패턴 인식

영상 처리

차례

Preview

■ 영상 처리

- 특정 목적을 달성하기 위해 원래 영상을 개선된 새로운 영상으로 변환하는 작

(a) 안개 낀 도로 영상　　(b) 히스토그램 평활화로 개선한 영상

그림 3-1 영상 처리로 화질 개선

■ 화질 개선 자체가 목적인 경우

- 예) 도주 차량의 번호판 식별. 병변 위치 찾기 등

■ 컴퓨터 비전은 전처리로 활용하여 인식 성능을 향상

3.1 디지털 영상 기초

- **현대는 인터넷에 수많은 영상이 쌓임**
  - 컴퓨터 비전 알고리즘을 개발하는데 중요한 실험 데이터로 활용됨

3.1.1 영상 획득과 표현

■ 핀홀 카메라 모델

- 영상 획득 과정은 매우 복잡
- 핀홀 카메라 모델은 핵심을 설명

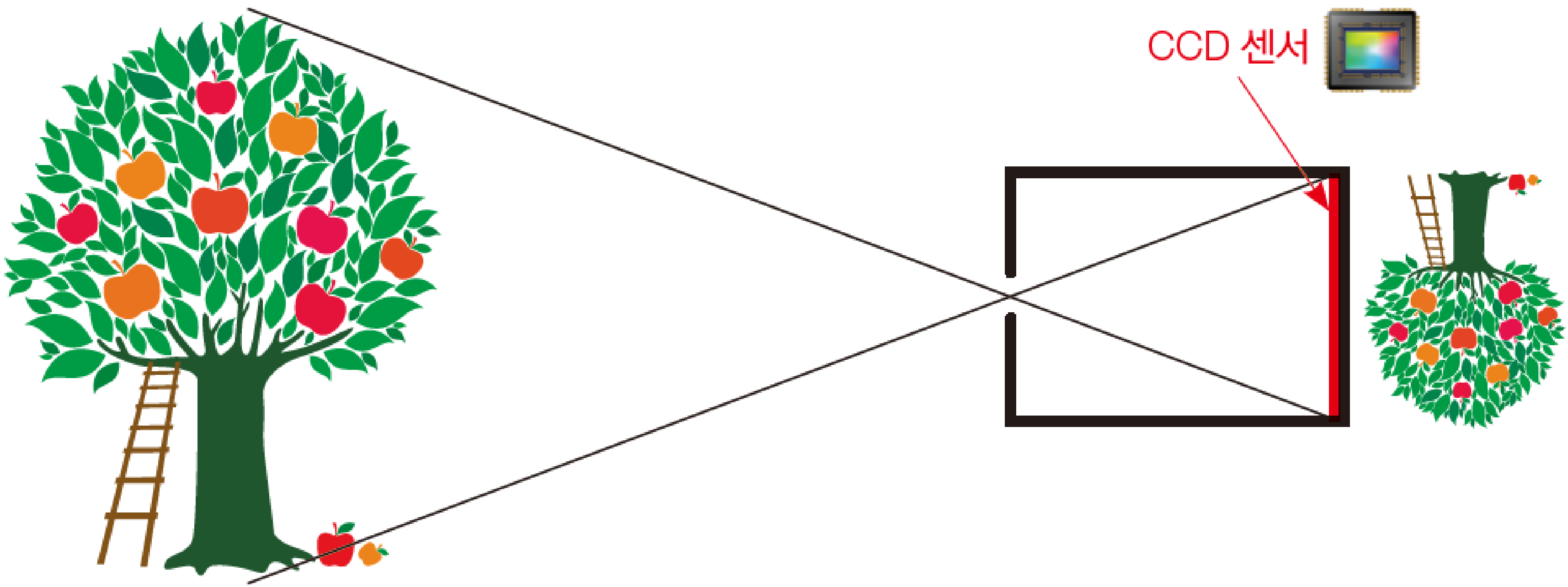
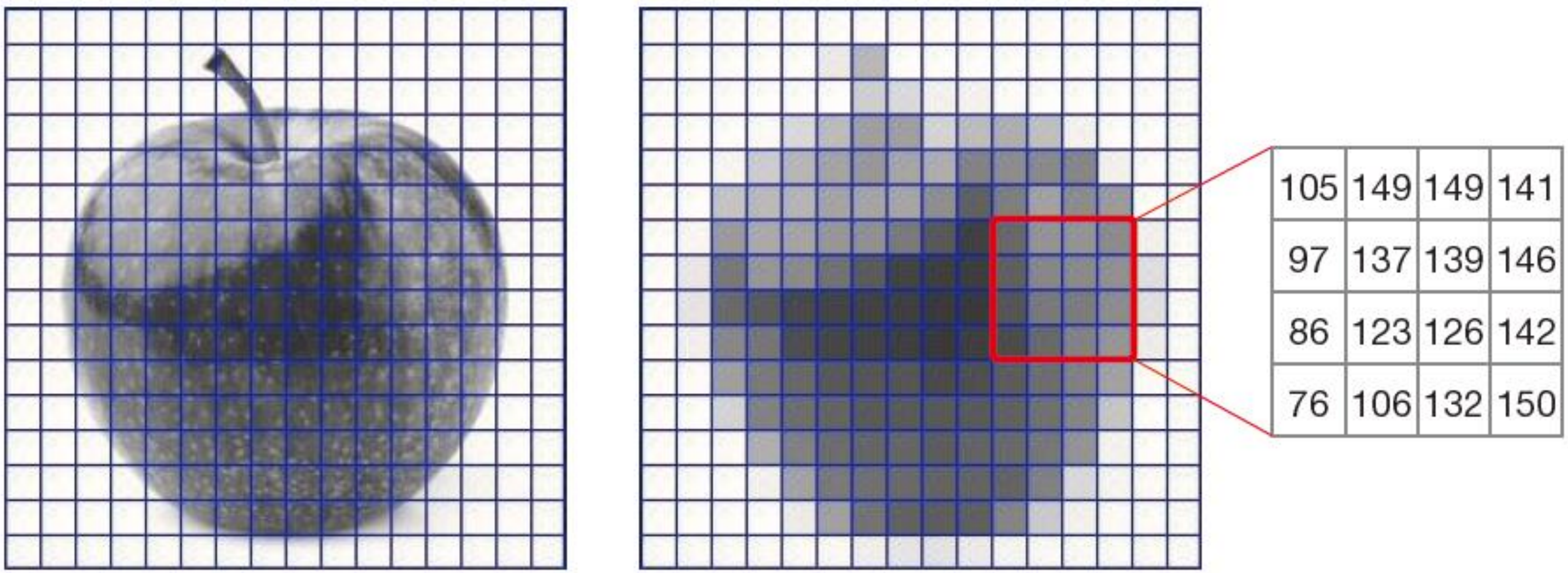


그림 3-2 핀홀 카메라 모델과 CCD 센서

3.1.1 영상 획득과 표현

■ 디지털 변환

- M*N 영상으로 샘플링sampling
- L 단계로 양자화quantization



(a) 샘플링 (b) 양자화

그림 3-3 피사체가 반사하는 빛 신호를 샘플링과 양자화를 통해 디지털 영상으로 변환

TIP 엄밀히 말해 해상도는 물리적 단위 공간에서 식별 가능한 점의 개수를 뜻한다. 예를 들어 인치 당 점의 개수를 뜻하는 dpi(dot per inch)는 해상도다. 이 책에서는 화소의 개수를 해상도라고 부른다.

3.1.1 영상 획득과 표현

■ 영상 좌표계

- 왼쪽 위 구석이 원점
- (y,x) 표기

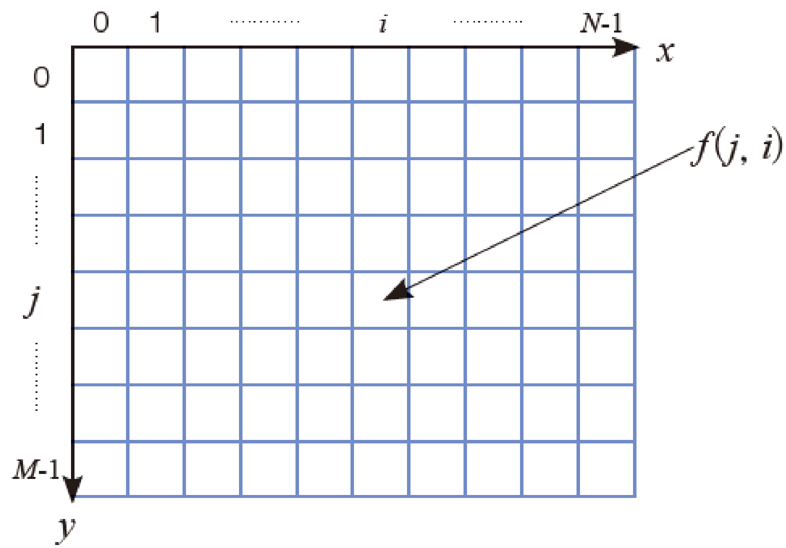


그림 3-4 디지털 영상의 좌표계

- 함수에 따라 (x,y) 표기 사용하니 주의할 필요. 예) cv.line 함수

```
cv.line(img,(10,20),(100,20),….)
```

■ OpenCV는 numpy.ndarray로 영상 표현

- numpy.ndarray가 지원하는 다양한 함수를 사용할 수 있다는 큰 장점
- 예) min, max, argmin, argmax, mean, sort, reshape, transpose, … …

3.1.2 다양한 종류의 영상

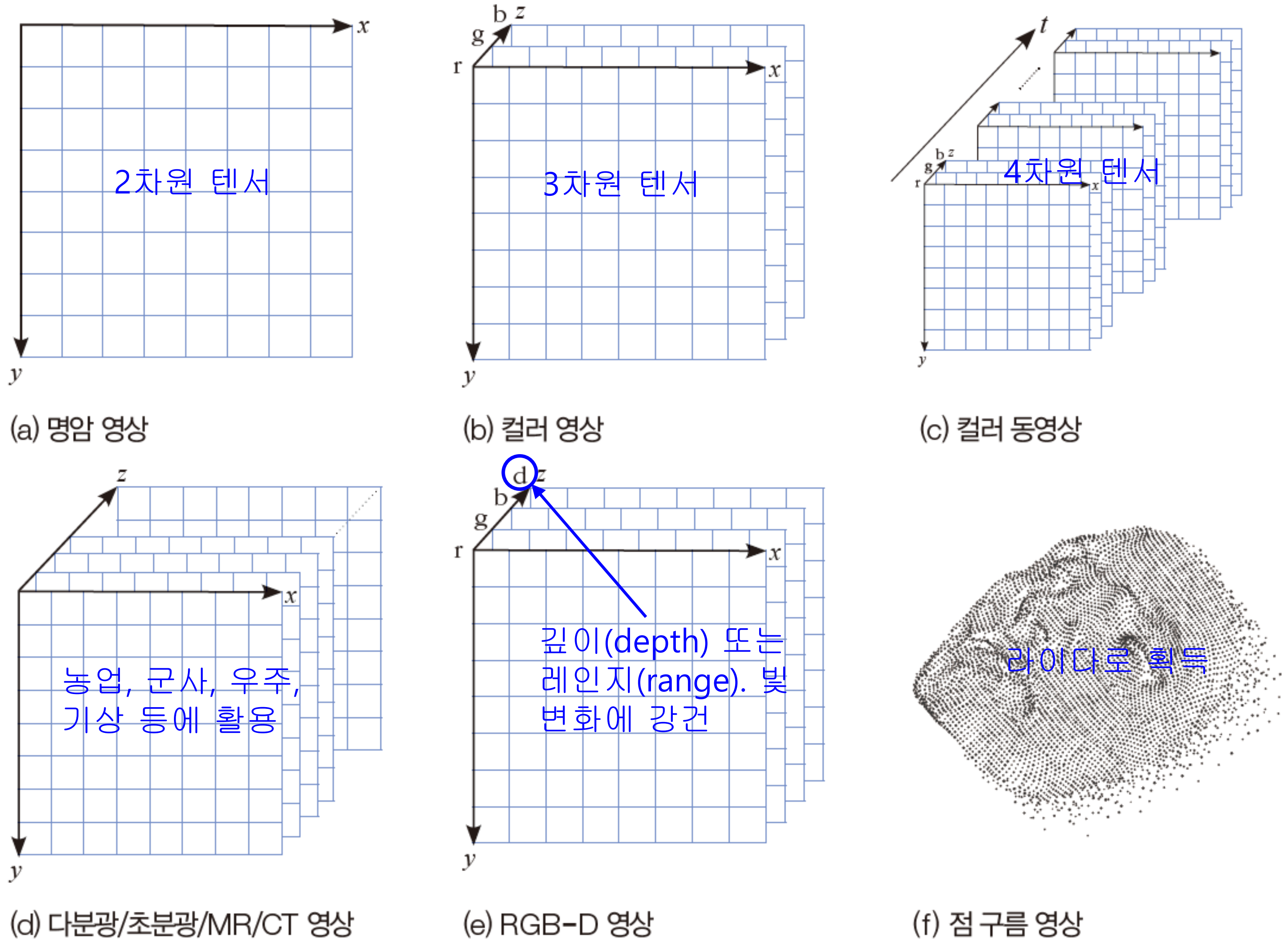


그림 3-5 다양한 형태의 디지털 영상

3.1.3 컬러 모델

■ RGB 컬러 모델

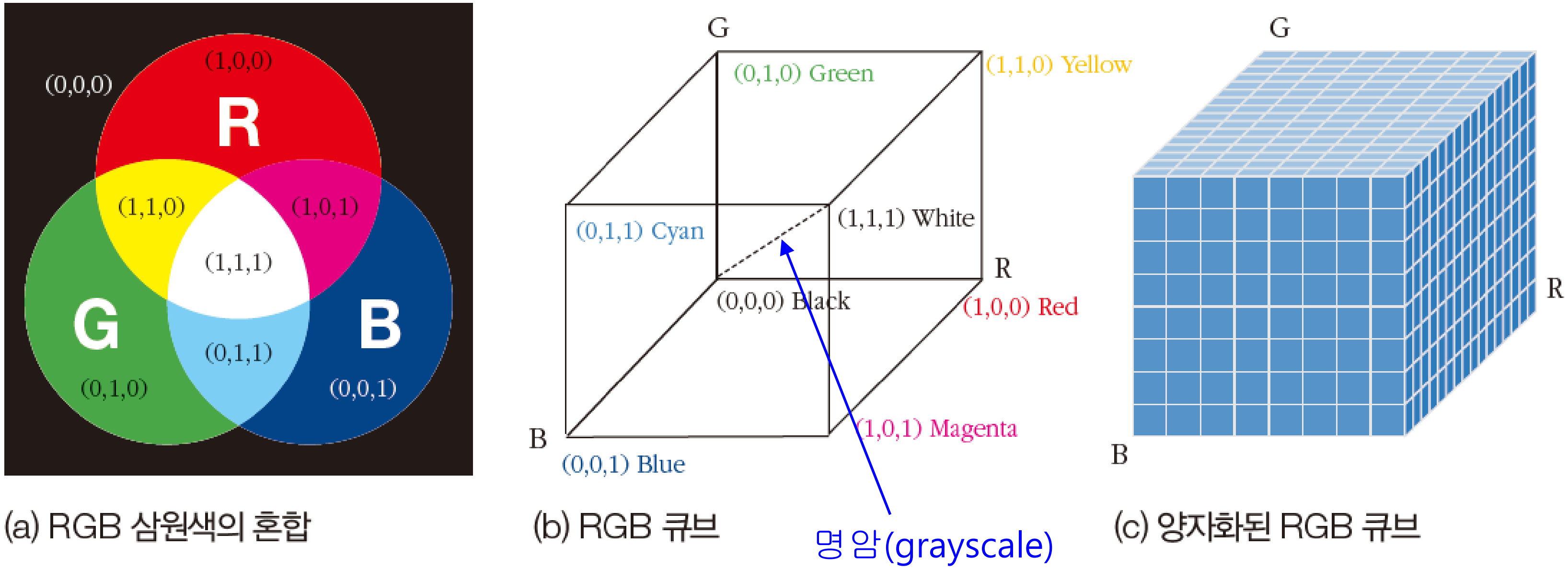


그림 3-6 RGB 컬러 공간

3.1.3 컬러 모델

■ HSV 컬러 모델

- 빛의 밝기가 V 요소에 집중
- RGB보다 빛 변환에 강건

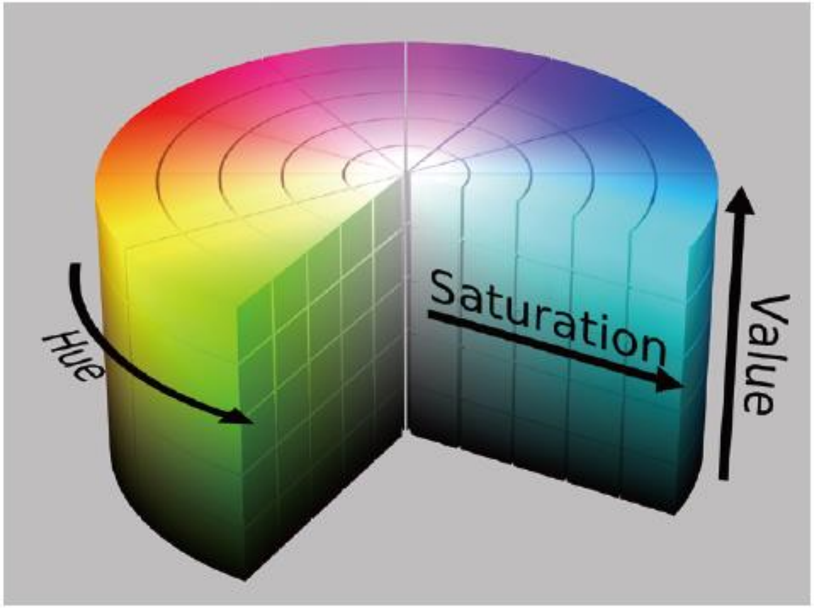
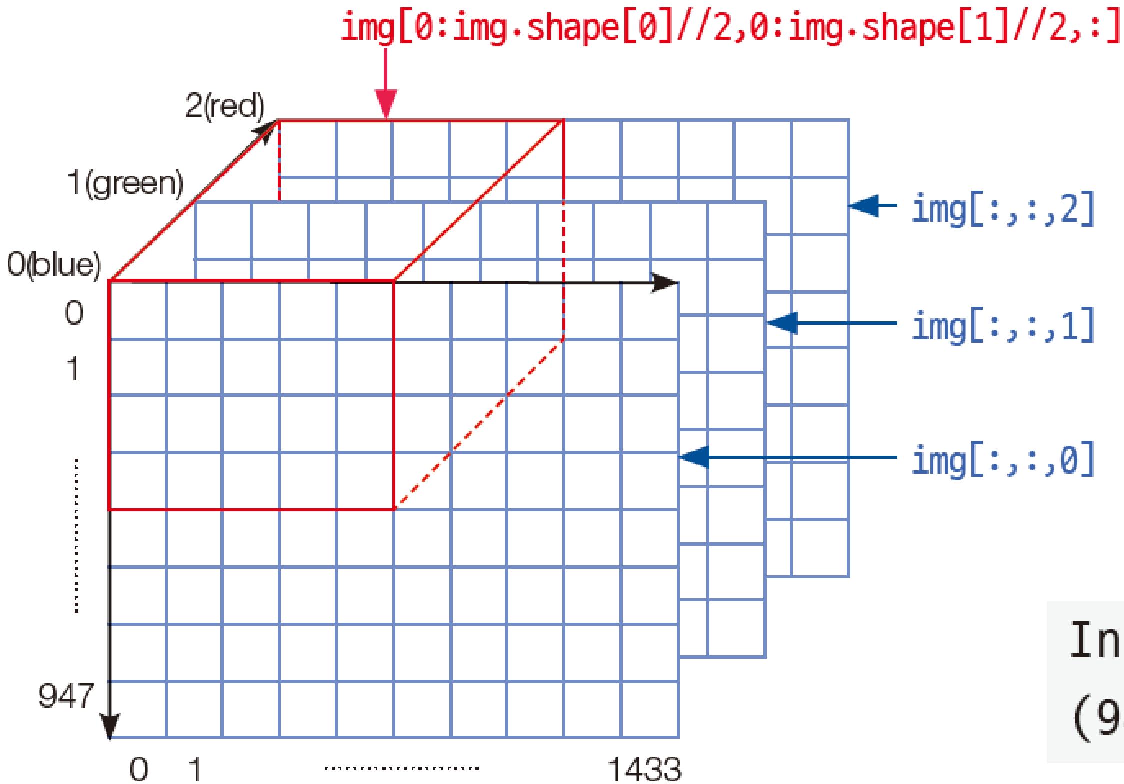


그림 3-7 HSV 컬러 모델

3.1.4 RGB 채널별로 디스플레이

■ numpy의 슬라이싱 기능을 이용하여 RGB 채널별로 디스플레이




```
In [0]: img.shape
(948,1434,3)
```

그림 3-8 numpy.ndarray의 슬라이싱을 이용한 영상 일부분 자르기([프로그램 3-1]의 10행)

TIP 온라인 부록 A에서 이 책을 공부하는 데 필요한 최소한의 numpy 지식을 제공한다.

3.1.4 프로그래밍 실습: RGB 채널별로 디스플레이

프로그램 3-1	RGB 컬러 영상을 채널별로 구분해 디스플레이하기

```python
import cv2 as cv
import sys

img=cv.imread('soccer.jpg')

if img is None:
    sys.exit('파일을 찾을 수 없습니다.')

cv.imshow('original_RGB',img)
cv.imshow('Upper left half',img[0:img.shape[0]//2,0:img.shape[1]//2,:])
cv.imshow('Center half',img[img.shape[0]//4:3*img.shape[0]//4,img.shape[1]//4:3*img.shape[1]//4,:])

cv.imshow('R channel',img[:,:,2])
cv.imshow('G channel',img[:,:,1])
cv.imshow('B channel',img[:,:,0])

cv.waitKey()
cv.destroyAllWindows()
```

3.1.4 프로그래밍 실습: RGB 채널별로 디스플레이

3.2 이진 영상

■ 이진 영상

- 화소가 0(흑) 또는 1(백)인 영상
- 1비트면 저장할 수 있는데, 편의상 1바이트 사용하는 경우 많음
- 에지 검출 결과를 표시하거나 물체와 배경을 구분하여 표시하는 응용 등에 사용

3.2.1 이진화

■ 알고리즘

- 임계값 T보다 큰 화소는 1, 그렇지 않은 화소는 0으로 바꿈. 임계값 결정이 중요

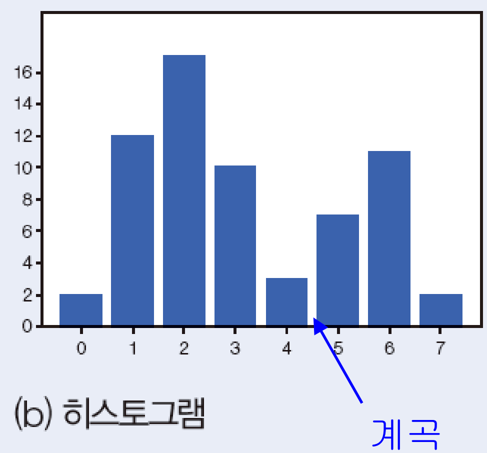
$$b(j,i)=\begin{cases}1, f(j,i)\geq T \\ 0, f(j,i)<T\end{cases} \qquad (3.1)$$

- 히스토그램에서 계곡 부근으로 결정하는 방법([예시 3-1])

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

(a) 입력 영상

$h=$	2	12	17	10	3	7	11	2



(b) 히스토그램

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

(c) 이진 영상

그림 3-9 히스토그램을 이용한 이진화

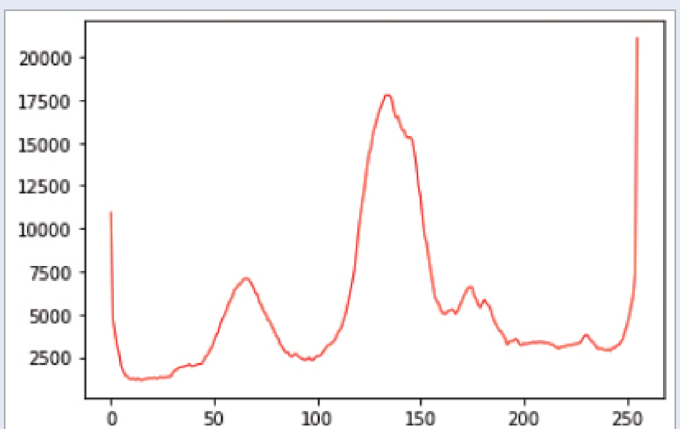
3.2.1 이진화

■ 알고리즘 (계속.....)

- 실제 영상에서는 계곡이 아주 많이 나타나서 구현이 쉽지 않음

프로그램 3-2 실제 영상에서 히스토그램 구하기

```python
import cv2 as cv
import matplotlib.pyplot as plt

img=cv.imread('soccer.jpg')
h=cv.calcHist([img],[2],None,[256],[0,256]) # 2번 채널인 R 채널에서 히스토그램 구함
plt.plot(h,color='r',linewidth=1)
```



3.2.2 오츄 알고리즘

■ 오츄 알고리즘

- 이진화를 최적화 문제로 바라봄. 최적값 $\hat{t}$을 임계값 T로 이용

$$\hat{t} = \underset{t \in \{0,1,2,\cdots,L-1\}}{\operatorname{argmin}} J(t) \qquad (3.2)$$

- 목적 함수 $J(t)$는 임계값 t의 좋은 정도를 측정함(작을수록 좋음)
    - t로 이진화했을 때 0이 되는 화소들의 분산($v_0(t)$)과 1이 되는 화소들의 분산($v_1(t)$)의 가중치($n_0(t)$와 $n_1(t)$) 합을 J로 사용

$$J(t) = n_0(t)v_0(t) + n_1(t)v_1(t) \qquad (3.3)$$

3.2.2 오츄 알고리즘

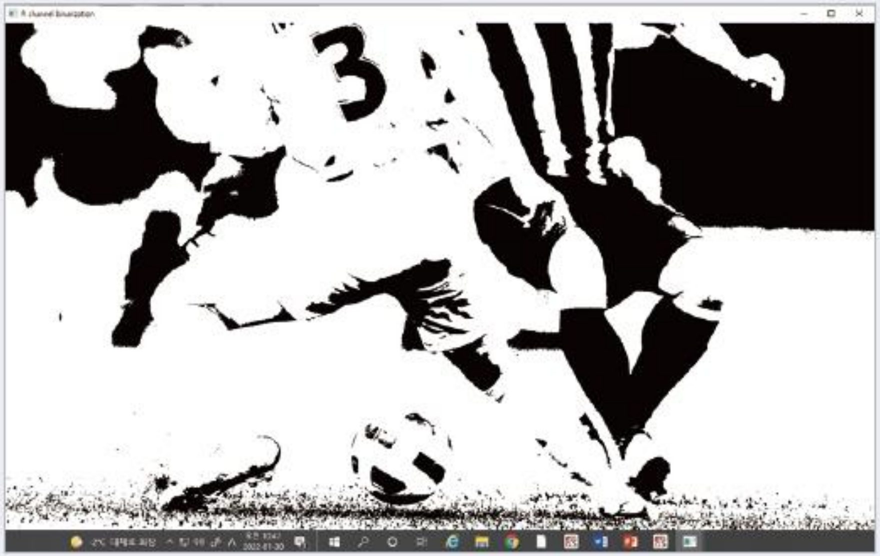
■ 프로그래밍 실습

프로그램 3-3 오츄 알고리즘으로 이진화하기

```python
import cv2 as cv
import sys

img=cv.imread('soccer.jpg')

t,bin_img=cv.threshold(img[:,:,2],0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
print('오츄 알고리즘이 찾은 최적 임곗값=',t) ①

cv.imshow('R channel',img[:,:,2])                      # R 채널 영상
cv.imshow('R channel binarization',bin_img)     # R 채널 이진화 영상

cv.waitKey()
cv.destroyAllWindows()
```

3.2.2 오츄 알고리즘

오츄 알고리즘이 찾은 최적 임겟값= 113.0 ①



3.2.3 연결 요소

■ 4-연결성과 8-연결성

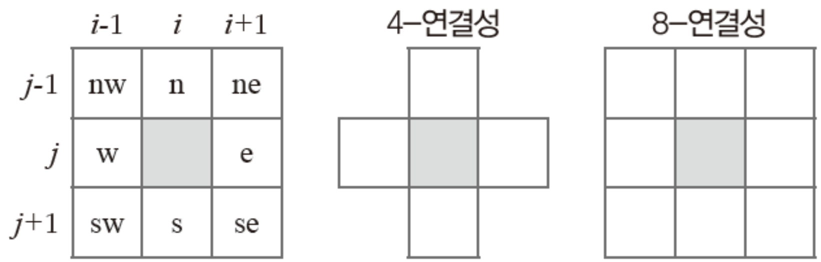


그림 3–10 화소의 연결성

■ 연결 요소

[예시 3–2] 연결 요소

[그림 3–11]에서 (a)는 입력 이진 영상이고, (b)와 (c)는 각각 4–연결성과 8–연결성으로 찾은 연결 요소다. 연결 요소는 고유한 정수 번호로 구분한다.

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

(a) 입력 이진 영상

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

(b) 4–연결성으로 찾은 연결 요소

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

(c) 8–연결성으로 찾은 연결 요소

그림 3–11 연결 요소 찾기

3.2.4 모폴로지

- 모폴로지는 구조 요소$_{\text{structuring element}}$를 이용하여 영역의 모양을 조작

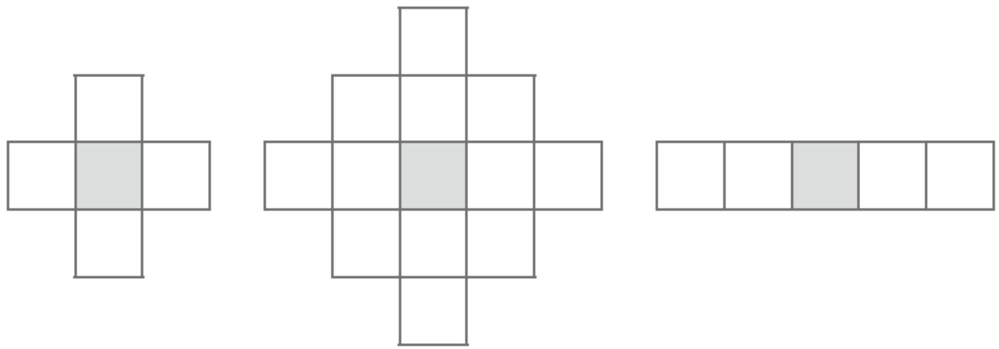


그림 3-12 모폴로지가 사용하는 구조 요소

- 팽창$_{\text{dilation}}$, 침식$_{\text{erosion}}$, 열림$_{\text{opening}}$, 닫힘$_{\text{closing}}$
  - 팽창은 작은 홈을 메우거나 끊어진 영역을 연결하는 효과. 영역을 키움
  - 침식은 경계에 솟은 돌출 부분을 깎는 효과. 영역을 작게 만듦
  - 열림은 침식한 결과에 팽창 적용. 원래 영역 크기 유지
  - 닫힘은 팽창한 결과에 침식을 적용. 원래 영역 크기 유지

3.2.4 모폴로지

■ [예시 3-3] 팽창과 침식 연산

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

구조 요소

(a) 입력 영상과 구조 요소

0	0	0	0	0	0	0	0
0	0	0	P 1	0	0	0	0
0	1	1	1	1	1	0	0
0	1	1	0	1	1	1	0

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

(b) 팽창

0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	P 1	1	1	1	1	0	0
0	1	1	0	1	1	1	0

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

(c) 침식

그림 3-13 팽창과 침식

3.2.4 모폴로지

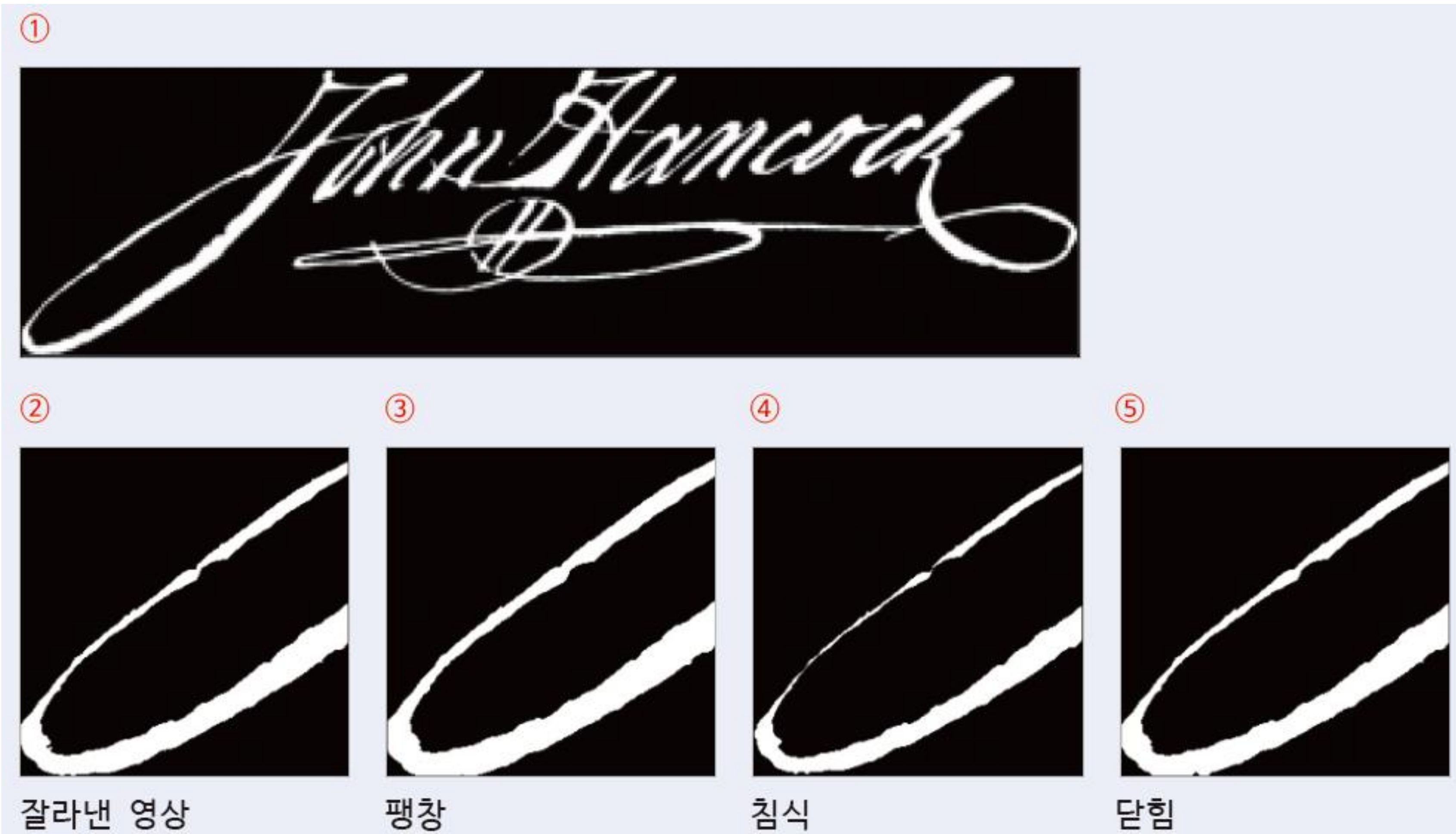
프로그램 3-4 모폴로지 연산 적용하기

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

img=cv.imread('JohnHancocksSignature.png',cv.IMREAD_UNCHANGED)

t,bin_img=cv.threshold(img[:,:,3],0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
plt.imshow(bin_img,cmap='gray'), plt.xticks([]), plt.yticks([]) ①
plt.show()
```

3.2.4 모폴로지

```
b=bin_img[bin_img.shape[0]//2:bin_img.shape[0],0:bin_img.shape[0]//2+1]
plt.imshow(b,cmap='gray'), plt.xticks([]), plt.yticks([]) ②
plt.show()

se=np.uint8([[0,0,1,0,0],                                # 구조 요소
             [0,1,1,1,0],
             [1,1,1,1,1],
             [0,1,1,1,0],
             [0,0,1,0,0]])

b_dilation=cv.dilate(b,se,iterations=1)                  # 팽창
plt.imshow(b_dilation,cmap='gray'), plt.xticks([]), plt.yticks([]) ③
plt.show()

b_erosion=cv.erode(b,se,iterations=1)                    # 침식
plt.imshow(b_erosion,cmap='gray'), plt.xticks([]), plt.yticks([]) ④
plt.show()

b_closing=cv.erode(cv.dilate(b,se,iterations=1),se,iterations=1) # 닫기
plt.imshow(b_closing,cmap='gray'), plt.xticks([]), plt.yticks([]) ⑤
plt.show()
```

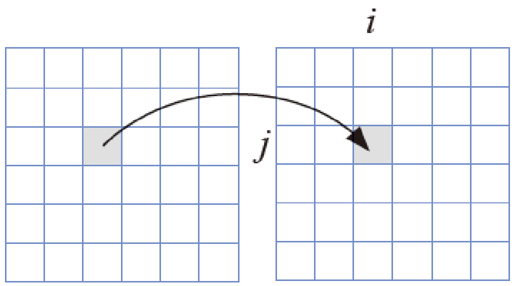
3.2.4 모폴로지



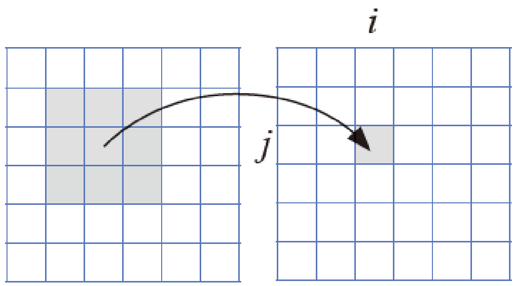
3.3 점 연산

■ 세 종류의 영상 처리 연산

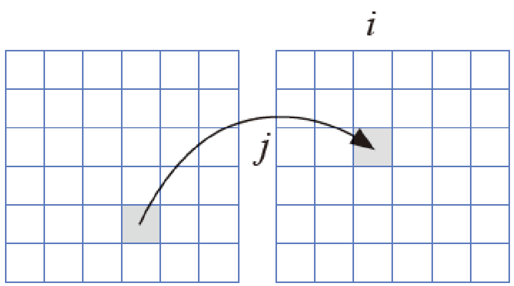
- 화소가 새로운 값을 어디서 받느냐에 따라 세 가지로 구분
- 점 연산$_{\text{point operation}}$: 자기 자신에게서 받음(3.3절)
- 영역 연산$_{\text{area operation}}$: 이웃 화소들에서 받음(3.4절)
- 기하 연산$_{\text{geometric operation}}$: 기하학적 변환이 정해주는 곳에서 받음(3.5절)



(a) 점 연산



(b) 영역 연산



(c) 기하 연산

그림 3-14 세 종류의 영상 처리 연산

3.3.1 명암 조절

■ 영상을 밝거나 어둡게 조정

- 선형 연산

$$f'(j,i)=\begin{cases}\min\left(f(j,i)+a,L-1\right) & \text{밝게}\\ \max\left(f(j,i)-a,0\right) & \text{어둡게}\\ (L-1)-f(j,i) & \text{반전}\end{cases} \qquad (3.4)$$

- 비선형 연산: 예) 감마 연산

$$f'(j,i)=(L-1)\times \dot{f}(j,i)^{\gamma} \qquad (3.5)$$

3.3.1 명암 조절

■ 프로그래밍 실습: 감마 보정

프로그램 3-5 감마 보정 실험하기

```
import cv2 as cv
import numpy as np

img=cv.imread('soccer.jpg')
img=cv.resize(img,dsize=(0,0),fx=0.25,fy=0.25)

def gamma(f,gamma=1.0):
    f1=f/255.0                              # L=256이라고 가정
    return np.uint8(255*(f1**gamma))

gc=np.hstack((gamma(img,0.5),gamma(img,0.75),gamma(img,1.0),gamma(img,2.0),gamma(img,3.0)))
cv.imshow('gamma',gc)

cv.waitKey()
cv.destroyAllWindows()
```

numpy.float64 형

numpy.uint8 형으로 변환

numpy.hstack 함수로 이어붙이기

gamma=0.5 gamma=0.75 gamma=1.0 gamma=2.0 gamma=3.0

3.3.2 히스토그램 평활화

- **히스토그램 평활화**histogram equalization
  - 히스토그램이 평평하게 되도록 영상을 조작해 영상의 명암 대비를 높이는 기법

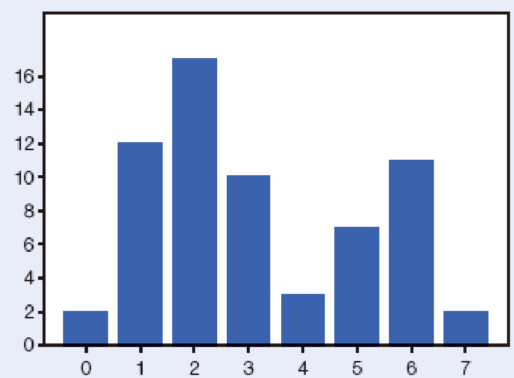
$$l' = \text{round}\left(\ddot{h}(l) \times (L-1)\right) \qquad (3.6)$$

- **[예시 3-4] ([그림 3-9]를 재활용)**

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

(a) 입력 영상

$h=$	2	12	17	10	3	7	11	2



(b) 히스토그램

[그림 3-9]

3.3.2 히스토그램 평활화

l	h	$\dot{h}$	$\ddot{h}$	$\ddot{h}\times 7$	l'
0	2	0.03125	0.03125	0.21875	0
1	12	0.1875	0.21875	1.53125	2
2	17	0.265625	0.484375	3.390625	3
3	10	0.15625	0.640625	4.484375	4
4	3	0.046875	0.6875	4.8125	5
5	7	0.109375	0.796875	5.578125	6
6	11	0.171875	0.96875	6.78125	7
7	2	0.03125	1.0	7.0	7

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

$h=$	2	0	12	17	10	3	7	13

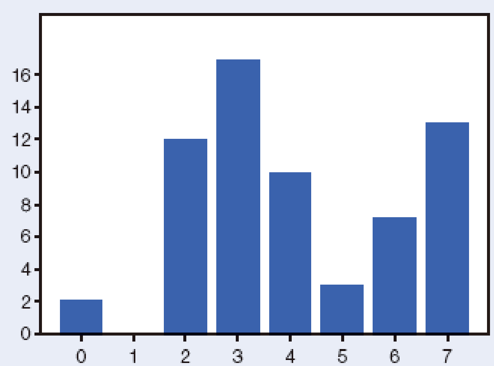


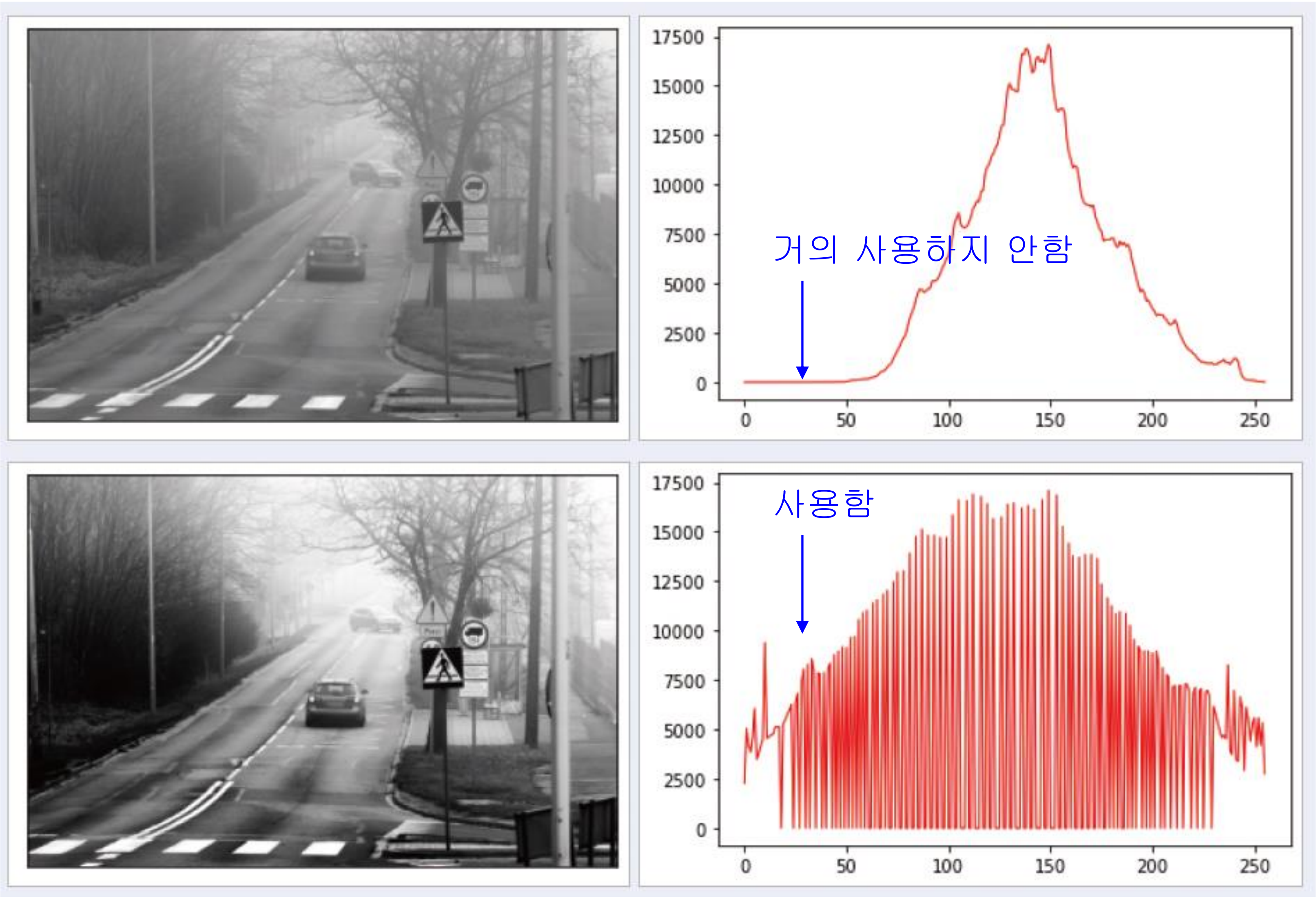
그림 3-15 히스토그램 평활화된 영상

3.3.2 히스토그램 평활화

프로그램 3-6 히스토그램 평활화하기

```python
import cv2 as cv
import matplotlib.pyplot as plt

img=cv.imread('mistyroad.jpg')

gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)        # 명암 영상으로 변환하고 출력
plt.imshow(gray,cmap='gray'), plt.xticks([]), plt.yticks([]), plt.show()

h=cv.calcHist([gray],[0],None,[256],[0,256])     # 히스토그램을 구해 출력
plt.plot(h,color='r',linewidth=1), plt.show()

equal=cv.equalizeHist(gray)                      # 히스토그램을 평활화하고 출력
plt.imshow(equal,cmap='gray'), plt.xticks([]), plt.yticks([]), plt.show()

h=cv.calcHist([equal],[0],None,[256],[0,256])    # 히스토그램을 구해 출력
plt.plot(h,color='r',linewidth=1), plt.show()
```

3.3.2 히스토그램 평활화



3.4 영역 연산

- 영역 연산은 이웃 화소를 고려해서 새로운 값을 결정
- 주로 컨볼루션 연산을 통해 이루어짐

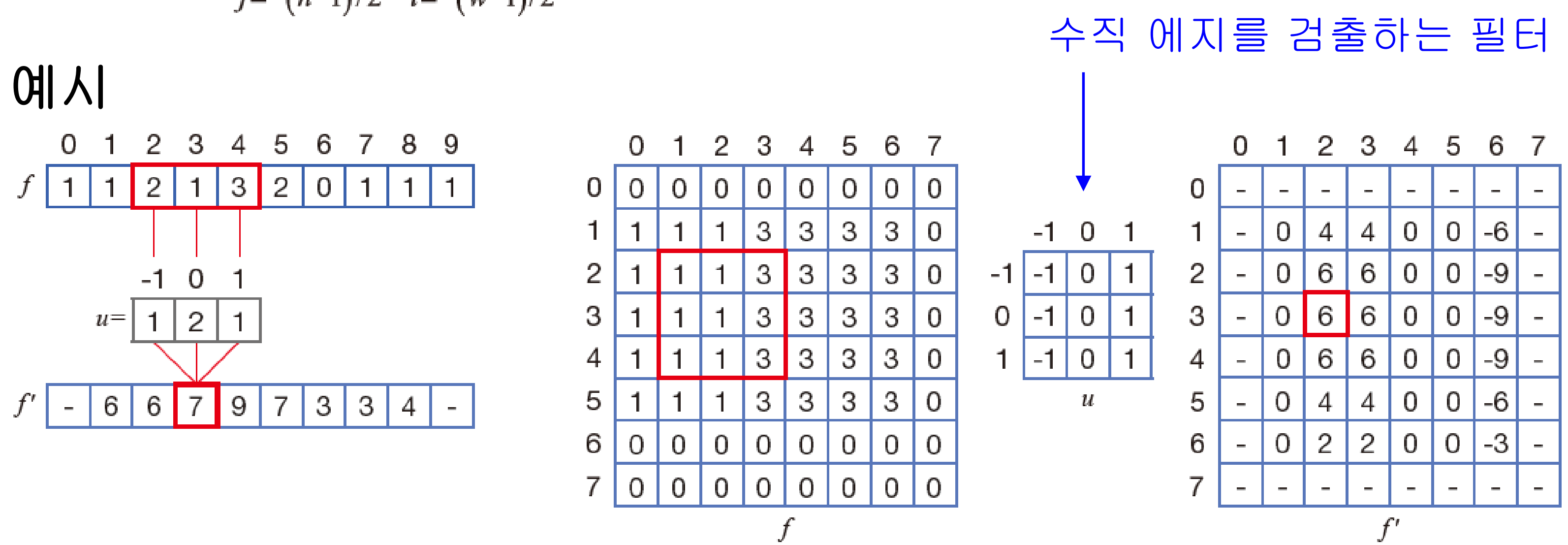
3.4.1 컨볼루션

■ 컨볼루션은 각 화소에 필터 **u**를 적용해 곱의 합을 구하는 연산

$$f'(x) = \sum_{i=-(w-1)/2}^{(w-1)/2} u(i) f(x+i) \qquad (3.7)$$

$$f'(y,x) = \sum_{j=-(h-1)/2}^{(h-1)/2} \sum_{i=-(w-1)/2}^{(w-1)/2} u(j,i) f(y+j, x+i) \qquad (3.8)$$

■ 예시



(a) 1차원 영상에 컨볼루션 적용 (b) 2차원 영상에 컨볼루션 적용

그림 3-16 컨볼루션의 원리

3.4.2 다양한 필터

■ 목적에 따라 다양한 필터 사용

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

0.0030	0.0133	0.0219	0.0133	0.0030
0.0133	0.0596	0.0983	0.0596	0.0133
0.0219	0.0983	0.1621	0.0983	0.0219
0.0133	0.0596	0.0983	0.0596	0.0133
0.0030	0.0133	0.0219	0.0133	0.0030

(a) 스무딩 필터

0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

(b) 샤프닝 필터

-1	0	0
0	0	0
0	0	1

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

(c) 엠보싱 필터

그림 3-17 잡음 제거와 대비 향상을 위한 필터

3.4.2 다양한 필터

■ 가우시안 필터

$$\text{1차원 가우시안:} \quad g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$$

$$\text{2차원 가우시안:} \quad g(y, x) = \frac{1}{\sigma^2 2\pi} e^{-\frac{y^2+x^2}{2\sigma^2}} \tag{3.9}$$

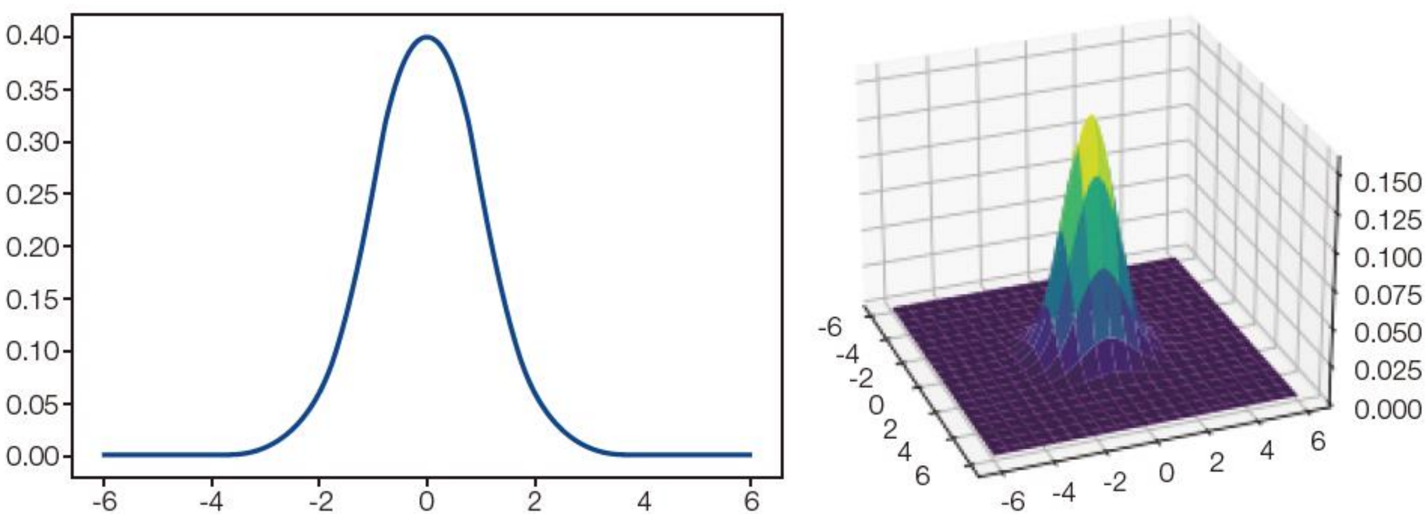


그림 3-18 1차원과 2차원 가우시안 함수

3.4.3 데이터 형과 컨볼루션

■ 연산 결과를 저장하는 변수의 유효 값 범위

- OpenCV는 주의를 기울여 작성되어 있음
- 때로 프로그래머가 직접 신경 써야 하는 경우 있음. 예) filter2D 함수

■ 데이터 형

- Opencv는 영상 화소를 주로 numpy.uint8 형으로 표현 ([0,255] 범위)

```
In [1]: print(type(img[0,0,0]))
        numpy.uint8
```

- [0,255] 범위를 벗어나는 경우 문제 발생

```
In [2]: a=np.array([-3,-2,-1,0,1,254,255,256,257,258],dtype=np.uint8)
In [3]: print(a)
        [253 254 255   0   1 254 255   0   1   2]
```

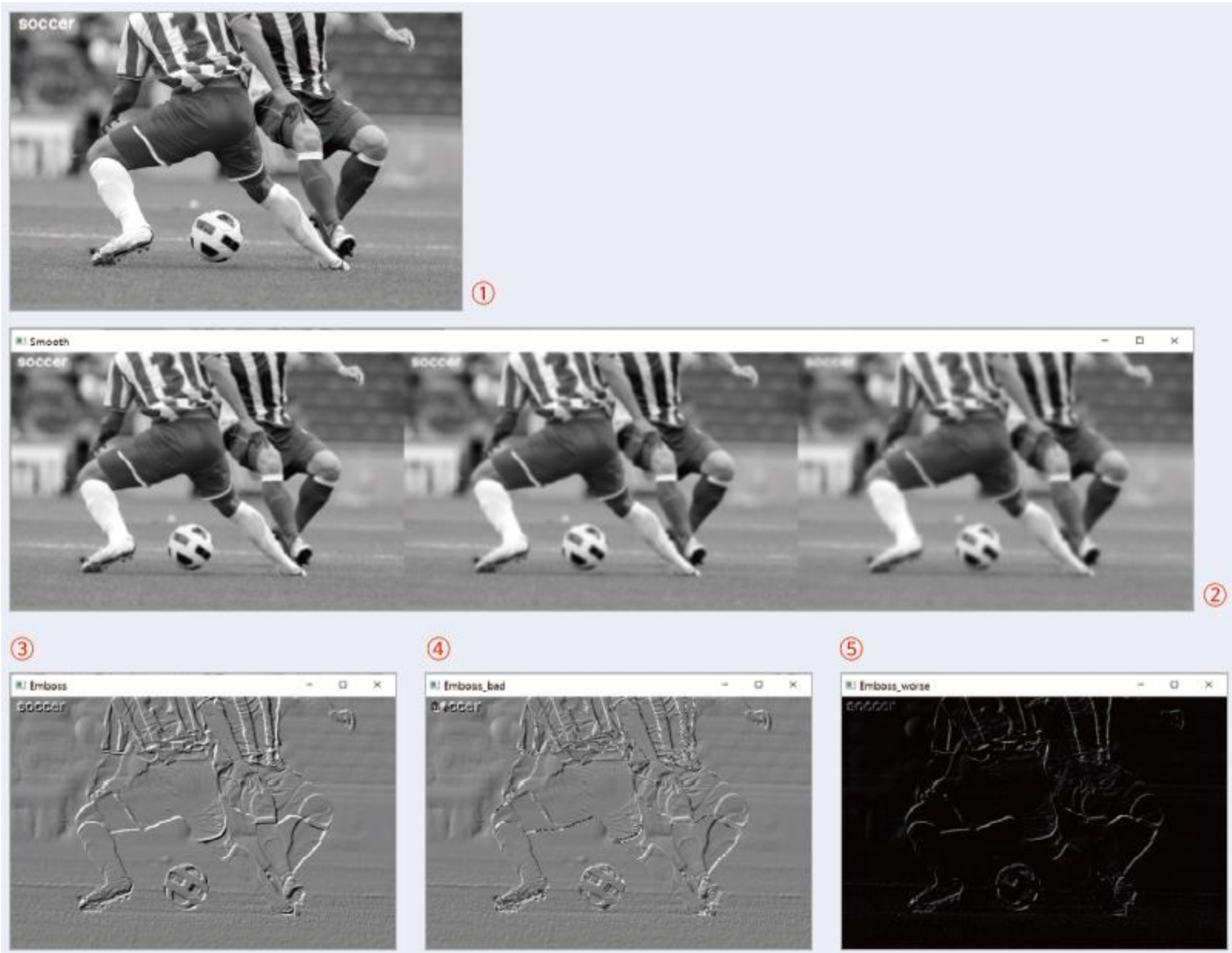
- 예) 엠보싱의 경우 [-255~255] 발생하는데 어떻게 해결하나?

3.4.3 데이터 형과 컨볼루션

프로그램 3-7 컨볼루션 적용(가우시안 스무딩과 엠보싱)하기

```
import cv2 as cv
import numpy as np

img=cv.imread('soccer.jpg')
img=cv.resize(img,dsize=(0,0),fx=0.4,fy=0.4)
gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)
cv.putText(gray,'soccer',(10,20),cv.FONT_HERSHEY_SIMPLEX,0.7,(255,255,255),2)
cv.imshow('Original',gray) ①

smooth=np.hstack((cv.GaussianBlur(gray,(5,5),0.0),cv.
                 GaussianBlur(gray,(9,9),0.0),cv.GaussianBlur(gray,(15,15),0.0)))
cv.imshow('Smooth',smooth) ②

femboss=np.array([[-1.0, 0.0, 0.0],
                  [ 0.0, 0.0, 0.0],
                  [ 0.0, 0.0, 1.0]])

gray16=np.int16(gray)
emboss=np.uint8(np.clip(cv.filter2D(gray16,-1,femboss)+128,0,255))
emboss_bad=np.uint8(cv.filter2D(gray16,-1,femboss)+128)
emboss_worse=cv.filter2D(gray,-1,femboss)

cv.imshow('Emboss',emboss) ③
cv.imshow('Emboss_bad',emboss_bad) ④
cv.imshow('Emboss_worse',emboss_worse) ⑤

cv.waitKey()
cv.destroyAllWindows()
```

3.4.3 데이터 형과 컨볼루션



3.5 기하 연산

- 기하 연산이 정해준 위치의 화소에서 값을 가져옴
  - 주로 물체의 이동, 크기, 회전에 따른 기하 변환

3.5.1 동차 좌표와 동차 행렬

- **동차 좌표**homogeneous coordinate
  - 2차원 좌표에 1을 추가해 3차원 벡터로 표현
  - 3개 요소에 같은 값을 곱하면 같은 좌표. 예) (-2,4,1)과 (-4,8,2)는 (-2,4)에 해당

$$\bar{p} = (x, y, 1) \qquad (3.10)$$

- **여러 가지 기하 변환**

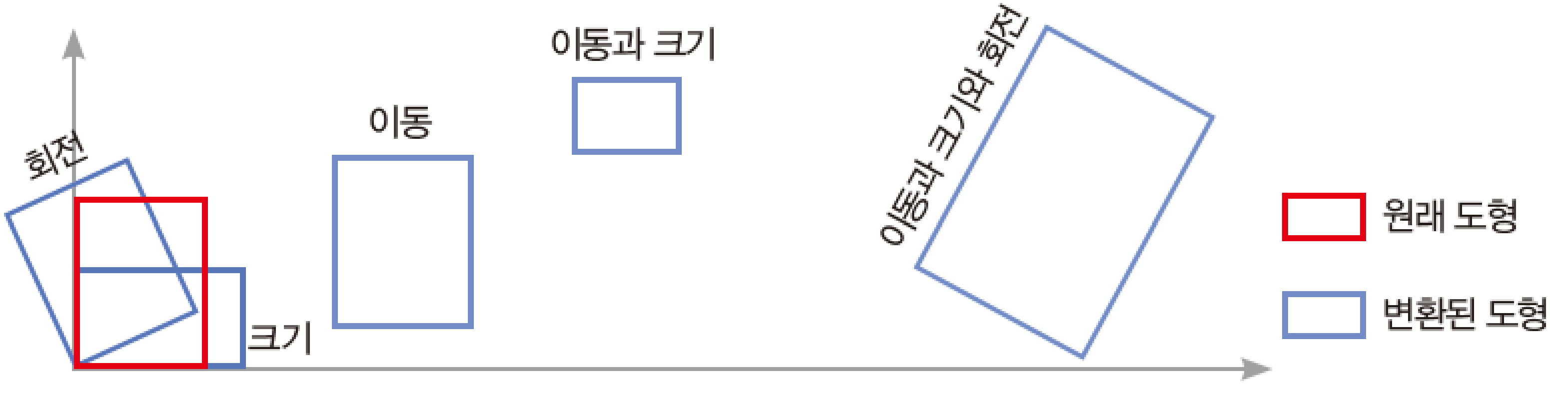


그림 3-19 여러 가지 기하 변환

3.5.1 동차 좌표와 동차 행렬

■ 기하 연산을 동차 행렬$_{\text{homogeneous matrix}}$로 표현

- [표 3-1] 변환은 모두 어파인 변환$_{\text{affine transform}}$: 평행을 평행으로 유지

표 3-1 3가지 기하 변환

기하 변환	동차 행렬	설명
이동	$T(t_x,t_y)=\begin{pmatrix}1&0&t_x\\0&1&t_y\\0&0&1\end{pmatrix}$	x 방향으로 t_x, y 방향으로 t_y만큼 이동
회전	$R(\theta)=\begin{pmatrix}\cos\theta&\sin\theta&0\\-\sin\theta&\cos\theta&0\\0&0&1\end{pmatrix}$	원점을 중심으로 반시계 방향으로 θ만큼 회전
크기	$S(s_x,s_y)=\begin{pmatrix}s_x&0&0\\0&s_y&0\\0&0&1\end{pmatrix}$	x 방향으로 s_x, y 방향으로 s_y만큼 크기 조정(1보다 크면 확대, 1보다 작으면 축소)

3.5.1 동차 좌표와 동차 행렬

■ [예시 3-5] 동차 행렬을 이용한 기하 변환

- 정사각형을 x 방향으로 2, y 방향으로 -1만큼 이동한 다음 반시계 방향으로 30도 회전

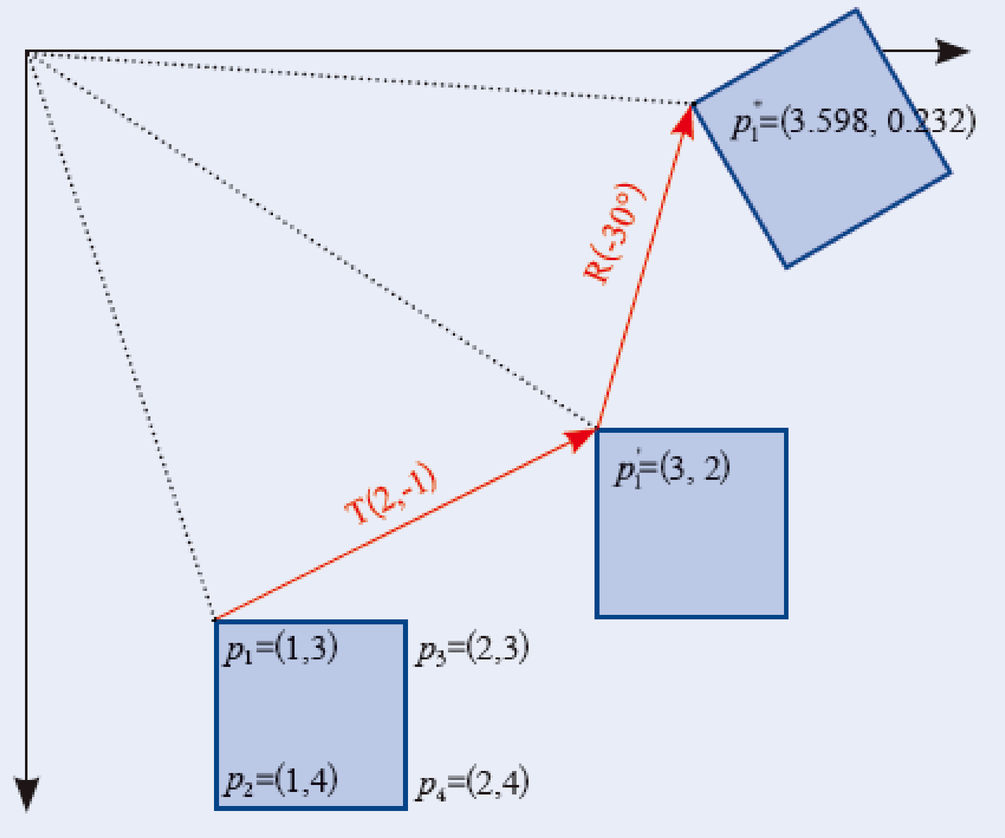


그림 3-20 동차 행렬을 이용한 기하 변환

3.5.1 동차 좌표와 동차 행렬

■ 변환을 위한 동차 행렬

$$T(2,-1)=\begin{pmatrix}1 & 0 & 2\\ 0 & 1 & -1\\ 0 & 0 & 1\end{pmatrix},\quad R(30°)=\begin{pmatrix}0.8660 & 0.5000 & 0\\ -0.5000 & 0.8660 & 0\\ 0 & 0 & 1\end{pmatrix}$$

■ 이동 적용

$$\overline{p}_1^{'T}=T(2,-1)\,\overline{p}_1^{T}=\begin{pmatrix}1 & 0 & 2\\ 0 & 1 & -1\\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}1\\ 3\\ 1\end{pmatrix}=\begin{pmatrix}3\\ 2\\ 1\end{pmatrix}$$

■ 회전 적용

$$\overline{p}_1^{''T}=R(30°)\,\overline{p}_1^{'T}=\begin{pmatrix}0.8660 & 0.5000 & 0\\ -0.5000 & 0.8660 & 0\\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}3\\ 2\\ 1\end{pmatrix}=\begin{pmatrix}3.598\\ 0.232\\ 1\end{pmatrix}$$

3.5.1 동차 좌표와 동차 행렬

■ 동차 행렬을 이용하면 계산이 효율적임

- 복합 변환을 위한 행렬을 미리 곱해 놓으면, 모든 점에 대해 한번의 행렬 곱셈으로 기하 변환 가능(행렬 곱셈은 결합 법칙 성립)

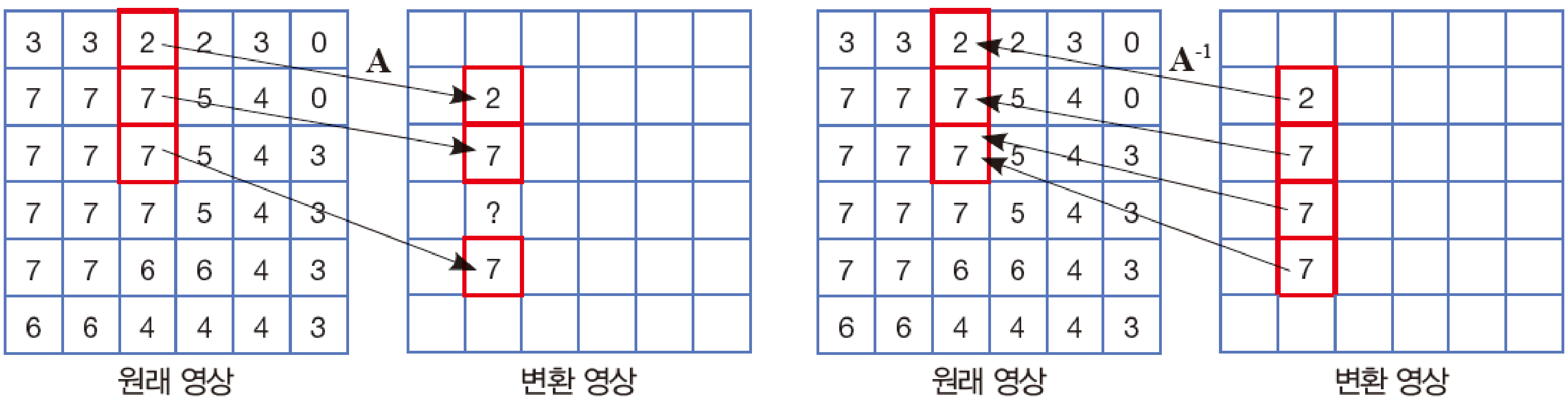
$$\mathbf{A}=\mathrm{R}(30°)\mathrm{T}(2,-1)=\begin{pmatrix}0.8660 & 0.5000 & 0\\ -0.5000 & 0.8660 & 0\\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}1 & 0 & 2\\ 0 & 1 & -1\\ 0 & 0 & 1\end{pmatrix}=\begin{pmatrix}0.8660 & 0.5000 & 1.232\\ -0.5000 & 0.8660 & -1.866\\ 0 & 0 & 1\end{pmatrix}$$

$$\mathbf{A}\bar{p}_1^{\mathrm{T}}=\begin{pmatrix}0.8660 & 0.5000 & 1.232\\ -0.5000 & 0.8660 & -1.866\\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}1\\ 3\\ 1\end{pmatrix}=\begin{pmatrix}3.598\\ 0.232\\ 1\end{pmatrix}$$

3.5.2 영상의 기하 변환

■ 화소 좌표에 동차 행렬 적용하여 기하 변환

- 값을 받지 못하는 화소가 생기는 에일리어싱 aliasing 현상 가능성
- 후방 연산을 통한 안티 에일리어싱



(a) 전방 변환

(b) 후방 변환

그림 3-21 영상의 기하 변환

3.5.3 영상 보간

■ 실수 좌표를 정수로 변환하는 방법

- 최근접 이웃 방법: 반올림 사용(에일리어싱 발생)
- 양선형 보간법: 걸치는 비율에 따라 선형 곱을 함으로써 안티 에일리어싱

$$f(j',i') = \alpha\beta f(j,i) + (1-\alpha)\beta f(j,i+1) + \alpha(1-\beta) f(j+1,i) + (1-\alpha)(1-\beta) f(j+1,i+1) \quad (3.11)$$

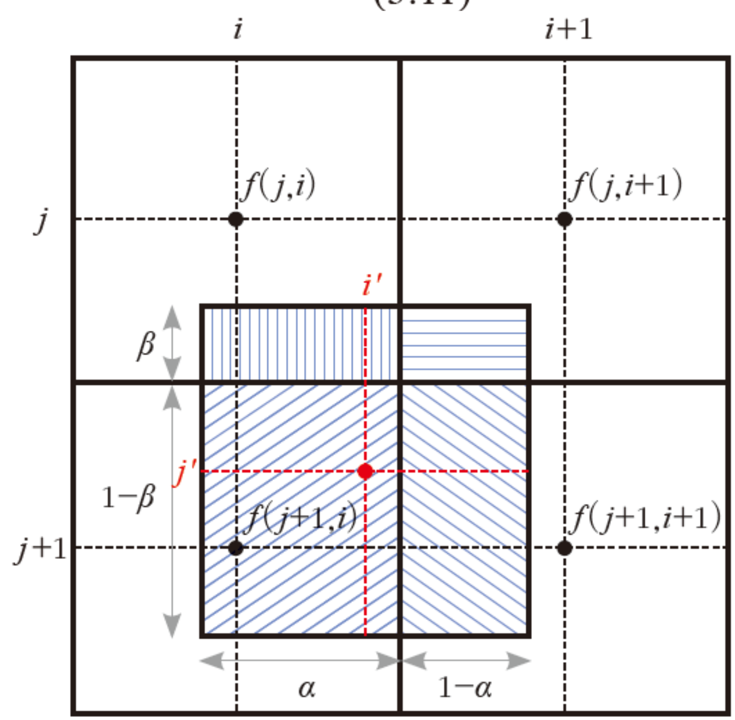


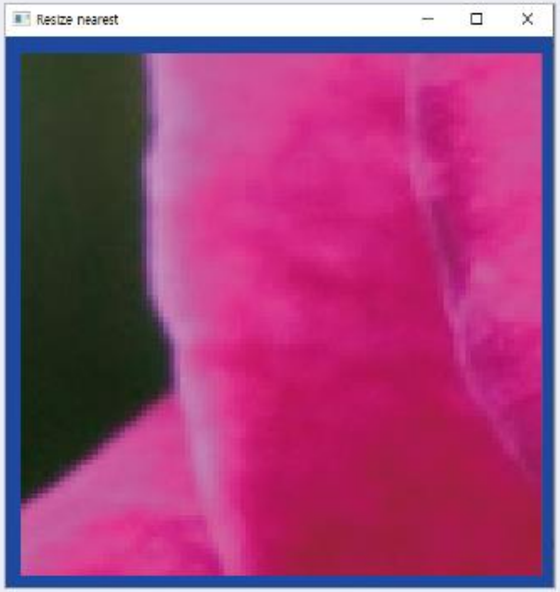
그림 3-22 실수 좌표의 화솟값을 보간하는 과정

3.5.3 영상 보간

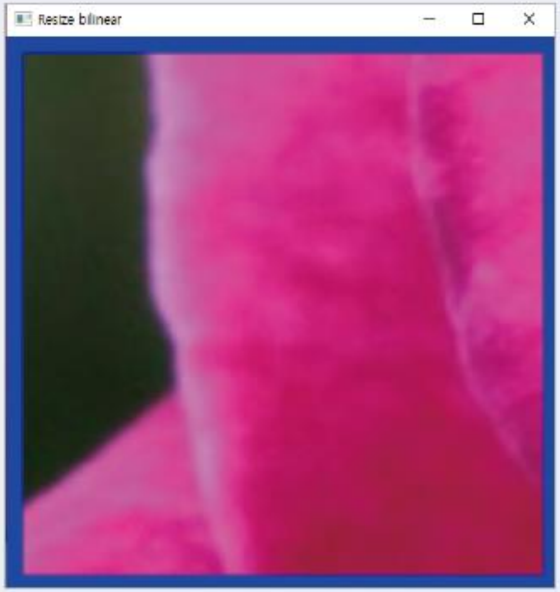
프로그램 3-8 보간을 이용해 영상의 기하 변환하기

```
import cv2 as cv

img=cv.imread('rose.png')
patch=img[250:350,170:270,:]

img=cv.rectangle(img,(170,250),(270,350),(255,0,0),3)
patch1=cv.resize(patch,dsize=(0,0),fx=5,fy=5,interpolation=cv.INTER_NEAREST)
patch2=cv.resize(patch,dsize=(0,0),fx=5,fy=5,interpolation=cv.INTER_LINEAR)
patch3=cv.resize(patch,dsize=(0,0),fx=5,fy=5,interpolation=cv.INTER_CUBIC)

cv.imshow('Original',img)
cv.imshow('Resize nearest',patch1)
cv.imshow('Resize bilinear',patch2)
cv.imshow('Resize bicubic',patch3)

cv.waitKey()
cv.destroyAllWindows()
```

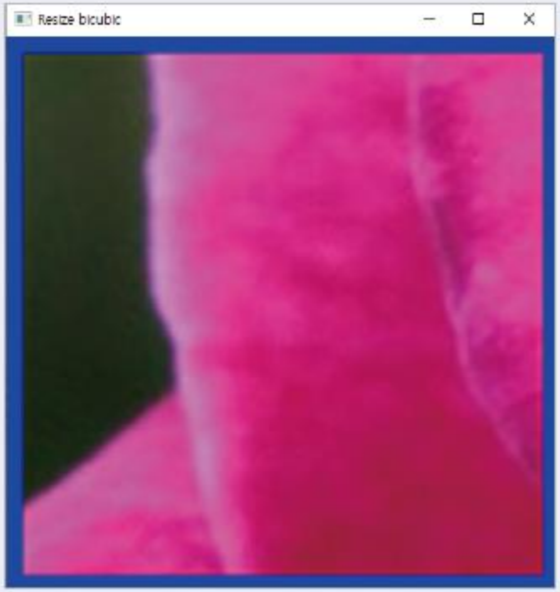
3.5.3 영상 보간



최근접 이웃



양선형 보간



양3차 보간

3.6 OpenCV의 시간 효율

- 컴퓨터 비전은 인식 정확률 뿐 아니라 시간 효율도 중요
  - 특히 실시간 처리가 요구되는 응용

- **OpenCV**는 효율적으로 구현되었기 때문에 **OpenCV** 함수를 사용하는 것이 유리
  - C/C++로 구현하고 인텔 마이크로프로세서에 최적화

- 직접 구현하는 경우 파이썬의 배열 연산 사용하는 것이 유리

3.6 OpenCV의 시간 효율

프로그램 3-9 직접 작성한 함수와 OpenCV가 제공하는 함수의 시간 비교하기

```
import cv2 as cv
import numpy as np
import time

def my_cvtGray1(bgr_img):
    g=np.zeros([bgr_img.shape[0],bgr_img.shape[1]])
    for r in range(bgr_img.shape[0]):
        for c in range(bgr_img.shape[1]):
            g[r,c]=0.114*bgr_img[r,c,0]+0.587*bgr_img[r,c,1]+0.299*bgr_img[r,c,2]
    return np.uint8(g)

def my_cvtGray2(bgr_img):
    g=np.zeros([bgr_img.shape[0],bgr_img.shape[1]])
    g=0.114*bgr_img[:,:,0]+0.587*bgr_img[:,:,1]+0.299*bgr_img[:,:,2]
    return np.uint8(g)

```

3.6 OpenCV의 시간 효율

```python
img=cv.imread('girl_laughing.png')

start=time.time()
my_cvtGray1(img)
print('My time1:',time.time()-start) ①

start=time.time()
my_cvtGray2(img)
print('My time2:',time.time()-start) ②

start=time.time()
cv.cvtColor(img,cv.COLOR_BGR2GRAY)
print('OpenCV time:',time.time()-start) ③
```

```
My time1: 4.798288106918335 ①
My time2: 0.015836000442504883 ②
OpenCV time: 0.013601541519165039 ③
```