Industrial Computer Vision
- Boundary Extraction

5^{th} lecture, 2021.10.06
Lecturer: Youngbae Hwang

Contents

- Edge Detection
- Hough Transform
- Connected Component Labeling

Image Segmentation

- Image segmentation divides an image into regions that are connected and have some similarity within the region and some difference between adjacent regions.
- The goal is usually to find individual objects in an image.
- For the most part there are fundamentally two kinds of approaches to segmentation: discontinuity and similarity.
  - Similarity may be due to pixel intensity, color or texture.
  - Differences are sudden changes (discontinuities) in any of these, but especially sudden changes in intensity along a boundary line, which is called an edge.

Detection of Discontinuities

- There are three kinds of discontinuities of intensity: points, lines and edges.
- The most common way to look for discontinuities is to scan a small mask over the image. The mask determines which kind of discontinuity to look for.

$$R = w_1 z_1 + w_2 z_2 + \ldots + w_9 z_9 = \sum_{i=1}^{9} w_i z_i$$

FIGURE 10.1 A general 3×3 mask.

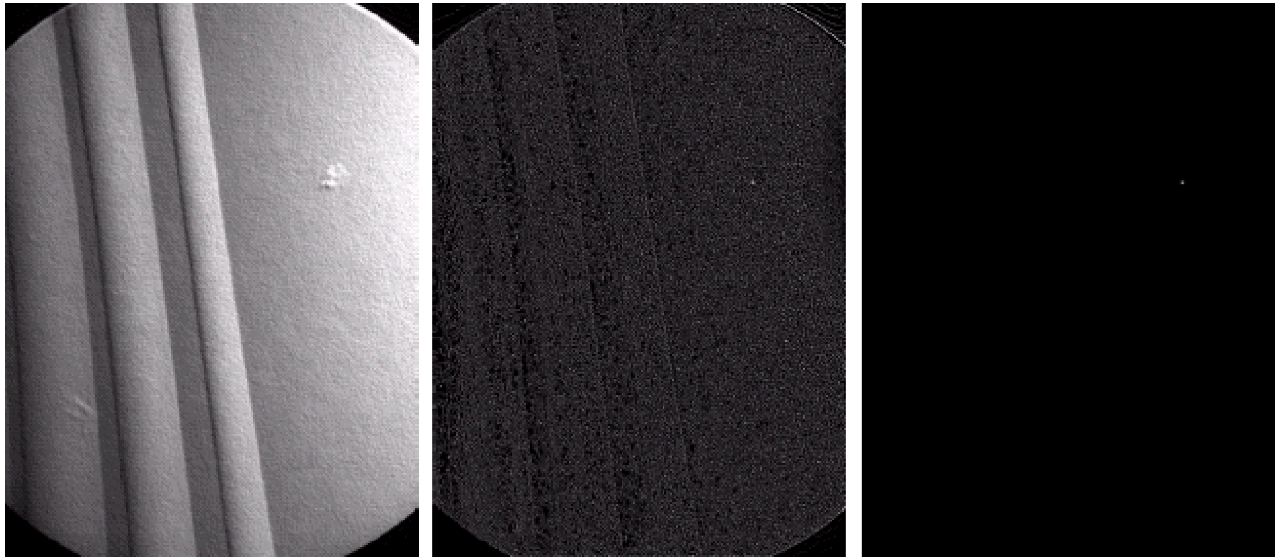
w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Detection of Discontinuities: Point Detection

$$|R| \ge T$$

where T : a nonnegativ e threshold

–1	–1	–1
–1	8	–1
–1	–1	–1



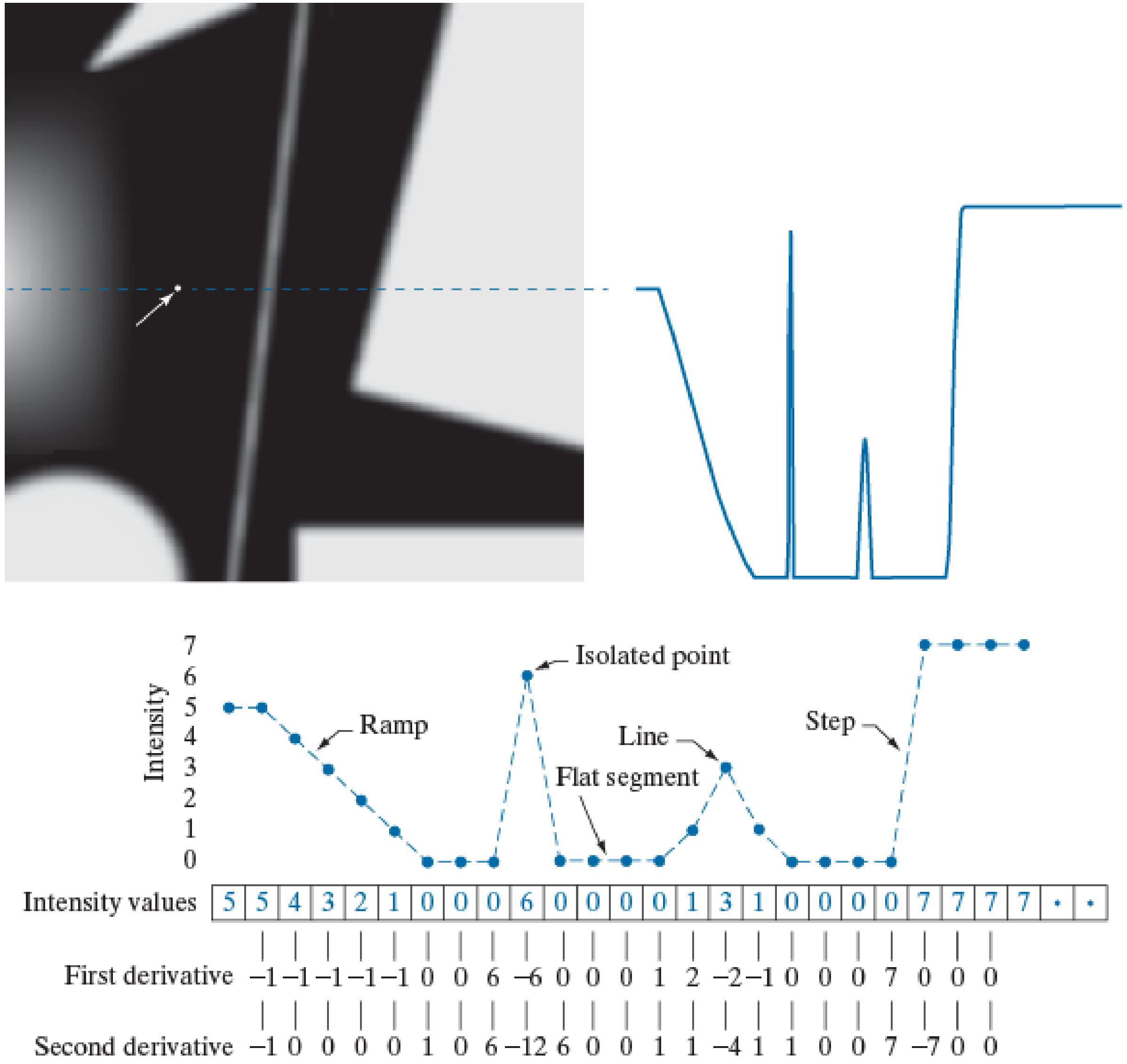
a
b c d

FIGURE 10.2
(a) Point detection mask.
(b) X-ray image of a turbine blade with a porosity.
(c) Result of point detection.
(d) Result of using Eq. (10.1-2).
(Original image courtesy of X-TEK Systems Ltd.)

Detection of Discontinuities: Line Detection

a b
c

FIGURE 10.2
(a) Image.
(b) Horizontal intensity profile that includes the isolated point indicated by the arrow.
(c) Subsampled profile; the dashes were added for clarity. The numbers in the boxes are the intensity values of the dots shown in the profile. The derivatives were obtained using Eqs. (10-4) for the first derivative and Eq. (10-7) for the second.



Detection of Discontinuities: Line Detection

- Only slightly more common than point detection is to find a one pixel wide li ne in an image.
- For digital images the only three point straight lines are only horizontal, verti cal, or diagonal (+ or –45°).

FIGURE 10.3 Line masks.

–1	–1	–1
2	2	2
–1	–1	–1

Horizontal

–1	–1	2
–1	2	–1
2	–1	–1

+45°

–1	2	–1
–1	2	–1
–1	2	–1

Vertical

2	–1	–1
–1	2	–1
–1	–1	2

–45°

Detection of Discontinuities: Line Detection

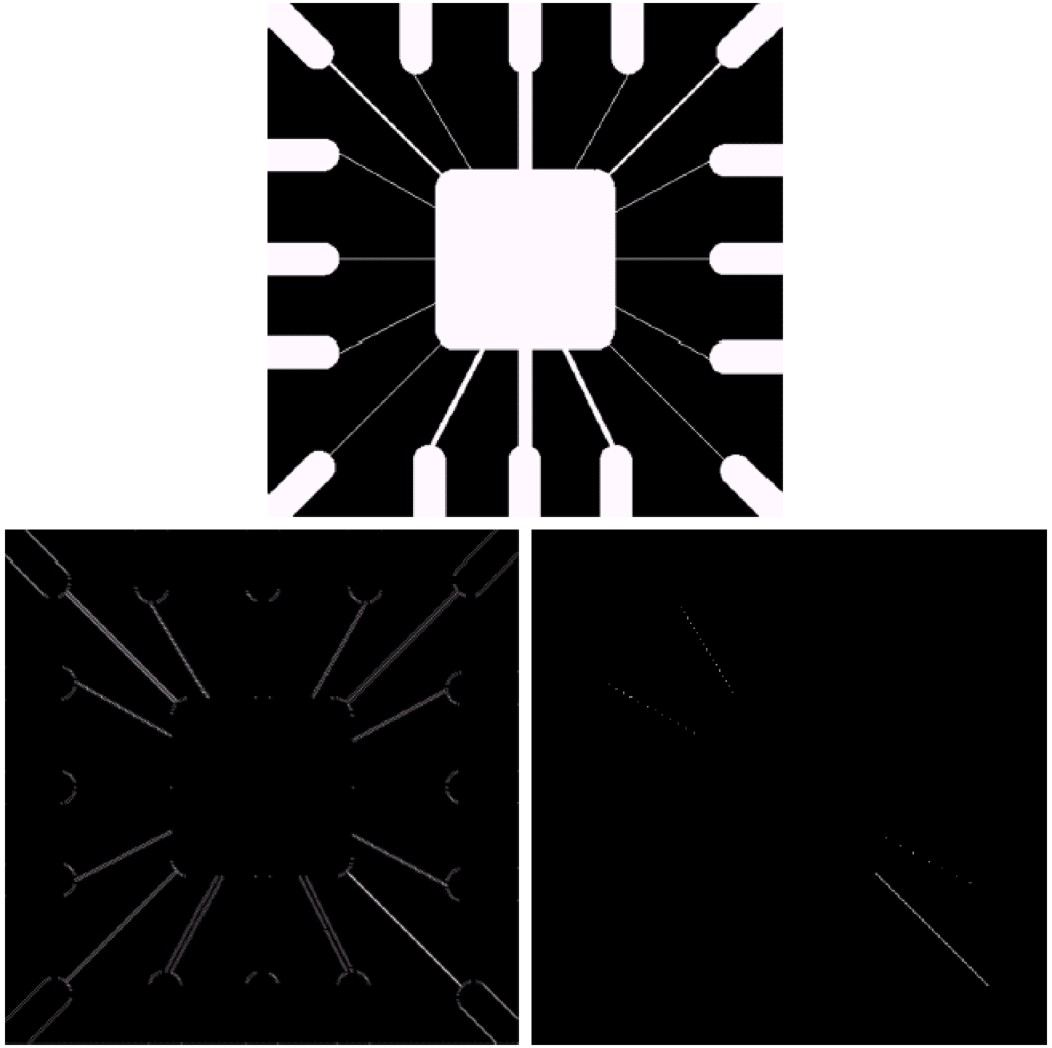
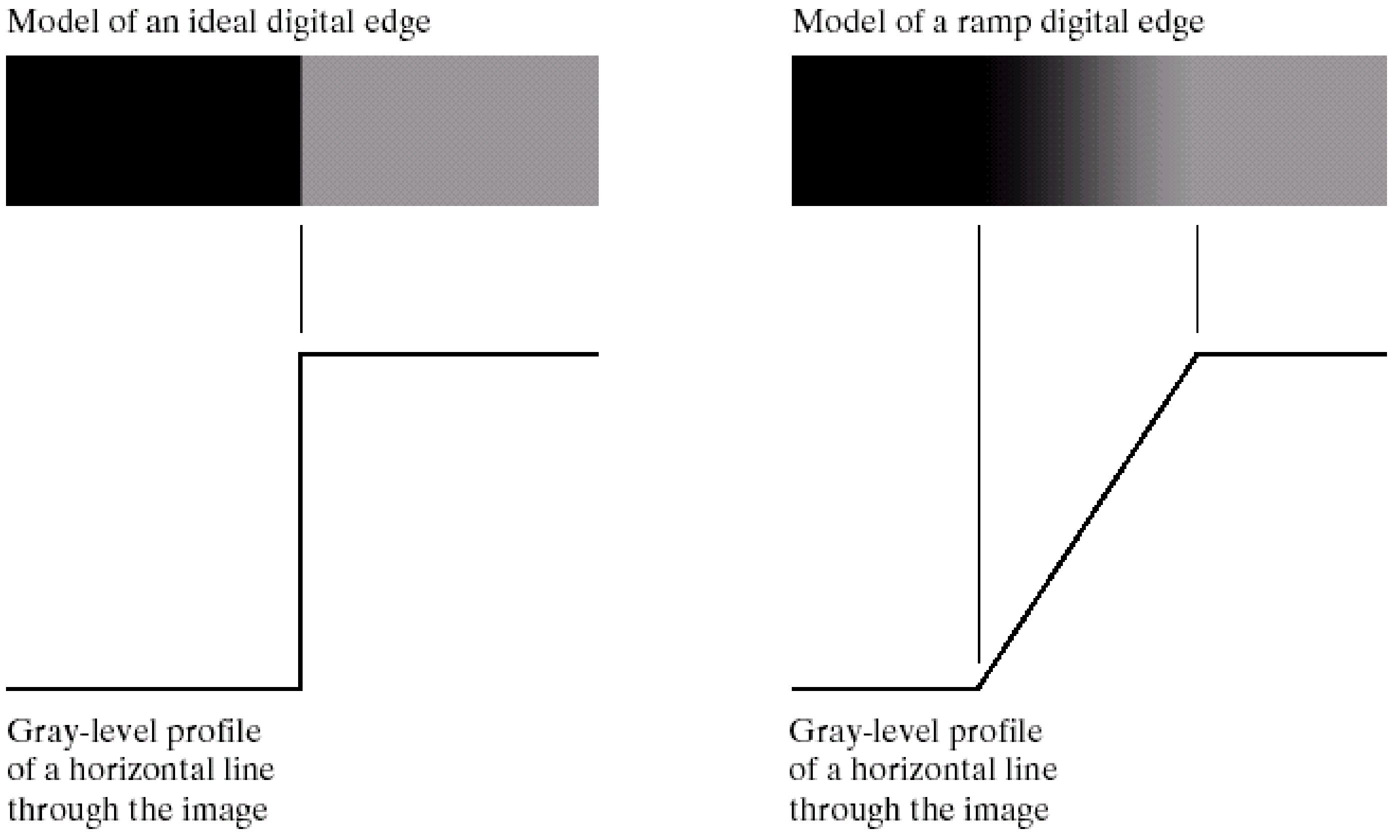


FIGURE 10.4 Illustration of line detection. (a) Binary wire-bond mask. (b) Absolute value of result after processing with −45° line detector. (c) Result of thresholding image (b).

Detection of Discontinuities: Edge Detection



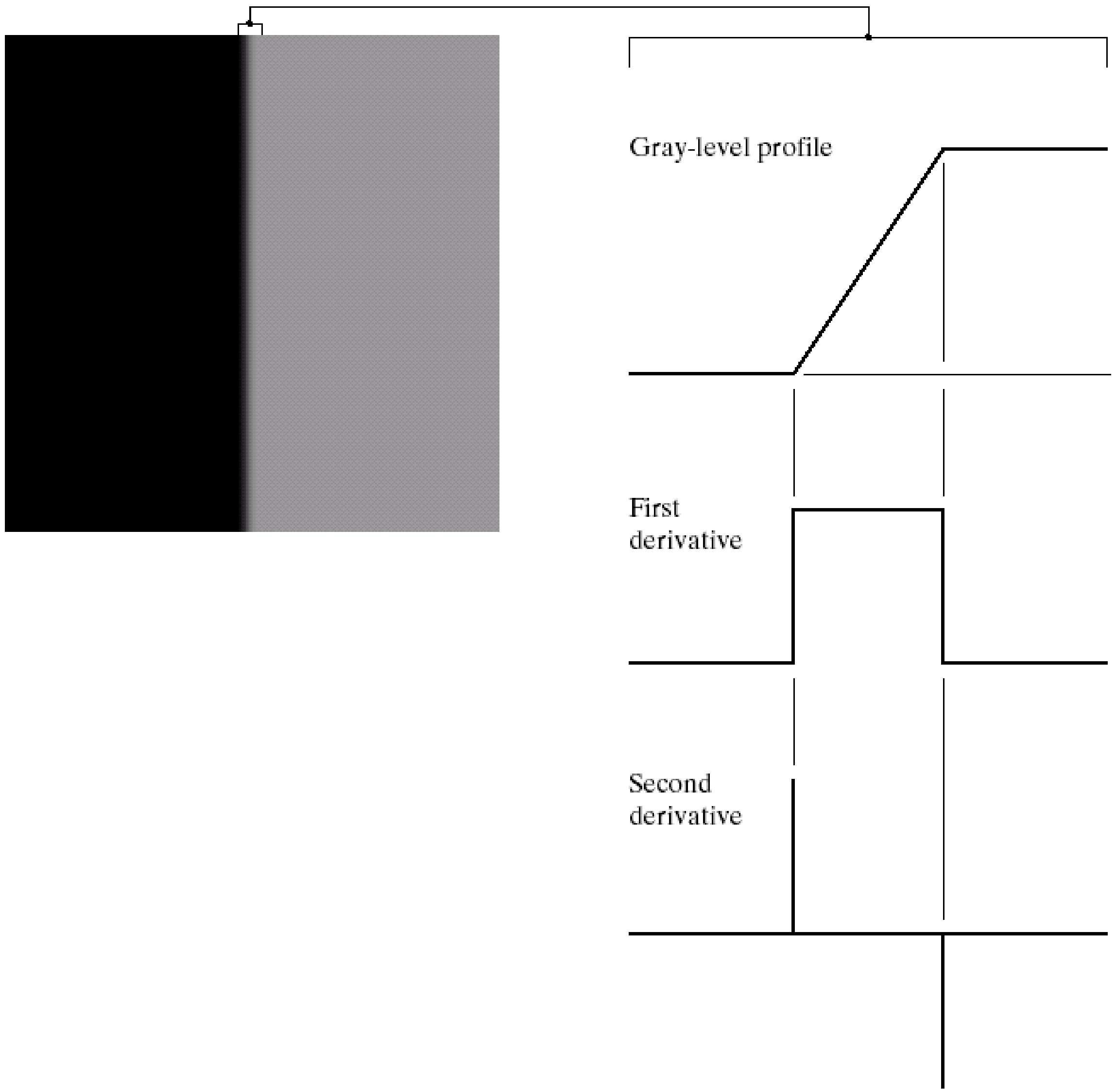
a b

FIGURE 10.5 (a) Model of an ideal digital edge. (b) Model of a ramp edge. The slope of the ramp is proportional to the degree of blurring in the edge.

Detection of Discontinuities: Edge Detection

a b

FIGURE 10.6
(a) Two regions separated by a vertical edge.
(b) Detail near the edge, showing a gray-level profile, and the first and second derivatives of the profile.



Detection of Discontinuities: Edge Detection

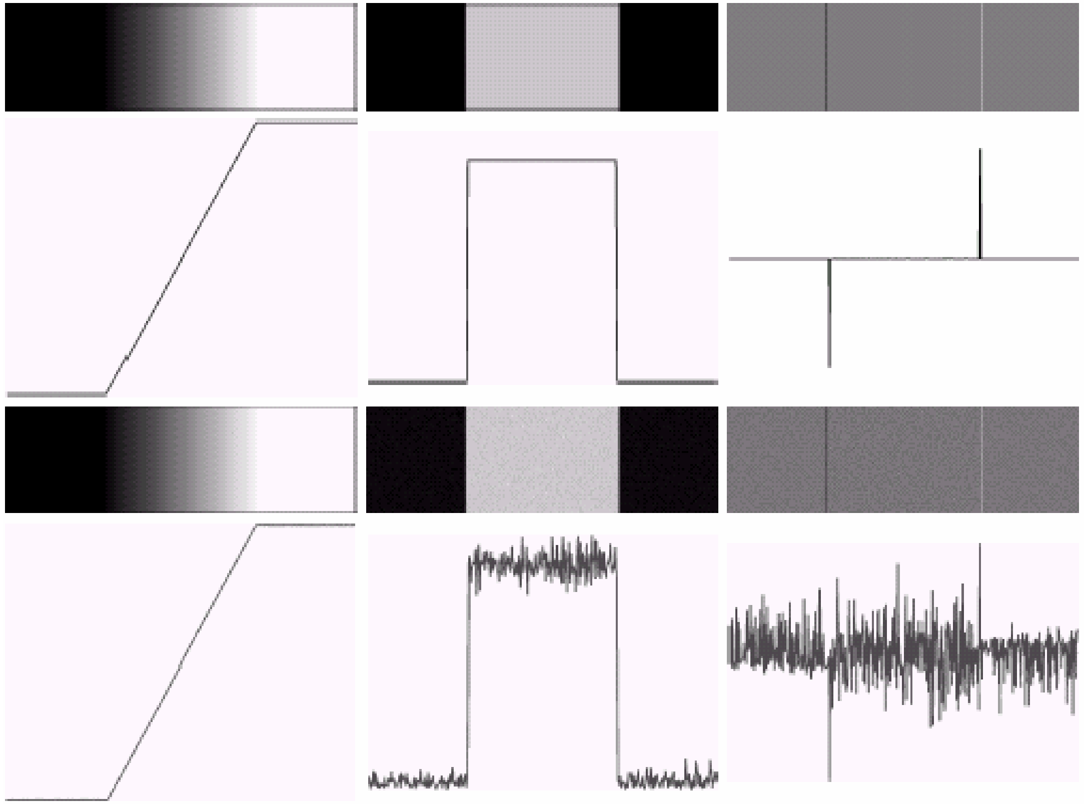


FIGURE 10.7 First column: images and gray-level profiles of a ramp edge corrupted by random Gaussian noise of mean 0 and $\sigma = 0.0, 0.1, 1.0,$ and 10.0, respectively. Second column: first-derivative images and gray-level profiles. Third column: second-derivative images and gray-level profiles.

a
b
c
d

Detection of Discontinuities: Edge Detection

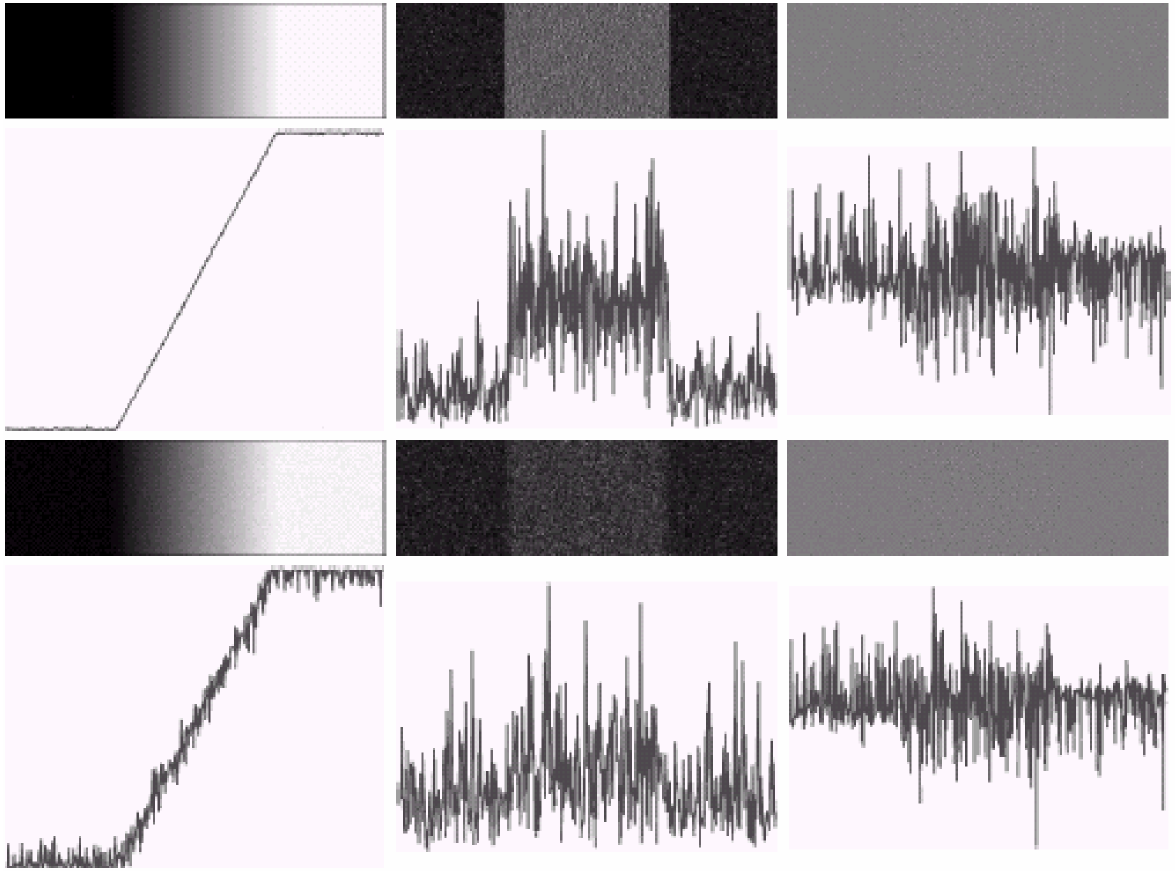


FIGURE 10.7 First column: images and gray-level profiles of a ramp edge corrupted by random Gaussian noise of mean 0 and $\sigma = 0.0, 0.1, 1.0,$ and 10.0, respectively. Second column: first-derivative images and gray-level profiles. Third column: second-derivative images and gray-level profiles.

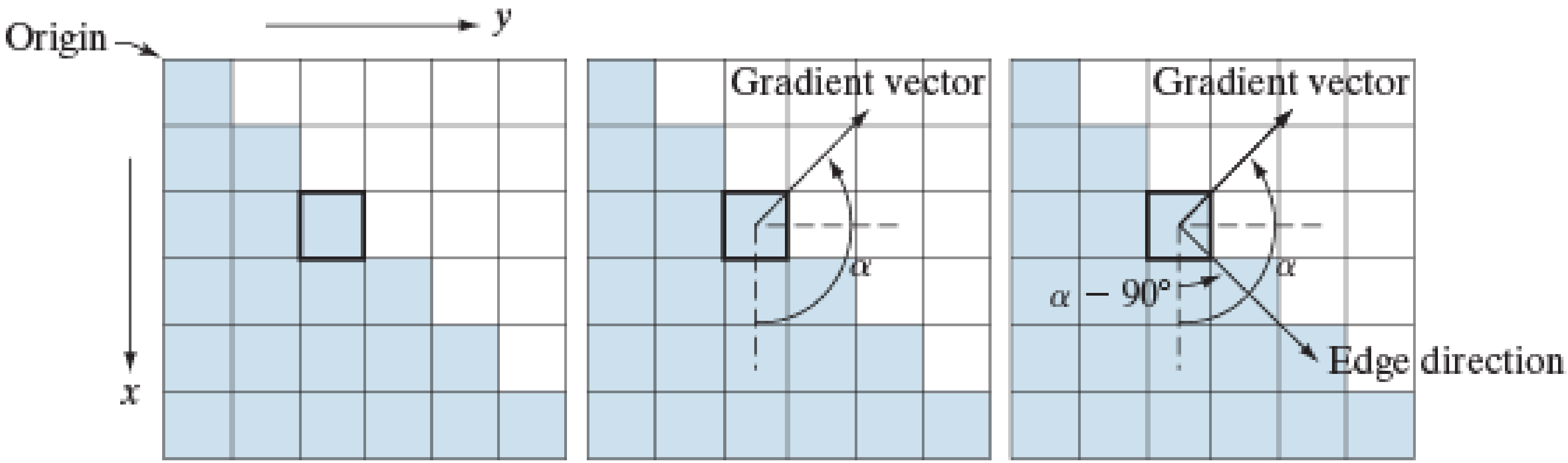
a
b
c
d

Detection of Discontinuities: Gradient Operators

- First-order derivatives:
  - The gradient of an image f(x,y) at location (x,y) is defined as the vector:

$$\nabla \mathbf{f} = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

  - The magnitude of this vector: $\alpha(x, y) = \tan^{-1}\left(\frac{G_x}{G_y}\right)$
  - The direction of this vector: $\nabla f = \mathrm{mag}(\nabla \mathbf{f}) = \left[G_x^2 + G_y^2\right]^{1/2}$



Detection of Discontinuities: Gradient Operators

Roberts cross-gradient operators

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

Roberts

Prewitt operators

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Prewitt

Sobel operators

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Sobel

Detection of Discontinuities: Gradient Operators

Prewitt masks for
detecting diagonal edges

0	1	1
−1	0	1
−1	−1	0

−1	−1	0
−1	0	1
0	1	1

Prewitt

Sobel masks for
detecting diagonal edges

0	1	2
−1	0	1
−2	−1	0

−2	−1	0
−1	0	1
0	1	2

Sobel

a b
c d

FIGURE 10.9 Prewitt and Sobel masks for detecting diagonal edges.

Detection of Discontinuities: Gradient Operators: Example

a	b
c	d

FIGURE 10.10 (a) Original image. (b) $|G_x|$, component of the gradient in the x-direction. (c) $|G_y|$, component in the y-direction. (d) Gradient image, $|G_x| + |G_y|$.

$$\nabla f \approx |G_x| + |G_y|$$

Detection of Discontinuities: Gradient Operators: Example

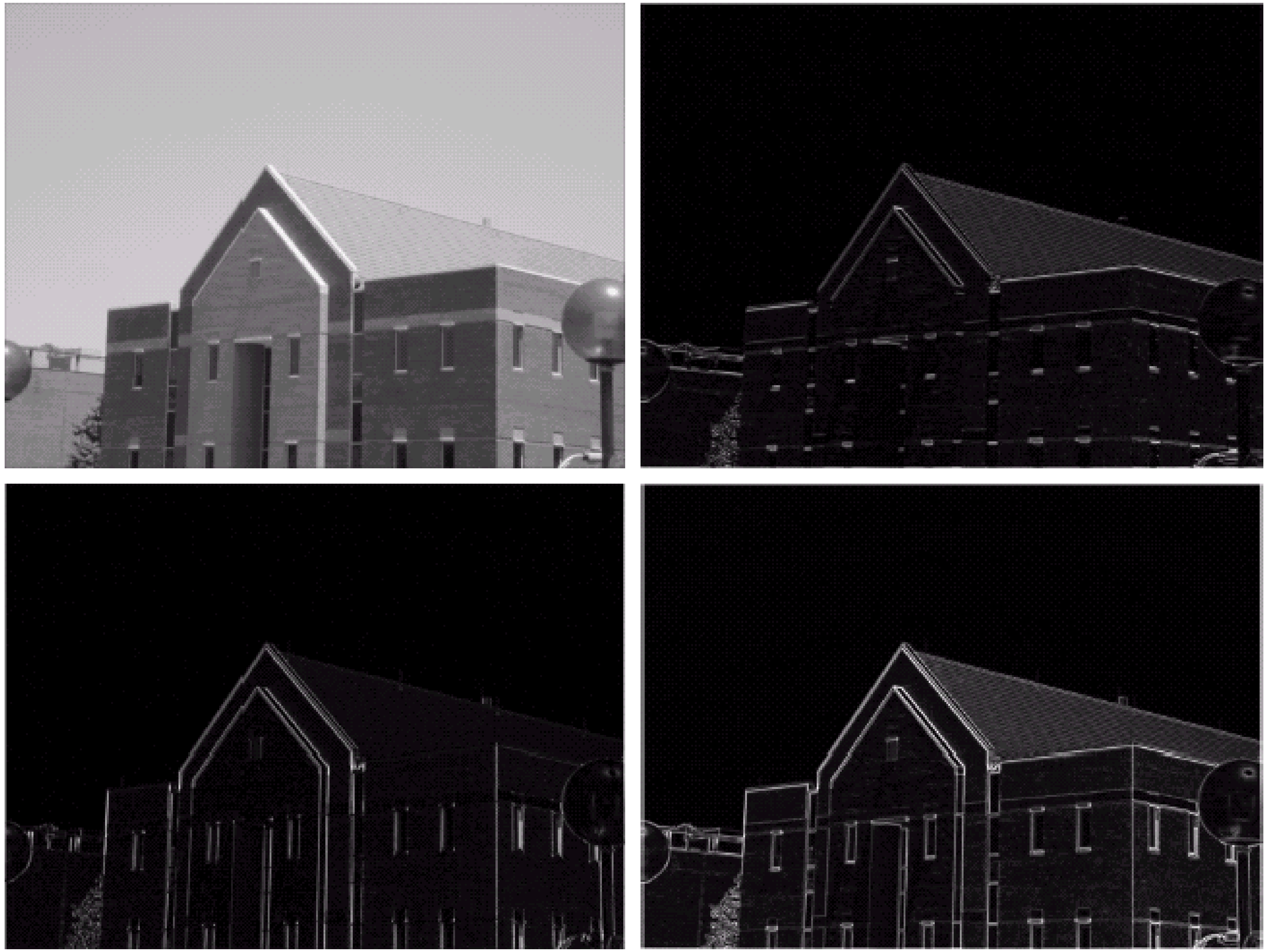
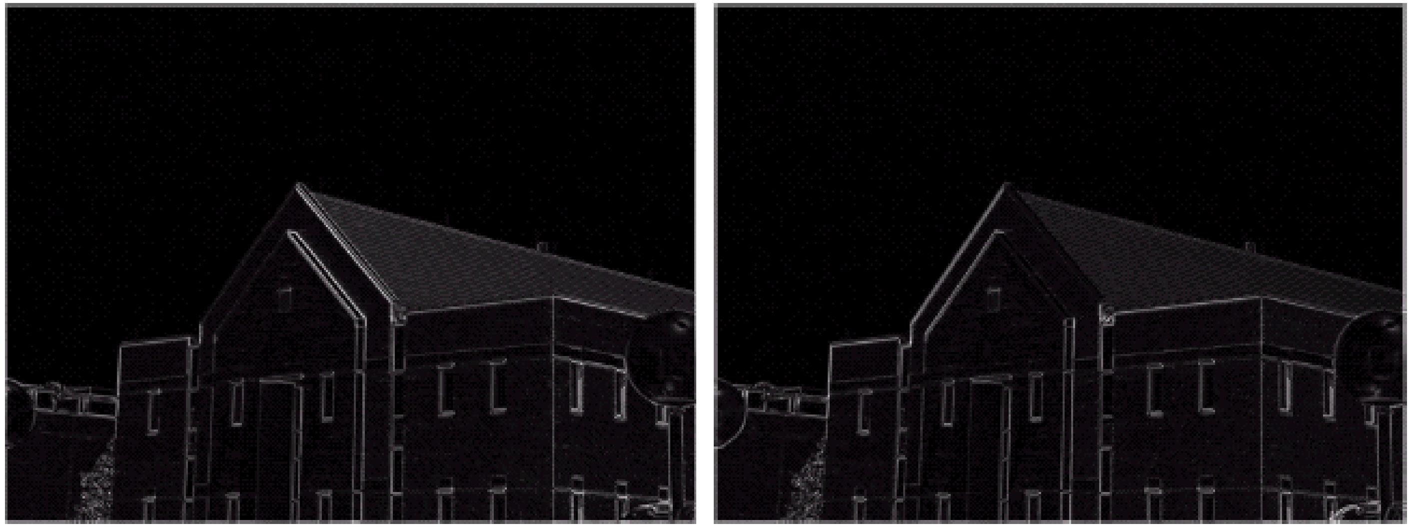


FIGURE 10.11 Same sequence as in Fig. 10.10, but with the original image smoothed with a 5 × 5 averaging filter.

Detection of Discontinuities: Gradient Operators: Example



a b

FIGURE 10.12 Diagonal edge detection. (a) Result of using the mask in Fig. 10.9(c). (b) Result of using the mask in Fig. 10.9(d). The input in both cases was Fig. 10.11(a).

0	1	2
−1	0	1
−2	−1	0

−2	−1	0
−1	0	1
0	1	2

Detection of Discontinuities: Gradient Operators

- Second-order derivatives: (The Laplacian)
  - The Laplacian of an 2D function f(x,y) is defined as

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

  - Two forms in practice:

FIGURE 10.13 Laplacian masks used to implement Eqs. (10.1-14) and (10.1-15), respectively.

0	−1	0
−1	4	−1
0	−1	0

−1	−1	−1
−1	8	−1
−1	−1	−1

Detection of Discontinuities: Gradient Operators

- Consider the function:

A Gaussian function

$$h(r) = -e^{-\frac{r^2}{2\sigma^2}} \quad \text{where} \quad r^2 = x^2 + y^2$$

$$\text{and} \quad \sigma : \text{the standard deviation}$$

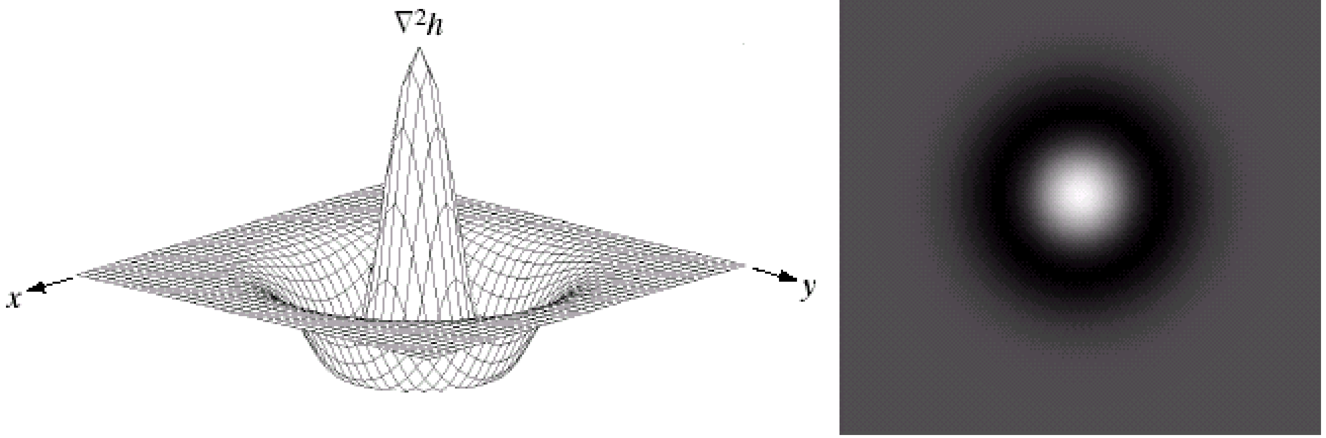
- The Laplacian of h is

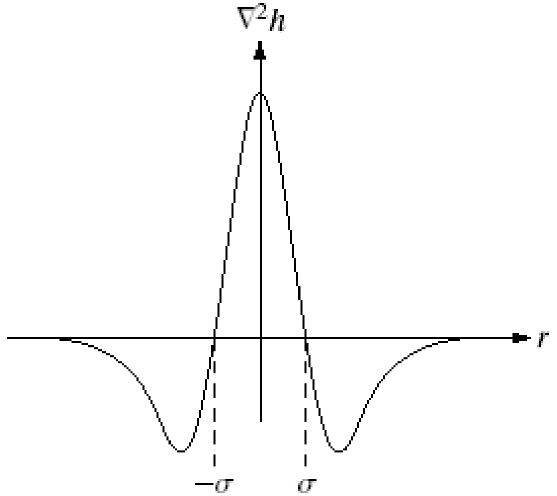
$$\nabla^2 h(r) = -\left[\frac{r^2 - \sigma^2}{\sigma^4}\right] e^{-\frac{r^2}{2\sigma^2}}$$

The Laplacian of a Gaussian (LoG)

- The Laplacian of a Gaussian sometimes is called the Mexican hat function. It also can be computed by smoothing the image with the Gaussian smoothing mask, followed by application of the Laplacian mask.

Detection of Discontinuities: Gradient Operators



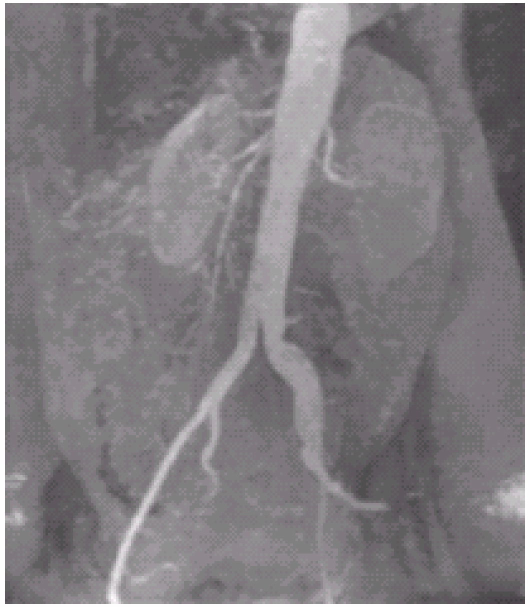


0	0	−1	0	0
0	−1	−2	−1	0
−1	−2	16	−2	−1
0	−1	−2	−1	0
0	0	−1	0	0

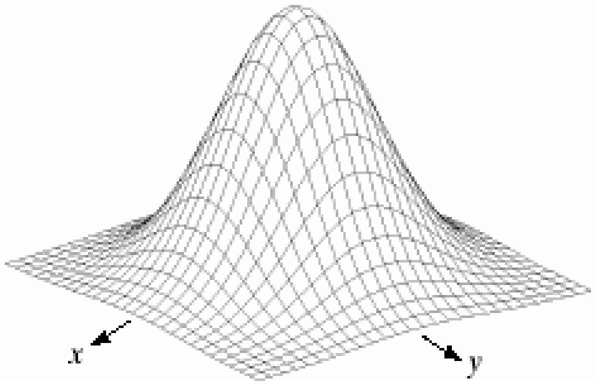
a b
c d

FIGURE 10.14 Laplacian of a Gaussian (LoG). (a) 3-D plot. (b) Image (black is negative, gray is the zero plane, and white is positive). (c) Cross section showing zero crossings. (d) 5×5 mask approximation to the shape of (a).

Detection of Discontinuities: Gradient Operators: Example



Sobel gradient

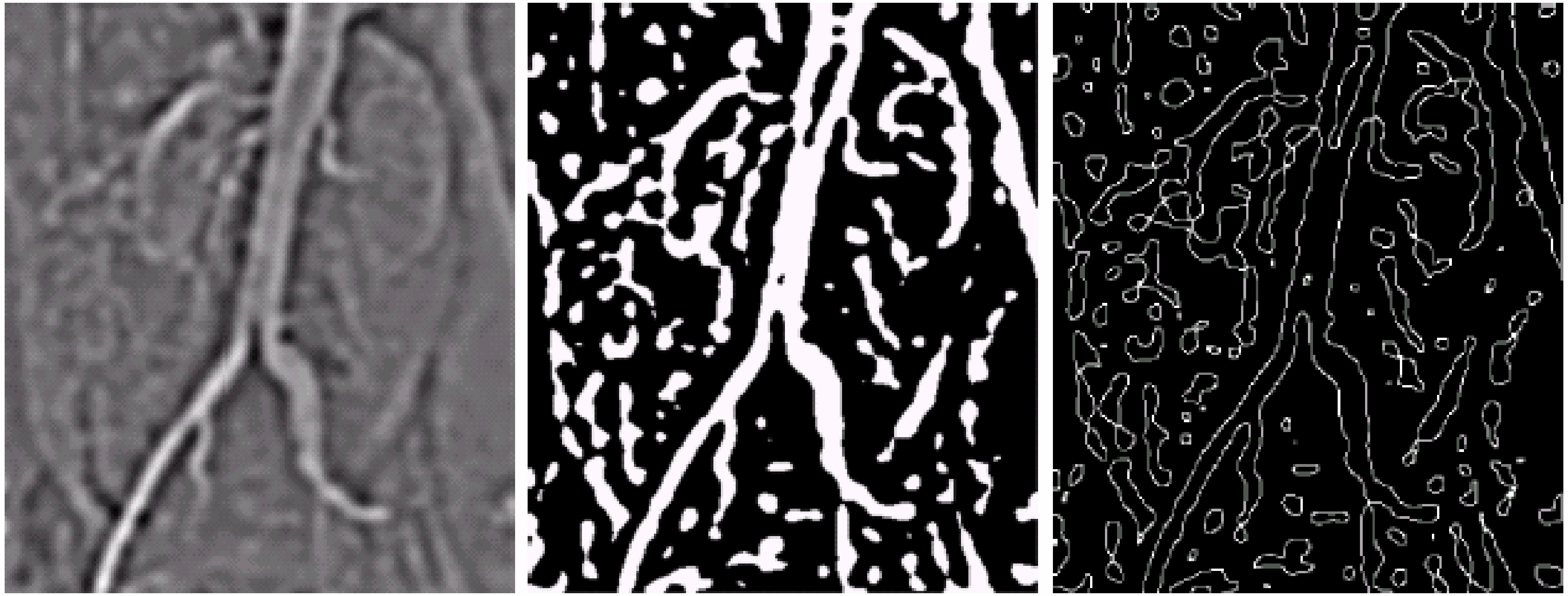


Gaussian smooth function

–1	–1	–1
–1	8	–1
–1	–1	–1

Laplacian mask

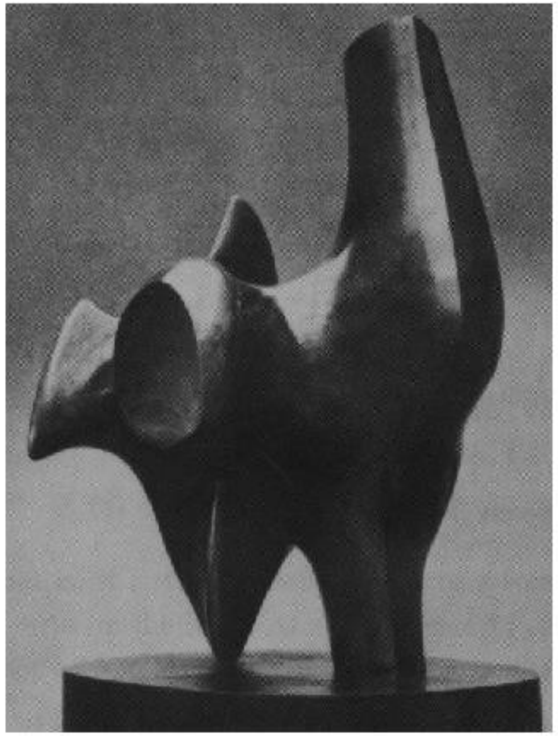
Detection of Discontinuities Gradient Operators: Example

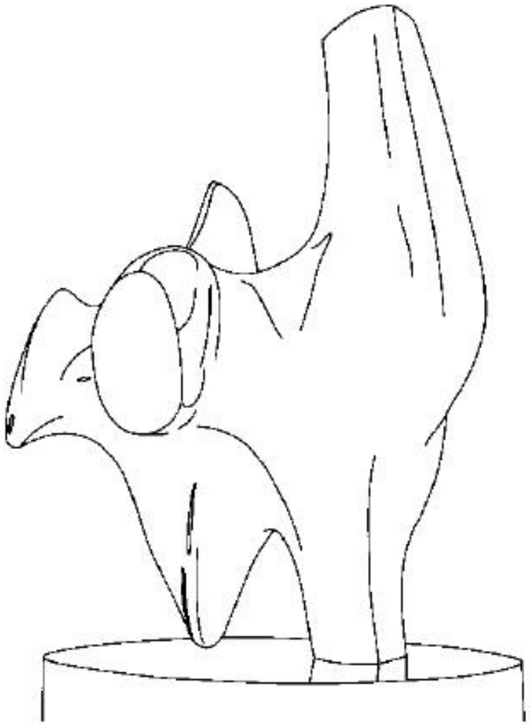


a b
c d
e f g

FIGURE 10.15 (a) Original image. (b) Sobel gradient (shown for comparison). (c) Spatial Gaussian smoothing function. (d) Laplacian mask. (e) LoG. (f) Thresholded LoG. (g) Zero crossings. (Original image courtesy of Dr. David R. Pickens, Department of Radiology and Radiological Sciences, Vanderbilt University Medical Center.)

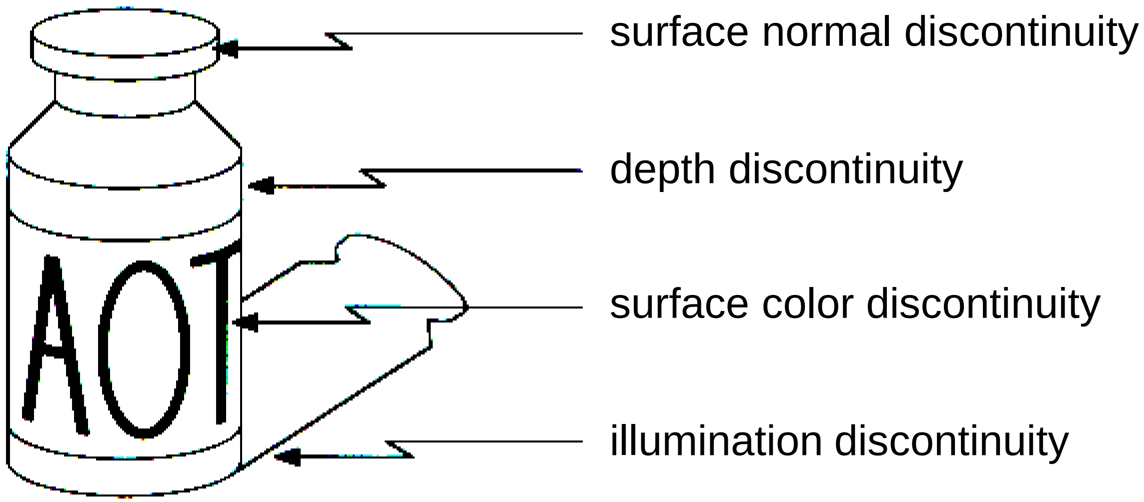
Edge detection





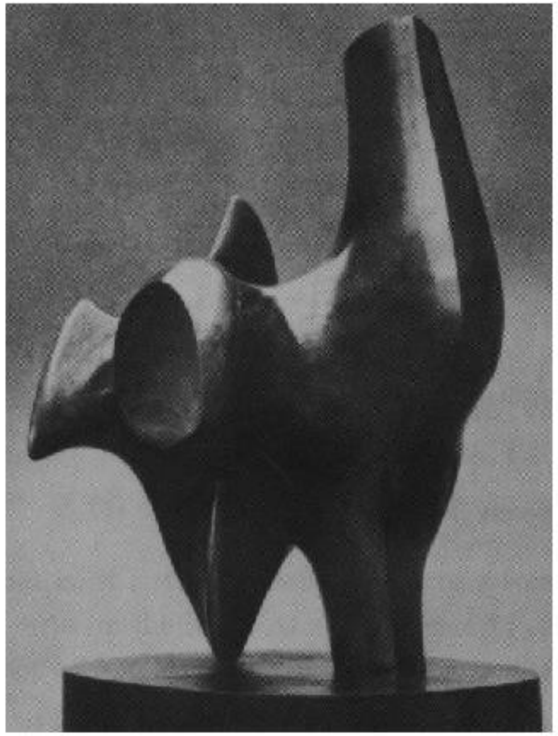
- Convert a 2D image into a set of curves
  - Extracts salient features of the scene
  - More compact than pixels

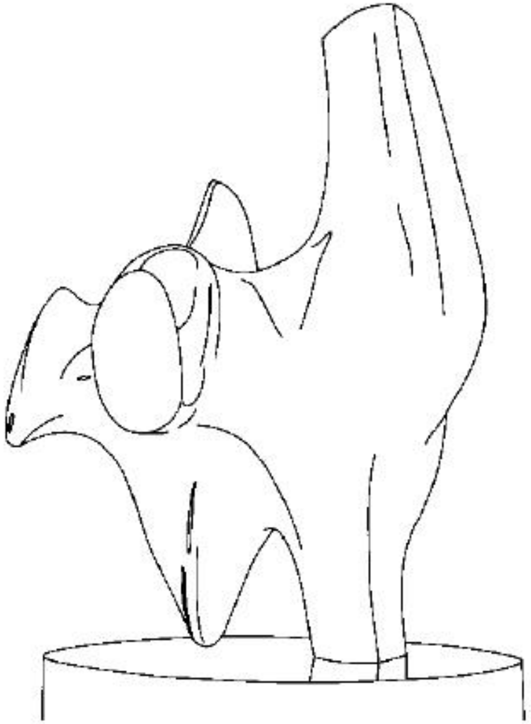
Origin of Edges



- Edges are caused by a variety of factors

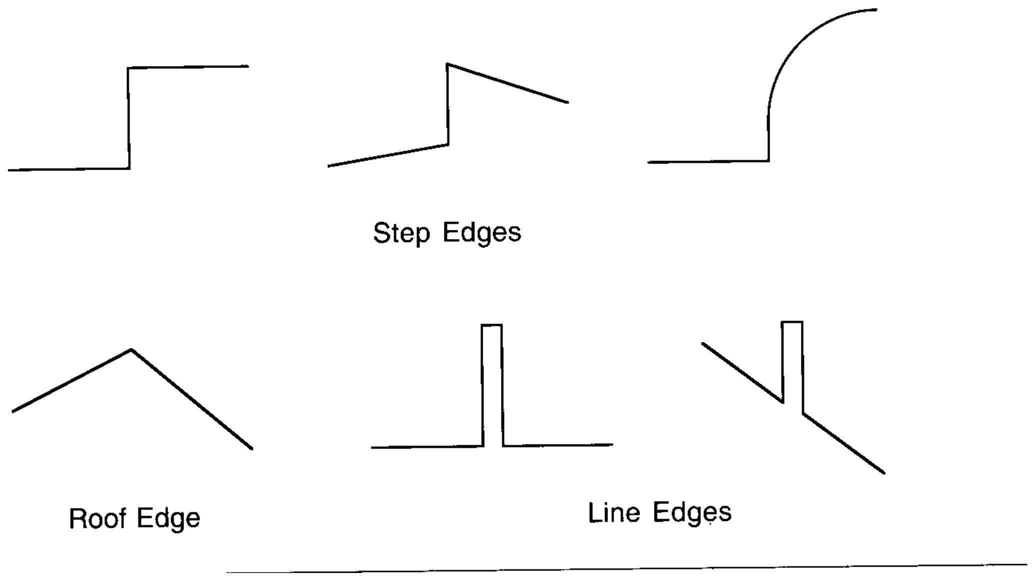
Edge detection





- How can you tell that a pixel is on an edge?

Profiles of image intensity edges



Edge detection

1. Detection of short linear edge segments (edgels)

2. Aggregation of edgels into extended edges

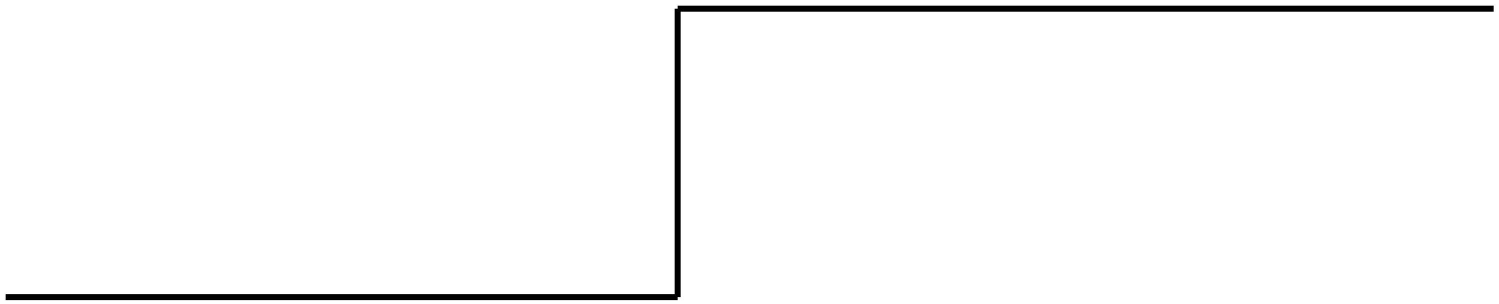
- (maybe parametric description)

Edgel detection

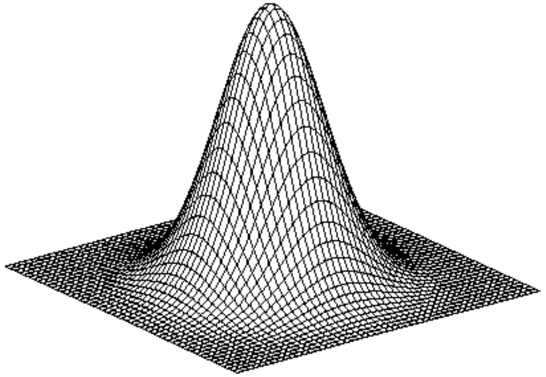
- Difference operators
- Parametric-model matchers

Edge is Where Change Occurs

- Change is measured by derivative in 1D
- Biggest change, derivative has maximum magnitude
- Or 2nd derivative is zero.

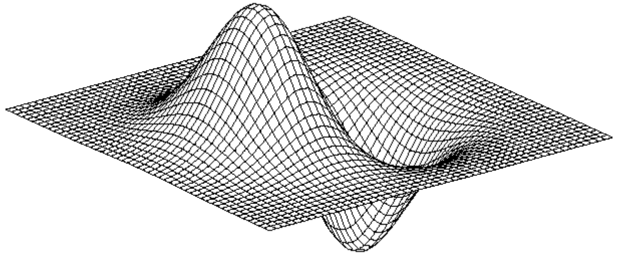


2D edge detection filters



Gaussian

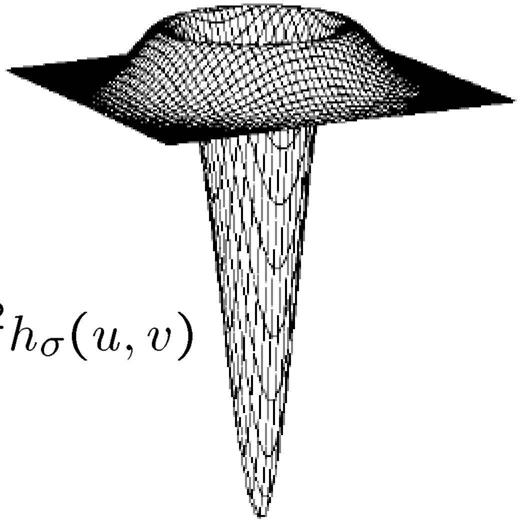
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



derivative of Gaussian

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

Laplacian of Gaussian



$$\nabla^2 h_\sigma(u,v)$$

∇^2 is the **Laplacian** operator:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Optimal Edge Detection: Canny

- Assume:
  - Linear filtering
  - Additive iid Gaussian noise
- Edge detector should have:
  - Good Detection. Filter responds to edge, not noise.
  - Good Localization: detected edge near true edge.
  - Single Response: one per edge.

Optimal Edge Detection: Canny (continued)

- Optimal Detector is approximately Derivative of Gaussian.
- Detection/Localization trade-off
  - More smoothing improves detection
  - And hurts localization.
- This is what you might guess from (detect change) + (remove noise)

The Canny edge detector

- original image (Lena)

The Canny edge detector

norm of the gradient

The Canny edge detector

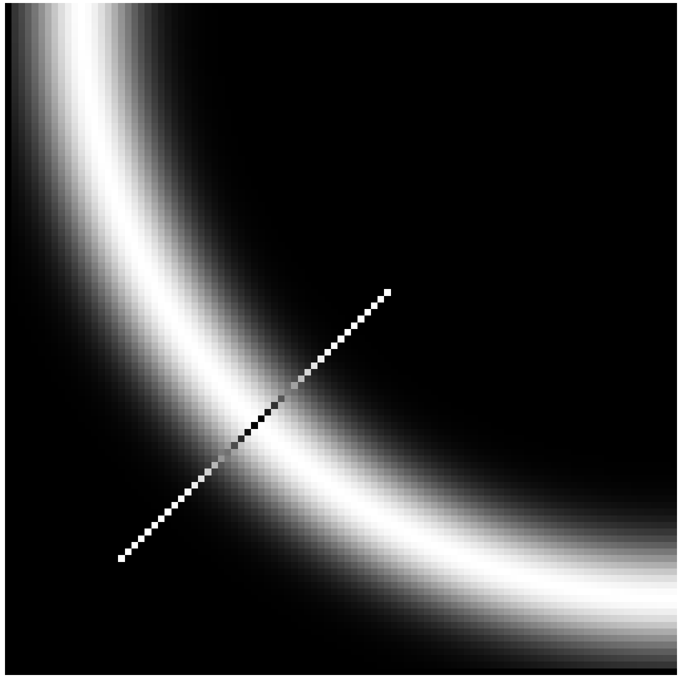
thresholding

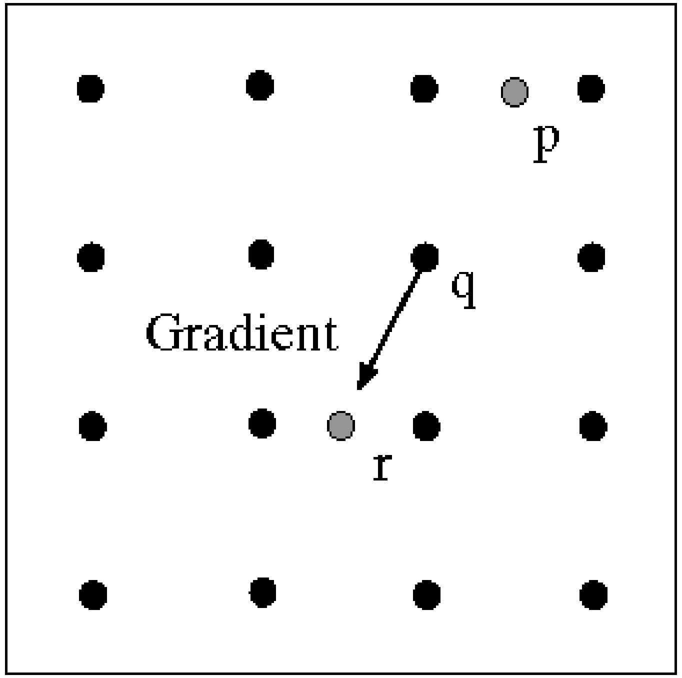
The Canny edge detector

thinning

(non-maximum suppression)

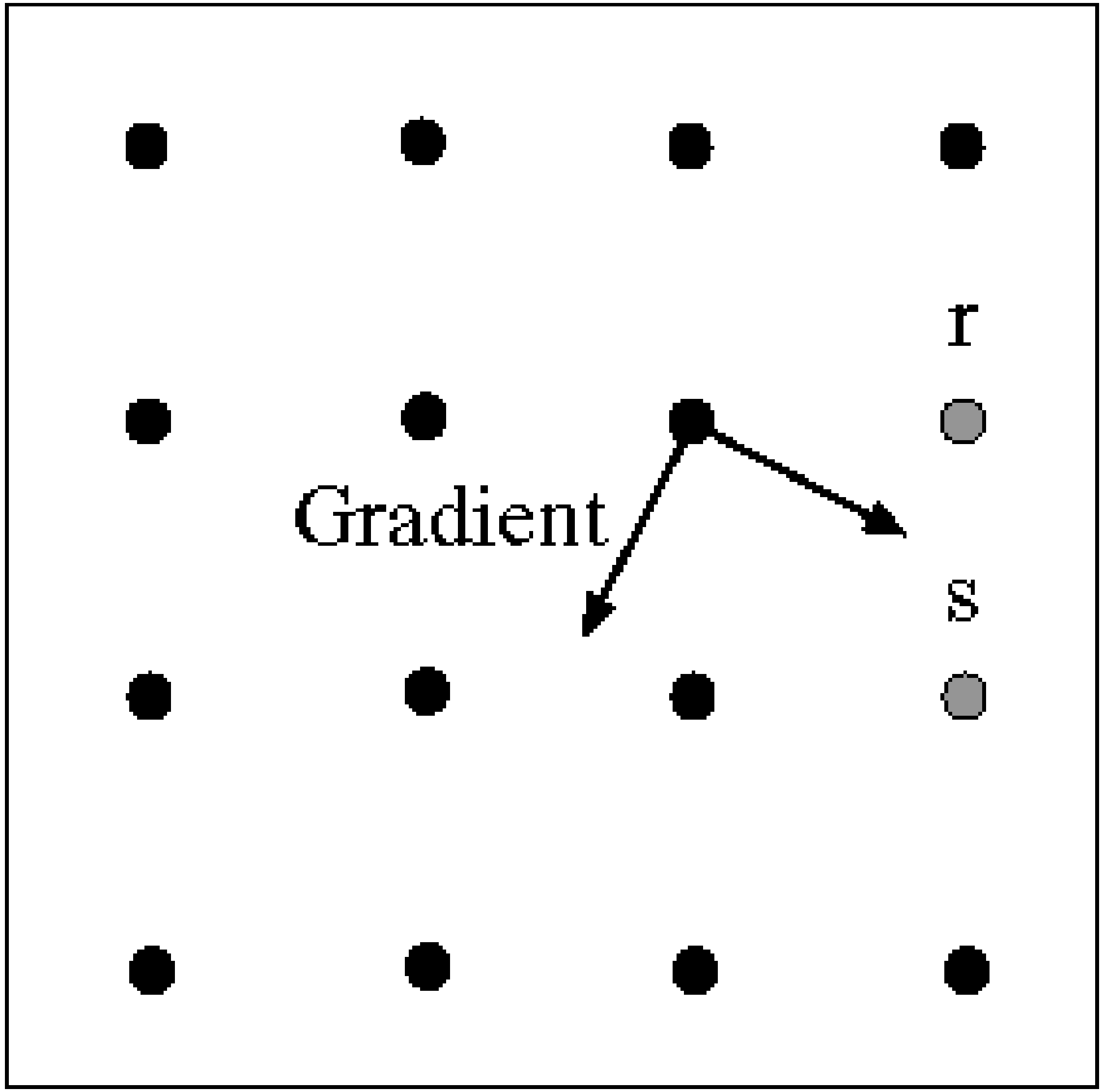
Non-maximum suppression





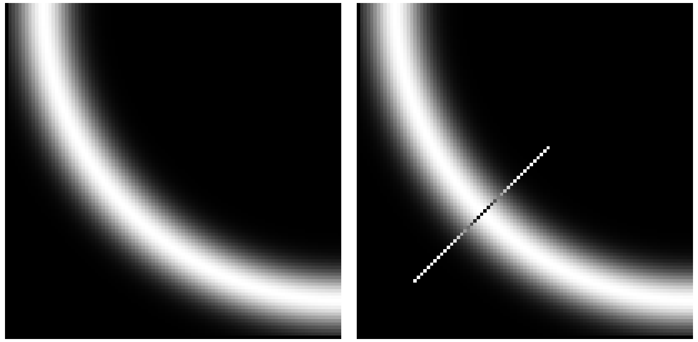
- Check if pixel is local maximum along gradient direction
  - requires checking interpolated pixels p and r

Non-maximum suppression



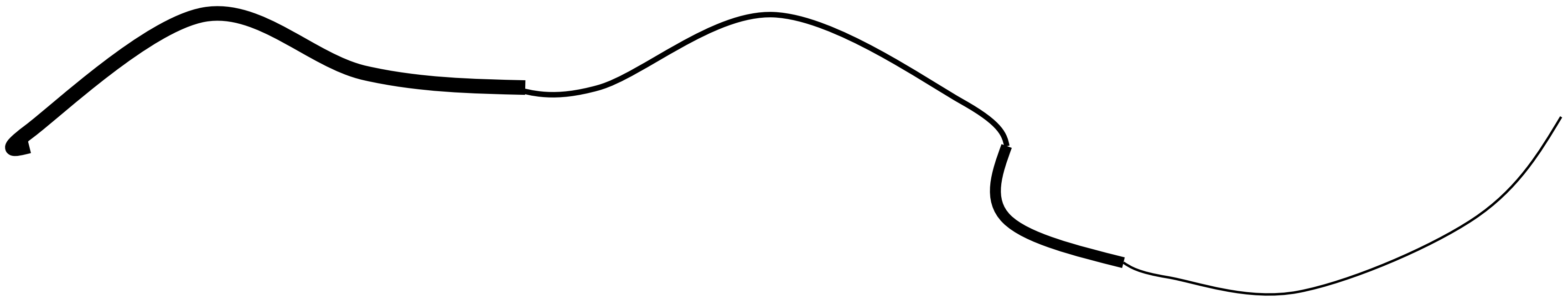
Predicting
the next
edge point

Assume the marked point is an edge point. Then we construct the tangent to the edge curve (which is normal to the gradient at that point) and use this to predict the next points (here either r or s).



Hysteresis

- Check that maximum value of gradient value is sufficiently large
  - drop-outs? use **hysteresis**
    - use a high threshold to start edge curves and a low threshold to continue them.



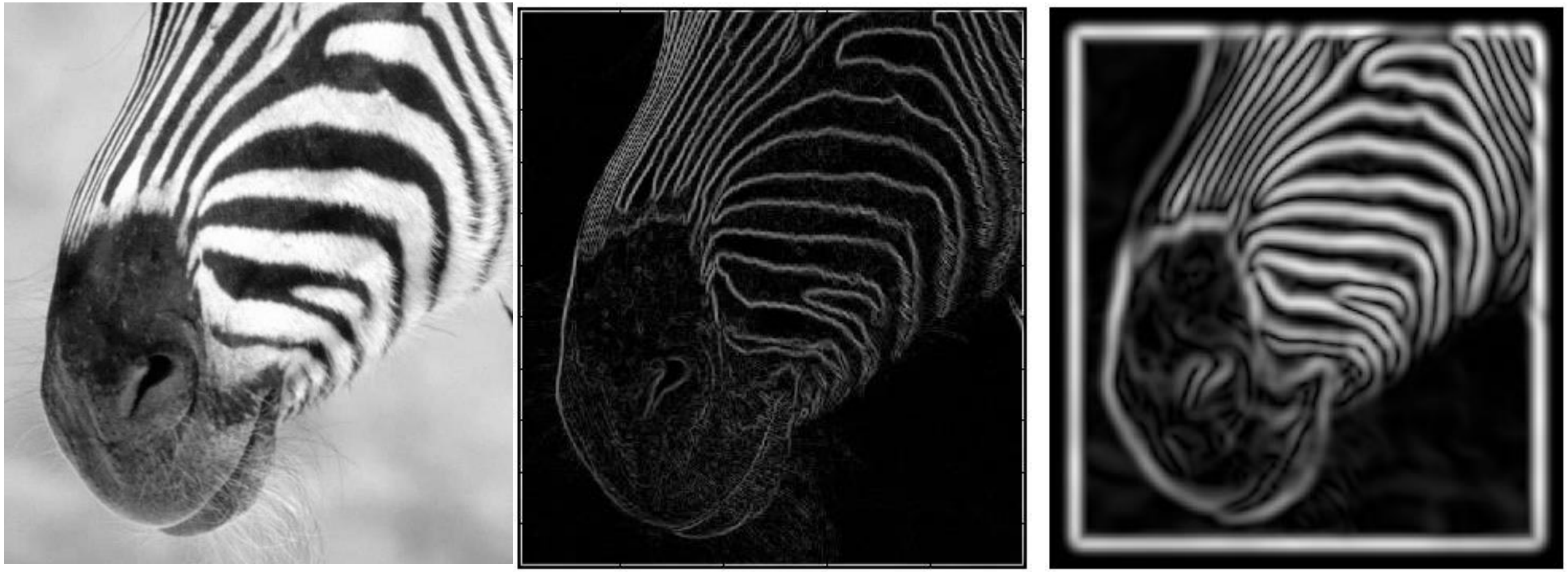
Effect of σ (Gaussian kernel size)

original | Canny with $\sigma = 1$ | Canny with $\sigma = 2$

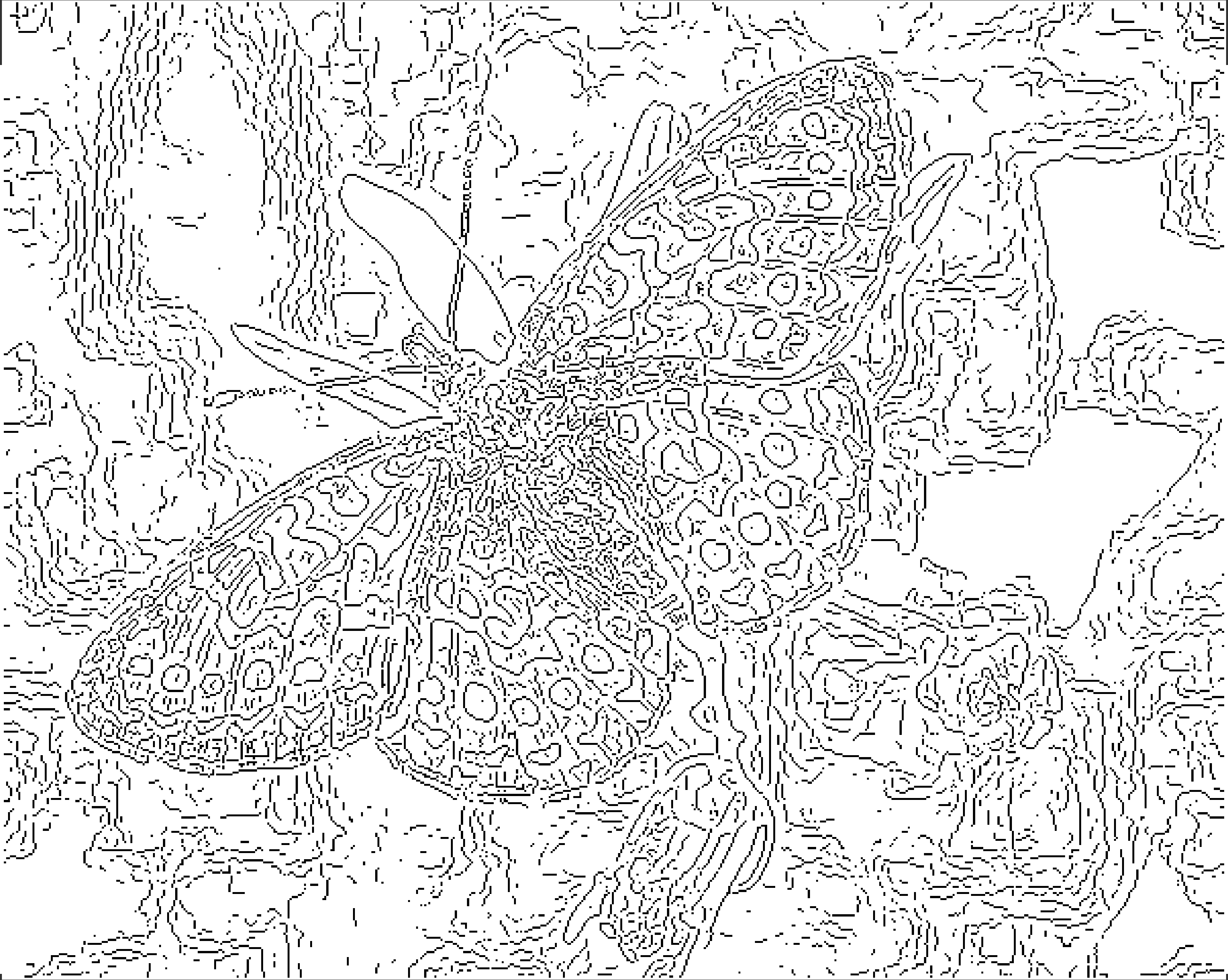
The choice of σ depends on desired behavior

- large σ detects large scale edges
- small σ detects fine features

Scale



- Smoothing
- Eliminates noise edges.
- Makes edges smoother.
- Removes fine detail.

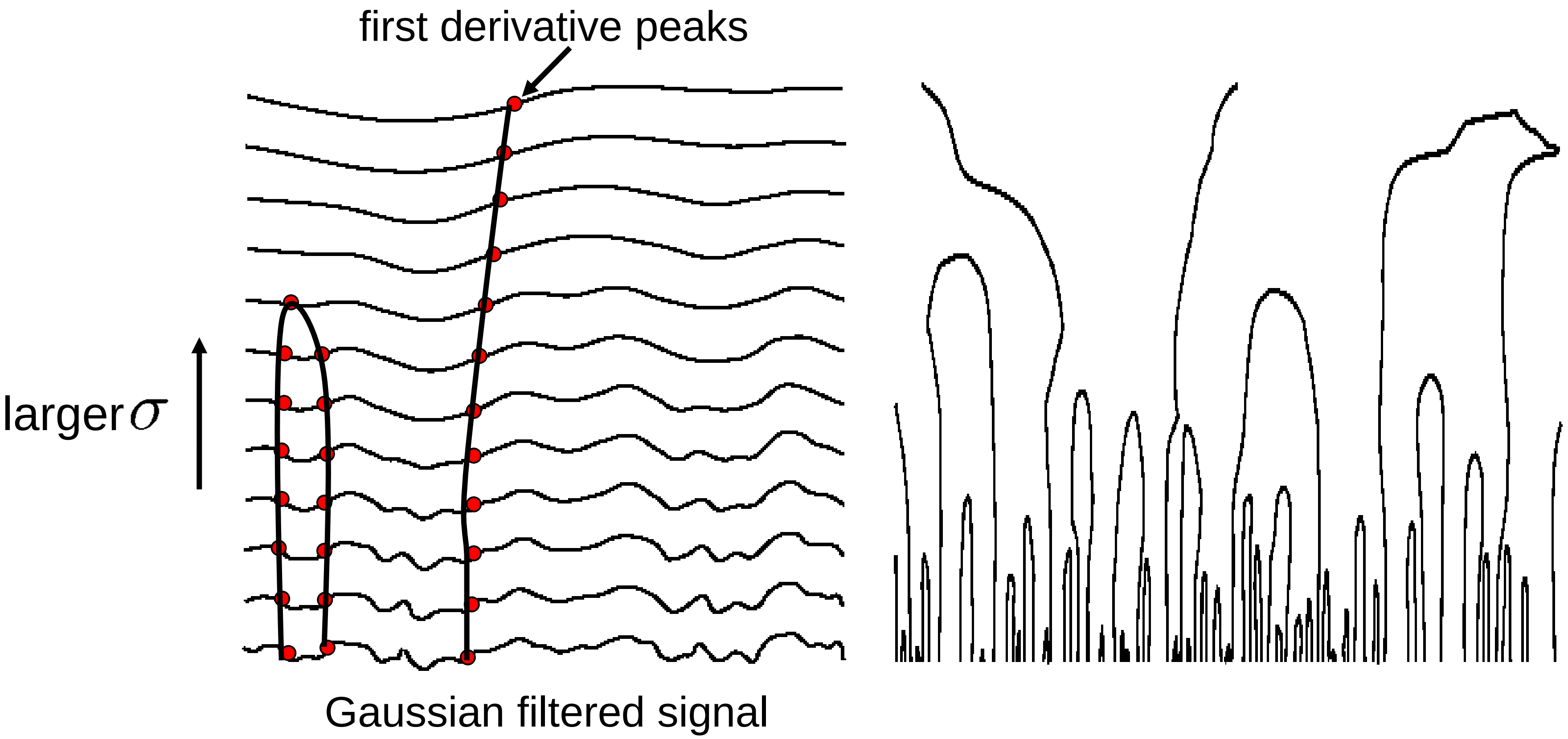


fine scale
high
threshold

coarse
scale,
high
threshold

coarse
scale
low
threshold

Scale space (Witkin 83)



- Properties of scale space (w/ Gaussian smoothing)
  - edge position may shift with increasing scale (σ)
  - two edges may merge with increasing scale
  - an edge may ***not*** split into two with increasing scale

Edge detection by subtraction

original

Edge detection by subtraction

smoothed (5x5 Gaussian)

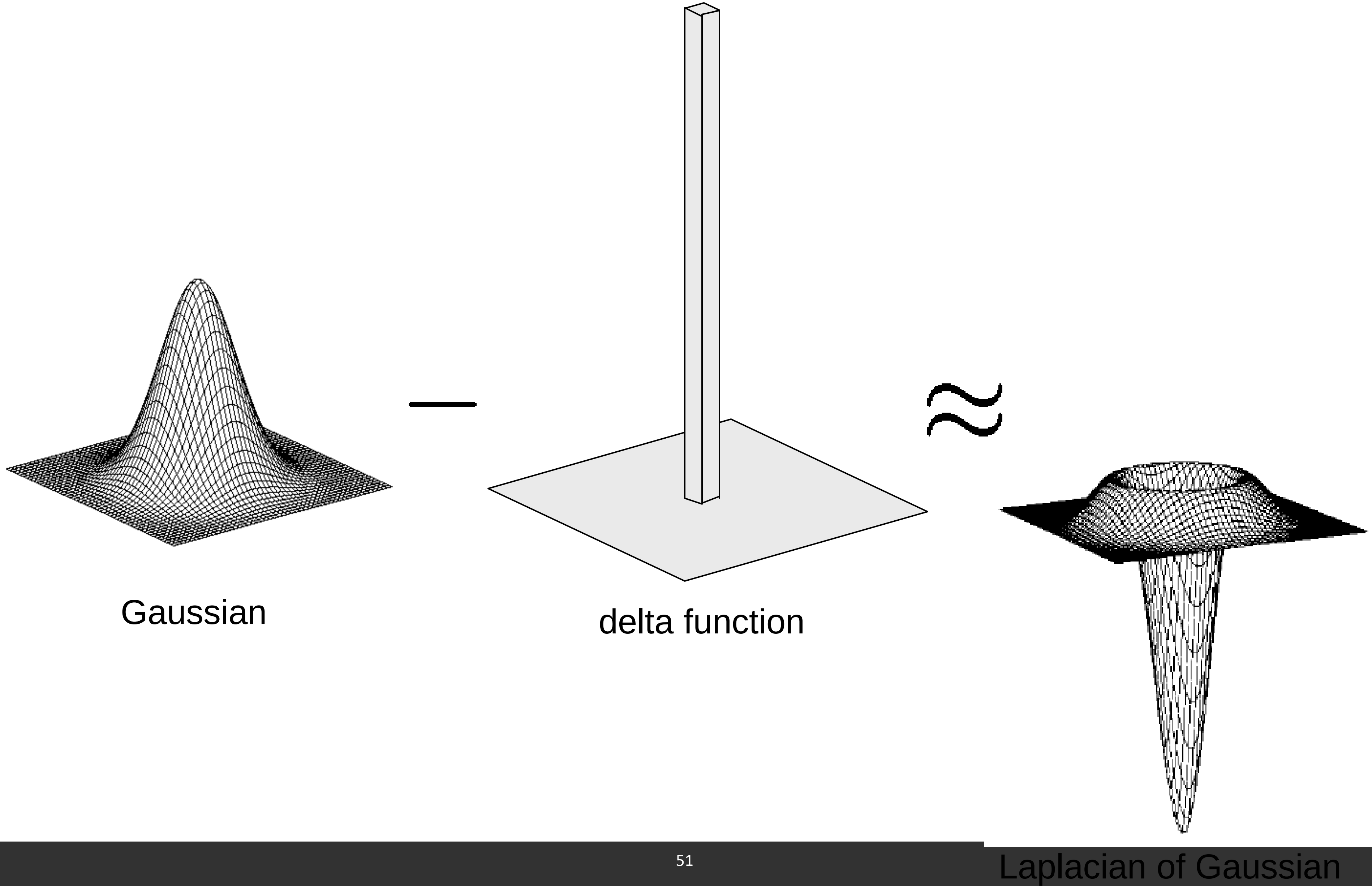
Edge detection by subtraction

smoothed – original
(scaled by 4, offset +128)

Why does this work?

filter demo

Gaussian - image filter



An edge is not a line...

How can we detect ***lines*** ?

Edge Linking and Boundary Detection: Local Processing

- Two properties of edge points are useful for edge linking:
  - the strength (or magnitude) of the detected edge points
  - their directions (determined from gradient directions)
- This is usually done in local neighborhoods.
- Adjacent edge points with similar magnitude and direction are linked.
- For example, an edge pixel with coordinates (x0,y0) in a predefined neighborhood of (x,y) is similar to the pixel at (x,y) if

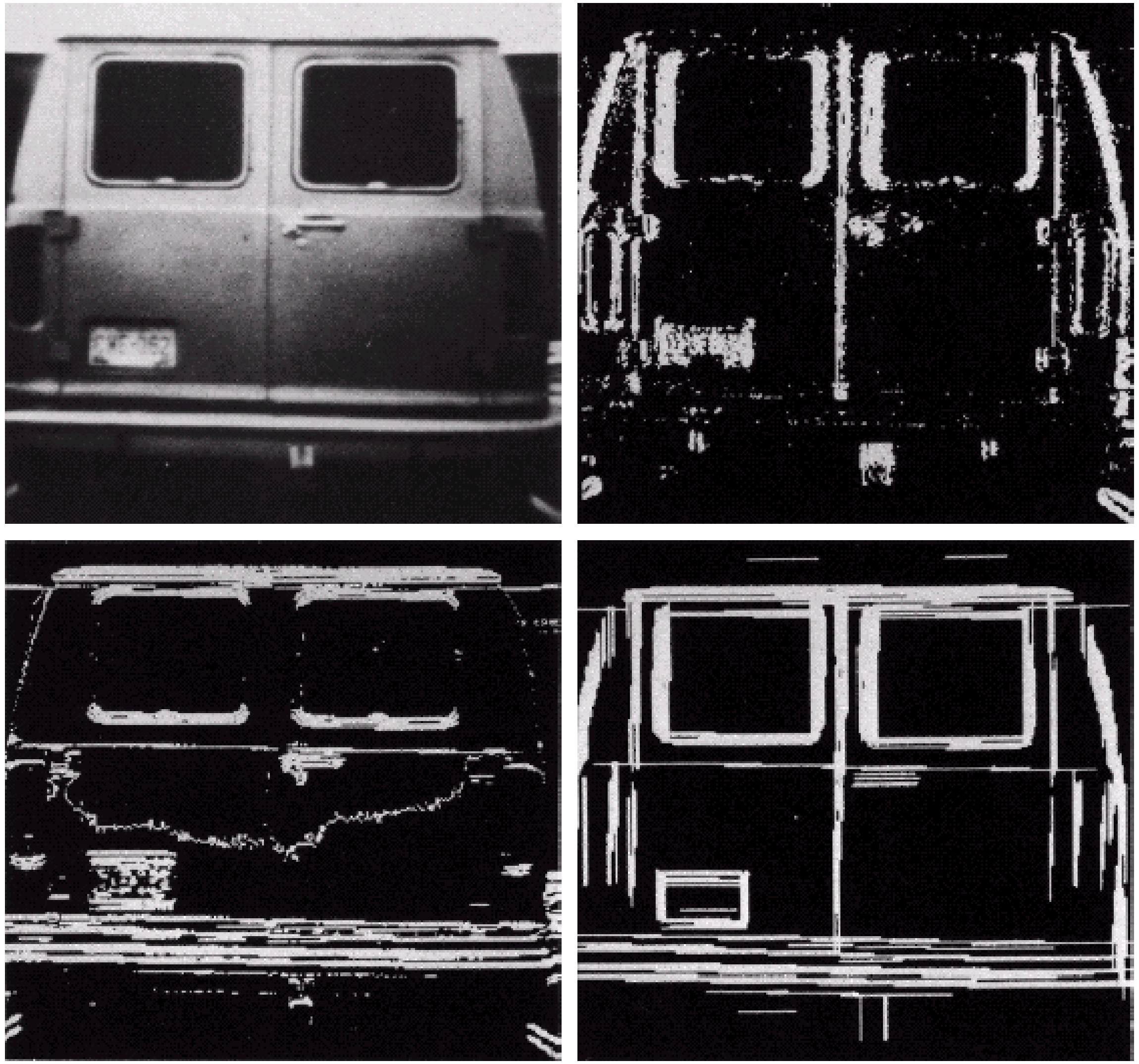
$$|\nabla f(x,y) - \nabla(x_0,y_0)| \leq E, \qquad E\text{: a nonnegative threshold}$$

$$|\alpha(x,y) - \alpha(x_0,y_0)| < A, \qquad A\text{: a nonegative angle threshold}$$

Edge Linking and Boundary Detection: Local Processing: Example

a b
c d

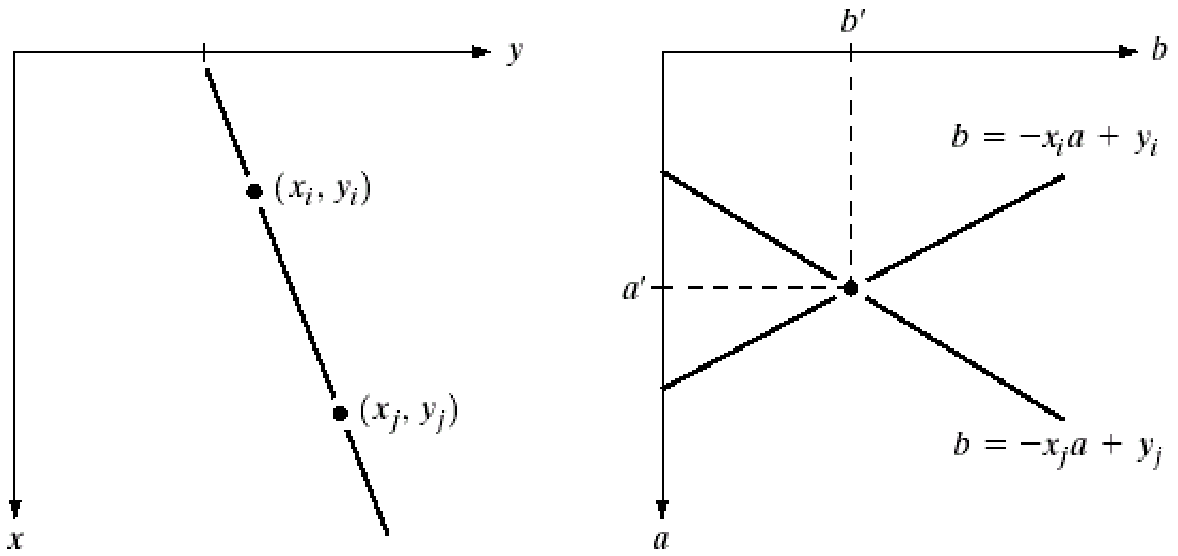
FIGURE 10.16
(a) Input image.
(b) G_y component of the gradient.
(c) G_x component of the gradient.
(d) Result of edge linking. (Courtesy of Perceptics Corporation.)



In this example, we can find the license plate candidate after edge linking process.

Edge Linking and Boundary Detection : Hough Transform

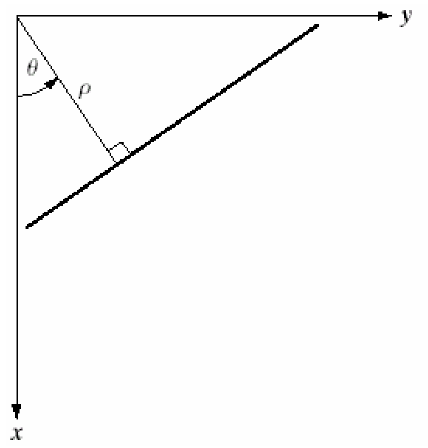
- Hough transform: a way of finding edge points in an image that lie along a straight line. $y_i = ax_i + b$
- Example: xy-plane v.s. ab-plane (parameter space)



a b

FIGURE 10.17
(a) *xy*-plane.
(b) Parameter space.

- The Hough transform consists of finding all pairs of values of θ and ρ which satisfy the equations that pass through (x,y).
- These are accumulated in what is basically a 2-dimensional histogram.
- When plotted these pairs of θ and ρ will look like a sine wave. The process is repeated for all appropriate (x,y) locations.



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

Edge Linking and Boundary Detection: Hough Transform Example

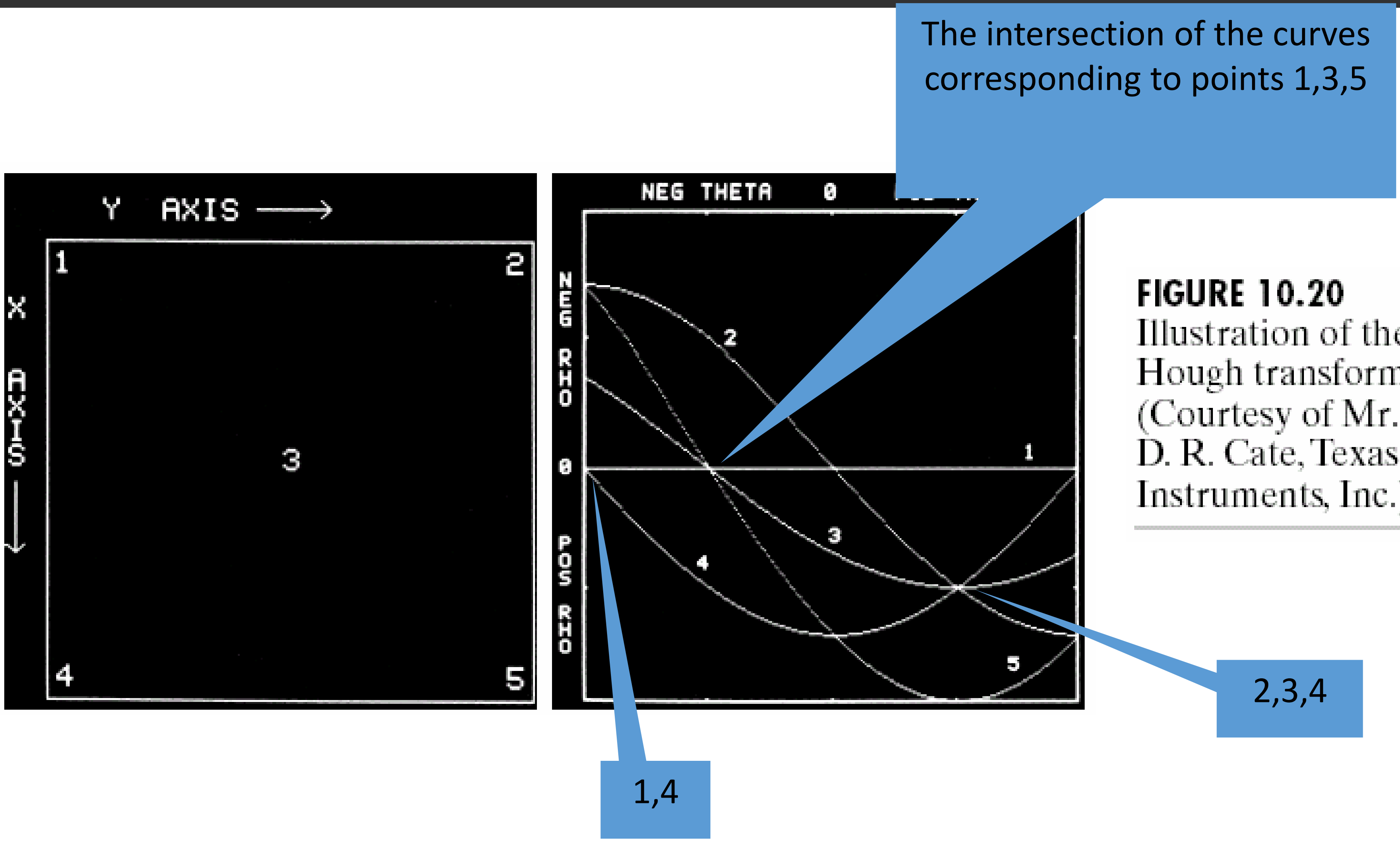
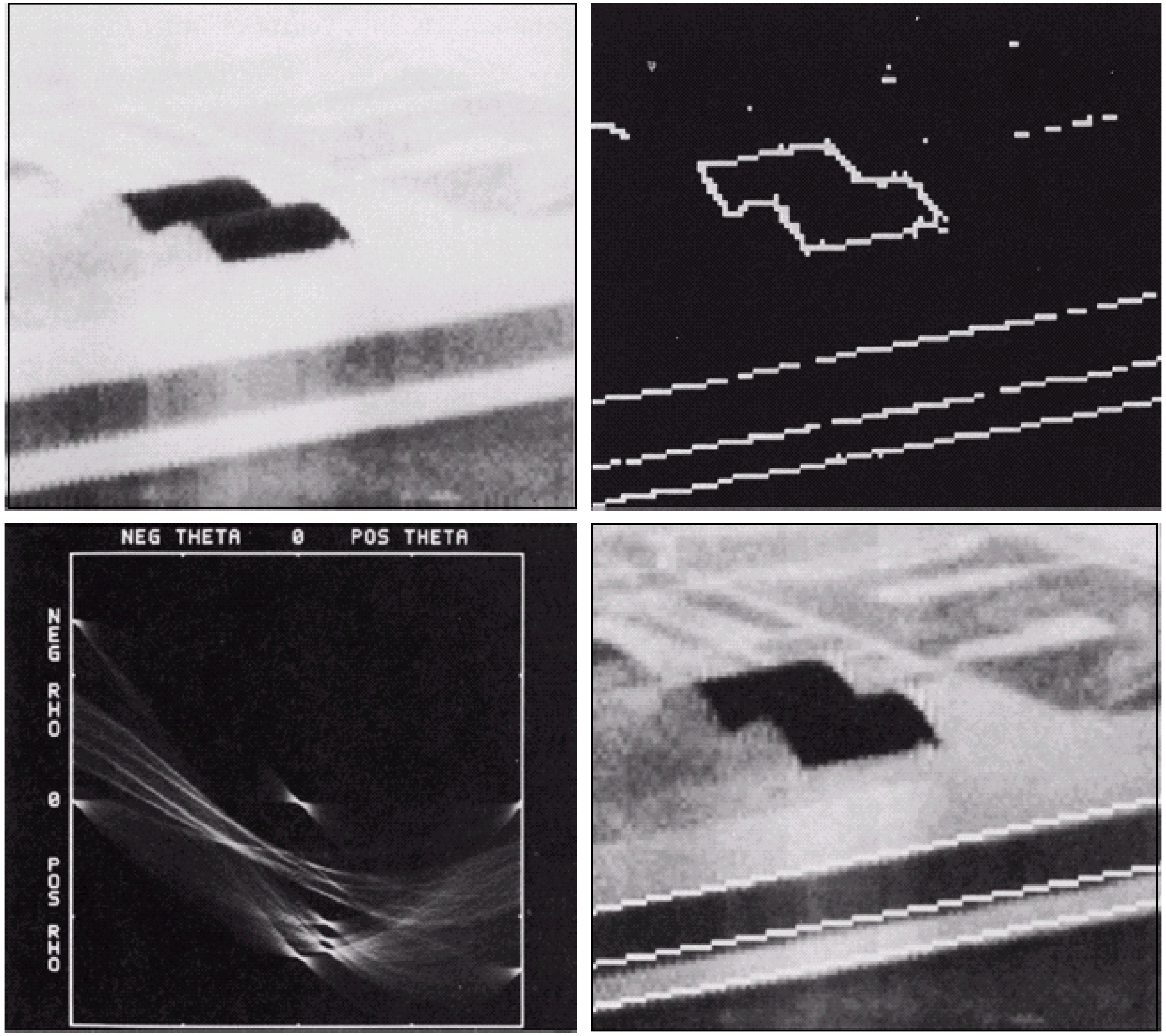


FIGURE 10.20 Illustration of the Hough transform. (Courtesy of Mr. D. R. Cate, Texas Instruments, Inc.)

Edge Linking and Boundary Detection: Hough Transform Example



a b
c d

FIGURE 10.21
(a) Infrared image.
(b) Thresholded gradient image.
(c) Hough transform.
(d) Linked pixels.
(Courtesy of Mr. D. R. Cate, Texas Instruments, Inc.)

Connected Component Labeling

- Ability to assign different labels to various disjoint component of an image is called connected comp onent labeling.

- This labeling is a fundamental step in

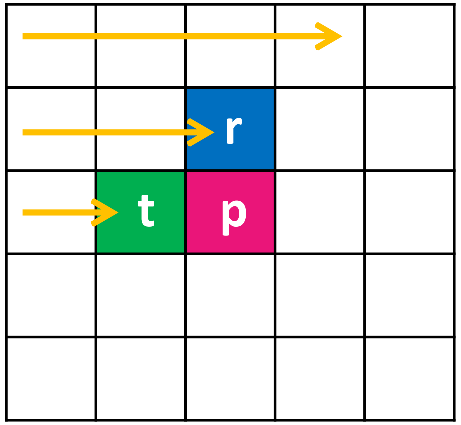
  a) Shape

  b) Area

  c) Boundary

Basic Scanning Method

- Scan the image from left to right and top to bottom.
- Assume 4-adjacency.
- Let p be a pixel at any step in the scanning process.
- Before p the pixel r and t are scanned.



Labeling Algorithm

- This algorithm makes two passes over the image:
  - The first pass to assign temporary labels and record equivalence classes.
  - The second pass to replace each temporary label by the smallest label of its equivalence class.

Steps in First Pass

- Conditions to check:
  - Does the pixel to the left (West) have the same value as the current pixel?
    - Yes – We are in the same region. Assign the same label to the current pixel
    - No – Check next condition
  - Do both pixels to the North and West of the current pixel have the same value as the current pixel but not the same label?
    - Yes –Assign the current pixel the minimum of the North and West labels, and record their equivalence relationship
    - No – Check next condition

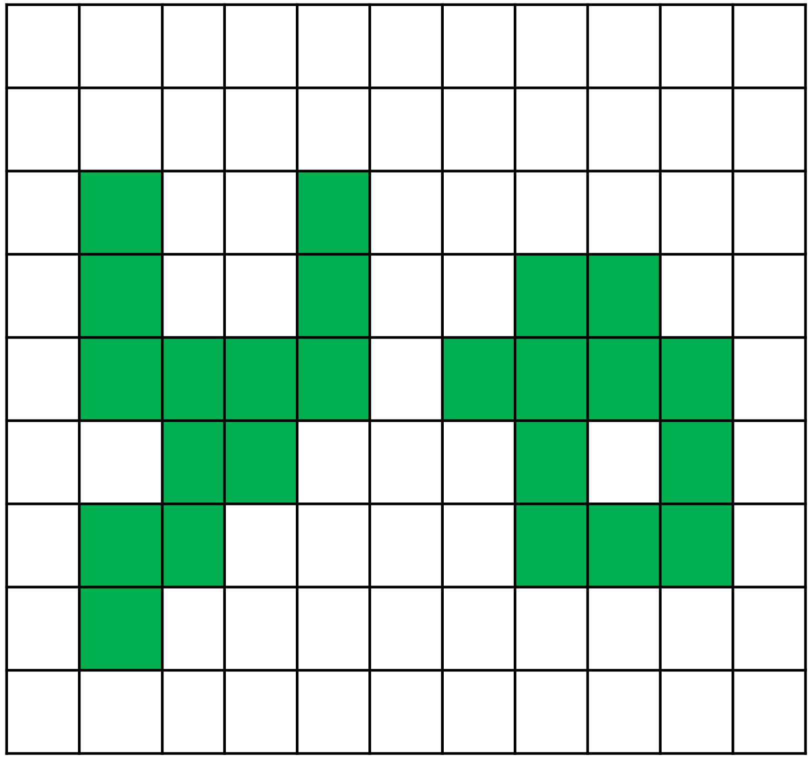
Steps in First Pass..

- Does the pixel to the left (West) have a different value and the one to the North the same value as the current pixel?
  - Yes – Assign the label of the North pixel to the current pixel
  - No – Check next condition
- Do the pixel's North and West neighbors have different pixel values than current pixel?
  - Yes – Create a new label id and assign it to the current pixel

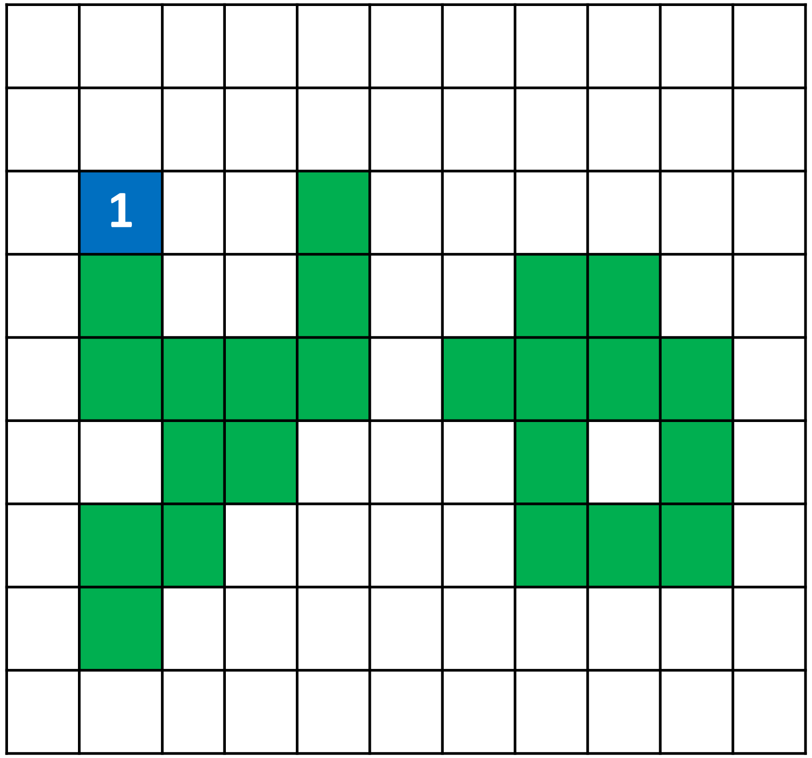
Steps in Second Pass

- In the First pass we record some equivalence relationships.
- In Second Pass:
  - Process Equivalence pairs to form equivalent classes.
  - Re-label the element with the label assigned to its equivalent classes.

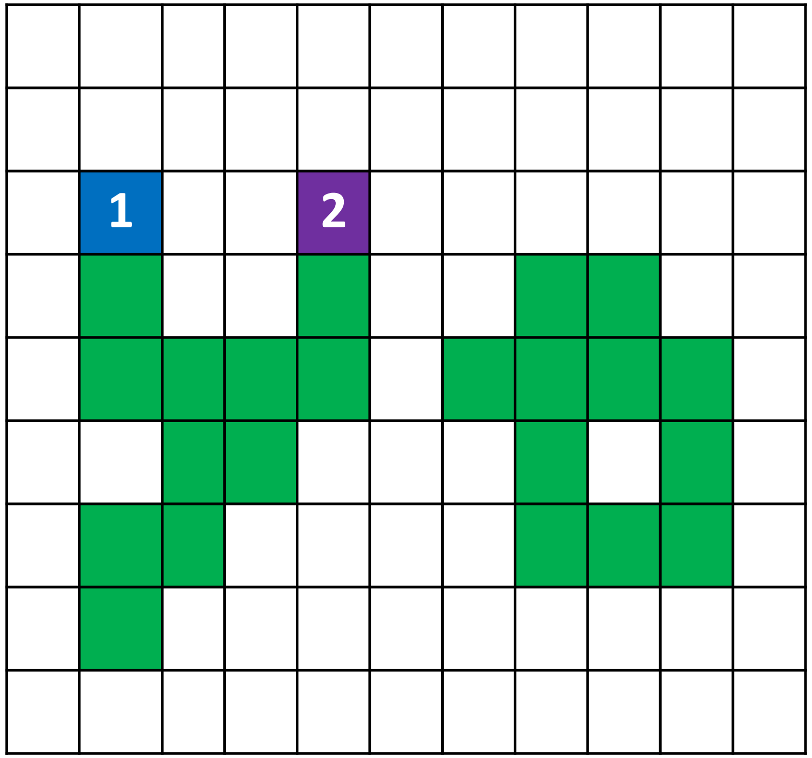
Connected Component Labeling (example)



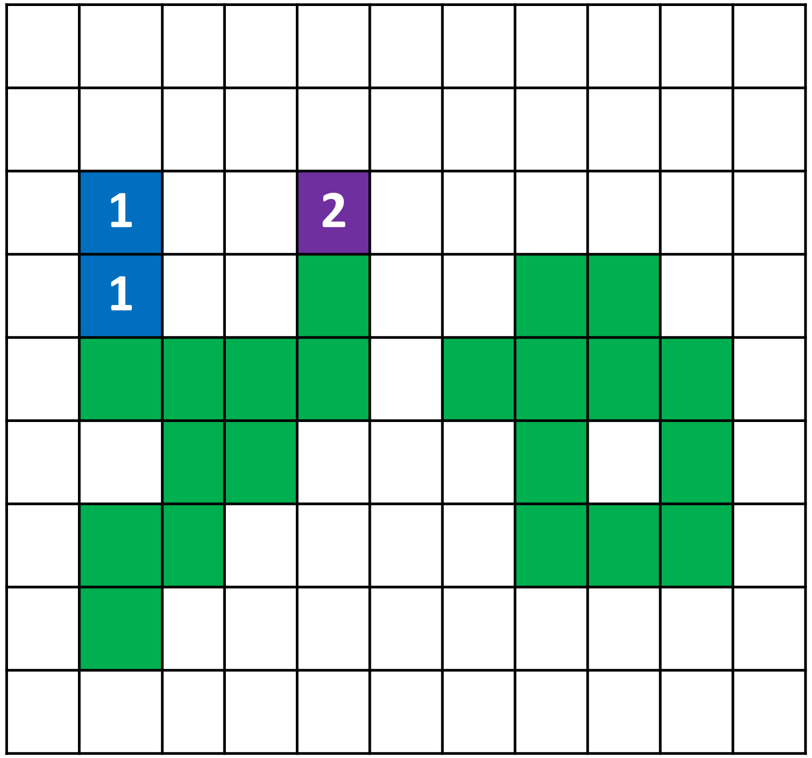
Connected Component Labeling (example)



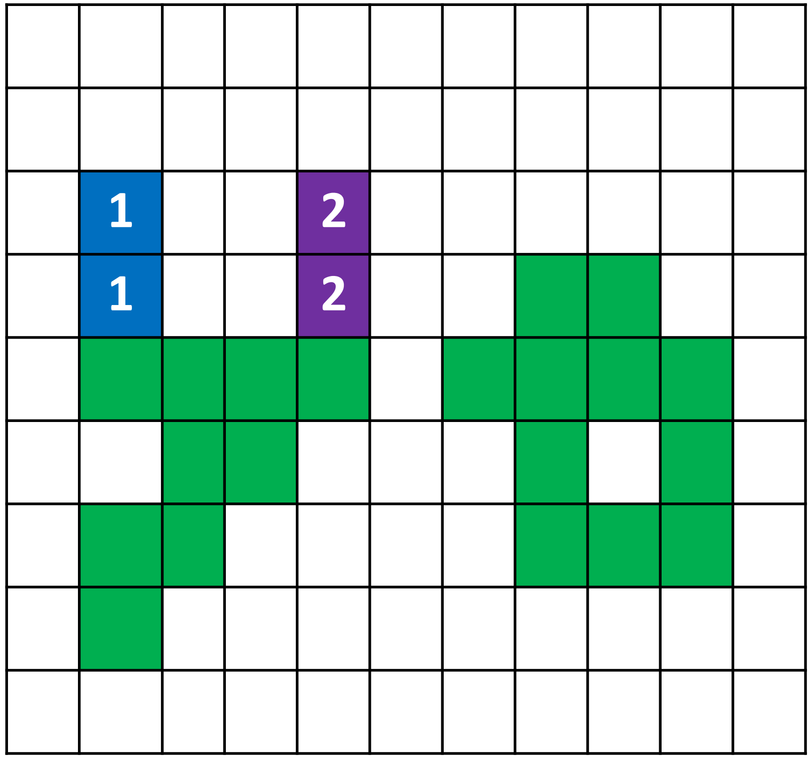
Connected Component Labeling (example)



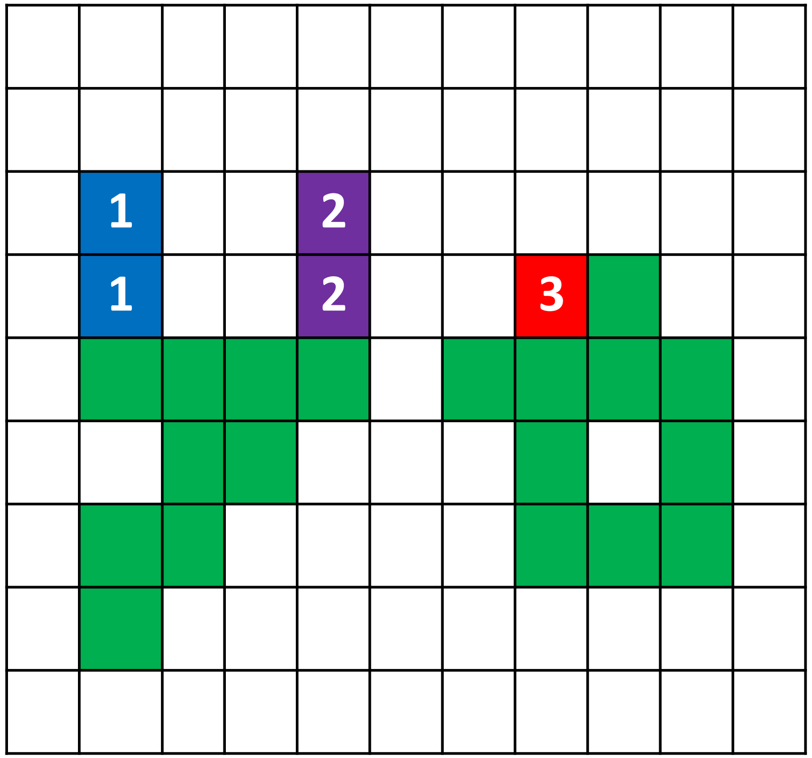
Connected Component Labeling (example)



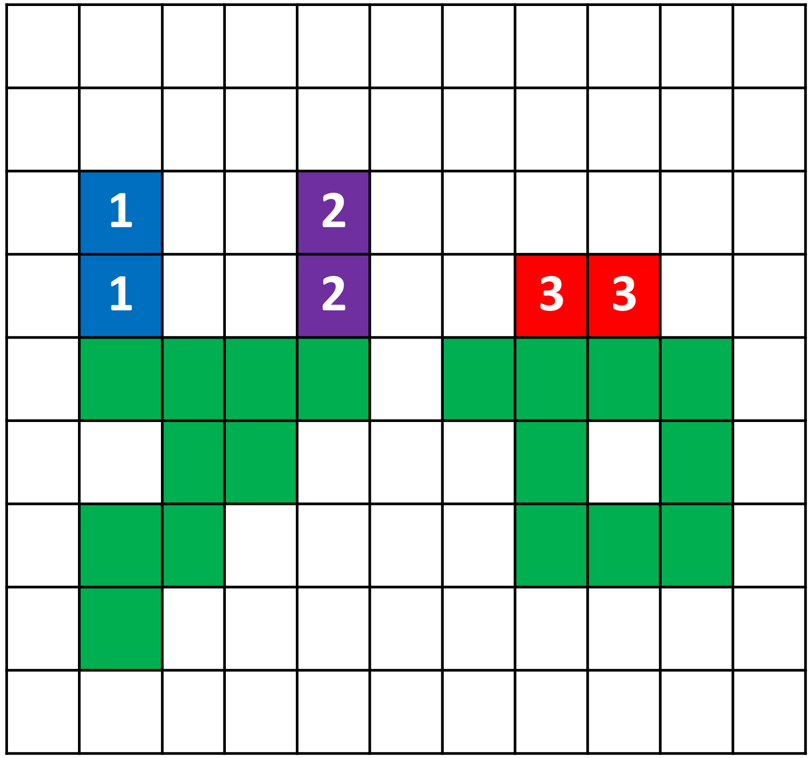
Connected Component Labeling (example)



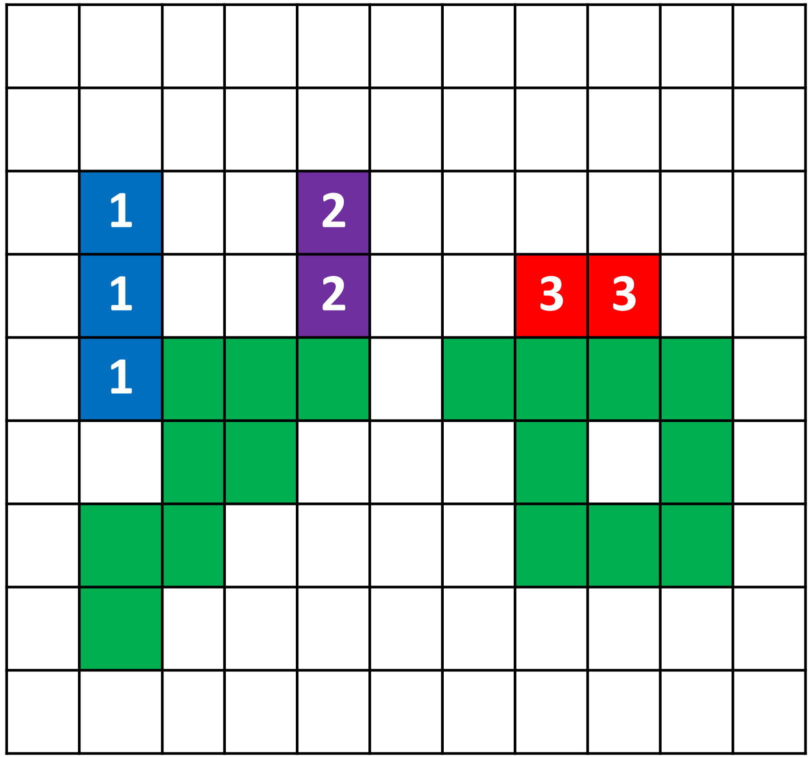
Connected Component Labeling (example)



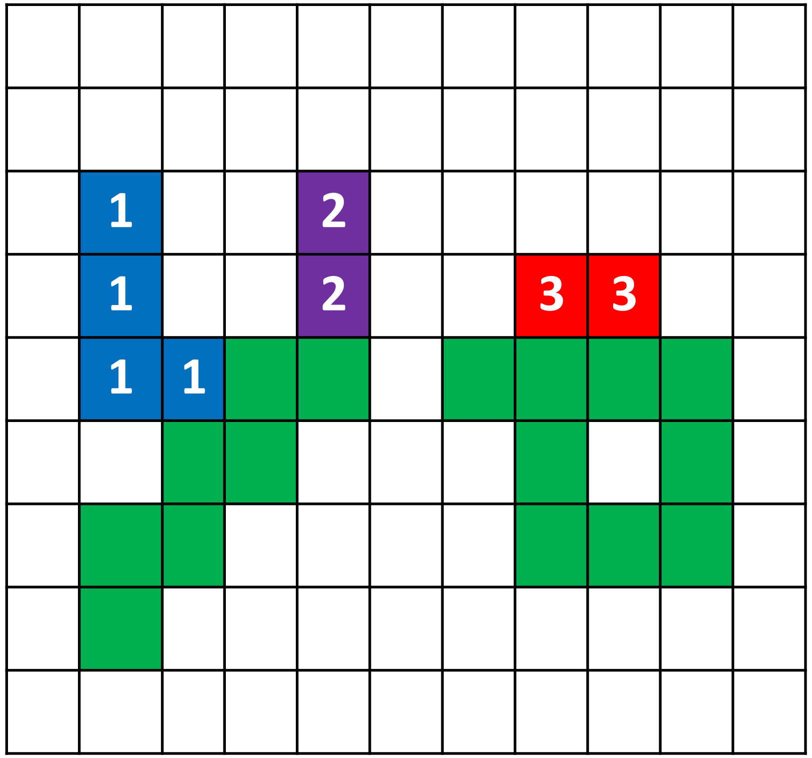
Connected Component Labeling (example)



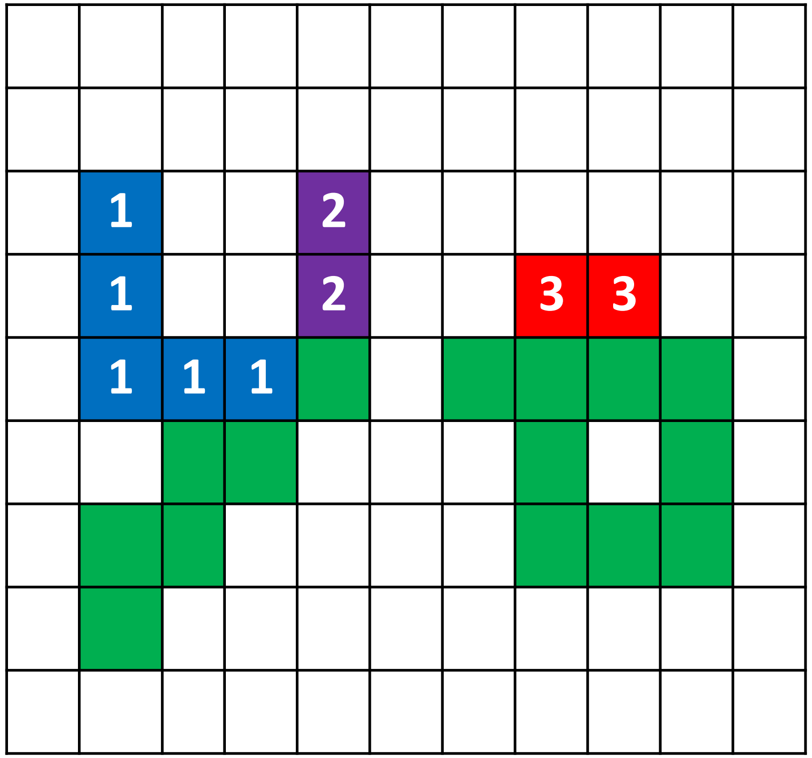
Connected Component Labeling (example)



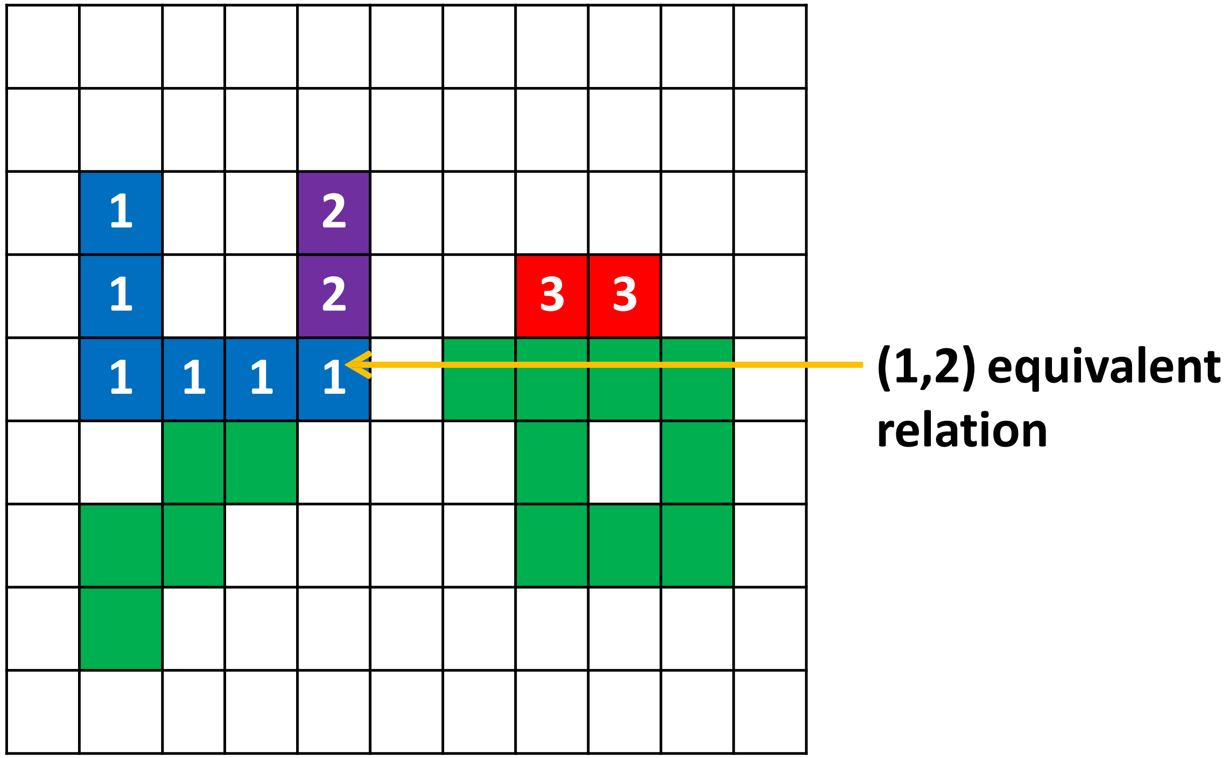
Connected Component Labeling (example)



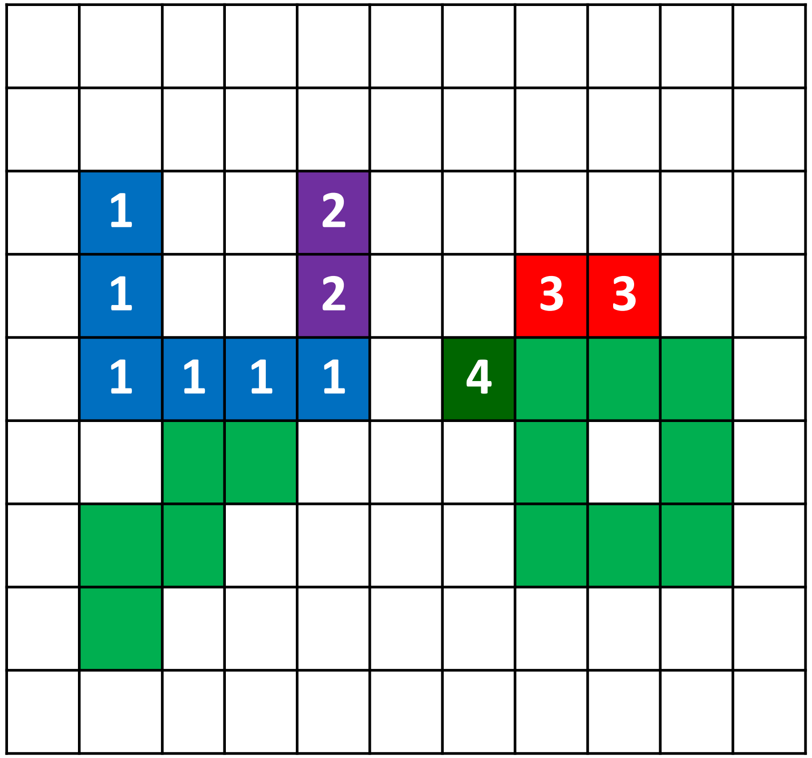
Connected Component Labeling (example)



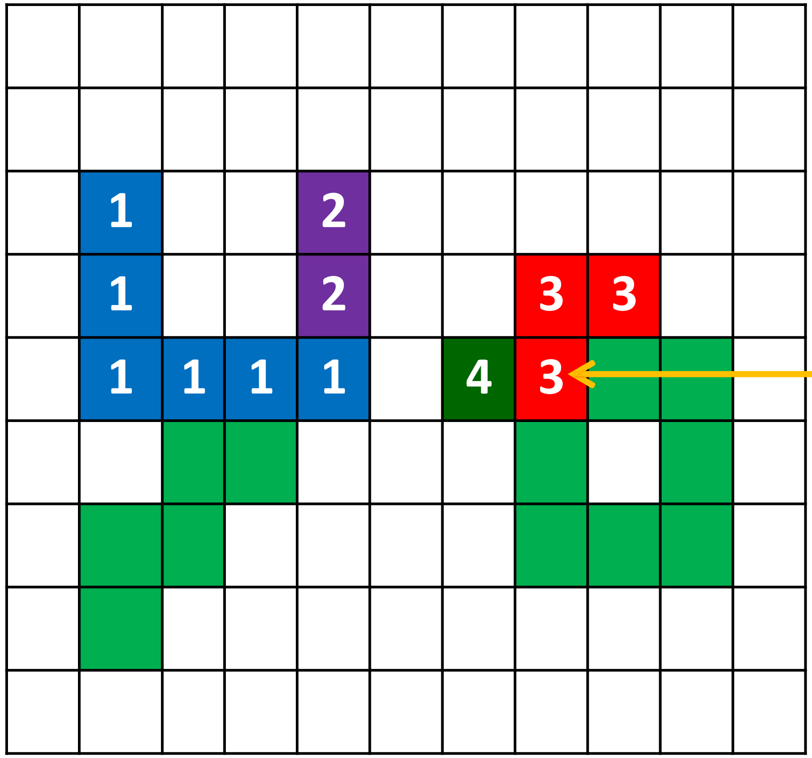
Connected Component Labeling (example)



Connected Component Labeling (example)

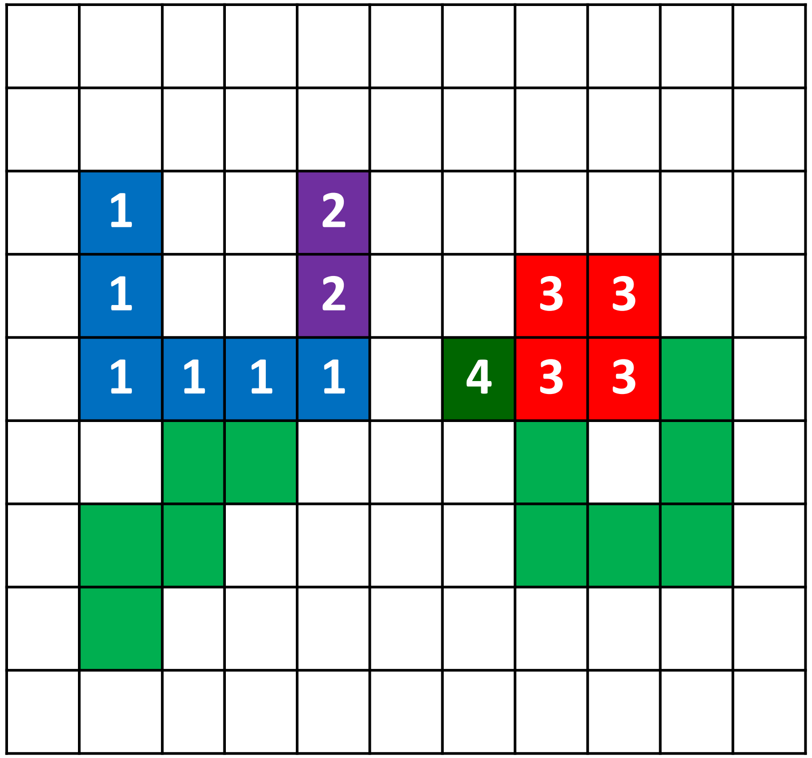


Connected Component Labeling (example)

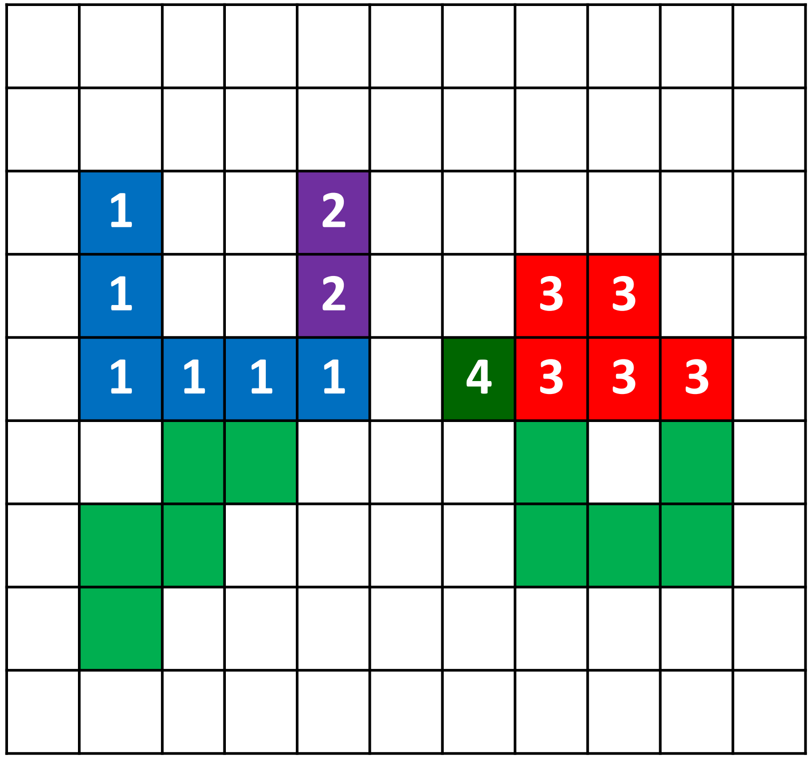


(3,4) equivalent relation

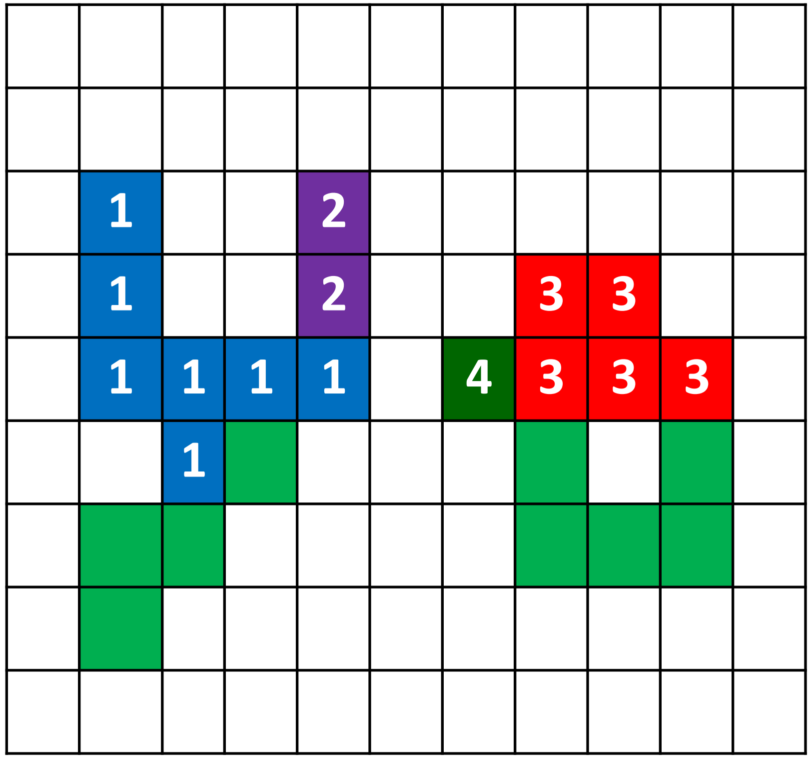
Connected Component Labeling (example)



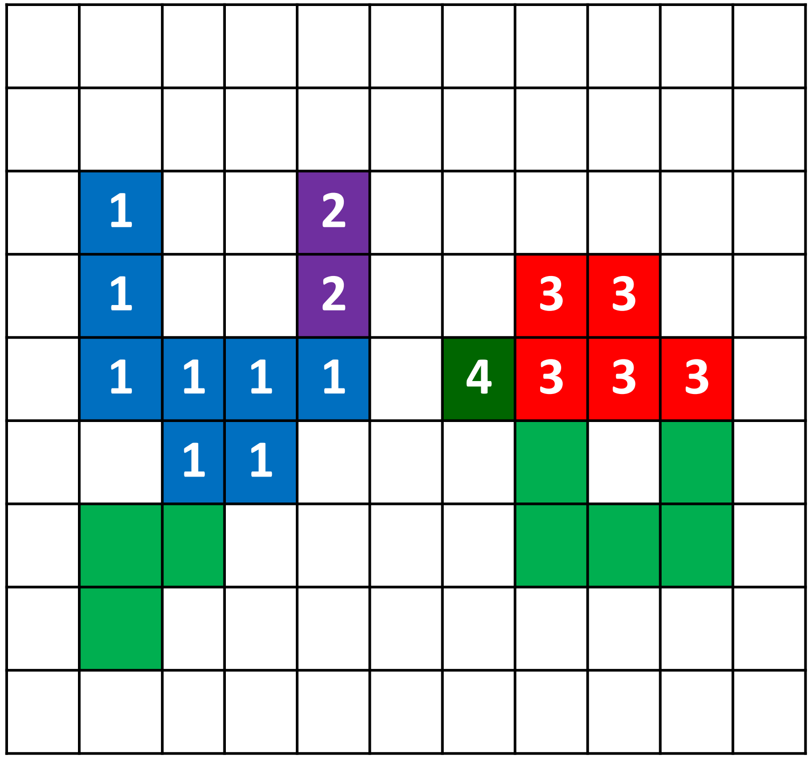
Connected Component Labeling (example)



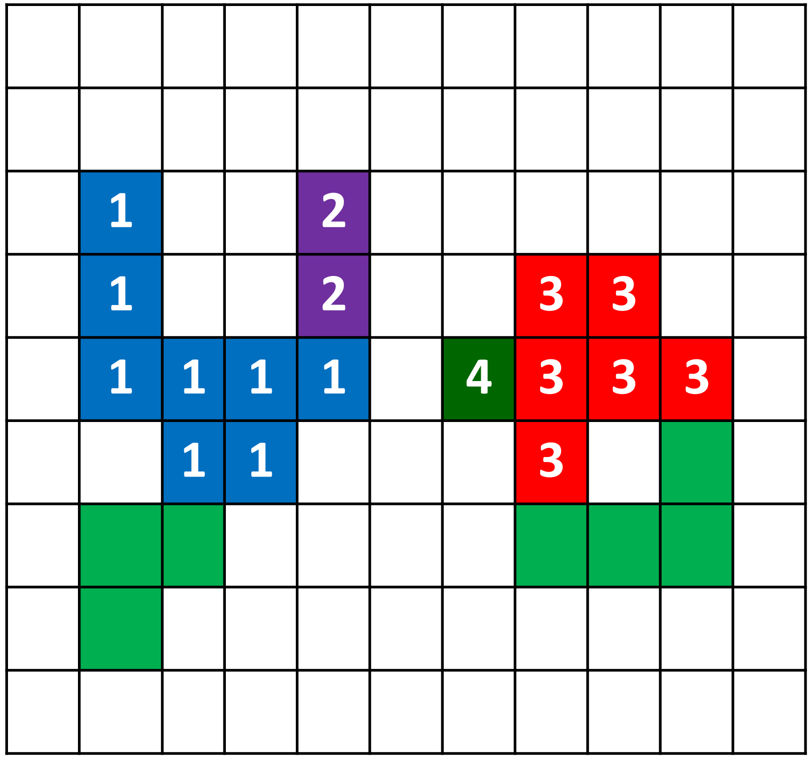
Connected Component Labeling (example)



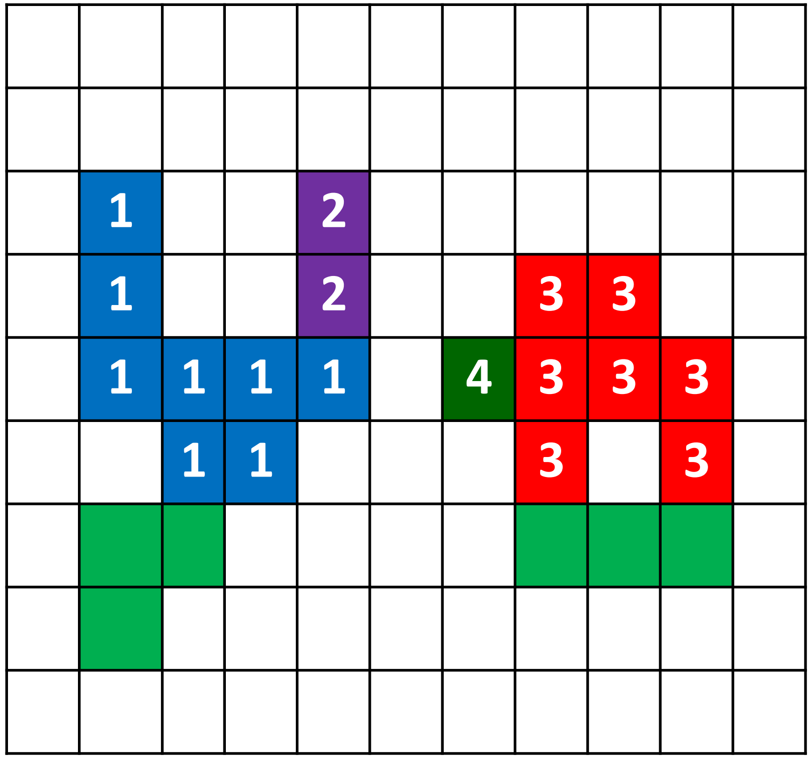
Connected Component Labeling (example)



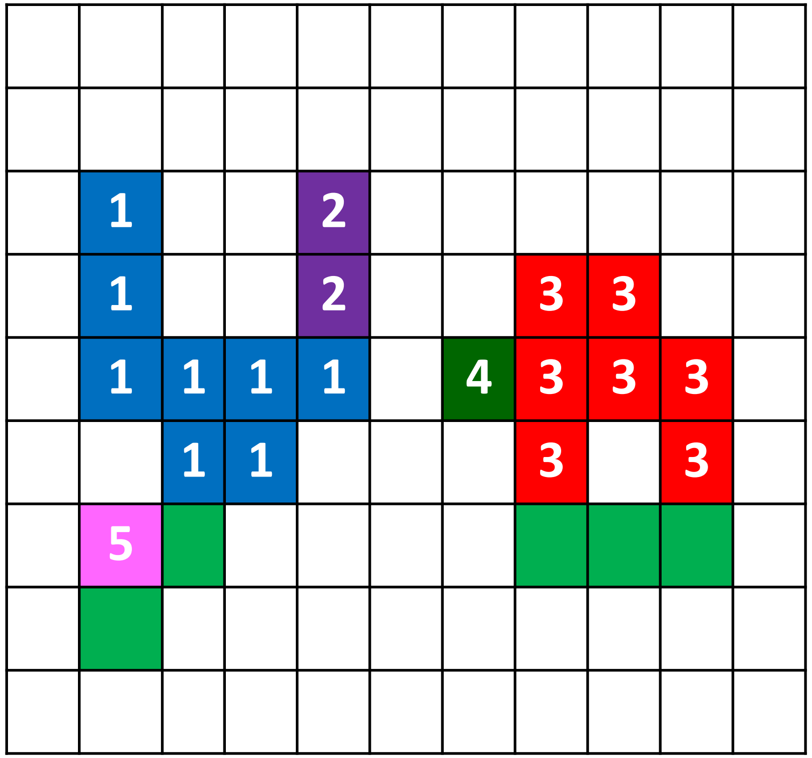
Connected Component Labeling (example)



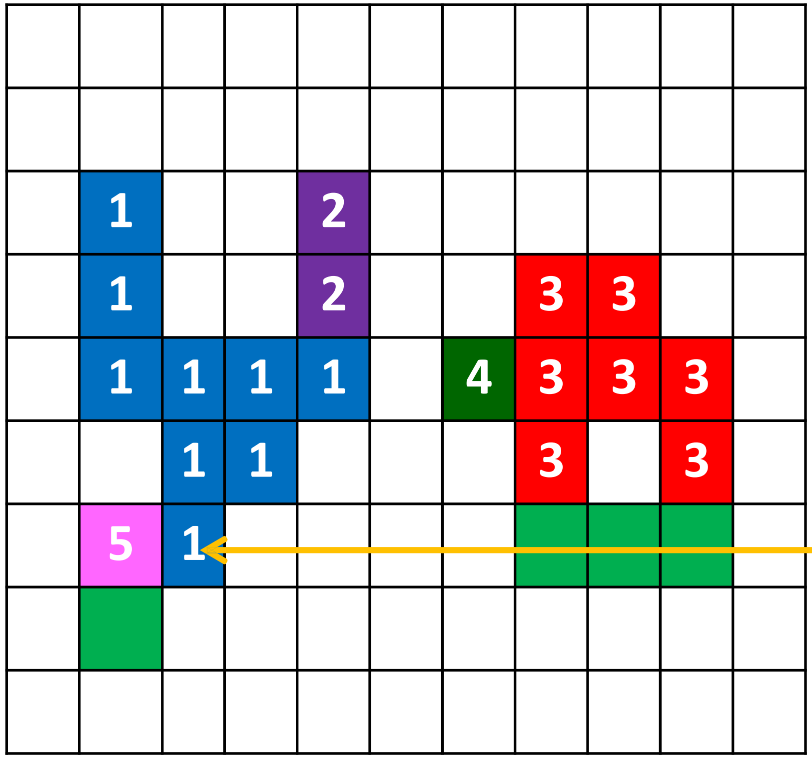
Connected Component Labeling (example)



Connected Component Labeling (example)

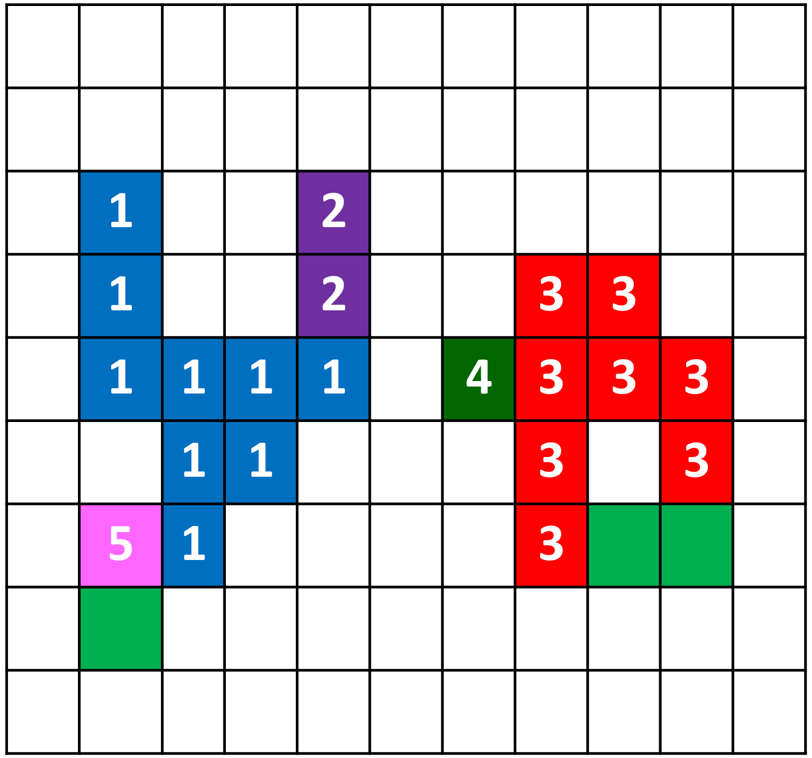


Connected Component Labeling (example)

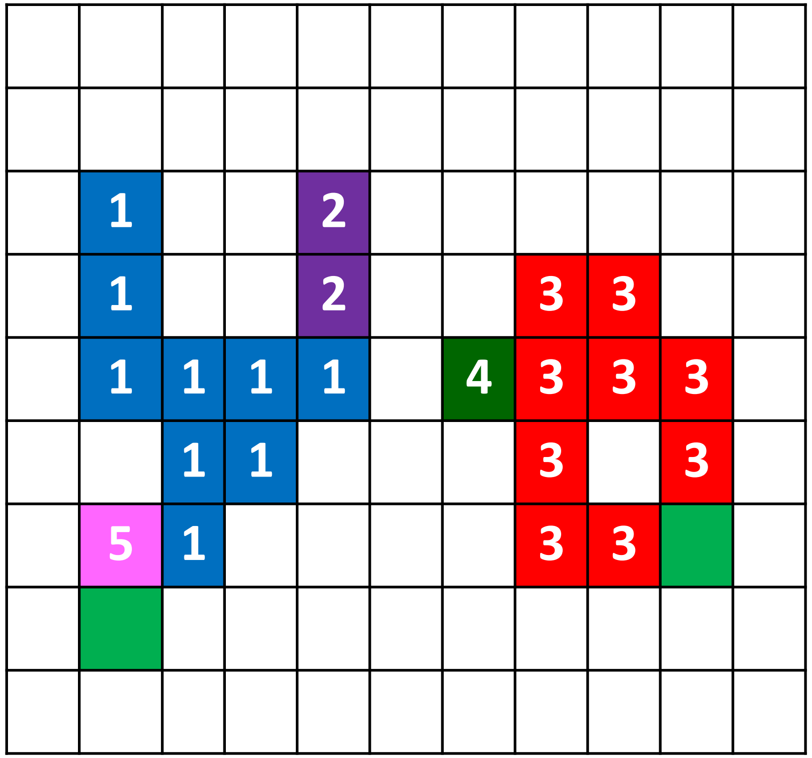


(1,5) equivalent relation

Connected Component Labeling (example)



Connected Component Labeling (example)



Connected Component Labeling (example)

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

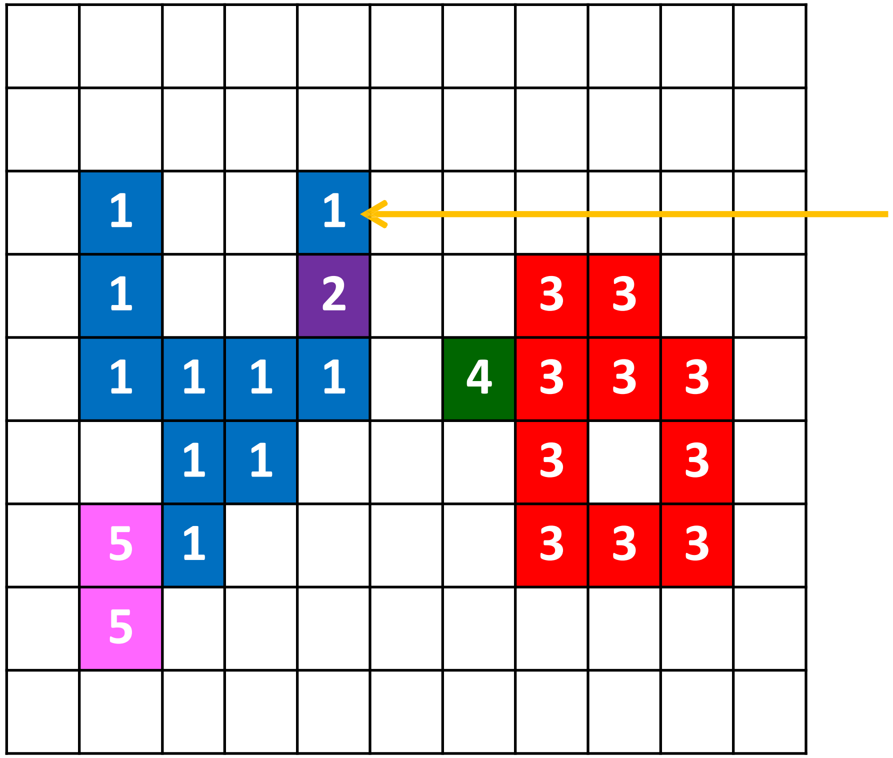
Connected Component Labeling (example)

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

Connected Component Labeling (example)

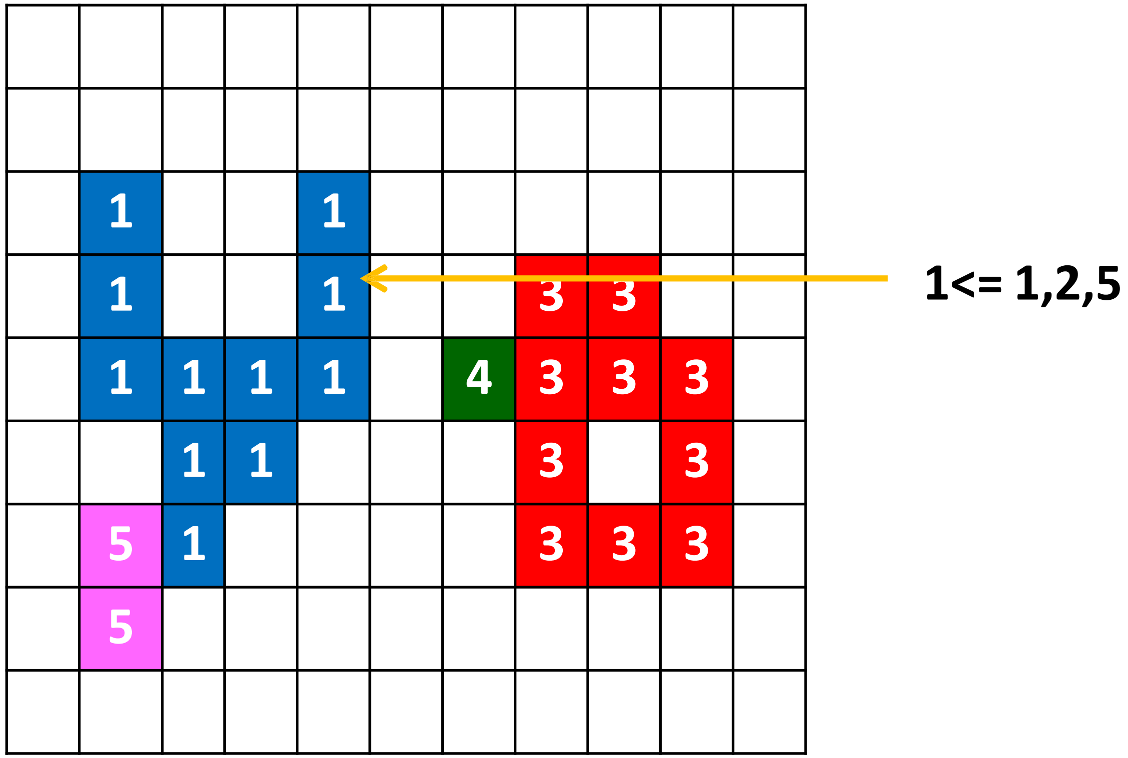
	1			1						
	1			2			3	3		
	1	1	1	1		4	3	3	3	
		1	1				3		3	
	5	1					3	3	3	
	5									

Connected Component Labeling (example)

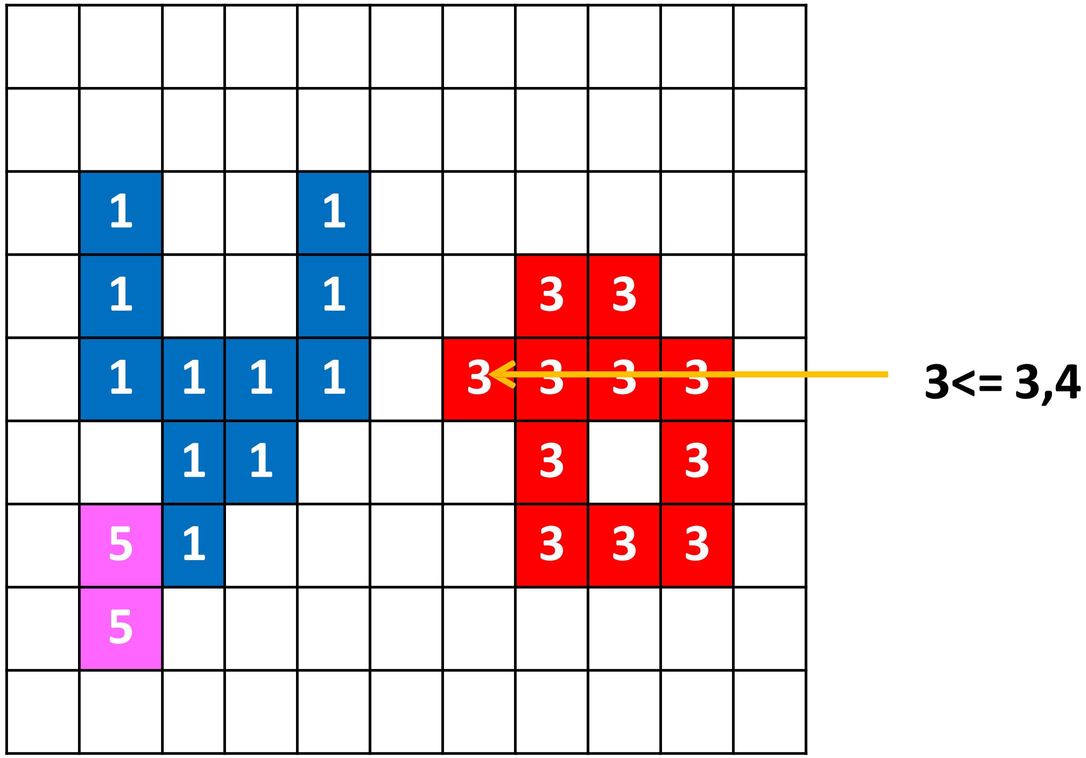


1<= 1,2,5

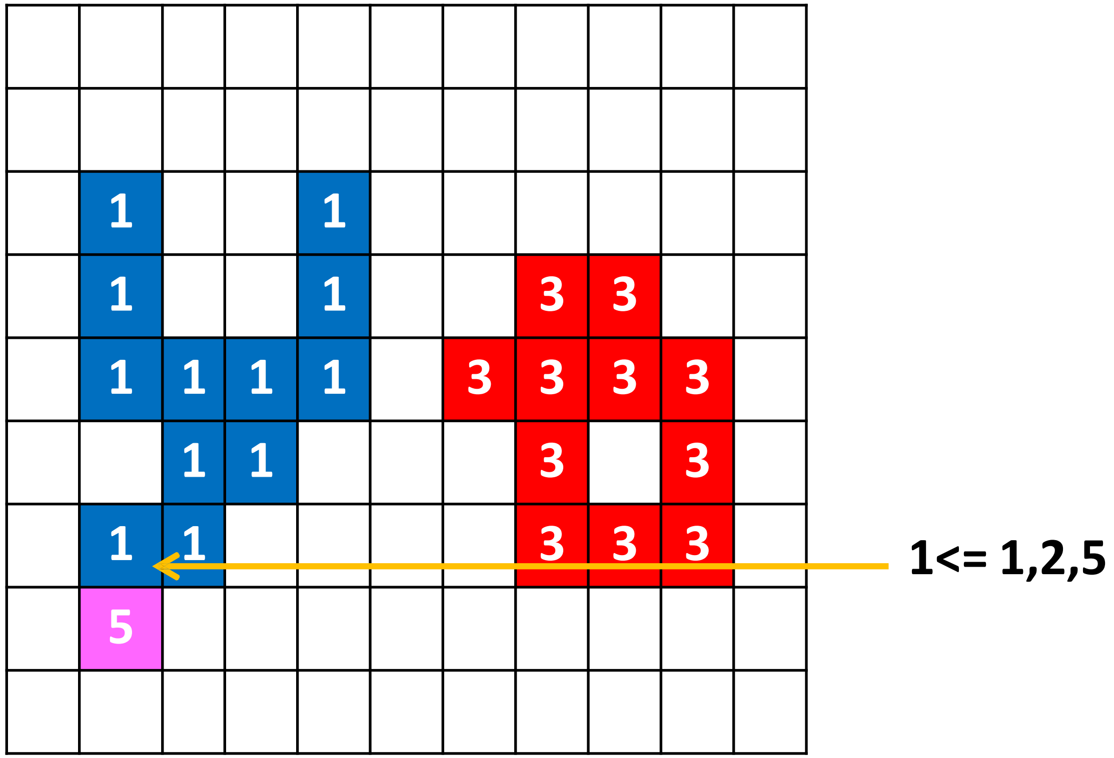
Connected Component Labeling (example)



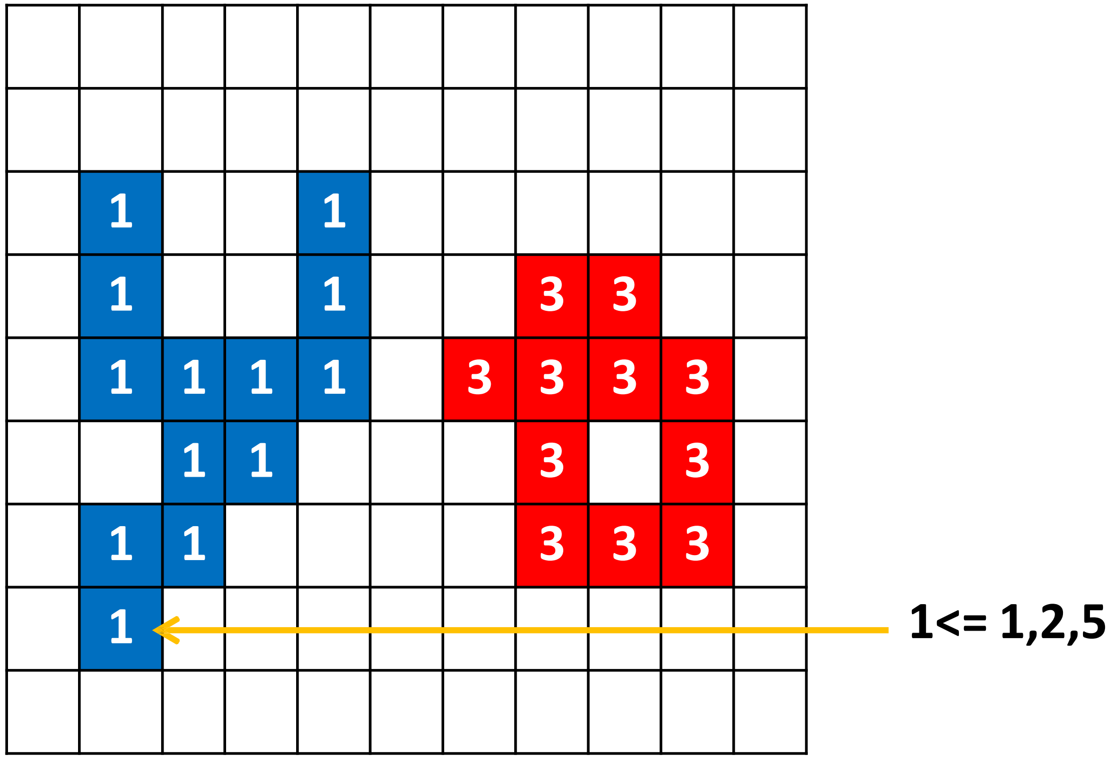
Connected Component Labeling (example)



Connected Component Labeling (example)



Connected Component Labeling (example)



Connected Component Labeling (example)

	1			1						
	1			1			3	3		
	1	1	1	1		3	3	3	3	
		1	1				3		3	
	1	1					3	3	3	
	1									

Here we observed that the image contain two distinct class of regions

Binarization using Otsu algorithm

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

image = cv2.imread('../data/Lena.png', 0)

otsu_thr, otsu_mask = cv2.threshold(image, -1, 1, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
print('Estimated threshold (Otsu):', otsu_thr)

plt.figure(figsize=(6,3))
plt.subplot(121)
plt.axis('off')
plt.title('original')
plt.imshow(image, cmap='gray')
plt.subplot(122)
plt.axis('off')
plt.title('Otsu threshold')
plt.imshow(otsu_mask, cmap='gray')
plt.tight_layout()
plt.show()
```

Finding external and internal contours

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

image = cv2.imread('../data/BnW.png', 0)

contours, hierarchy = cv2.findContours(image, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)

image_external = np.zeros(image.shape, image.dtype)
for i in range(len(contours)):
    if hierarchy[0][i][3] == -1:
        cv2.drawContours(image_external, contours, i, 255, -1)

image_internal = np.zeros(image.shape, image.dtype)
for i in range(len(contours)):
    if hierarchy[0][i][3] != -1:
        cv2.drawContours(image_internal, contours, i, 255, -1)

plt.figure(figsize=(10,3))
plt.subplot(131)
plt.axis('off')
plt.title('original')
plt.imshow(image, cmap='gray')
plt.subplot(132)
plt.axis('off')
plt.title('external')
plt.imshow(image_external, cmap='gray')
plt.subplot(133)
plt.axis('off')
plt.title('internal')
plt.imshow(image_internal, cmap='gray')
plt.tight_layout()
plt.show()
```

Extracting connected component

```
import cv2
import numpy as np

img = cv2.imread('../data/BnW.png', cv2.IMREAD_GRAYSCALE)

connectivity = 8
num_labels, labelmap = cv2.connectedComponents(img, connectivity, cv2.CV_32S)

img = np.hstack((img, labelmap.astype(np.float32)/(num_labels - 1)))
cv2.imshow('Connected components', img)
cv2.waitKey()
cv2.destroyAllWindows()

img = cv2.imread('../data/Lena.png', cv2.IMREAD_GRAYSCALE)
otsu_thr, otsu_mask = cv2.threshold(img, -1, 1, cv2.THRESH_BINARY | cv2.THRESH_OTSU)

output = cv2.connectedComponentsWithStats(otsu_mask, connectivity, cv2.CV_32S)

num_labels, labelmap, stats, centers = output

colored = np.full((img.shape[0], img.shape[1], 3), 0, np.uint8)

for l in range(1, num_labels):
    if stats[l][4] > 200:
        colored[labelmap == l] = (0, 255*l/num_labels, 255*(num_labels-l)/num_labels)
        cv2.circle(colored,
                   (int(centers[l][0]), int(centers[l][1])), 5, (255, 0, 0), cv2.FILLED)

img = cv2.cvtColor(otsu_mask*255, cv2.COLOR_GRAY2BGR)

cv2.imshow('Connected components', np.hstack((img, colored)))
cv2.waitKey()
cv2.destroyAllWindows()
```

Fitting lines and circles

```python
import cv2
import numpy as np
import random

img = np.full((512, 512, 3), 255, np.uint8)

axes = (int(256*random.uniform(0, 1)), int(256*random.uniform(0, 1)))
angle = int(180*random.uniform(0, 1))
center = (256, 256)

pts = cv2.ellipse2Poly(center, axes, angle, 0, 360, 1)
pts += np.random.uniform(-10, 10, pts.shape).astype(np.int32)

cv2.ellipse(img, center, axes, angle, 0, 360, (0, 255, 0), 3)

for pt in pts:
    cv2.circle(img, (int(pt[0]), int(pt[1])), 3, (0, 0, 255))

cv2.imshow('Fit ellipse', img)
cv2.waitKey()
cv2.destroyAllWindows()

ellipse = cv2.fitEllipse(pts)
cv2.ellipse(img, ellipse, (0, 0, 0), 3)

cv2.imshow('Fit ellipse', img)
cv2.waitKey()
cv2.destroyAllWindows()
```

```python
img = np.full((512, 512, 3), 255, np.uint8)

pts = np.arange(512).reshape(-1, 1)
pts = np.hstack((pts, pts))
pts += np.random.uniform(-10, 10, pts.shape).astype(np.int32)

cv2.line(img, (0,0), (512, 512), (0, 255, 0), 3)

for pt in pts:
    cv2.circle(img, (int(pt[0]), int(pt[1])), 3, (0, 0, 255))

cv2.imshow('Fit line', img)
cv2.waitKey()
cv2.destroyAllWindows()

vx,vy,x,y = cv2.fitLine(pts, cv2.DIST_L2, 0, 0.01, 0.01)
y0 = int(y - x*vy/vx)
y1 = int((512 - x)*vy/vx + y)
cv2.line(img, (0, y0), (512, y1), (0, 0, 0), 3)

cv2.imshow('Fit line', img)
cv2.waitKey()
cv2.destroyAllWindows()
```

Calculating image moments

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

image = np.zeros((480, 640), np.uint8)
cv2.ellipse(image, (320, 240), (200, 100), 0, 0, 360, 255, -1)

m = cv2.moments(image)
for name, val in m.items():
    print(name, '\t', val)

print('Center X estimated:', m['m10'] / m['m00'])
print('Center Y estimated:', m['m01'] / m['m00'])
```

Working with curves

```python
import cv2, random
import numpy as np

img = cv2.imread('../data/bw.png', cv2.IMREAD_GRAYSCALE)

contours, hierarchy = cv2.findContours(img, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

color = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
cv2.drawContours(color, contours, -1, (0,255,0), 3)

cv2.imshow('contours', color)
cv2.waitKey()
cv2.destroyAllWindows()

contour = contours[0]

print('Area of contour is %.2f' % cv2.contourArea(contour))
print('Signed area of contour is %.2f' % cv2.contourArea(contour, True))
print('Signed area of contour is %.2f' % cv2.contourArea(contour[::-1], True))

print('Length of closed contour is %.2f' % cv2.arcLength(contour, True))
print('Length of open contour is %.2f' % cv2.arcLength(contour, False))

hull = cv2.convexHull(contour)
cv2.drawContours(color, [hull], -1, (0,0,255), 3)
```

```python
cv2.imshow('contours', color)
cv2.waitKey()
cv2.destroyAllWindows()

print('Convex status of contour is %s' % cv2.isContourConvex(contour))
print('Convex status of its hull is %s' % cv2.isContourConvex(hull))

cv2.namedWindow('contours')

img = np.copy(color)

def trackbar_callback(value):
    global img
    epsilon = value*cv2.arcLength(contour, True)*0.1/255
    approx = cv2.approxPolyDP(contour, epsilon, True)
    img = np.copy(color)
    cv2.drawContours(img, [approx], -1, (255,0,255), 3)

cv2.createTrackbar('Epsilon', 'contours', 1, 255, lambda v: trackbar_callback(v
while True:
    cv2.imshow('contours', img)
    key = cv2.waitKey(3)
    if key == 27:
        break

cv2.destroyAllWindows()
```

Checking the location of points

```python
import cv2, random
import numpy as np

img = cv2.imread('../data/bw.png', cv2.IMREAD_GRAYSCALE)

contours, hierarchy = cv2.findContours(img, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

color = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
cv2.drawContours(color, contours, -1, (0,255,0), 3)

cv2.imshow('contours', color)
cv2.waitKey()
cv2.destroyAllWindows()

contour = contours[0]
image_to_show = np.copy(color)
measure = True

def mouse_callback(event, x, y, flags, param):
    global contour, image_to_show

    if event == cv2.EVENT_LBUTTONUP:
        distance = cv2.pointPolygonTest(contour, (x,y), measure)
        image_to_show = np.copy(color)
        if distance > 0:
            pt_color = (0, 255, 0)
        elif distance < 0:
            pt_color = (0, 0, 255)
        else:
            pt_color = (128, 0, 128)
        cv2.circle(image_to_show, (x,y), 5, pt_color, -1)
        cv2.putText(image_to_show, '%.2f' % distance, (0, image_to_show.shape[1] - 5),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255))
cv2.namedWindow('contours')
cv2.setMouseCallback('contours', mouse_callback)
```

```python
while(True):
    cv2.imshow('contours', image_to_show)
    k = cv2.waitKey(1)

    if k == ord('m'):
        measure = not measure
    elif k == 27:
        break

cv2.destroyAllWindows()
```

Computing distance to 2d point set

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

image = np.full((480, 640), 255, np.uint8)
cv2.circle(image, (320, 240), 100, 0)

distmap = cv2.distanceTransform(image, cv2.DIST_L2, cv2.DIST_MASK_PRECISE)

plt.figure()
plt.imshow(distmap, cmap='gray')
plt.show()
```