第１０回

二値画像処理(3)

膨張・収縮処理

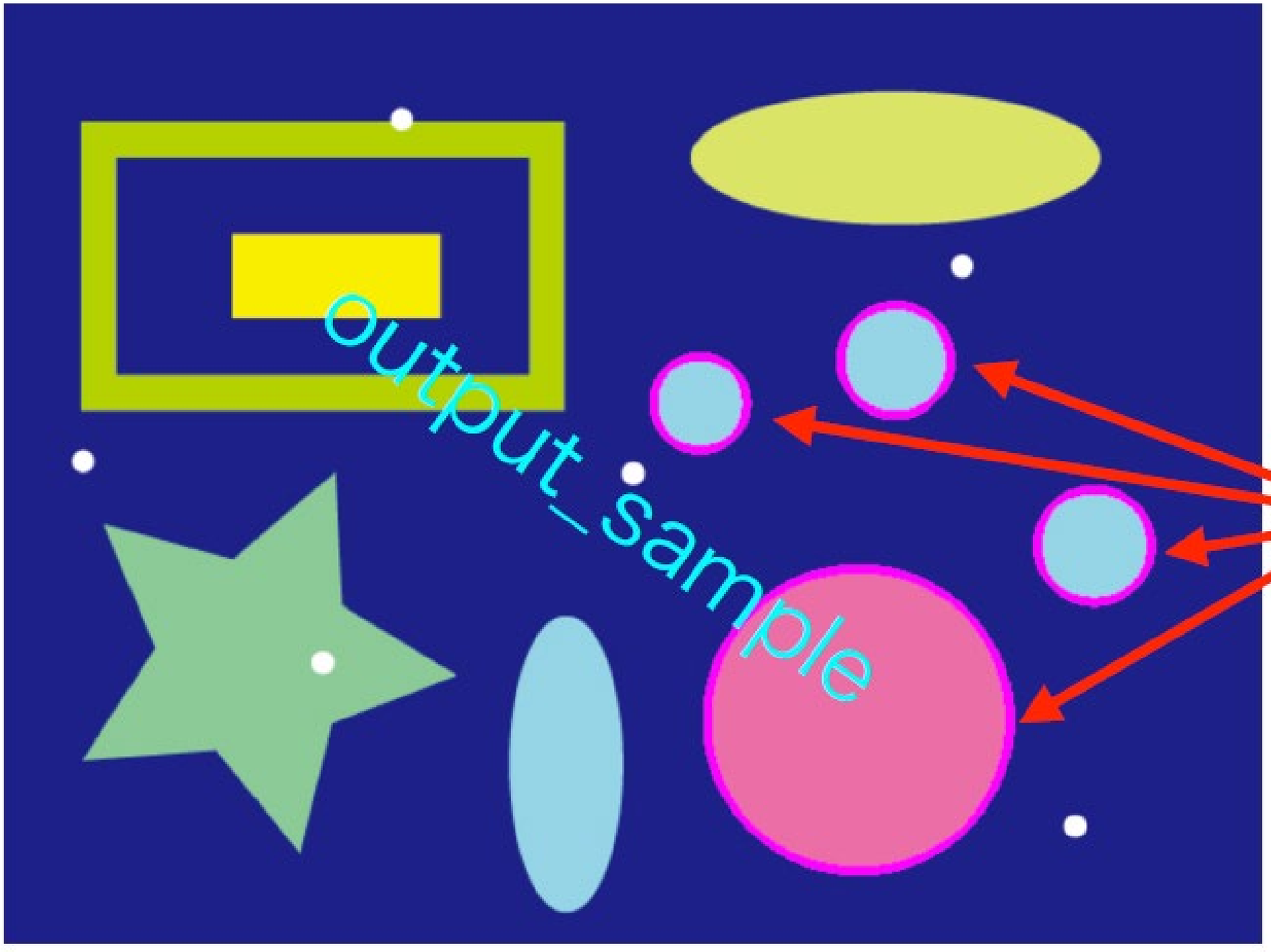
森本、澤野、塚田

第9回 課題解説

課題1

- 演習で用いた入力画像sample.jpgに対して、**円の輪郭だけを描画する**プログラムを作成し、結果画像を出力せよ
  （ただし小さな白色の円は除く）
- 提出ファイル
  - ソースプログラム：09_01_contour_学籍番号.cpp
  - 出力画像：09_01_output_学籍番号.jpg（カラー画像jpgでなくてもよい）

課題１の出力画像例



課題1のヒント

- **まず輪郭を求める**
- 各輪郭に対し**面積**を求める
- 周囲長も求めて、各輪郭の**円形度**を算出する
  - 円周率πは 定数 M_PI で用いることができる
- 面積と円形度が**条件を満たすときだけ**輪郭を描画する
  - ある程度面積が大きな輪郭を対象にすれば白色の円を除去できる
  - 円かどうかの判定は 円形度 ≧ 0.8 程度でよい
- 出力画像には予め入力画像をコピーしておく
  - 輪郭を上書きする

解答例（要点のみ）

```
//面積のしきい値
#define AREA_TH (100)
//円形度のしきい値
#define CIRC_TH (0.8)
 (中略)
    //8. 条件に適合した輪郭の描画
    for (int i=0; i<contours.size(); i++) {
        //周囲長・面積・円形度の取得・表示
        double L,S,R;
        L = cv::arcLength(contours[i], true);
        S = cv::contourArea(contours[i]);
        R = 4*M_PI*S/(L*L);
       //輪郭の描画
        if (S > AREA_TH && R > CIRC_TH) {
            cv::drawContours(dst_img, contours, i,
               CV_RGB(255, 0, 255), 3); // magenta
        }
    }
```

面積・円形度取得

面積と円形度の条件

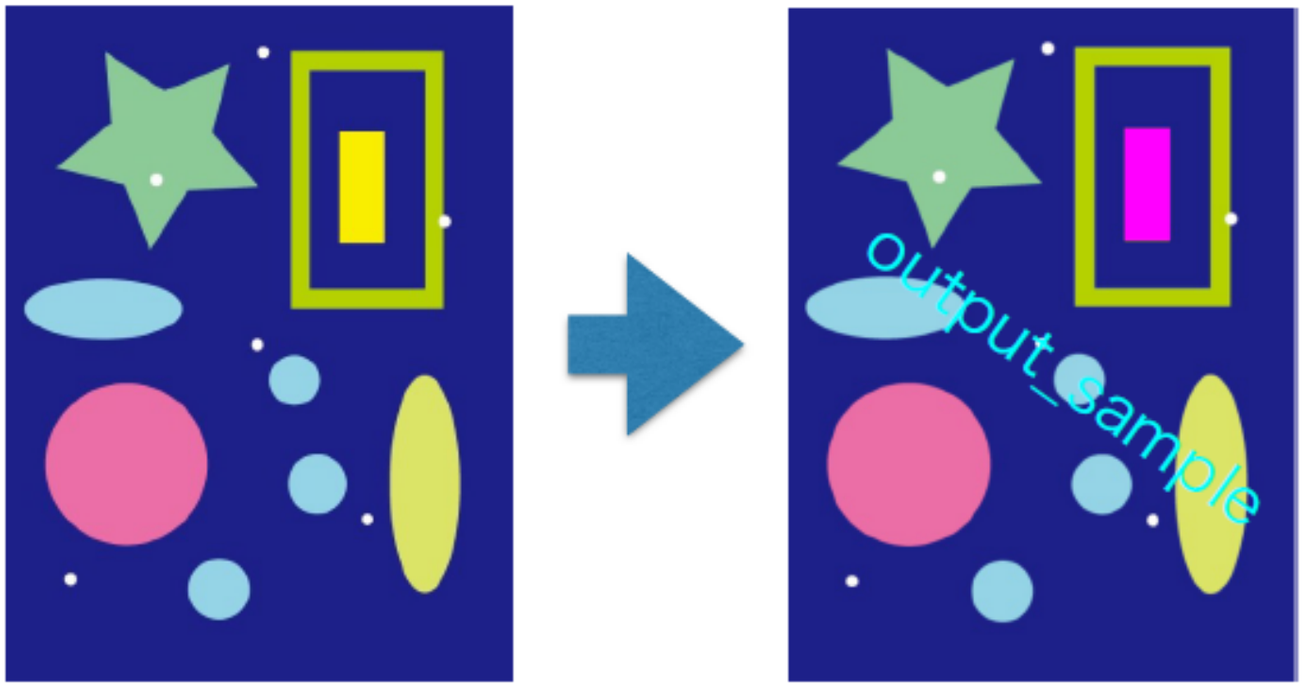
課題２（提出自由・加点対象）

- 課題１と同じ入力画像sample.jpgに対して、
  領域特徴量を用いて**指定された図形領域を**
  マゼンタで塗り潰すプログラムを作成し、
  結果画像を出力せよ　※(R,G,B)=(255,0,255)でマゼンタ
  - 但し、90度回転したsample_rotate.jpgを入力画像としても、**同じプログラムで**同じ領域が塗りつぶされること
- 提出ファイル
  - ソースプログラム：09_02_rectangle_学籍番号.cpp
  - 出力画像：09_02_output_学籍番号.jpg（カラー画像jpgでなくてもよい）

課題２の指定領域と出力画像例

sample.jpg

出力画像例



sample_rotate.jpg

sample_rotate.jpgでも
同じプログラムで
同じ領域が塗りつぶされる

課題2のヒント

- 課題１と同じように輪郭追跡から領域特徴量を求める
- 塗り潰しはdrawContoursで線幅を-1（負の値）にすれば実施される
- 指定領域だけを描画するための領域特徴量と条件を，試行錯誤して決める
  - 例
    - ✓ 円形度と面積を用いる
    - ✓ 外接長方形の幅・高さを用いると良い
    - ✓ 複数の条件分岐で判定する（どれか一つだけだと難しい）
    - ✓ 円形度は長方形だけを選別するためには当てにならない（円の除去には使えるが楕円は残ることが多い）

解答例（面積の一致度）

```
//面積のしきい値
#define AREA_TH (10000)
//円形度のしきい値
#define CIRC_TH (0.8)
//輪郭面積と外接長方形面積の差に関するしきい値
#define DIFF_TH (500)

        //周囲長・面積・円形度の取得・表示
        double L,S,R;
        L = cv::arcLength(contours[i], true);
        S = cv::contourArea(contours[i]);
        R = 4*M_PI*S/(L*L);
        //外接長方形の取得
        cv::Rect bb = cv::boundingRect(contours[i]);
        //外接長方形の面積を計算
        int area = bb.width * bb.height;
        //輪郭の描画
        if (S < AREA_TH && R < CIRC_TH && abs(S - area) < DIFF_TH) {
            cv::drawContours(dst_img, contours, i, CV_RGB(255, 0, 255),
-1); // magenta
        }
```

外接長方形から面積

面積と円形度の条件

課題救済措置アナウンス

- 対象　第8回までの未達課題
  チャレンジ課題は除く
- 未達課題はTAよりアナウンス
- 提出締め切り
  KX　6月27日（月）23:59
  KK　6月28日（火）23:59
- 通常提出よりも点数は低い！

第１０回

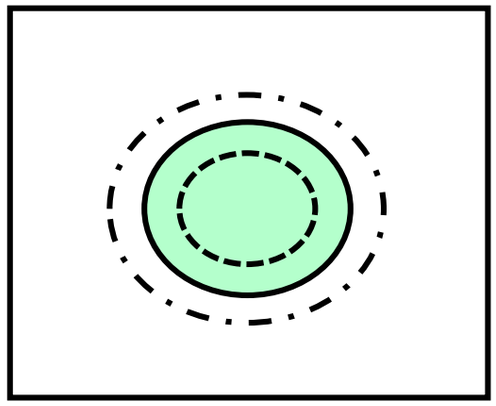
二値画像処理(3)

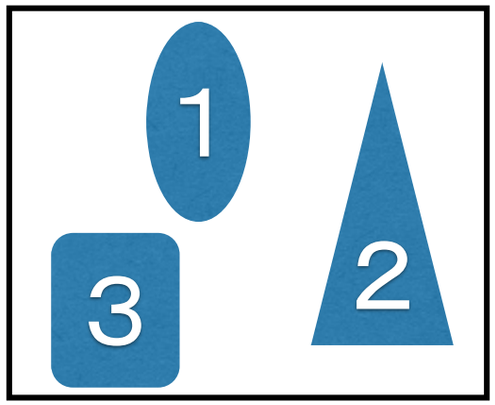
膨張・収縮処理

森本、澤野、塚田

今日の内容

- 講義: 膨張収縮処理，ラベリング
- 演習: 膨張収縮処理によるノイズ除去
  - cv::erode関数
  - cv::dilate関数





これまでに

第８回　二値画像処理（１）

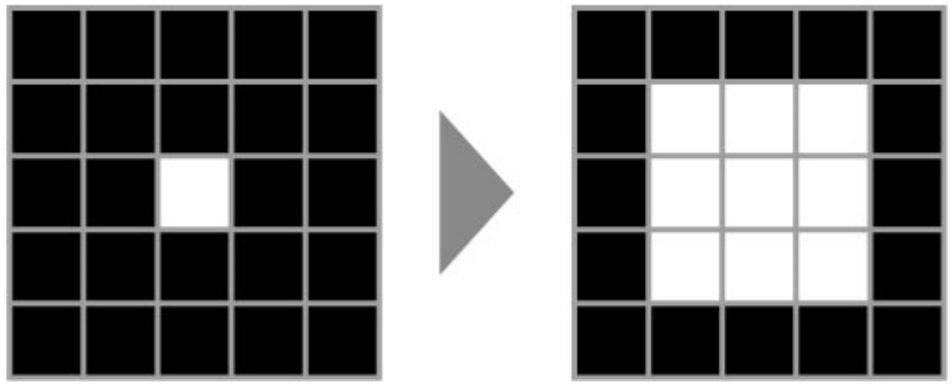
二値化　（固定しきい値法、P-タイル法、判別分析法）

第９回　二値画像処理（２）

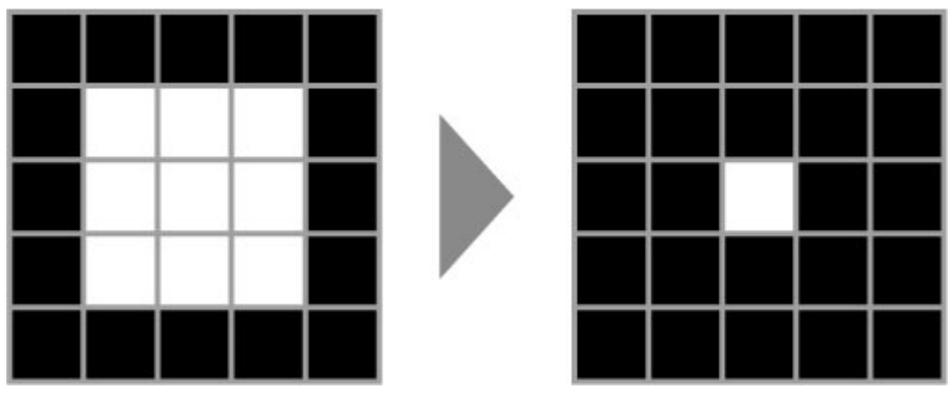
輪郭追跡、連結性、領域特徴量

膨張・収縮処理 (p. 65)

- 膨張処理: 背景または穴に接する対象の画素に，画素を一回り加える処理
  - 複数の連結成分どうしが繋がる
  - 連結性を保存しない

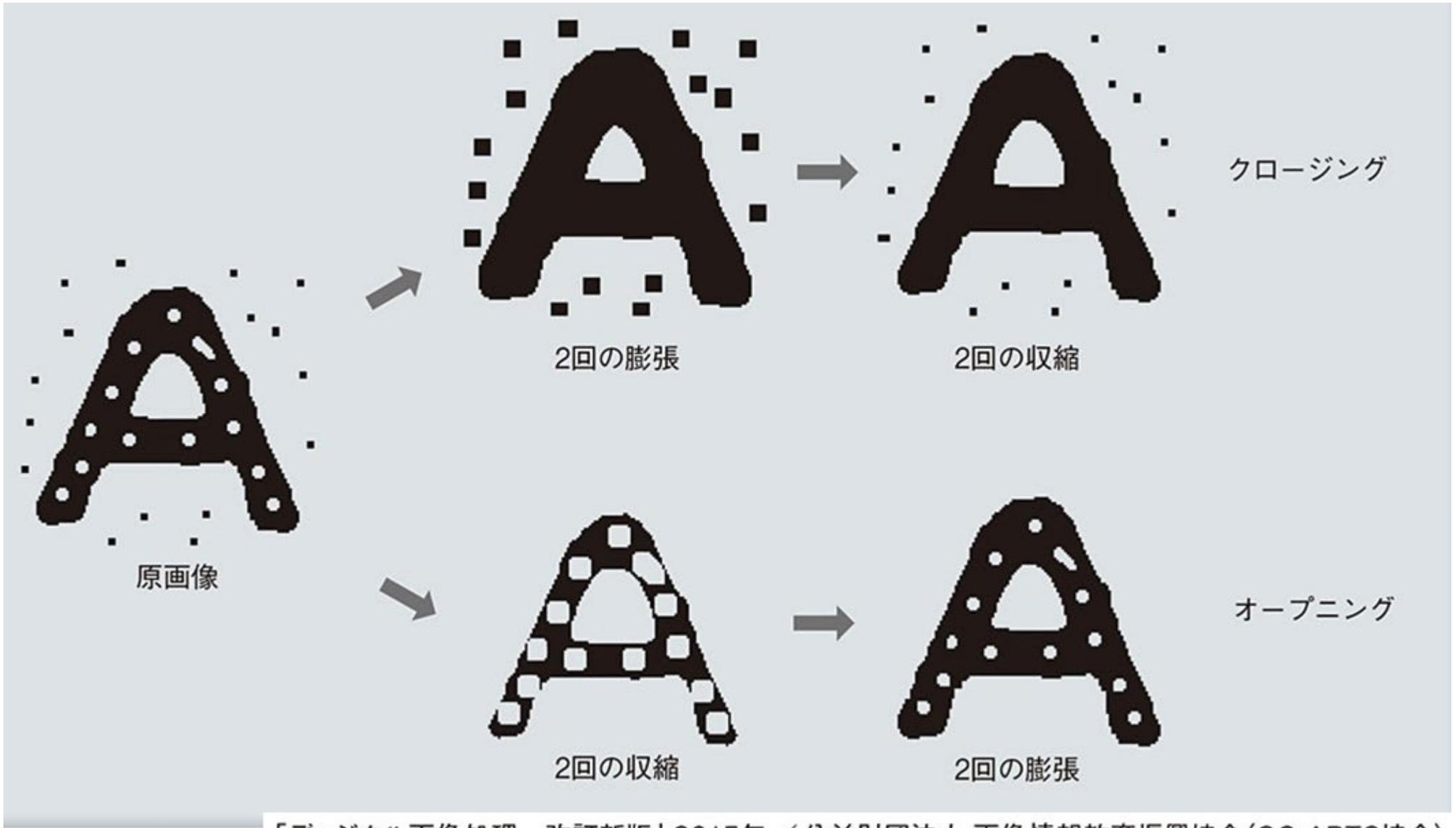


- 収縮処理: 一回り削る処理
  - やがて消える
  - 連結性を保存しない



クロージングとオープニング

- クロージング: 同じ回数だけ膨張して収縮
- オープニング: 同じ回数だけ収縮して膨張



「ディジタル画像処理　改訂新版」2015年 ／ 公益財団法人 画像情報教育振興協会(CG-ARTS協会)

注意事項

① オープニングとクロージングの処理の順序を
入れ替えた場合，同じ結果であるとは限らない

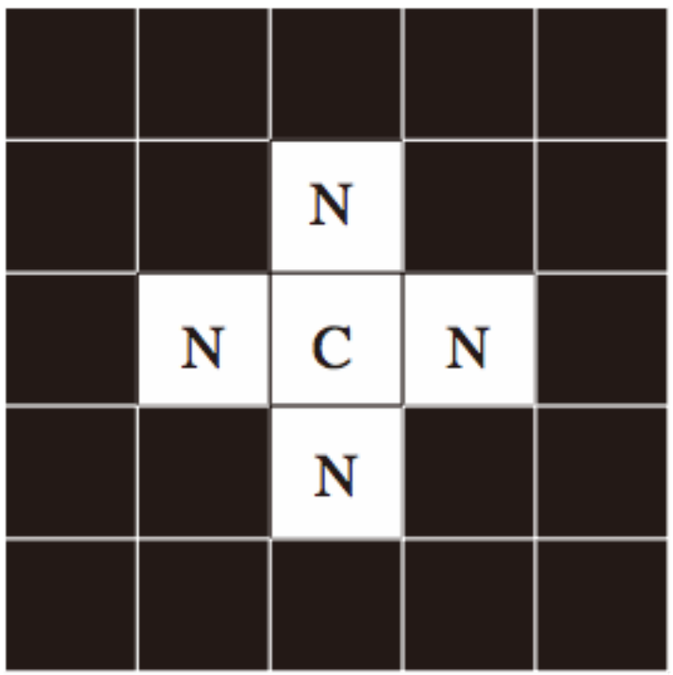
② 膨張・収縮の回数は、どれくらいの大きさの穴と
連結成分をノイズとして削除したいかに応じて決
める

③ 連結性（4連結、8連結）の違いによる影響に注意

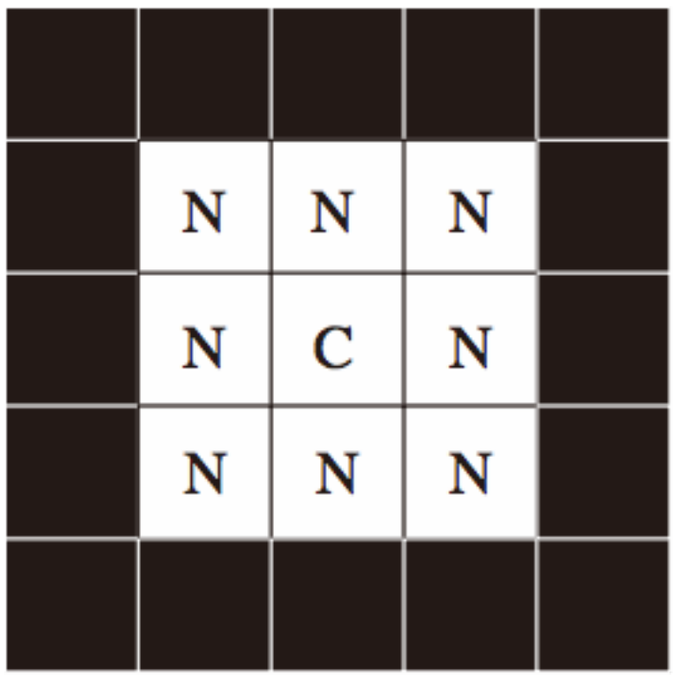
復習

連結性（p. 64）

- つながっている画素の基準
  - 斜めに位置する画素を近傍（neighbor）とみなすかどうか



4連結

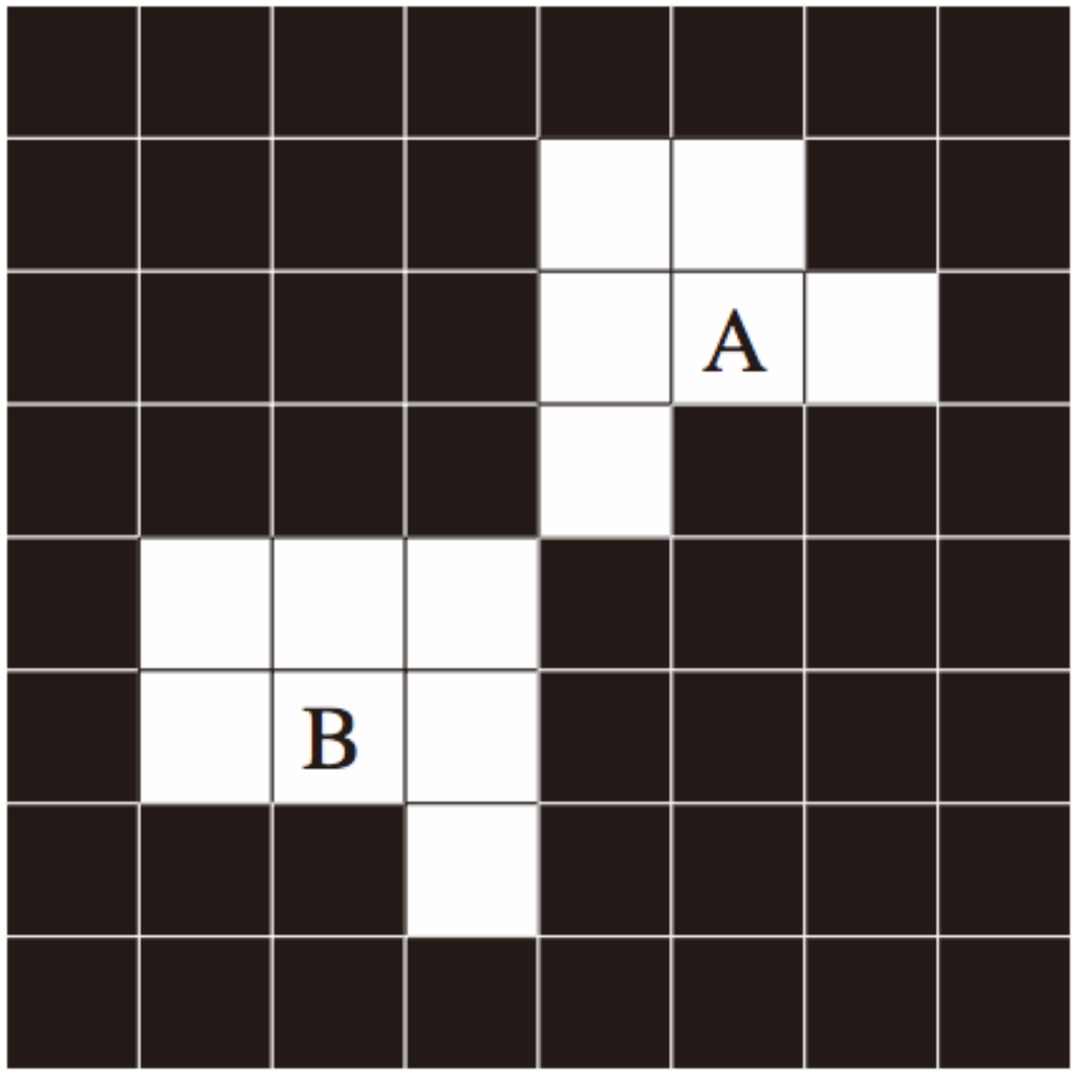


8連結

C: 注目画素に対し N: 近傍画素
左のNは４近傍　右のNは８近傍

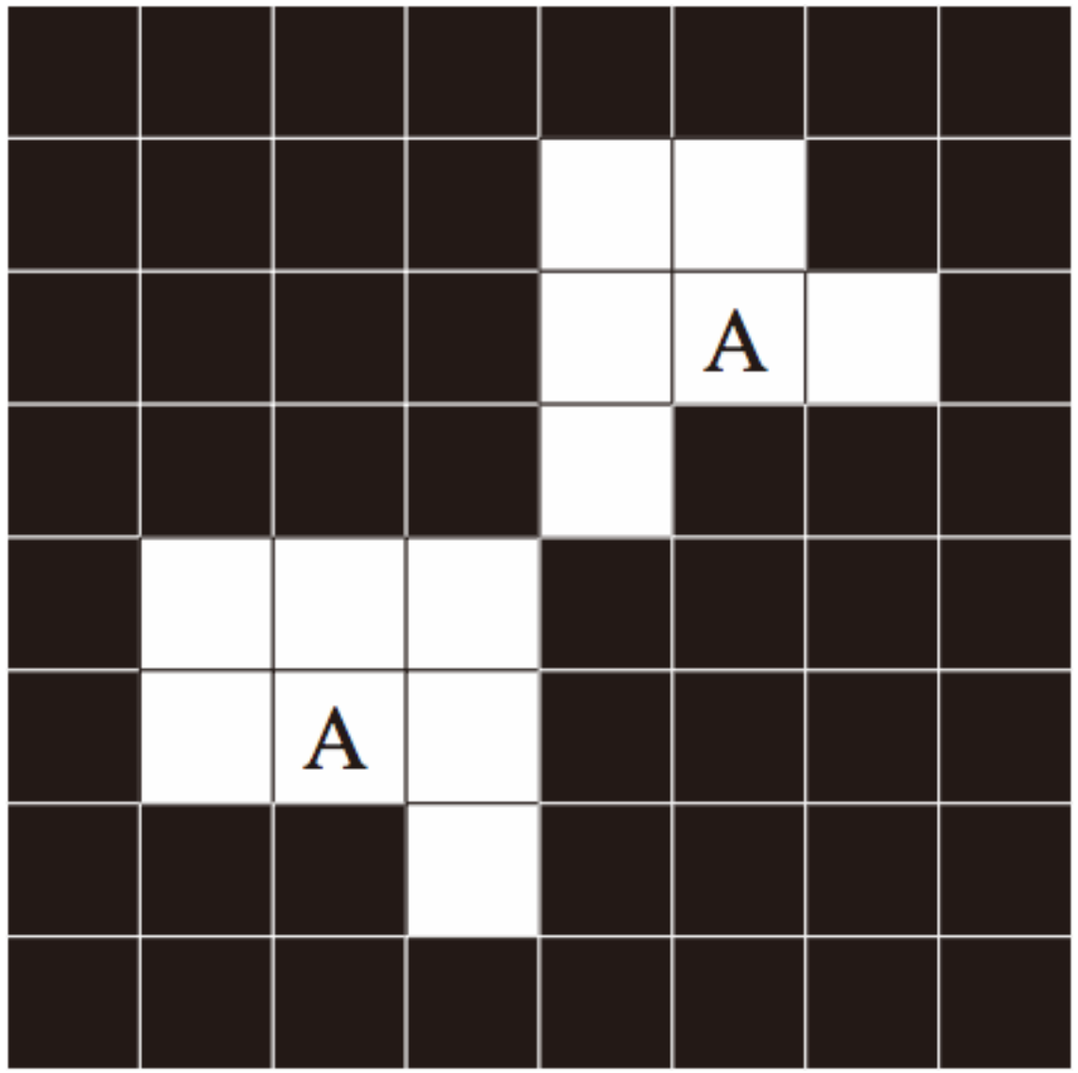
復習

連結性による同一領域の違い



4連結

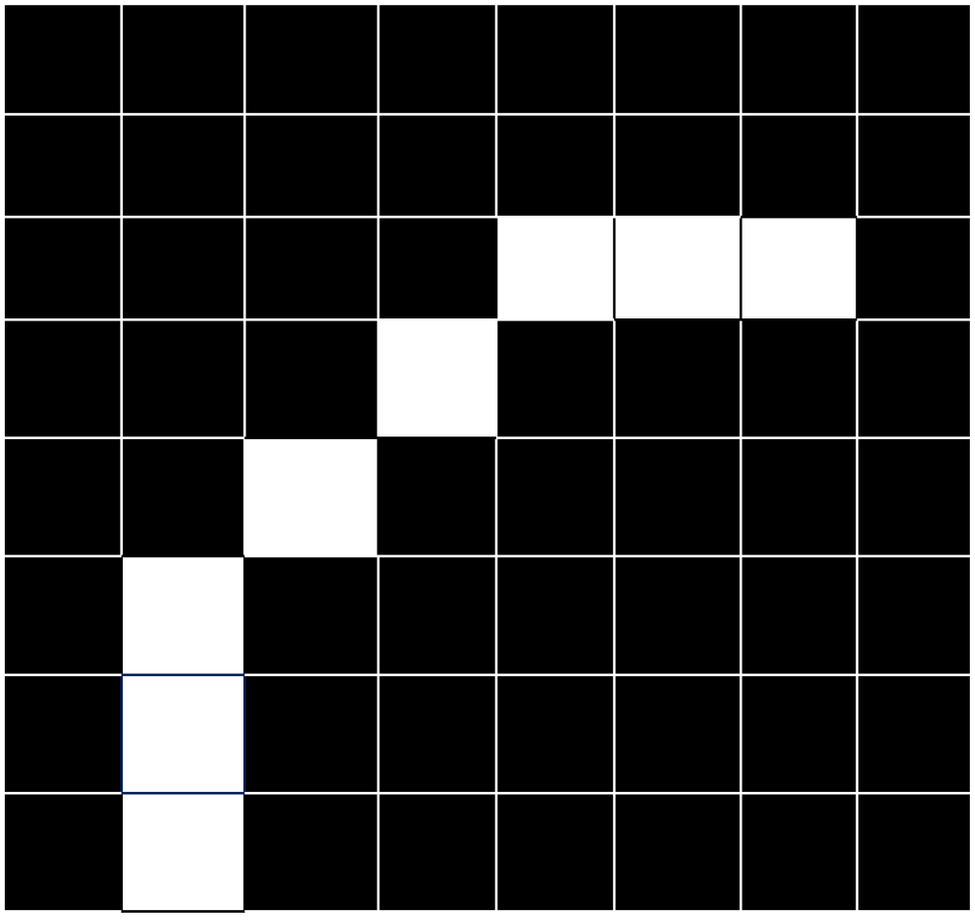
二つの領域となる



8連結

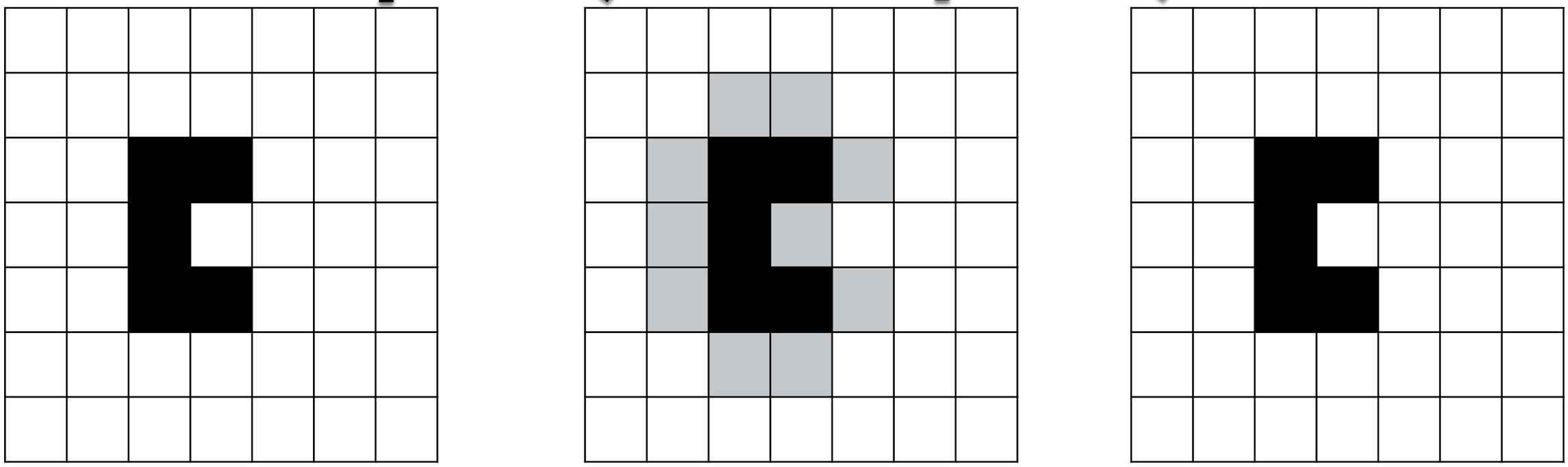
一つの領域となる

復習 この線はつながっている？

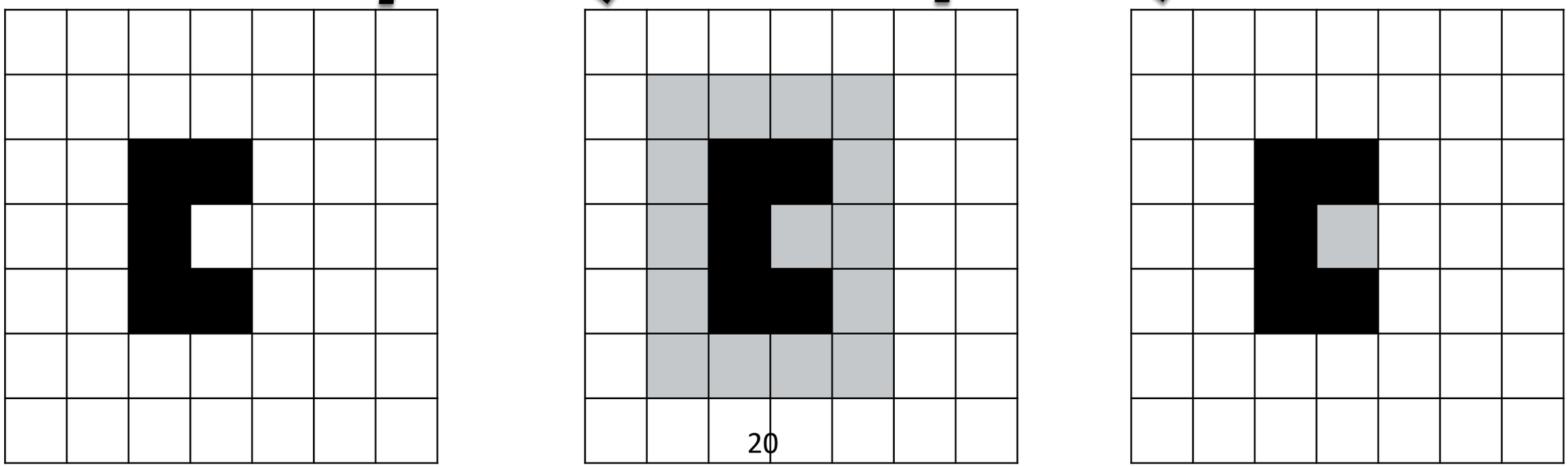


連結性によって変わる→輪郭追跡で重要

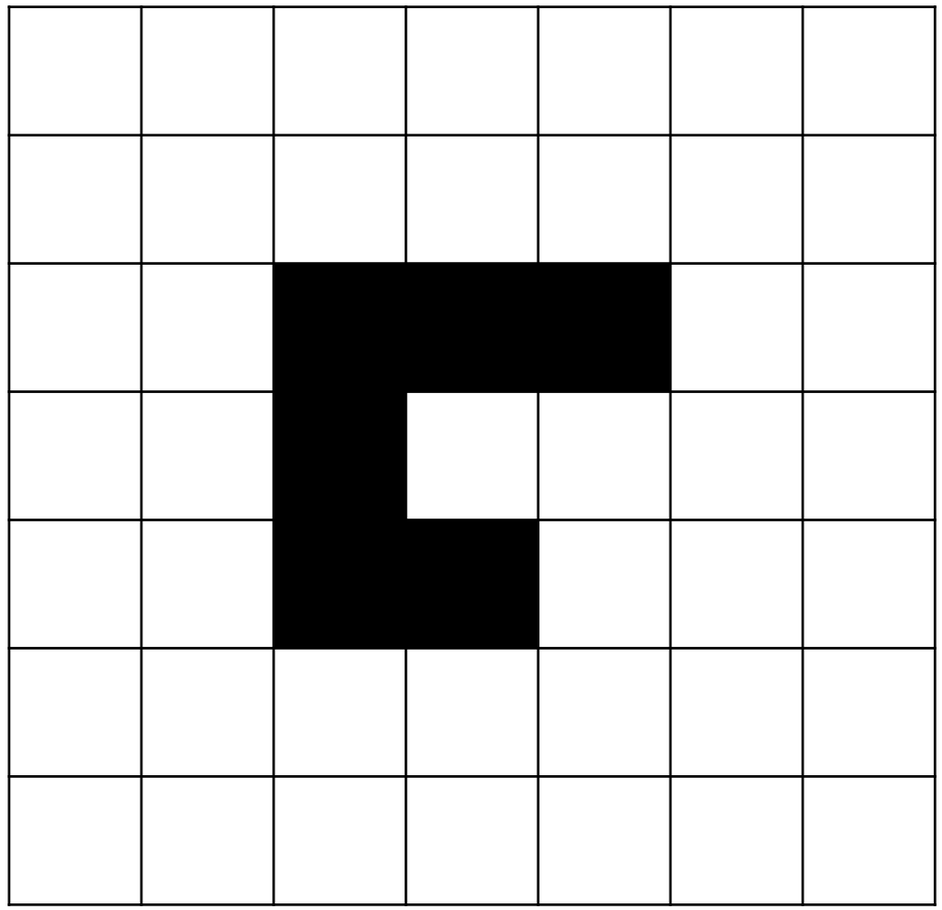
4連結　　膨張1回　　収縮1回



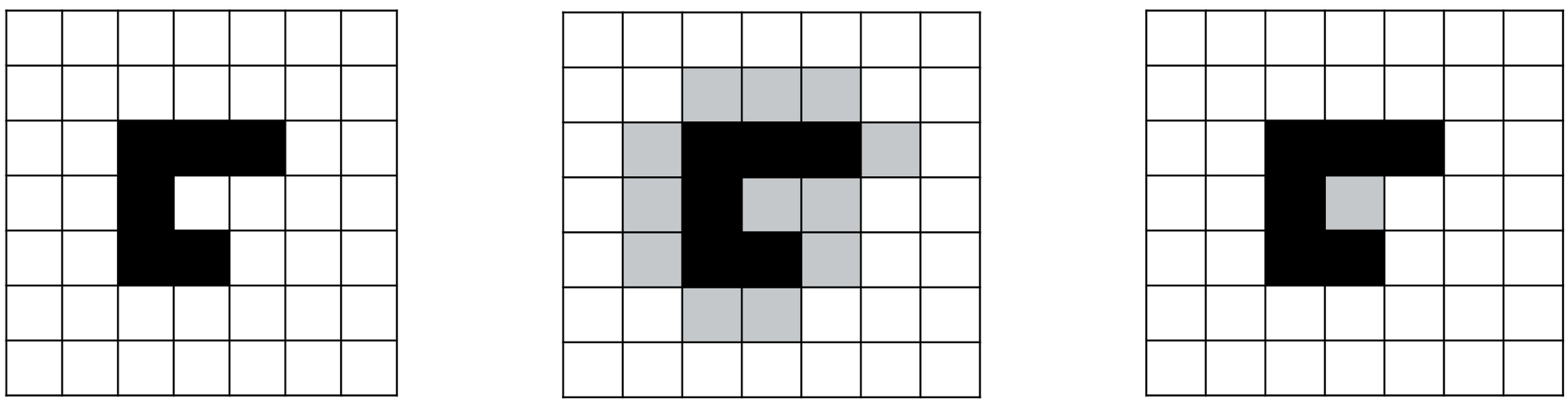
8連結　　膨張1回　　収縮1回



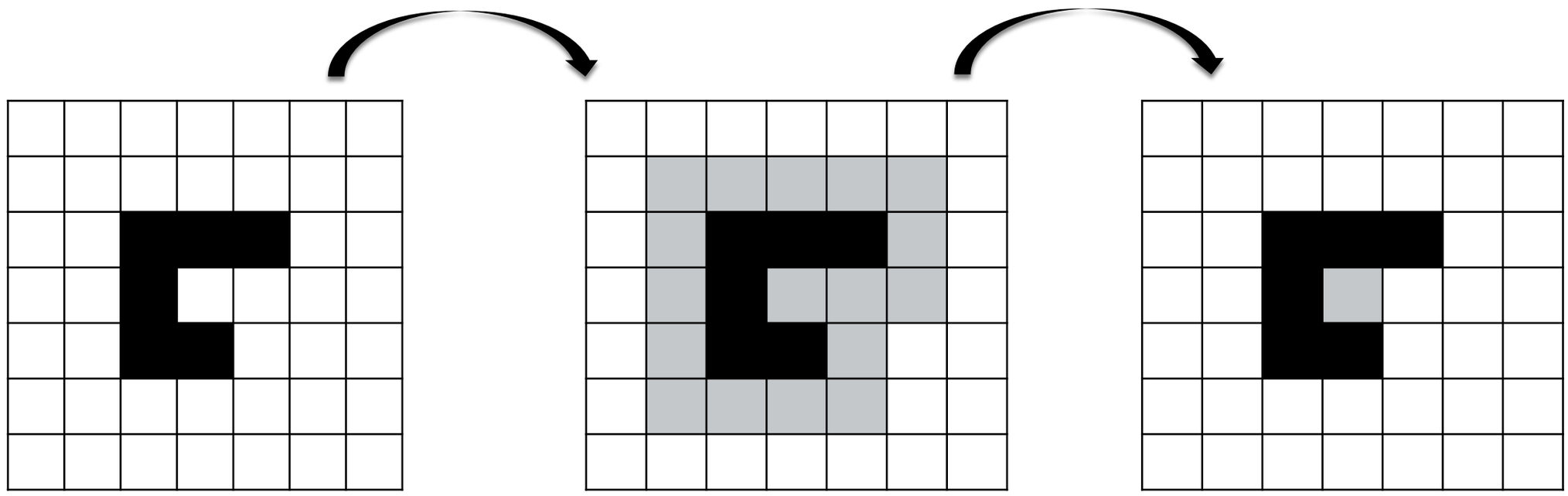
４/８連結で膨張1回、収縮1回実施後の結果を求めよ



4連結　　膨張1回　　収縮1回



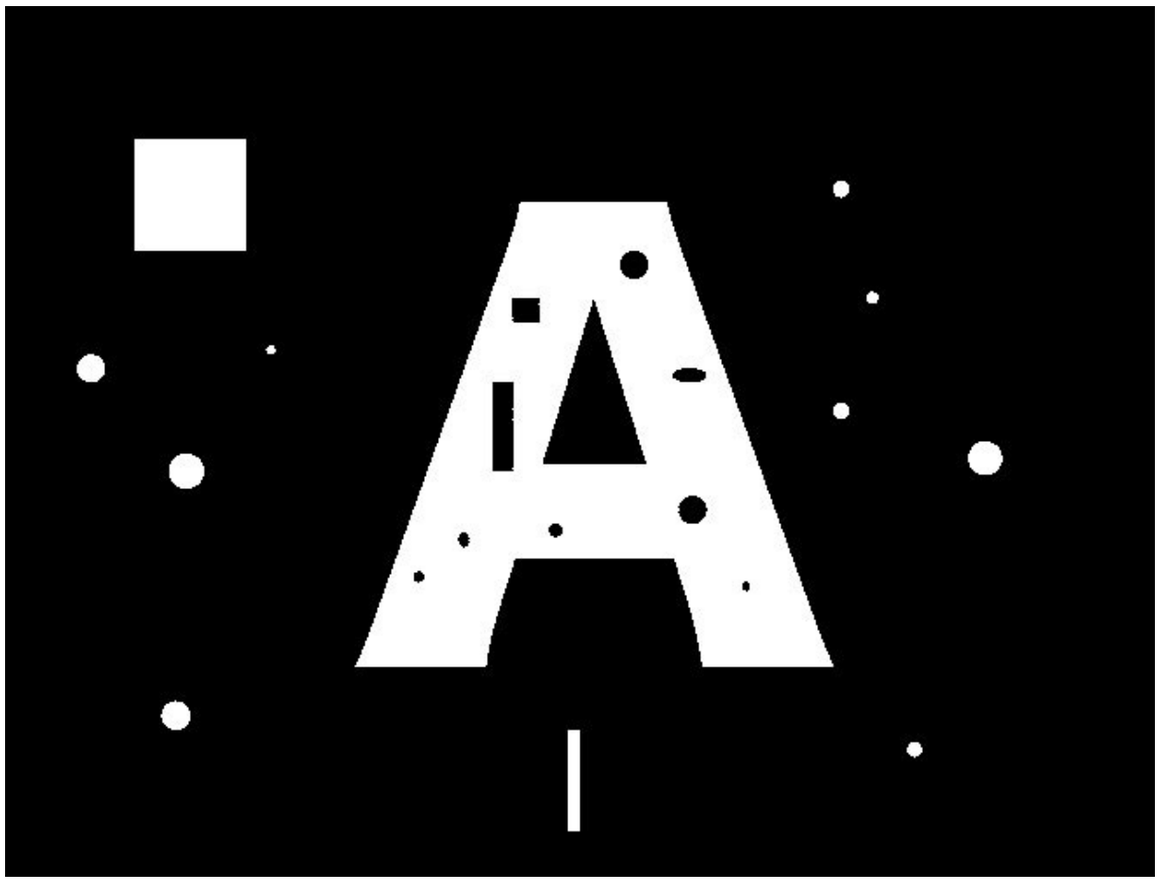
8連結　　膨張1回　　収縮1回



演習

膨張・収縮の効果確認

- プロジェクト名: morphology
- 膨張・収縮処理の実践
- 画像のダウンロード (a_img.jpg)



事前準備

1. グレースケールで画像を入力
2. 二値化 (固定閾値: 100)
3. 表示

膨張処理用の関数 (OpenCV)

- 入力画像src_imgに対して膨張処理を行う

```
cv::dilate(入力画像, 出力画像, 構造要素, 位置,　回数);
```

- 構造要素にcv::Mat()を入れることで3x3の矩形構造要素が８近傍で使用される
- 位置は構造要素の中心

```
cv::dilate(bin_img, dst_img, cv::Mat(), cv::Point(-1, -1), 10);
```

10回膨張（８近傍）の例

収縮処理用の関数 (OpenCV)

- 入力画像src_imgに対して収縮処理を行う

```
cv::erode(入力画像, 出力画像, 構造要素, 位置,　回数);
```

- 構造要素にcv::Mat()を入れることで3x3の矩形構造要素が８近傍で使用される
- 位置は構造要素の中心

```
cv::erode(bin_img, dst_img, cv::Mat(), cv::Point(-1, -1), 10);
```

10回収縮（８近傍）の例

10回膨張させよう

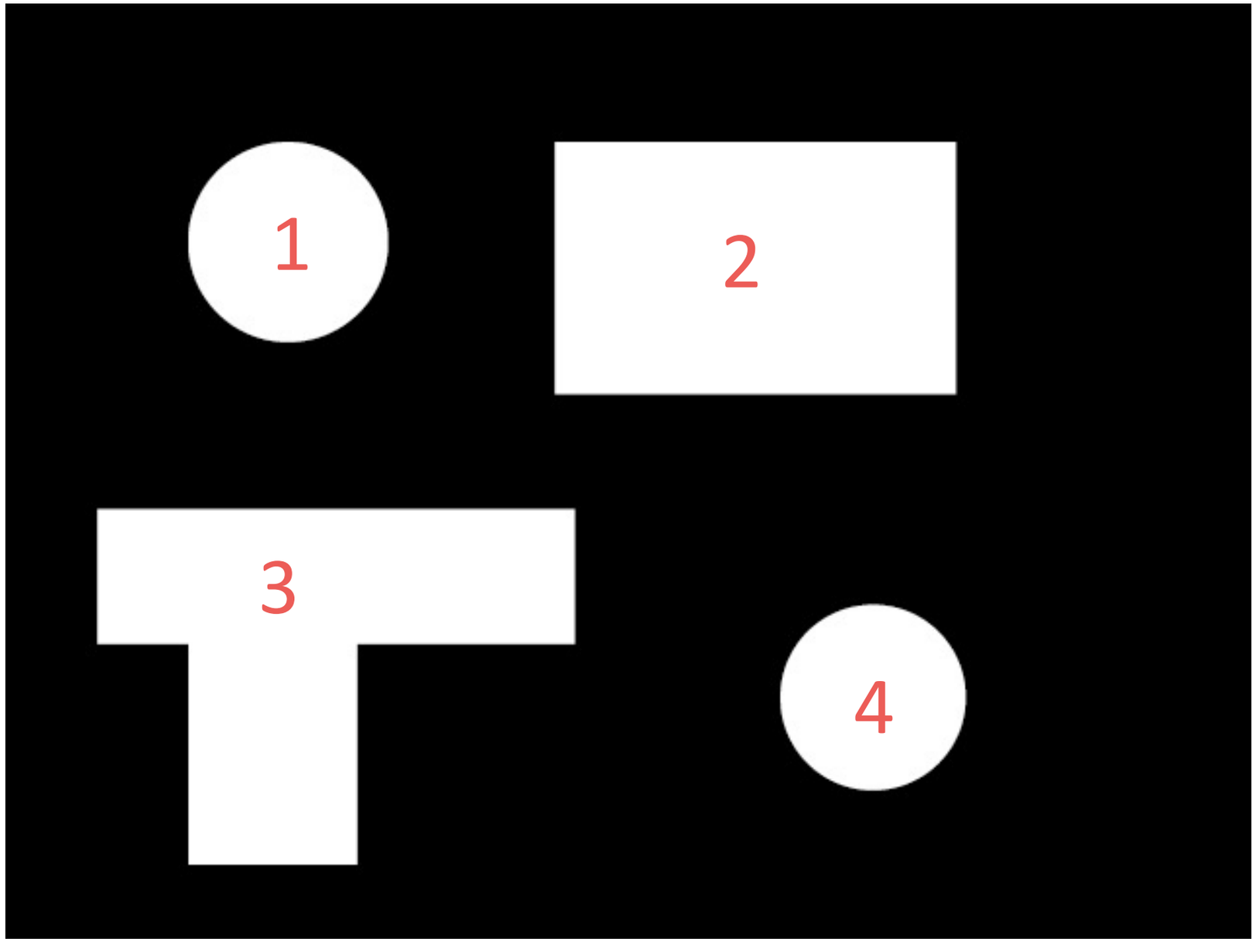
- キー入力で変化の確認

```
//膨張
for (i=0; i<10; i++) {
//膨張
cv::dilate(bin_img, dst_img, cv::Mat(), cv::Point(-1, -1), i);
cv::imshow(WINDOW_NAME_OUTPUT, dst_img); //表示
cv::waitKey(); //キー入力待ち
}
```

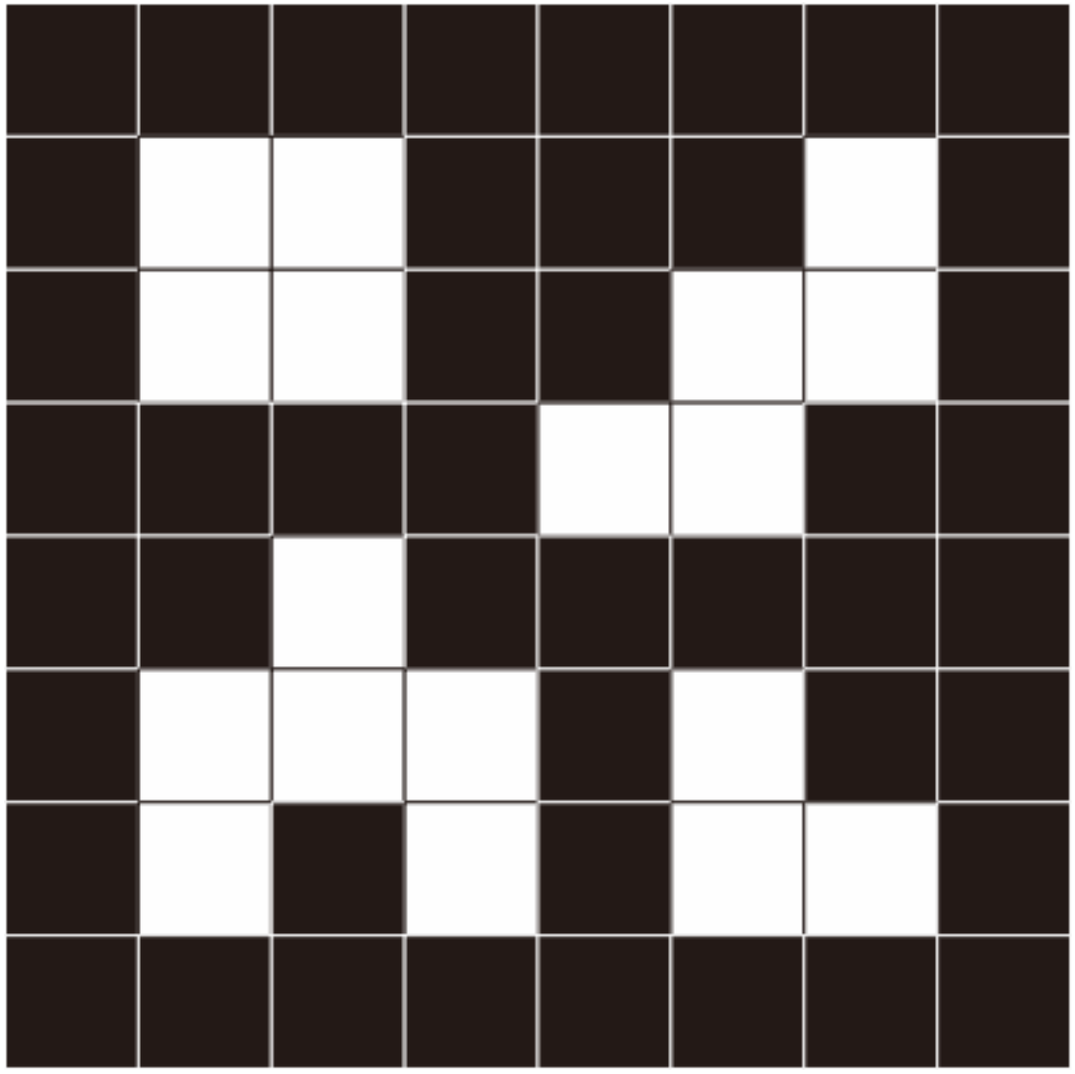
- その後，10回収縮し，確認すること
- さらに、収縮⇒膨張と順番を入れ替えて確認

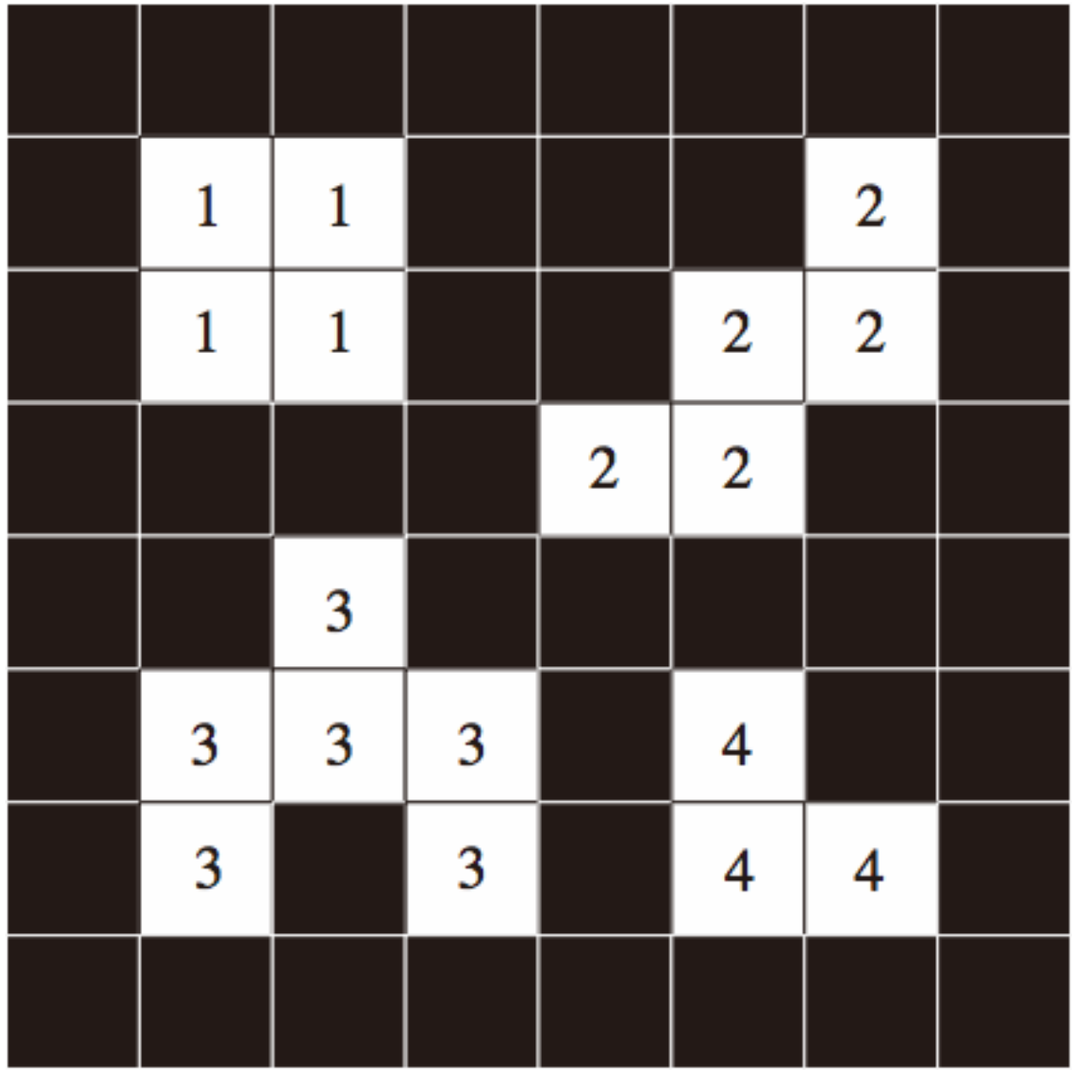
ラベリング (p. 69)

- 同じ連結成分を構成する画素に
  同じ番号(ラベル)を付与すること



ラベリング処理

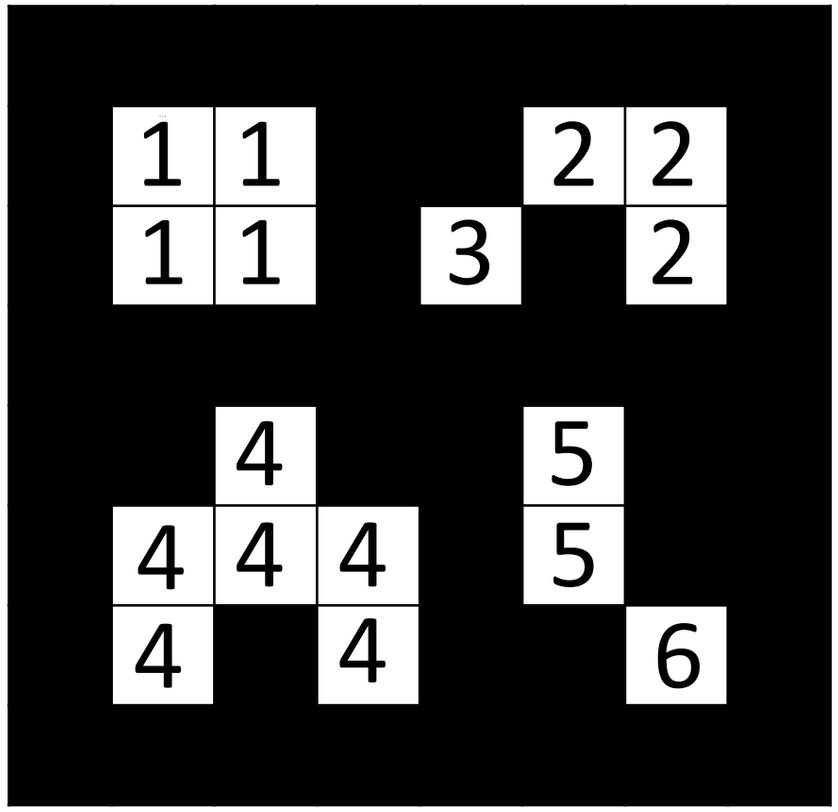




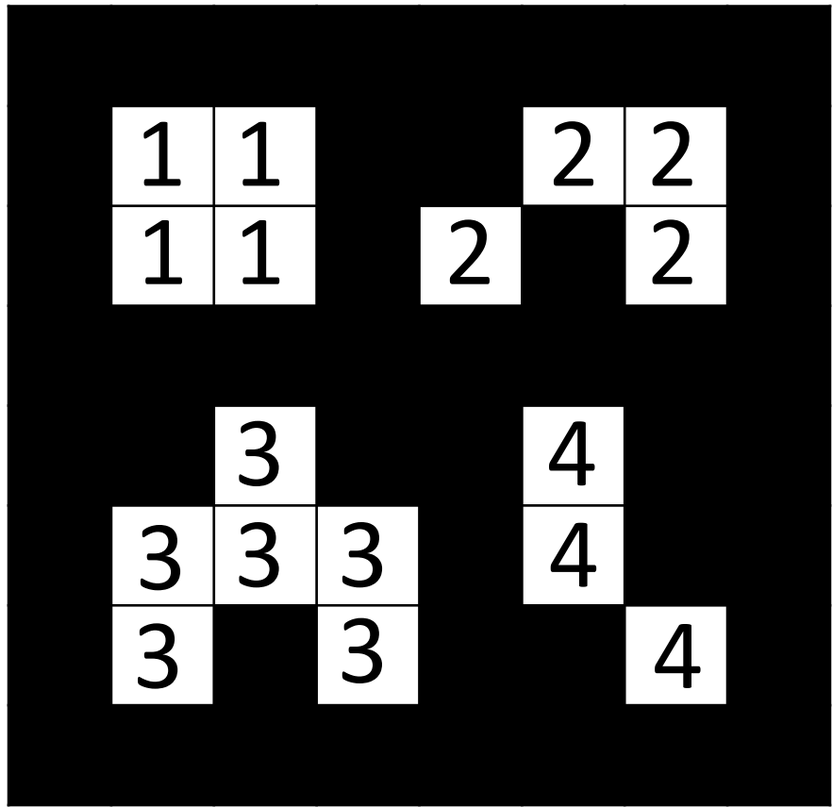
ラベリング処理

連結性に注意！

4連結ラベリング



8連結ラベリング



ラベリングアルゴリズム（簡略版）

1.注目画素を中心に、走査済み領域の対象画素を探索

2.見つかった白画素と同じラベルを付与

3.複数の白画素で異なるラベルが付与されている場合には、小さい方のラベルを付与した後、大きい値のラベルを探索して小さい値に書き換える

走査	済み	領域
	C	

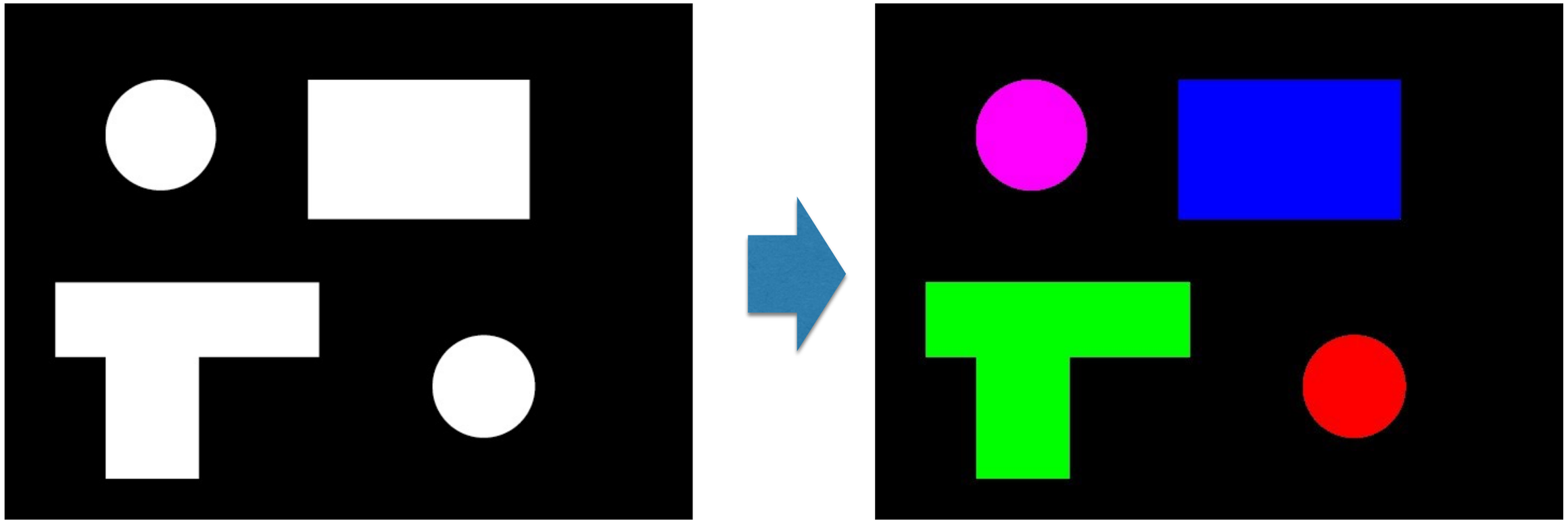
1	1	1		2		
		1		2	2	2
		C				

1	1	1		2		
		1		2	2	2
		1	C			

1	1	1		1		
		1		1	1	1
		1	1	C		

cv::findContours 関数を利用したラベリング

- 対象をラベリングしてラベルに応じて異なる色を塗る
- 以下をダウンロード
  - labeling.jpg
  - labeling4student.zip



```
 8      #include <stdio.h>
 9      #include <iostream>
10      #include <opencv2/opencv.hpp>
11
12      //ファイル
13      #define FILE_NAME "labeling.jpg"
14      //ウィンドウ名
15      #define WINDOW_NAME_INPUT "input"
16      #define WINDOW_NAME_BINARY "binary"
17      #define WINDOW_NAME_OUTPUT "output"
18
19      //二値化閾値
20      #define BIN_TH (100)
21
22
23      int main(int argc, const char * argv[]) {
24          //1. 画像の宣言（入力画像，グレースケール画像，二値画像，一時的な画像，出力画像）
25          cv::Mat src_img, gray_img, bin_img, tmp_img, dst_img;
26
27          //色値の設定
28
29          //2. 輪郭の座標リストの宣言
30          std::vector< std::vector< cv::Point > > contours;
31
32          //3. 画像の入力（カラーで入力）
33          src_img = cv::imread(FILE_NAME);
34          if (src_img.empty()) { //入力失敗の場合
35              fprintf(stderr, "File is not opened.¥n");
36              return (-1);
37          }
```

```
38
39          //4. 入力画像を結果画像にコピー
40          dst_img = src_img.clone();
41
42          //5. グレースケール化
43          cv::cvtColor(src_img, gray_img,cv::COLOR_BGR2GRAY);
44
45          //6. 二値化（固定閾値で実装. 閾値: 100）
46          cv::threshold(gray_img, bin_img, BIN_TH, 255, cv::THRESH_BINARY);
47
48          //二値画像コピー
49          tmp_img = bin_img.clone();
50
51          //7. 輪郭抽出
52          cv::findContours(tmp_img, contours, cv::RETR_LIST, cv::CHAIN_APPROX_NONE);
53
54
55          //8. 輪郭の描画
56          for (int i=0; i<contours.size(); i++) {
57              //輪郭の描画
58              cv::drawContours(dst_img, contours, i, CV_RGB(255, 0, 0), 3);
59          }
60
61          //9. 表示
62          cv::imshow(WINDOW_NAME_INPUT, src_img);
63          cv::imshow(WINDOW_NAME_BINARY, bin_img);
64          cv::imshow(WINDOW_NAME_OUTPUT, dst_img);
65          cv::waitKey();
66
67
68          return 0;
69      }
70
```

塗りつぶしに

色値の設定

- 4オブジェクト分の色値を確保
- 以下を追記 (コメント文が目印)

```
//色値の設定
cv::Scalar color[] = {CV_RGB(255, 0, 0),
                      CV_RGB(0, 255, 0),
                      CV_RGB(0, 0, 255),
                      CV_RGB(255, 0, 255)};
```

オブジェクト内部の塗りつぶし

- 輪郭を描画する関数 (記載済み)

```
cv::drawContours(dst_img, contours, i, CV_RGB(255, 0, 0), 3);
```

- 塗りつぶし用

```
cv::drawContours(dst_img, contours, i, color[i%4], cv::FILLED);
```

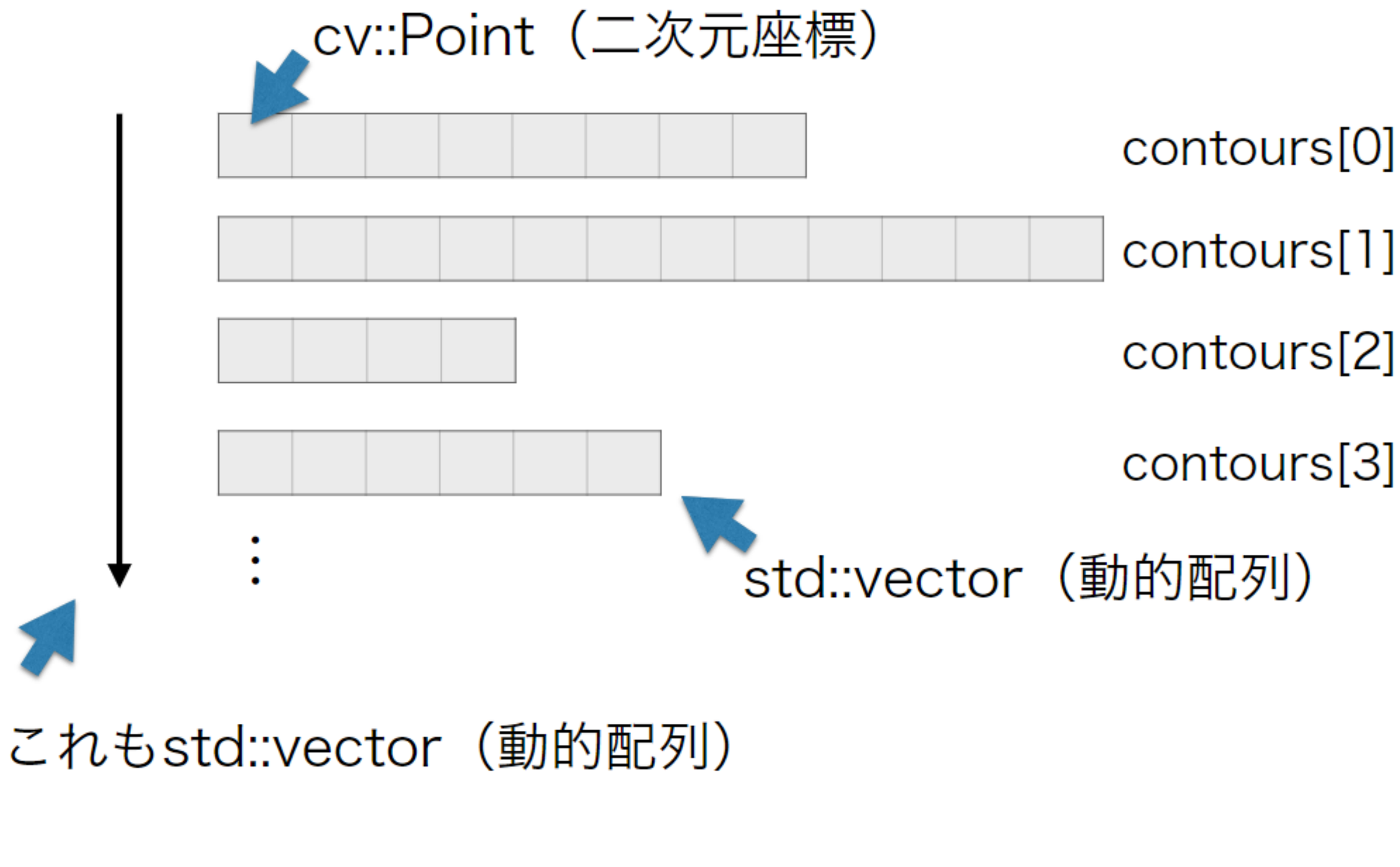
メモ

輪郭の座標リストの宣言

```
std::vector< std::vector< cv::Point > > contours;
```

- std::vector: 動的配列
  - 配列の大きさが固定されない（可変長）
- cv::Point: 座標
  - 二次元の座標
- **座標位置の 並び（＝輪郭）**をリスト化
  (cv::Point) の (std::vector) の (std::vector)

contoursの構造



メモ

輪郭追跡関数

- findContours：入力二値画像の輪郭を追跡する

```
cv::findContours(二値画像, 輪郭, 追跡モード, 輪郭近似手法);
```

- 今回の記載例

```
//7. 輪郭追跡(New!)
cv::findContours(tmp_img, contours,
   cv::RETR_LIST, cv::CHAIN_APPROX_NONE);
```

- cv::RETR_LIST: すべての輪郭追跡、リスト出力
- cv::CHAIN_APPROX_NONE: 8近傍、近似なし

輪郭追跡結果がcontoursに格納される

メモ

各輪郭へのアクセス方法

- for文でcontoursの各要素（各輪郭）にアクセス

```
//8. 輪郭の描画(New!)
for (int i=0; i<contours.size(); i++) {


}
```

- contours.size()：動的配列contoursのサイズ
  （輪郭が何個あるか）

  ✳contours[i]　でｉ番目の輪郭にアクセスできる
  （このプログラムではこの記載は使いません）

メモ

輪郭描画関数

- drawContours：輪郭を描画する

```
cv::drawContours(出力画像, 輪郭情報, 輪郭番号,
                                    輪郭の色, 描画用の線幅);
```

※線幅を負の値にすると輪郭内部も塗りつぶす

- 今回の記載例

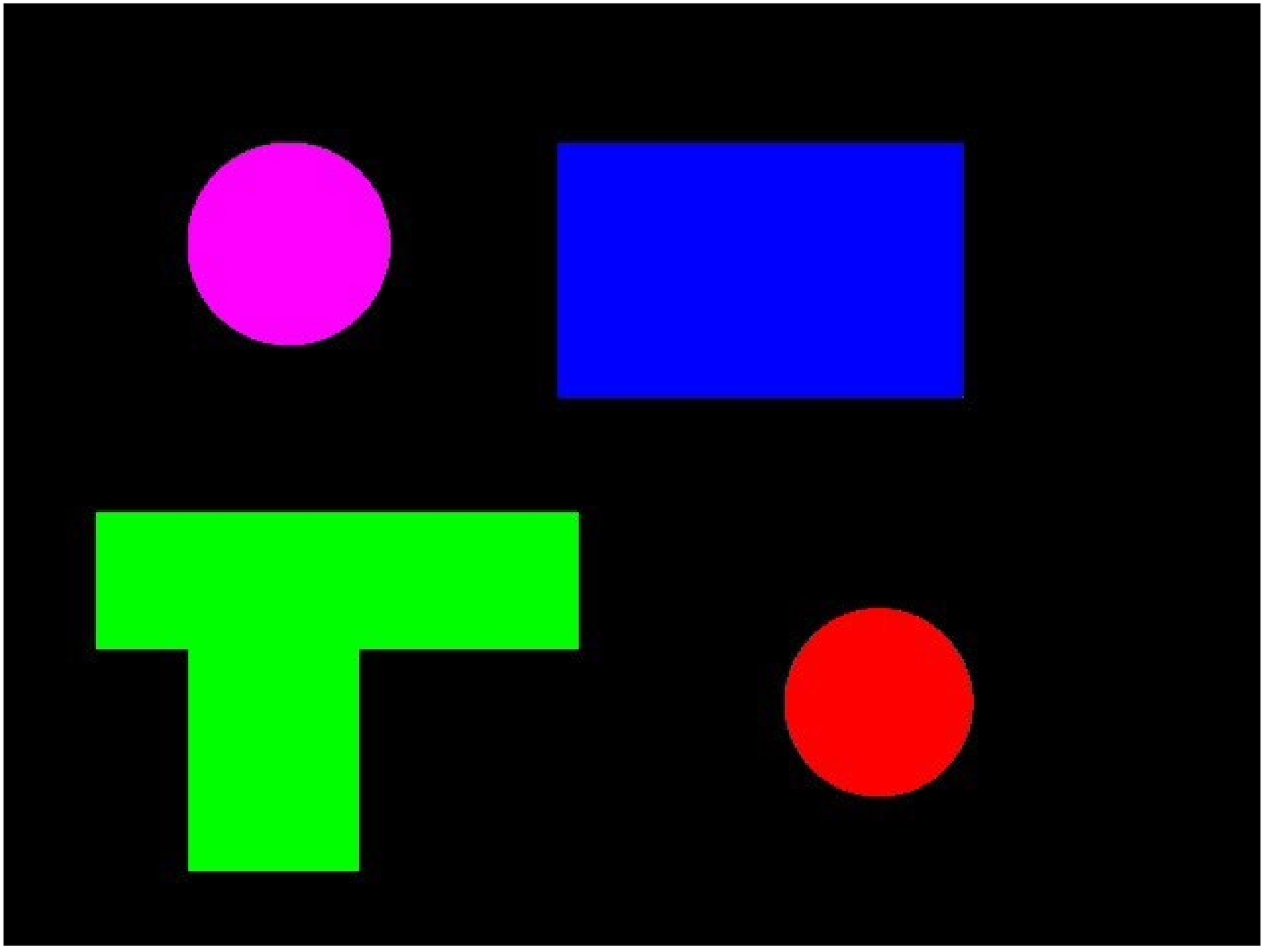
```
//輪郭の描画
cv::drawContours(dst_img, contours, i,
                                CV_RGB(255, 0, 255), 3);
```

※CV_RGBで色を指定　B=R=255でマゼンタとしている

- この描画文をfor文の中に入れる

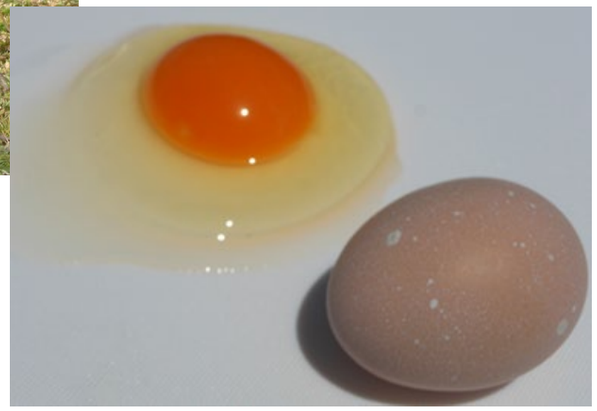
➡輪郭番号を指定して（この場合 i）描画する

塗りつぶし結果



ラベリングの適用例　名古屋コーチン卵の斑点評価

◆ 名古屋コーチン（鶏肉・鶏卵）は愛知県産のブランド



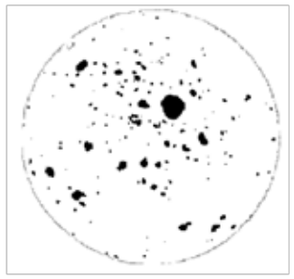
卵は『桜吹雪』
殻は桜色に白い斑点

ヒトが目視で主観的に評価
⇒ 画像処理で定量的に評価

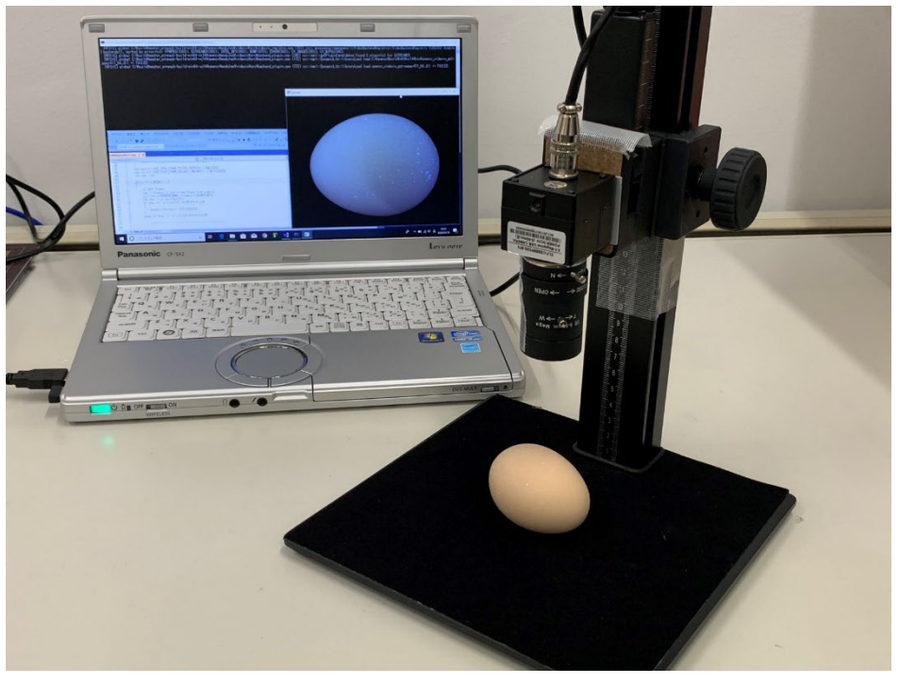
評価点	評価基準
0	白斑点なし
1	1mm未満の白斑点が認められる
2	1～2mmの白斑点が認められる 2mm以上の白斑点が5個未満認められる
3	2mm以上の白斑点が鈍端部に5個以上、 鋭端部に5個未満認められる
4	2mm以上の白斑点が鈍端部に5個以上、 鋭端部に5個未満認められる

ラベリングの適用例　名古屋コーチン卵の斑点評価

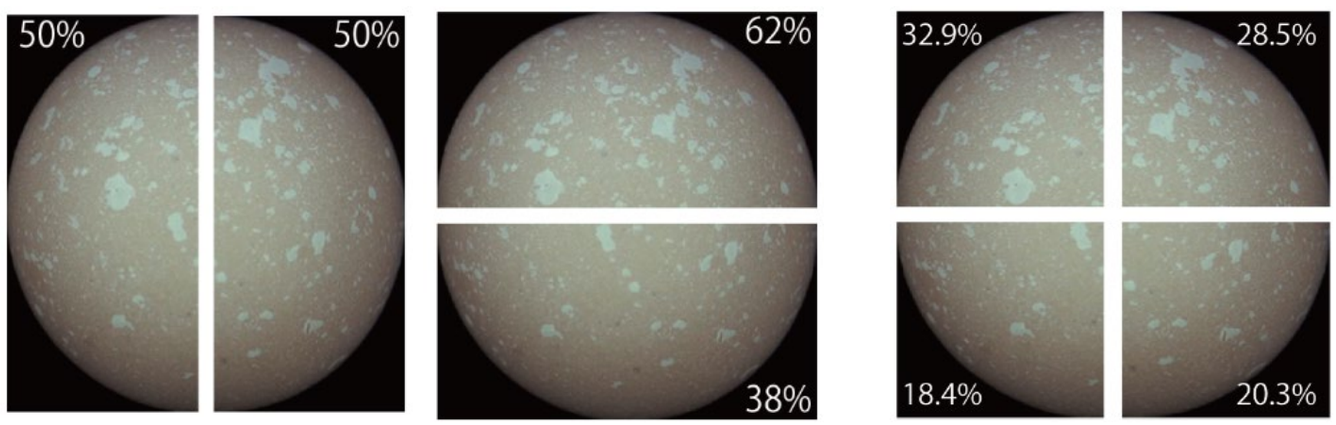
卵の画像からフィルタ処理により白斑点を抽出して
ラベリング処理により白斑点の数と面積を計測



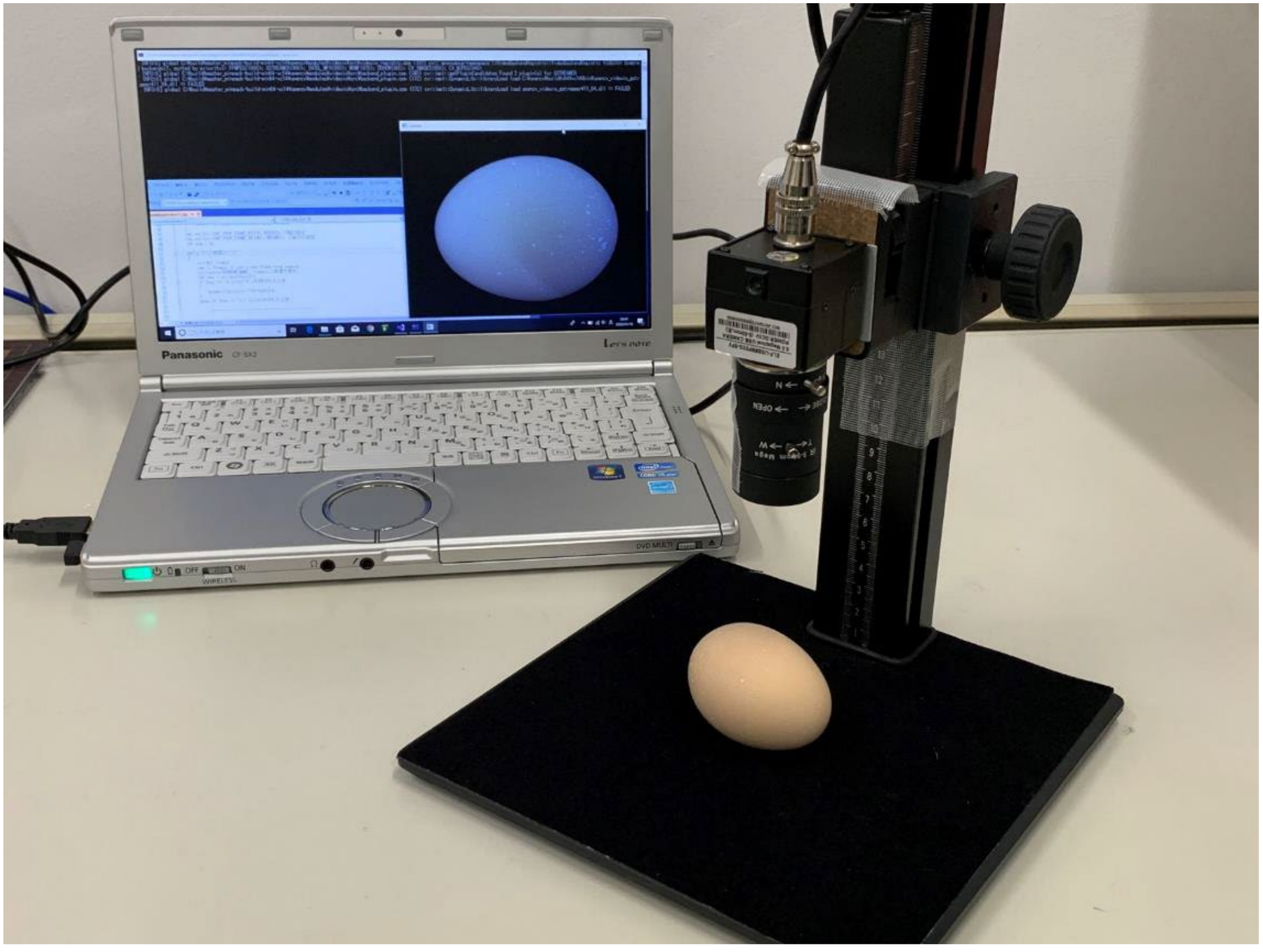
評価点	評価基準
0	白斑点なし
1	1mm未満の白斑点が認められる
2	1～2mmの白斑点が認められる 2mm以上の白斑点が5個未満認められる
3	2mm以上の白斑点が鈍端部に5個以上、 鋭端部に5個未満認められる
4	2mm以上の白斑点が鈍端部に5個以上、 鋭端部に5個未満認められる



画像を分割して分布を評価



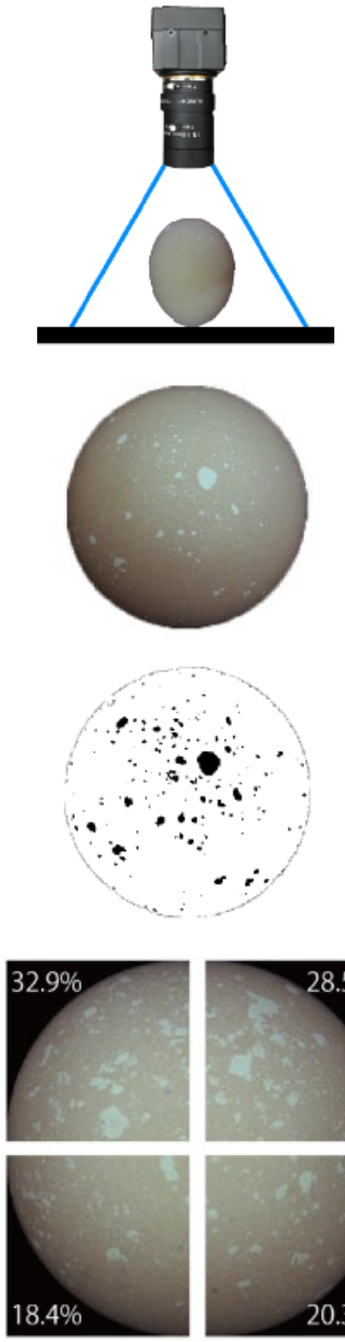
大きさ、形、重さの測定器（試作）



④ 白斑点　数と面積を計測

◆ 画像から卵の白斑点を計測して評価

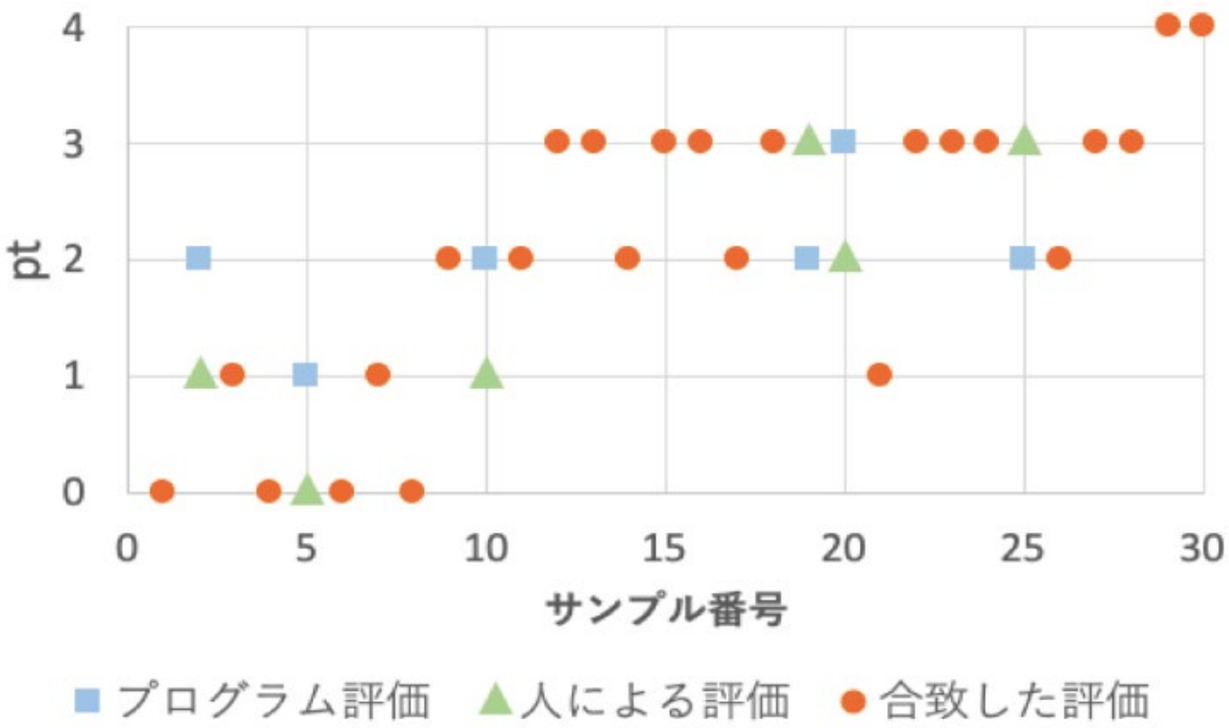
◆ 白斑点の評価手順



◆ 白斑点の評価基準

評価点	評価基準
0	白斑点なし
1	1mm未満の白斑点が認められる
2	1～2mmの白斑点が認められる 2mm以上の白斑点が5個未満認められる
3	2mm以上の白斑点が鈍端部に5個以上、 鋭端部に5個未満認められる
4	2mm以上の白斑点が鈍端部に5個以上、 鋭端部に5個未満認められる

◆ 評価結果（人との比較）

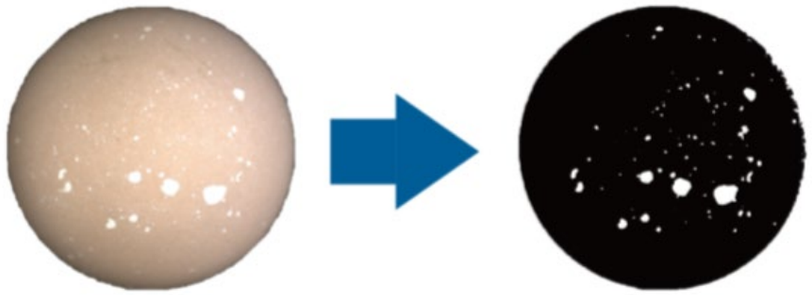


白斑点抽出の画像処理

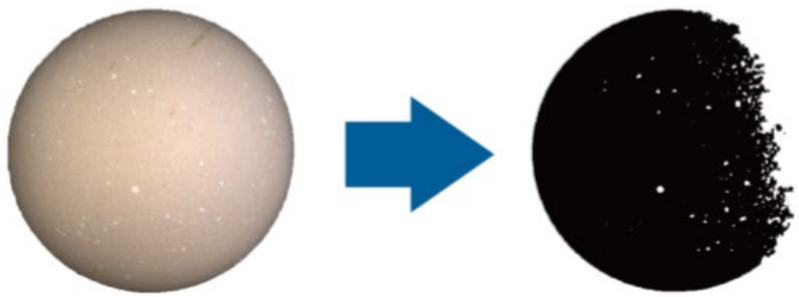
◆ 照明状況（明暗変化）の影響を受けにくい白斑点抽出

◆ しきい値による抽出

抽出に成功した場合

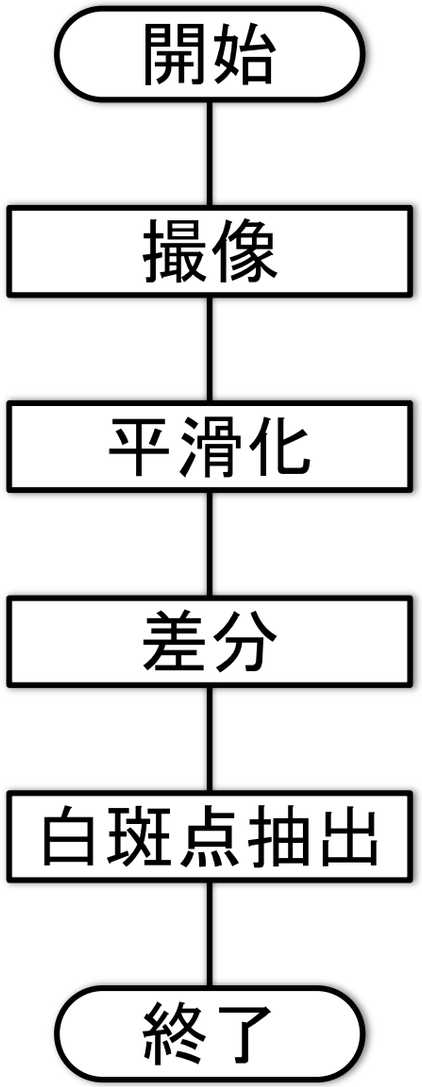


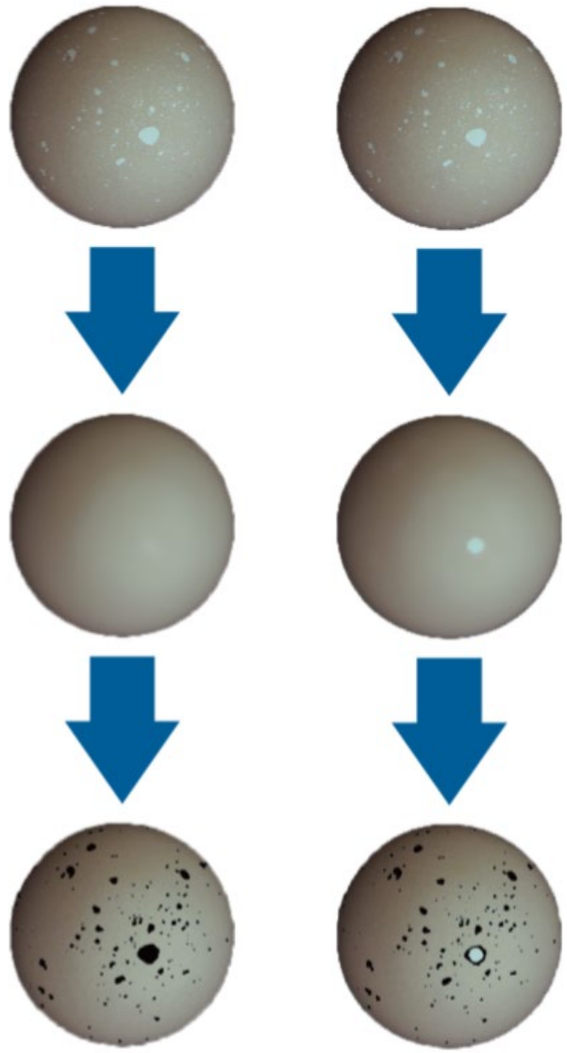
抽出に失敗した場合



色のばらつきや照明により簡単には
白斑点を抽出できない場合がある

◆ 白斑点を平滑化した画像で差分



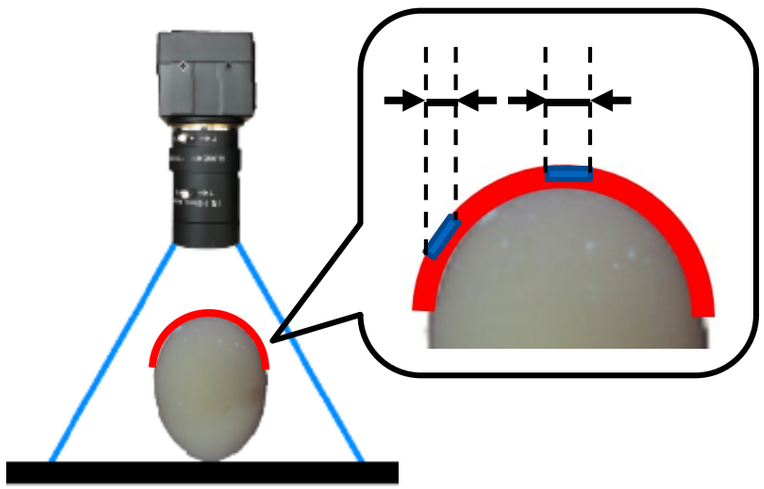


平滑化フィルタ
サイズのチューニング

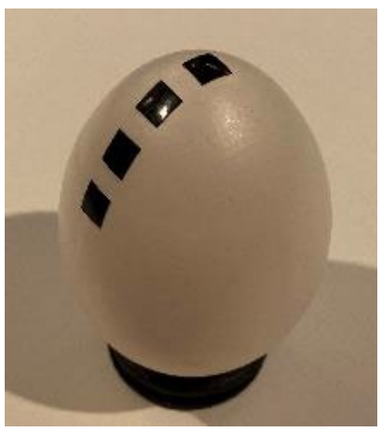
④ 白斑点面積の補正

◆ 卵形状による場所による撮像面積の違いを補正

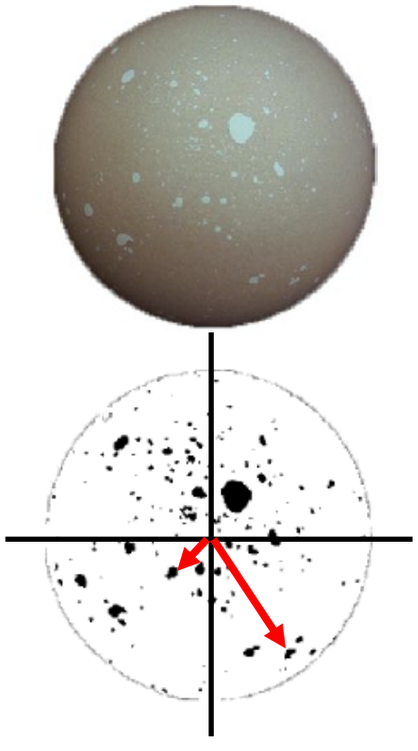
場所による撮像面積の違い



補正係数の求め方



補正方法



補正結果