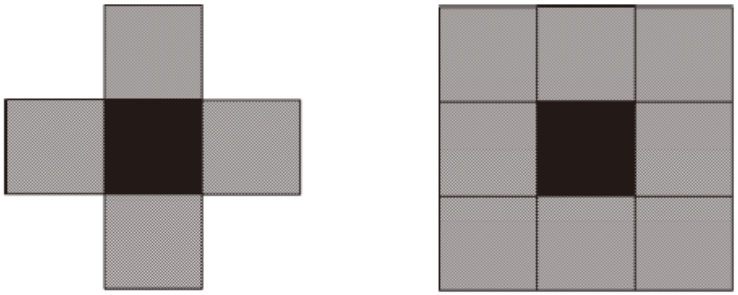
2値画像に対する処理

連結と近傍

- 注目する画素同士が幾何学的につながっているか離れているかを判断する基準
- 隣接画素と連結成分であるかどうか判定する際に**4連結**で見るか**8連結**で見るかによって異なる



（a） 4近傍　　（b） 8近傍

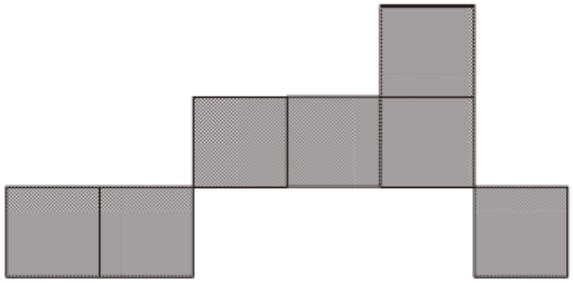
図 6.2　連結と近傍

表 6.1　近傍と連結

4 近傍	注目画素の上下左右方向の画素
4 連結	4 近傍中に画素が存在する場合
8 近傍	注目画素の上下左右斜め方向の画素
8 連結	8 近傍中に画素が存在する場合

図形画素と背景画素の連結性

- ある連結成分が他の色の連結成分を内部に含むときそれを**孔**（hole）という
- 図形画素（黒）を**8連結**で考えるとき，
  背景画素（白）は**4連結**で考える必要がある



（連結性により個数が異なる）

図 6.3　図 形 成 分

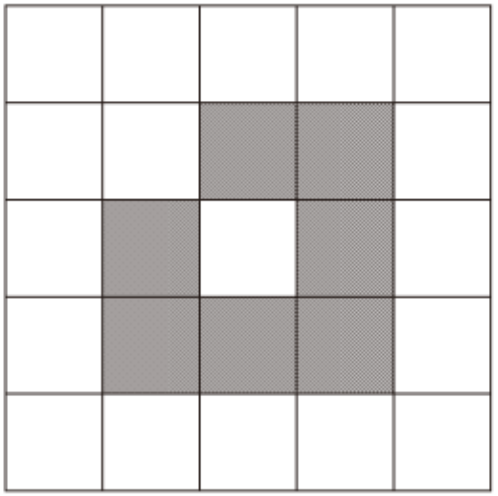


図 6.4　孔

輪郭線

- 連結成分の画素列のうち，境界点の画素を抽出した画素列を**輪郭線**（**境界線**）とよぶ

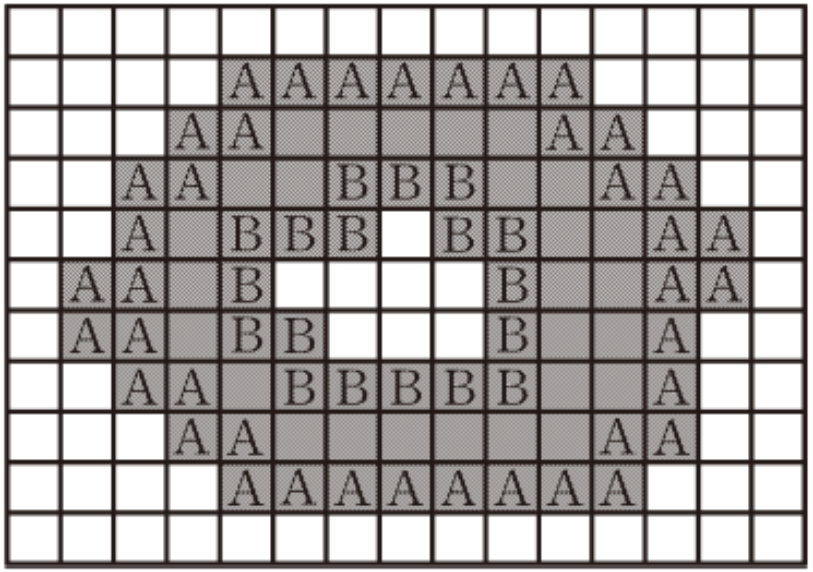


図 6.5　境　界　線

連結数

- 画素が結合している連結成分の個数を表す数
- 図は3×3画素の中央の画素の連結性を示す

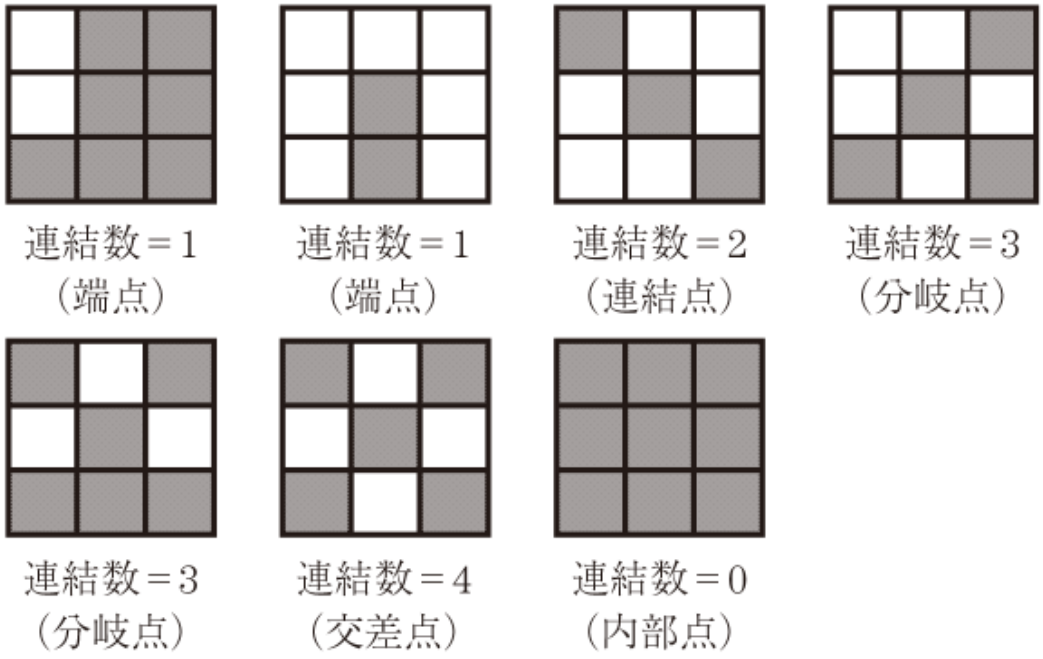


図 6.6　3×3 画素の中央の画素の連結数

距離

- 画素間の遠近の程度を表す尺度
- ユークリッド距離：$\sqrt{(i-k)^2+(j-h)^2}$
- 4近傍距離：$|i-k|+|j-h|$
- 8近傍距離：$\max(|i-k|,|j-h|)$

$\sqrt{8}$	$\sqrt{5}$	2	$\sqrt{5}$	$\sqrt{8}$
$\sqrt{5}$	$\sqrt{2}$	1	$\sqrt{2}$	$\sqrt{5}$
2	1	0	1	2
$\sqrt{5}$	$\sqrt{2}$	1	$\sqrt{2}$	$\sqrt{5}$
$\sqrt{8}$	$\sqrt{5}$	2	$\sqrt{5}$	$\sqrt{8}$

（a） ユークリッド距離

4	3	2	3	4
3	2	1	2	3
2	1	0	1	2
3	2	1	2	3
4	3	2	3	4

（b） 4近傍距離

2	2	2	2	2
2	1	1	1	2
2	1	0	1	2
2	1	1	1	2
2	2	2	2	2

（c） 8近傍距離

図 6.7 画素間の距離

2値画像処理：ラベリング

- 連結している図形成分に番号をつけて
  ひとくくりにまとめる処理

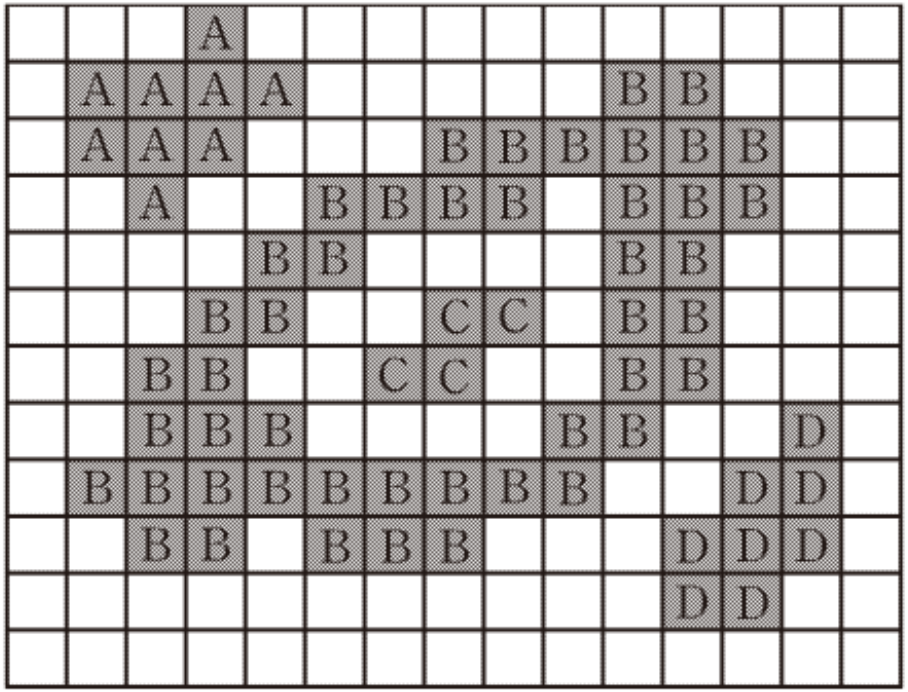
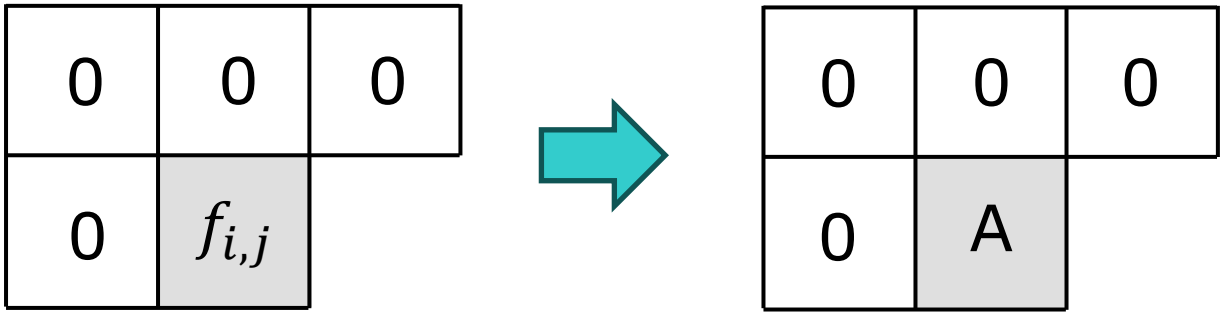


図 6.8　連結成分に対するラベリング

ラベリングの手順（8連結）

- 左上から右下に向かって2回の走査（スキャン処理）を行うことでラベリングできる
- 1回目の走査（スキャン処理）

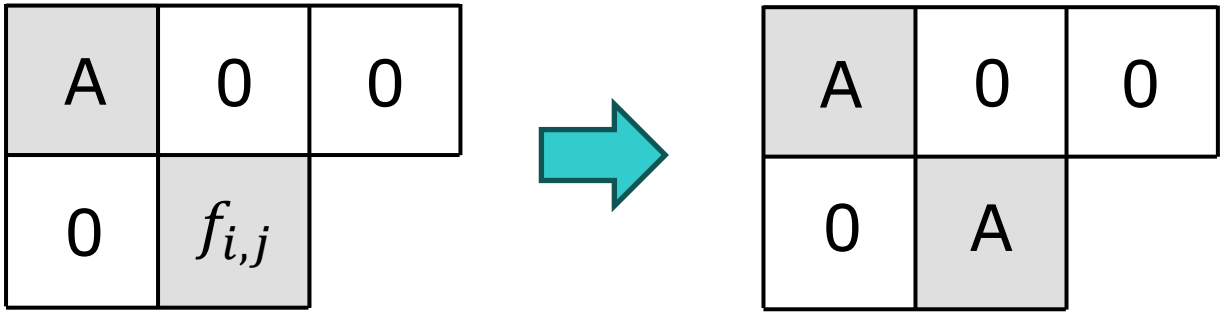
1. 4つの画素が全て0のとき$f_{i,j}$に新しいラベルを付与



ラベリングの手順（8連結）

- 1回目の走査（スキャン処理）

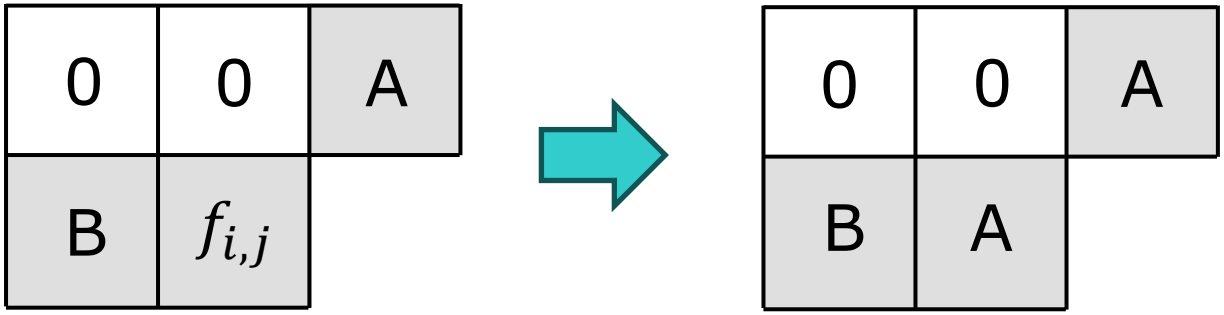
2. 4つの画素のうち，0以外の値に1種類のラベルが付いている場合，$f_{i,j}$に同じラベルを付与



ラベリングの手順（8連結）

- 1回目の走査（スキャン処理）

3. 2種類以上ラベルが付いている場合，その中で最も小さい番号のラベルを$f_{i,j}$につける
   これらの連結成分が同じ連結成分であることを記憶しておく

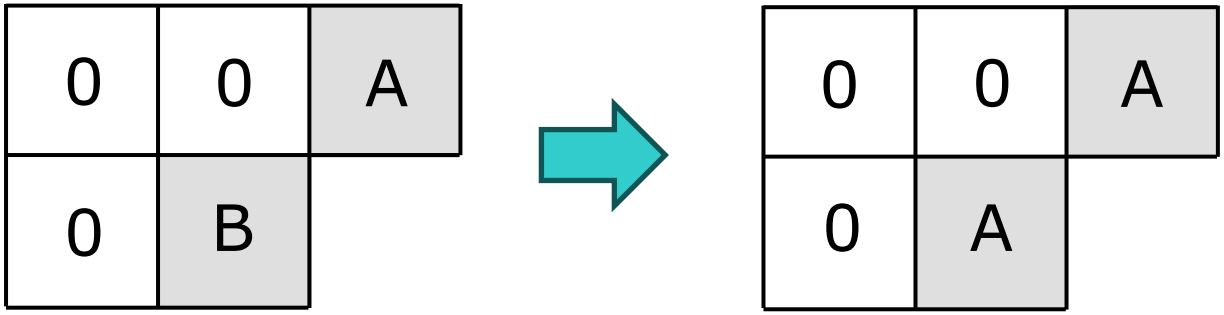


BがAと同じ連結成分であると記憶

ラベリングの手順(8連結)

- 2回目の走査

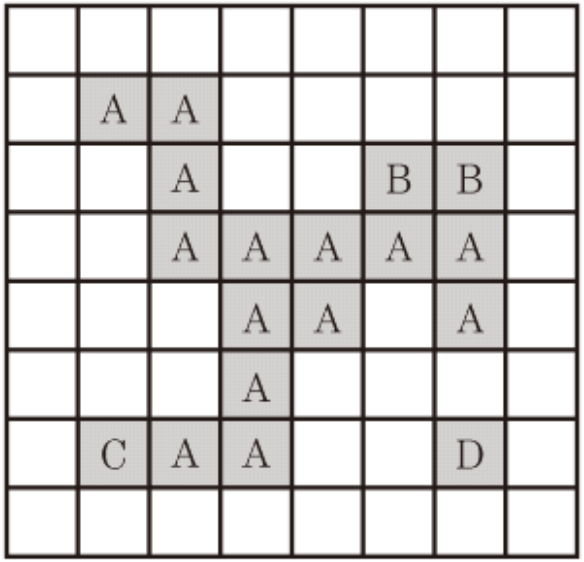
4. 最後まで走査が終わったら，左上から2回目の走査を行い，一つの連結成分に対して同一のラベルを付け直す



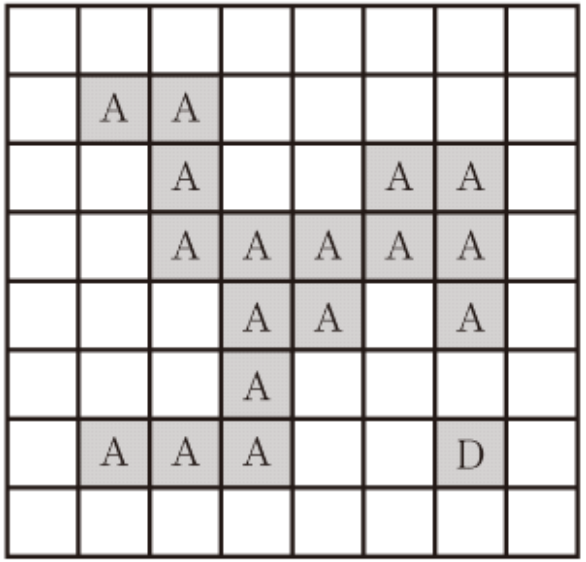
記録：B⇔A

ラベリングの手順（8連結）

- 2回の走査でラベリングが実現できる



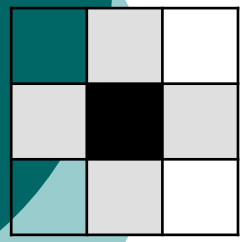
（a） 1回目の走査の結果



（b） 2回目の走査の結果

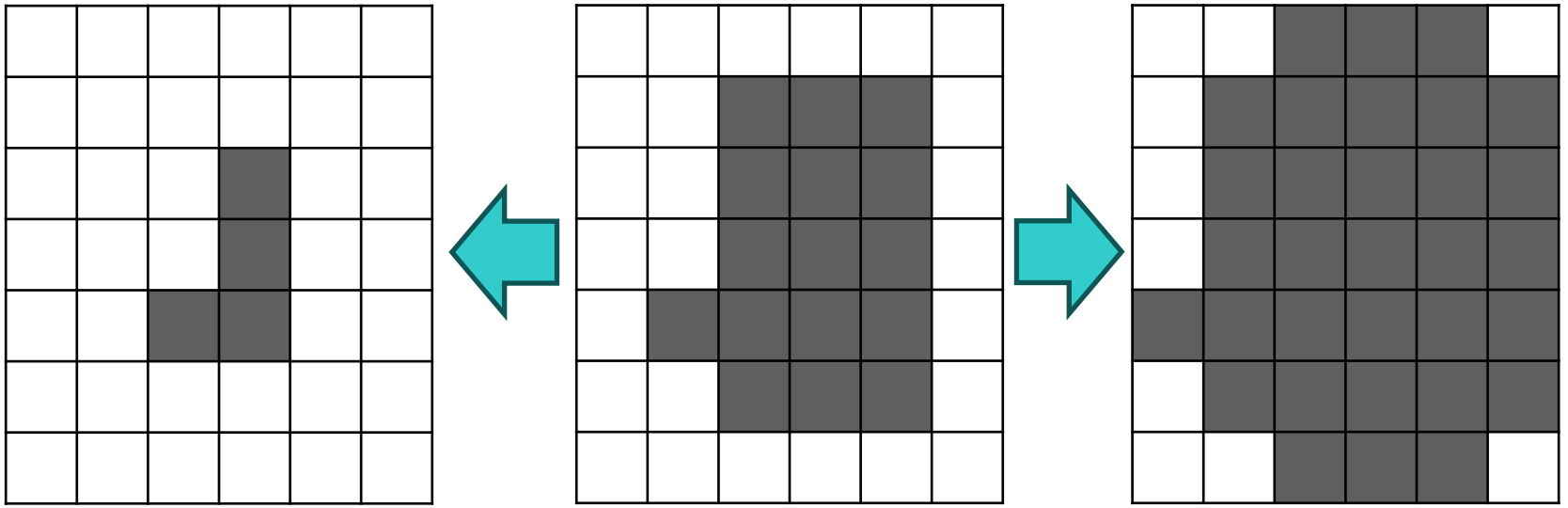
図 6.10 ラベリング結果の例

モルフォロジー演算：膨張・収縮処理



4近傍

- 膨張処理
  - 近傍に黒画素がある白画素を黒画素に変換
- 収縮処理
  - 近傍に白画素がある黒画素を白画素に変換



収縮処理（4近傍）　二値画像　膨張処理（4近傍）

クロージング・オープニング

- 膨張処理と収縮処理を組み合わせてノイズ除去

- クロージング
  - 膨張→収縮処理
  - 小さな穴を塞ぐ
- オープニング
  - 収縮→膨張処理
  - 小さな連結成分を除去

（a） 膨張と収縮の組み合わせ処理の例

（b） 収縮と膨張の組み合わせ処理の例

図 6.11 膨張・収縮処理の例

細線化

- 画像中の細長い図形を途切れることなく線幅が1の図形に変換する処理
- 収縮処理とは異なり，図形の連結関係は保存される

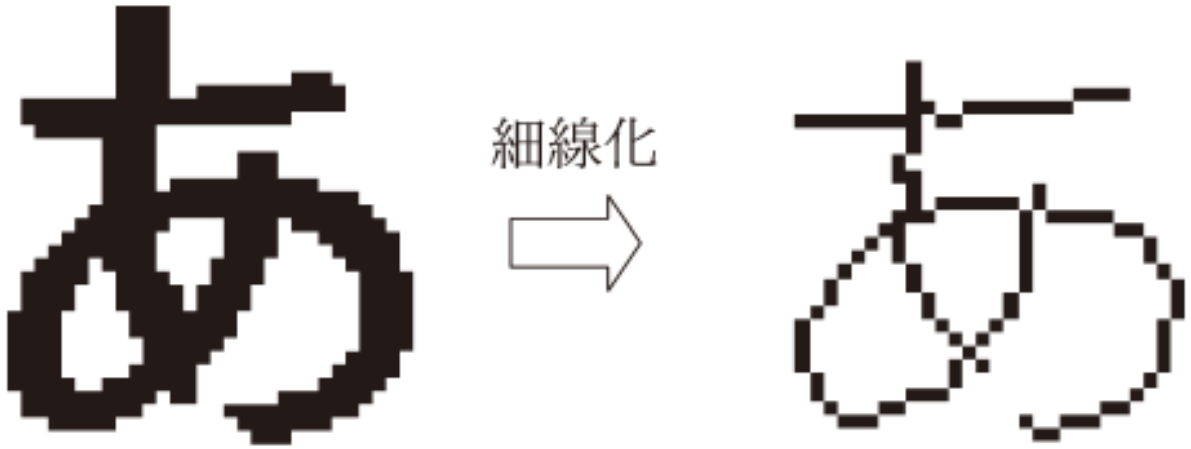
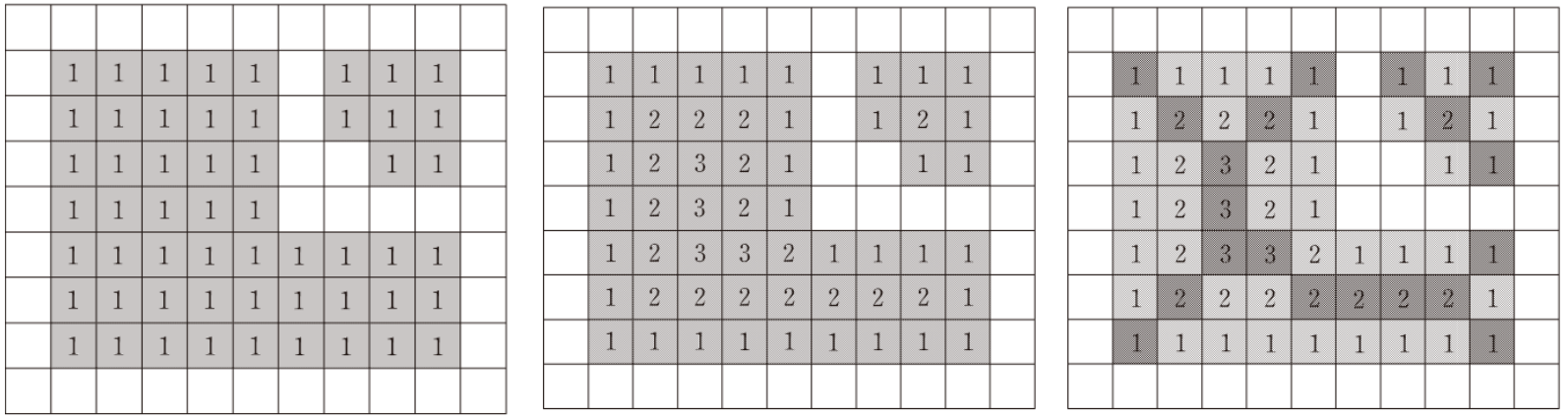


図 6.12　細線化の例

距離変換と骨格化

- 距離変換
  - 周辺部からの距離の大きさを画素の値とする変換
- 骨格化
  - 距離変換された図形から4近傍での値が最大となる画素の集合を求める操作

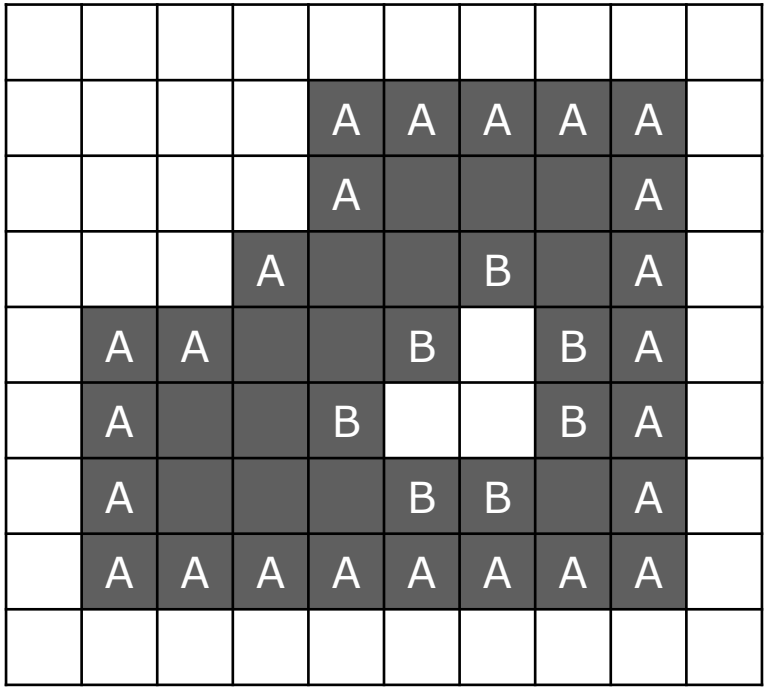


（a） 2値画像　　（b） 距離変換　　（c） 抽出された骨格

図 6.13　距離変換と骨格化（4 近傍型）

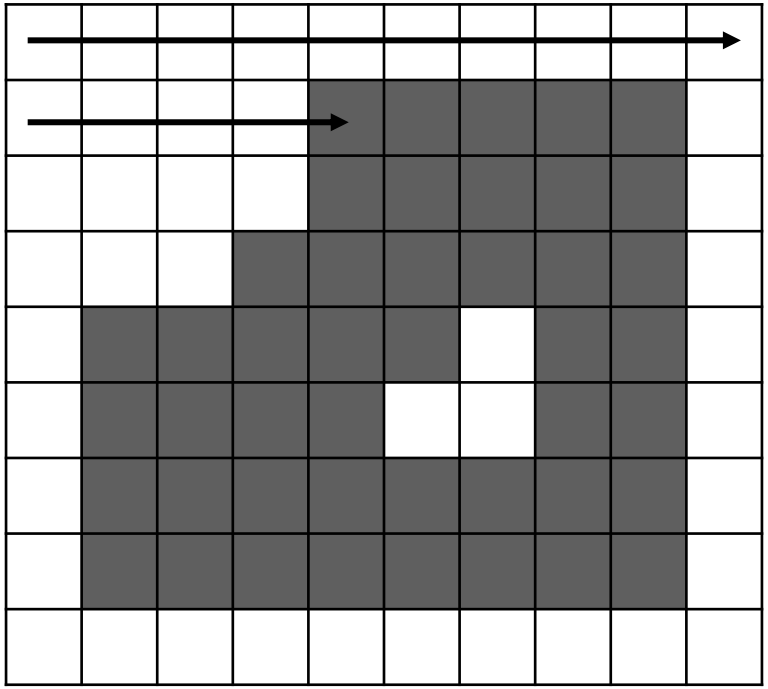
輪郭線追跡

- 広がりのある塊状図形の境界画素を抽出
- 図形の形状を解析するのに利用
- 単に境界画素の集合を抽出するだけでなく，順序付けられた画素の系列として抽出できる



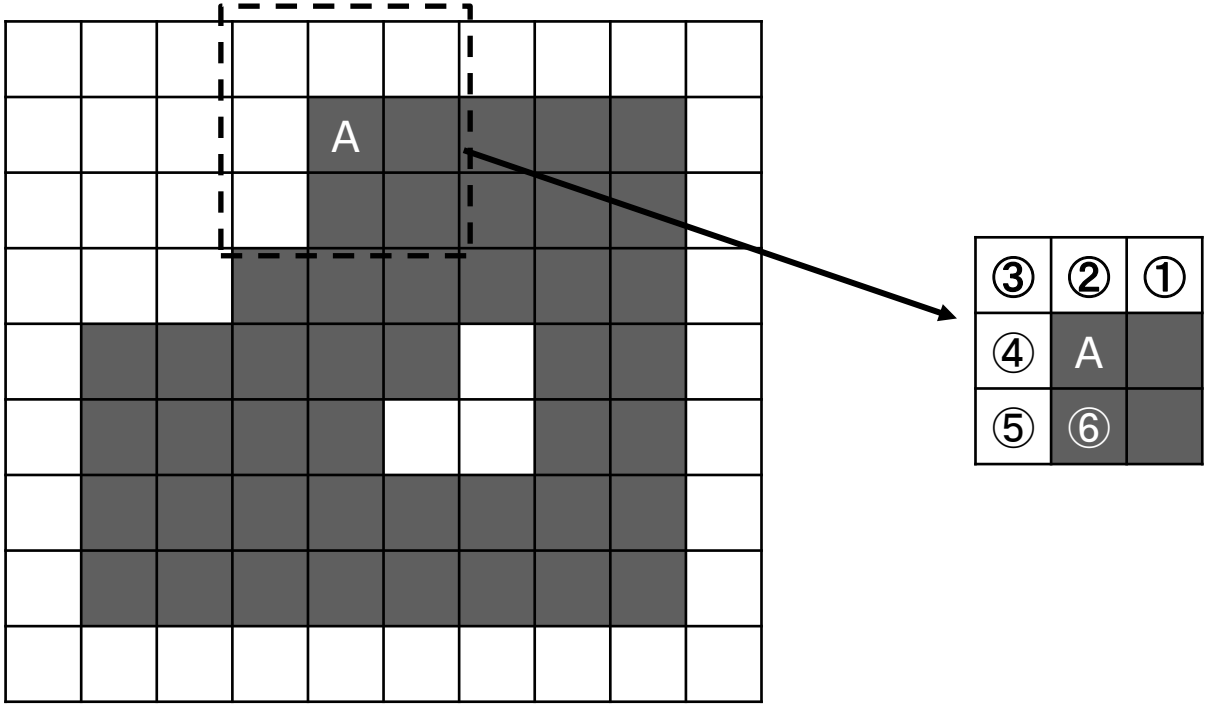
境界線追跡アルゴリズム（8近傍）

- 左上の画素から走査し，左側に背景画素（白）が隣接する図形画素（黒）を見つけ，それを最初の境界画素とする
- 既に境界画素として認識されていれば無視する



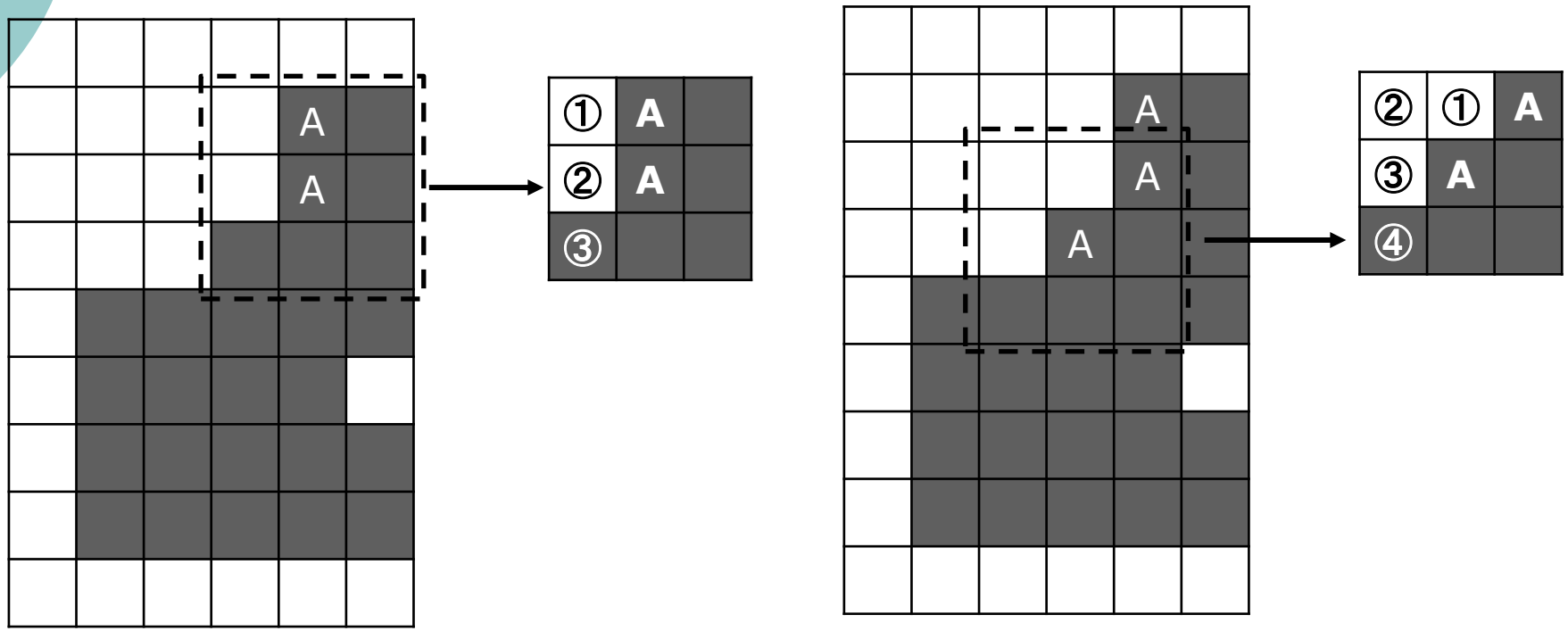
境界線追跡アルゴリズム（8近傍）

- 最初の境界画素を中心とする8近傍を，反時計回りに調べ，**背景画素（白）から始めて図形画素（黒）に変わったとき**の図形画素を見つける
- その画素を次の境界画素とする



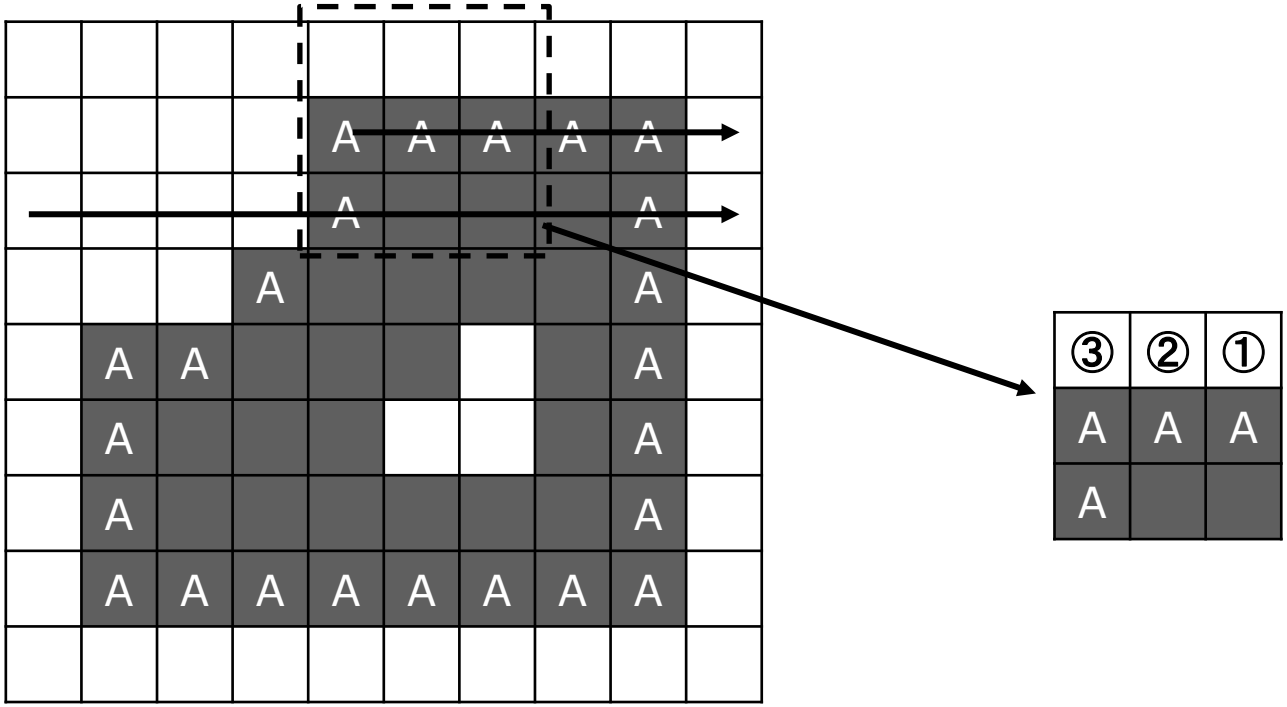
境界線追跡アルゴリズム（8近傍）

- 次の境界画素は，前の境界画素に対して，（反時計回りで）隣の画素から探索する



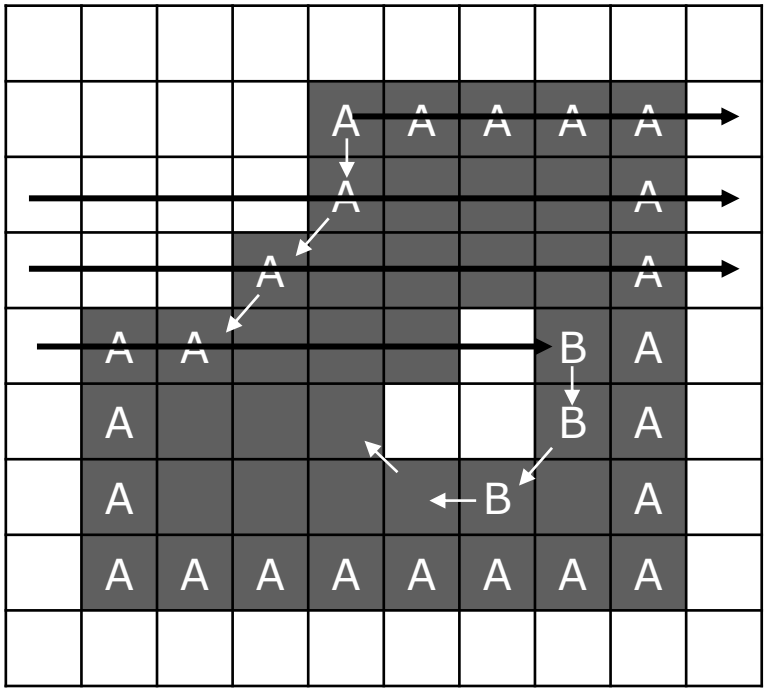
境界線追跡アルゴリズム（8近傍）

- 次の境界画素が，最初に認識された境界画素と一致したら終了し，次の走査を行う
- 走査を行う上で，既に境界画素とされたものは無視



境界線追跡アルゴリズム（8近傍）

- 外側の境界は左回りに追跡される
- 内側の境界は右回りに追跡される



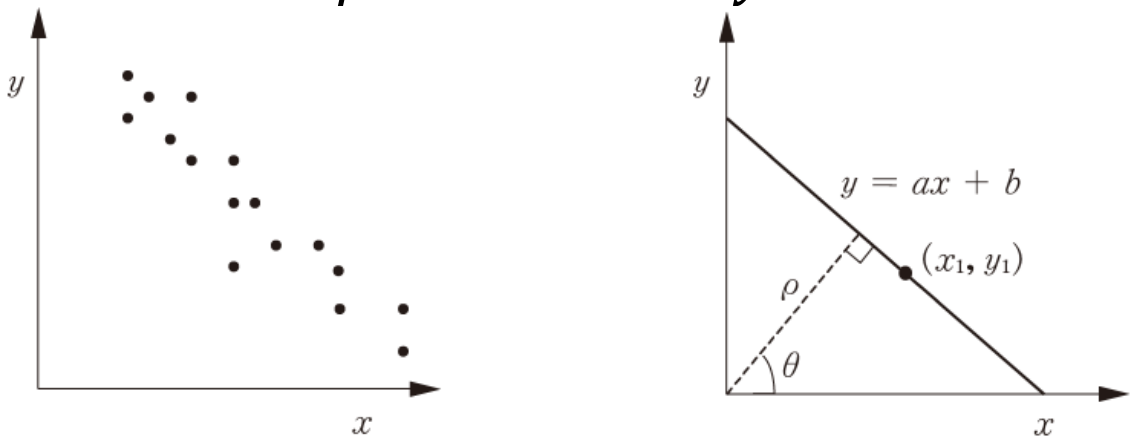
ハフ変換を用いた直線検出

- ある1点(x_1, y_1)が与えられたとき，これを通る直線は，傾きaと切片bのパラメータを用いると

$$y = ax + b$$

- 原点から直線におろした垂線の長さρと，垂線がx軸となす角度θを用いると

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

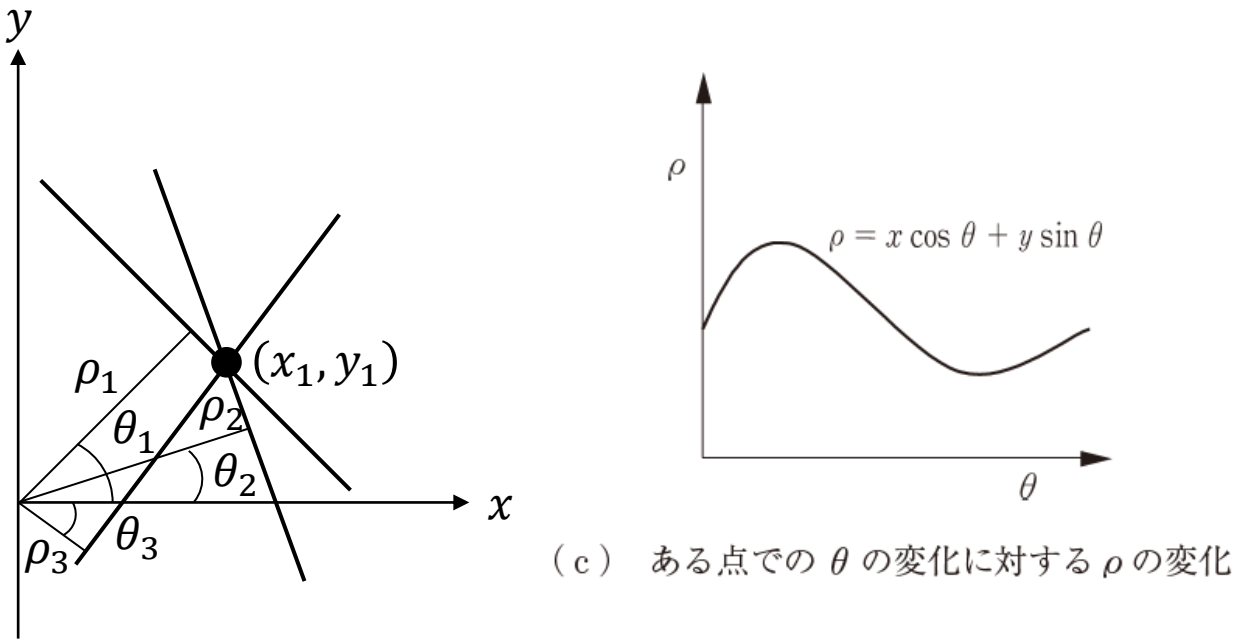


（a） 線の候補となる画素　　（b） 点（x_1, y_1）を通る直線

山田 宏，末松 良一 「画像処理と画像認識 ―AI時代の画像処理入門」 コロナ社より

ハフ変換を用いた直線検出

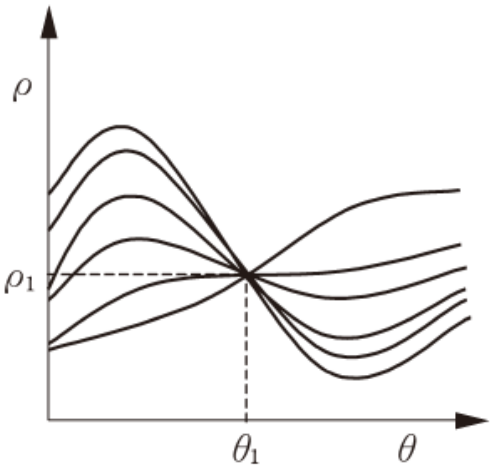
- 点(x_1, y_1)を通るすべての直線をパラメータ(θ, ρ)で表したとき，θの変化に対するρの変化をグラフ化すると下図のようになる



（c） ある点での θ の変化に対する ρ の変化

ハフ変換を用いた直線検出

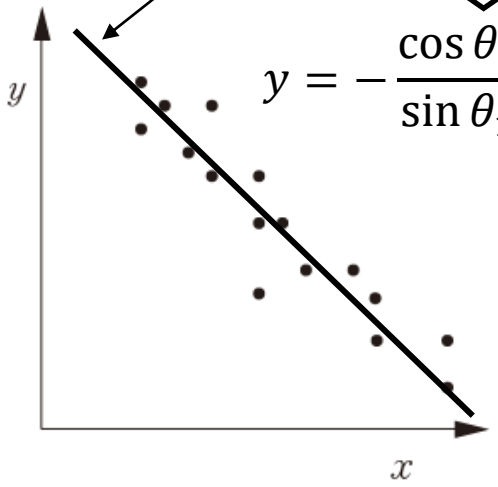
- 与えられた全ての点に対して，同様の曲線を描く
- 曲線が最も多く交わる点(θ_1, ρ_1)を見つければ，1本の直線を特定できる



（d） 直線の候補の決定

$$\rho_1 = x\cos\theta_1 + y\sin\theta_1$$

$$y = -\frac{\cos\theta_1}{\sin\theta_1}x + \frac{\rho_1}{\sin\theta_1}$$



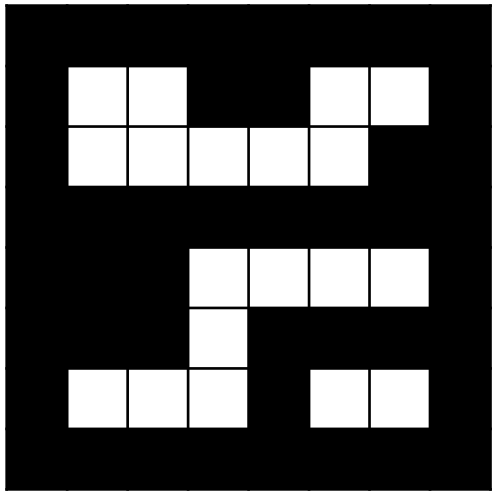
（a） 線の候補となる画素

画像処理プログラミング5
2値画像に対する処理

ラベリング処理

- ラベリング処理　cv2.connectedComponents
  - 第1引数：入力画像データ
  - 戻り値1：ラベル数（背景画素を含む）
  - 戻り値2：ラベリング後の画像データ

```
num_labels, labels = cv2.connectedComponents(binary_img)
```



binary_img

0	0	0	0	0	0	0	0
0	1	1	0	0	1	1	0
0	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0
0	0	0	2	2	2	0	0
0	0	0	2	0	0	0	0
0	2	2	2	0	3	3	0
0	0	0	0	0	0	0	0

labels(num_labels=4)

ラベリング処理の例

```
#2値画像の例（0が背景、255が物体）
binary_img = np.array([  [0,   0,   0,   0,   0,   0,   0,   0],
                         [0, 255, 255,   0,   0, 255, 255,   0],
                         [0, 255, 255, 255, 255, 255,   0,   0],
                         [0,   0,   0,   0,   0,   0,   0,   0],
                         [0,   0,   0, 255, 255, 255,   0,   0],
                         [0,   0,   0, 255,   0,   0,   0,   0],
                         [0, 255, 255, 255,   0, 255, 255,   0],
                         [0,   0,   0,   0,   0,   0,   0,   0],
                         ], dtype=np.uint8)

#ラベリング処理
num_labels, labels = cv2.connectedComponents(binary_img)

#ラベリングされた画像を数値で表示
print("ラベリングされた画像:")
print(labels)
print("検出された連結成分の数:", num_labels - 1)  # 背景を除くために-1

#(背景を除いて）ラベルごとに濃度値を割り当てる
labels_img = (labels / num_labels * 255).astype(np.uint8)

#結果を表示する
show_images(binary_img, labels_img, 'Original Binary Image', 'Labeled Image')
```

膨張・収縮処理のカーネル

- カーネルの生成　cv2.getStructuringElement
  - 第1引数：カーネルの種類
    - cv2.MORPH_RECT（矩形）
    - cv2.MORPH_CROSS（十字）
    - cv2.MORPH_ELLIPSE（楕円）
  - 第2引数：カーネルサイズ

```
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
```

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

cv2.MORPH_RECT

0	0	1	0	0
0	0	1	0	0
1	1	1	1	1
0	0	1	0	0
0	0	1	0	0

cv2.MORPH_CROSS

0	0	1	0	0
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
0	0	1	0	0

cv2.MORPH_ELLIPSE

膨張処理・収縮処理

- 膨張処理　cv2.dilate
  - 第1引数：入力画像データ
  - 第2引数：前ページで生成したカーネル
  - 第3引数：膨張回数

```
dilate_img = cv2.dilate(binary_img, kernel, iterations=2)
```

- 収縮処理　cv2.erode
  - 第1引数：入力画像データ
  - 第2引数：前ページで生成したカーネル
  - 第3引数：収縮回数

```
erode_img = cv2.erode(binary_img, kernel, iterations=2)
```

膨張処理の例

```
#2値画像の例（0が背景、255が物体）
binary_img2 = np.array([ [0,   0,   0,   0,   0,   0,   0,   0],
                         [0,   0,   0,   0,   0,   0,   0,   0],
                         [0,   0,   0,   0, 255, 255,   0,   0],
                         [0, 255, 255, 255, 255, 255,   0,   0],
                         [0, 255, 255, 255, 255, 255,   0,   0],
                         [0, 255, 255, 255, 255, 255,   0,   0],
                         [0,   0,   0,   0,   0,   0,   0,   0],
                         [0,   0,   0,   0,   0,   0,   0,   0],
                         ], dtype=np.uint8)


# カーネルを定義
dilate_kernel = cv2.getStructuringElement(cv2.MORPH_CROSS, (3, 3))

# 膨張処理
dilate_img = cv2.dilate(binary_img2, dilate_kernel, iterations=1)

# 結果を数値で表示する
print(dilate_img)

#結果を表示する
show_images(binary_img2, dilate_img, 'Original Binary Image', 'Dilated Image')
```

収縮処理の例

```
#2値画像の例（0が背景、255が物体）
binary_img3 = np.array([ [0,   0,   0,   0,   0,   0,   0,   0],
                         [0, 255, 255, 255, 255, 255, 255,   0],
                         [0, 255, 255, 255, 255, 255, 255,   0],
                         [0, 255, 255, 255, 255, 255, 255,   0],
                         [0,   0,   0, 255, 255, 255, 255,   0],
                         [0,   0,   0, 255, 255, 255, 255,   0],
                         [0,   0,   0, 255, 255, 255, 255,   0],
                         [0,   0,   0,   0,   0,   0,   0,   0],
                         ], dtype=np.uint8)


# カーネルを定義
erode_kernel = cv2.getStructuringElement(cv2.MORPH_CROSS, (3, 3))

# 収縮処理
erode_img = cv2.erode(binary_img3, erode_kernel, iterations=1)

# 収縮処理
print(erode_img)

#結果を表示する
show_images(binary_img3, erode_img, 'Original Binary Image', 'Eroded Image')
```

プログラミング演習

1. 「ex8-1.png」を読み込んで，
   ラベリングを行うプログラムを実装し，
   動作結果を確認してください

2. 「ex8-2.png」を読み込んで，
   クロージング処理を行うプログラムを実装し，
   動作結果を確認してください
   ノイズの除去が上手くいかなければ，
   カーネルサイズや種類を変えてみてください