Edge and Region

Computer Vision and Pattern Recognition

Byeongjoon Noh

powernoh@sch.ac.kr

SCH SOON CHUN HYANG UNIVERSITY

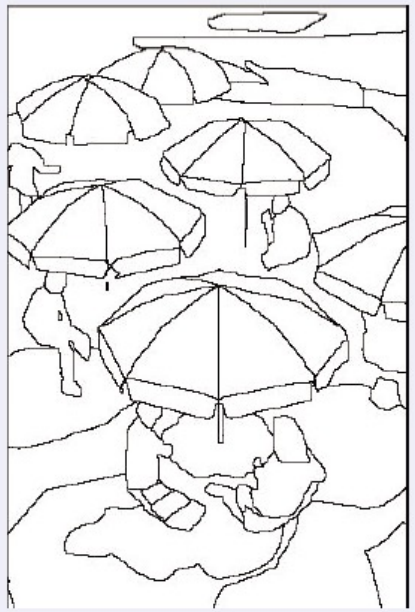
Most of the slides are available on Senseable AI Lab homepage: https://sailab.space/courses/

Contents

1. Edge detection

Getting Started

- 엣지(Edge)와 영역(Region) 분할은 컴퓨터 비전 초창기부터 중요한 연구 주제
  - 컴퓨터 비전 알고리즘이 사람 수준으로 분할할 수 있을까?



(a) 원래 영상 (b) 영역 영상(사람이 분할)

그림 4-1 영역 분할을 위한 BSDS 데이터셋

- https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/

Getting Started

- Edge와 region은 서로 다른 접근 방법 사용
  - Edge ➔ 특성이 다른 곳 검출
  - Region ➔ 유사한 화소를 묶음

- Human ➔ Semantic segmentation (의미 분할)에 능숙
  - 사람은 머리 속에 기억된 물체 모델을 활용하여 의미 분할
  - 본 장에서 학습하는 고전적인 방법으로는 의미 분할 불가능
  - ➔ Deep learning은 의미 분할 가능

Edge detection

- Edge detection algorithm
  - 물체 내부는 명암이 서서히 변하고 경계는 급격히 변하는 특성을 활용

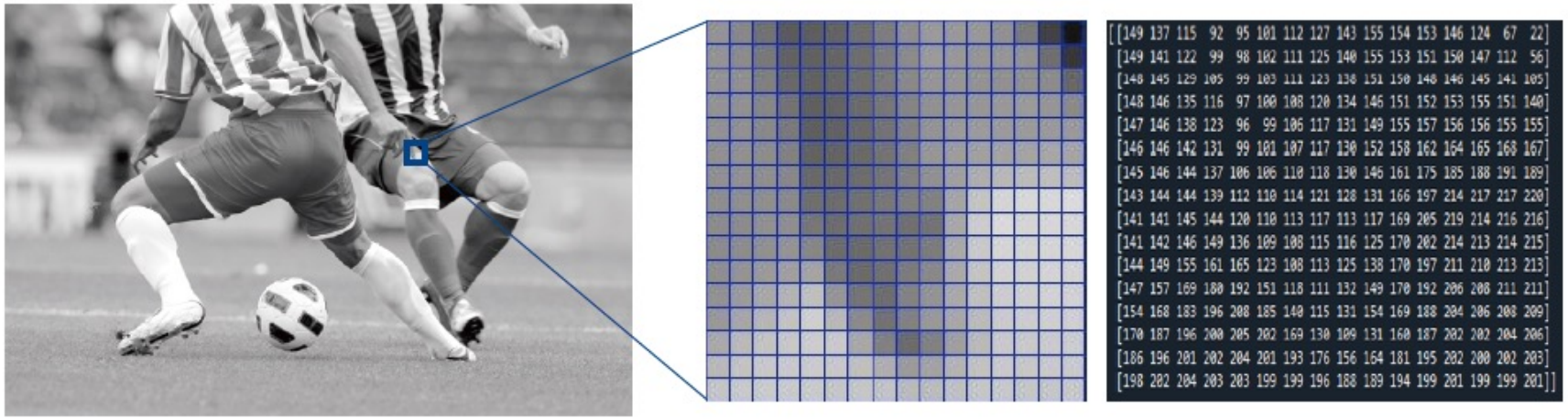


그림 4-2 명암 변화를 확인하기 위해 영상 일부를 확대

  - 어떻게 명암의 변화를 탐지할 수 있을까? ➔ 미분

영상의 미분

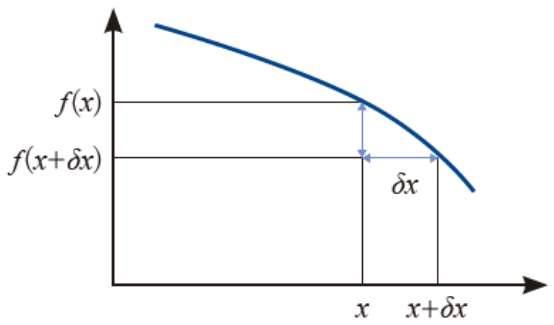
- 미분
  - 변수 x가 미세하게 증가할 때 함수 변화량을 측정하는 수학적 기법
  - 연속 함수에서의 미분

$$f'(x)=\lim_{\delta x\to 0}\frac{f(x+\delta x)-f(x)}{\delta x} \qquad (4.1)$$

  - 디지털 영상(이산)에서의 미분: 주변 화소와의 값 차이

$$f'(x)=\frac{f(x+\delta x)-f(x)}{\delta x}=f(x+1)-f(x) \qquad (4.2)$$

- 실제 구현은 필터 u로 convolution 연산
  - filter u: edge detector



(a) 연속 함수의 미분

(b) 디지털 영상의 미분(필터 u로 컨볼루션)

그림 4-3 연속 함수와 디지털 영상의 미분

Edge detector

- 현실 세계의 ramp edge
  - 명암이 몇 화소에 걸쳐서 변함
  - ➔ 미분을 2번 해야 경계가 드러남

그림 4-4 현실 세계에서 발생하는 램프 에지

두꺼운 엣지로 인해 위치 찾기 문제 발생

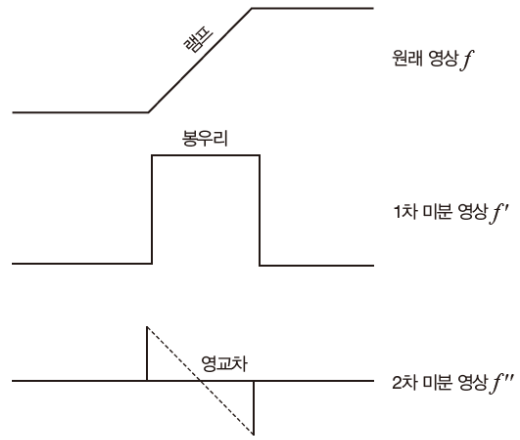
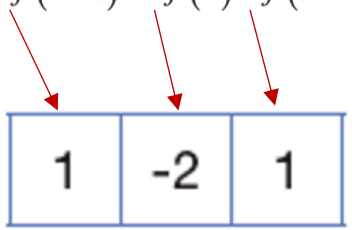


그림 4-5 램프 에지에서 발생하는 봉우리와 영교차

- 2차 미분: 미분을 2번 할 필요 없이 필터를 활용하여 2차 미분과 같은 효과

$$f''(x)=\frac{f'(x)-f'(x-\delta)}{\delta}=f'(x)-f'(x-1)$$
$$=\left(f(x+1)-f(x)\right)-\left(f(x)-f(x-1)\right)$$
$$=f(x+1)-2f(x)+f(x-1) \quad (4.3)$$



엣지 연산자

- 1차 미분 기반의 edge detector (filter) – 영상의 밝기가 급격하게 변하는 지점을 검출
  - Sobel filter (Sobel edge detector)
  - Prewitt filter
  - Canny edge detector
  - ...

- 2차 미분 기반의 엣지 검출 필터 (Marr-Hildreth의 zero-crossing 이론) – 영상의 밝기 변화율의 변화를 검출
  - Laplacian filter
  - Laplacian of Gaussian (LoG) filter
  - ...

1차 미분 기반의 edge detector

- 실제 영상에 있는 잡음을 흡수하기 위해 크기가 2인 필터를 크기 3으로 확장

$$f_x'(y,x) = f(y,x+1) - f(y,x-1)$$
$$f_y'(y,x) = f(y+1,x) - f(y-1,x) \qquad (4.4)$$

이 식을 구현하는 필터는 u_x= | -1 | 0 | 1 | 와 u_y= | -1 | / | 0 | / | 1 |

- 1차원을 2차원으로 확장

u_x=

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

u_y=

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

(a) 프레윗(Prewitt) 연산자

u_x=

-1	0	1
-2	0	2
-1	0	1

u_y=

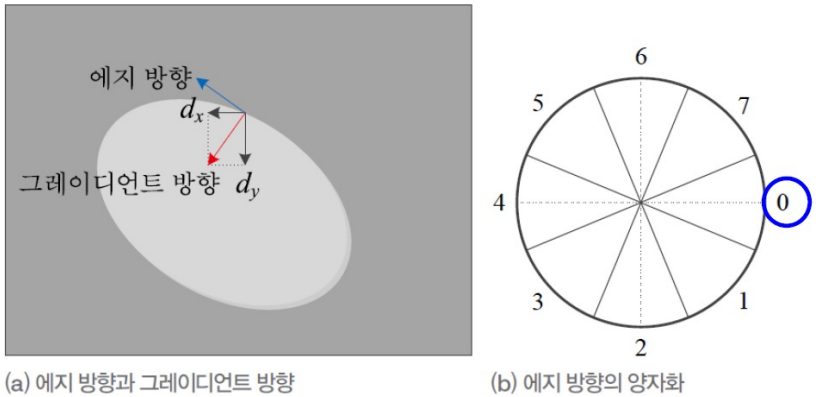
-1	-2	-1
0	0	0
1	2	1

(b) 소벨(Sobel) 연산자

그림 4-6 에지 연산자

Edge magnitude and edge direction

- Edge detection 과정에서 중요한 정보를 제공 (임계값, 특징 추출 등)
- Edge magnitude (edge strength, 강도)
  - 영상의 한 지점에서 밝기 변화의 크기
    - 엣지일 가능성 또는 신뢰도(confidence)로 해석 가능
  - 주로 gradient의 크기로 계산됨
  - gradient
    - $\nabla f = \left(\frac{\partial f}{\partial y}, \frac{\partial f}{\partial x}\right) = (d_y, d_x)$
- Edge direction (방향)
  - 영상의 밝기 변화가 발생하는 방향 (gradient의 수직 방향)
  - 엣지가 가리키는 방향으로 해석 가능



(a) 에지 방향과 그레이디언트 방향 (b) 에지 방향의 양자화

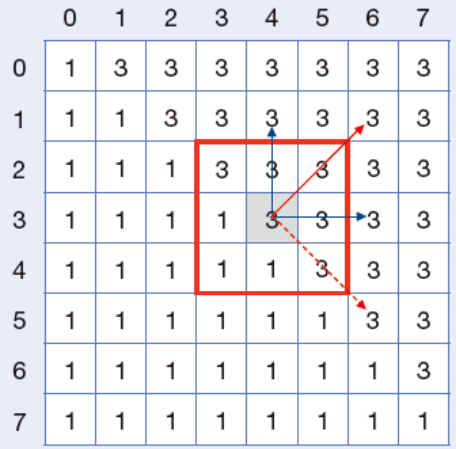
에지 강도: $s(y,x) = \sqrt{f_x'(y,x)^2 + f_y'(y,x)^2}$

그레이디언트 방향: $d(y,x) = \arctan\left(\frac{f_y'(y,x)}{f_x'(y,x)}\right)$ (4.5)

Edge magnitude and edge direction

[예시 4-1] 소벨 연산자 적용 과정

[그림 4-7]은 대각선을 기준으로 위쪽은 3, 아래쪽은 1인 가상의 영상에 소벨 에지 연산자를 적용하는 과정을 예시한다. 회색으로 표시한 (3,4) 화소에 대한 자세한 계산 과정을 설명한다.



$$f_y'(3,4) = -6,\ f_x'(3,4) = 6$$

$$s(3,4) = \sqrt{6^2 + (-6)^2} = 8.485$$

$$d(3,4) = \arctan\left(\frac{-6}{6}\right) = -45°$$

f_x'와 f_y'

그레이디언트 방향

에지 방향

그림 4-7 소벨 연산자 적용 사례

- **소벨(Sobel) 필터**

u_x=

-1	0	1
-2	0	2
-1	0	1

u_y=

-1	-2	-1
0	0	0
1	2	1

$$f_y'(3,4) = (-1\times3) + (-2\times3) + (-1\times3) + (1\times1) + (2\times1) + (1\times3) = -6$$

$$f_x'(3,4) = (-1\times3) + (1\times3) + (-2\times1) + (2\times3) + (-1\times1) + (1\times3) = 6$$

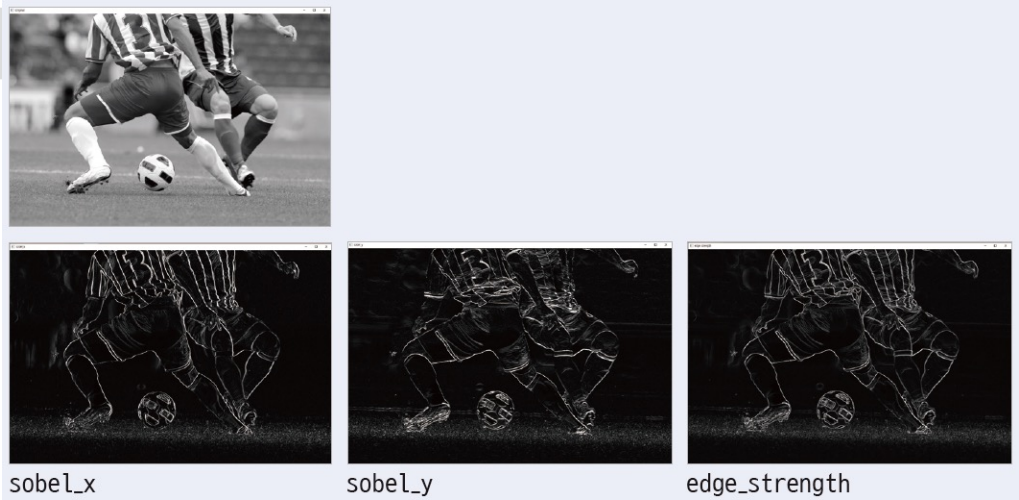
프로그래밍 실습: Sobel filter를 활용한 edge detection

프로그램 4-1 소벨 에지 검출(Sobel 함수 사용)하기

```
01  import cv2 as cv
02
03  img=cv.imread('soccer.jpg')
04  gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)
05
06  grad_x=cv.Sobel(gray,cv.CV_32F,1,0,ksize=3)      # 소벨 연산자 적용
07  grad_y=cv.Sobel(gray,cv.CV_32F,0,1,ksize=3)
08
09  sobel_x=cv.convertScaleAbs(grad_x)               # 절댓값을 취해 양수 영상으로 변환
10  sobel_y=cv.convertScaleAbs(grad_y)
11
12  edge_strength=cv.addWeighted(sobel_x,0.5,sobel_y,0.5,0)   # 에지 강도 계산
13
14  cv.imshow('Original',gray)
15  cv.imshow('sobelx',sobel_x)
16  cv.imshow('sobely',sobel_y)
17  cv.imshow('edge strength',edge_strength)
18
19  cv.waitKey()
20  cv.destroyAllWindows()
```

cv.CV_8U(numpy의 uint8)로 변환
0보다 작으면 0, 255보다 크면 255로 바꿈

addWeighted(i1,a,i2,b,c)는 i1*a+i2*b+c를 계산
i1과 i2가 같은 데이터 형이면 결과는 같은 데이터 형, 다르면 오류 발생
i1과 i2가 CV_8U인데 계산 결과가 255를 넘으면 255를 기록



Canny edge

- Sobel ➔ '그럴듯해 보이는' edge detection 기법

- Canny edge detector (by John F. Canny, 1986)
  - edge detection을 최적화 문제의 관점에서 접근
  - 세 가지 목적 함수
    - 1) 최소 오류율
    - 2) 위치 정확도
    - 3) 단일 (edge) 두께

A computational approach to edge detection
J Canny - Pattern Analysis and Machine Intelligence, IEEE ..., 1986 - ieeexplore.ieee.org
Abstract-This paper describes a computational approach to edge detection. The success of the approach depends on the definition of a comprehensive set of goals for the computation of edge points. These goals must be precise enough to delimit the desired behavior of the ...
19422회 인용 관련 학술자료 전체 17개의 버전 Web of Science: 6073 인용 저장

Canny edge

- 1) 최소 오류율 (low error rate)
  - 영상에서 실제 edge에 해당하는 부분만을 정확하게 검출하는 것을 목표로 함
  - edge가 아닌 부분을 edge로 검출하는 것 (false positive), 실제 edge를 검출하지 못하는 것 (false negative)를 최소화
  - 구현 방법
    - Gaussian filter를 통한 noise 제거
    - 이력 임계값 처리 (hysteresis thresholding)을 통해 중요하지 않은 edge 제거
      - 이력 임계값 T_{high}, T_{low}

Canny edge

- 2) 위치 정확도 (accurate localization)
  - 검출된 edge는 실제 edge의 위치(중심)와 가까워야 함
  - edge 검출의 정밀도를 높이는 작업
  - 구현 방법
    - Non-maximum suppression (NMS, 비최대 억제)
    - gradient direction을 고려하여, 실제 edge 방향에 수직인 방향에서 최댓값을 갖는 픽셀을 edge로 선정

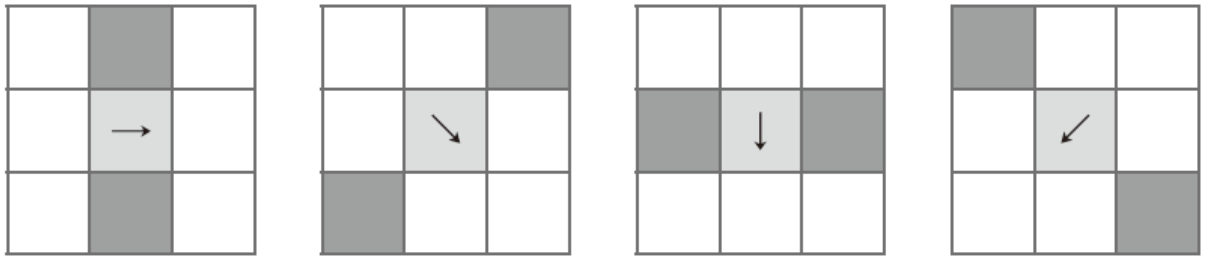
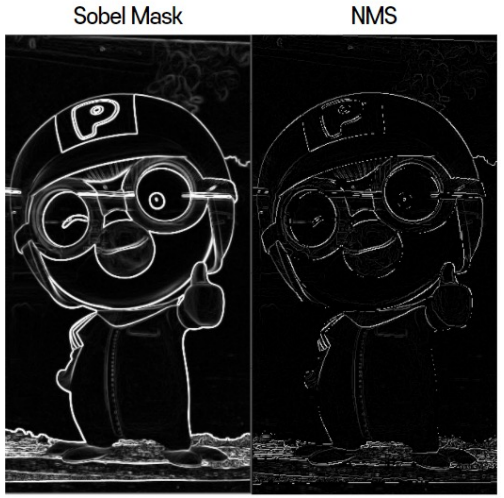


그림 4-8 비최대 억제



Canny edge

- 3) 단일 edge 두께 (single edge response)
  - 하나의 edge에 대해 하나의 edge가 검출되어야 함
  - edge가 두껍게 검출되는 것을 방지 (해석을 단순화)
  - 구현 방법
    - Greedy algorithm에 기반한 “근사해” (approximation)을 찾는 방법으로 접근
    - NMS 과정에서 gradient 방향을 따라서만 edge 후보를 검사하고,
      국지적 최댓값을 가지지 않는 픽셀은 edge로 선택하지 않음

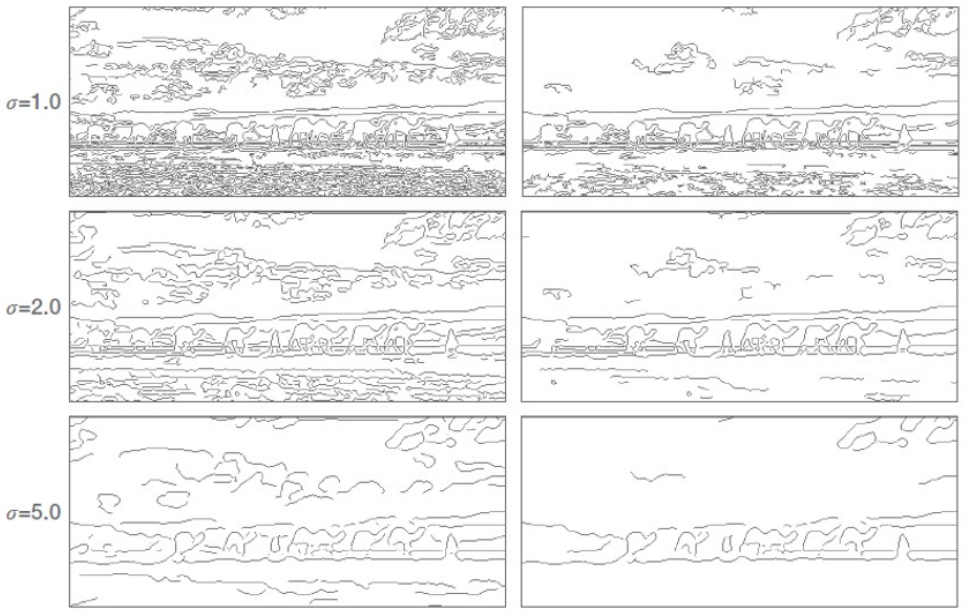
Canny edge

- Canny edge detection algorithm
  - input: image f, gaussian s.d. σ
  - output: edged-binary image (edge = 1, non-edge = 0)
  - algorithm
    - 1) f에 크기가 σ인 Gaussian smoothing 적용 (noise 제거) ➔ Why noise remove? Why Gaussian?
    - 2) edge magnitude / direction 계산 (보통 Sobel filter 적용)
    - 3) NMS 적용하여 얇은 두께의 edge map 생성
    - 4) 이력 임계값(hysteresis threshold)을 활용하여 false positive 제거

Canny edge

- Canny edge detection 예시

(a) 원래 영상(342×800)



(b) 낮은 임계값 (c) 높은 임계값

- Canny edge의 한계점
  - 물체 경계와 그림자 edge를 구별하지 못함
  - ➔ deep learning

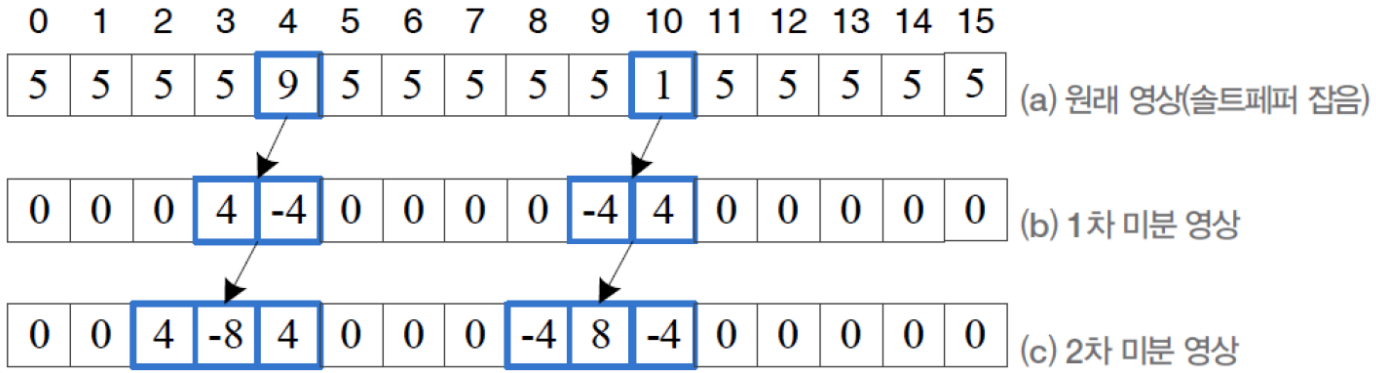
프로그래밍 실습: Canny edge detection

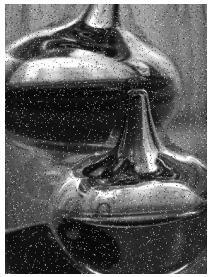
프로그램 4-2 캐니 에지 실험하기

```
01  import cv2 as cv
02
03  img=cv.imread('soccer.jpg')         # 영상 읽기
04
05  gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)
06
07  canny1=cv.Canny(gray,50,150)        # Tlow=50, Thigh=150으로 설정
08  canny2=cv.Canny(gray,100,200)       # Tlow=100, Thigh=200으로 설정
09
10  cv.imshow('Original',gray)
11  cv.imshow('Canny1',canny1)
12  cv.imshow('Canny2',canny2)
13
14  cv.waitKey()
15  cv.destroyAllWindows()
```

LoG filter

- Why noise remove?
  - 미분은 잡음(noise)를 증폭시킬 수 있음 ➔ 미분 연산 전에는 smoothing이 필요





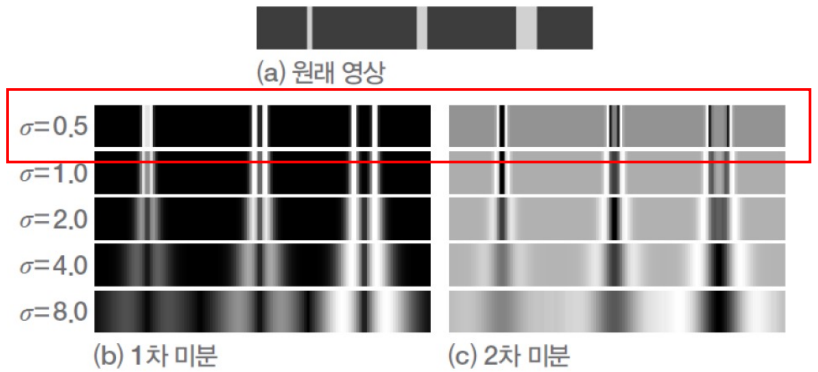
Salt/pepper noise

원래 영상

Laplacian filter

- Why Gaussian?
  - σ를 조절하여 smoothing 정도를 조절할 수 있음

    ➔ edge 스케일 조절 가능
  - σ가 작으면 물체 디테일 edge, 크면 큰 edge



LoG filter

- Laplacian of Gaussian
  - Laplacian
    - 함수 f의 각 parameter의 2차 편미분함수의 합

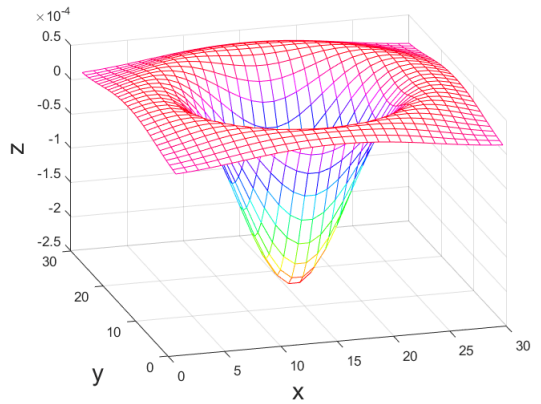
$$\nabla^2 f(y,x) = \frac{\partial f^2}{\partial y^2} + \frac{\partial f^2}{\partial x^2}$$

$$= \big(f(y+1,x) + (y-1,x) - 2f(y,x)\big) + \big(f(y,x+1) + f(y,x-1) - 2f(y,x)\big)$$

$$= f(y+1,x) + f(y-1,x) + f(y,x+1) + f(y,x-1) - 4f(y,x)$$

- ➔ 이에 해당하는 필터 $L =$

0	1	0
1	-4	1
0	1	0



- LoG ➔ Gaussian smoothing을 통해 noise 제거 후 Laplacian filter 적용

프로그래밍 실습: LoG filter

```
import cv2
import numpy as np


image = cv2.imread('./data/dog1.jpg', cv2.IMREAD_GRAYSCALE)

# Gaussian Blur
blurred_image = cv2.GaussianBlur(image, (3, 3), 0.1)

# Laplacian
log_image = cv2.Laplacian(blurred_image, cv2.CV_64F)

log_image = np.uint8(np.absolute(log_image))

cv2.imshow('Original Image', image)
cv2.imshow('Laplacian of Gaussian Image', log_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

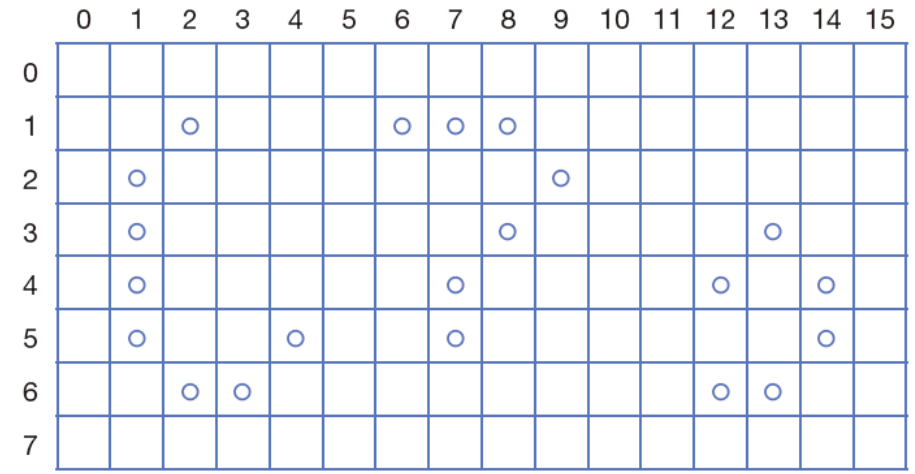
2. 직선 검출

직선 검출

- Edge를 명시적으로 연결하여 경계선을 찾고 직선으로 변환
  - 이후 처리 단계인 물체 표현이나 인식에 유리
  - Edge detection (edge map) ➔ edge 토막 (그룹화) ➔ 직선화

경계선 찾기

- 8-connectivity edge 화소를 연결해 경계선(contour) 구성



경계선1: (1,2)(2,1)(3,1)(4,1)(5,1)(6,2)(6.3)(5,4)
경계선2: (1,6)(1,7)(1,8)(2,9)(3,8)(4,7)(5,7)
경계선3: (4,12)(3,13)(4,14)(5,14)(6,13)(6,12)

그림 4-9 에지 맵에서 경계선 찾기

프로그래밍 실습: Edge map에서 경계선 찾기

프로그램 4-3 에지 맵에서 경계선 찾기

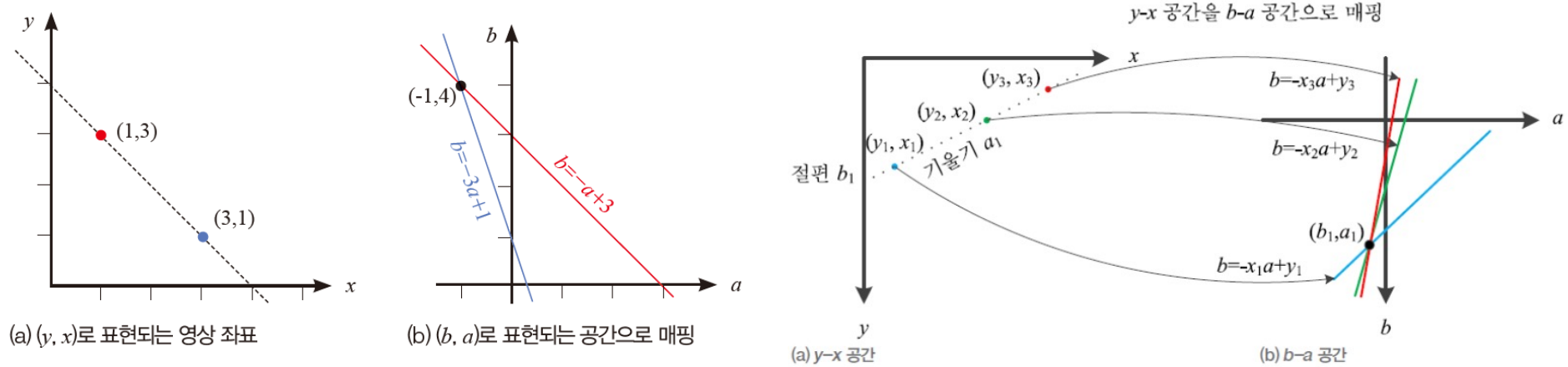
```
01  import cv2 as cv
02  import numpy as np
03
04  img=cv.imread('soccer.jpg')              # 영상 읽기
05  gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)
06  canny=cv.Canny(gray,100,200)
07
08  contour,hierarchy=cv.findContours(canny,cv.RETR_LIST,cv.CHAIN_APPROX_NONE)
09
10  lcontour=[]
11  for i in range(len(contour)):
12      if contour[i].shape[0]>100:          # 길이가 100보다 크면
13          lcontour.append(contour[i])
14
15  cv.drawContours(img,lcontour,-1,(0,255,0),3)
16
17  cv.imshow('Original with contours',img)
18  cv.imshow('Canny',canny)
19
20  cv.waitKey()
21  cv.destroyAllWindows()
```

이 매개변수를 통해
여러 가지 근사 방법 제공

시작점으로 돌아올 때까지 추적하므로
실제로는 50보다 크면

Hough transform

- 허프 변환
  - 영상에서 직선, 원 등과 같은 기하학적 형태를 검출하기 위한 기법
  - 도로, 건물의 윤곽, 원형 패턴 등 다양한 형태의 객체 인식에 사용됨 (before deep learning era...)
  - Idea: 끊긴 edge를 모아서 직선/원 등을 검출해보자
- 직선 검출의 원리: **공간의 변환** $y = ax + b$ ➔ $b = -xa + y$
  - 각각의 점 (y_i, x_i)에 대해 (b, a)공간에 직선 $b = -ax_i + y_i$를 그림
  - (b, a)공간에서 직선이 만나는 점을 절편과 기울기로 취함 ➔ 교점은 투표로 계산

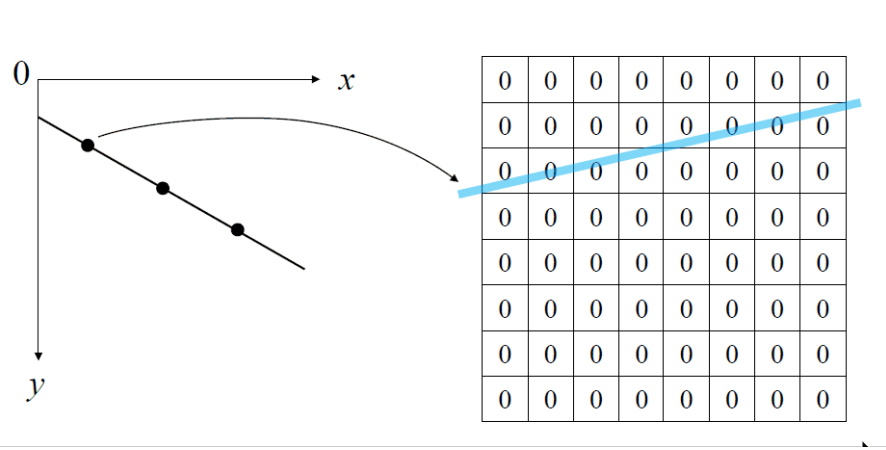


(a) (y, x)로 표현되는 영상 좌표 (b) (b, a)로 표현되는 공간으로 매핑

그림 4-10 허프 변환의 원리

Hough transform

- Voting 방법
  - (b, a) 공간을 이산화하고 누적 배열을 만들어 투표를 기록
    - 직선은 자신이 지나는 칸에 1만큼 씩 투표
    - 다수 표를 얻은 점을 결정할 때 NMS 적용 ➔ 지역 최대점 찾음
  - 예를 들어, 직선들의 교점 (a_1, b_1)에 겹치는 직선이 많을수록 $y = m_1 x + b_1$ 이라는 직선이 존재할 가능성이 높아짐을 의미
  - ➔해당 교점 (a_1, b_1)을 지나는 직선이 많으면 검출에 활용
  - ➔ 일직선 상에 있는 점을 한곳으로 모으는 원리



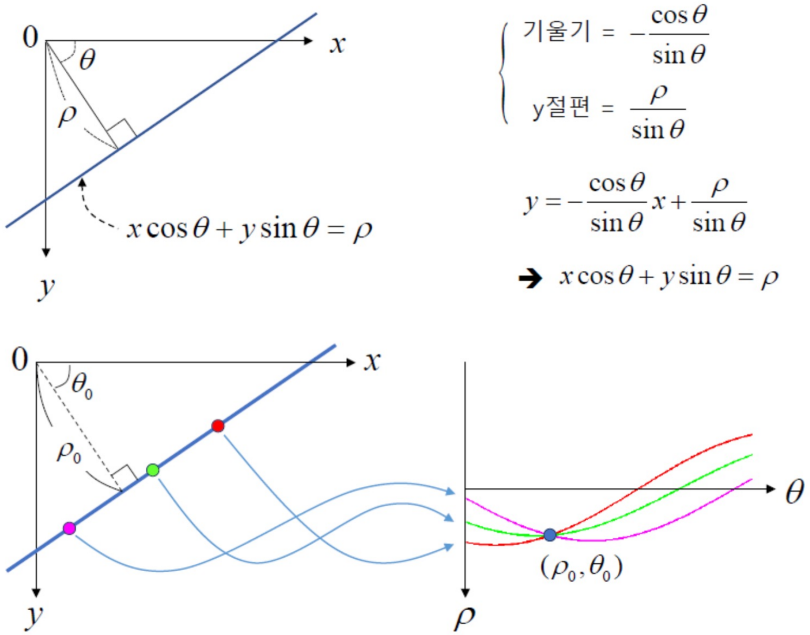
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

0	1	0	0	0	0	0	0
0	2	2	0	1	3	0	0
0	3	5	3	2	0	0	0
0	2	4	2	6	7	0	0
0	2	3	3	5	8	6	0
0	1	0	0	0	4	5	3

그림 4-11 비최대 억제로 찾은 극점 2개

Hough transform

- 직선의 허프 변환
  - $y = ax + b$ ➔ $b = -xa + y$ 변환은 기울기가 수직일 때 표현 불가능
  - ➔ 극좌표계로 변환
    - 극좌표계의 직선의 방정식: $y\cos\theta + x\sin\theta = \rho$



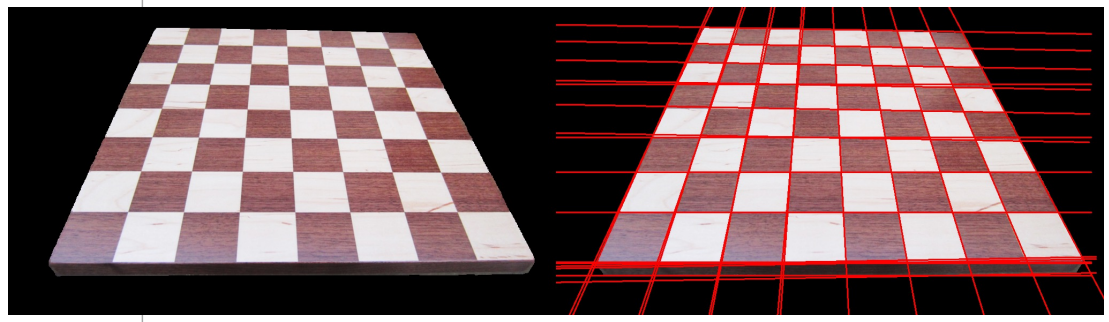
- 원의 허프 변환
  - $(x-a)^2 + (y-b)^2 = r^2$

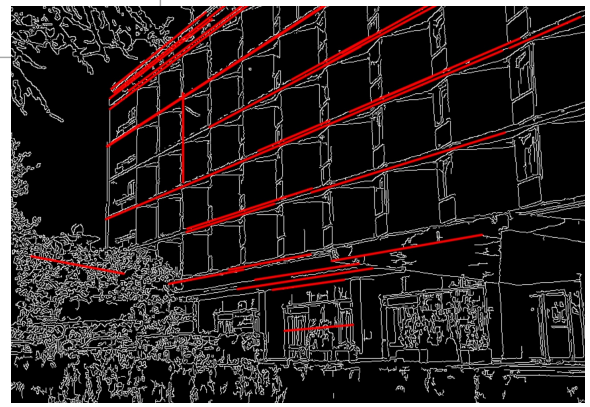
프로그래밍 실습: 허프 변환을 통한 직선/원 검출

프로그램 4-4 허프 변환을 이용해 사과 검출하기

```
01  import cv2 as cv
02
03  img=cv.imread('apples.jpg')
04  gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)
05
06  apples=cv.HoughCircles(gray,cv.HOUGH_GRADIENT,1,200,param1=150,param2=20,
                           minRadius=50,maxRadius=120)
07
08  for i in apples[0]:
09      cv.circle(img,(int(i[0]),int(i[1])),int(i[2]),(255,0,0),2)
10
11  cv.imshow('Apple detection',img)
12
13  cv.waitKey()
14  cv.destroyAllWindows()
```

cv2.HoughLines(image, rho, theta, threshold[, lines[, srn[, stn[, min_theta[, max_theta]]]]]) → lines





- (참고) 또 다른 직선 검출 알고리즘: RANSAC

3. 영역 분할

영역 분할

- 영역 분할 (region segmentation)
  - 사람은 물체의 3차원 모델을 사용하고 주변 상황을 고려하여 의미 분할 수행
  - 컴퓨터비전에서의 영역 분할
    - 물체가 점유한 영역을 구분하는 작업
    - 말 그대로 이미지를 서로 다른 영역으로 나누는 과정
  - ➔ 고전적 컴퓨터비전(영상처리) 방법론/모델
    - 명암 또는 컬러 등 이미지의 특징을 보고 영역을 결정

님 주차 편하게하세요!!

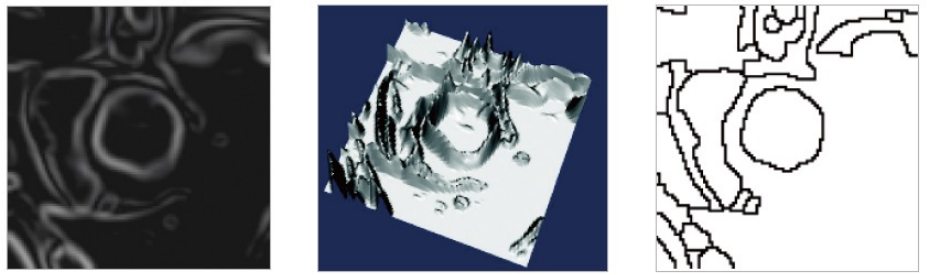
ㄴㄴ 님이 편하게

- VS 의미 분할 (semantic segmentation)
  - 이미지 내의 각 픽셀을 특성 클래스에 할당하는 과정
  - 이미지의 의미를 파악 ➔ 딥러닝 기반 모델 활용

배경이 단순한 영상의 영역 분할

- 단순한 영상의 예
  - 스캔한 책 영상
  - 컨베이어 벨트 위를 움직이는 물체의 영상

- 영역 분할 알고리즘
  - 이진화 알고리즘 (여러 임계값을 사용하는 Otsu 알고리즘, 군집화 알고리즘 등)
  - Watershed 알고리즘
  - Divide-and-conquer
  - Graph-based
  - Optimization, etc.
  - ➔ 2011년 이후로 별로 진전이 없음

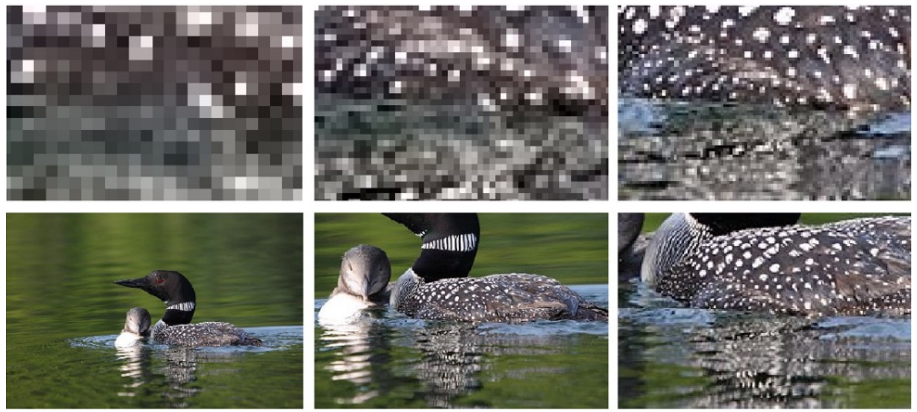


(a) 에지 강도 맵 (b) 지형으로 간주 (c) 워터셰드

그림 4-14 워터셰드 분할 알고리즘[Cousty2007]

영상 분할의 원리

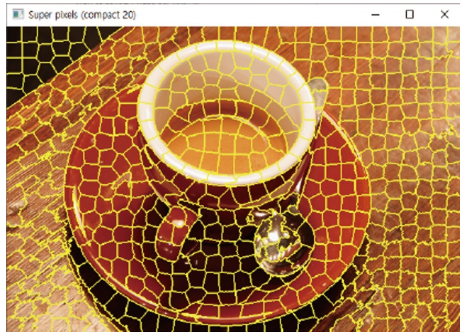
- 분할의 어려움
  - 이웃 화소 몇 개를 보고 영역 내부 또는 외부를 판단할 수 있는가?
  - ➔ global 연산의 필요성

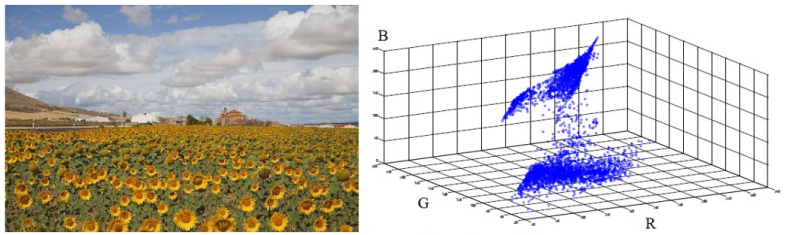


- Edge vs. Region
  - 개념적으로 edge == 영역의 경계
  - 그러나, edge 만으로는 영역 검출에 한계
    - 폐곡선을 이루지 못함

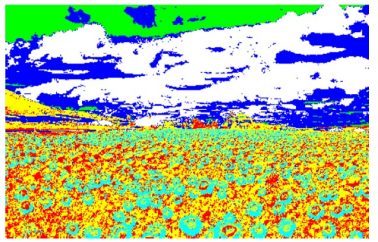
슈퍼 화소 분할

- 슈퍼 화소 (Superpixel)
  - 비슷한 색상, 밝기, 텍스처 등을 가진 인접 픽셀들의 그룹
    - 일반적으로 화소(pixel)보다 크지만 물체보다는 작은 영역
  - 이미지 내에서 보다 의미 있는 구조를 형성 ➔ 중간 수준의 표현 제공
    - 전체 이미지를 이해하는 데 도움





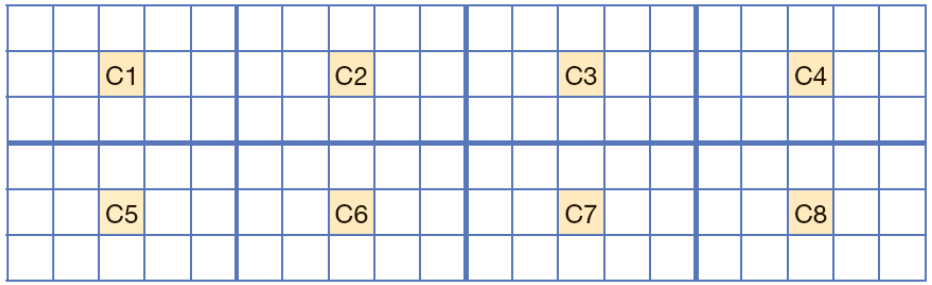
(a) 원래 영상 (b) 3차원 공간에 매핑한 결과



- 슈퍼 화소 분할 기법
  - SLIC (Simple Linear Iterative Clustering)
  - Quick Shift
  - Watershed
  - LSC (Linear Spectral Clustering)

SLIC algorithm

- K-means clustering algorithm과 매우 유사
  - 색상과 공간적 접근성을 기반으로 픽셀을 클러스터링 하는 개념
  - 픽셀 할당 단계와 군집 중심 갱신 단계의 반복



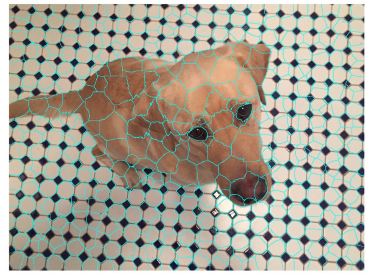


그림 4-15 SLIC 알고리즘의 초기 군집 중심

- 한계점
  - 지역적 명암/밝기/화소의 변화를 조사 ➔ 좁은 크기의 슈퍼 화소 그룹 형성 (이미지 전체에서 영역 분할 불가)
    - ex) 양말의 색이 배경과 비슷하면 양말이 배경 영역에 섞임
  - 전역적 정보를 고려할 필요가 있음
    - 지역적으로 색상 변화가 약하지만 전역적으로 유리하다면 (물체의) 경계로 간주
    - 영상을 그래프로 표현, 최적화 관점으로 접근

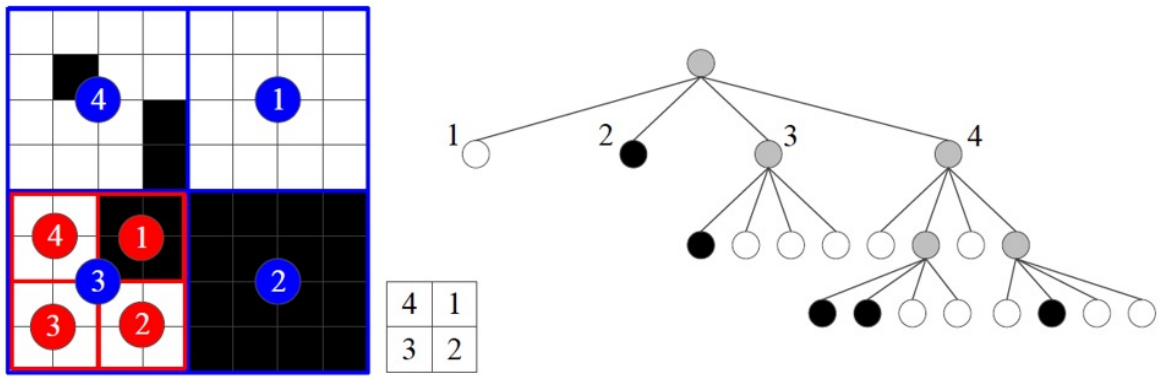
프로그래밍 실습: SLIC을 이용한 슈퍼 화소 분할

프로그램 4-5 SLIC 알고리즘으로 입력 영상을 슈퍼 화소 분할하기

```
01  import skimage
02  import numpy as np
03  import cv2 as cv
04
05  img=skimage.data.coffee()
06  cv.imshow('Coffee image',cv.cvtColor(img,cv.COLOR_RGB2BGR))
07
08  slic1=skimage.segmentation.slic(img,compactness=20,n_segments=600)
09  sp_img1=skimage.segmentation.mark_boundaries(img,slic1)
10  sp_img1=np.uint8(sp_img1*255.0)
11
12  slic2=skimage.segmentation.slic(img,compactness=40,n_segments=600)
13  sp_img2=skimage.segmentation.mark_boundaries(img,slic2)
14  sp_img2=np.uint8(sp_img2*255.0)
15
16  cv.imshow('Super pixels (compact 20)',cv.cvtColor(sp_img1,cv.COLOR_RGB2BGR))
17  cv.imshow('Super pixels (compact 40)',cv.cvtColor(sp_img2,cv.COLOR_RGB2BGR))
18
19  cv.waitKey()
20  cv.destroyAllWindows()
```

(참고) Divide-and-Conquer 기반의 분할

- 원리
  - 영역의 "균일성"을 측정하는 $Q(r_i)$를 이용하여 분할-합병 반복
    - $Q(r_i)$이 균일성 검증을 통과하지 못하면 ➔ 네 개의 영역으로 분할
    - $Q(r_i \cup r_j)$이 균일하면 ➔ r_i와 r_j를 합병하여 한 영역으로 취급
    - ➔ 재귀 반복
    - 분할 결과는 4진 트리로 표현

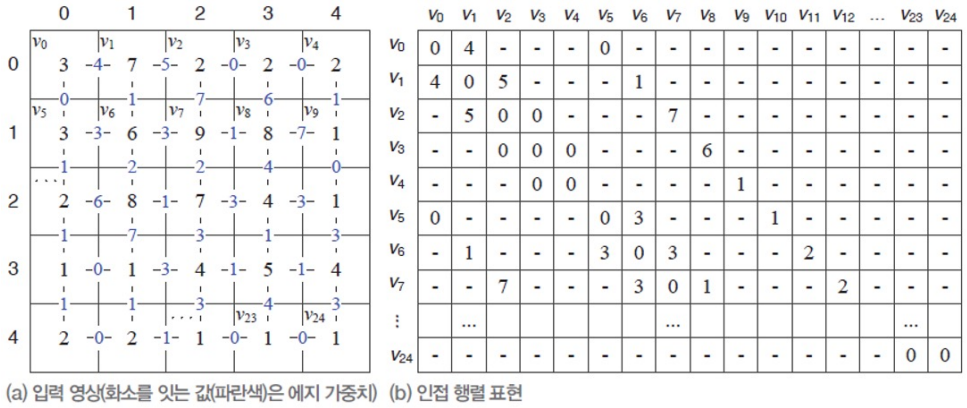


최적화 분할

- 영상의 그래프 표현
  - Node (vertex): 화소 또는 슈퍼 화소
  - Edge: 두 노드의 유사도 s_{pq}

$$d_{pq} = \begin{cases} \|f(v_p) - f(v_q)\|, & if\ v_q \in neighbor(v_p) \\ \infty, & otherwise \end{cases}$$

$$s_{pq} = \begin{cases} D - d_{pq} \quad \text{or} \quad \dfrac{1}{e^{d_{pq}}}, & if\ v_q \in neighbor(v_p) \\ \infty, & otherwise \end{cases}$$



(a) 입력 영상(화소를 잇는 값(파란색)은 에지 가중치) (b) 인접 행렬 표현

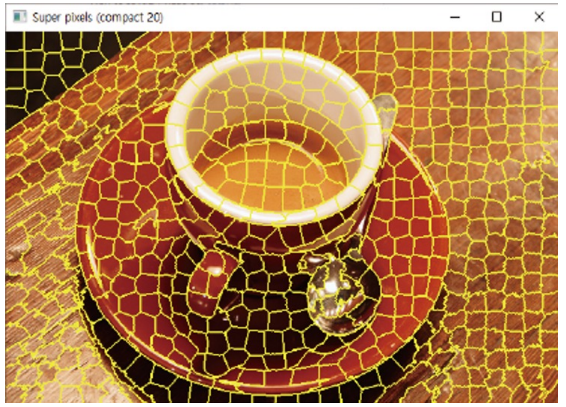
- $f(v) = \{y, x, r, g, b\}$
  - 화소의 색상(R, G, B)와 위치(y, x)를 결합한 5차원 벡터
- v가 슈퍼 화소인 경우 화소 평균을 사용

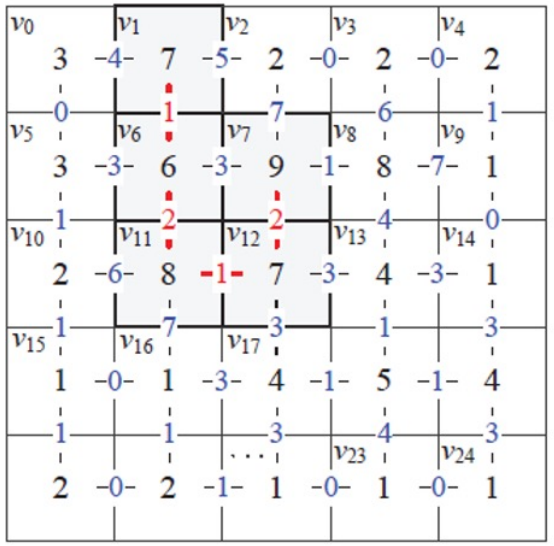
최적화 분할

- 정규화 절단(normalized cut) 알고리즘
  - cut
    - 영상을 두 영역(C_1, C_2)으로 분할했을 때 분할의 좋은 정도를 측정해주는 목적 함수

$$cut(C_1, C_2) = \sum_{v_p \in C_1, v_q \in C_2} s_{pq} \qquad (4.9)$$

      - C_1과 C_2가 클수록 둘 사이에 edge가 많아짐 ➔ cut의 값이 덩달아 커지는 효과
      - 전체 영상 내 영역들의 크기가 균일하지 않음 (영역의 크기가 작아짐)





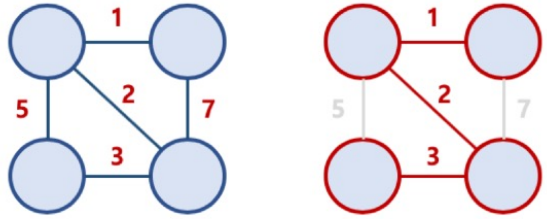
최적화 분할

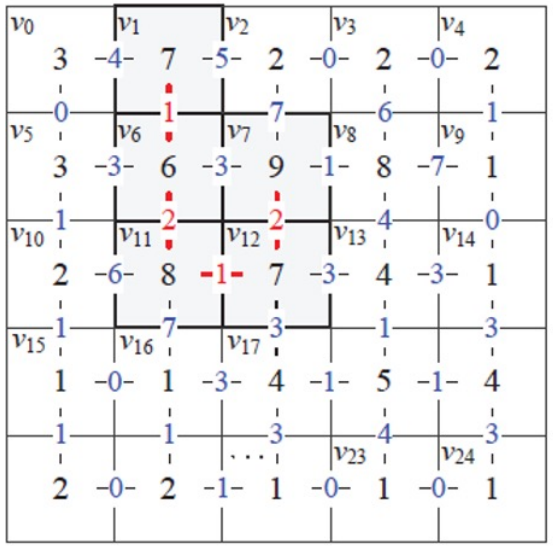
- 정규화 절단(normalized cut) 알고리즘
  - ncut
    - cut을 정규화하여 전체 이미지에 대해 균형있는 영역 분할

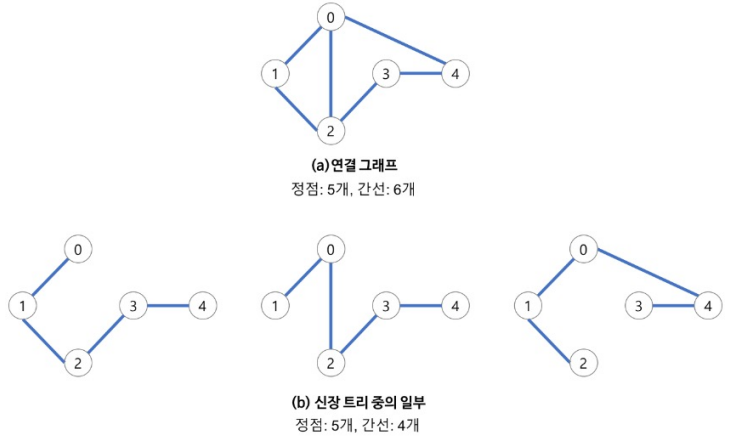
$$ncut(C_1, C_2) = \frac{cut(C_1, C_2)}{cut(C_1, C)} + \frac{cut(C_1, C_2)}{cut(C_2, C)} \qquad (4.10)$$

(참고) Minimum Spanning Tree 분할

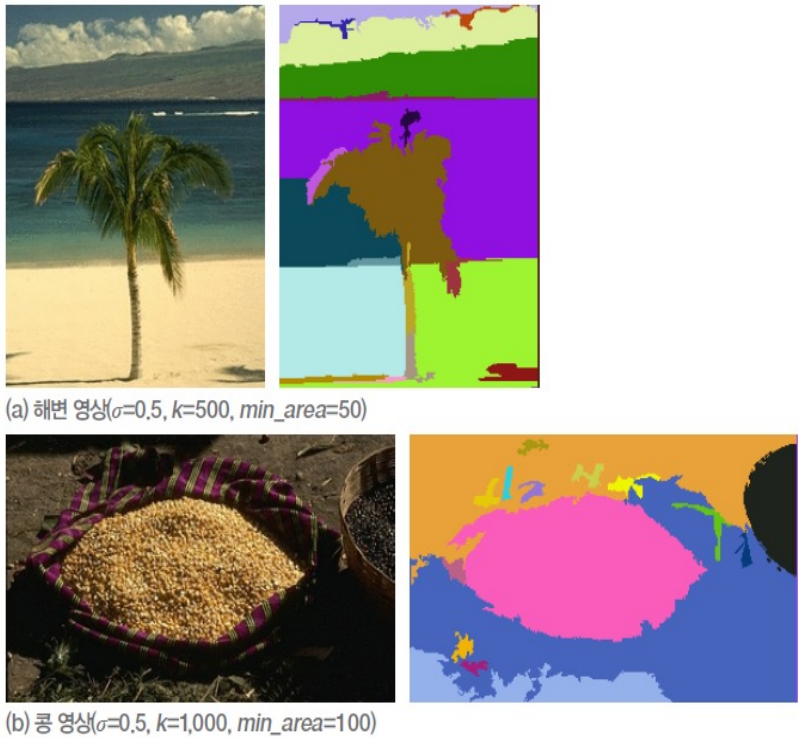
- MST
  - 모든 노드를 연결하면서 tree 조건을 만족하는 그래프 중에서 edge간 weight의 합이 최소인 spanning tree







$$MST(C) = \{(v_1, v_6), (v_6, v_{11}), (v_{11}, v_{12}), (v_{12}, v_7)\}$$


(a) 해변 영상(σ=0.5, k=500, min_area=50)

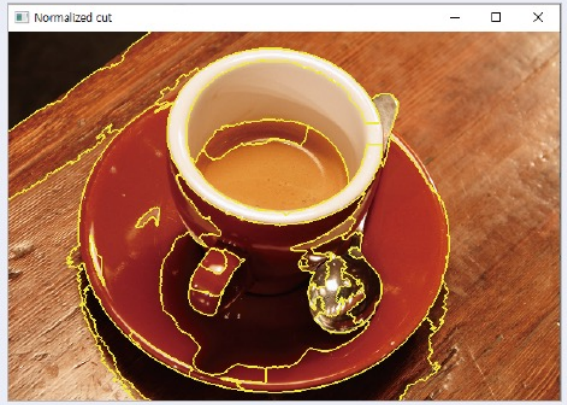
(b) 콩 영상(σ=0.5, k=1,000, min_area=100)

프로그래밍 실습: 정규화 절단 알고리즘

프로그램 4-6 정규화 절단 알고리즘으로 영역 분할하기

```
01  import skimage
02  import numpy as np
03  import cv2 as cv
04  import time
05
06  coffee=skimage.data.coffee()
07
08  start=time.time()
09  slic=skimage.segmentation.slic(coffee,compactness=20,n_segments=600,start_
                                   label=1)
10  g=skimage.future.graph.rag_mean_color(coffee,slic,mode='similarity')
11  ncut=skimage.future.graph.cut_normalized(slic,g)  # 정규화 절단
12  print(coffee.shape,' Coffee 영상을 분할하는 데 ',time.time()-start,'초 소요')
13
14  marking=skimage.segmentation.mark_boundaries(coffee,ncut)
15  ncut_coffee=np.uint8(marking*255.0)
16
17  cv.imshow('Normalized cut',cv.cvtColor(ncut_coffee,cv.COLOR_RGB2BGR))
18
19  cv.waitKey()
20  cv.destroyAllWindows()
```

```
(400, 600, 3)  Coffee 영상을 분할하는 데 6.4380834102630615초 소요
```



4. 영역 특징

영역 특징

- 영역의 레이블링과 기하 변환 예시

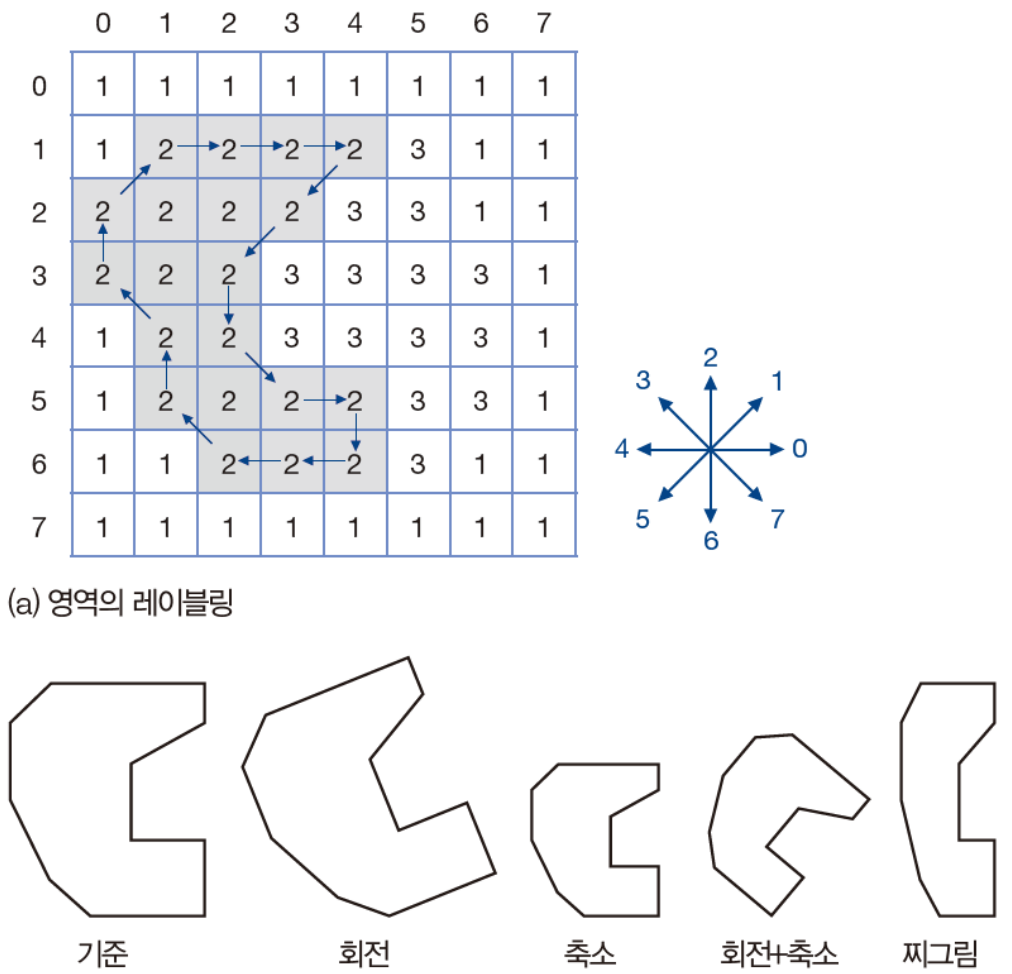


그림 4-17 영역의 레이블링과 기하 변환

특징의 불변성과 등변성

- 불변성 (Invariant)
  - 변환을 해도 값이 변하지 않는 특징
  - ex) 성별 ➔ 나이에 불변

- 등변성 (Equivariant)
  - 변환에 따라 값이 변하는 특징
  - ex) 면적 ➔ 축소에는 등변, 회전에는 등변이 아님

- 목적에 따라 특징 선택이 중요
  - ex) 물체 인식 후 물체를 집는 로봇 ➔ 회전에 등변인 특징을 사용해야 함

영상에서의 영역 특징

- 모먼트(moment)와 모양(shape) 특징
  - 영역 R의 모먼트

$$m_{qp}(R) = \sum_{(y,x)\in R} y^q x^p \qquad (4.13)$$

  - moment-based features

$$\left.\begin{aligned} &\text{면적: } a = m_{00} \\ &\text{중점: } (\dot{y},\dot{x}) = \left(\frac{m_{10}}{a}, \frac{m_{01}}{a}\right) \end{aligned}\right\} \qquad (4.14)$$

$$\mu_{qp} = \sum_{(y,x)\in R} (y-\dot{y})^q (x-\dot{x})^p \qquad (4.15)$$

$$\left.\begin{aligned} &\text{열 분산: } \nu_{cc} = \frac{\mu_{20}}{a} \\ &\text{행 분산: } \nu_{rr} = \frac{\mu_{02}}{a} \\ &\text{열행 분산: } \nu_{rc} = \frac{\mu_{11}}{a} \end{aligned}\right\} \qquad (4.16)$$

$$\eta_{qp} = \frac{\mu_{qp}}{\mu_{00}^{\left(\frac{q+p}{2}+1\right)}} \qquad (4.17)$$

영상에서의 영역 특징

- **영역의 둘레와 둥근 정도**

$$\left.\begin{aligned} \text{둘레: } & p = n_{even} + n_{odd}\sqrt{2} \\ \text{둥근 정도: } & r = \frac{4\pi a}{p^2} \end{aligned}\right\} \qquad (4.18)$$

- **주축의 방향**

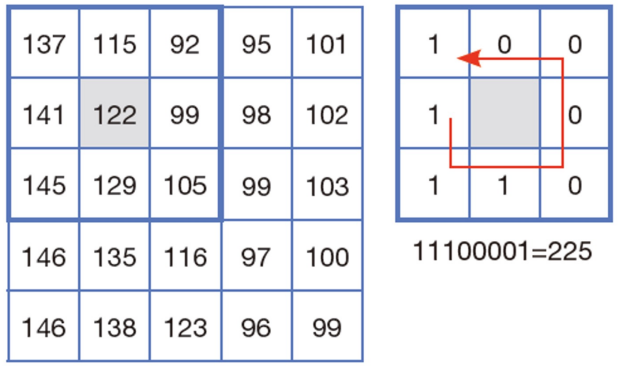
$$\text{주축의 방향: } \theta = \frac{1}{2}\arctan\left(\frac{2\mu_{11}}{\mu_{20} - \mu_{02}}\right) \qquad (4.19)$$

영상에서의 영역 특징

- Texture 특징: 일정 패턴의 반복
  - edge 통계량으로 texture 특징을 측정

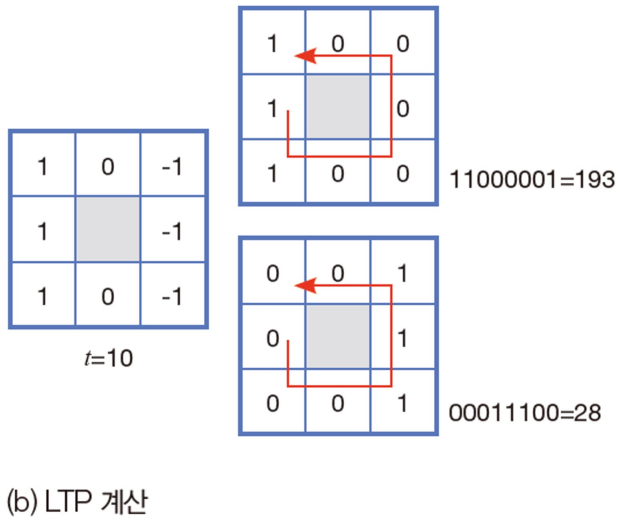
$$T_{edge} = \left(busy, mag\left(i\right), dir\left(j\right)\right), 0 \le i \le q-1, 0 \le j \le 7 \qquad (4.20)$$

  - LBP (local binary pattern)와 LTP (local ternary pattern)
    - LBP는 중심 화소와 주위 화소 명암을 비교해서 텍스처 측정 (256차원 특징벡터)
    - LTP는 작은 명암 변화에 민감한 LBP 단점을 개선 (512차원 특징벡터)
      - 중심화소에서 t를 기준으로 +/-에 따라 texture 측정



(a) LBP 계산

그림 4-18 LBP와 LTP 구하기



(b) LTP 계산

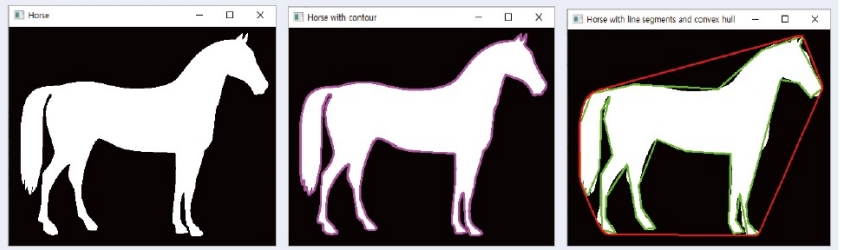
프로그래밍 실습: 영역 특징 추출

프로그램 4-8 이진 영역의 특징을 추출하는 함수 사용하기

```
01  import skimage
02  import numpy as np
03  import cv2 as cv
04
05  orig=skimage.data.horse()
06  img=255-np.uint8(orig)*255
07  cv.imshow('Horse',img) ①
08
09  contours,hierarchy=cv.findContours(img,cv.RETR_EXTERNAL,cv.CHAIN_APPROX_NONE)
10
11  img2=cv.cvtColor(img,cv.COLOR_GRAY2BGR)              # 컬러 디스플레이용 영상
12  cv.drawContours(img2,contours,-1,(255,0,255),2)
13  cv.imshow('Horse with contour',img2) ②
14
15  contour=contours[0]
16
17  m=cv.moments(contour)                                # 몇 가지 특징
18  area=cv.contourArea(contour)
19  cx,cy=m['m10']/m['m00'],m['m01']/m['m00']
20  perimeter=cv.arcLength(contour,True)
21  roundness=(4.0*np.pi*area)/(perimeter*perimeter)
22  print('면적=',area,'\n중점=(',cx,',',cy,')','\n둘레=',perimeter,'\n둥근 정도=',
        roundness) ③
23
24  img3=cv.cvtColor(img,cv.COLOR_GRAY2BGR)              # 컬러 디스플레이용 영상
25
26  contour_approx=cv.approxPolyDP(contour,8,True)       # 직선 근사
27  cv.drawContours(img3,[contour_approx],-1,(0,255,0),2)
28
29  hull=cv.convexHull(contour)                          # 볼록 헐
30  hull=hull.reshape(1,hull.shape[0],hull.shape[2])
31  cv.drawContours(img3,hull,-1,(0,0,255),2)
32
33  cv.imshow('Horse with line segments and convex hull',img3) ④
34
35  cv.waitKey()
36  cv.destroyAllWindows()
```

```
면적= 42390.0 ③
중점=( 187.72464024534088 , 144.43640402610677 )
둘레= 2296.7291333675385
둥근 정도= 0.1009842680321435
```

① ② ④



End of slide