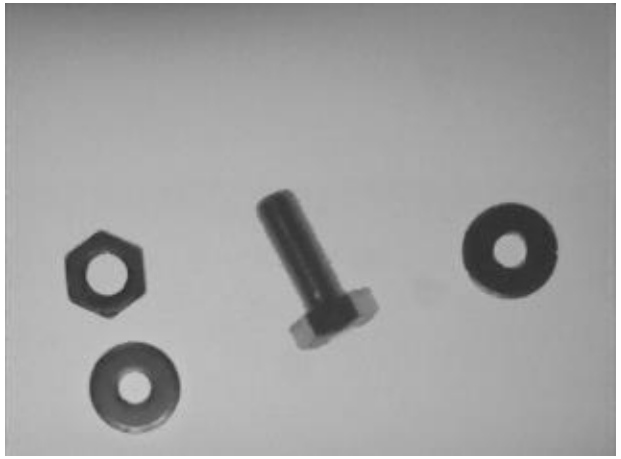
Computer Vision

Ch.7 Connected Component Labeling

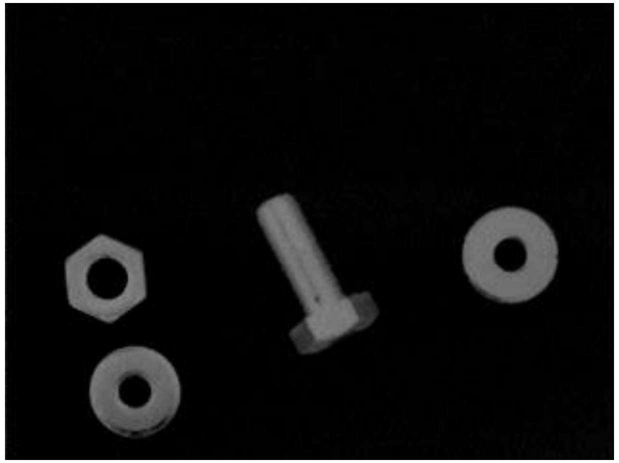
Prof. Po-Yueh Chen (陳伯岳)
E-mail: pychen@cc.ncue.edu.tw
Ext: 8440

NCUE CSIE

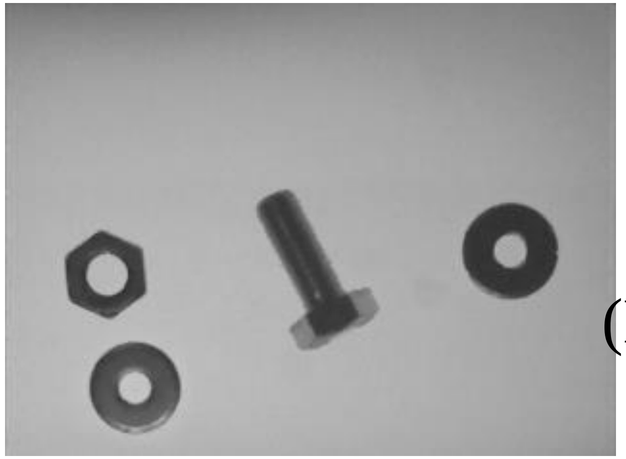
Review (1/3)



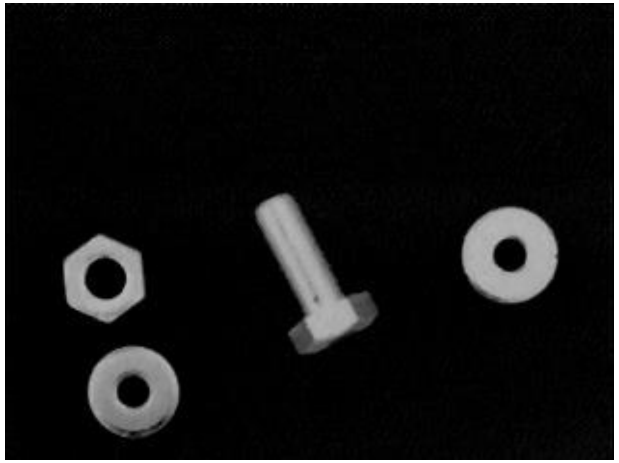
(Minus)



Subtraction

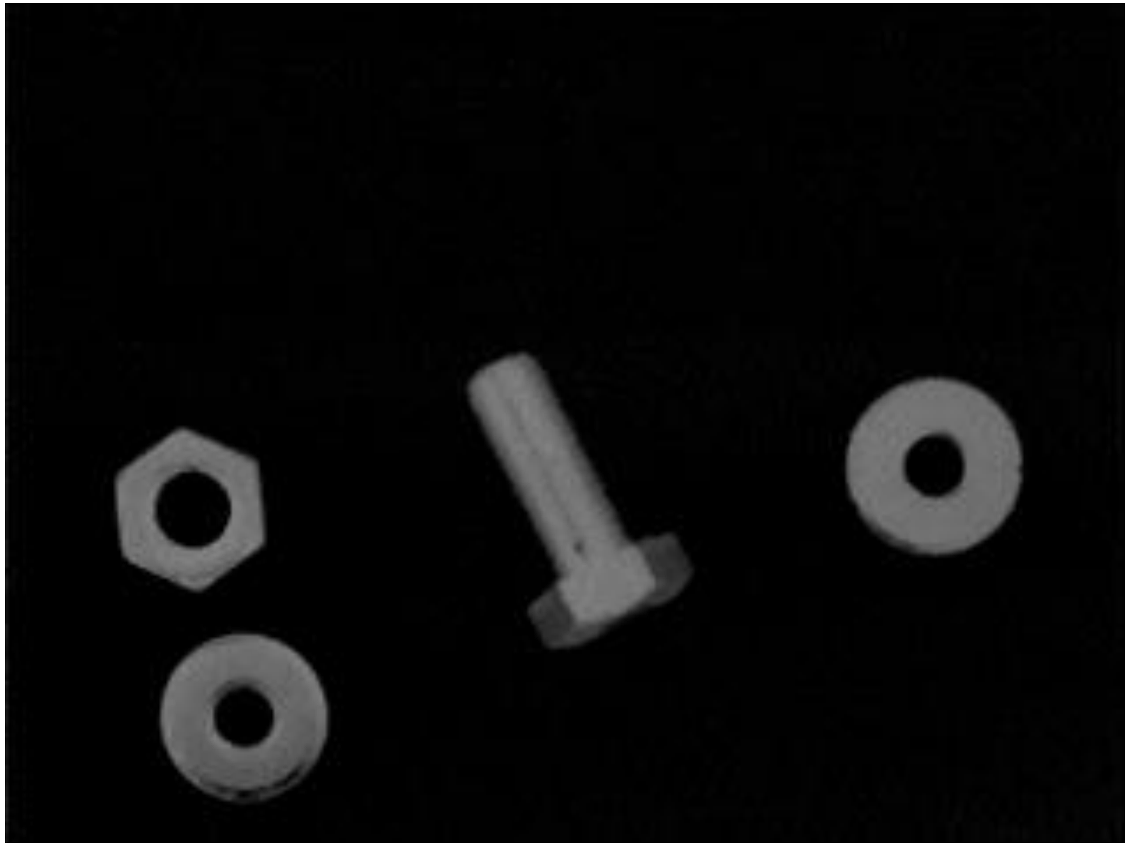


(Divided by)

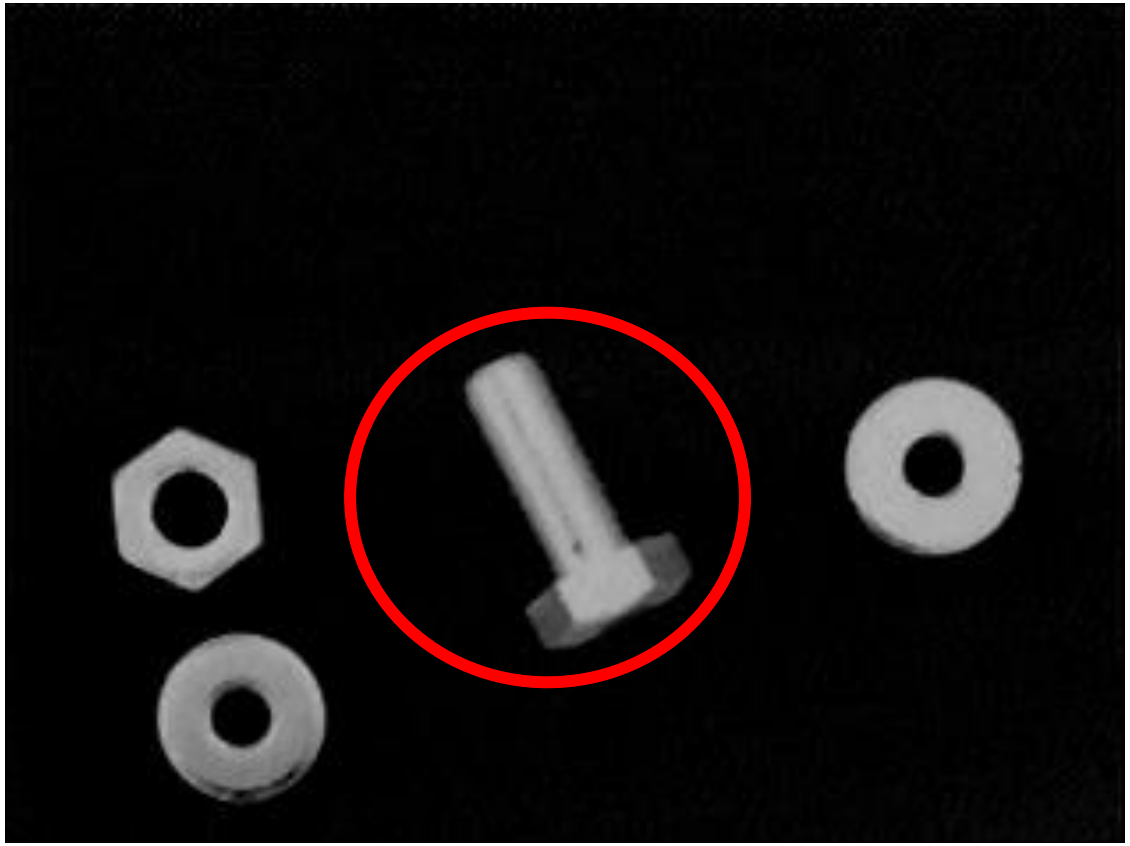


Division

Review (2/3)



Subtraction



Division

Review (3/3)

- ✓ In previous chapter, we have known about …
  - Morphological operations

➢ Remove noise by dilate and erode

➢ Extract boundary

Connected component labeling (1/22)

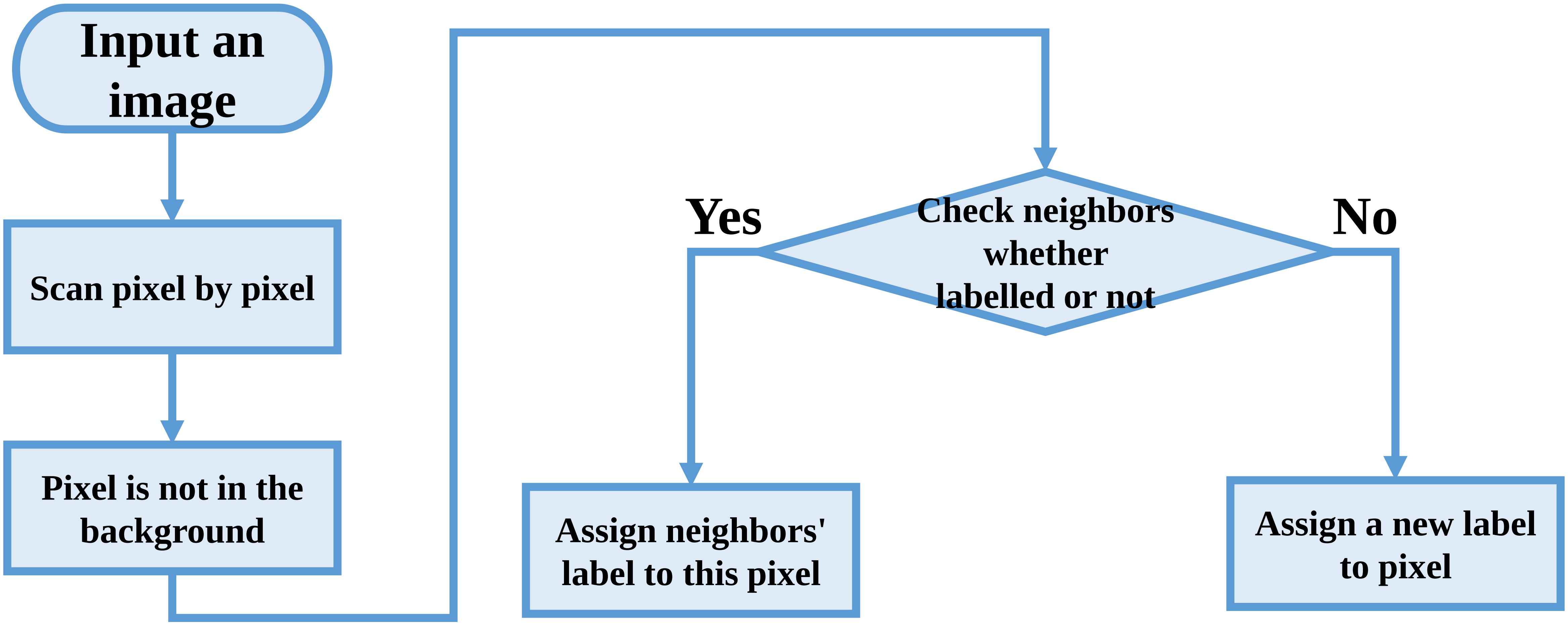
✓ In this episode, we are going to talk about …

➢ **Connected component labeling**

Extracting connected components from an image is crucial to many automated computer vision applications.

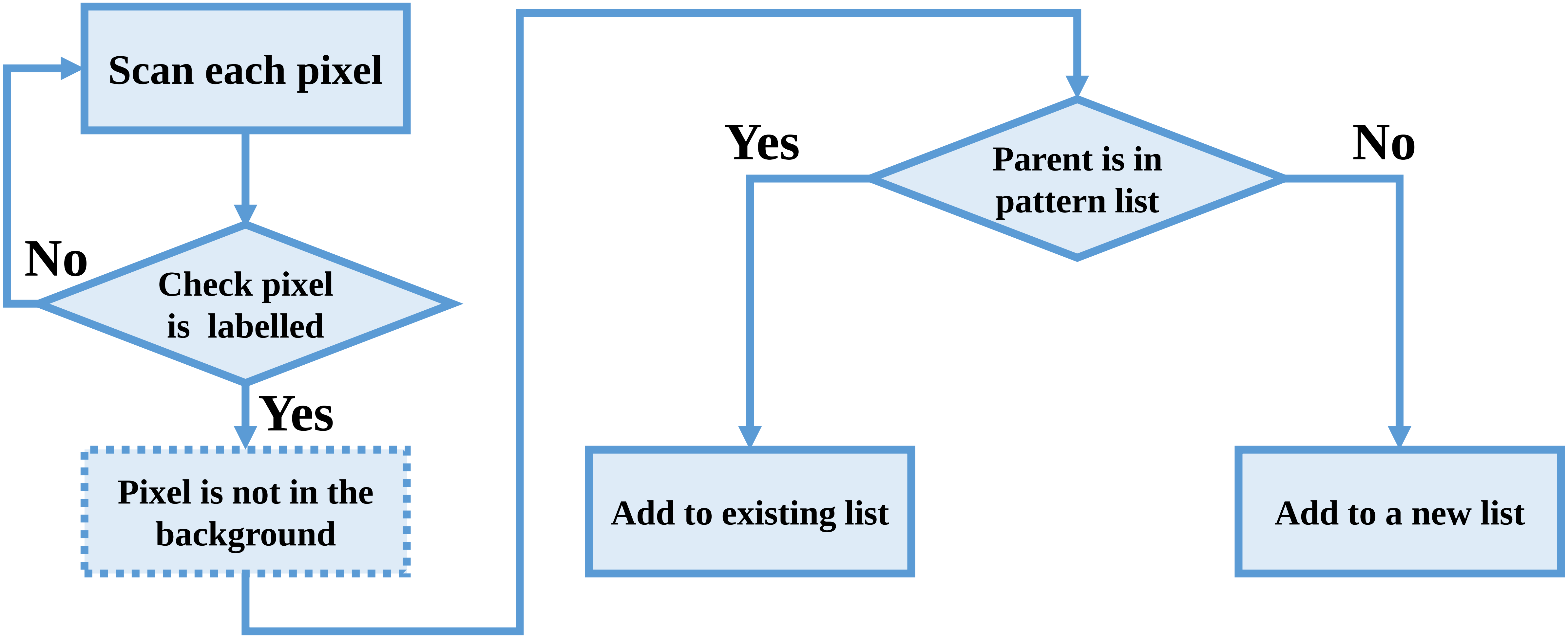
(Extracted)

Connected component labeling (2/22)



- The first pass (assigning labels)

Connected component labeling (3/22)



- The second pass (aggregation)

Connected component labeling (4/22)

	1	
1	1	1
	1	

4-neighbor

1	1	1
1	1	1
1	1	1

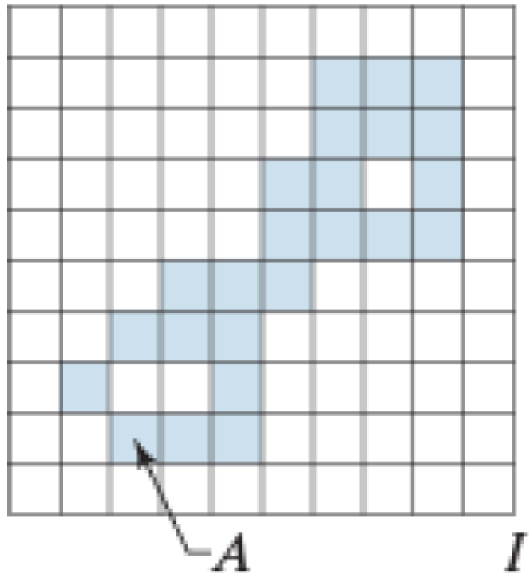
8-neighbor

- ✓ Here, we can use two different tracing methods.
- ✓ You can check its 4-or 8-neighbors, depending on your application.
- ✓ In the following, we use 8-neighbors to search the example image.

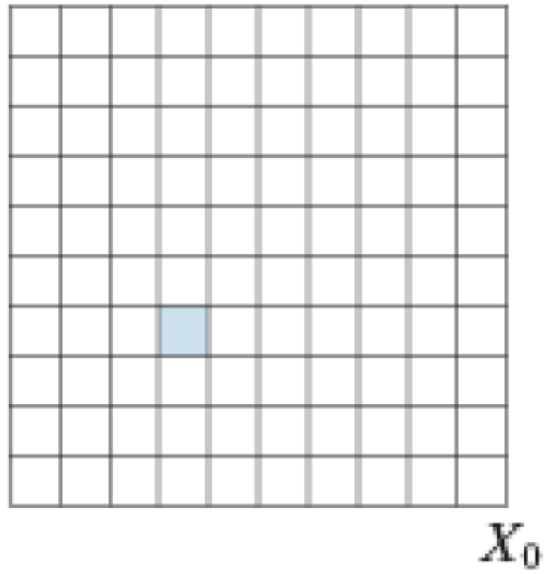
Connected component labeling (5/22)

➢ Recursive approach

- Let A be a set of foreground pixels consisting of one or more connected components.



- Form an image X_0 (of the same size as I) whose elements are 0's, except for a known foreground pixel we set to1.



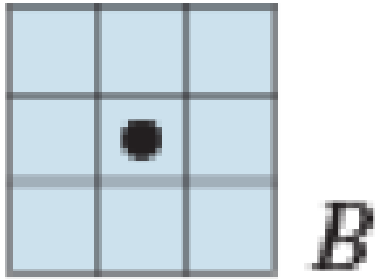
- The objective is to start with X_0 and find all the connected components in I.

Connected component labeling (6/22)

- We can use the following iterative procedure:

$$X_k = (X_{k-1} \oplus B) \cap A$$

- B is the kernel.

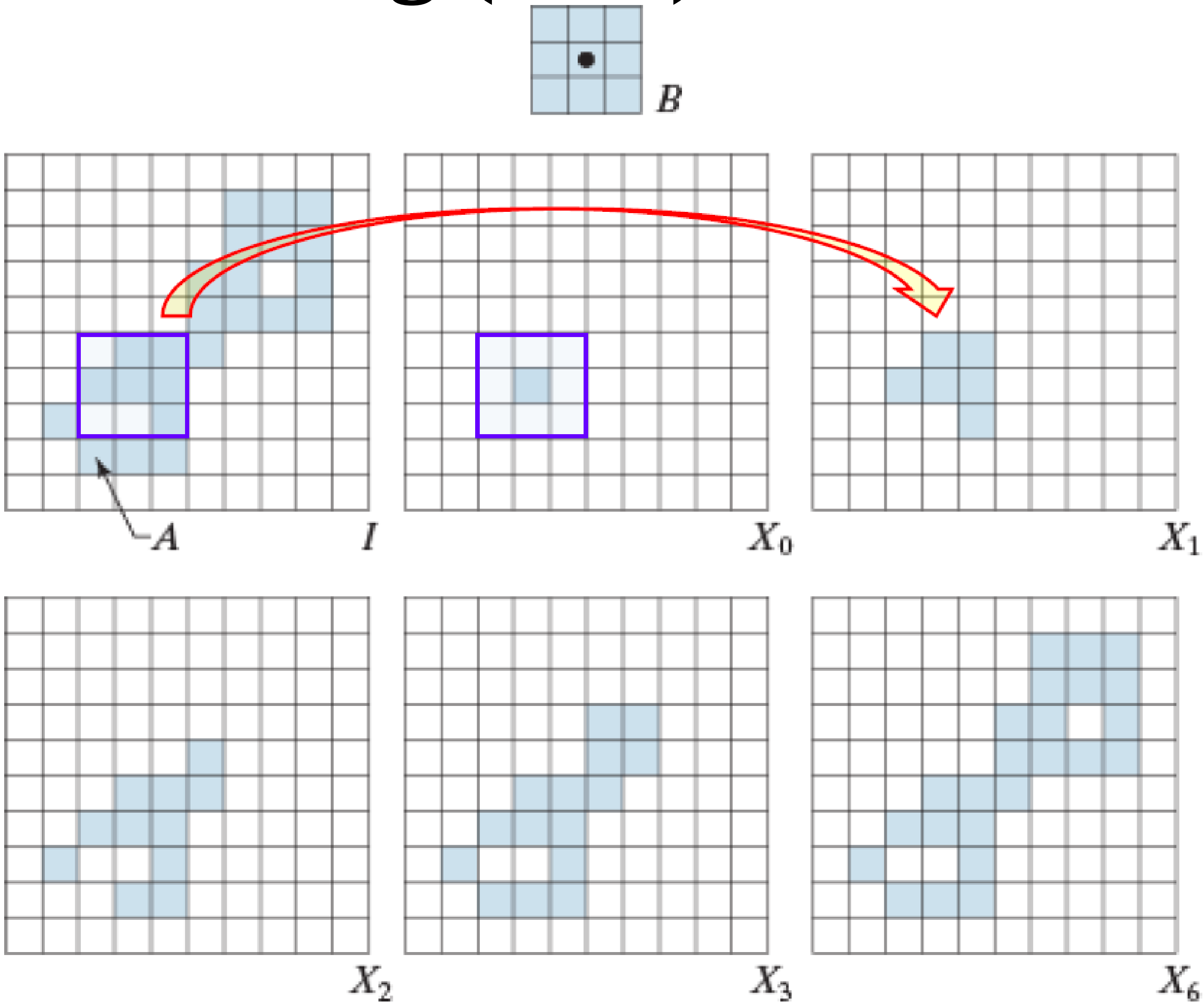


- The procedure terminates when $X_k = X_{k-1}$.

Connected component labeling (7/22)

$$X_k = (X_{k-1} \oplus B) \cap A$$

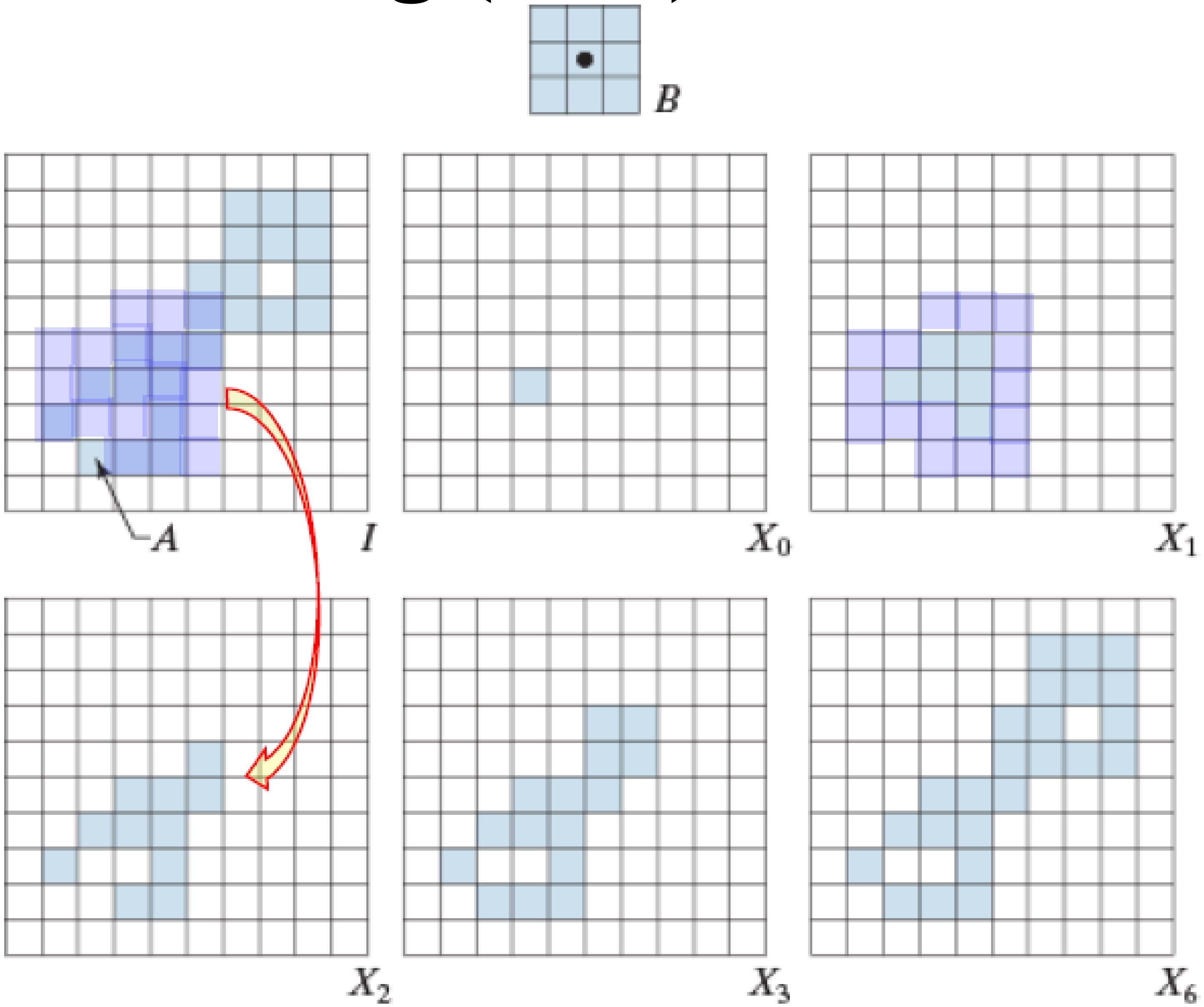
- At the first iteration, dilate X_0 with the kernel B, and then compute its intersection with A to get X_1.



Connected component labeling (8/22)

$$X_k = (X_{k-1} \oplus B) \cap A$$

- At the second iteration, dilate X_1 with the kernel B, and then compute its intersection with A to get X_2.

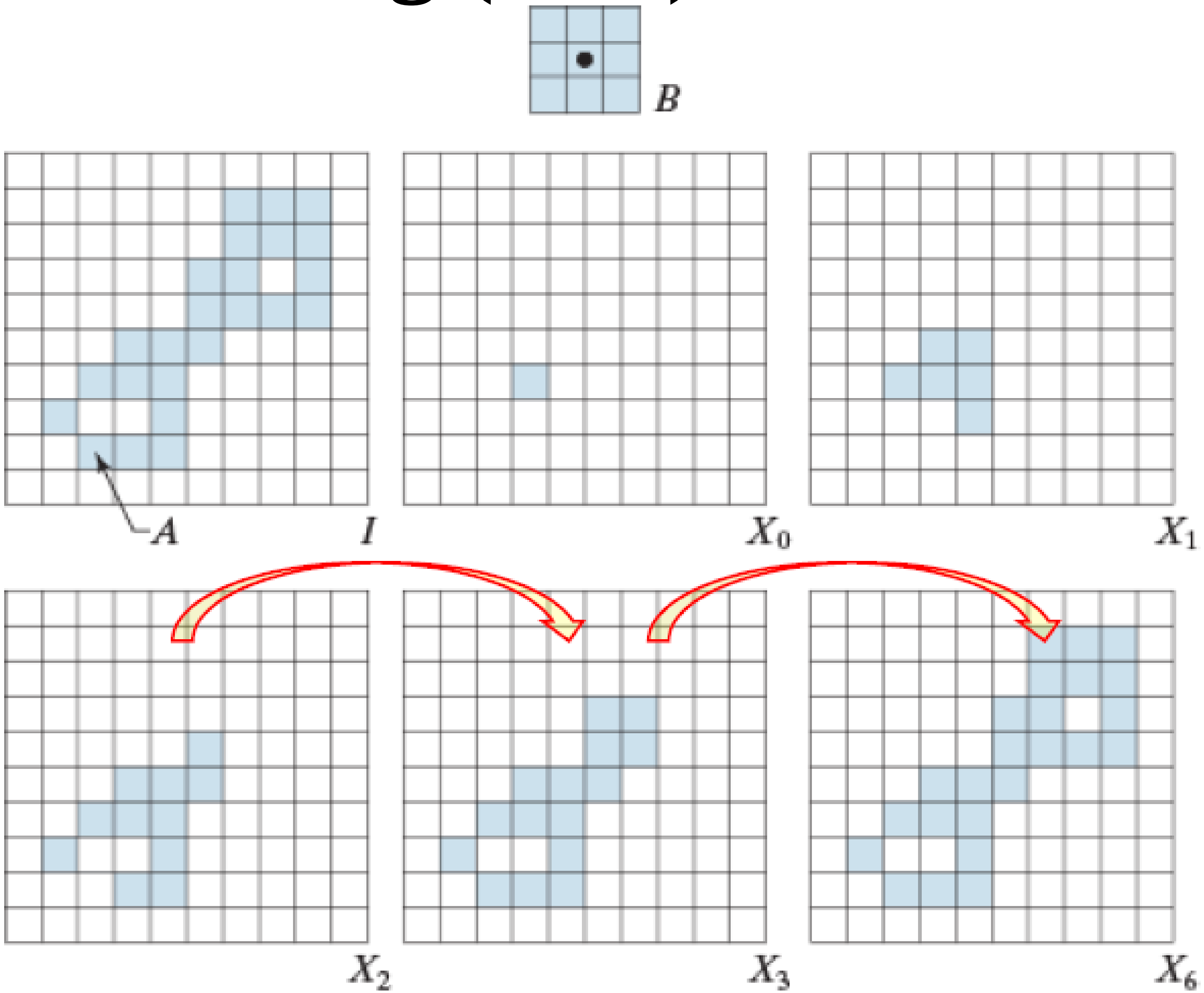


Connected component labeling (9/22)

$$X_k = (X_{k-1} \oplus B) \cap A$$

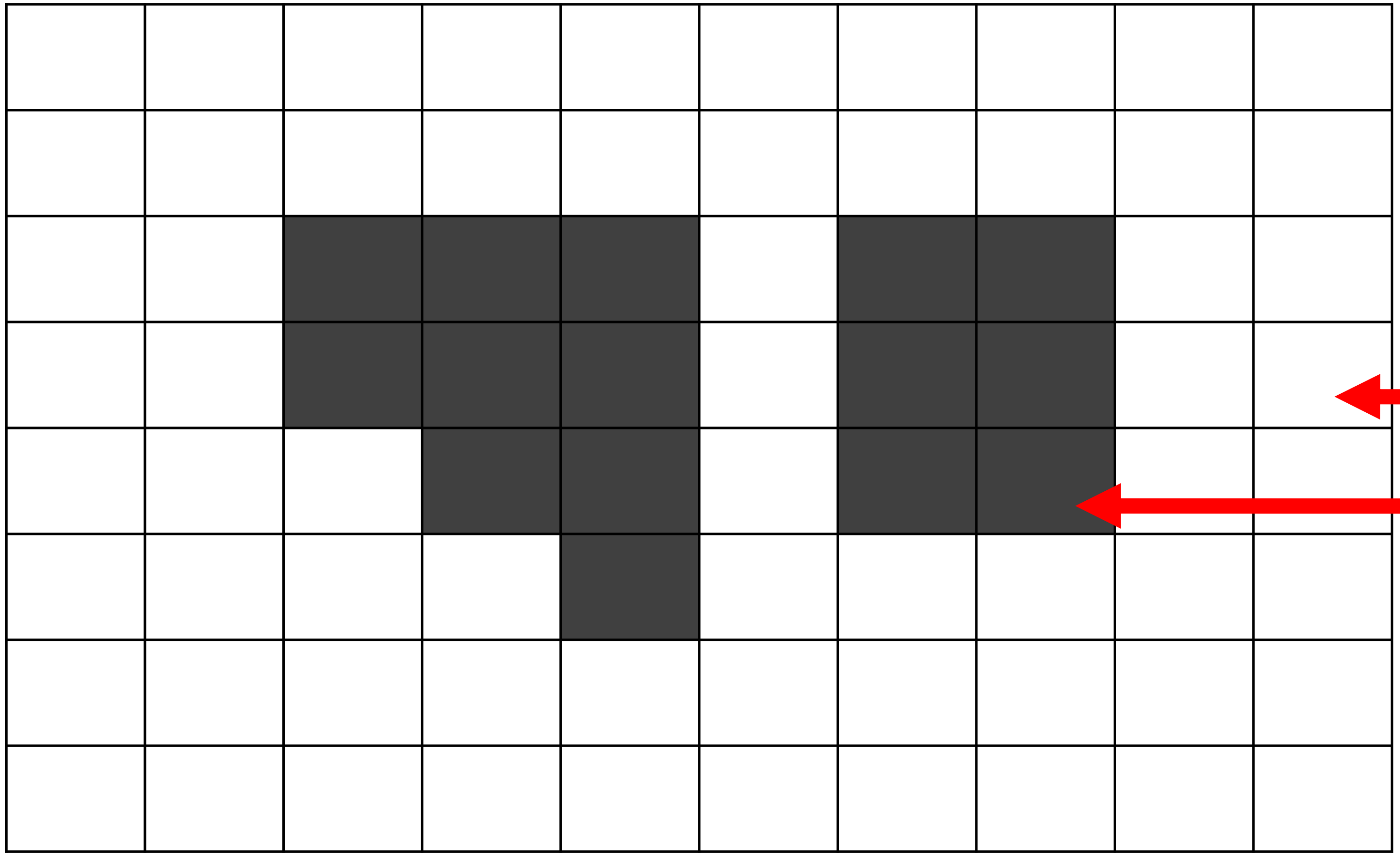
- In this method, the connected component labeling can be done by dilation and intersection.

- Continue the procedure until $X_k = X_{k-1}$.



Connected component labeling (10/22)

- **Sequential approach**



Binary image

0: Background

1: Foreground

- There are two connected components in this image.

Connected component labeling (11/22)

	1	
1	**1**	**1**
	1	

4-neighbor

1	**1**	**1**
1	**1**	**1**
1	**1**	**1**

8-neighbor

- ✓ Here, we can use two different tracing methods.
- ✓ You can check its 4-or 8-neighbors, depending on your application.
- ✓ In the following, we use 4-neighbors to search the example image.

Connected component labeling (12/22)

-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	-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
-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	-1	-1	-1	-1	-1

Labeling

-1: Unlabeled
0 : Background
1 : Object #1
2 : Object #2
⋮

This mask is used to record the information of each traced pixel.

Form a mask (of the same size as the image) and initialize all its pixel values to -1.

Connected component labeling (13/22)

Trace →

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	-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	-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	-1	-1	-1	-1	-1	-1	-1	-1

Labeling

-1: Unlabeled
0 : Background
1 : Object #1
2 : Object #2
⋮

	1	
1	1	1
	1	

4-neighbor

In this section, due to all the pixels are background, so we label the corresponding points in the mask to 0.

Connected component labeling (14/22)

Trace →

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	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	-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	-1	-1	-1	-1	-1	-1	-1	-1

Labeling

-1: Unlabeled
0 : Background
1 : Object #1
2 : Object #2
⋮

Label the corresponding point in the mask to 1, meaning this pixel belongs to Object #1.

Connected component labeling (15/22)

Trace →

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	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	-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	-1	-1	-1	-1	-1	-1	-1	-1

Labeling

-1: Unlabeled
0 : Background
1 : Object #1
2 : Object #2
⋮

	1	
1	1	1
	1	

4-neighbor

Connected component labeling (16/22)

Trace →

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	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	-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	-1	-1	-1	-1	-1	-1	-1	-1

Labeling

-1: Unlabeled
0 : Background
1 : Object #1
2 : Object #2
⋮

	1	
1	1	1
	1	

4-neighbor

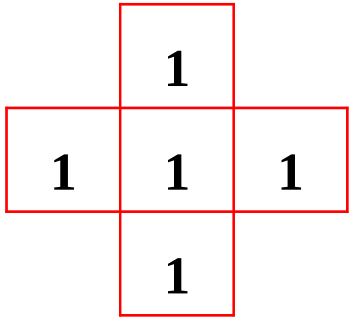
Connected component labeling (17/22)

Trace →

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	-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	-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	-1	-1	-1	-1

Labeling

-1: Unlabeled
0 : Background
1 : Object #1
2 : Object #2
⋮



4-neighbor

Connected component labeling (18/22)

Trace →

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	-1	-1	-1	-1
-1	-1	-1	-1	1	0	-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	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Labeling

-1: Unlabeled
0 : Background
1 : Object #1
2 : Object #2
⋮

	1	
1	1	1
	1	

4-neighbor

Connected component labeling (19/22)

Trace →

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	-1	-1	-1	-1
-1	0	1	1	1	0	-1	-1	-1	-1
-1	-1	0	1	1	0	-1	-1	-1	-1
-1	-1	-1	0	1	0	-1	-1	-1	-1
-1	-1	-1	-1	0	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Labeling

-1: Unlabeled
0 : Background
1 : Object #1
2 : Object #2
⋮

	1	
1	1	1
	1	

4-neighbor

Connected component labeling (20/22)

Trace

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	$\boxed{2}$	-1	-1	-1
-1	0	1	1	1	0	-1	-1	-1	-1
-1	-1	0	1	1	0	-1	-1	-1	-1
-1	-1	-1	0	1	0	-1	-1	-1	-1
-1	-1	-1	-1	0	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Labeling

-1: Unlabeled
0 : Background
1 : Object #1
2 : Object #2
⋮

We meet a foreground pixel here, and label the corresponding point in the mask to the next number, i.e., 2, meaning this pixel belongs to Object #2.

Connected component labeling (21/22)

Labeling

-1: Unlabeled
0 : Background
1 : Object #1
2 : Object #2
⋮

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

Trace →

After tracing all pixels in the image, all connected components are labelled.

Connected component labeling (22/22)

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

Trace →

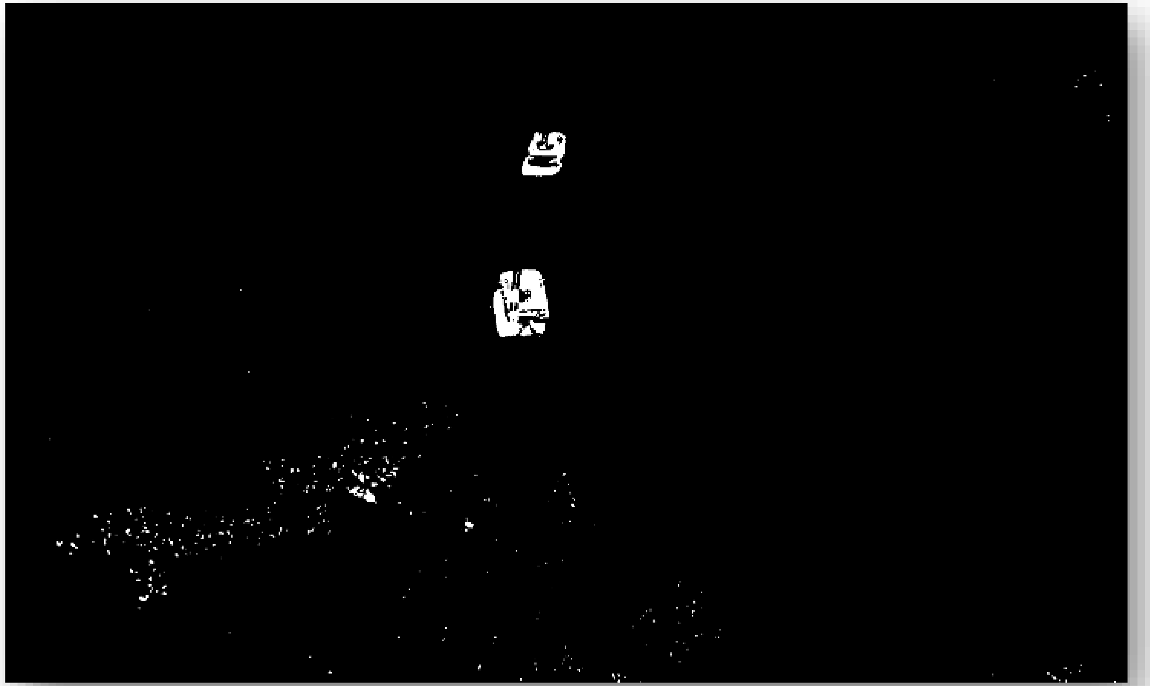
Labeling

-1: Unlabeled
0 : Background
1 : Object #1
2 : Object #2
⋮

When performing connected component labelling, we can also record the max *x*, max *y*, min *x*, min *y* for each object to get its bounding box (or bounding rectangle).

Example of Connected component labeling (1/3)

Original Video Frame

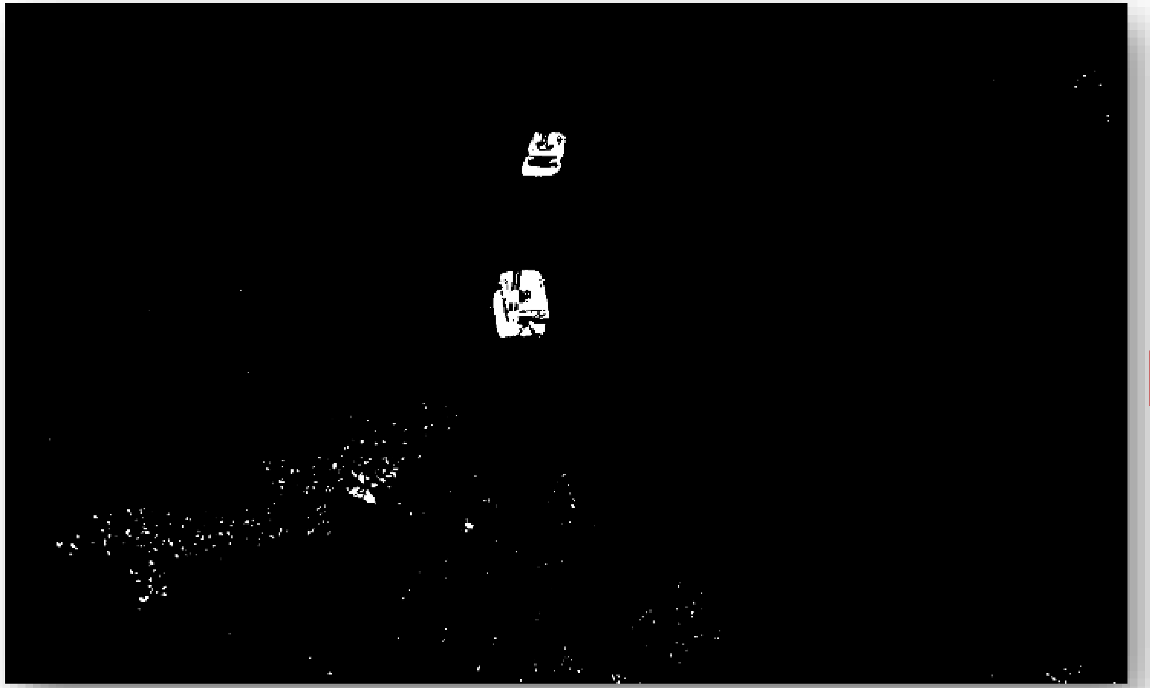


Foreground Pixels

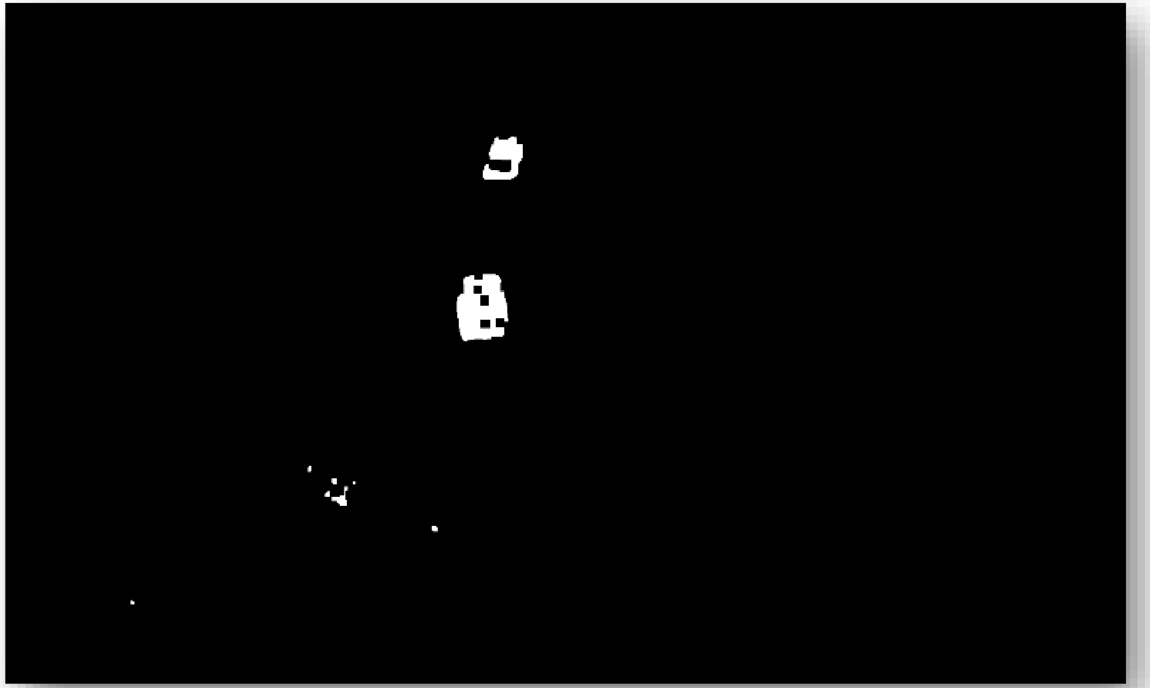
✓ **Background subtraction (in previous episode)**

Source: Connected component labeling - Chen Hua-Tsung

Example of Connected component labeling (2/3)



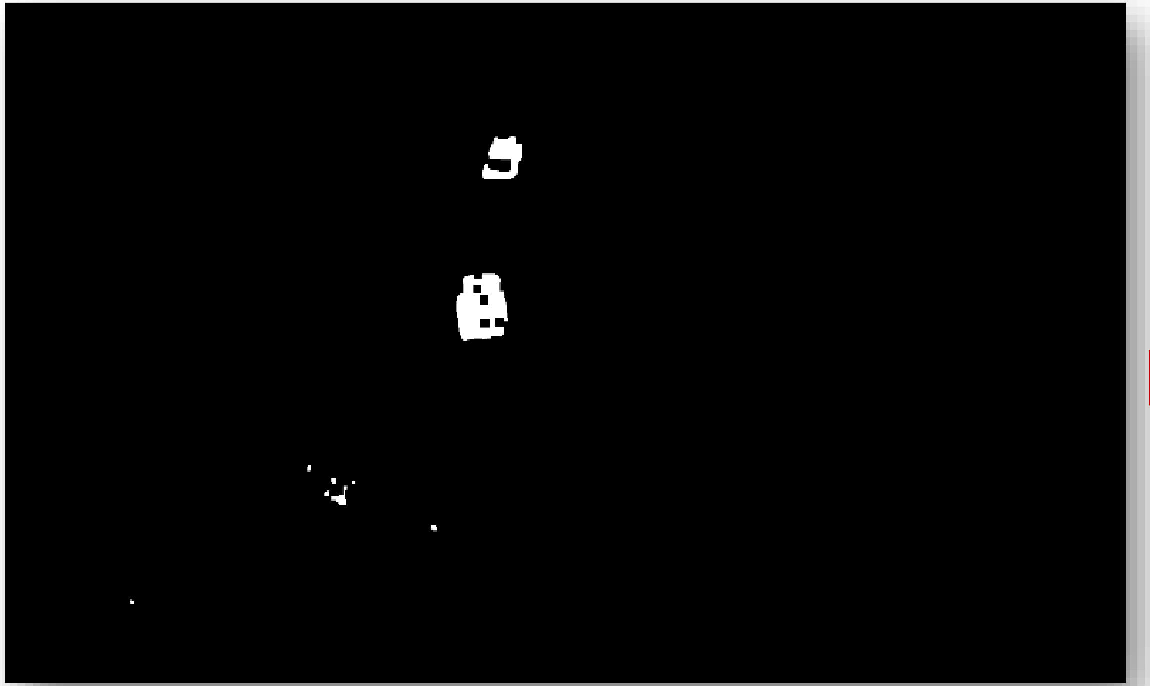
Foreground Pixels



Remove Noise

✓ **Morphological operations**

Example of Connected component labeling (3/3)



Remove Noise

Object marked

✓ **Connected component labeling**

Source: Connected component labeling - Chen Hua-Tsung

OpenCV – connectedComponents(1/4)

➢ Code

Syntax:

connectedComponents(src, labels, connectivity, type);

src – Input 8-bit single-channel (Gray Level).
labels – Output label map.
connectivity – 4- or 8-connected components
type– Output label type (CV_32S or CV_16U)

OpenCV – connectedComponents(2/4)

➢ Demo Code

```
Mat img;
int threshval = 100;  // Bar Default Value.

int main(int argc, const char** argv)
{

    img = imread(“D:/Data.jpg", IMREAD_GRAYSCALE);

    imshow("Image", img);

    namedWindow("Connected Components", WINDOW_AUTOSIZE);
    createTrackbar("Threshold", "Connected Components", &threshval, 255, on_trackbar);
    on_trackbar(threshval, 0);

    waitKey(0);
    return 0;
}
```

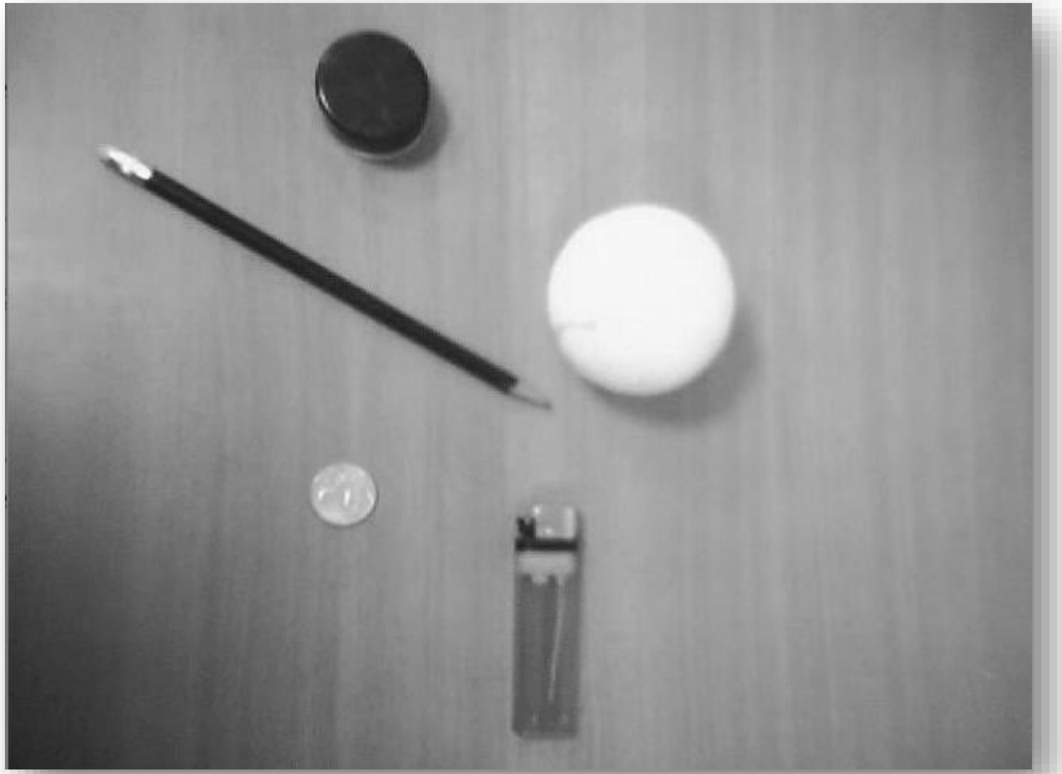
OpenCV – connectedComponents(3/4)

➢ Demo Code

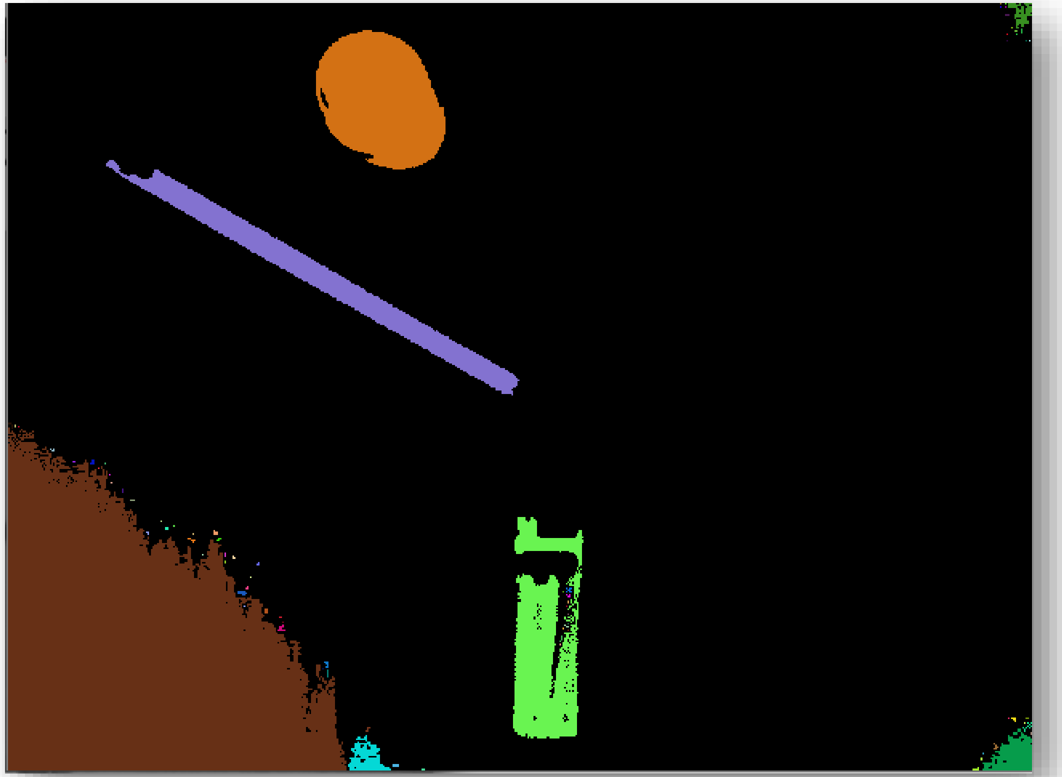
```
static void on_trackbar(int, void*)
{
    Mat bw = threshval < 128 ? (img < threshval) : (img > threshval);
    Mat labelImage(img.size(), CV_32S);

    int nLabels = connectedComponents(bw, labelImage, 8);

    vector<Vec3b> colors(nLabels);
    colors[0] = Vec3b(0, 0, 0);  //Background
    for (int label = 1; label < nLabels; ++label)
    {
        colors[label] = Vec3b((rand() & 255), (rand() & 255), (rand() & 255));
    }
    // Draw color to the whole image.
    Mat dst(img.size(), CV_8UC3);
    for (int r = 0; r < dst.rows; ++r) {
        for (int c = 0; c < dst.cols; ++c) {
                int label = labelImage.at<int>(r, c);
                Vec3b &pixel = dst.at<Vec3b>(r, c);
                pixel = colors[label];
            }
        }
    imshow("Connected Components", dst);
}
```

OpenCV – connectedComponents(4/4)

➢ Demo Result



Original Image



Connected Component Results

Practice

OpenCV – findContours (1/4)

➢ Code

Syntax:

findContours(src, contours, hierarchy, mode, method, offset);

src – Source, an 8-bit single-channel image.
contours – Detected contours. Each contour is stored as a vector of points.
hierarchy – Optional output vector, containing information about the image topology. It has as many elements as the number of contours.
mode – Contour retrieval mode
method – Contour approximation method.
offset – Optional offset by which every contour point is shifted.

OpenCV – findContours (2/4)

White foreground regions are on a black background.

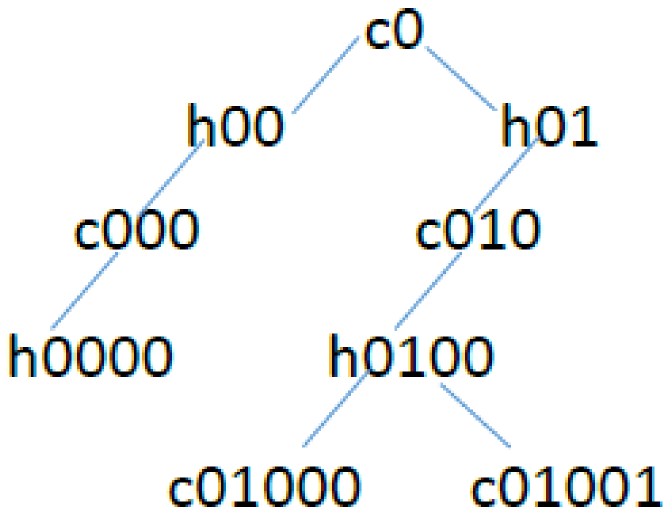
c: contour

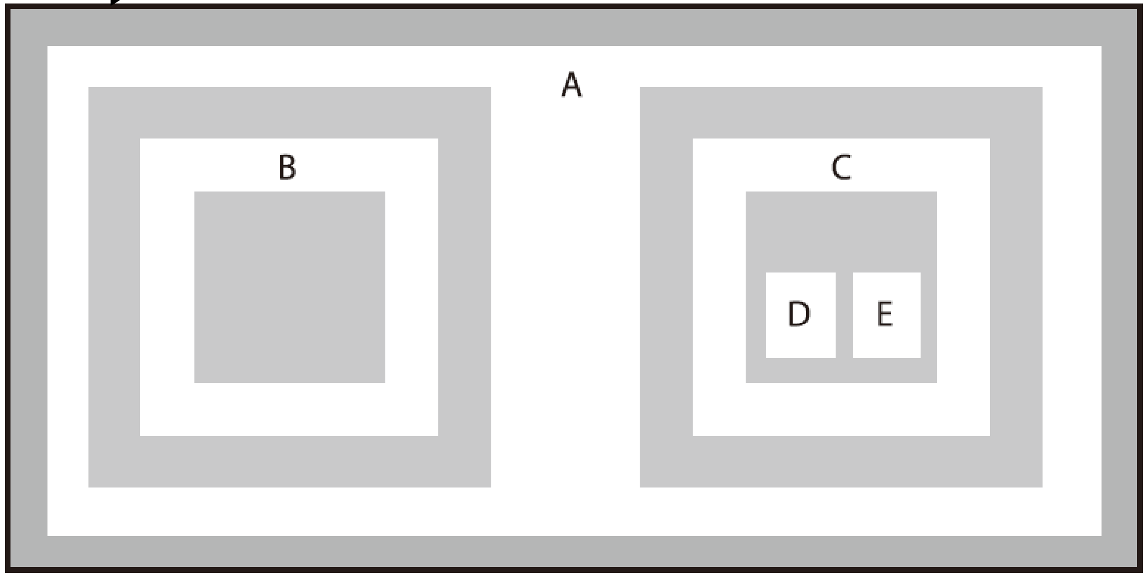
h: hole

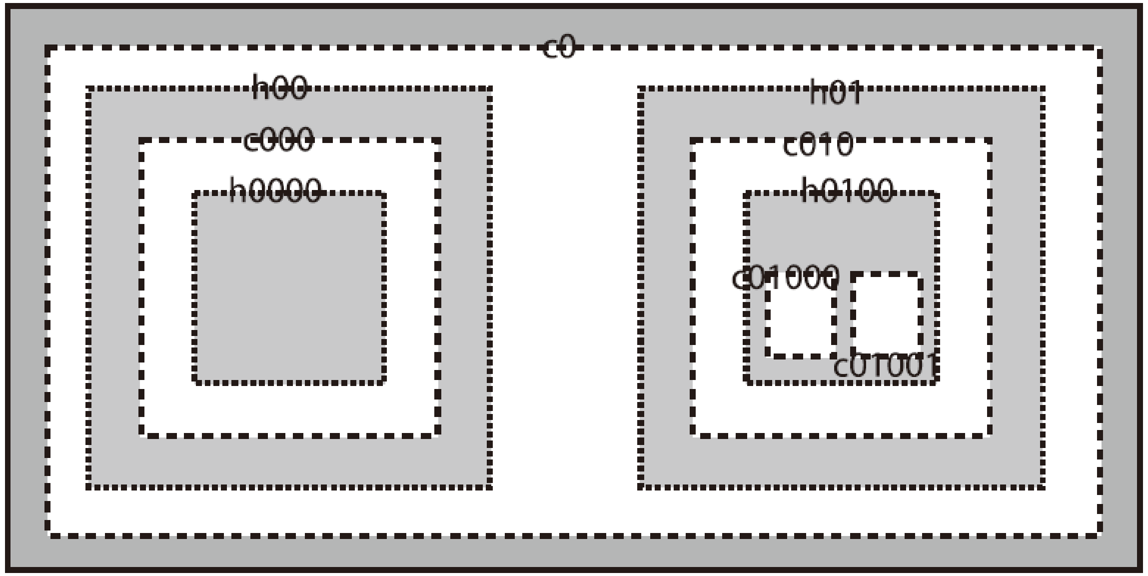
Contours (dashed lines): exterior boundaries

Contours (dotted lines): interior boundaries

The contour hierarchy can be represented as a tree:







Source: Connected component labeling - Chen Hua-Tsung

OpenCV – findContours (3/4)

- **Contour mode**

  Syntax:

  findContours(src, contours, hierarchy, **mode**, method, offset)

  - **mode** – Contour retrieval mode.

    CV_RETR_EXTERNAL: Retrieves only the extreme outer contours.

    CV_RETR_LIST：Retrieves all of the contours without establishing any hierarchical relationships.

    CV_RETR_CCOMP：Retrieves all of the contours and organizes them into a two-level hierarchy.
    At the top level, there are external boundaries of the components.
    At the second level, there are boundaries of the holes.
    If there is another contour inside a hole of a connected component, it is still put at the top level.

    CV_RETR_TREE：Retrieves all of the contours and reconstructs a full hierarchy of nested contours.

OpenCV – findContours (4/4)

➢ Contour method

Syntax:

➢ findContours(src, contours, hierarchy, mode, **method**, offset)

method – Contour approximation method.

CV_CHAIN_APPROX_NONE: Stores absolutely all the contour points.

- That is, any 2 subsequent points (*x1*, *y1*) and (*x2*, *y2*) of the contour will be either horizontal, vertical or diagonal neighbors, that is, max(*abs(x1-x2),abs(y2-y1)*)==1.

CV_CHAIN_APPROX_SIMPLE: Compresses horizontal, vertical, and diagonal segments and leaves only their end points. For example, an up-right rectangular contour is encoded with 4 points.

CV_CHAIN_APPROX_TC89_L1,CV_CHAIN_APPROX_TC89_KCOS: Applies one of the flavors of the Teh-Chin chain approximation algorithm. See the OpenCV doc.[TehChin89] for more the details.

OpenCV – drawContours (1/2)

➢ Code

Syntax:

```
drawContours(src, contours, index, color, thickness, lineType, hierarchy, maxLevel, offset);
```

src – Destination image where you would like to draw contours.

contours – All the input contours (obtained from findContours).
- Each contour is stored as a point vector.

index – Parameter indicating a contour to draw. If it is negative, all the input contours are drawn.

color – Color of the contours.

thickness – Thickness of lines the contours are drawn with. If it is negative (e.g., thickness = CV_FILLED), the contour interiors are drawn.

lineType – Line connectivity. (**8**: 8-connected, **4**: 4-connected, **CV_AA**: anti-aliased line)

OpenCV – drawContours (1/2)

➢ Code

Syntax:

EX: drawContours(drawing, contours, (int)i, color, 2, LINE_8, hierarchy, 0);

drawContours(src, contours, index, color, thickness, lineType, hierarchy, maxLevel, offset);

hierarchy – Optional information about hierarchy.
It is only needed if you want to draw only some of the contours (see maxLevel).

maxLevel – Maximal level for drawn contours.
- If it is 0, only the contour specified by contour index is drawn.
- If it is 1, the function draws the contour(s) and all the nested contours. (E.g., contours 2 and 3 are drawn.)
- If it is 2, the function draws the contours, all the nested contours, all the nested-to-nested contours, and so on. (E.g., contours 2, 3, 4 are drawn.)
- This parameter is only taken into account when there is hierarchy available

OpenCV – Canny Edge Detector

➢ Code

Syntax:

```
Canny(src, edges, threshold1, threshold2, apertureSize, L2gradient=false );
```

src – Source, an 8-bit single-channel image.
edges – output edge map; it has the same size and type as image .
threshold1 – First threshold for the hysteresis procedure.
threshold2 – Second threshold for the hysteresis procedure.
apertureSize – Aperture size for the Sobel() operator.

L2gradient – A flag, indicating whether a more accurate L_2 norm $= \sqrt{(dI/dx)^2+(dI/dy)^2}$ should be used to calculate the image gradient magnitude (L2gradient=true), or whether L_1 the default norm $=|dI/dx|+|dI/dy|$ is enough (L2gradient=false).

Give it a try (1/3)

➢ Demo Code

```
int main(int argc, char** argv)
{
Mat src = imread("D:/Data.jpg"); // Load source image

cvtColor(src, src_gray, COLOR_BGR2GRAY);
blur(src_gray, src_gray, Size(3, 3)); // Convert image to gray and blur it to get  clear contour.

const char* source_window = "Source";
namedWindow(source_window);
imshow(source_window, src);

const int max_thresh = 255;
createTrackbar("Canny Thresh:", source_window, &thresh, max_thresh, thresh_callback);
thresh_callback(0, 0);

waitKey();
return 0;
}
```

Give it a try (2/3)

- ➢ **Demo Code**

```
void thresh_callback(int, void*)
{
Mat canny_output; // Detect edges using Canny
Canny(src_gray, canny_output, thresh, thresh*2); // Canny Edge Detector to reinforce the contour result.

vector<vector<Point> > contours; // Find contours
vector<Vec4i> hierarchy;
findContours(canny_output, contours, hierarchy, RETR_TREE,
                  CHAIN_APPROX_SIMPLE);

Mat drawing = Mat::zeros(canny_output.size(), CV_8UC3); // Declare background to zero.
for (size_t i = 0; i < contours.size(); i++) // Draw contours for loop
{
    Scalar color = Scalar(255, 255, 255);
    drawContours(drawing, contours, (int)i, color, 2, LINE_8, hierarchy, 0);
}

imshow("Contours", drawing);
}
```

Give it a try (3/3)

➢ Demo Result

Original Image

Result of Contours

OpenCV – boundingRect

- **Code**

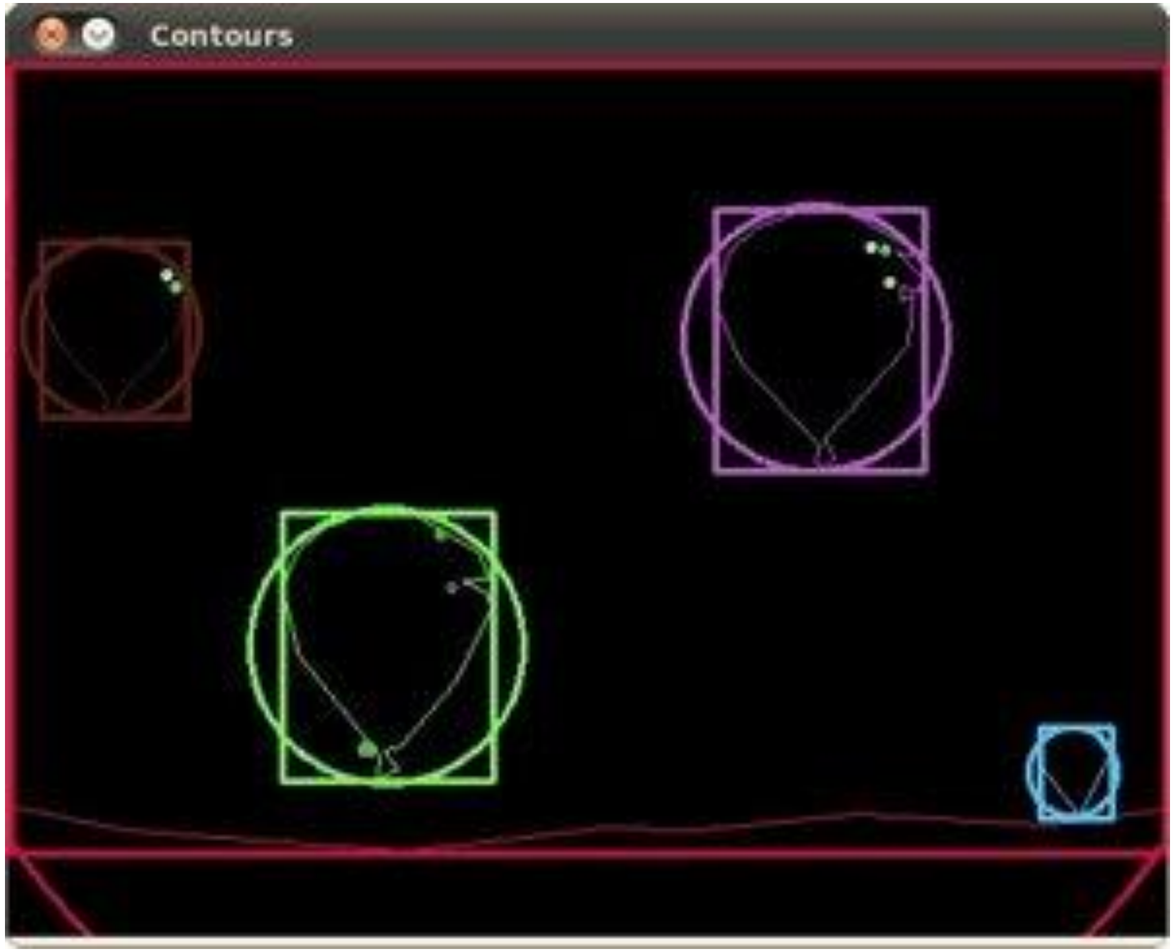
Syntax:

Rect boundingRect(InputArray **points**);

points – Input 2D point set, stored in std::vector or Mat.

- ✓ The function calculates and returns the minimal up-right bounding rectangle for the specified point set.

Give it a try



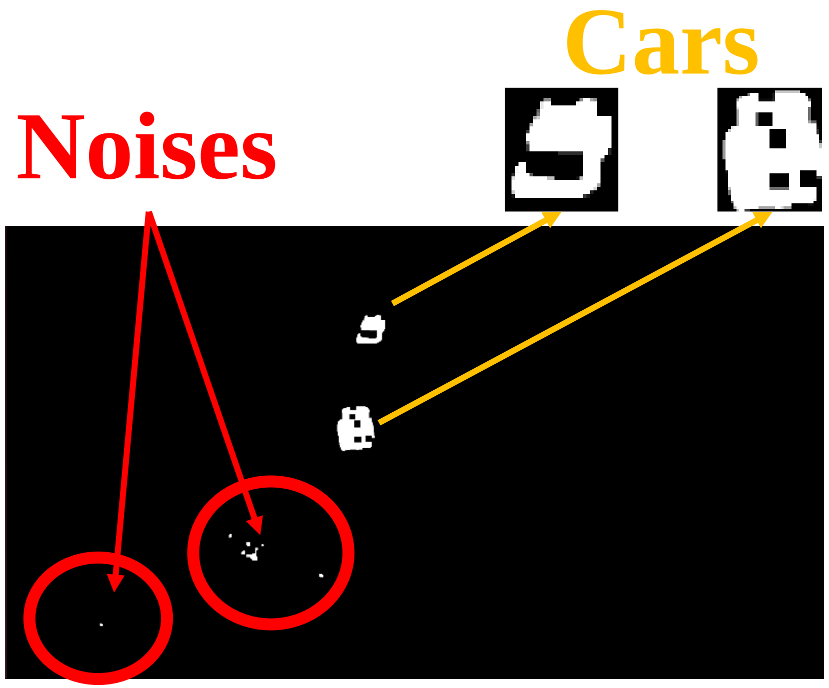
✓ You can try this on ➢ https://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/bounding_rects_circles/bounding_rects_circles.html

Demo

Practice

Basic object filtering (1/5)

- After morphological operations, there may be still some noises which do not belong to foreground objects.
- Object filtering is required to eliminate such noises.
  - ✓ Object size
  - ✓ Shape
  - ✓ Compactness

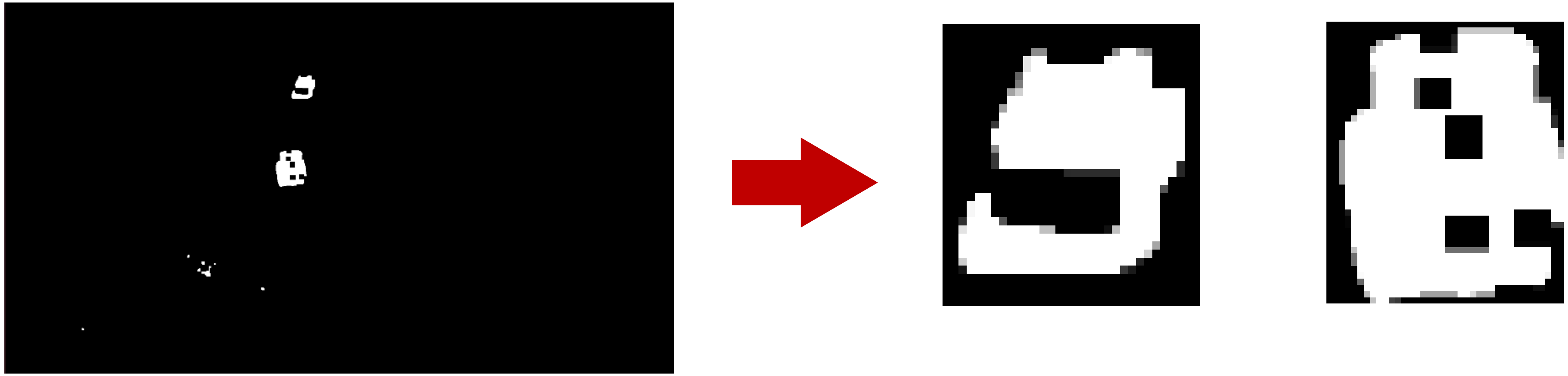


Source: Connected component labeling - Chen Hua-Tsung

Basic object filtering (2/5)

➢ Object size

- ✓ Area of the bounding box: width × height
- ✓ Number of foreground pixels



- Filter out the objects which are too small or too large.

Basic object filtering (3/5)

- **Shape**
  - ✓ Aspect ratio of the bounding box.
    → R= width / height.

Stand

R=0.3

Squat

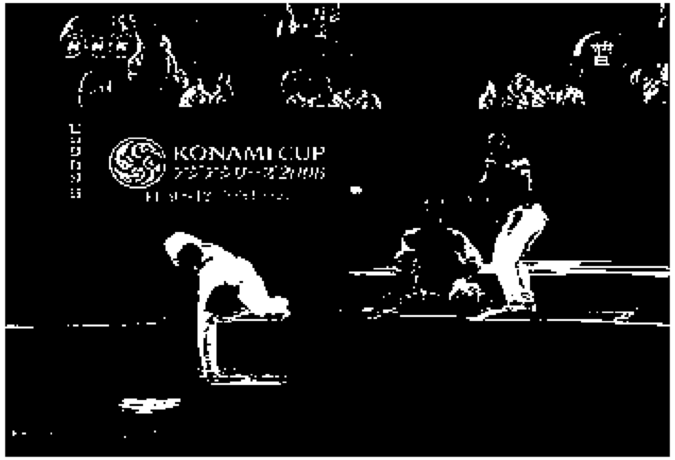
R=0.7

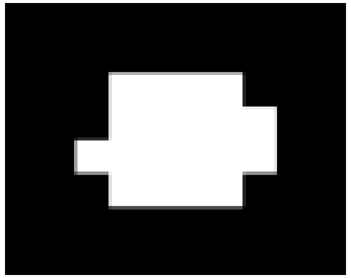
Lie down

R=5.5

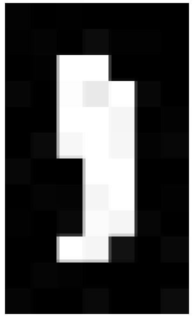
Basic object filtering (4/5)

- **Shape**
  - ✓ Ball detection (Aspect ratio = 1)





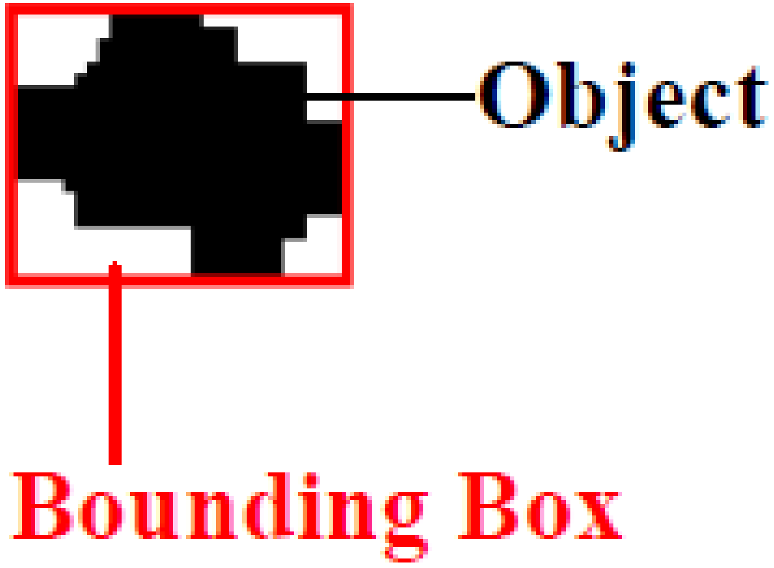
○



×

Basic object filtering (5/5)

- **Compactness**
  - ✓ Object size / Bounding box area



Thanks!

Any questions?