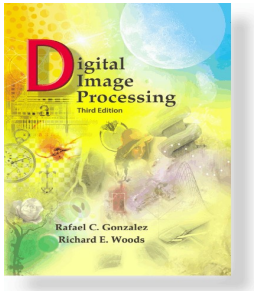


Image Analysis

– Techniques for extracting information from an image

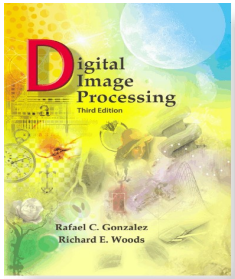
Image analysis: mainly, three steps

- *segmentation:* subdivides an image into its constituent regions or objects

- *representation* (in terms of external – boundary – or internal – texture – characteristics) and *description* (e.g. length of the boundary; mean or st. dev. of the gray level)

- *object/pattern recognition* using classification

Image Analysis:
- **Chapter 10: Segmentation**
- **Chapter 11: Representation and description**
- **Chapter 12: Object recognition**



Chapter 10

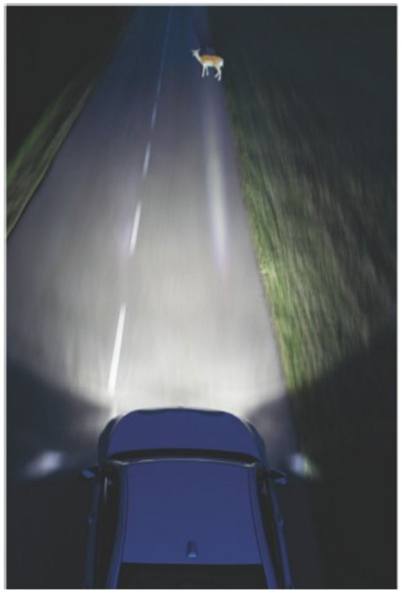
Segmentation

Example [Spectrum, March 2007]:

A robotic sentry for Korea's demilitarized zone (continuously surveilled, 250 km, one guard-post every 50 m, two guards per post)

- Under development by Samsung
- Uses 3 high sensitivity color cameras: 2 for stereo vision and tracking, 1 for zooming-in
- Distinguishes humans from other animals and from objects
- May fire its machine gun
- Does *not* distinguish friend from foe

Digital

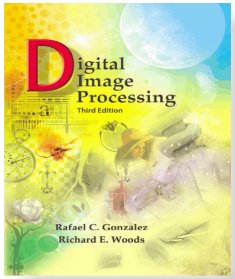


[Spectrum, Nov. 2013]

Example [http://www.prnewswire.co.uk/, Jan. 2015]: Horus Technology (Genova) Receives a US $900,000 Investment from 5Lion Holdings to Develop Innovative Solutions for the Blind and Visually Impaired

Example [Spectrum, sept. 2015]: FBI wants better automated image analysis for tattoos

- Today police take photographs of tattoos when suspects are booked, categorizing them using keywords
- searching by keyword is problematic because the categories aren't granular enough and different people often tag the same tattoo differently
- FBI would prefer to use image-based tattoo recognition technology to compare and match features extracted from the image itself.



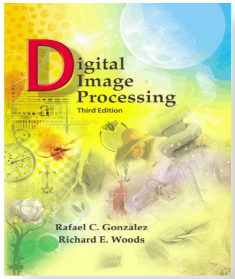
Chapter 10

Segmentation

Example [Spectrum, Feb. 2016]: self driving cars

Computer drivers

- are in principle fundamentally safer drivers: they never text, do their makeup, or fall asleep at the wheel (human error, in contrast, causes roughly 93% of crashes.)
- can have 360-degree vision, and thanks to lidar, radar, and ultrasonic sensors, they can see through fog and in the dark
- can have "telepathy" and react faster
- can take far more rigorous driver tests than the 20-minute road tests
- have the potential to accumulate far more wisdom than any human.



Chapter 10

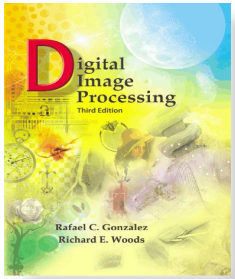
Segmentation

Image analysis:

First step: **Segmentation**, i.e. subdivision of the image into its constituent parts or objects. Autonomous segmentation is one of the most difficult tasks in image processing!

Segmentation algorithms are based on two basic properties of gray-level values:

- **Discontinuity**: the image is partitioned based on abrupt changes in gray level. Main approach is **edge detection**.
- **Similarity**: partition an image into regions that are similar. Main approaches are **thresholding**, **region growing**, and **region splitting and merging**.



Chapter 10

Segmentation

Segmentation: Subdivision of an image into its constituent parts

The level of segmentation depends on the application

10.1 Fundamentals

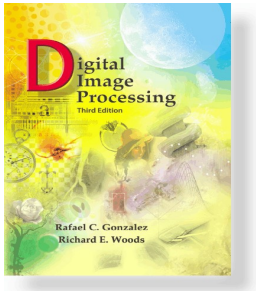
Segmentation based on:

- **(1) Discontinuities**
  - **(1) Isolated points**
  - **(2) Lines**
  - **(3) Edges**
- **(2) Similarity**
  - **(1) Thresholding**
  - **(2) Region growing**
  - **(3) Region splitting/merging**

(1) Edge-based segmentation

(2) Region-based segmentation

Credits: H. Coetzer, Department of Applied Mathematics, Stellenbosch University, S. Africa



Chapter 10

Segmentation

Let R represent the entire image region
The segmentation process partitions R into n subregions, R_1, R_2, ..., R_n, such that...

(a) $\bigcup_{i=1}^{n} R_i = R$

(b) R_i is a connected set, $i = 1, 2, \ldots, n$

(c) $R_i \bigcap R_j = \emptyset$ for all i and $j, i \neq j$

(d) $Q(R_i)$ = TRUE for $i = 1, 2, \ldots, n$

(e) $Q(R_i \bigcup R_j)$ = FALSE for any adjacent regions R_i and R_j

Here $Q(R_k)$ is logical predicate defined over all points in R_k

(a) Every pixel must be in a region

(b) All the points in a region must be "connected"

(c) Regions must be disjoint

(d) For example $Q(R_i)$ = TRUE if all the pixels in R_i have the same gray level

(e) Regions R_i and R_j are different in some sense

Segmentation algorithms for monochrome images generally are based on one of two basic categories dealing with properties of intensity values: discontinuity and similarity

Another example of Q(Ri) = TRUE if average intensity valuse of pixels is less/more than some given value and/or std. dev. is less/more than some given value.

Credits: H. Coetzer, Department of Applied Mathematics, Stellenbosch University, S. Africa

Boundary detection based on intensity discontinuities

Example illustrating
Region Based Segmentation

Edge based segmentation not a suitable approach here (As boundary detection difficult because many of the non-zero intensity changes are connected to the boundary)

4x4 size subregions considered and std. dev based rule applied

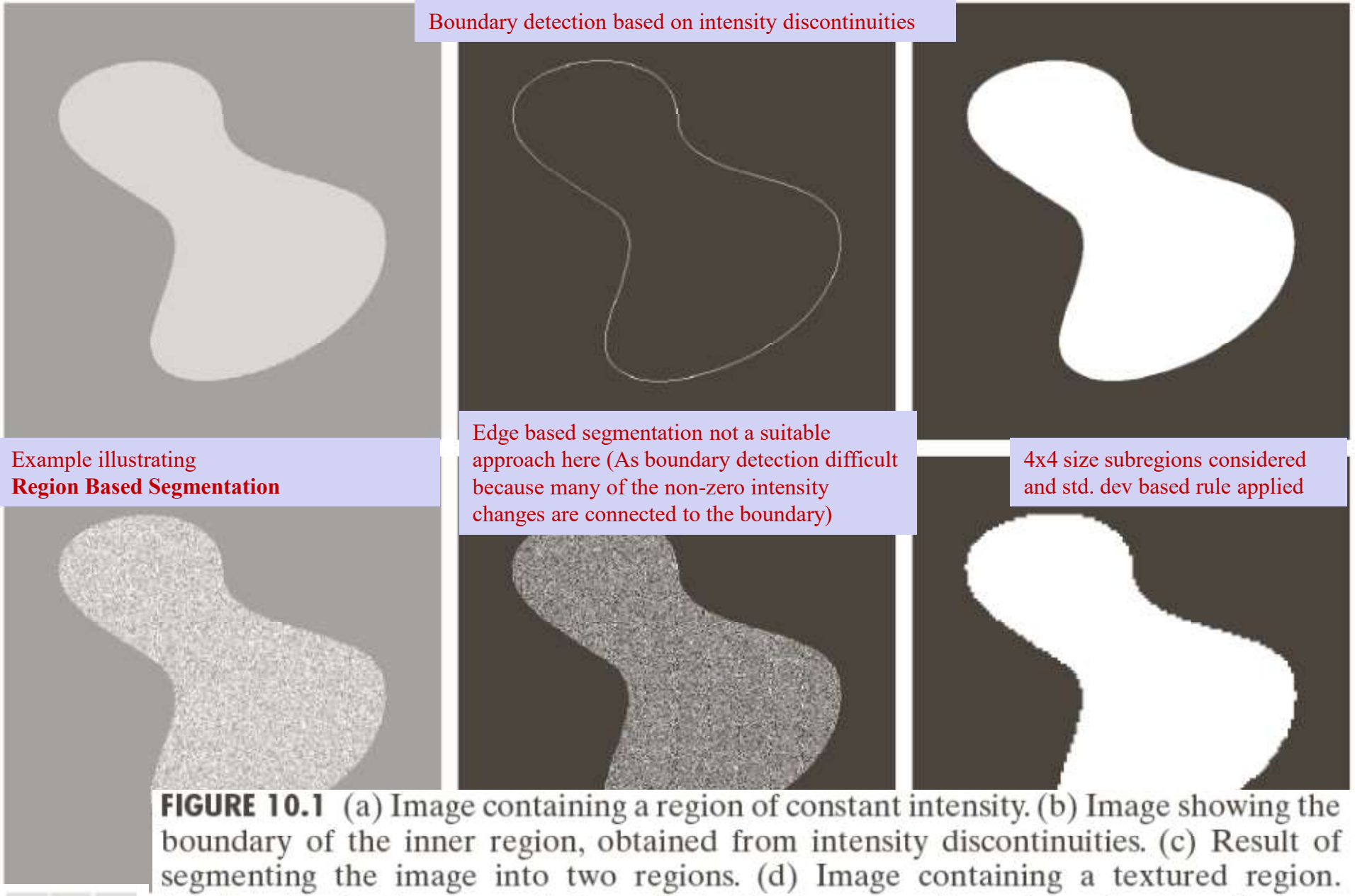


FIGURE 10.1 (a) Image containing a region of constant intensity. (b) Image showing the boundary of the inner region, obtained from intensity discontinuities. (c) Result of segmenting the image into two regions. (d) Image containing a textured region. (e) Result of edge computations. Note the large number of small edges that are connected to the original boundary, making it difficult to find a unique boundary using only edge information. (f) Result of segmentation based on region properties.

a b c
d e f

2
Regions in Binary Images

In binary images, a pixel can take on exactly one of two values. These values are often thought of as representing the "foreground" and "background" in the image, even though these concepts often are not applicable to natural scenes. In this chapter we focus on connected regions in images and how to isolate and describe such structures.

Let us assume that our task is to devise a procedure for finding the number and type of objects contained in a figure like Fig. 2.1. As long as we continue

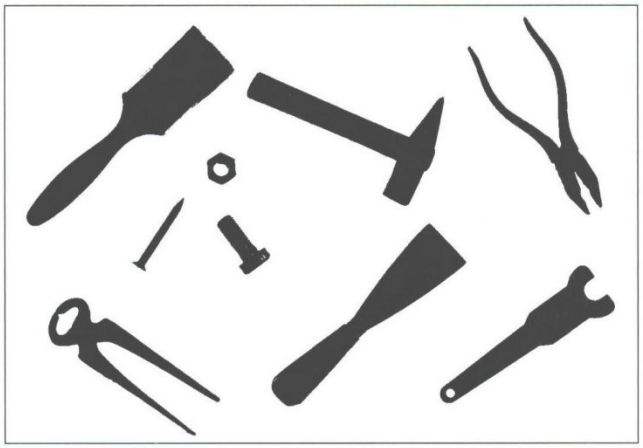


Figure 2.1 Binary image with nine objects. Each object corresponds to a connnected region of related foreground pixels.

W. Burger, M.J. Burge, *Principles of Digital Image Processing*, Undergraduate Topics in Computer Science, DOI 10.1007/978-1-84800-195-4_2, © Springer-Verlag London Limited, 2009

to consider each pixel in isolation, we will not be able to determine how many objects there are overall in the image, where they are located, and which pixels belong to which objects. Therefore our first step is to find each object by grouping together all the pixels that belong to it. In the simplest case, an object is a group of touching foreground pixels; that is, a connected *binary region.*

2.1 Finding Image Regions

In the search for binary regions, the most important tasks are to find out which pixels belong to which regions, how many regions are in the image, and where these regions are located. These steps usually take place as part of a process called *region labeling* or *region coloring.* During this process, neighboring pixels are pieced together in a stepwise manner to build regions in which all pixels within that region are assigned a unique number ("label") for identification. In the following sections, we describe two variations on this idea. In the first method, region marking through *flood filling*, a region is filled in all directions starting from a single point or "seed" within the region. In the second method, *sequential region marking*, the image is traversed from top to bottom, marking regions as they are encountered. In Sec. 2.2.2, we describe a third method that combines two useful processes, region labeling and contour finding, in a single algorithm.

Independent of which of the methods above we use, we must first settle on either the 4- or 8-connected definition of neighboring (see Vol. 1 [14, Fig. 7.5]) for determining when two pixels are "connected" to each other, since under each definition we can end up with different results. In the following region-marking algorithms, we use the following convention: the original binary image $I(u, v)$ contains the values 0 and 1 to mark the *background* and *foreground*, respectively; any other value is used for numbering (labeling) the regions, i. e., the pixel values are

$$I(u,v) = \begin{cases} 0 & \text{a } \textit{background} \text{ pixel} \\ 1 & \text{a } \textit{foreground} \text{ pixel} \\ 2, 3, \ldots & \text{a region } \textit{label.} \end{cases}$$

2.1.1 Region Labeling with Flood Filling

The underlying algorithm for region marking by *flood filling* is simple: search for an unmarked foreground pixel and then fill (visit and mark) all the rest of the neighboring pixels in its region (Alg. 2.1). This operation is called a "flood fill" because it is as if a flood of water erupts at the start pixel and flows out across a flat region. There are various methods for carrying out the fill operation that

Algorithm 2.1 Region marking using *flood filling* (Part 1). The binary input image I uses the value 0 for background pixels and 1 for foreground pixels. Unmarked foreground pixels are searched for, and then the region to which they belong is filled. The actual FloodFill() procedure is described in Alg. 2.2.

```
1: RegionLabeling(I)
       I: binary image; I(u,v) = 0: background, I(u,v) = 1: foreground
       The image I is labeled (destructively modified) and returned.
2:     Let m ← 2                          ▷ value of the next label to be assigned
3:     for all image coordinates (u,v) do
4:         if I(u,v) = 1 then
5:             FloodFill(I,u,v,m)         ▷ use any version from Alg. 2.2
6:             m ← m + 1.
7:     return the labeled image I.
```

ultimately differ in how to select the coordinates of the next pixel to be visited during the fill. We present three different ways of performing the FloodFill() procedure: a recursive version, an iterative *depth-first* version, and an iterative *breadth-first* version (see Alg. 2.2):

(A) **Recursive Flood Filling:** The recursive version (Alg. 2.2, lines 1–8) does not make use of explicit data structures to keep track of the image coordinates but uses the local variables that are implicitly allocated by recursive procedure calls.[1] Within each region, a tree structure, rooted at the starting point, is defined by the neighborhood relation between pixels. The recursive step corresponds to a *depth-first traversal* [20] of this tree and results in very short and elegant program code. Unfortunately, since the maximum depth of the recursion—and thus the size of the required stack memory—is proportional to the size of the region, stack memory is quickly exhausted. Therefore this method is risky and really only practical for very small images.

(B) **Iterative Flood Filling (*depth-first*):** Every recursive algorithm can also be reformulated as an iterative algorithm (Alg. 2.2, lines 9–20) by implementing and managing its own *stacks*. In this case, the stack records the "open" (that is, the adjacent but not yet visited) elements. As in the recursive version (A), the corresponding tree of pixels is traversed in *depth-first* order. By making use of its own dedicated stack (which is created in the much larger *heap* memory), the depth of the tree is no longer limited

[1] In Java, and similar imperative programming languages such as C and C++, local variables are automatically stored on the *call stack* at each procedure call and restored from the stack when the procedure returns.

Algorithm 2.2 Region marking using *flood filling* (Part 2). Three variations of the FLOODFILL() procedure: *recursive*, *depth-first*, and *breadth-first*.

```
1:  FLOODFILL(I, u, v, label)                          ▷ Recursive Version
2:    if (u, v) is inside the image and I(u, v) = 1 then
3:      Set I(u, v) ← label
4:      FLOODFILL(I, u+1, v, label)
5:      FLOODFILL(I, u, v+1, label)
6:      FLOODFILL(I, u, v−1, label)
7:      FLOODFILL(I, u−1, v, label)
8:    return.
9:  FLOODFILL(I, u, v, label)                          ▷ Depth-First Version
10:   Create an empty stack S
11:   Put the seed coordinate (u, v) onto the stack: PUSH(S, (u, v))
12:   while S is not empty do
13:     Get the next coordinate from the top of the stack:
            (x, y) ← POP(S)
14:     if (x, y) is inside the image and I(x, y) = 1 then
15:       Set I(x, y) ← label
16:       PUSH(S, (x+1, y))
17:       PUSH(S, (x, y+1))
18:       PUSH(S, (x, y−1))
19:       PUSH(S, (x−1, y))
20:   return.
21: FLOODFILL(I, u, v, label)                          ▷ Breadth-First Version
22:   Create an empty queue Q
23:   Insert the seed coordinate (u, v) into the queue: ENQUEUE(Q, (u, v))
24:   while Q is not empty do
25:     Get the next coordinate from the front of the queue:
            (x, y) ← DEQUEUE(Q)
26:     if (x, y) is inside the image and I(x, y) = 1 then
27:       Set I(x, y) ← label
28:       ENQUEUE(Q, (x+1, y))
29:       ENQUEUE(Q, (x, y+1))
30:       ENQUEUE(Q, (x, y−1))
31:       ENQUEUE(Q, (x−1, y))
32:   return.
```

to the size of the call stack.

(C) **Iterative Flood Filling (*breadth-first*):** In this version, pixels are traversed in a way that resembles an expanding wave front propagating out from the starting point (Alg. 2.2, lines 21–32). The data structure used to hold the as yet unvisited pixel coordinates is in this case a *queue* instead of a stack, but otherwise it is identical to version B.

Java implementation

The recursive version (A) of the algorithm corresponds practically 1:1 to its Java implementation. However, a normal Java runtime environment does not support more than about 10,000 recursive calls of the FloodFill() procedure (Alg. 2.2, line 1) before the memory allocated for the call stack is exhausted. This is only sufficient for relatively small images with fewer than approximately 200×200 pixels.

Program 2.1 gives the complete Java implementation for both variants of the iterative FloodFill() procedure. In implementing the stack (S) in the iterative *depth-first* Version (B), we use the stack data structure provided by the Java class `Stack` (Prog. 2.1, line 1), which serves as a container for generic Java objects. For the queue data structure (Q) in the *breadth-first* variant (C), we use the Java class `LinkedList`[2] with the methods `addFirst()`, `removeLast()`, and `isEmpty()` (Prog. 2.1, line 18). We have specified `<Point>` as a type parameter for both generic container classes so they can only contain objects of type `Point`.[3]

Figure 2.2 illustrates the progress of the region marking in both variants within an example region, where the start point (i. e., seed point), which would normally lie on a contour edge, has been placed arbitrarily within the region in order to better illustrate the process. It is clearly visible that the *depth-first* method first explores *one* direction (in this case horizontally to the left) completely (that is, until it reaches the edge of the region) and only then examines the remaining directions. In contrast the *breadth-first* method markings proceed outward, layer by layer, equally in all directions.

Due to the way exploration takes place, the memory requirement of the *breadth-first* variant of the *flood-fill* version is generally much lower than that of the *depth-first* variant. For example, when flood filling the region in Fig. 2.2 (using the implementation given Prog. 2.1), the stack in the *depth-first* variant

[2] The class `LinkedList` is a part of the *Java Collection Framework* (see also Vol. 1 [14, Appendix B.2]).

[3] Generic types and templates (i. e., the ability to specify a parameterization for a container) have only been available since Java 5 (1.5).

Depth-first variant (using a *stack*):

```
1 void floodFill(int x, int y, int label) {
2   Stack<Point> s = new Stack<Point>(); // stack
3   s.push(new Point(x,y));
4   while (!s.isEmpty()){
5     Point n = s.pop();
6     int u = n.x;
7     int v = n.y;
8     if ((u>=0) && (u<width) && (v>=0) && (v<height)
9           && ip.getPixel(u,v)==1) {
10      ip.putPixel(u, v, label);
11      s.push(new Point(u+1, v));
12      s.push(new Point(u, v+1));
13      s.push(new Point(u, v-1));
14      s.push(new Point(u-1, v));
15    }
16  }
17 }
```

Breadth-first variant (using a *queue*):

```
18 void floodFill(int x, int y, int label) {
19   LinkedList<Point> q = new LinkedList<Point>();
20   q.addFirst(new Point(x, y));
21   while (!q.isEmpty()) {
22     Point n = q.removeLast();
23     int u = n.x;
24     int v = n.y;
25     if ((u>=0) && (u<width) && (v>=0) && (v<height)
26           && ip.getPixel(u,v)==1) {
27       ip.putPixel(u, v, label);
28       q.addFirst(new Point(u+1, v));
29       q.addFirst(new Point(u, v+1));
30       q.addFirst(new Point(u, v-1));
31       q.addFirst(new Point(u-1, v));
32     }
33   }
34 }
```

Program 2.1 Flood filling (Java implementation). The standard class `Point` (defined in `java.awt`) represents a single pixel coordinate. The *depth-first* variant uses the standard stack operations provided by the methods `push()`, `pop()`, and `isEmpty()` of the Java class `Stack`. The *breadth-first* variant uses the Java class `LinkedList` (with access methods `addFirst()` for ENQUEUE() and `removeLast()` for DEQUEUE()) for implementing the queue data structure.

grows to a maximum of 28,822 elements, while the queue used by the *breadth-first* variant never exceeds a maximum of 438 nodes.

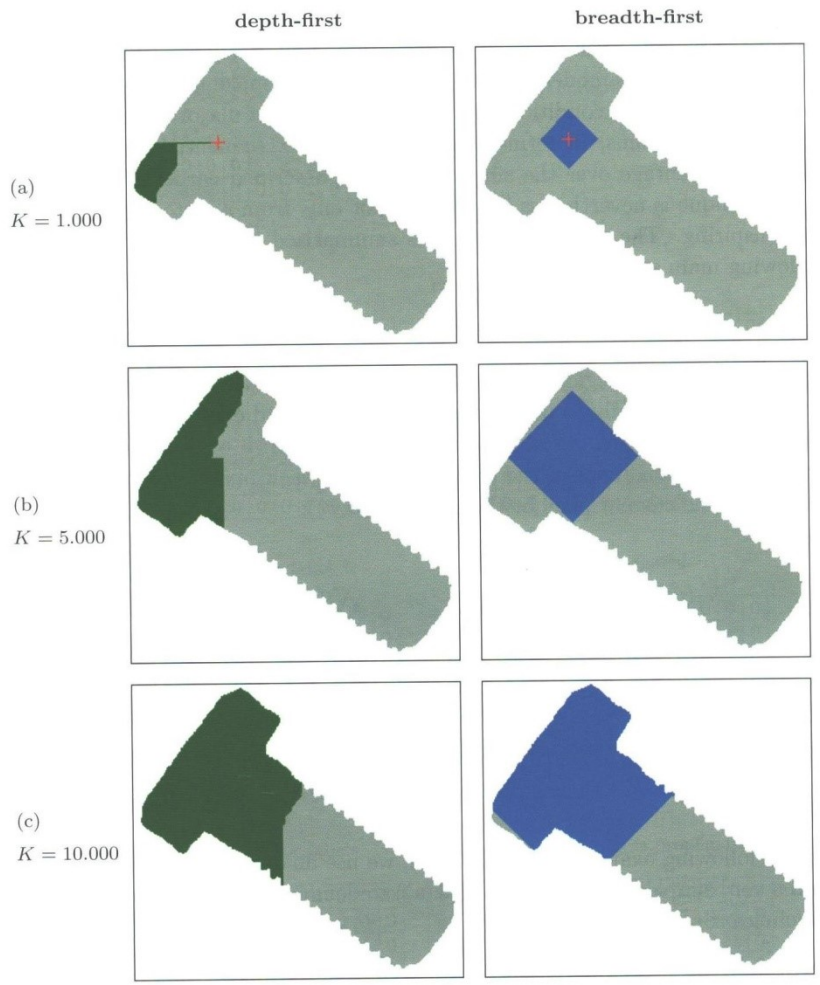


Figure 2.2 Iterative *flood filling*—comparison between the *depth-first* and *breadth-first* approach. The starting point, marked + in the top two image (a), was arbitrarily chosen. Intermediate results of the *flood fill* process after 1000 (a), 5000 (b), and 10,000 (c) marked pixels are shown. The image size is 250×242 pixels.

2.1.2 Sequential Region Labeling

Sequential region marking is a classical, nonrecursive technique that is known in the literature as "region labeling". The algorithm consists of two steps: (1) a preliminary labeling of the image regions and (2) resolving cases where more

than one label occurs (i. e., has been assigned in the previous step) in the same connected region. Even though this algorithm is relatively complex, especially its second stage, its moderate memory requirements make it a good choice under limited memory conditions. However, this is not a major issue on modern computers and thus, in terms of overall efficiency, sequential labeling offers no clear advantage over the simpler methods described earlier. The sequential technique is nevertheless interesting (not only from a historic perspective) and inspiring. The complete process is summarized in Alg. 2.3–2.4, with the following main steps:

Step 1: Initial labeling

In the first stage of region labeling, the image is traversed from top left to bottom right sequentially to assign a preliminary label to every foreground pixel. Depending on the definition of neighborhood (either 4- or 8-connected) used, the following neighbors in the direct vicinity of each pixel must be examined ($\times$ marks the current pixel at the position (u, v)):

$$\mathcal{N}_4(u,v) = \begin{array}{cc} & N_2 \\ N_1 & \times \end{array} \qquad \text{or} \qquad \mathcal{N}_8(u,v) = \begin{array}{ccc} N_2 & N_3 & N_4 \\ N_1 & \times & \end{array}$$

When using the 4-connected neighborhood $\mathcal{N}_4$, only the two neighbors $N_1 = I(u-1, v)$ and $N_2 = I(u, v-1)$ need to be considered, but when using the 8-connected neighborhood $\mathcal{N}_8$, all four neighbors $N_1 \ldots N_4$ must be examined.

Example

In the following example (Figs. 2.3–2.5), we use an 8-connected neighborhood and a very simple test image (Fig. 2.3 (a)) to demonstrate the sequential region labeling process.

Propagating labels. Again we assume that, in the image, the value $I(u, v) = 0$ represents background pixels and the value $I(u, v) = 1$ represents foreground pixels. We will also consider neighboring pixels that lie outside of the image matrix (e. g., on the array borders) to be part of the background. The neighborhood region $\mathcal{N}(u, v)$ is slid over the image horizontally and then vertically, starting from the top left corner. When the current image element $I(u, v)$ is a foreground pixel, it is either assigned a new region number or, in the case where one of its previously examined neighbors in $\mathcal{N}(u, v)$ was a foreground pixel, it takes on the region number of the neighbor. In this way, existing region numbers propagate in the image from the left to the right and from the top to the bottom, as shown in (Fig. 2.3 (b, c)).

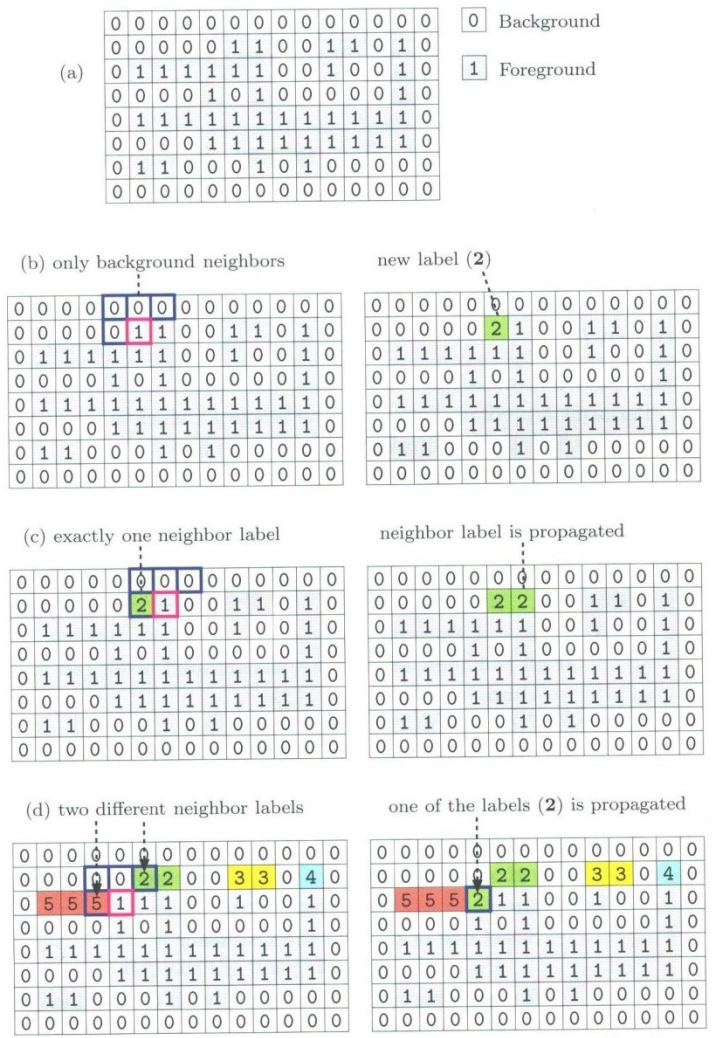


Figure 2.3 Sequential region labeling—label propagation. Original image (a). The first foreground pixel (marked **1**) is found in (b): all neighbors are background pixels (marked **0**), and the pixel is assigned the first label (**2**). In the next step (c), there is exactly *one* neighbor pixel marked with the label **2**, so this value is propagated. In (d) there are *two* neighboring pixels, and they have differing labels (**2** and **5**); one of these values is propagated, and the collision $\langle \mathbf{2}, \mathbf{5} \rangle$ is registered.

Algorithm 2.3 Sequential region labeling (Part 1). The binary input image I contains the values $I(u,v) = 0$ for background pixels and $I(u,v) = 1$ for foreground (region) pixels. The resulting region labels in I have the values $2 \ldots m-1$.

```
1:  SequentialLabeling(I)
        I: binary image; I(u,v) = 0: background, I(u,v) = 1: foreground
        The image I is labeled (destructively modified) and returned.
        m: number of assigned labels; C: set of label collisions.
2:      (m, C) ← AssignInitialLabels(I)                    ▷ see Alg. 2.4
3:      R ← ResolveLabelCollisions(m, C)                   ▷ see Alg. 2.4
4:      RelabelImage(I, R)
5:      return I.
6:  AssignInitialLabels(I)
        Performs a preliminary labeling on image I (which is modified).
        Returns the number of assigned labels (m) and
        the set of detected label collisions (C).
7:      Initialize m ← 2 (the value of the next label to be assigned).
8:      C ← {}                                             ▷ empty set of collisions
9:      for v ← 0 ... H − 1 do                             ▷ H = height of image I
10:         for u ← 0 ... W − 1 do                         ▷ W = width of image I
11:             if I(u,v) = 1 then do one of:
12:                 if all neighbors of (u,v) are background pixels (all n_i = 0)
                        then
13:                     I(u,v) ← m
14:                     m ← m + 1
15:                 else if exactly one of the neighbors has a label value n_k > 1
                        then
16:                     set I(u,v) ← n_k
17:                 else if several neighbors of (u,v) have label values n_j > 1
                        then
18:                     Select one of them as the new label:
                            I(u,v) ← k ∈ {n_j}.
19:                     for all other neighbors of (u,v) with label values n_i > 1
                            and n_i ≠ k do
20:                         Create a new label collision: c_i = ⟨n_i, k⟩.
21:                         Record the collision: C ← C ∪ {c_i}
        Remark: The image I now contains label values 0, 2, ... m − 1.
22:     return (m, C).
```

continued in Alg. 2.4 ▷▷

Algorithm 2.4 Sequential region labeling (Part 2).

1: RESOLVELABELCOLLISIONS($m, \mathcal{C}$)

Resolves the label collisions contained in the set $\mathcal{C}$.
Returns $\mathcal{R}$, a vector of sets that represents a partitioning of the complete label set into equivalent labels.

2: Let $\mathcal{L} = \{2, 3, \ldots m-1\}$ be the set of preliminary region labels.

3: Create a partitioning of $\mathcal{L}$ as a *vector of sets*, one set for each label value:
$\mathcal{R} \leftarrow [\mathcal{R}_2, \mathcal{R}_3, \ldots, \mathcal{R}_{m-1}] = [\{2\}, \{3\}, \{4\}, \ldots, \{m-1\}]$,
so $\mathcal{R}_i = \{i\}$ for all $i \in \mathcal{L}$.

4: **for all** collisions $\langle a, b \rangle \in \mathcal{C}$ **do**

5: Find in $\mathcal{R}$ the sets $\mathcal{R}_a$, $\mathcal{R}_b$:
$\mathcal{R}_a \leftarrow$ the set that currently contains label a
$\mathcal{R}_b \leftarrow$ the set that currently contains label b

6: **if** $\mathcal{R}_a \neq \mathcal{R}_b$ (a and b are contained in different sets) **then**

7: Merge sets $\mathcal{R}_a$ and $\mathcal{R}_b$ by moving all elements of $\mathcal{R}_b$ to $\mathcal{R}_a$:
$\mathcal{R}_a \leftarrow \mathcal{R}_a \cup \mathcal{R}_b, \quad \mathcal{R}_b \leftarrow \{\}$

Remark: All *equivalent* label values (i. e., all labels of pixels in the same region) are now contained in the same set $\mathcal{R}_i$ within $\mathcal{R}$.

8: **return** $\mathcal{R}$.

9: RELABELIMAGE($I, \mathcal{R}$)

Relabels the image I using the label partitioning in $\mathcal{R}$.
The image I is modified.

10: **for all** image locations (u, v) **do**

11: **if** $I(u, v) > 1$ **then** ▷ $I(u, v)$ contains a region label

12: Find the set $\mathcal{R}_i$ in $\mathcal{R}$ that contains the label $I(u, v)$

13: Choose one unique representative element k from the set $\mathcal{R}_i$, e. g., the minimum value:
$k = \min(\mathcal{R}_i)$

14: Replace the image label:
$I(u, v) \leftarrow k$

15: **return**.

Label collisions. In the case where two or more neighbors have labels belonging to *different* regions, then a label collision has occurred; that is, pixels within a single connected region have different labels. For example, in a U-shaped region, the pixels in the left and right arms are at first assigned different labels

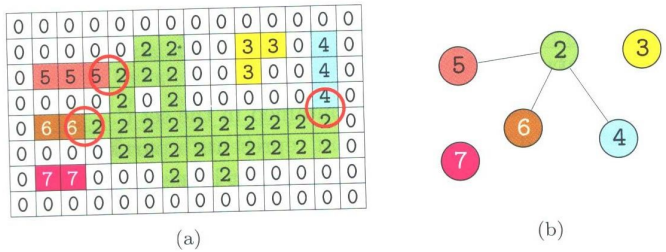


Figure 2.4 Sequential region labeling—intermediate result after Step 1. Label collisions indicated by circles (a); the nodes of the undirected graph (b) correspond to the labels, and its edges correspond to the collisions.

since it is not immediately apparent that they are actually part of a single region. The two labels will propagate down independently from each other until they eventually collide in the lower part of the "U" (Fig. 2.3 (d)).

When two labels a, b collide, then we know that they are actually "equivalent"; i. e., they are contained in the same image region. These collisions are registered but otherwise not dealt with during the first step. Once all collisions have been registered, they are then resolved in the second step of the algorithm. The number of collisions depends on the content of the image. There can be only a few or very many collisions, and the exact number is only known at the end of the first step, once the whole image has been traversed. For this reason, collision management must make use of dynamic data structures such as lists or hash tables. Upon the completion of the first steps, all the original foreground pixels have been provisionally marked, and all the collisions between labels within the same regions have been registered for subsequent processing.

The example in Fig. 2.4 illustrates the state upon completion of step 1: all foreground pixels have been assigned preliminary labels (Fig. 2.4 (a)), and the following collisions (depicted by circles) between the labels $\langle 2,4\rangle$, $\langle 2,5\rangle$, and $\langle 2,6\rangle$ have been registered. The labels $\mathcal{L} = \{2,3,4,5,6,7\}$ and collisions $\mathcal{C} = \{\langle 2,4\rangle, \langle 2,5\rangle, \langle 2,6\rangle\}$ correspond to the nodes and edges of an undirected graph (Fig. 2.4 (b)).

Step 2: Resolving collisions

The task in the second step is to resolve the label collisions that arose in the first step in order to merge the corresponding "partial" regions. This process is nontrivial since it is possible for two regions with different labels to be connected transitively (e. g., $\langle a,b\rangle \cap \langle b,c\rangle \Rightarrow \langle a,c\rangle$) through a third region or, more generally, through a series of regions. In fact, this problem is identical to the problem of finding the *connected components* of a graph [20], where the labels $\mathcal{L}$ determined in Step 1 constitute the "nodes" of the graph and the registered

collisions $\mathcal{C}$ make up its "edges" (Fig. 2.4 (b)).

Step 3: Relabeling the image

Once all the distinct labels within a single region have been collected, the labels of all the pixels in the region are updated so they carry the same label (for example, chosing the smallest label number in the region), as shown in Fig. 2.5.

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	2	2	0	0	3	3	0	2	0
0	2	2	2	2	2	2	0	0	3	0	0	2	0
0	0	0	0	2	0	2	0	0	0	0	0	2	0
0	2	2	2	2	2	2	2	2	2	2	2	2	0
0	0	0	0	2	2	2	2	2	2	2	2	2	0
0	7	7	0	0	0	2	0	2	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

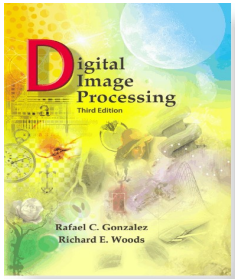
Figure 2.5 Sequential region labeling—final result after Step 3. All equivalent labels have been replaced by the smallest label within that region.

2.1.3 Region Labeling—Summary

In this section, we described a selection of algorithms for finding and labeling connected regions in images. We discovered that the elegant idea of labeling individual regions using a simple recursive flood-filling method (Sec. 2.1.1) was not useful because of practical limitations on the depth of recursion and the high memory costs associated with it. We also saw that classical sequential region labeling (Sec. 2.1.2) is relatively complex and offers no real advantage over iterative implementations of the *depth-first* and *breadth-first* methods. In practice, the iterative breadth-first method is generally the best choice for large and complex images.

2.2 Region Contours

Once the regions in a binary image have been found, the next step is often to find the contours (that is, the outlines) of the regions. Like so many other tasks in image processing, at first glance this appears to be an easy one: simply follow along the edge of the region. We will see that, in actuality, describing this apparently simple process algorithmically requires careful thought, which has made contour finding one of the classic problems in image analysis.



Chapter 10

Segmentation

10.3 Thresholding

10.3.1 Foundation

A thresholded image $g(x, y)$ is defined as

$$g(x,y) = \begin{cases} 1, & \text{if } f(x,y) > T \\ 0, & \text{if } f(x,y) \leq T \end{cases},$$

where 1 is object and 0 is background

Global thresholding: T is constant applicable over whole image

Variable (local/regional) thresholding: T changes over an image

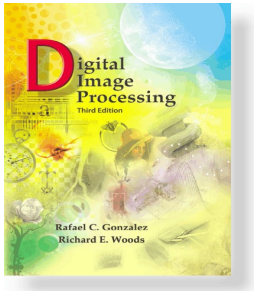
Dynamic (adaptive) thresholding: T depends on spatial coordinates (x, y)

Multiple thresholding:

$$g(x,y) = \begin{cases} a, & \text{if } f(x,y) > T_2 \\ b, & \text{if } T_1 < f(x,y) \leq T_2 \\ c, & \text{if } f(x,y) \leq T_1 \end{cases},$$

Segmentation requiring more than two thresholds is very difficult and variable thresholding (10.3.7) or region growing (10.4) is often preferred

Credits: H. Coetzer, Department of Applied Mathematics, Stellenbosch University, S. Africa

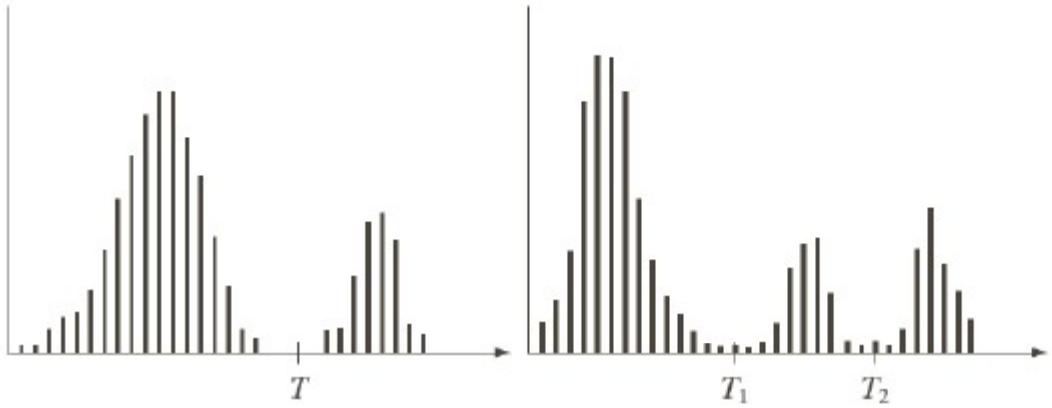


Digital Image Processing, 3rd ed.

Gonzalez & Woods

www.ImageProcessingPlace.com

Chapter 10
Segmentation



a b

FIGURE 10.35 Intensity histograms that can be partitioned (a) by a single threshold, and (b) by dual thresholds.

Width and depth of valleys (in histogram) affect success of thresholding

Properties of valleys are affected by:

(1) separation between peaks;

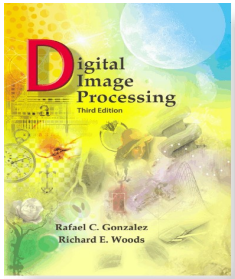
(2) noise content;

(3) relative sizes of objects and background;

(4) uniformity of illumination source;

(5) uniformity of reflectance properties of image

Credits: H. Coetzer, Department of Applied Mathematics, Stellenbosch University, S. Africa



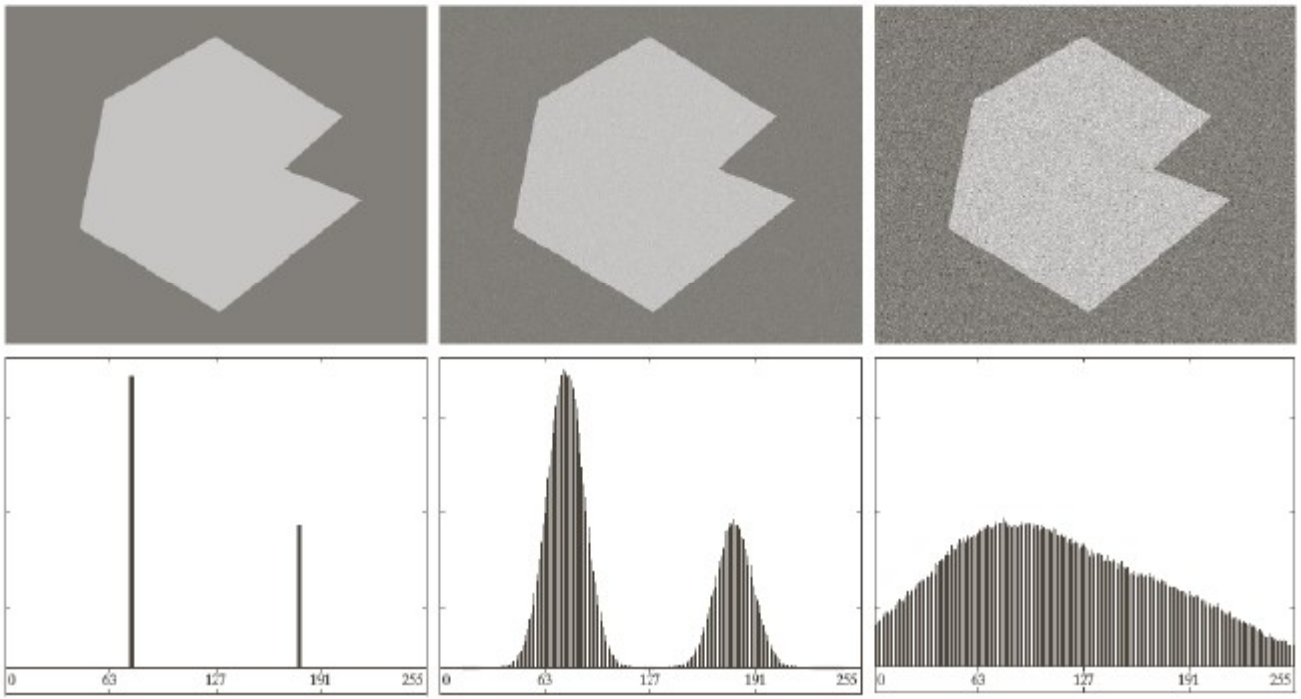
Digital Image Processing, 3rd ed.

Gonzalez & Woods

www.ImageProcessingPlace.com

Chapter 10
Segmentation

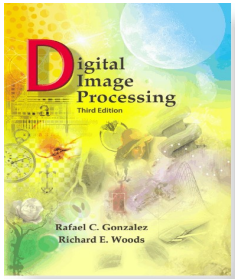
The role of noise in image thresholding



a b c
d e f

FIGURE 10.36 (a) Noiseless 8-bit image. (b) Image with additive Gaussian noise of mean 0 and standard deviation of 10 intensity levels. (c) Image with additive Gaussian noise of mean 0 and standard deviation of 50 intensity levels. (d)–(f) Corresponding histograms.

Credits: H. Coetzer, Department of Applied Mathematics, Stellenbosch University, S. Africa



Digital Image Processing, 3rd ed.

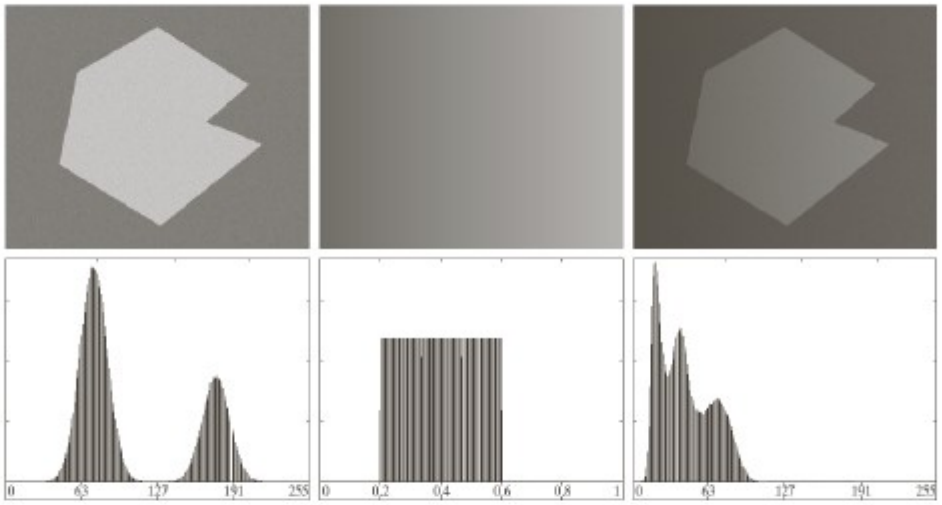
Gonzalez & Woods

www.ImageProcessingPlace.com

Chapter 10

Segmentation

The role of illumination and reflectance

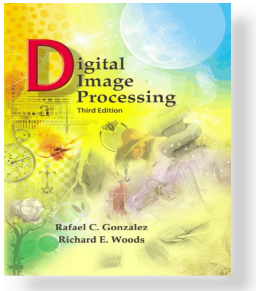


a b c
d e f

FIGURE 10.37 (a) Noisy image. (b) Intensity ramp in the range [0.2, 0.6]. (c) Product of (a) and (b). (d)–(f) Corresponding histograms.

Options for correcting non-uniform illumination: (1) Multiply with inverse of pattern by imaging flat surface with constant intensity; (2) Processing using top-hat transformation (Sec 9.6.3); (3) Variable thresholding (Sec 10.3.7)

Credits: H. Coetzer, Department of Applied Mathematics, Stellenbosch University, S. Africa



Digital Image Processing, 3rd ed.

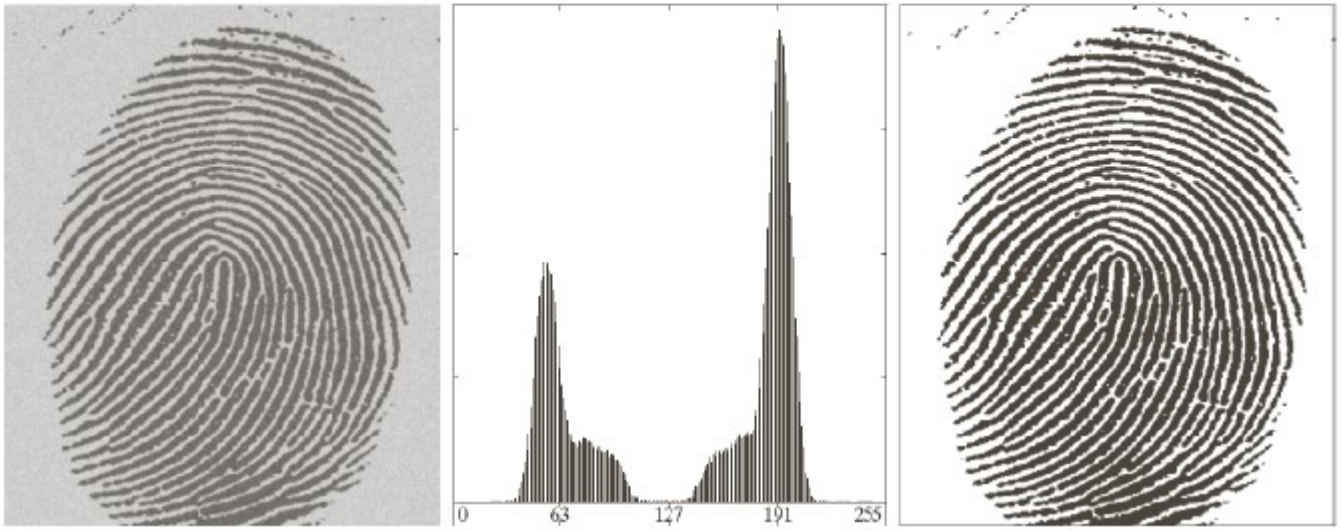
Gonzalez & Woods

www.ImageProcessingPlace.com

Chapter 10

Segmentation

Example 10.15: Global thresholding



a b c

FIGURE 10.38 (a) Noisy fingerprint. (b) Histogram. (c) Segmented result using a global threshold (the border was added for clarity). (Original courtesy of the National Institute of Standards and Technology.)

Start with average gray level and $\Delta T = 0$

Algorithm results in $\tilde{T} = 125.4$ after 3 iterations, so let $T = 125$

Credits: H. Coetzer, Department of Applied Mathematics, Stellenbosch University, S. Africa

10.3.3 Optimal global thresholding using Otsu's method

- Otsu's method (1979) maximizes between-class variance
- Based entirely on computations performed on histogram (1-D) of image
- Normalized histogram: $p_i = \frac{n_i}{MN}$, $i = 0, \ldots, L-1$, with $\sum_{i=0}^{L-1} p_i = 1$, $p_i \geq 0$
- Select threshold $T(k)$ to segment image $\longrightarrow$ Class C_1 (values $[0, k]$)
  Class C_2 (values $[k+1, L-1]$)

$\Rightarrow$ Prob of pixel assigned to C_1 (ie of C_1 occuring): $P_1(k) = \sum_{i=0}^{k} p_i$

$\Rightarrow$ Prob of pixel assigned to C_2 (ie of C_2 occuring): $P_2(k) = \sum_{i=k+1}^{L-1} p_i = 1 - P_1(k)$

Otsu's method is based on a very simple idea:

Find the threshold that *minimizes the weighted within-class variance.*

This turns out to be the same as *maximizing the between-class variance.*

Credits: H. Coetzer, Department of Applied Mathematics, Stellenbosch University, S. Africa

$\Rightarrow$ **Mean value of pixels assigned to C_1:**

$$m_1(k) = \sum_{i=0}^{k} i\, P(i/C_1)$$

$$= \sum_{i=0}^{k} i \; \overbrace{P(C_1/i)}^{=1} \; \underbrace{P(i)}_{=p_i} \; / \; \overbrace{P(C_1)}^{=P_1(k)} \quad \textbf{(Bayes' formula)}$$

$$= \frac{1}{P_1(k)} \sum_{i=0}^{k} i\, p_i$$

$\Rightarrow$ **Mean value of pixels assigned to C_2:**

$$m_2(k) = \sum_{i=k+1}^{L-1} i\, P(i/C_2)$$

$$= \frac{1}{P_2(k)} \sum_{i=k+1}^{L-1} i\, p_i$$

$\Rightarrow$ **Mean intensity up to level k:** $m(k) = \sum_{i=0}^{k} i\, p_i$

- **Global mean:** $m_G = \sum_{i=0}^{L-1} i\, p_i$

$\Rightarrow$ $\boxed{P_1 m_1 + P_2 m_2 = m_G}$ **and** $\boxed{P_1 + P_2 = 1}$ **(ks temporarily omitted)**

- **"Goodness" of threshold at level k evaluated by dimensionless metric:**

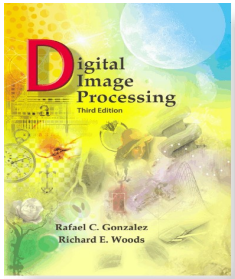
$$\boxed{\eta = \frac{\sigma_B^2}{\sigma_G^2}}$$

$$\sigma_G^2 = \sum_{i=0}^{L-1} (i - m_G)^2\, p_i \quad \textbf{(Global variance)}$$

$$\sigma_B^2 = P_1(m_1 - m_G)^2 + P_2(m_2 - m_G)^2 \quad \textbf{(Between-class variance)}$$

Also: $\sigma_B^2 = P_1 P_2 (m_1 - m_2)^2 = \dfrac{(m_G P_1 - m)^2}{P_1(1 - P_1)}$ $\leftarrow$ **most efficient**

Credits: H. Coetzer, Department of Applied Mathematics, Stellenbosch University, S. Africa



Chapter 10

Segmentation

Reintroduce $k \rightsquigarrow$ final results:

$$\eta(k) = \frac{\sigma_B^2(k)}{\sigma_G^2}$$

$$\sigma_B^2(k) = \frac{[m_G P_1(k) - m(k)]^2}{P_1(k)[1 - P_1(k)]}$$

Optimum threshold is k^* that maximizes $\sigma_B^2(k)$:

$$\sigma_B^2(k^*) = \max_{k \in [0, L-1]} \sigma_B^2(k)$$

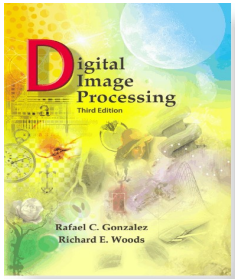
Segmentation is as follows:

$$g(x,y) = \begin{cases} 1, & \text{if } f(x,y) > k^* \\ 0, & \text{if } f(x,y) \leq k^* \end{cases},$$

The metric $\eta(k^*)$ can be used to obtain a quantative estimate of the separability of the classes and has values in the range:

$$\eta(k^*) \in [0, 1]$$

Credits: H. Coetzer, Department of Applied Mathematics, Stellenbosch University, S. Africa



Chapter 10

Segmentation

Summary of Otsu's algorithm

(1) Compute normalized histogram of the image, $p_i = \dfrac{n_i}{MN},\ i = 0, \ldots, L-1$

(2) Compute cumulative sums, $P_1(k) = \displaystyle\sum_{i=0}^{k} p_i,\ k = 0, \ldots, L-1$

(3) Compute cumulative means, $m(k) = \displaystyle\sum_{i=0}^{k} i\, p_i,\ k = 0, \ldots, L-1$

(4) Compute global intensity mean, $m_G = \displaystyle\sum_{i=0}^{L-1} i\, p_i$

(5) Compute between-class variance, $\sigma_B^2(k) = \dfrac{[m_G P_1(k) - m(k)]^2}{P_1(k)[1 - P_1(k)]},\ k = 0, ., L-1$

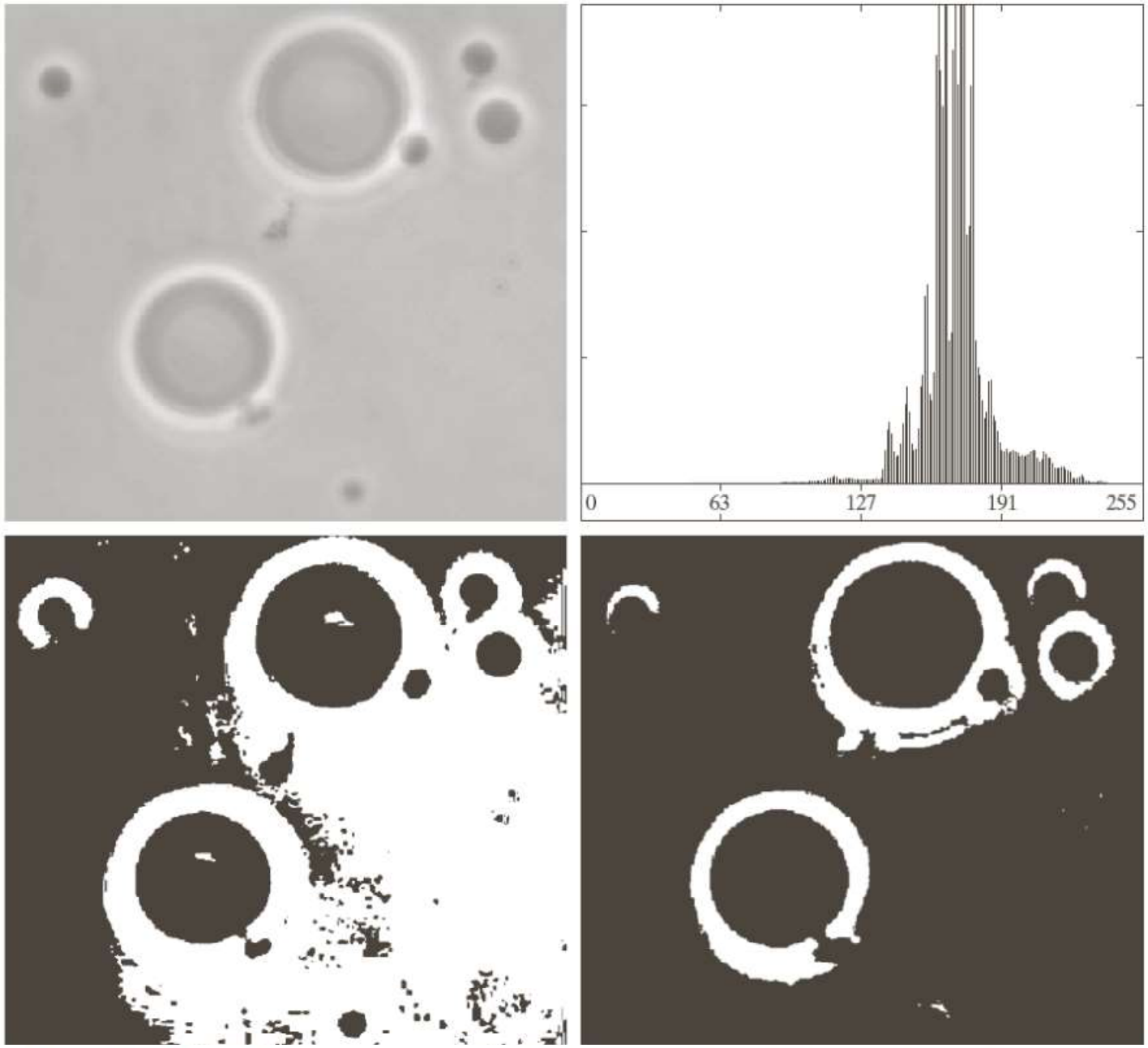
(6) Obtain the Otsu threshold, k^*, that is the value of k for which $\sigma_B^2(k^*)$ is a maximum – if this maximum is not unique, obtain k^* by avaraging the values of k that correspond to the various maxima detected

(7) Obtain the separability measure $\eta(k^*) = \dfrac{\sigma_B^2(k^*)}{\sigma_G^2}$

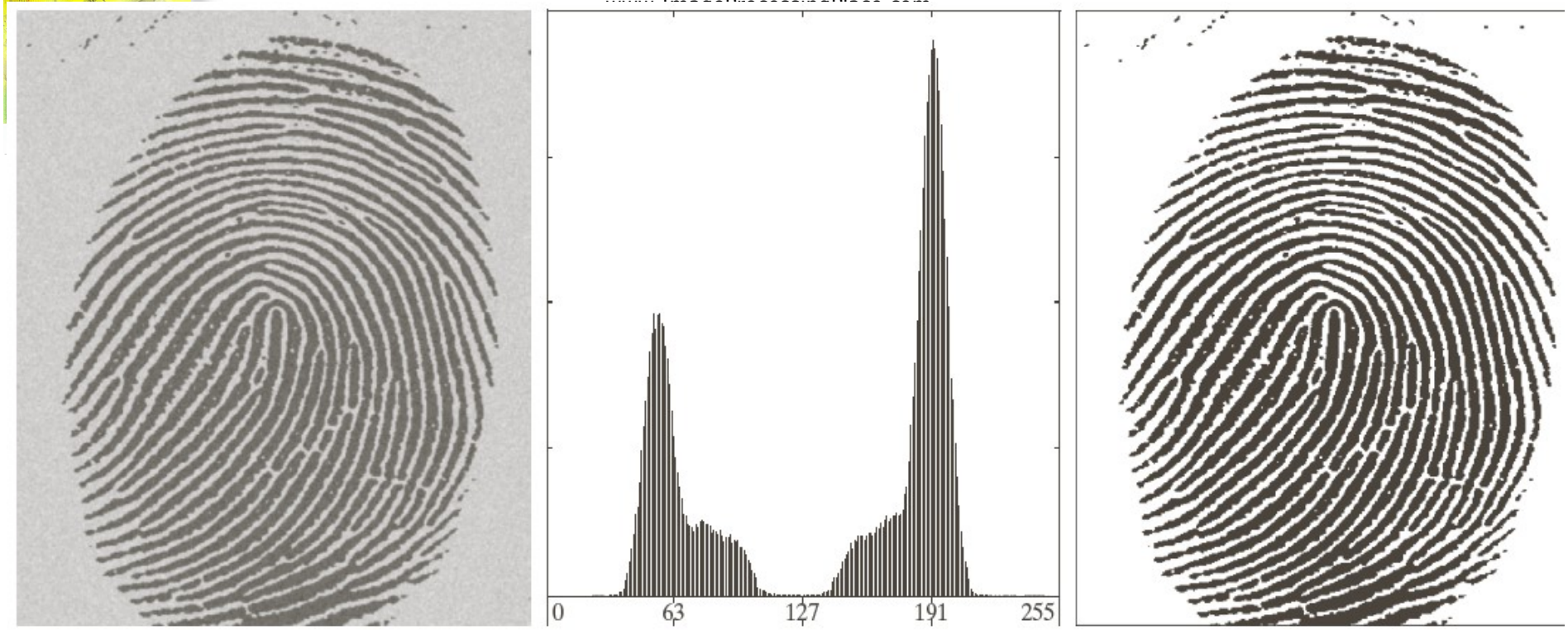
Credits: H. Coetzer, Department of Applied Mathematics, Stellenbosch University, S. Africa

a b
c d

FIGURE 10.39
(a) Original image.
(b) Histogram (high peaks were clipped to highlight details in the lower values).
(c) Segmentation result using the basic global algorithm from Section 10.3.2.
(d) Result obtained using Otsu's method.
(Original image courtesy of Professor Daniel A. Hammer, the University of Pennsylvania.)



For the above image... Threshold found by basic algorithm: $T = 161$;
Threshold found by Otsu's algorithm: $T = 181$ **(Sep measure:** $\eta = 0.467$**)**

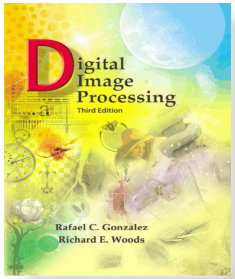


a b c

FIGURE 10.38 (a) Noisy fingerprint. (b) Histogram. (c) Segmented result using a global threshold (the bord was added for clarity). (Original courtesy of the National Institute of Standards and Technology.)

For fingerprint image... basic and Otsu's algorithm: $T = 125$ $(\eta = 0.944)$

$T_{BGT} = 125$ $T_{OGT}=125$ Separability measure, n=0.944



Multiple thresholds

Otsu's method extended to K classes, $C_1, C_2, \ldots, C_K$

Between-class variance:

$$\sigma_B^2 = \sum_{k=1}^{K} P_k(m_k - m_G)^2, \quad \text{where} \quad P_k = \sum_{i \in C_k} p_i \quad \text{and} \quad m_k = \sum_{i \in C_k} i p_i$$

The K classes are separated by $K-1$ thresholds whose values $k_1^*, k_2^*, \ldots k_{K-1}^*$ maximize

$$\sigma_B^2(k_1^*, k_2^*, \ldots, k_{K-1}^*) = \max_{0<k_1<k_2<\ldots<k_{K-1}<L-1} \sigma_B^2(k_1, k_2, \ldots, k_{K-1})$$

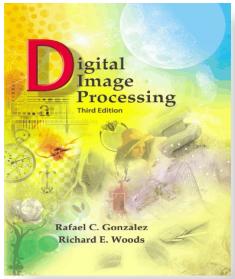
Otsu's method extended to three classes, C_1, C_2, C_3

Between-class variance:

$$\sigma_B^2 = P_1(m_1 - m_G)^2 + P_2(m_2 - m_G)^2 + P_3(m_3 - m_G)^2$$

$$P_1 = \sum_{i=0}^{k_1} p_i, \quad P_2 = \sum_{i=k_1+1}^{k_2} p_i, \quad P_3 = \sum_{i=k_2+1}^{L-1} p_i$$

Credits: H. Coetzer, Department of Applied Mathematics, Stellenbosch University, S. Africa



Multiple thresholds

$$m_1 = \frac{1}{P_1}\sum_{i=0}^{k_1} ip_i, \quad m_2 = \frac{1}{P_2}\sum_{i=k_1+1}^{k_2} ip_i, \quad m_3 = \frac{1}{P_3}\sum_{i=k_2+1}^{L-1} ip_i$$

The following relationships also hold:

$$P_1 m_1 + P_2 m_2 + P_3 m_3 = m_G, \qquad P_1 + P_2 + P_3 = 1$$

The three classes are separated by two thresholds whose values k_1^* and k_2^* maximize

$$\sigma_B^2(k_1^*, k_2^*) = \max_{0<k_1<k_2<L-1} \sigma_B^2(k_1, k_2)$$

Algorithm

(1) Let $k_1 = 1$

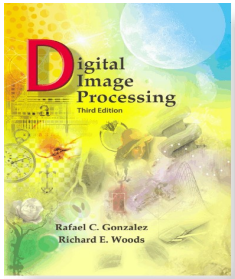
(2) Increment k_2 through all its values greater than k_1 and less than $L - 1$

(3) Increment k_1 to its next value and increment k_2 through all its values greater than k_1 and less than $L - 1$

(4) Repeat until $k_1 = L - 3$

This results in a 2-D array $\sigma_B^2(k_1, k_2)$, after which k_1^* and k_2^* that correspond to the maximum value in the array, are selected

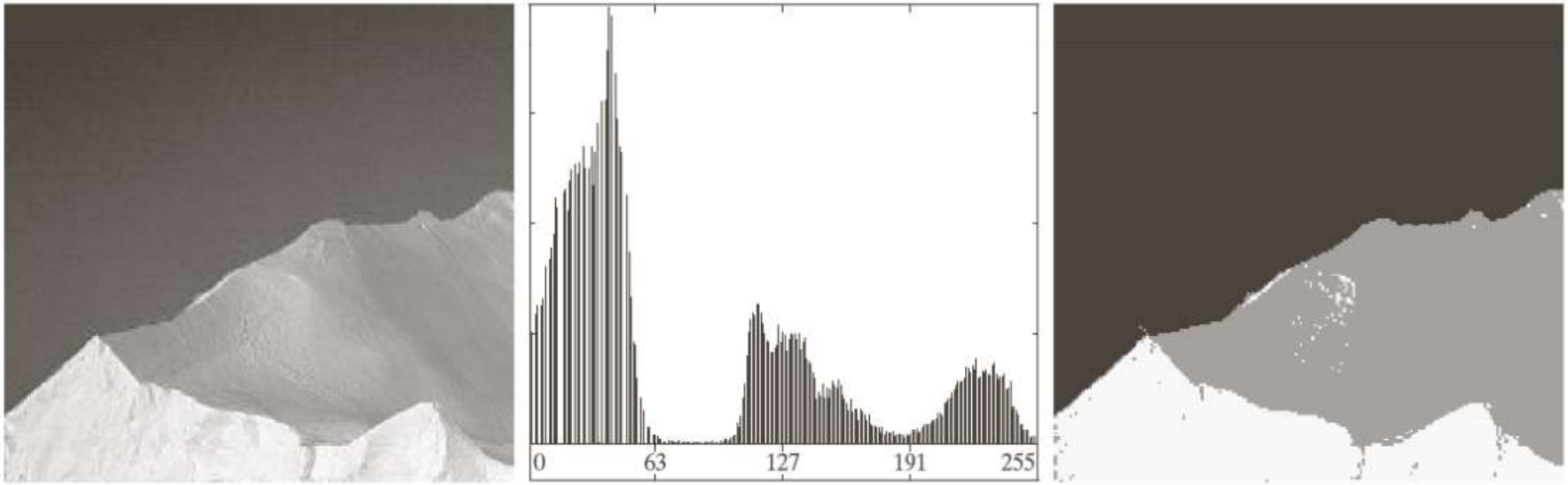
Credits: H. Coetzer, Department of Applied Mathematics, Stellenbosch University, S. Africa



Multiple thresholds

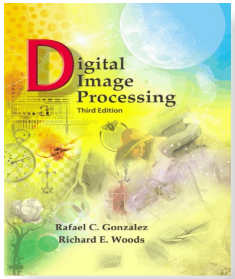
Segmentation is as follows: $g(x,y) = \begin{cases} a, & \text{if } f(x,y) \leq k_1^* \\ b, & \text{if } k_1^* < f(x,y) \leq k_2^* \\ c, & \text{if } f(x,y) > k_2^* \end{cases}$

Separability measure: $\eta(k_1^*, k_2^*) = \dfrac{\sigma_B^2(k_1^*, k_2^*)}{\sigma_G^2}$



a b c

FIGURE 10.45 (a) Image of iceberg. (b) Histogram. (c) Image segmented into three regions using dual Otsu thresholds. (Original image courtesy of NOAA.)

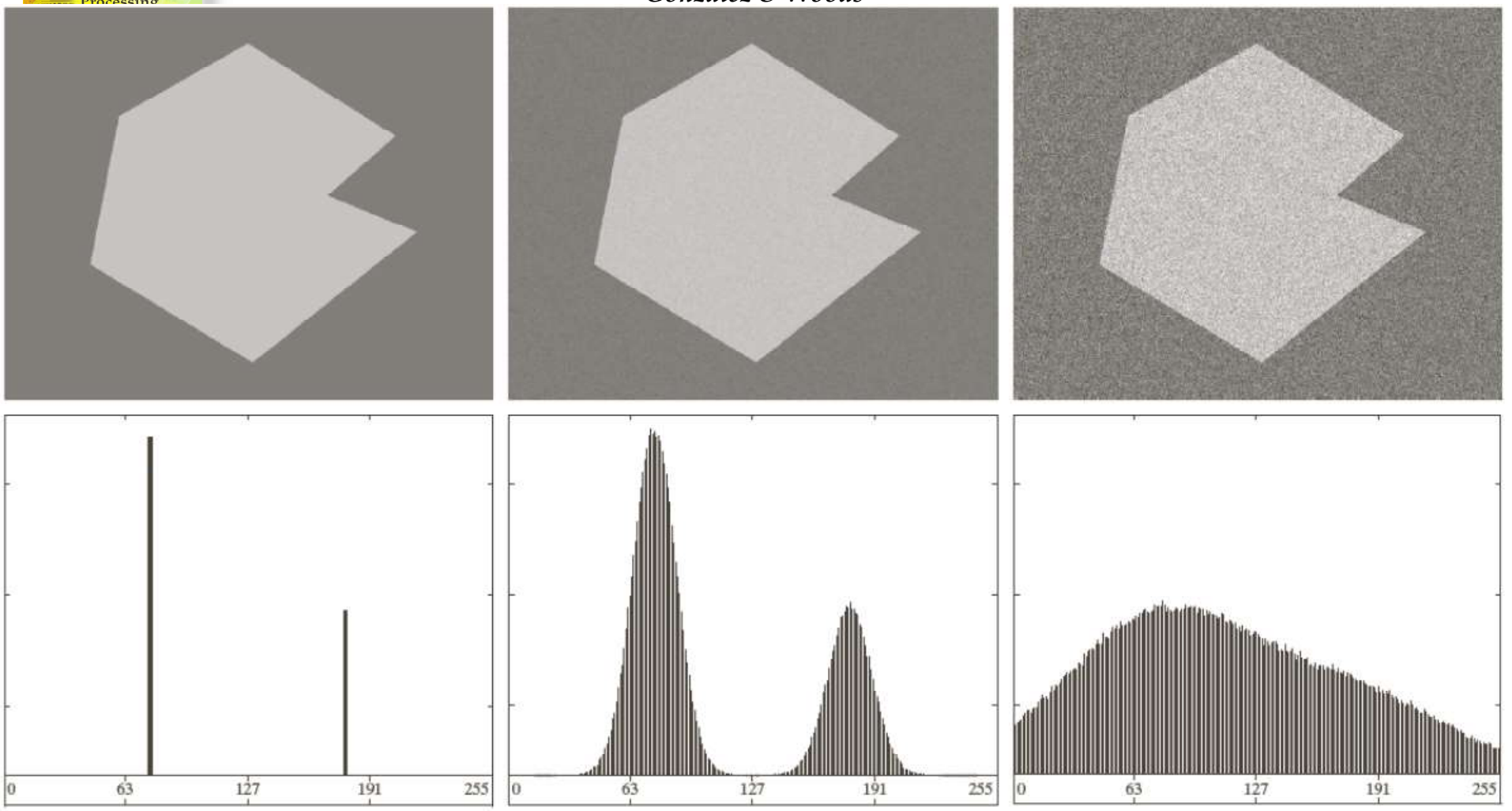


Chapter 10

Segmentation

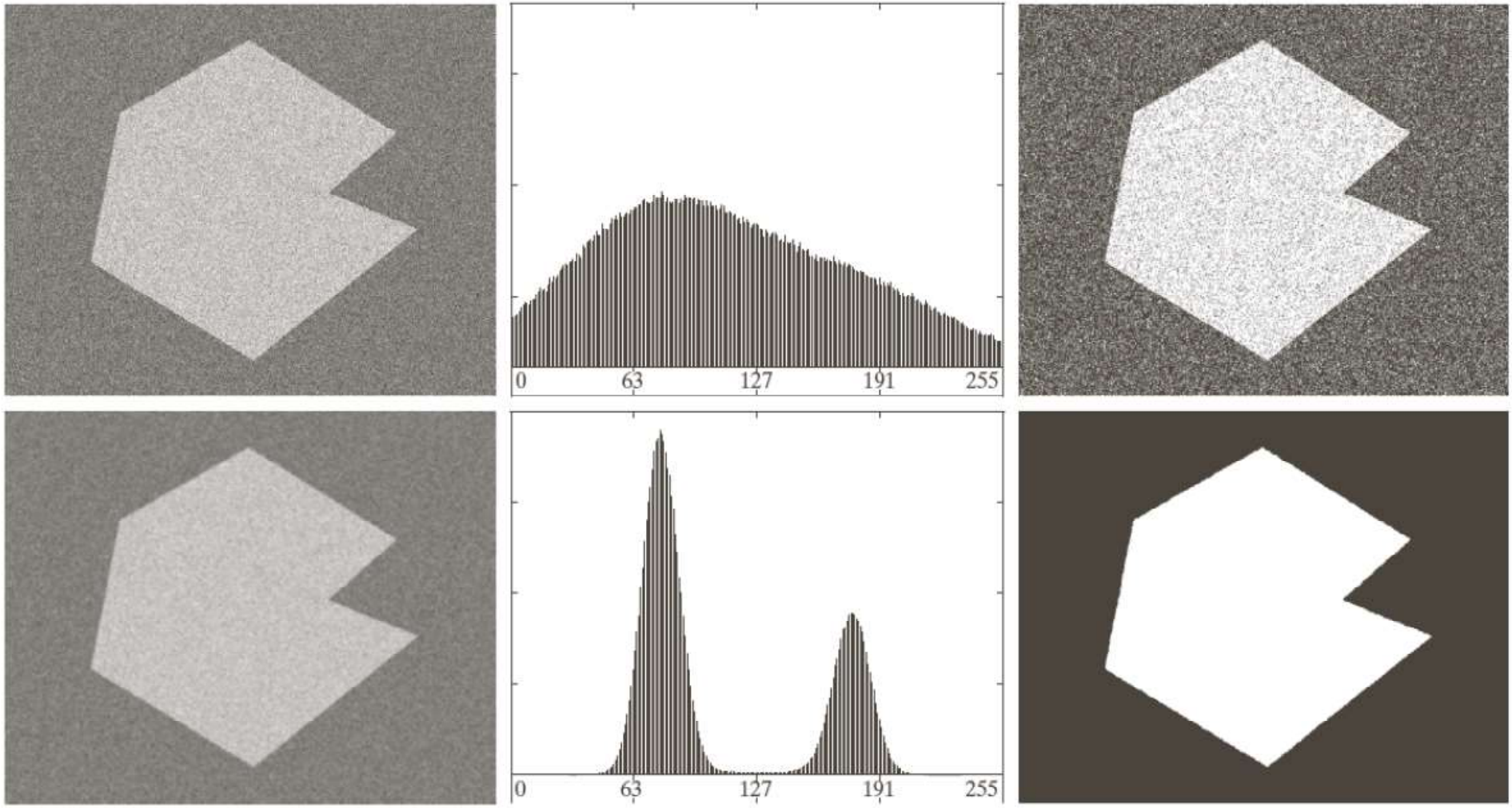
10.3.4 Using Image Smoothing to Improve Global Thresholding

As noted in Fig. 10.36, noise can turn a simple thresholding problem into an unsolvable one. When noise cannot be reduced at the source, and thresholding is the segmentation method of choice, a technique that often enhances performance is to smooth the image prior to thresholding. We illustrate the approach with an example.



a b c
d e f

FIGURE 10.36 (a) Noiseless 8-bit image. (b) Image with additive Gaussian noise of mean 0 and standard deviation of 10 intensity levels. (c) Image with additive Gaussian noise of mean 0 and standard deviation of 50 intensity levels. (d)–(f) Corresponding histograms.



a b c
d e f

FIGURE 10.40 (a) Noisy image from Fig. 10.36 and (b) its histogram. (c) Result obtained using Otsu's method. (d) Noisy image smoothed using a 5 × 5 averaging mask and (e) its histogram. (f) Result of thresholding using Otsu's method.

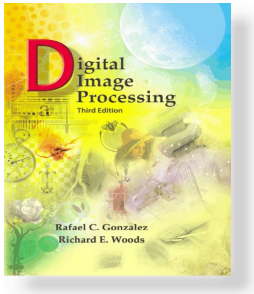
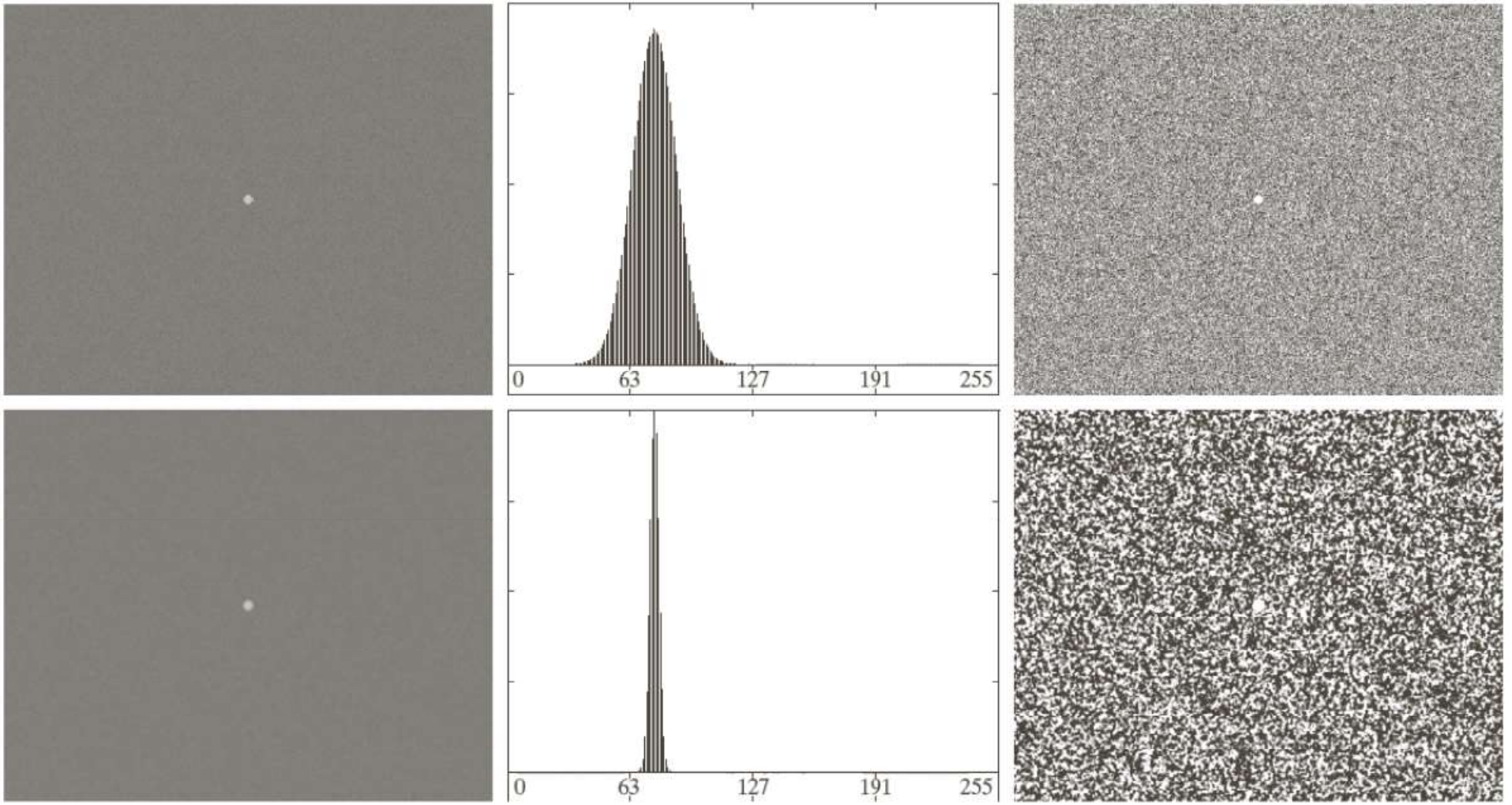


Figure 10.40(a) is the image from Fig. 10.36(c), Fig. 10.40(b) shows its histogram, and Fig. 10.40(c) is the image thresholded using Otsu's method. Every black point in the white region and every white point in the black region is a thresholding error, so the segmentation was highly unsuccessful. Figure 10.40(d) shows the result of smoothing the noisy image with an averaging mask of size 5×5 (the image is of size 651×814 pixels), and Fig. 10.40(e) is its histogram. The improvement in the shape of the histogram due to smoothing is evident, and we would expect thresholding of the smoothed image to be nearly perfect. As Fig. 10.40(f) shows, this indeed was the case. The slight distortion of the boundary between object and background in the segmented, smoothed image was caused by the blurring of the boundary. In fact, the more aggressively we smooth an image, the more boundary errors we should anticipate in the segmented result.



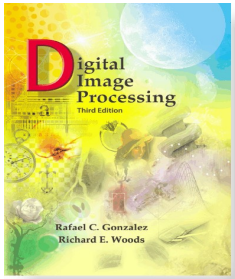
a b c
d e f

FIGURE 10.41 (a) Noisy image and (b) its histogram. (c) Result obtained using Otsu's method. (d) Noisy image smoothed using a 5×5 averaging mask and (e) its histogram. (f) Result of thresholding using Otsu's method. Thresholding failed in both cases.

10.3.5 Using Edges to Improve Global Thresholding

Based on the discussion in the previous four sections, we conclude that the chances of selecting a "good" threshold are enhanced considerably if the histogram peaks are tall, narrow, symmetric, and separated by deep valleys. One approach for improving the shape of histograms is to consider only those pixels that lie on or near the edges between objects and the background. An immediate and obvious improvement is that histograms would be less dependent on the relative sizes of objects and the background. For instance, the histogram of an image composed of a small object on a large background area (or vice versa) would be dominated by a large peak because of the high concentration of one type of pixels. We saw in the previous section that this can lead to failure in thresholding.

If only the pixels on or near the edges between objects and background were used, the resulting histogram would have peaks of approximately the same height. In addition, the probability that any of those pixels lies on an object would be approximately equal to the probability that it lies on the background, thus improving the symmetry of the histogram modes. Finally, as indicated in the following paragraph, using pixels that satisfy some simple measures based on gradient and Laplacian operators has a tendency to deepen the valley between histogram peaks.



10.3.5 Using Edges to Improve Global Thresholding

The approach just discussed assumes that the edges between objects and background are known. This information clearly is not available during segmentation, as finding a division between objects and background is precisely what segmentation is all about. However, with reference to the discussion in Section 10.2, an indication of whether a pixel is on an edge may be obtained by computing its gradient or Laplacian. For example, the average value of the Laplacian is 0 at the transition of an edge (see Fig. 10.10), so the valleys of histograms formed from the pixels selected by a Laplacian criterion can be expected to be sparsely populated. This property tends to produce the desirable deep valleys discussed above. In practice, comparable results typically are obtained using either the gradient or Laplacian images, with the latter being favored because it is computationally more attractive and is also an isotropic edge detector.

10.3.5 Using Edges to Improve Global Thresholding

The preceding discussion is summarized in the following algorithm, where $f(x, y)$ is the input image:

1. Compute an edge image as either the magnitude of the gradient, or absolute value of the Laplacian, of $f(x, y)$ using any of the methods discussed in Section 10.2.
2. Specify a threshold value, T.
3. Threshold the image from Step 1 using the threshold from Step 2 to produce a binary image, $g_T(x, y)$. This image is used as a *mask image* in the following step to select pixels from $f(x, y)$ corresponding to "strong" edge pixels.
4. Compute a histogram using only the pixels in $f(x, y)$ that correspond to the locations of the 1-valued pixels in $g_T(x, y)$.
5. Use the histogram from Step 4 to segment $f(x, y)$ globally using, for example, Otsu's method.

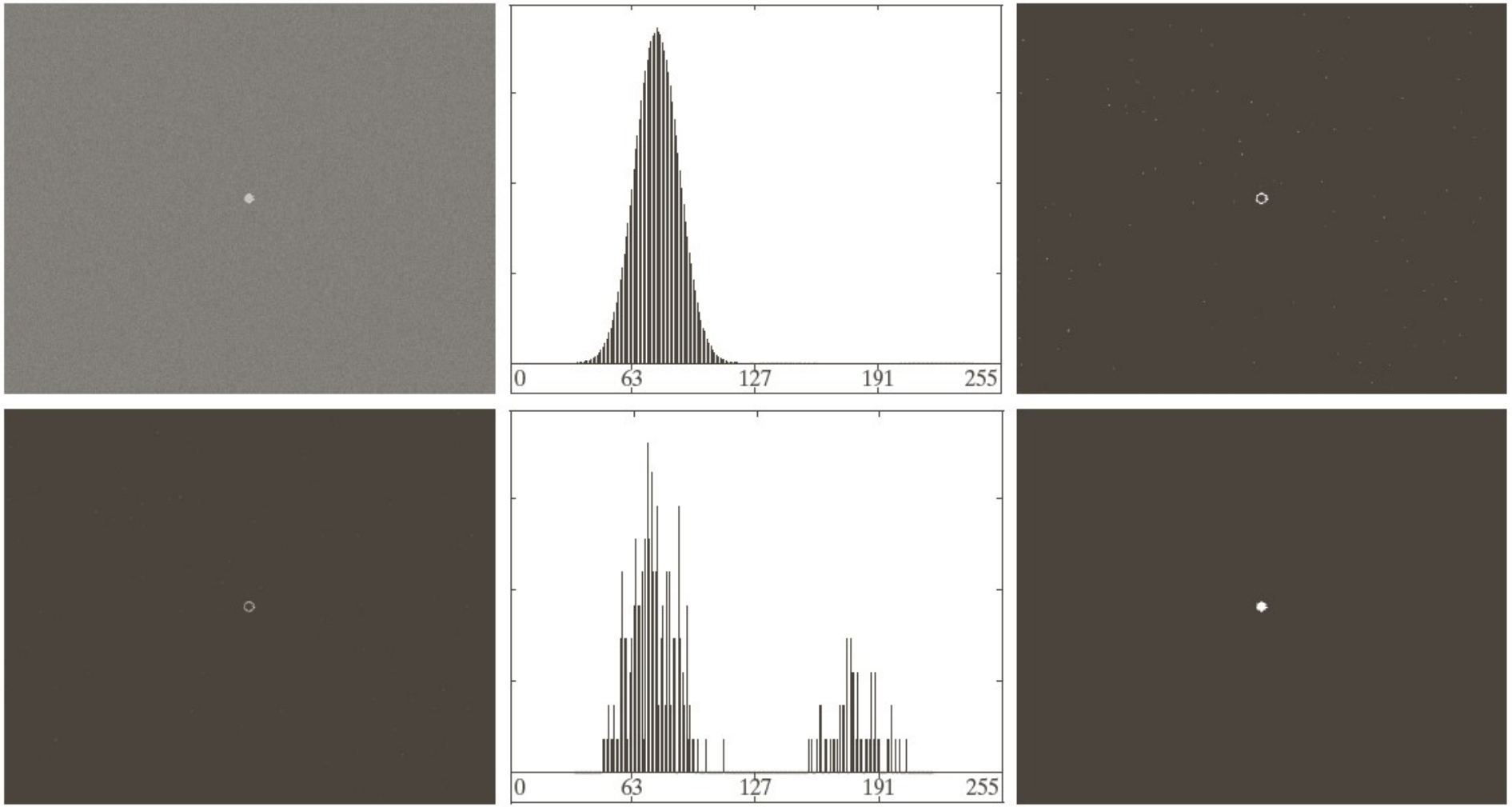


FIGURE 10.42 (a) Noisy image from Fig. 10.41(a) and (b) its histogram. (c) Gradient magnitude image thresholded at the 99.7 percentile. (d) Image formed as the product of (a) and (c). (e) Histogram of the nonzero pixels in the image in (d). (f) Result of segmenting image (a) with the Otsu threshold based on the histogram in (e). The threshold was 134, which is approximately midway between the peaks in this histogram.

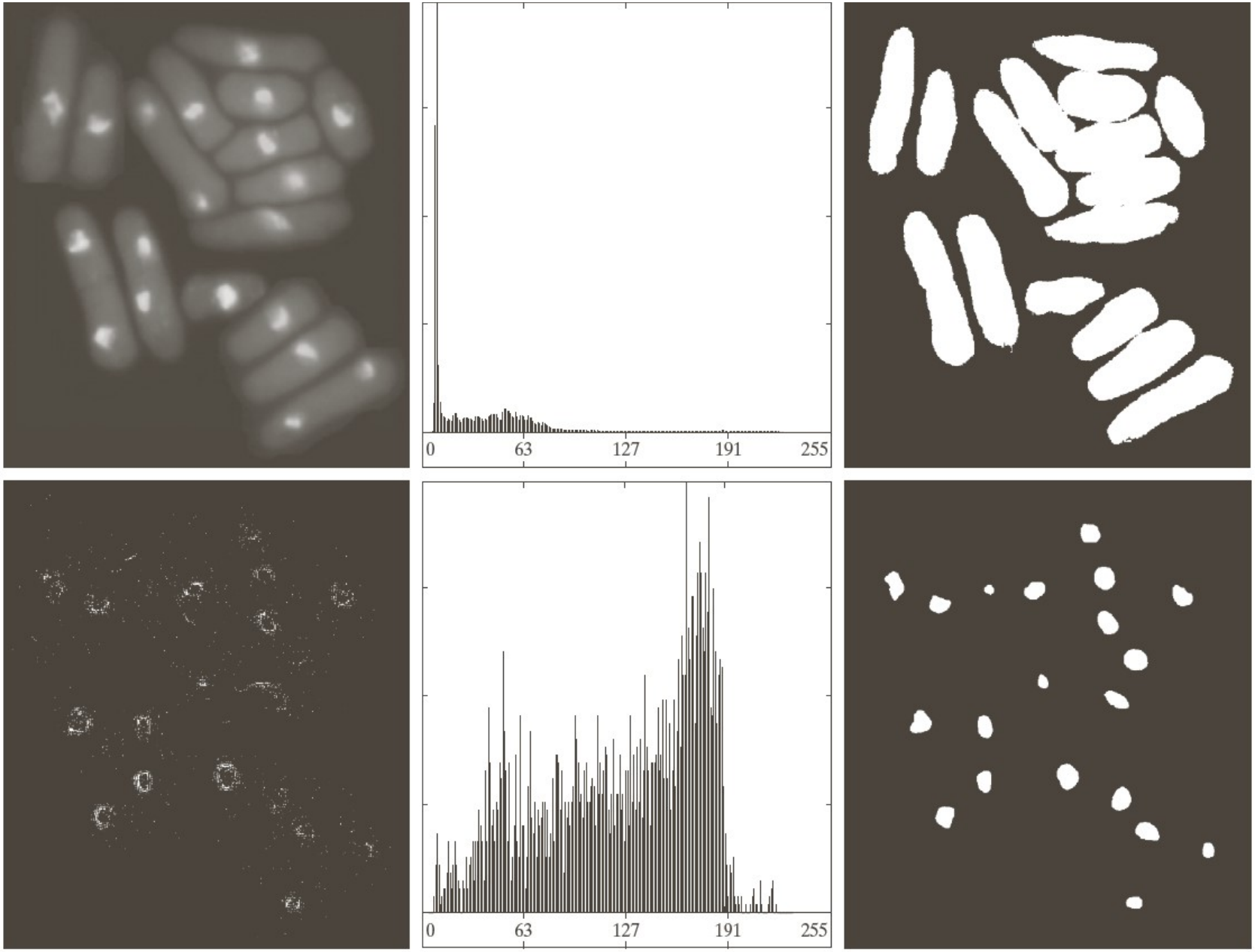


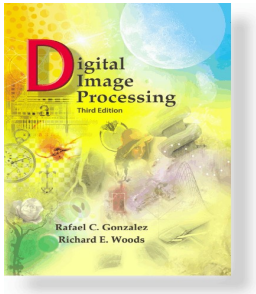
FIGURE 10.43 (a) Image of yeast cells. (b) Histogram of (a). (c) Segmentation of (a) with Otsu's method using the histogram in (b). (d) Thresholded absolute Laplacian. (e) Histogram of the nonzero pixels in the product of (a) and (d). (f) Original image thresholded using Otsu's method based on the histogram in (e). (Original image courtesy of Professor Susan L. Forsburg, University of Southern California.)

In this example we consider a more complex thresholding problem. Figure 10.43(a) shows an 8-bit image of yeast cells in which we wish to use global thresholding to obtain the regions corresponding to the bright spots. As a starting point, Fig. 10.43(b) shows the image histogram, and Fig. 10.43(c) is the result obtained using Otsu's method directly on the image, using the histogram shown. We see that Otsu's method failed to achieve the original objective of detecting the bright spots, and, although the method was able to isolate some of the cell regions themselves, several of the segmented regions on the right are not disjoint. The threshold computed by the Otsu method was 42 and the separability measure was 0.636.

Figure 10.43(d) shows the image $g_T(x, y)$ obtained by computing the absolute value of the Laplacian image and then thresholding it with T set to 115 on an intensity scale in the range $[0, 255]$. This value of T corresponds approximately to the 99.5 percentile of the values in the absolute Laplacian image, so thresholding at this level should result in a sparse set of pixels, as Fig. 10.43(d) shows. Note in this image how the points cluster near the edges of the bright spots, as expected from the preceding discussion. Figure 10.43(e) is the histogram of the nonzero pixels in the product of (a) and (d). Finally, Fig. 10.43(f) shows the re-

Figure 10.43(e) is the histogram of the nonzero pixels in the product of (a) and (d). Finally, Fig. 10.43(f) shows the result of globally segmenting the original image using Otsu's method based on the histogram in Fig. 10.43(e). This result agrees with the locations of the bright spots in the image. The threshold computed by the Otsu method was 115 and the separability measure was 0.762, both of which are higher than the values obtained by using the original histogram.

By varying the percentile at which the threshold is set we can even improve on the segmentation of the cell regions. For example, Fig. 10.44 shows the result obtained using the same procedure as in the previous paragraph, but with the threshold set at 55, which is approximately 5% of the maximum value of the absolute Laplacian image. This value is at the 53.9 percentile of the values in that image. This result clearly is superior to the result in Fig. 10.43(c) obtained using Otsu's method with the histogram of the original image.



Chapter 10

Segmentation

FIGURE 10.44
Image in Fig. 10.43(a) segmented using the same procedure as explained in Figs. 10.43(d)–(f), but using a lower value to threshold the absolute Laplacian image.

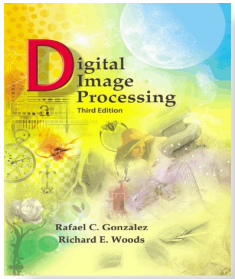
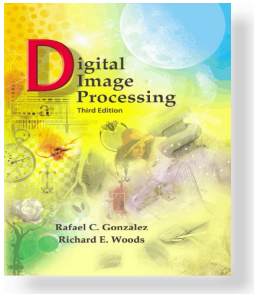
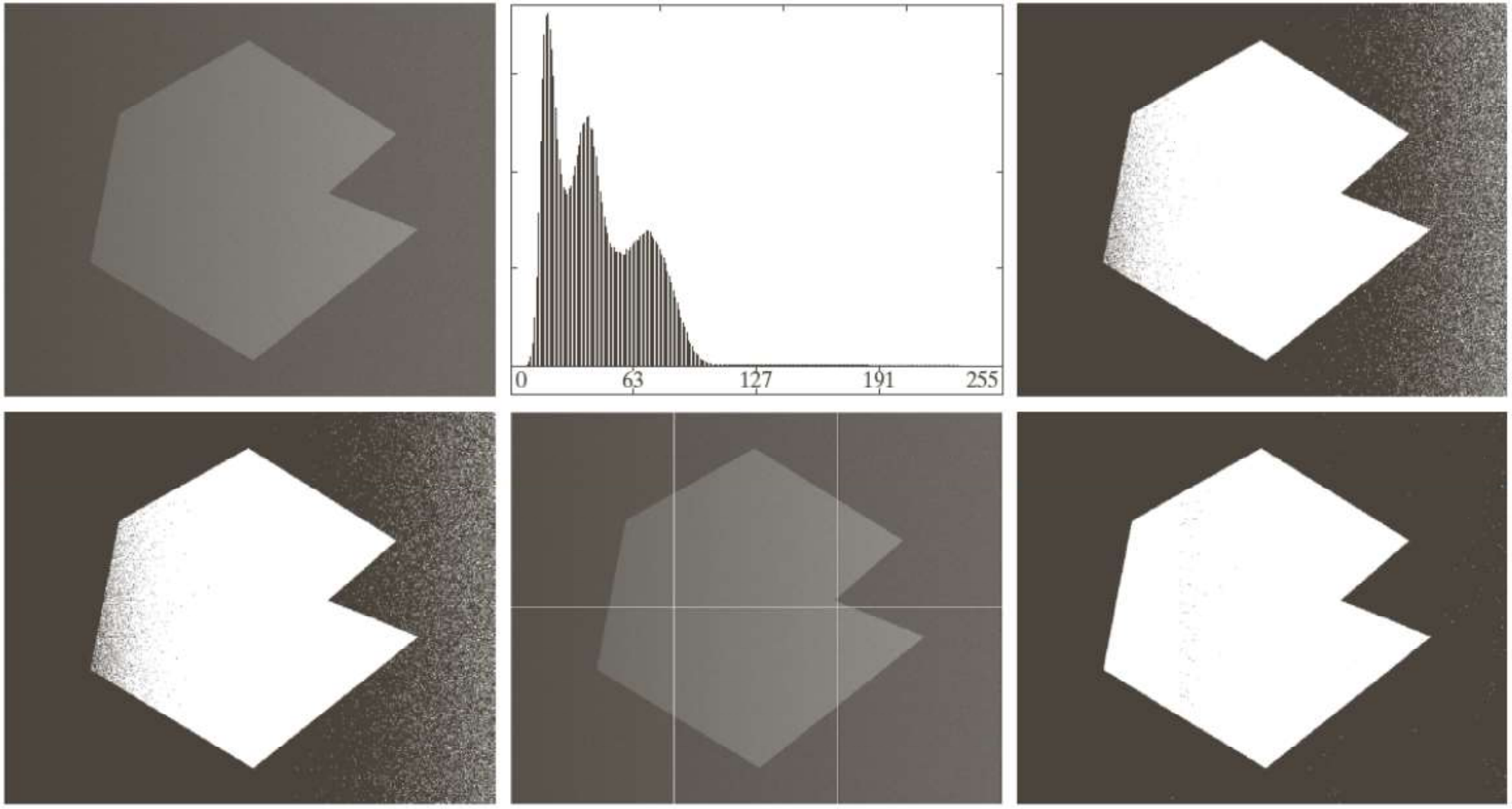


Figure 10.45(a) shows an image of an iceberg. The objective of this example is to segment the image into three regions: the dark background, the illuminated area of the iceberg, and the area in shadows. It is evident from the image histogram in Fig. 10.45(b) that two thresholds are required to solve this problem. The procedure discussed above resulted in the thresholds $k_1^* = 80$ and $k_2^* = 177$, which we note from Fig. 10.45(b) are near the centers of the two histogram valleys. Figure 10.45(c) is the segmentation that resulted using these two thresholds in Eq. (10.3-31). The separability measure was 0.954. The principal reason this example worked out so well can be traced to the histogram having three distinct modes separated by reasonably wide, deep valleys.



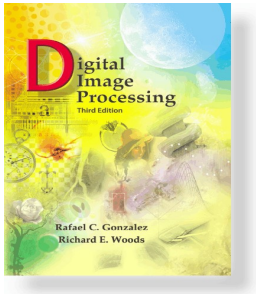
Variable Thresholding

As discussed in Section 10.3.1, factors such as noise and nonuniform illumination play a major role in the performance of a thresholding algorithm. We showed in Sections 10.3.4 and 10.3.5 that image smoothing and using edge information can help significantly. However, it frequently is the case that this type of preprocessing is either impractical or simply ineffective in improving the situation to the point where the problem is solvable by any of the methods discussed thus far. In such situations, the next level of thresholding complexity involves variable thresholding. In this section, we discuss various techniques for choosing variable thresholds.



a b c
d e f

FIGURE 10.46 (a) Noisy, shaded image and (b) its histogram. (c) Segmentation of (a) using the iterative global algorithm from Section 10.3.2. (d) Result obtained using Otsu's method. (e) Image subdivided into six subimages. (f) Result of applying Otsu's method to each subimage individually.



Chapter 10

Segmentation

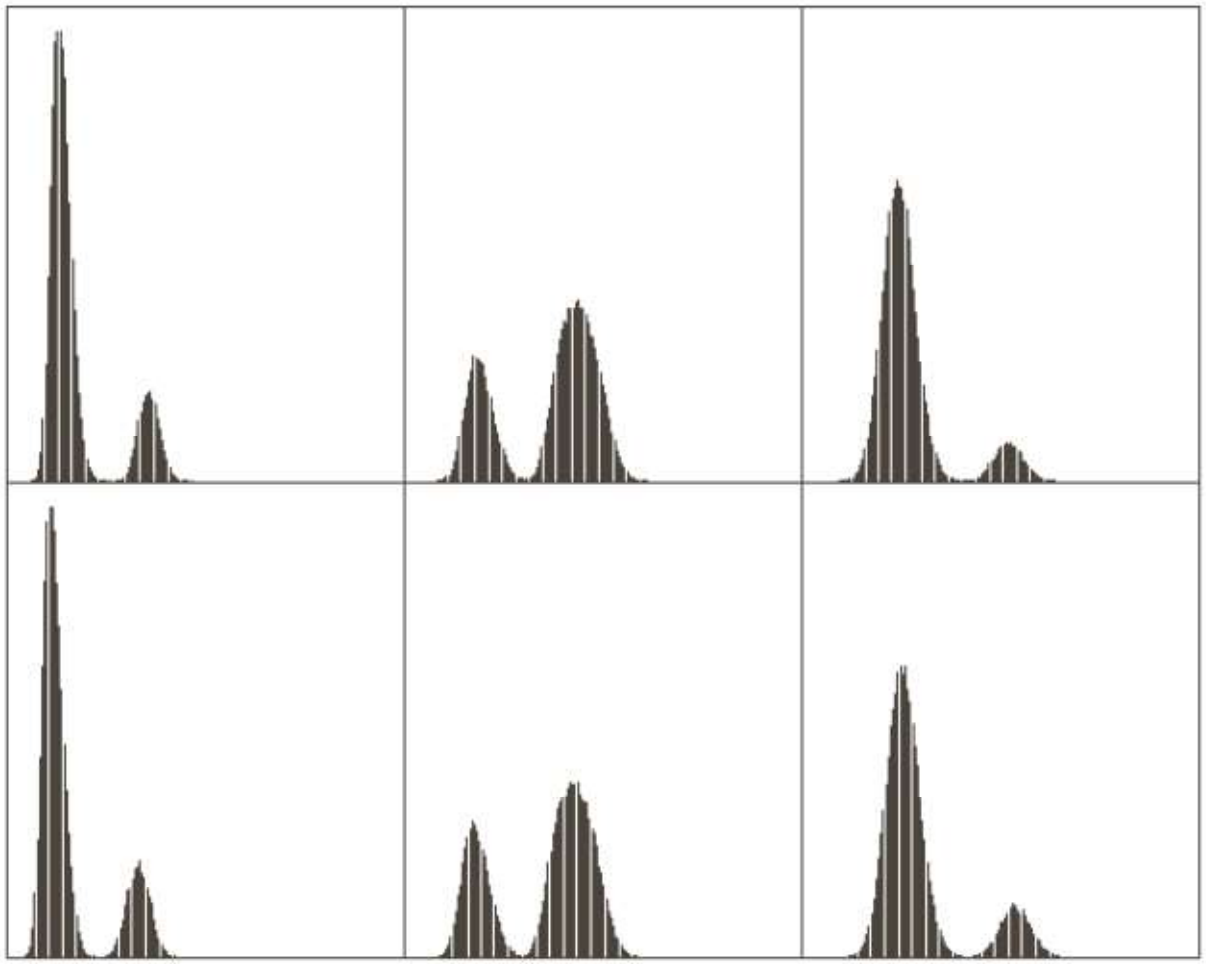
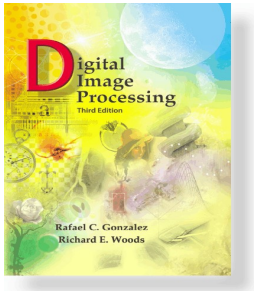


FIGURE 10.47 Histograms of the six subimages in Fig. 10.46(e).

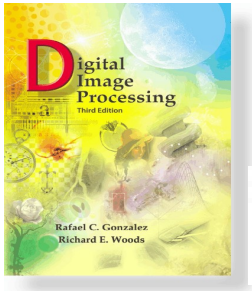


Variable Thresholding

Image partitioning

One of the simplest approaches to variable thresholding is to subdivide an image into nonoverlapping rectangles. This approach is used to compensate for non-uniformities in illumination and/or reflectance. The rectangles are chosen small enough so that the illumination of each is approximately uniform. We illustrate this approach with an example.

Image subdivision generally works well when the objects of interest and the background occupy regions of reasonably comparable size, as in Fig. 10.46. When this is not the case, the method typically fails because of the likelihood of subdivisions containing only object or background pixels. Although this situation can be addressed by using additional techniques to determine when a subdivision contains both types of pixels, the logic required to address different scenarios can get complicated. In such situations, methods such as those discussed in the remainder of this section typically are preferable. ■



Variable thresholding based on local image properties

A more general approach than the image subdivision method discussed in the previous section is to compute a threshold at every point, (x, y), in the image based on one or more specified properties computed in a neighborhood of (x, y). Although this may seem like a laborious process, modern algorithms and hardware allow for fast neighborhood processing, especially for common functions such as logical and arithmetic operations.

Examples: **(1)** $T_{xy} = a\,\sigma_{xy} + b\,m_{xy}$ **(2)** $T_{xy} = a\,\sigma_{xy} + b\,m_G$

where a and b are nonnegative constants
m_G is the global image mean.
m_{xy} mean value and
σ_{xy} the standard deviation
of the set of pixels contained in a neighborhood, S_{xy}, centered at coordinates (x, y) in an image

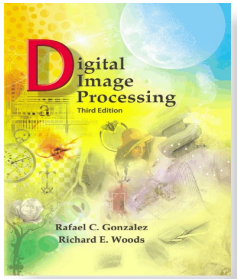
$$g(x,y) = \begin{cases} 1, & \textbf{if } f(x,y) > T_{xy} \\ 0, & \textbf{if } f(x,y) \leq T_{xy} \end{cases}$$

OR

$$g(x, y) = \begin{cases} 1 & \text{if } Q(\text{local parameters}) \text{ is true} \\ 0 & \text{if } Q(\text{local parameters}) \text{ is false} \end{cases} \qquad (10.3\text{-}36)$$

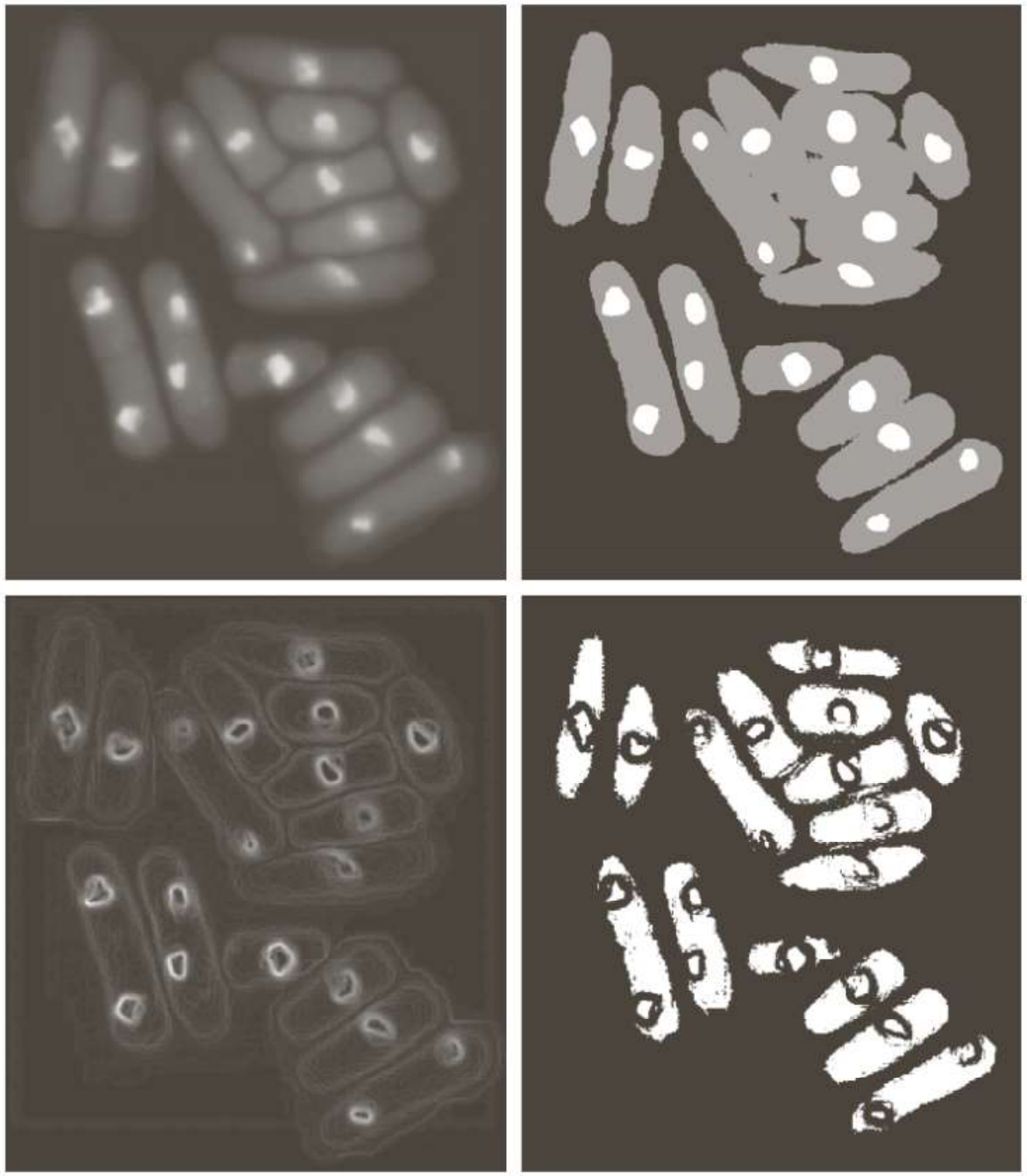
where Q is a *predicate* based on parameters computed using the pixels in neighborhood S_{xy}. For example, consider the following predicate, $Q(\sigma_{xy}, m_{xy})$, based on the local mean and standard deviation:

$$Q(\sigma_{xy}, m_{xy}) = \begin{cases} \text{true} & \text{if } f(x, y) > a\sigma_{xy} \text{ AND } f(x, y) > bm_{xy} \\ \text{false} & \text{otherwise} \end{cases} \qquad (10.3\text{-}37)$$

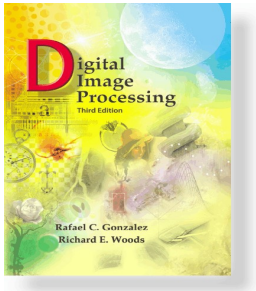


a b
c d

FIGURE 10.48
(a) Image from Fig. 10.43. (b) Image segmented using the dual thresholding approach discussed in Section 10.3.6. (c) Image of local standard deviations. (d) Result obtained using local thresholding.



■ Figure 10.48(a) shows the yeast image from Example 10.18. This image has three predominant intensity levels, so it is reasonable to assume that perhaps dual thresholding could be a good segmentation approach. Figure 10.48(b) is the result of using the dual thresholding method explained in Section 10.3.6. As the figure shows, it was possible to isolate the bright areas from the background, but the mid-gray regions on the right side of the image were not segmented properly (recall that we encountered a similar problem with Fig. 10.43(c) in Example 10.18). To illustrate the use of local thresholding, we computed the local standard deviation σ_{xy} for all (x, y) in the input image using a neighborhood of size 3×3. Figure 10.48(c) shows the result. Note how the faint outer lines correctly delineate the boundaries of the cells. Next, we formed a predicate of the form shown in Eq. (10.3-37) but using the global mean instead of m_{xy}. Choosing the global mean generally gives better results when the background is nearly constant and all the object intensities are above or below the background intensity. The values $a = 30$ and $b = 1.5$ were used in completing the specification of the predicate (these values were determined experimentally, as is usually the case in applications such as this). The image was then segmented using Eq. (10.3-36). As Fig. 10.48(d) shows, the result agrees quite closely with the two types of intensity regions prevalent in the input image. Note in particular that all the outer regions were segmented properly and that most of the inner, brighter regions were isolated correctly. ■

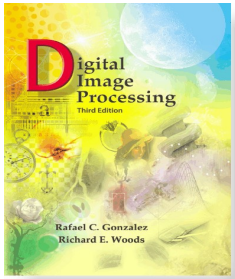


Chapter 10
Segmentation

Point, Line and Edge Detection

- Based on detecting sharp, LOCAL changes in intensity
- 3 prominent types of image features
  - Points, Lines and Edges

- **Edge pixels:**
  - Pixels at which intensity value changes abruptly
- **Edges**
  - Sets of connected edge pixels
- **Edge detectors**
  - Local image processing methods designed to detect edge pixels



Chapter 10

Segmentation

10.2 Point, line and edge detection

10.2.1 Background

$$\frac{\partial f}{\partial x} = f'(x) = f(x+1) - f(x)$$

$$\begin{aligned}\frac{\partial^2 f}{\partial x^2} &= \frac{\partial f'(x)}{\partial x} \\ &= f'(x+1) - f'(x) \\ &= f(x+2) - f(x+1) - f(x+1) + f(x) \\ &= f(x+2) - 2f(x+1) + f(x)\end{aligned}$$

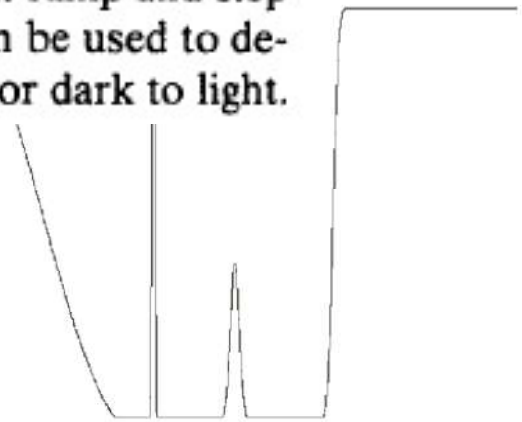
The above represents a truncated Taylor expansion about the point $x+1$

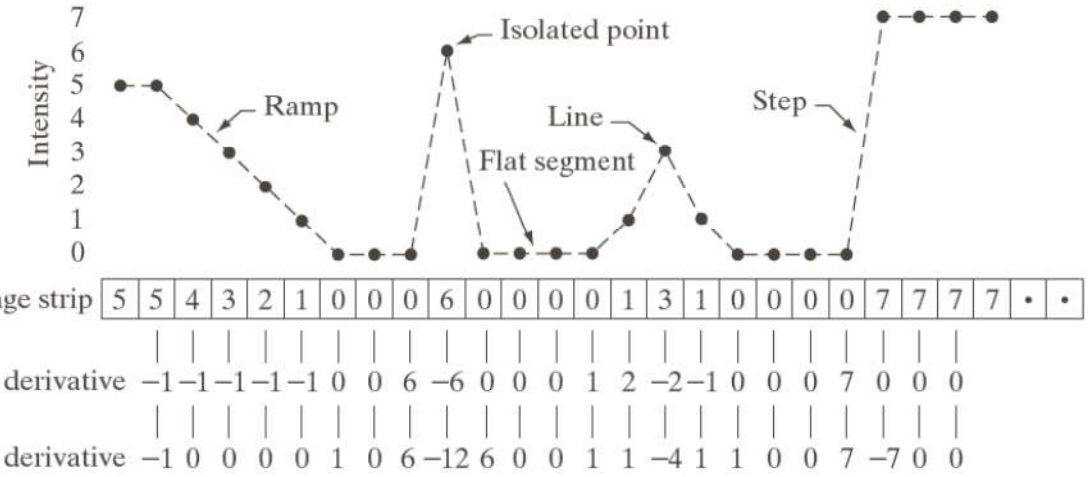
Expansion about point x:

$$\frac{\partial^2 f}{\partial x^2} = f''(x) = f(x+1) + f(x-1) - 2f(x)$$

Credits: H. Coetzer, Department of Applied Mathematics, Stellenbosch University, S. Africa

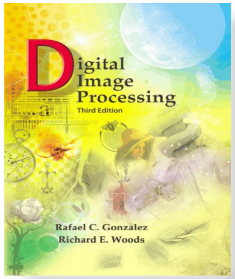
In summary, we arrive at the following conclusions: (1) First-order derivatives generally produce thicker edges in an image. (2) Second-order derivatives have a stronger response to fine detail, such as thin lines, isolated points, and noise. (3) Second-order derivatives produce a double-edge response at ramp and step transitions in intensity. (4) The sign of the second derivative can be used to determine whether a transition into an edge is from light to dark or dark to light.





a b
c

FIGURE 10.2 (a) Image. (b) Horizontal intensity profile through the center of the image, including the isolated noise point. (c) Simplified profile (the points are joined by dashes for clarity). The image strip corresponds to the intensity profile, and the numbers in the boxes are the intensity values of the dots shown in the profile. The derivatives were obtained using Eqs. (10.2-1) and (10.2-2).



Chapter 10

Segmentation

General linear filter: (Response)

$$\begin{aligned} R &= w_1 z_1 + w_2 z_2 + \ldots + w_{mn} z_{mn} \\ &= \sum_{k=1}^{mn} w_k z_k \\ &= \mathbf{w}^T \mathbf{z} \end{aligned}$$

3x3 example: (Response)

$$\begin{aligned} R &= w_1 z_1 + w_2 z_2 + \ldots + w_9 z_9 \\ &= \sum_{k=1}^{9} w_k z_k \\ &= \mathbf{w}^T \mathbf{z} \end{aligned}$$

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Credits: H. Coetzer, Department of Applied Mathematics, Stellenbosch University, S. Africa

10.2.2 Detection of isolated points

Laplacian derivative operator

Continuous form:

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

Discrete form: x**-direction**

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y)$$

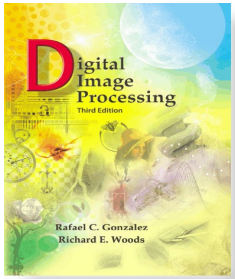
Discrete form: y**-direction**

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y)$$

Discrete form: 2-D Laplacian - sum of the two components

$$\nabla^2 f(x, y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

$$g(x, y) = \begin{cases} 1, & \textbf{if } |R(x, y)| \geq T \\ 0, & \textbf{otherwise} \end{cases}$$



Digital Image Processing, 3rd ed.

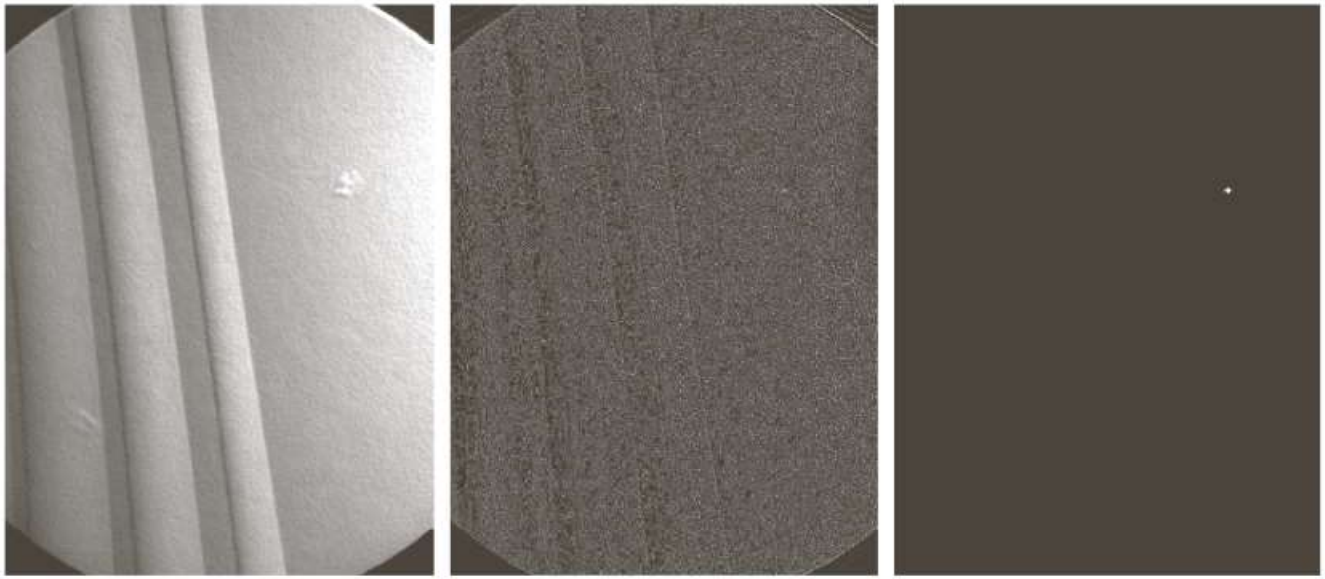
Gonzalez & Woods
www.ImageProcessingPlace.com

Chapter 10
Segmentation

1	1	1
1	−8	1
1	1	1

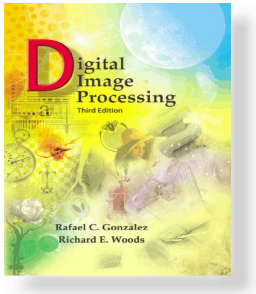
$$g(x,y) = \begin{cases} 1, & \textbf{if } |R(x,y)| \geq T \\ 0, & \textbf{otherwise} \end{cases}$$

Intuitively, the idea is that the intensity of an isolated point will be quite different from its surroundings and thus will be easily detectable.



a
b c d

FIGURE 10.4 (a) Point detection (Laplacian) mask. (b) X-ray image of turbine blade with a porosity. The porosity contains a single black pixel. (c) Result of convolving the mask with the image. (d) Result of using Eq. (10.2-8) showing a single point (the point was enlarged to make it easier to see). (Original image courtesy of X-TEK Systems, Ltd.)

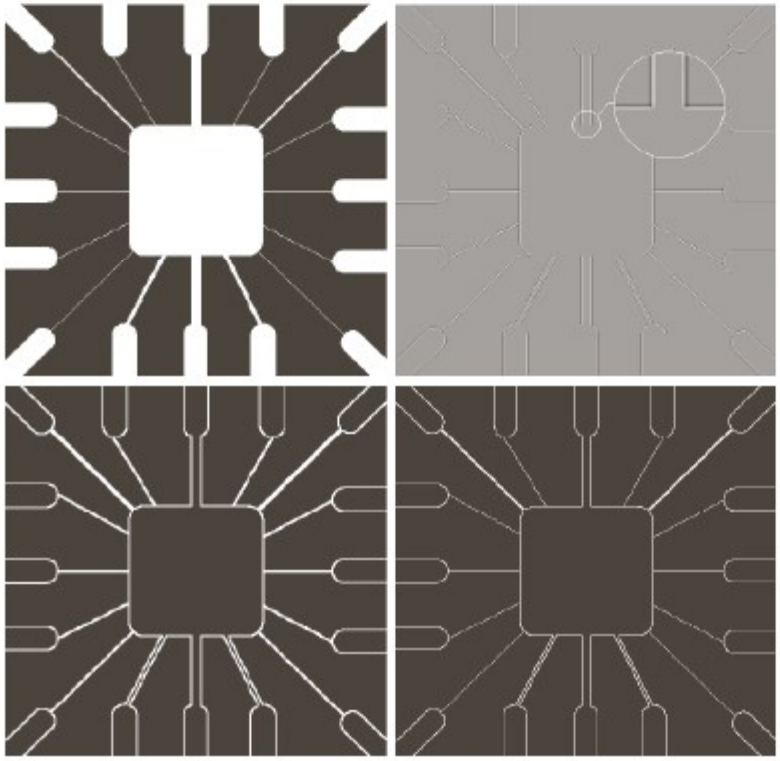


Chapter 10

Segmentation

10.2.3 Line detection

Laplacian is isotropic: R independent of direction



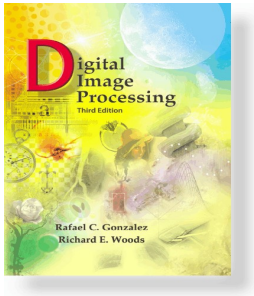
a b
c d

FIGURE 10.5
(a) Original image. (b) Laplacian image; the magnified section shows the positive/negative double-line effect characteristic of the Laplacian. (c) Absolute value of the Laplacian. (d) Positive values of the Laplacian.

Laplacian mask can(is generally) used for line detection but double line effect of the second derivative to be handled carefully

Detection of lines in a specified direction: If $|R_i| \geq |R_j| \; \forall \; j \neq i$ the point in question probably lies on a line that is detected by mask i

Credits: H. Coetzer, Department of Applied Mathematics, Stellenbosch University, S. Africa

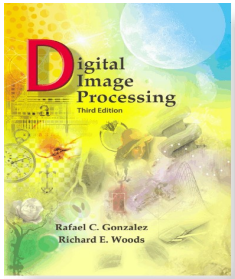


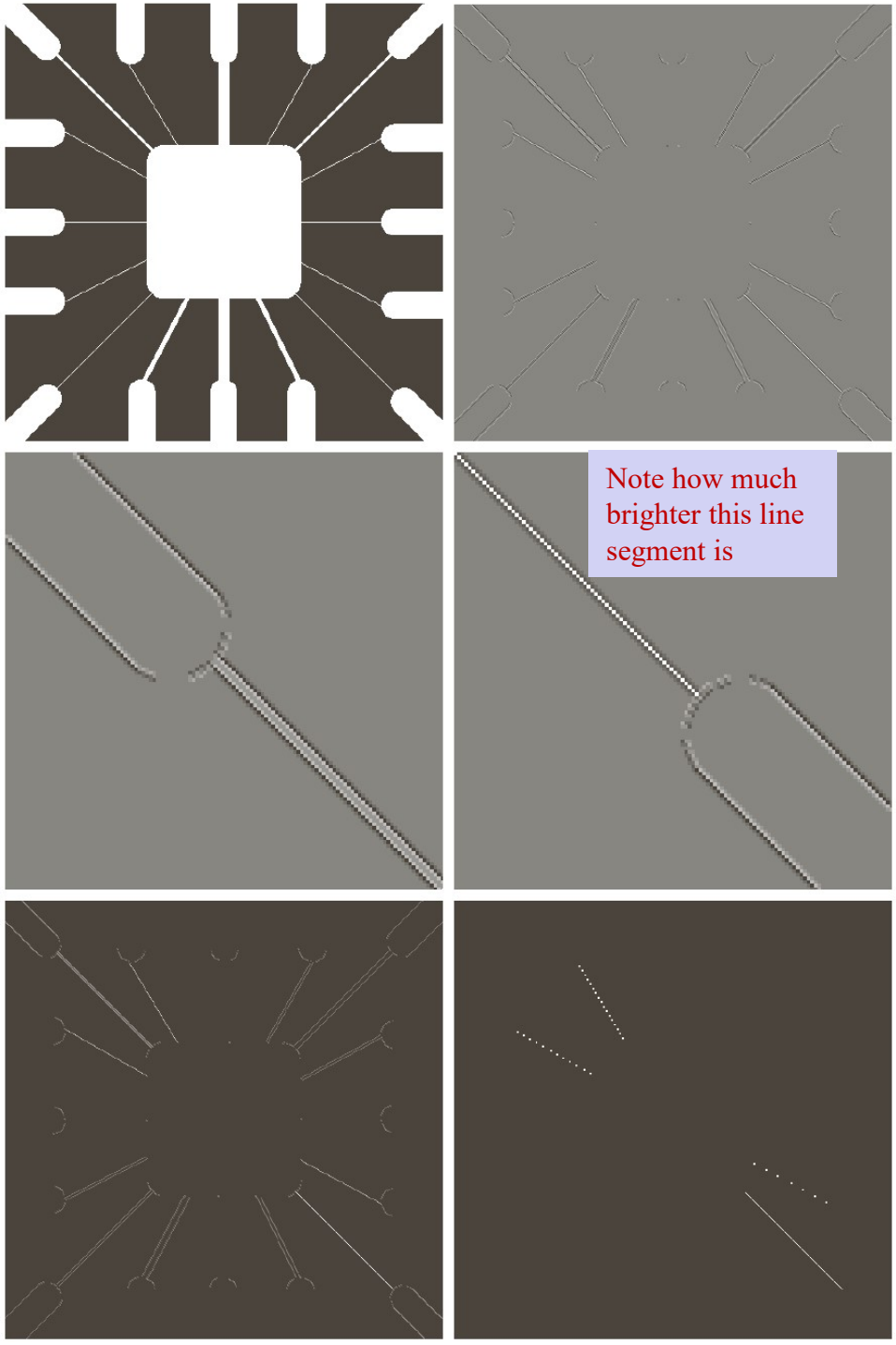
Chapter 10

Segmentation

−1	−1	−1	2	−1	−1	−1	2	−1	−1	−1	2
2	2	2	−1	2	−1	−1	2	−1	−1	2	−1
−1	−1	−1	−1	−1	2	−1	2	−1	2	−1	−1
Horizontal			+45°			Vertical			−45°		

FIGURE 10.6 Line detection masks. Angles are with respect to the axis system in Fig. 2.18(b).



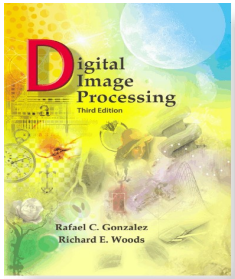


a b
c d
e f

FIGURE 10.7 (a) Image of a wire-bond template. (b) Result of processing with the +45° line detector mask in Fig. 10.6. (c) Zoomed view of the top left region of (b). (d) Zoomed view of the bottom right region of (b). (e) The image in (b) with all negative values set to zero. (f) All points (in white) whose values satisfied the condition $g \geq T$, where g is the image in (e). (The points in (f) were enlarged to make them easier to see.)

Note how much brighter this line segment is

Isolated points in last figure can be removed using morphological operations also



Chapter 10

Segmentation

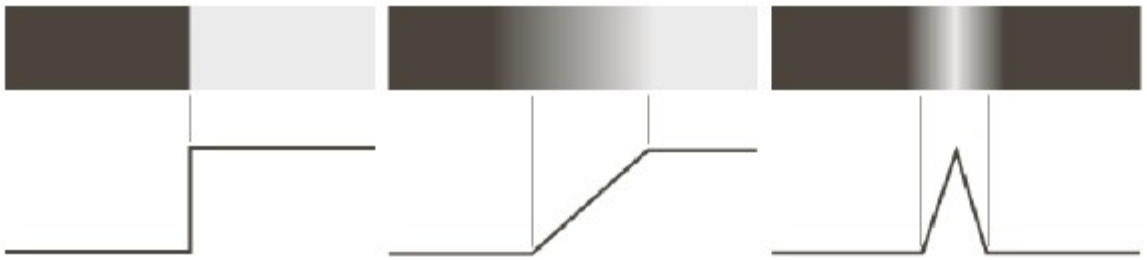
10.2.4 Edge models

Classified according to their intensity profiles:

(1) Step model (used to derive Canny algorithm [Section 10.2.6])

(2) Ramp model (slope inversely proportional to degree of blurring)

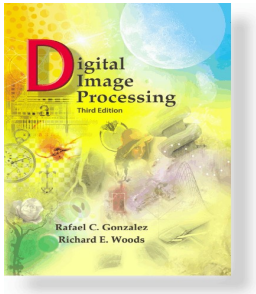
(3) Roof edge (models line through a region)



a b c

FIGURE 10.8 From left to right, models (ideal representations) of a step, a ramp, and a roof edge, and their corresponding intensity profiles.

Credits: H. Coetzer, Department of Applied Mathematics, Stellenbosch University, S. Africa

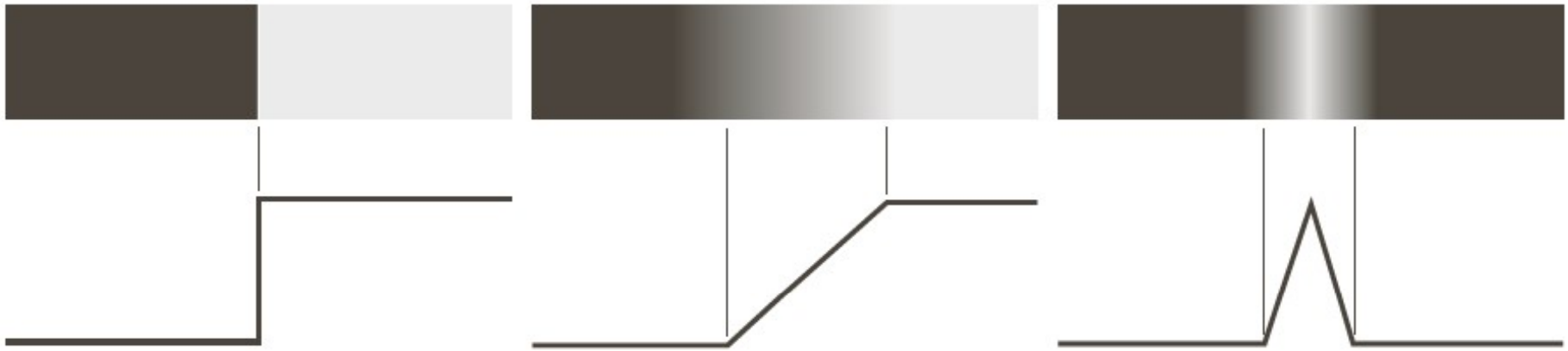


Digital Image Processing, 3rd ed.

Gonzalez & Woods

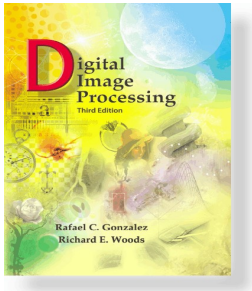
www.ImageProcessingPlace.com

Chapter 10
Segmentation



a b c

FIGURE 10.8
From left to right, models (ideal representations) of a step, a ramp, and a roof edge, and their corresponding intensity profiles.



Example: Image that contains all three types of edges

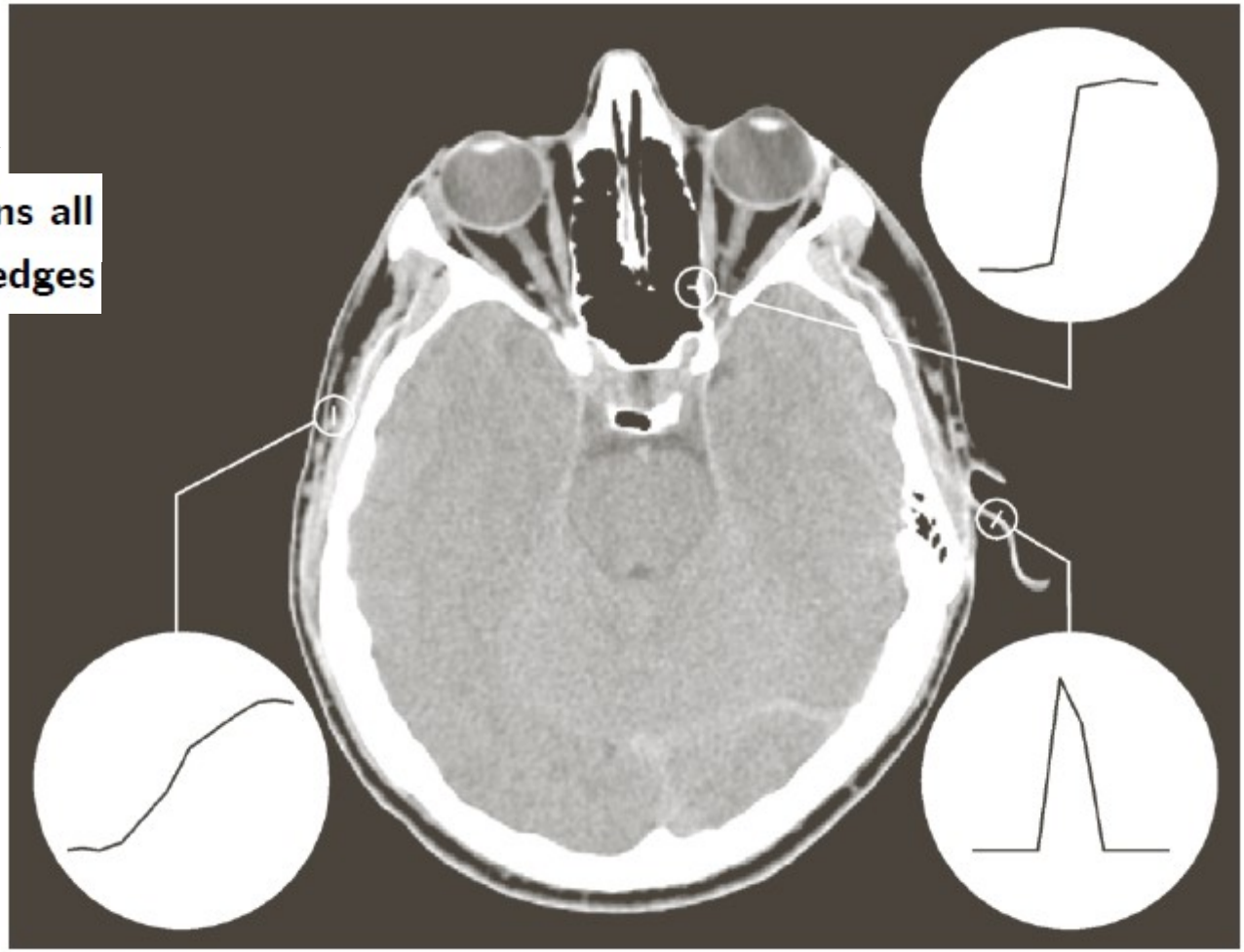
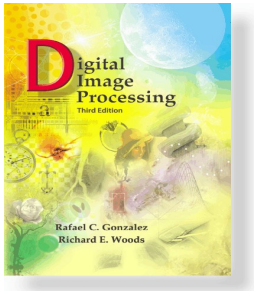


FIGURE 10.9 A 1508 × 1970 image showing (zoomed) actual ramp (bottom, left), step (top, right), and roof edge profiles. The profiles are from dark to light, in the areas indicated by the short line segments shown in the small circles. The ramp and "step" profiles span 9 pixels and 2 pixels, respectively. The base of the roof edge is 3 pixels. (Original image courtesy of Dr. David R. Pickens, Vanderbilt University.)

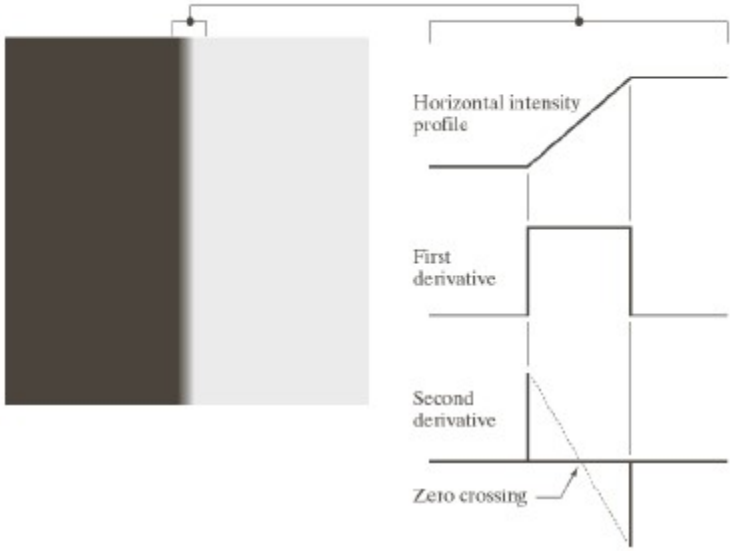


Digital Image Processing, 3rd ed.

Gonzalez & Woods

www.ImageProcessingPlace.com

Chapter 10
Segmentation



a b

FIGURE 10.10
(a) Two regions of constant intensity separated by an ideal vertical ramp edge. (b) Detail near the edge, showing a horizontal intensity profile, together with its first and second derivatives.

Observations:

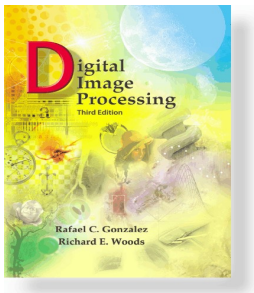
(1) Magnitude of first derivative detect presence of edge

(2) Sign of second derivative determines whether edge pixel is on dark or light side

(3) Second derivative produces two values for every edge (undesirable)

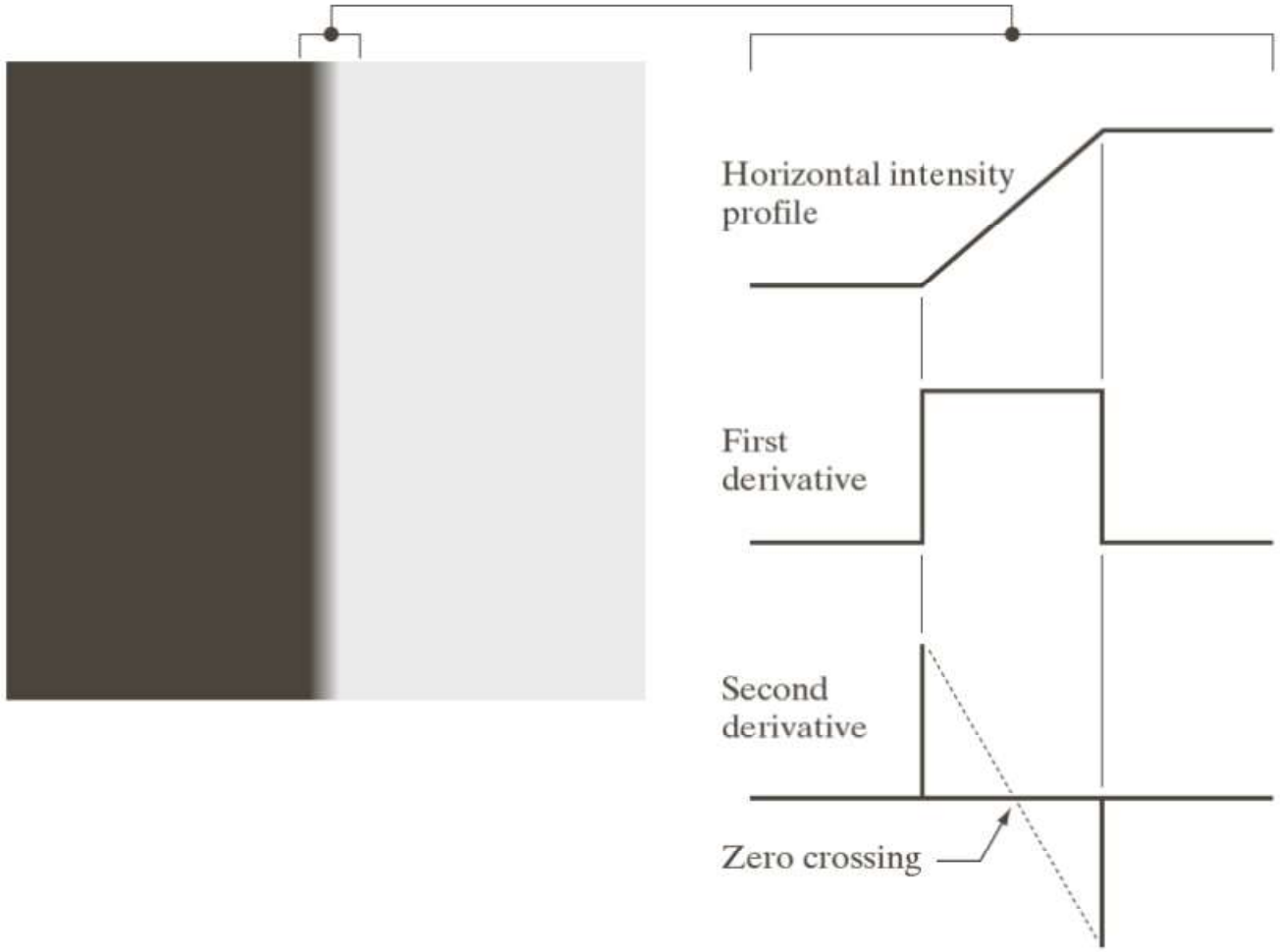
(4) The zero crossings of (3) can be used to locate centers of thick edges

Credits: H. Coetzer, Department of Applied Mathematics, Stellenbosch University, S. Africa



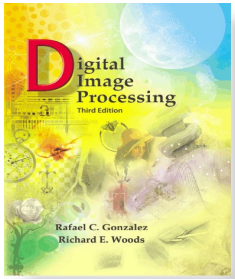
Chapter 10

Segmentation



a b

FIGURE 10.10
(a) Two regions of constant intensity separated by an ideal vertical ramp edge. (b) Detail near the edge, showing a horizontal intensity profile, together with its first and second derivatives.



Chapter 10

Segmentation

Conclusion: Three fundamental steps in edge detection

(1) Image smoothing for noise reduction

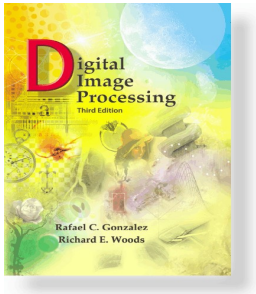
(2) Detection of edge points

- Find potential candidates to become edge points

(3) Edge localization

- Select true edge points from candidates

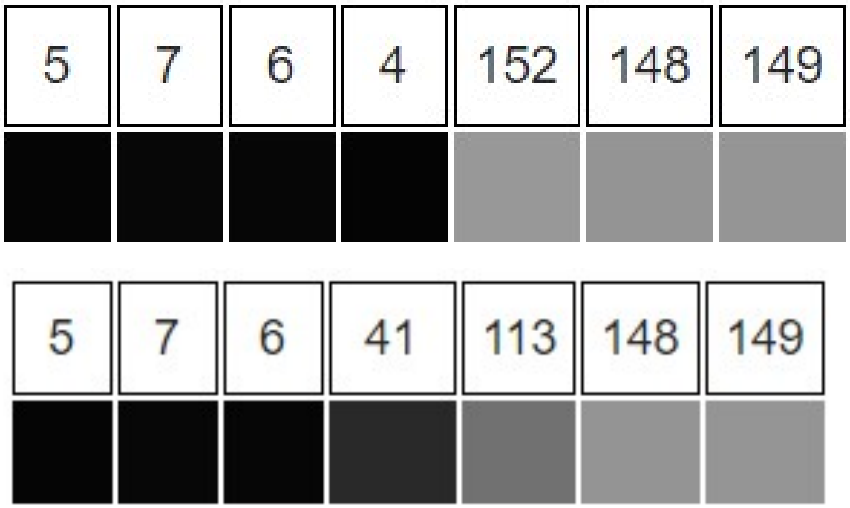
Credits: H. Coetzer, Department of Applied Mathematics, Stellenbosch University, S. Africa



Chapter 10

Segmentation

Edge Detection – Not so Trivial



Hence, to firmly state a specific threshold on how large the intensity change between two neighbouring pixels must be for us to say that there should be an edge between these pixels is not always simple.[4] Indeed, this is one of the reasons why edge detection may be a non-trivial problem unless the objects in the scene are particularly simple and the illumination conditions can be well controlled

https://en.wikipedia.org/wiki/Edge_detection

Edge Detection

- Edges
  - Edges naturally occur in images due to the discontinuities form by occlusion.
  - Edges often delineate the boundaries between distinct regions.
  - Edges often contain important visual and semantic information.
- Edge detection:
  - The process of identifying pixels that fall along edges.
  - As width any detect process subject to a trade-off between false alarm and missed detection rates.
- Performance Metrics:
  - Evaluation of edge detection scemes can be difficult.
  - Correct labeling of edge and non-edge pixels often requires subjective interpretation.
  - Best choice of edge detection scheme usually depends on task.
  - Performance metrics exist and usually use synthetic data input for evaluation.

Credits: Prof. C. A. Bouman, ECE Purdue University, ECE637 2015 notes

Gradient Based Edge Detection

- Compute local estimate of gradient

$$\nabla f(x,y) = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

- From these, compute gradient magnetude and angle.

$$|\nabla f| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

- Apply threshold to the magnitude of gradient

$$\text{edge} \quad |\nabla f| \geq T$$

$$\text{no edge} \quad |\nabla f| < T$$

- Choosing T
  - Too large $\Rightarrow$ missed detections
  - Too small $\Rightarrow$ false alarms

Credits: Prof. C. A. Bouman, ECE Purdue University, ECE637 2015 notes

How to Compute Gradient

- Directional derivatives can be computed by applying a spatial filter.
- Conventional (off center)

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

- Roberts (off center)

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

- Prewitt (on center)

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

- Sobel (on center)

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

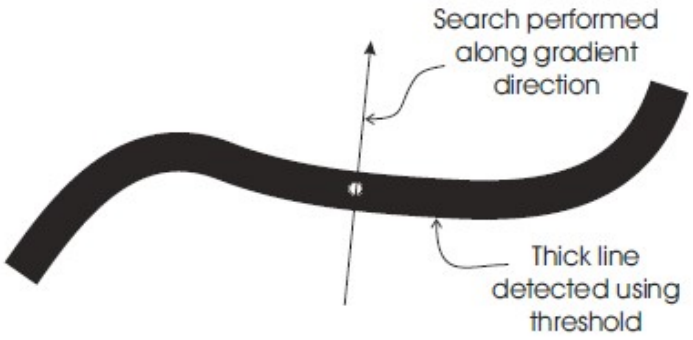
Credits: Prof. C. A. Bouman, ECE Purdue University, ECE637 2015 notes

Edge Thining

- Thresholding of gradient magnitude generally produces a thick edge.
- Edge should be thinned to produce most accurate result.

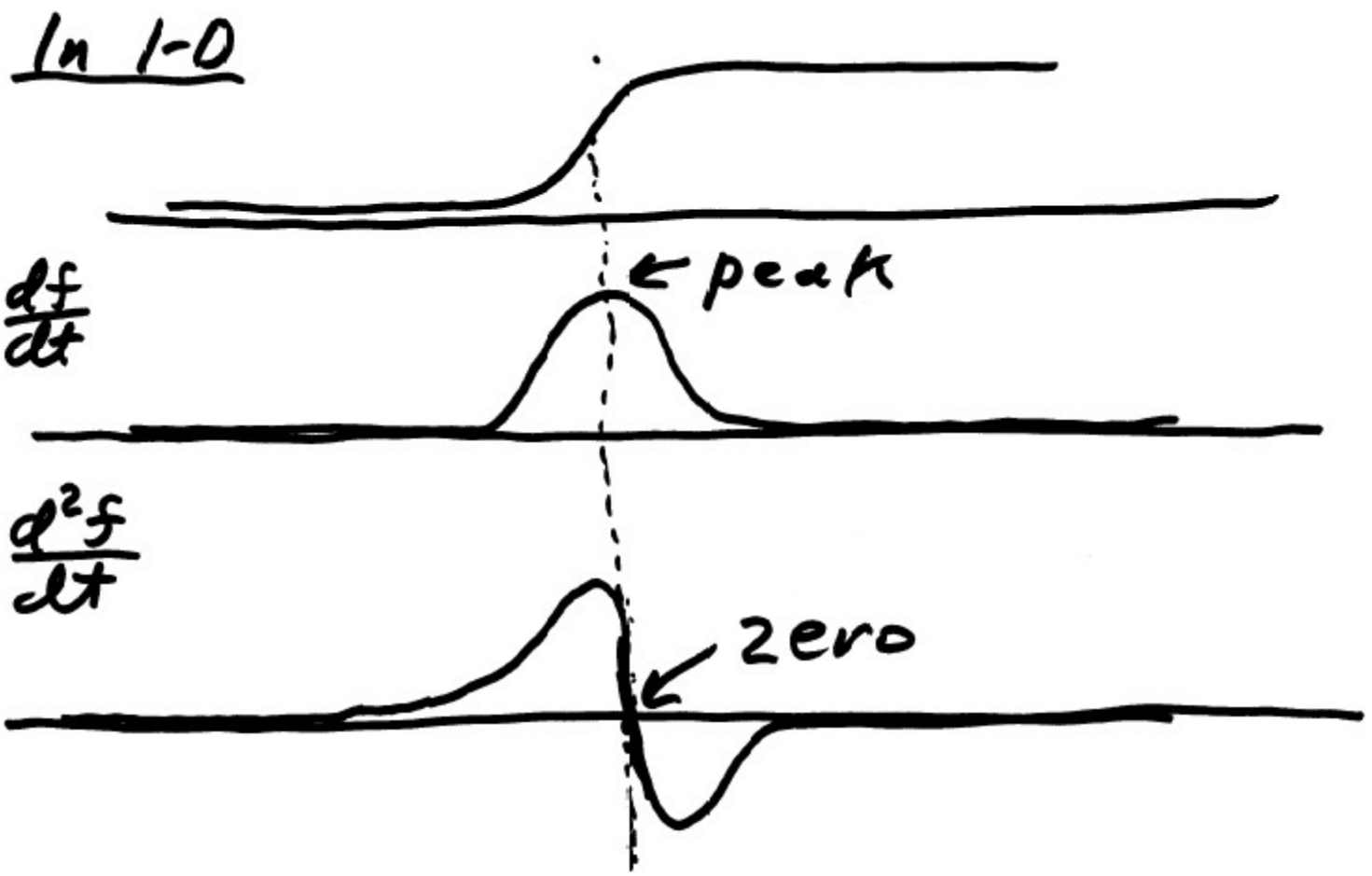
1. Set $S = \{s : |\nabla f(s)| \geq T\}$
2. Set $D = \emptyset$ (detected edge points)
3. For each $s \in S$
   (a) Compute $\theta =$ gradient direction at s.
   (b) Select out $P =$ all pixels in direction θ starting at s within maximum distance d_{max} from s.
   (c) If $|\nabla f(s)| \geq \max_{p \in P} \{|\nabla f(p)|\}$, then

$$D \leftarrow D + \{s\}$$



Credits: Prof. C. A. Bouman, ECE Purdue University, ECE637 2015 notes

Use of Second Derivative in Edge Detection



Credits: Prof. C. A. Bouman, ECE Purdue University, ECE637 2015 notes

Idea

- Look for points such that $\frac{d^2 f}{dt^2} = 0$

- Problem: Flat regions also have zero $\frac{d^2 f}{dt^2}$!

- Condition:

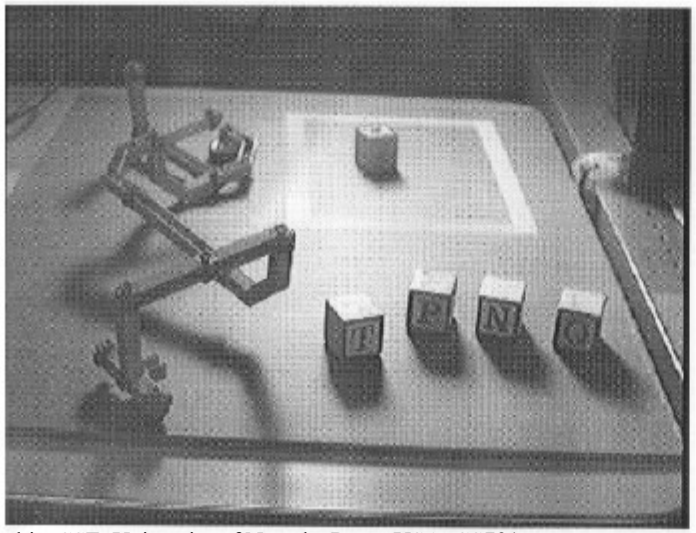
$$\left(\frac{d^2 f}{dt^2} = 0\right) \text{ and } \left(\left|\frac{df}{dx}\right| > T\right)$$

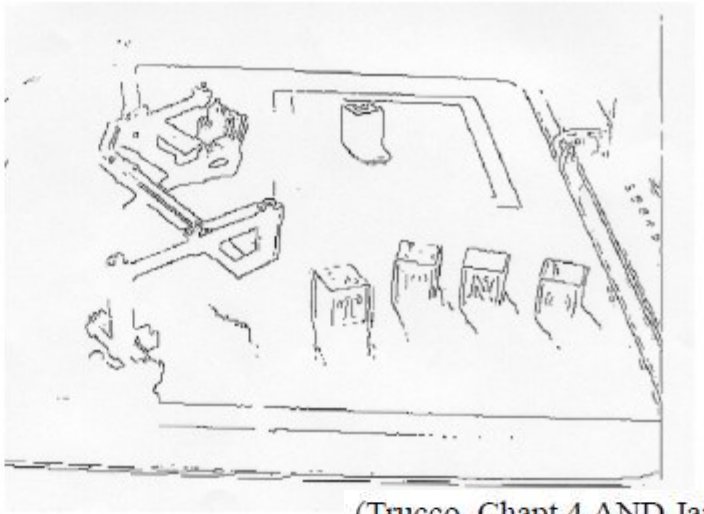
• What causes intensity changes?

- Various physical events cause intensity changes.

- Geometric events
  * object boundary (discontinuity in depth and/or surface color and texture)
  * surface boundary (discontinuity in surface orientation and/or surface color and texture)

- Non-geometric events
  * specularity (direct reflection of light, such as a mirror)
  * shadows (from other objects or from the same object)
  * inter-reflections



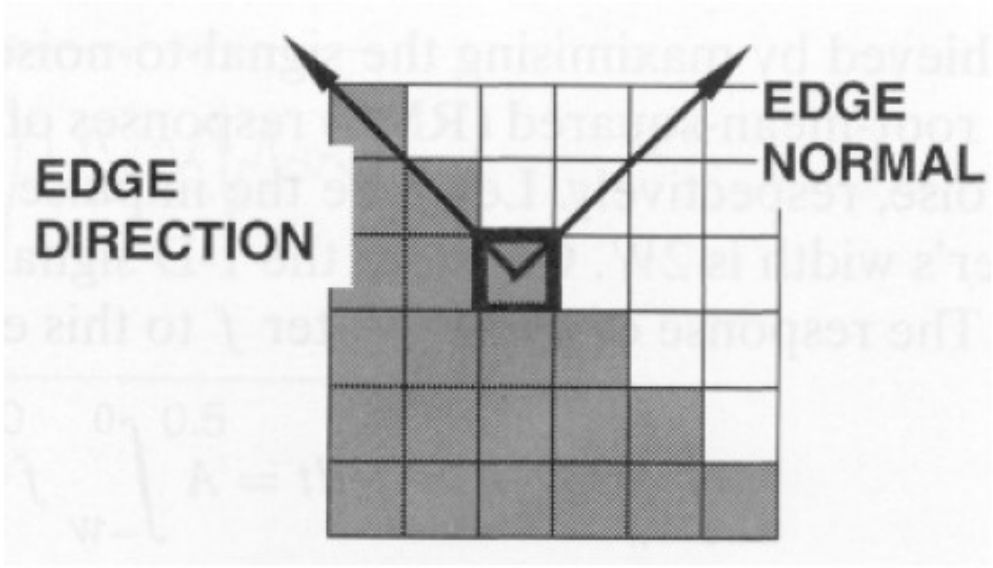


(Trucco, Chapt 4 AND Jain et al., Chapt 5)

Credits: Dr. George Bebis, CSE, University of Nevada, Reno, USA CS791 notes

- **Edge descriptors**

Edge normal: unit vector in the direction of maximum intensity change.
Edge direction: unit vector to perpendicular to the edge normal.
Edge position or center: the image position at which the edge is located.
Edge strength: related to the local image contrast along the normal.



(Trucco, Chapt 4 AND Jain et al., Chapt 5)

- **The four steps of edge detection**

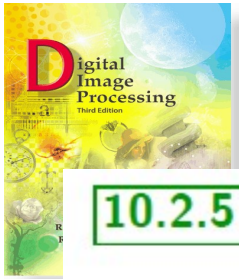
(1) Smoothing: suppress as much noise as possible, without destroying the true edges.

(2) Enhancement: apply a filter to enhance the quality of the edges in the image (sharpening).

(3) Detection: determine which edge pixels should be discarded as noise and which should be retained (usually, thresholding provides the criterion used for detection).

(4) Localization: determine the exact location of an edge (*sub-pixel* resolution might be required for some applications, that is, estimate the location of an edge to better than the spacing between pixels). Edge thinning and linking are usually required in this step.

(Trucco, Chapt 4 AND Jain et al., Chapt 5)



10.2.5 Basic edge detection

The image gradient and its properties

Definition

$$\nabla \mathbf{f} = \begin{pmatrix} g_x \\ g_y \end{pmatrix} = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix}$$

Magnitude

$$\nabla f = \text{mag}(\nabla \mathbf{f}) = [g_x^2 + g_y^2]^{1/2} = \left[\left(\frac{\partial f}{\partial x} \right)^2 + \left(\frac{\partial f}{\partial y} \right)^2 \right]^{1/2}$$

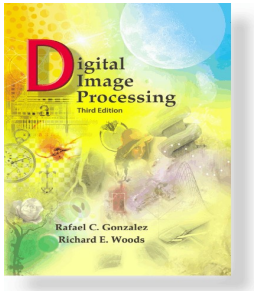
Direction

$$\alpha(x, y) = \tan^{-1} \left(\frac{g_y}{g_x} \right)$$

The direction of an edge at (x, y) is perpendicular to the direction of the gradient vector at that point

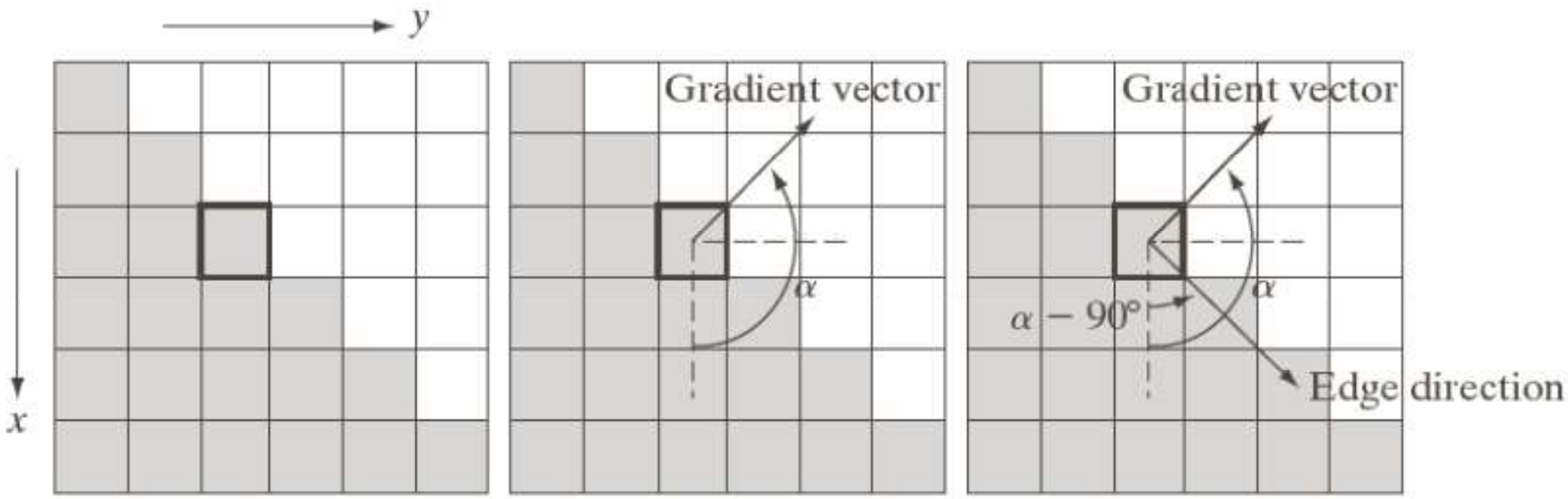
- The magnitude of gradient provides information about the strength of the edge.

Credits: H. Coetzer, Department of Applied Mathematics, Stellenbosch University, S. Africa



Chapter 10

Segmentation



a b c

FIGURE 10.12 Using the gradient to determine edge strength and direction at a point. Note that the edge is perpendicular to the direction of the gradient vector at the point where the gradient is computed. Each square in the figure represents one pixel.

Gradient operators

$$g_x = \frac{\partial f(x,y)}{\partial x} = f(x+1,y) - f(x,y)$$

$$g_y = \frac{\partial f(x,y)}{\partial y} = f(x,y+1) - f(x,y)$$

−1
1

−1	1

a b

FIGURE 10.13 One-dimensional masks used to implement Eqs. (10.2-12) and (10.2-13).

Roberts:

$$g_x = \frac{\partial f(x,y)}{\partial x} = (z_9 - z_5)$$

$$g_y = \frac{\partial f(x,y)}{\partial y} = (z_8 - z_6)$$

Prewitt:

$$g_x = (z_7 + z_8 + z_9) - (z_1 + z_2 + z_3)$$

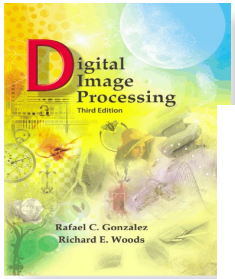
$$g_y = (z_3 + z_6 + z_9) - (z_1 + z_4 + z_7)$$

Sobel:

$$g_x = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

$$g_y = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$

Credits: H. Coetzer, Department of Applied Mathematics, Stellenbosch University, S. Africa



Digital Image Processing, 3rd ed.

Gonzalez & Woods

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

a
b c
d e
f g

FIGURE 10.14
A 3 × 3 region of an image (the z's are intensity values) and various masks used to compute the gradient at the point labeled z_5.

−1	0	0	−1
0	1	1	0

Roberts

−1	−1	−1	−1	0	1
0	0	0	−1	0	1
1	1	1	−1	0	1

Prewitt

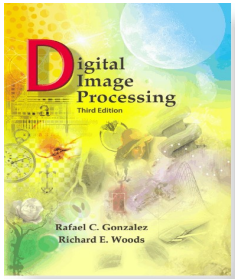
−1	−2	−1	−1	0	1
0	0	0	−2	0	2
1	2	1	−1	0	1

Sobel

Convenient approximation:

$$M(x, y) \approx |g_x| + |g_y|$$

Credits: H. Coetzer, Department of Applied Mathematics, Stellenbosch University, S. Africa



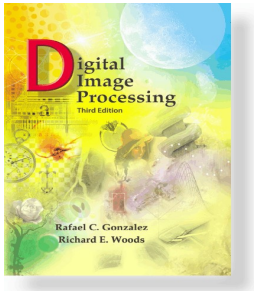
The Prewitt masks are simpler to implement than the Sobel masks, but, the slight computational difference between them typically is not an issue. The fact that the Sobel masks have better noise-suppression (smoothing) characteristics makes them preferable because, as mentioned in the previous section, noise suppression is an important issue when dealing with derivatives. Note that the coefficients of all the masks in Fig. 10.14 sum to zero, thus giving a response of zero in areas of constant intensity, as expected of a derivative operator.

The masks just discussed are used to obtain the gradient components g_x and g_y at every pixel location in an image. These two partial derivatives are then used to estimate edge strength and direction. Computing the magnitude of the gradient requires that g_x and g_y be combined in the manner shown in Eq. (10.2-10). However, this implementation is not always desirable because of the computational burden required by squares and square roots. An approach used frequently is to approximate the magnitude of the gradient by absolute values:

$$M(x, y) \approx |g_x| + |g_y| \qquad (10.2\text{-}20)$$

This equation is more attractive computationally, and it still preserves relative changes in intensity levels. The price paid for this advantage is that the resulting filters will not be isotropic (invariant to rotation) in general. However, this is not an issue when masks such as the Prewitt and Sobel masks are used to compute g_x and g_y, because these masks give isotropic results only for vertical and horizontal edges. Results would be isotropic only for edges in those two directions, regardless of which of the two equations is used. In addition, Eqs. (10.2-10) and (10.2-20) give identical results for vertical and horizontal edges when the Sobel or Prewitt masks are used (Problem 10.8).

It is possible to modify the 3×3 masks in Fig. 10.14 so that they have their strongest responses along the diagonal directions. Figure 10.15 shows the two additional Prewitt and Sobel masks needed for detecting edges in the diagonal directions.



Chapter 10

Segmentation

0	1	1	−1	−1	0
−1	0	1	−1	0	1
−1	−1	0	0	1	1

Prewitt

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

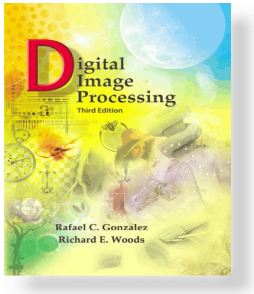
Sobel

a b
c d

FIGURE 10.15 Prewitt and Sobel masks for detecting diagonal edges.

a b
c d

FIGURE 10.16
(a) Original image of size 834 × 1114 pixels, with intensity values scaled to the range [0, 1]. (b) $|g_x|$, the component of the gradient in the x-direction, obtained using the Sobel mask in Fig. 10.14(f) to filter the image. (c) $|g_y|$, obtained using the mask in Fig. 10.14(g). (d) The gradient image, $|g_x| + |g_y|$.



Digital Image Processing, 3rd ed.

Gonzalez & Woods

www.ImageProcessingPlace.com

Chapter 10
Segmentation

$$\alpha(x, y) = \tan^{-1}\left[\frac{g_y}{g_x}\right]$$

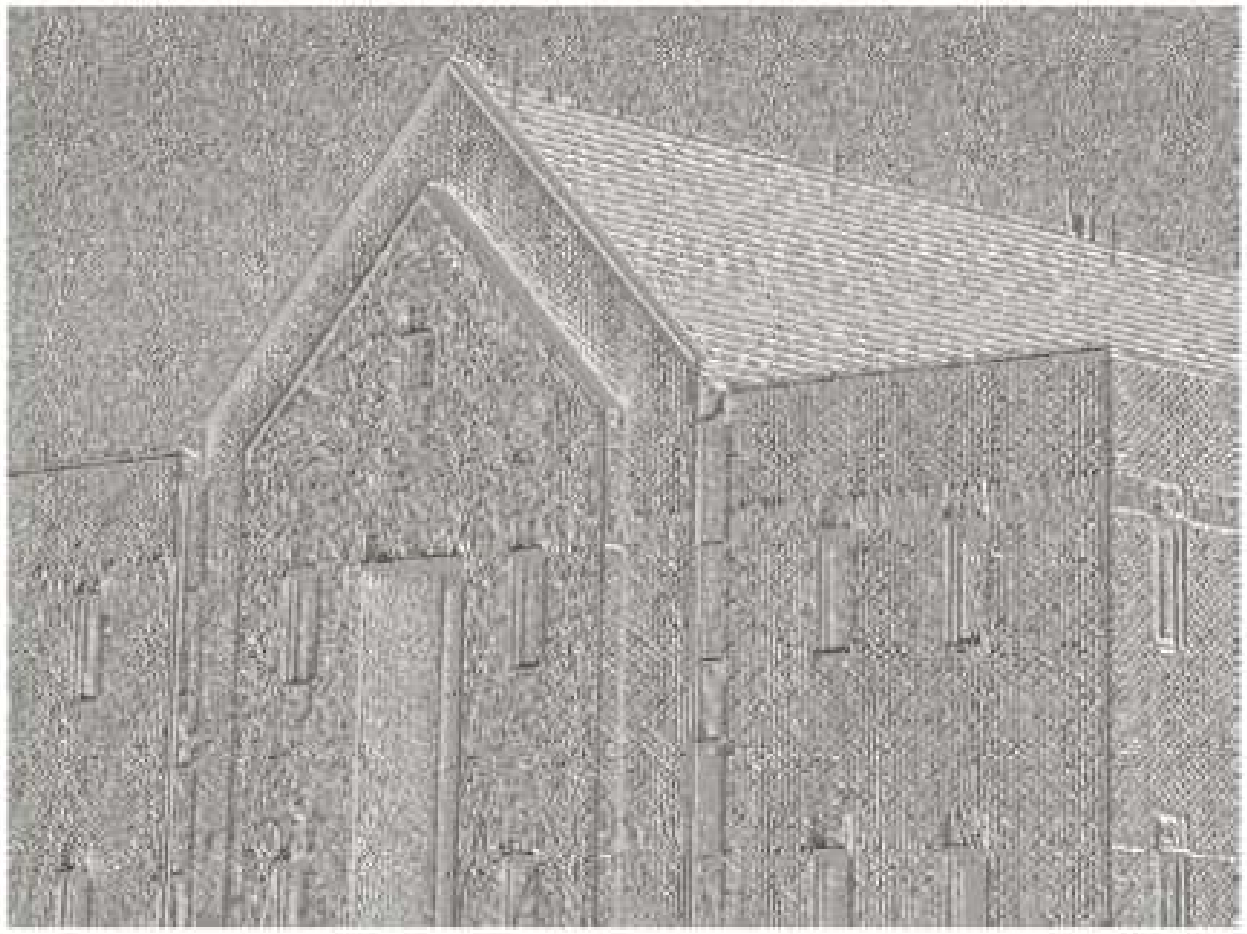
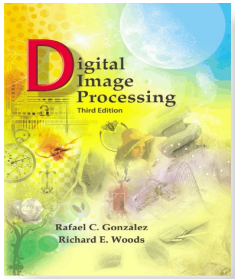


FIGURE 10.17 Gradient angle image computed using Eq. (10.2-11). Areas of constant intensity in this image indicate that the direction of the gradient vector is the same at all the pixel locations in those regions.

Angle info plays key supporting role in Canny edge detection

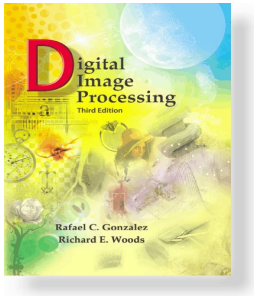


Chapter 10

Segmentation

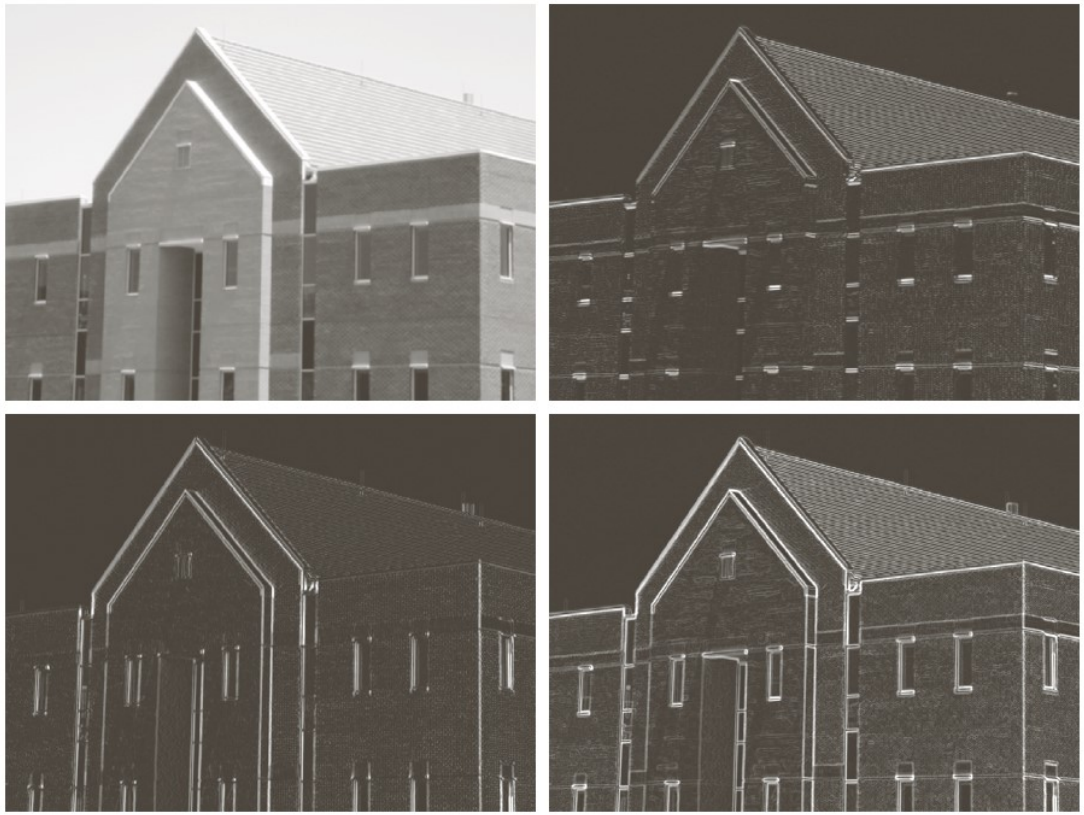
Figure 10.17 shows the gradient angle image computed using Eq. (10.2-11). **In general, angle images are not as useful as gradient magnitude images for edge detection, but they do complement the information extracted from an image using the magnitude of the gradient. For instance, the constant intensity areas in Fig.** 10.16(a), such as the front edge of the sloping roof and top horizontal bands of the front wall, are constant in Fig. 10.17, indicating that the gradient vector direction at all the pixel locations in those regions is the same.

Angle info plays key supporting role in Canny edge detection



Result with prior smoothing

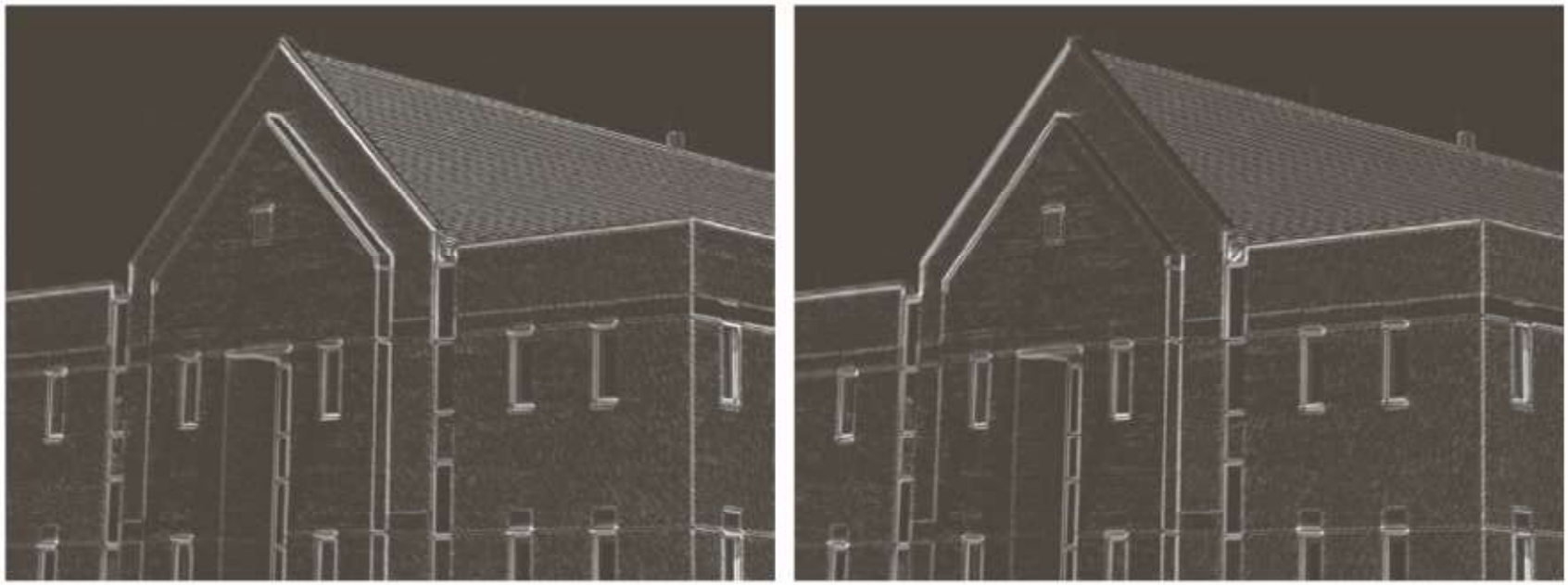
Fine detail info (e.g. bricks contribution in original image) often is undesirable in edge detection because it tends to act as noise, which is enhanced by derivative computations and thus complicates detection of principal edges in an image. One way to reduce the fine detail is to smooth the image



a b
c d

FIGURE 10.18
Same sequence as in Fig. 10.16, but with the original image smoothed using a 5 × 5 averaging filter prior to edge detection.

Now almost no contribution due to the bricks)



a b

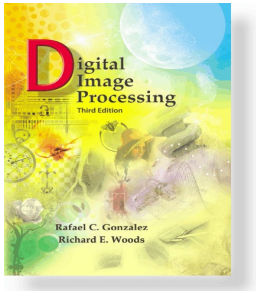
FIGURE 10.19
Diagonal edge detection.
(a) Result of using the mask in Fig. 10.15(c).
(b) Result of using the mask in Fig. 10.15(d). The input image in both cases was Fig. 10.18(a).

Diagonal edge detection

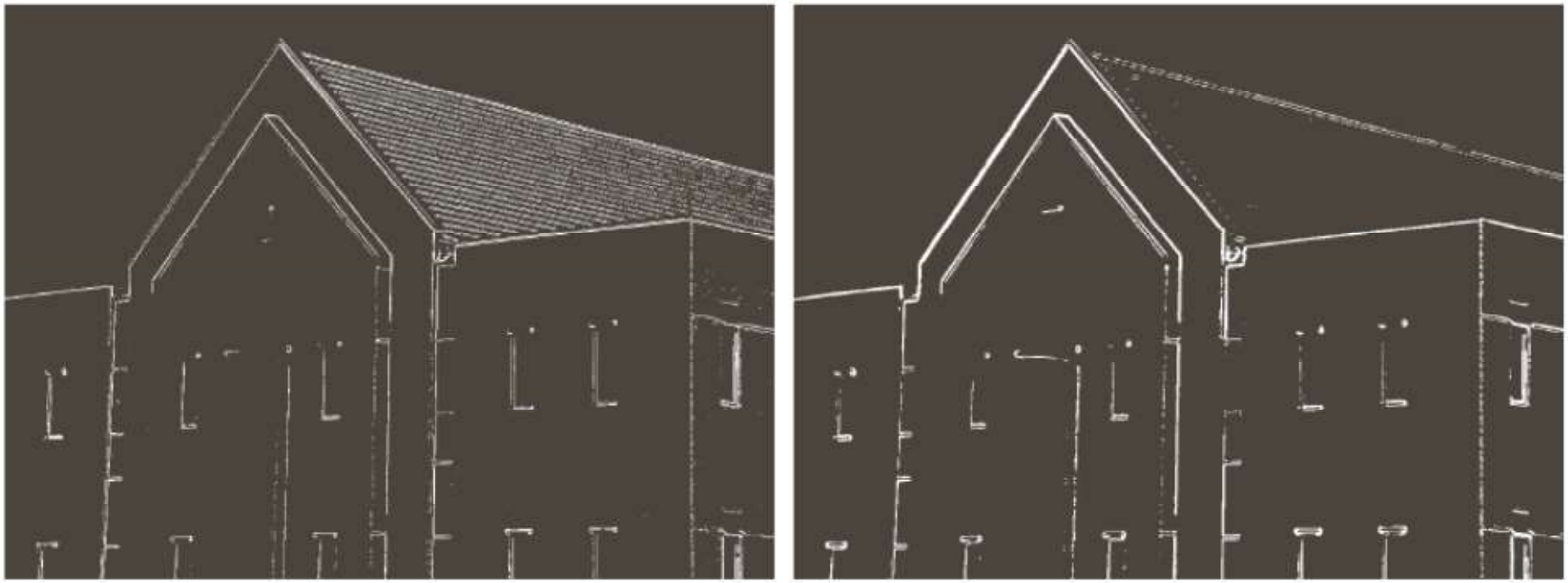
Combining the gradient with thresholding

The results in Fig. 10.18 show that edge detection can be made more selective by smoothing the image prior to computing the gradient. Another approach aimed at achieving the same basic objective is to threshold the gradient image. For example, Fig. 10.20(a) shows the gradient image from Fig. 10.16(d) thresholded, in the sense that pixels with values greater than or equal to 33% of the maximum value of the gradient image are shown in white, while pixels below the threshold value are shown in black. Comparing this image with Fig. 10.18(d), we see that there are fewer edges in the thresholded image, and that the edges in this image are much sharper (see, for example, the edges in the roof tile). On the other hand, numerous edges, such as the 45° line defining the far edge of the roof, are broken in the thresholded image.

When interest lies both in highlighting the principal edges and on maintaining as much connectivity as possible, it is common practice to use both smoothing and thresholding. Figure 10.20(b) shows the result of thresholding Fig. 10.18(d), which is the gradient of the smoothed image. This result shows a reduced number of broken edges; for instance, compare the 45° edges in Figs. 10.20(a) and (b). Of course, edges whose intensity values were severely attenuated due to blurring (e.g., the edges in the tile roof) are likely to be totally eliminated by thresholding. We return to the problem of broken edges in Section 10.2.7.



Combining the gradient with thresholding



a b

FIGURE 10.20 (a) Thresholded version of the image in Fig. 10.16(d), with the threshold selected as 33% of the highest value in the image; this threshold was just high enough to eliminate most of the brick edges in the gradient image. (b) Thresholded version of the image in Fig. 10.18(d), obtained using a threshold equal to 33% of the highest value in that image.

- **Criteria for optimal edge detection**

(1) Good detection: the optimal detector must minimize the probability of false positives (detecting spurious edges caused by noise), as well as that of false negatives (missing real edges).

(2) Good localization: the edges detected must be as close as possible to the true edges.

Single response constraint: the detector must return one point only for each true edge point; that is, minimize the number of local maxima around the true edge (created by noise).

(Trucco, Chapt 4 AND Jain et al., Chapt 5)