영상처리 실제

Image Processing Practice

신재혁

naezang@cbnu.ac.kr

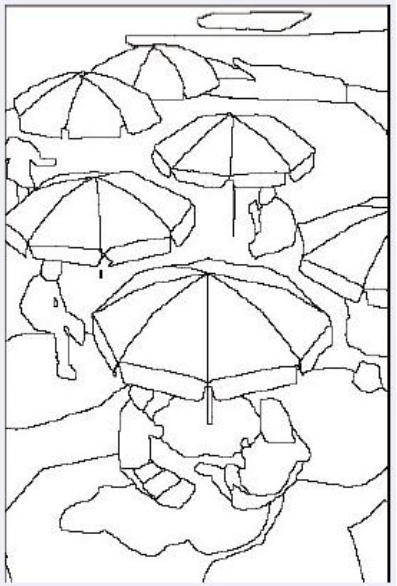
4장 에지와 영역

에지 검출

에지 검출

- 에지 검출과 영역 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/

에지 검출

- 에지와 영역은 쌍대 문제dual problem지만 다른 접근방법 사용
  - 에지는 특성이 다른 곳을 검출하지만 영역은 유사한 화소를 묶는 방법 사용
  - 쌍대문제: 원문제(primal problem)와 수학적으로 관련된 또 다른 최적화 문제

- 사람은 의미 분할semantic segmentation에 능숙
  - 사람은 머리 속에 기억된 물체 모델을 활용하여 의미 분할
  - 이 장에서 공부하는 고전적인 방법은 의미 분할 불가능
  - 딥러닝은 의미 분할이 가능해져 혁신을 일으킴(9장)

에지 검출

- 에지 검출 알고리즘
  - 물체 내부는 명암이 서서히 변하고 경계는 급격히 변하는 특성을 활용

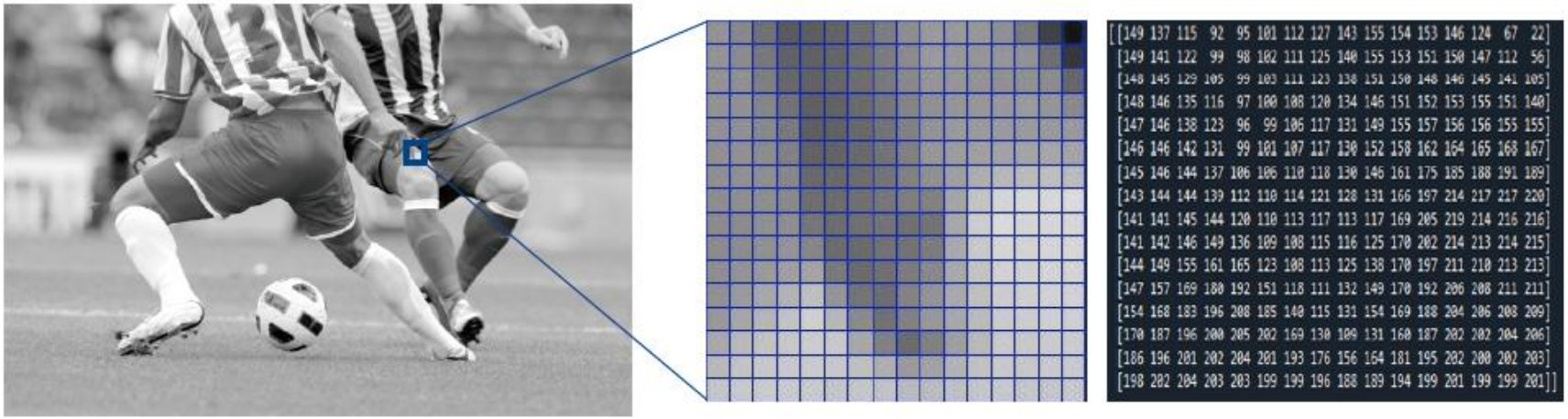


그림 4-2 명암 변화를 확인하기 위해 영상 일부를 확대

에지 검출

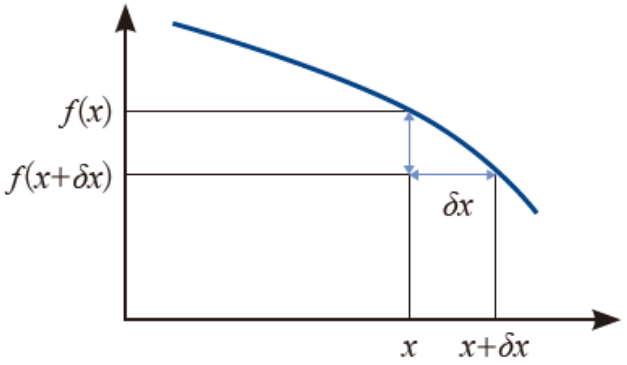
- 영상의 미분
  - 변수 x가 미세하게 증가했을 때 함수 변화량을 측정하는 수학 기법

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

  - 미분을 디지털 영상에 적용하면,

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

  - 실제 구현은 필터 u로 컨볼루션 (u를 에지 연산자라 부름)



(a) 연속 함수의 미분

(b) 디지털 영상의 미분(필터 u로 컨볼루션)

그림 4-3 연속 함수와 디지털 영상의 미분

에지 검출

- 현실 세계의 램프 에지
  - 명암이 몇 화소에 걸쳐 변함

- 1차 미분과 2차 미분

그림 4-4 현실 세계에서 발생하는 램프 에지

에지 검출

- 2차 미분

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

이 식을 구현하는 필터는

1	-2	1

에지 검출

- 에지 검출
  - 1차 미분에서 봉우리 찾기
  - 2차 미분에서 영교차$_{\text{zero-crossing}}$ 찾기

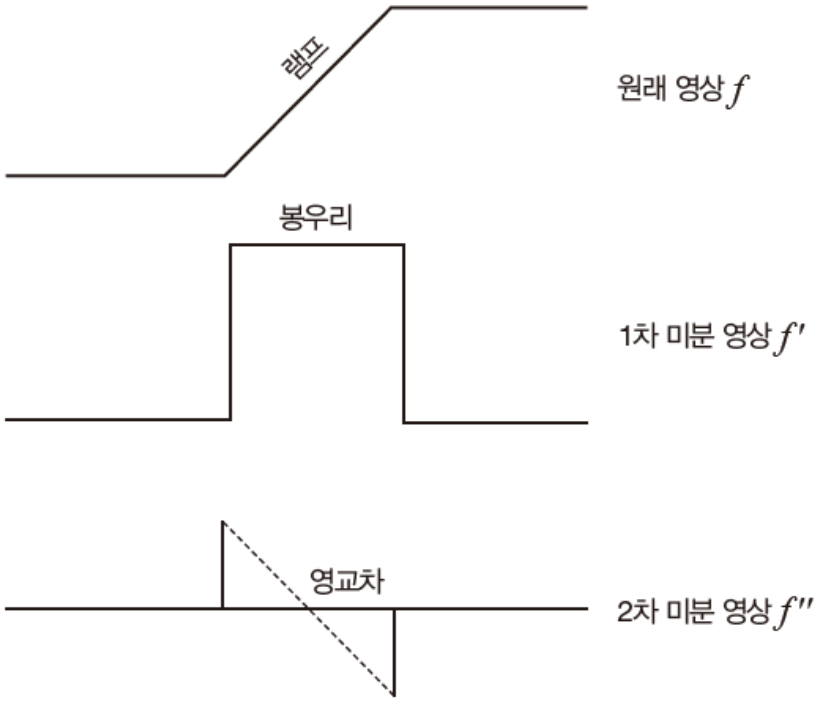


그림 4-5 램프 에지에서 발생하는 봉우리와 영교차

에지 검출

- 1차 미분에 기반한 에지 연산자
  - 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 |

  - 또한 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

에지 검출

- Prewitt: 경계 검출+ 약간의 스무딩 효과
- Sobel: 중앙 픽셀에 가중치 → 노이즈에 덜 민감

- 에지 강도와 에지 방향

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)

에지 검출

이들 맵은 음수를 포함하며
실수이므로 32비트 실수 형
cv.CV_32F로 지정할 필요

[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.

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

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

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

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

→ f_x'와 f_y'

→ 그레이디언트 방향

⇢ 에지 방향

그림 4-7 소벨 연산자 적용 사례

에지 검출

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

에지 검출

sobel_x

sobel_y

edge_strength

캐니 에지

캐니 에지

■ 에지 검출을 최적화 문제로 품

- 최소 오류율, 위치 정확도, 한 두께라는 세 가지 기준으로 목적 함수 정의
- 한 두께를 위해 비최대 억제non-maximum suppression(NMS) 적용

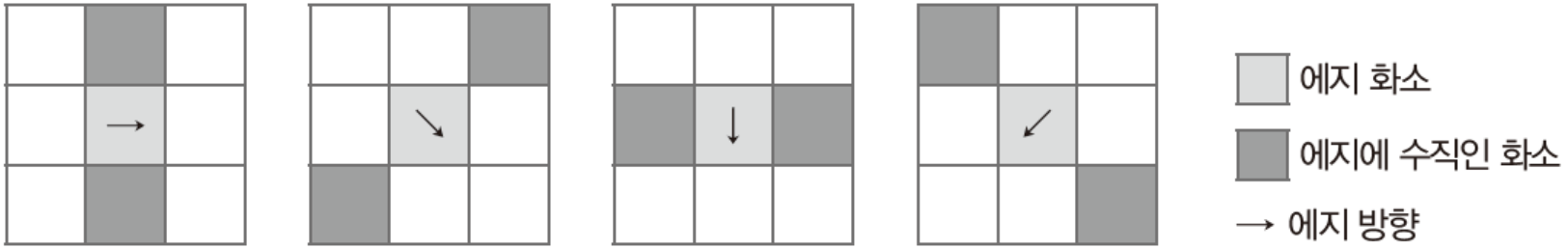


그림 4-8 비최대 억제

- 거짓 긍정을 줄이기 위해 두 개의 이력 임계값 사용
  - 에지 강도가 T_{high} 이상인 에지에서 에지 추적 시작
  - 이후 추적은 T_{low} 이상인 에지를 대상으로 진행

캐니 에지

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()
```

캐니 에지

물체 경계와 그림자 에지를 구별하지 못하는 한계

직선 검출

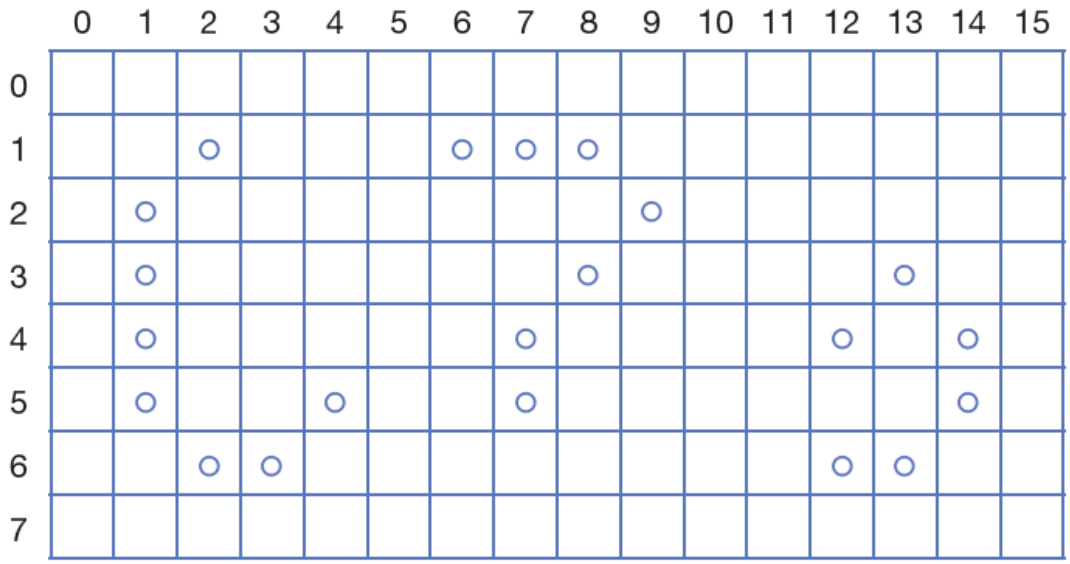
직선 검출

- 에지를 명시적으로 연결하여 경계선을 찾고 직선으로 변환
  - 이후 처리 단계인 물체 표현이나 인식에 유리

경계선 찾기

■ 경계선 찾기

- 8-연결된 에지 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

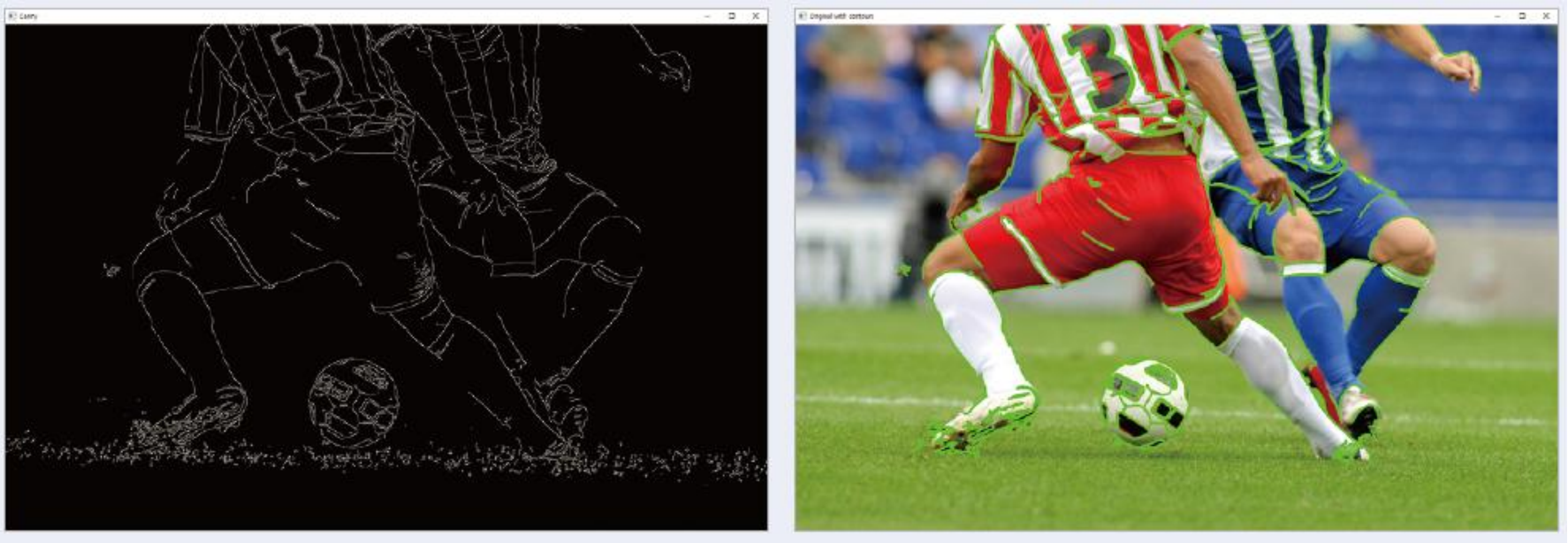
경계선 찾기

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()
```

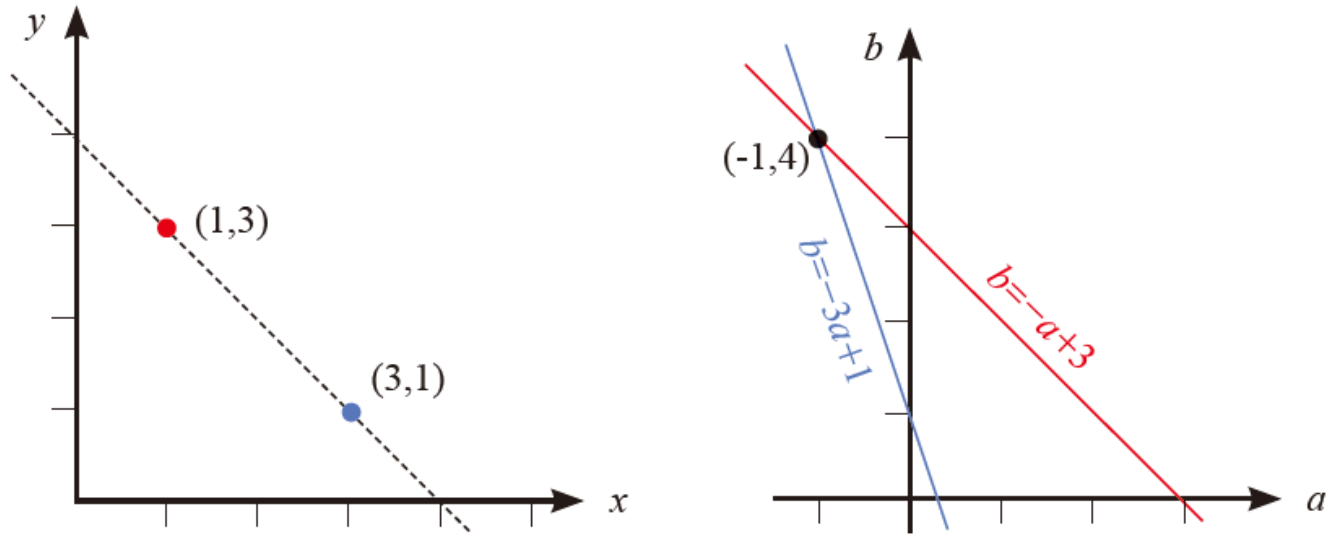
이 매개변수를 통해
여러 가지 근사 방법 제공

경계선 찾기



허프 변환

- **허프 변환은 끊긴 에지를 모아 직선 또는 원 등을 검출**
- **직선 검출의 원리**
  - 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. 만나는 점은 투표로 알아냄



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

그림 4-10 허프 변환의 원리

허프 변환

■ 구현

- (b, a) 공간을 이산화하고 누적 배열 만들어 투표를 기록
- 직선은 자신이 지나는 칸에 1만큼씩 투표
- 다수 표를 얻은 점을 결정할 때 비최대 억제 적용하여 지역 최대점 찾음

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

- 극좌표에서 정의된 직선의 방정식 사용 (기울기가 무한대인 경우 대처)

$$x\sin(\theta)+y\cos(\theta)=\rho \qquad (4.6)$$

허프 변환

■ **원 검출을 위한 허프 변환은 3차원 누적 배열 사용**

$$(x-a)^2+(y-b)^2=r^2 \qquad (4.7)$$

허프 변환

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()
```

RANSAC

- 허프 변환과 최소평균제곱오차LMSE; Least Mean Squared Error
  - 강인하지 않은 추정 기법
  - 아웃라이어를 걸러내는 기능이 전혀 없음 (모든 점에 같은 기회 부여)

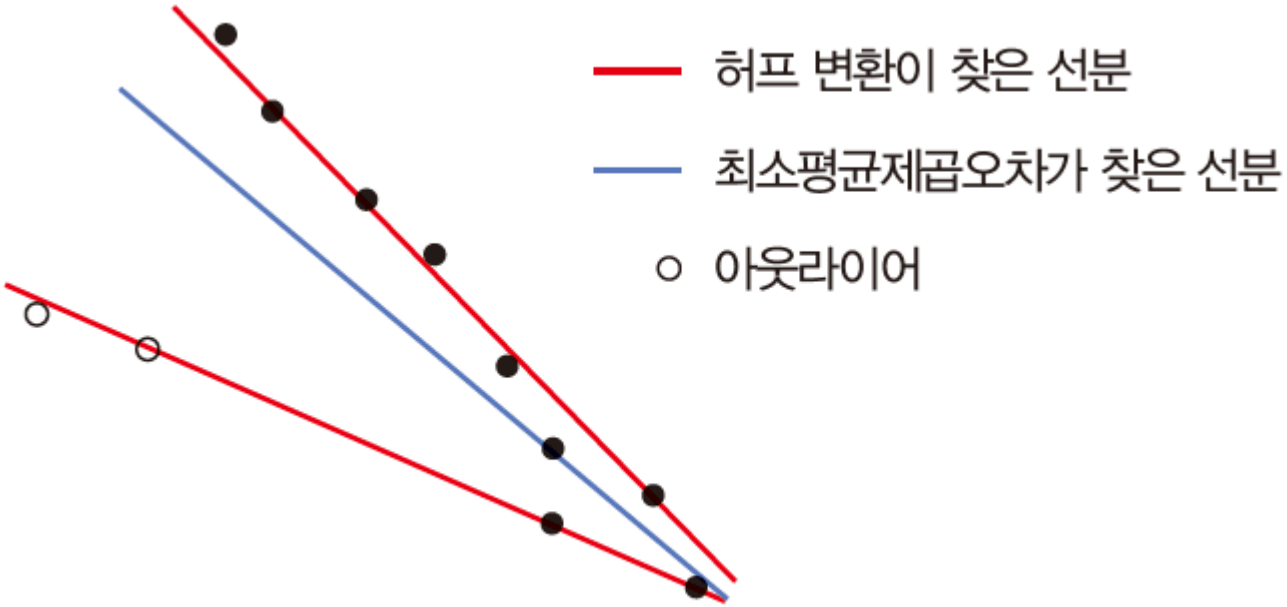
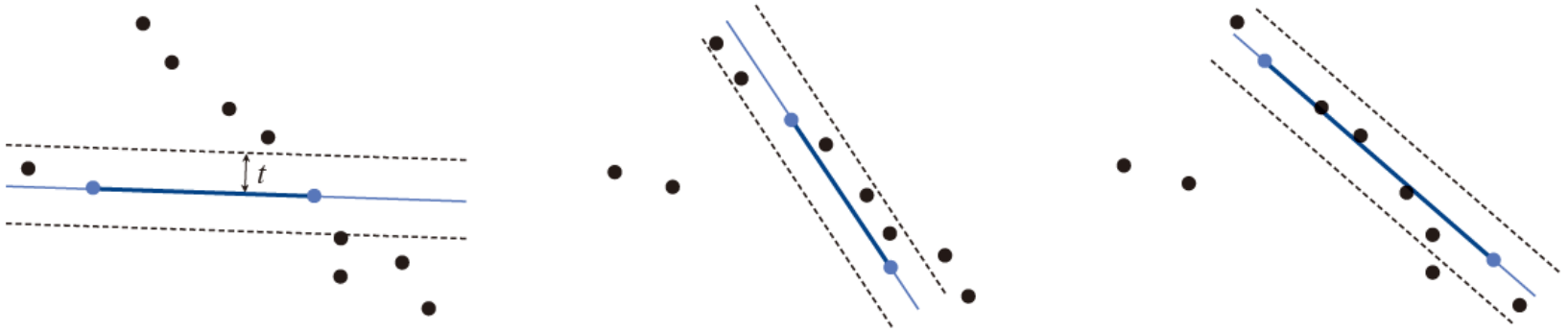


그림 4-12 강인하지 않은 기법의 선분 추정

RANSAC

- 강인한$_{\text{robust}}$ 추정 기법
  - 예) 물건의 길이를 5번 측정한 결과 {16,1,1,1,1}에 평균 적용하면 4, 중앙값$_{\text{median}}$ 적용하면 1. 중앙값은 강인한 추정 기법

- RANSAC은 강인한 추정 기법
  - RANSAC은 구체적인 문제에 두루 활용할 수 있는 메타 알고리즘
  - 난수를 생성하여 추정하는 일을 충분히 많이 반복하고 가장 신뢰할 수 있는 값 선택



(a) 1차 시도 (b) 2차 시도 (c) 3차 시도

그림 4-13 RANSAC으로 선분 추정(1차, 2차, 3차, 4차, … 시도를 반복)

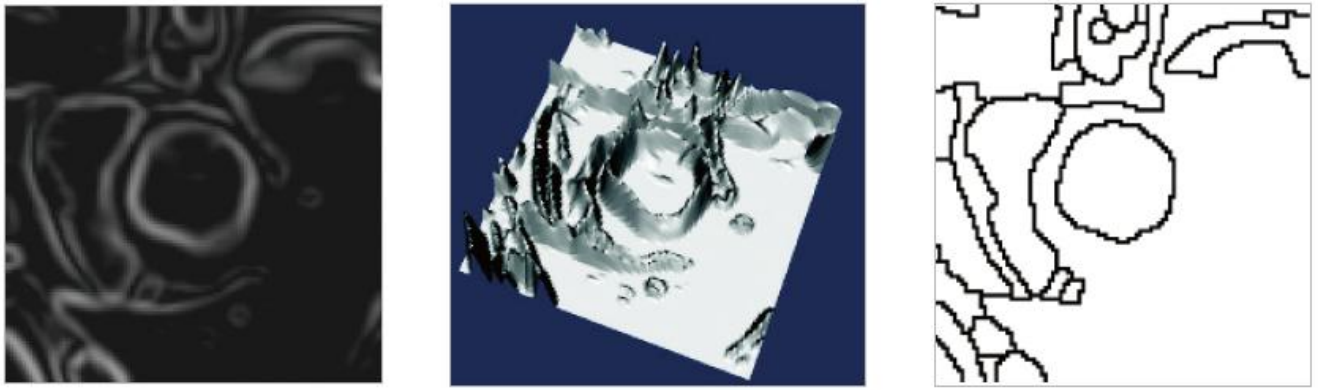
영역 분할

영역 분할

- 영역 분할region segmentation은 물체가 점유한 영역을 구분하는 작업
  - 사람은 물체의 3차원 모델을 사용하고 주의 집중을 적용하여 의미 분할 수행
  - 4장은 딥러닝 이전의 고전 기법을 다룸. 명암 또는 컬러만 보고 영역을 결정하기 때문에 의미 분할 불가능

배경이 단순한 영상의 영역 분할

- 단순한 영상의 예
  - 스캔한 책 영상 또는 컨베이어 벨트 위를 흐르는 물체의 영상
- 영역 분할 알고리즘
  - 이진화 알고리즘 (여러 임계값을 사용하는 오츄 알고리즘, 군집화 알고리즘 등) 적용
  - 워터셰드 알고리즘 적용



(a) 에지 강도 맵 (b) 지형으로 간주 (c) 워터셰드

그림 4-14 워터셰드 분할 알고리즘[Cousty2007]

슈퍼 화소 분할

- 슈퍼 화소는 화소보다 크지만 물체보다 작은 자잘한 영역으로 과잉 분할
- SLIC$_{\text{simple linear iterative clustering}}$ 알고리즘
  - k-평균 군집화와 비슷하게 동작 (화소 할당 단계와 군집 중심 갱신하는 단계를 반복)

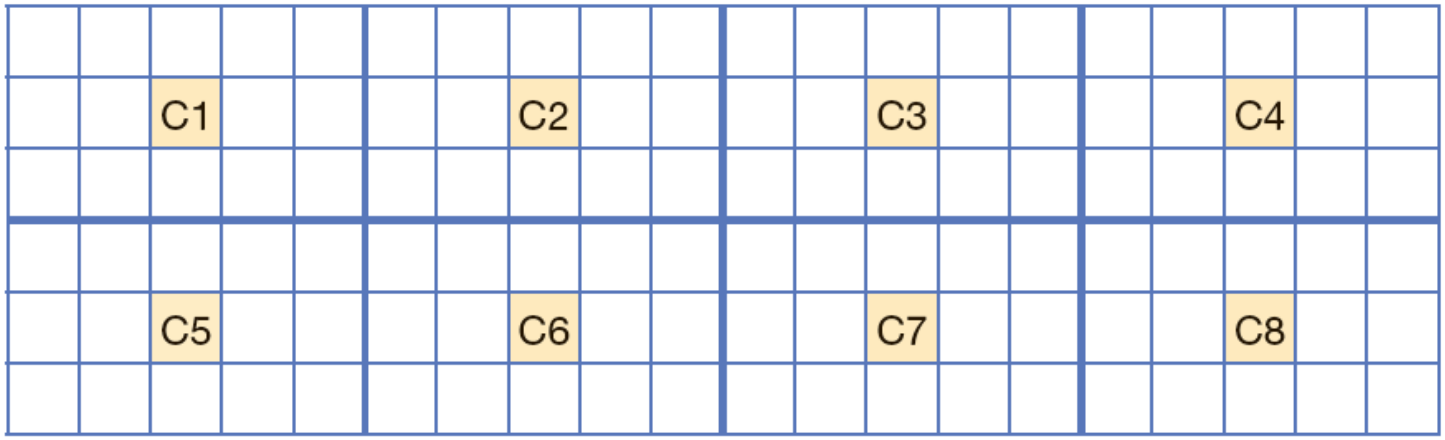


그림 4-15 SLIC 알고리즘의 초기 군집 중심

슈퍼 화소 분할

- 사이킷-이미지 (scikit-image)
  - Python 프로그래밍 언어를 위한 오픈 소스 이미지 처리 라이브러리
  - 설치: pip install scikit-image

```
PS C:\Users\naeza> pip install scikit-image
Collecting scikit-image
  Downloading scikit_image-0.25.2-cp313-cp313-win_amd64.whl.metadata (14 kB)
Requirement already satisfied: numpy>=1.24 in c:\users\naeza\appdata\local\programs\python\python313\lib\site-packages (
from scikit-image) (2.2.3)
Collecting scipy>=1.11.4 (from scikit-image)
  Downloading scipy-1.15.2-cp313-cp313-win_amd64.whl.metadata (60 kB)
Requirement already satisfied: networkx>=3.0 in c:\users\naeza\appdata\local\programs\python\python313\lib\site-packages
 (from scikit-image) (3.4.2)
Requirement already satisfied: pillow>=10.1 in c:\users\naeza\appdata\local\programs\python\python313\lib\site-packages
(from scikit-image) (11.1.0)
Collecting imageio!=2.35.0,>=2.33 (from scikit-image)
  Using cached imageio-2.37.0-py3-none-any.whl.metadata (5.2 kB)
Collecting tifffile>=2022.8.12 (from scikit-image)
  Downloading tifffile-2025.3.30-py3-none-any.whl.metadata (32 kB)
Collecting packaging>=21 (from scikit-image)
  Using cached packaging-24.2-py3-none-any.whl.metadata (3.2 kB)
Collecting lazy-loader>=0.4 (from scikit-image)
  Using cached lazy_loader-0.4-py3-none-any.whl.metadata (7.6 kB)
Downloading scikit_image-0.25.2-cp313-cp313-win_amd64.whl (12.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.9/12.9 MB 36.3 MB/s eta 0:00:00
Using cached imageio-2.37.0-py3-none-any.whl (315 kB)
Using cached lazy_loader-0.4-py3-none-any.whl (12 kB)
Using cached packaging-24.2-py3-none-any.whl (65 kB)
Downloading scipy-1.15.2-cp313-cp313-win_amd64.whl (41.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 41.0/41.0 MB 35.0 MB/s eta 0:00:00
Downloading tifffile-2025.3.30-py3-none-any.whl (226 kB)
Installing collected packages: tifffile, scipy, packaging, imageio, lazy-loader, scikit-image
Successfully installed imageio-2.37.0 lazy-loader-0.4 packaging-24.2 scikit-image-0.25.2 scipy-1.15.2 tifffile-2025.3.30
```

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()
```

슈퍼 화소 분할


Coffee Image
Super pixels (compact 20)
Super pixels (compact 40)
compactness=20
compactness=40

최적화 분할

- 지금까지 분할 알고리즘은 지역적 명암 변화만 살피기 때문에 한계
  - 예) 양말의 색이 배경과 비슷하면 양말이 배경 영역에 섞임
- 전역적 정보를 고려하여 문제 해결
  - 지역적으로 색상 변화가 약하지만 전역적으로 유리하다면 물체 경계로 간주
  - 영상을 그래프로 표현하고 최적화 알고리즘으로 분할 문제를 품

최적화 분할

■ 영상의 그래프 표현

- 화소 또는 슈퍼 화소를 노드로 취함
- 두 노드의 유사도를 식 (4.8)로 계산하여 에지에 부여
  - $f(v)$는 v에 해당하는 화소의 색상(r,g,b)와 위치 (x,y)를 결합한 5차원 벡터
  - v가 슈퍼 화소인 경우 화소 평균을 사용

$$\text{거리} \begin{cases} d_{pq} = \left\| f(v_p) - f(v_q) \right\|, \text{ 만일 } v_q \in neighbor(v_p) \\ \infty, \text{ 그렇지 않으면} \end{cases}$$

$$\text{유사도} \begin{cases} s_{pq} = D - d_{pq} \text{ 또는 } \dfrac{1}{e^{d_{pq}}}, \text{ 만일 } v_q \in neighbor(v_p) \\ 0, \text{ 그렇지 않으면} \end{cases} \qquad (4.8)$$

최적화 분할

- 정규화 절단normalized cut 알고리즘
  - cut은 영상을 두 영역으로 분할했을 때 분할의 좋은 정도를 측정해 주는 목적 함수
    - C1과 C2가 클수록 둘 사이에 에지가 많아 cut은 덩달아 커지므로 cut을 사용한 분할 알고리즘은 영역을 자잘하게 분할하는 경향

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

  - ncut은 cut을 정규화하여 영역의 크기에 중립이 되게 해줌

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

  - ncut을 빨리 계산하는 효율적 알고리즘이 개발되어 있음 [Shi2000]

- 에지 검출과 영역 분할을 결합하고 계층적으로 영역을 분할하는 알고리즘 [Arbelaez2011] 이후 별다른 발전 없음

최적화 분할

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()
```

최적화 분할

```
(400, 600, 3) Coffee 영상을 분할하는 데 6.43808341026306 15초 소요
```

- 고전 영역 분할 알고리즘은 색상 정보에만 의존하므로 의미 분할 불가능
  - 사람은 물체의 3차원 모델과 2차원 겉모습 모델$_{\text{appearance model}}$을 동시에 사용하여 의미 분할을 수행함
  - 9장에서는 딥러닝을 이용한 의미 분할을 다룸

대화식 분할

대화식 분할

- 앞에서 다룬 분할 알고리즘은 영상 전체를 여러 개의 영역으로 분할
- 때로 한 물체의 분할에만 관심이 있는 응용이 있음
  - 예) 특정 물체를 오려내고 다른 물체로 대치
  - 사용자가 초기 정보를 입력하면 반자동 분할해주는 여러 알고리즘이 있음

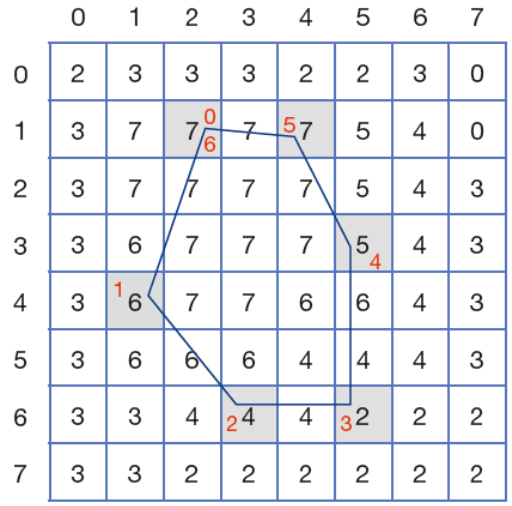
대화식 분할

- 앞에서 다룬 분할 알고리즘은 영상 전체를 여러 개의 영역으로 분할
- 때로 한 물체의 분할에만 관심이 있는 응용이 있음
  - 예) 특정 물체를 오려내고 다른 물체로 대치
  - 사용자가 초기 정보를 입력하면 반자동 분할해주는 여러 알고리즘이 있음

- 능동외곽선
- GrabCut

대화식 분할

- 능동 외곽선active contour
  - 물체 내부에 초기 곡선을 지정하면 곡선을 점점 확장하여 물체 외곽선으로 접근
  - 곡선이 꿈틀대면서 에너지가 최소인 상태를 찾아가기 때문에 스네이크라는 별명
- 곡선 표현 방법



$g(0)=(1,2)$
$g(1)=(4,1)$
$g(2)=(6,3)$
$g(3)=(6,5)$
$g(4)=(3,5)$
$g(5)=(1,4)$
$g(6)=(1,2)$

그림 4-16 디지털 공간에서 스네이크 곡선 표현

대화식 분할

■ 식 (4.11)은 스네이크 곡선의 에너지를 표현

$$E(g)=\sum_{l=0}^{n}\left(e_{image}\left(g(l)\right)+e_{internal}\left(g(l)\right)+e_{domain}\left(g(l)\right)\right) \qquad (4.11)$$

■ 영역 분할을 최소 에너지를 갖는 곡선을 찾는 문제로 공식화

$$\hat{g}=\underset{g}{\operatorname{argmin}}\,E(g) \qquad (4.12)$$

■ 사용자가 지정한 초기 곡선 g_0에서 시작하여 $g_1, g_2, \cdots$로 발전해 감

대화식 분할

[알고리즘 4-1] 스네이크로 물체 분할

입력: 명암 영상, 임곗값 T

출력: 최적 곡선 $\hat{g}$

1. 사용자 입력을 받아 초기 곡선 g를 설정한다.
2. while TRUE
3. $moved$=0
4. for i=0 to n−1
5. for $g(i)$의 9개 이웃점 각각에 대해 // 자신과 8-이웃을 포함한 9개 점
6. $g(i)$를 이웃점으로 이동한 곡선의 에너지 E를 식 (4.11)로 구한다.
7. if 에너지가 최소인 점이 $g(i)$와 다르면
8. $g(i)$를 최소점으로 이동하고 $moved$를 1 증가시킨다.
9. if $moved < T$ // 곡선의 이동량이 임계치보다 작으면 수렴했다고 간주하고 탈출
10. break

GrabCut

- GrabCut
  - 사용자가 붓으로 물체와 배경을 초기 지정
  - 붓 칠된 화소를 가지고 물체 히스토그램과 배경 히스토그램을 만듦
  - 나머지 화소들은 두 히스토그램과 유사성을 따져 물체일 확률과 배경일 확률을 추정
  - 이 확률 정보를 이용하여 물체 영역과 배경 영역을 갱신
  - 이 과정을 반복하다가 변화가 거의 없으면 수렴 간주하고 멈춤

GrabCut

프로그램 4-7 GrabCut을 이용해 물체 분할하기

```
01  import cv2 as cv
02  import numpy as np
03
04  img=cv.imread('soccer.jpg')                    # 영상 읽기
05  img_show=np.copy(img)                          # 붓칠을 디스플레이할 목적의 영상
06
07  mask=np.zeros((img.shape[0],img.shape[1]),np.uint8)
08  mask[:,:]=cv.GC_PR_BGD                         # 모든 화소를 배경일 것 같음으로 초기화
09
10  BrushSiz=9                                     # 붓의 크기
11  LColor,RColor=(255,0,0),(0,0,255)              # 파란색(물체)과 빨간색(배경)
12
13  def painting(event,x,y,flags,param):
14      if event==cv.EVENT_LBUTTONDOWN:
15          cv.circle(img_show,(x,y),BrushSiz,LColor,-1) # 왼쪽 버튼 클릭하면 파란색
16          cv.circle(mask,(x,y),BrushSiz,cv.GC_FGD,-1)
17      elif event==cv.EVENT_RBUTTONDOWN:
18          cv.circle(img_show,(x,y),BrushSiz,RColor,-1) # 오른쪽 버튼 클릭하면 빨간색
19          cv.circle(mask,(x,y),BrushSiz,cv.GC_BGD,-1)
20      elif event==cv.EVENT_MOUSEMOVE and flags==cv.EVENT_FLAG_LBUTTON:
21          cv.circle(img_show,(x,y),BrushSiz,LColor,-1)
                                                   # 왼쪽 버튼 클릭하고 이동하면 파란색
22          cv.circle(mask,(x,y),BrushSiz,cv.GC_FGD,-1)
23      elif event==cv.EVENT_MOUSEMOVE and flags==cv.EVENT_FLAG_RBUTTON:
24          cv.circle(img_show,(x,y),BrushSiz,RColor,-1)
                                                   # 오른쪽 버튼 클릭하고 이동하면 빨간색
25          cv.circle(mask,(x,y),BrushSiz,cv.GC_BGD,-1)
26
27      cv.imshow('Painting',img_show)
```

cv.GC_FGD: 확실히 물체
cv.GC_BGD: 확실히 배경
cv.GC_PR_FGD: 물체일 것 같음
cv.GC_PR_BGD: 배경일 것 같음

GrabCut

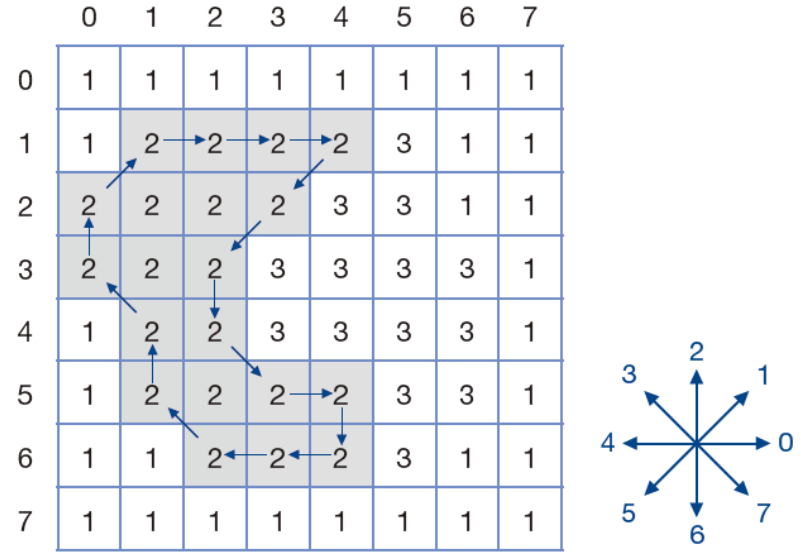
```
28
29  cv.namedWindow('Painting')
30  cv.setMouseCallback('Painting',painting)
31
32  while(True):                                # 붓칠을 끝내려면 'q' 키를 누름
33      if cv.waitKey(1)==ord('q'):
34          break
35
36  # 여기부터 GrabCut 적용하는 코드
37  background=np.zeros((1,65),np.float64)   # 배경 히스토그램 0으로 초기화
38  foreground=np.zeros((1,65),np.float64)   # 물체 히스토그램 0으로 초기화
39
40  cv.grabCut(img,mask,None,background,foreground,5,cv.GC_INIT_WITH_MASK)
41  mask2=np.where((mask==cv.GC_BGD)|(mask==cv.GC_PR_BGD),0,1).astype('uint8')
42  grab=img*mask2[:,:,np.newaxis]
43  cv.imshow('Grab cut image',grab)
44
45  cv.waitKey()
46  cv.destroyAllWindows()
```

GrabCut

영역 특징

영역 특징

- 영역의 레이블링과 기하 변환 예시



(a) 영역의 레이블링

(b) 영역의 기하 변환

그림 4-17 영역의 레이블링과 기하 변환

영역 특징

- 특징의 불변성과 등변성
  - 변환을 해도 값이 변하지 않으면 불변성$_{invariant}$이 있는 특징
    - 예) 성별이라는 특징은 나이에 불변. 근력은 나이에 불변이 아님
    - 예) 면적은 회전에 불변이지만 축소에는 불변이 아님. 주축은 회전에 불변이 아니지만 축소에는 불변
  - 변환에 따라 값이 따라 변하면 등변성$_{equivariant}$이 있는 특징
    - 예) 면적은 축소에 등변이지만 회전에는 등변이 아님
  - 명암 변화에 둔감한 광도 불변성도 중요

- 과업에 따라 특징을 선택하는 일이 중요
  - 예) 로봇이 물체를 인식한 다음 물체를 집어야 한다면 회전에 등변인 특징을 사용해야 함

영역 특징

- 모멘트와 모양 특징
  - 영역 R의 모멘트

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

  - 모멘트를 이용한 다양한 특징

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

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

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

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

영역 특징

- 영역의 둘레와 둥근 정도

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

- 주축의 방향

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

영역 특징

- 텍스처 특징. 텍스처는 일정한 패턴의 반복
  - 에지 통계량으로 텍스처 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 화소의 명암 값을 비교해서 텍스처 측정. 256차원 특징 벡터
    - LTP는 작은 명암 변화에 민감한 LBP 단점 개선. 512차원 특징 벡터

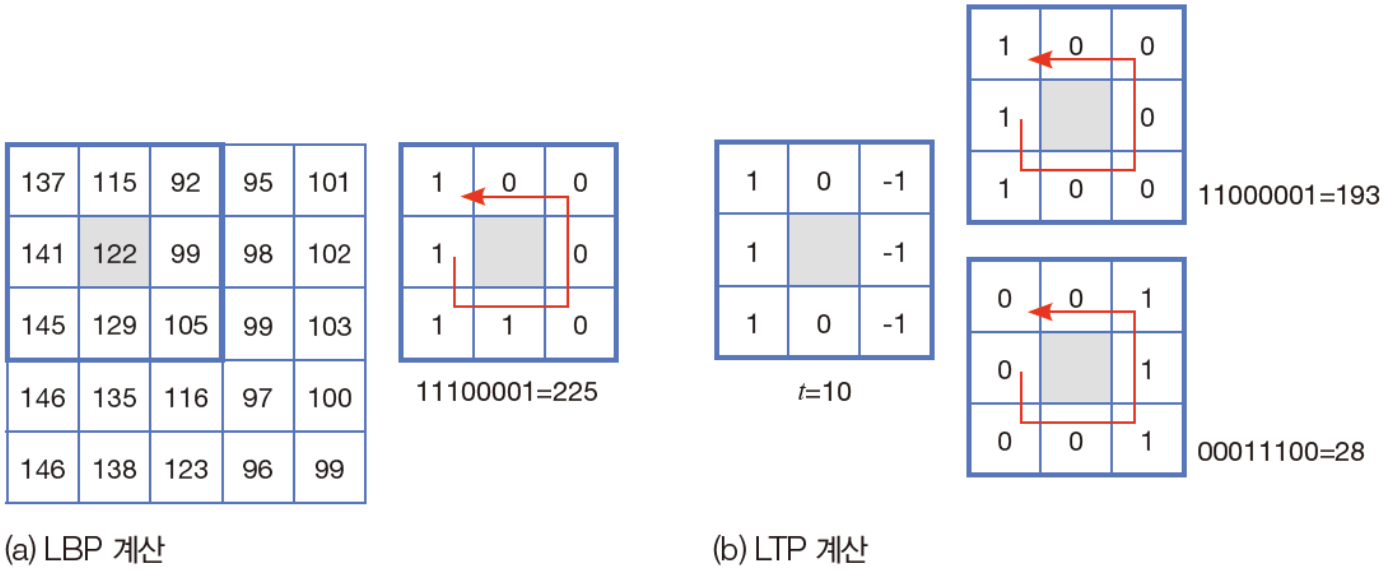


그림 4-18 LBP와 LTP 구하기

영역 특징

- 직선 근사 알고리즘

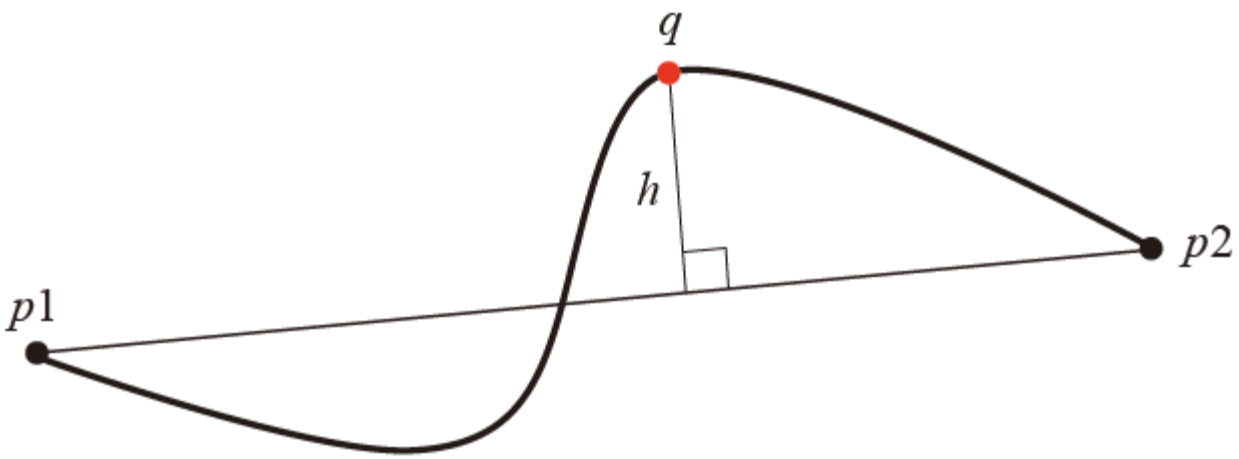


그림 4-19 직선 근사 알고리즘

영역 특징

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
```

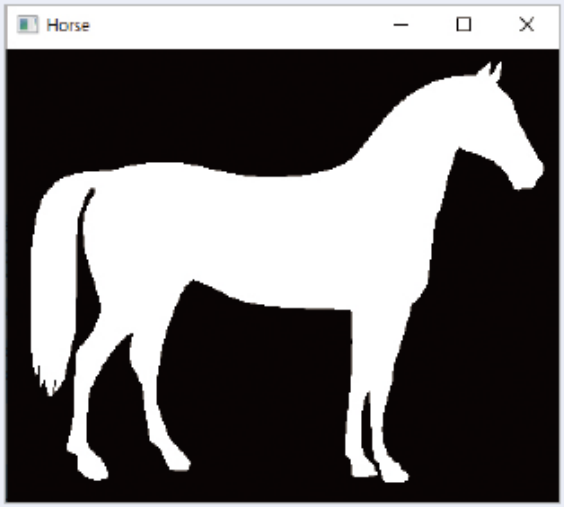
영역 특징

```
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()
```

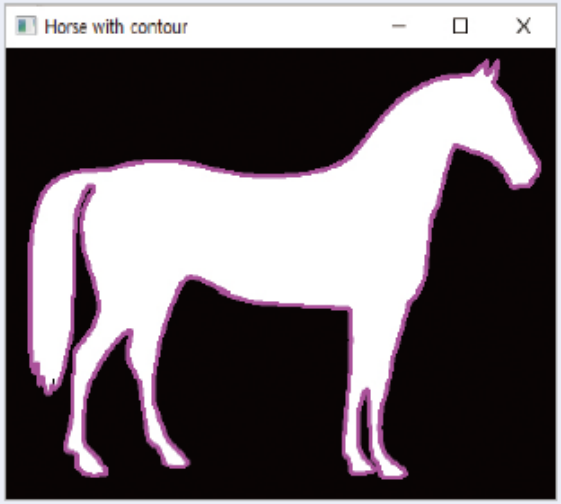
영역 특징

```
면적= 42390.0 ③
중점=( 187.72464024534088 , 144.43640402610677 )
둘레= 2296.7291333675385
둥근 정도= 0.10098426803214 35
```

①



②



④