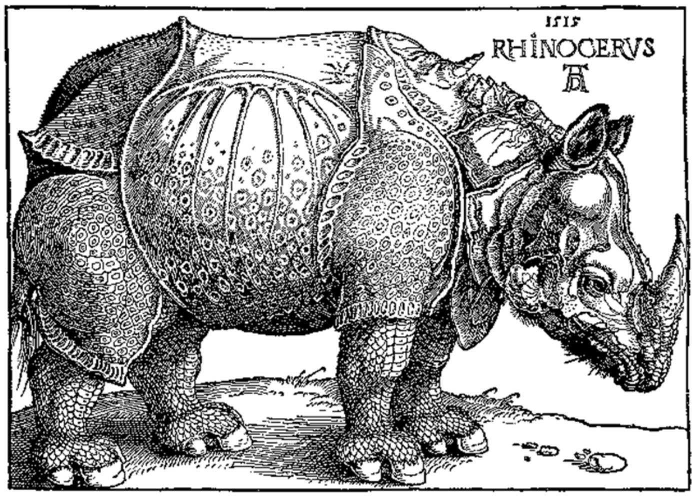


(a) 4-neighborhood (b) 8-neighborhood (c) Small Disk (circular)

- In practice, quasi-circular shaped structuring elements used
- Dilation with circular structuring of radius ***r*** adds thickness ***r***
- Erosion with circular structuring of radius ***r*** removes thickness ***r***

Example: Dilation and Erosion

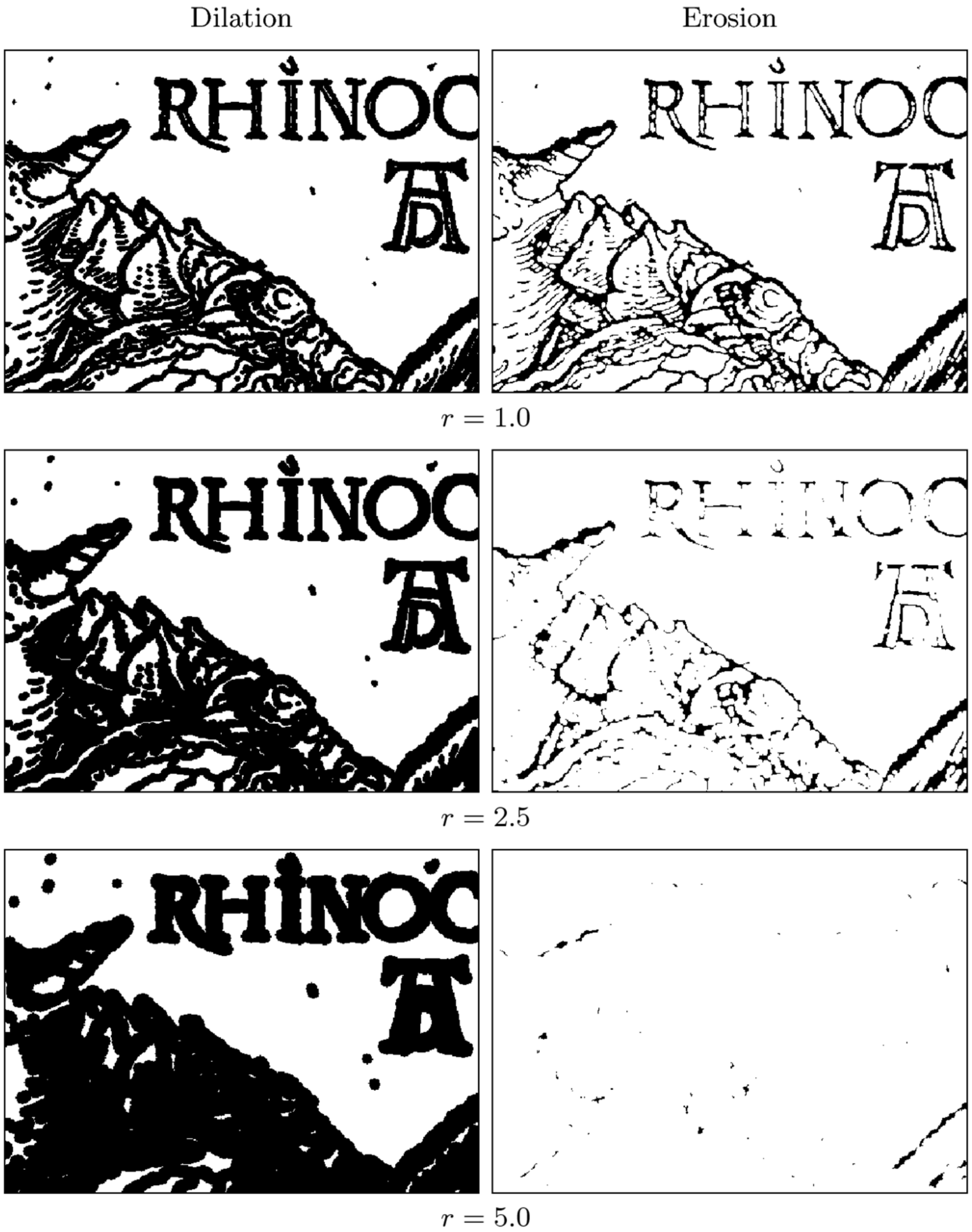
- What if we erode and dilate the following image with disk-shaped structuring element?

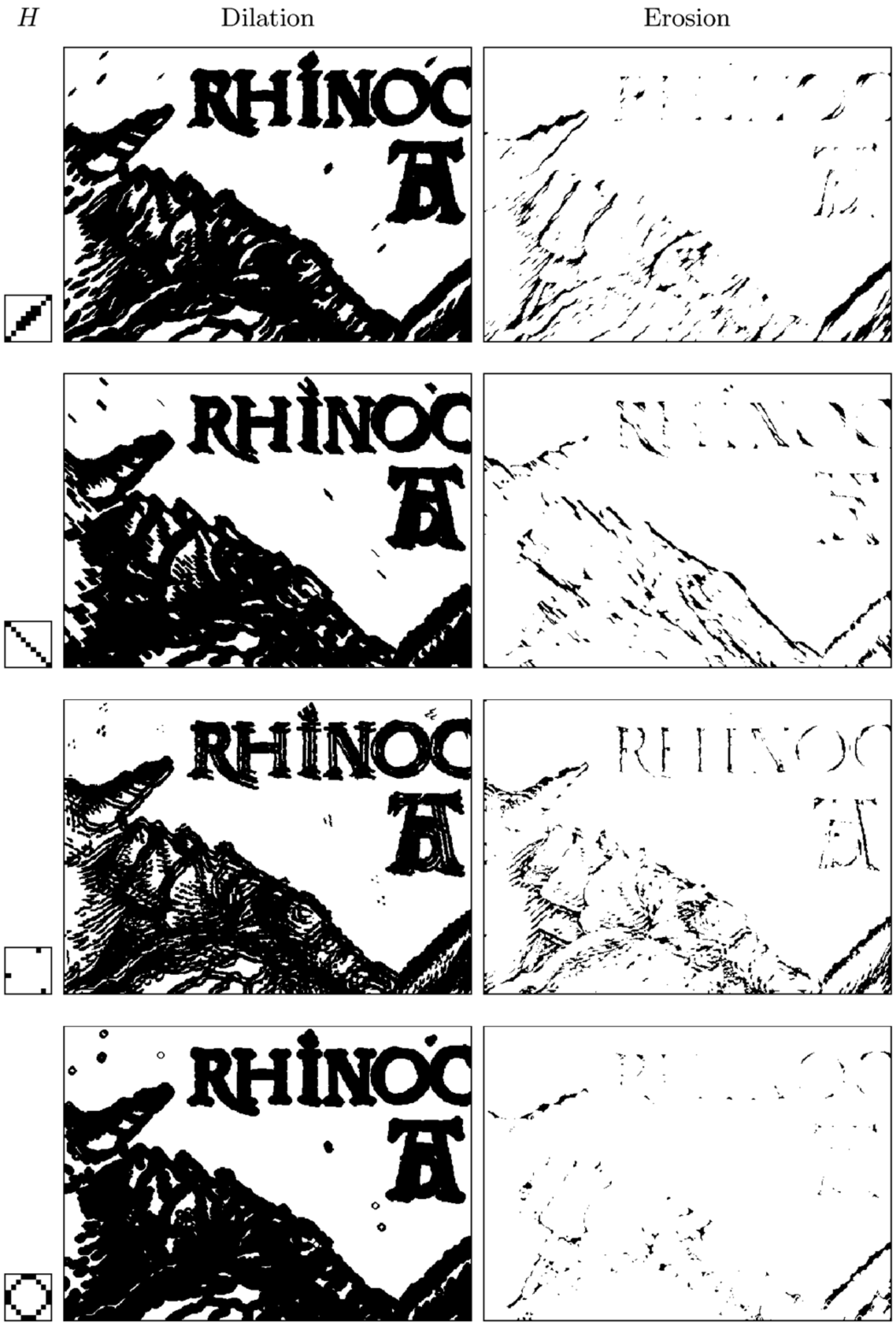


Original image

Apply dilation and erosion to this close up section

Example: Dilation and Erosion



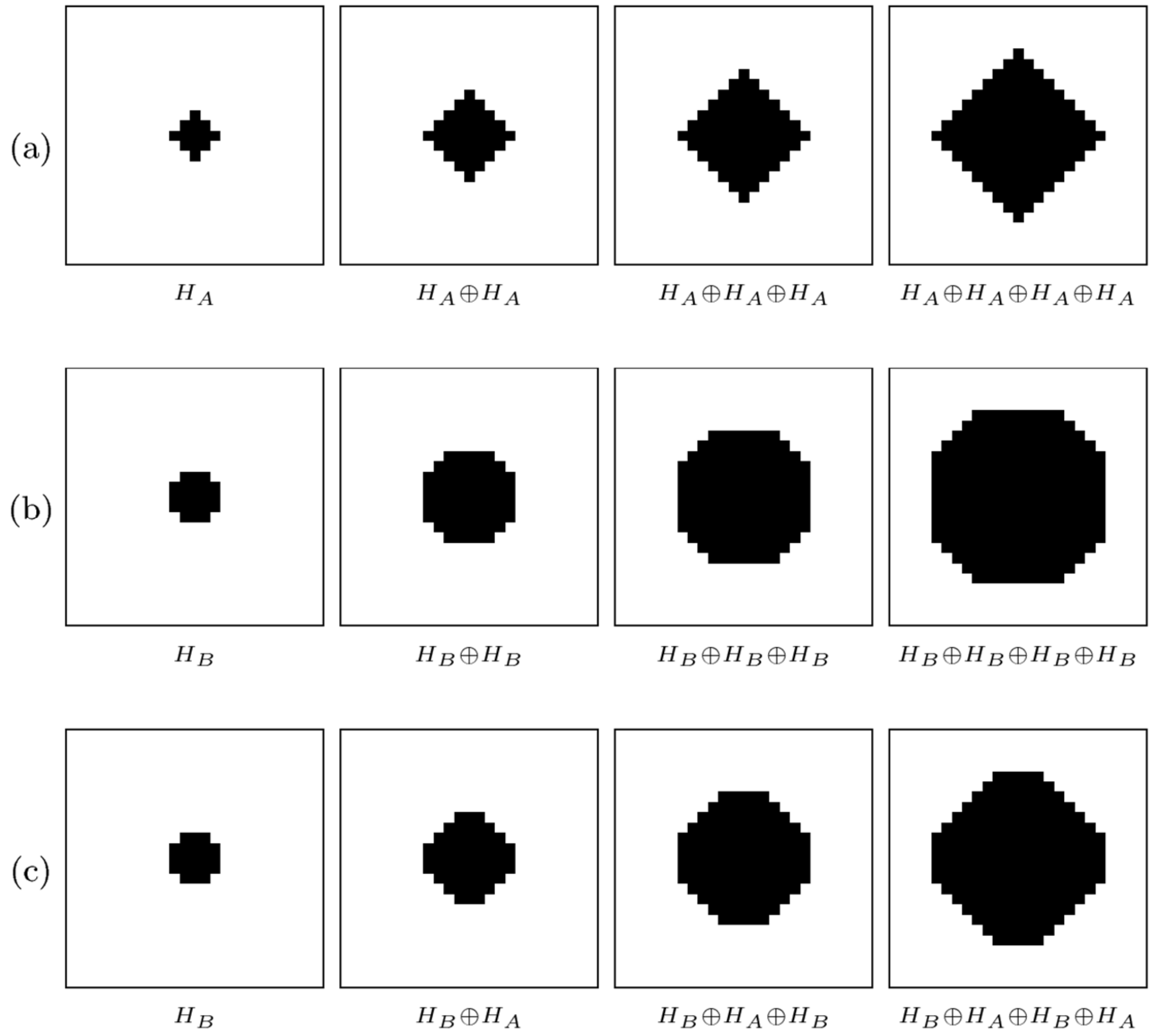


Dilation and Erosion using Different Structuring Elements

Example: Composing Large Filters by Repeatedly Applying Smaller Filters

- More efficient
- E.g. composing

Isotropic filter



References

- Wilhelm Burger and Mark J. Burge, Digital Image Processing, Springer, 2008
- University of Utah, CS 4640: Image Processing Basics, Spring 2012
- Rutgers University, CS 334, Introduction to Imaging and Multimedia, Fall 2012
- Gonzales and Woods, Digital Image Processing (3rd edition), Prentice Hall