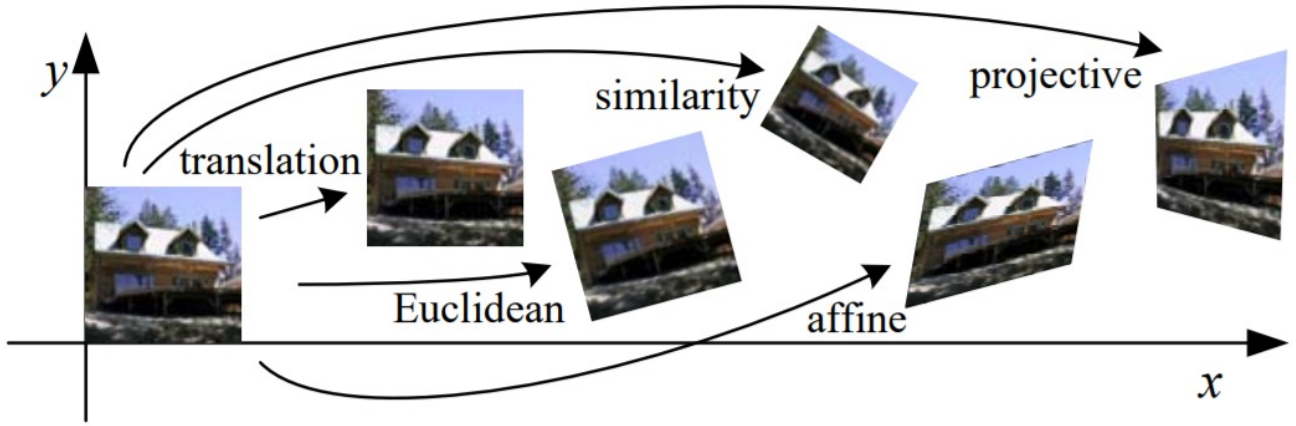
영상의 생성, 복사, 부분 영상 추출

- 지정한 크기로 새 영상 생성하기

```
numpy.empty(shape, dtype=float, ...) -> arr
numpy.zeros(shape, dtype=float, ...) -> arr
numpy.ones(shape, dtype=None, ...) -> arr
numpy.full(shape, fill_value, dtype=None, ...) -> arr
```

- shape: 각 차원의 크기. (h, w) 또는 (h, w, 3)
- dtype: 원소의 데이터 타입. 일반적인 영상이면 numpy.uint8 지정
- arr: 생성된 영상(numpy.ndarray)
- 참고사항:
  - numpy.empty() 함수는 임의의 값으로 초기화된 배열을 생성
  - numpy.zeros() 함수는 0으로 초기화된 배열을 생성
  - numpy.ones() 함수는 1로 초기화된 배열을 생성
  - numpy.full() 함수는 fill_value로 초기화된 배열을 생성

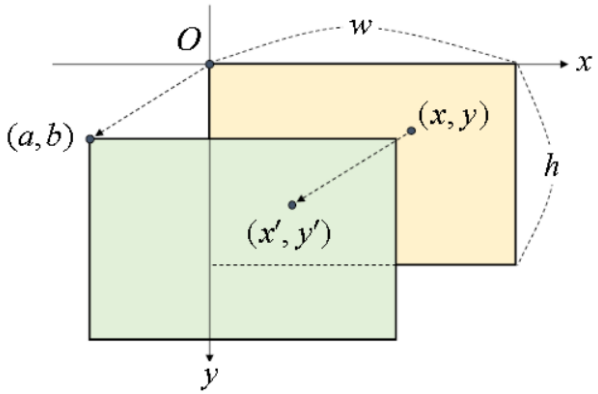
▪ 영상의 기하학적 변환



http://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf

영상의 이동 변환(Translation transformation)

- 가로 또는 세로 방향으로 영상을 특정 크기만큼 이동시키는 변환
- X축과 Y축 방향으로의 이동 변위를 지정



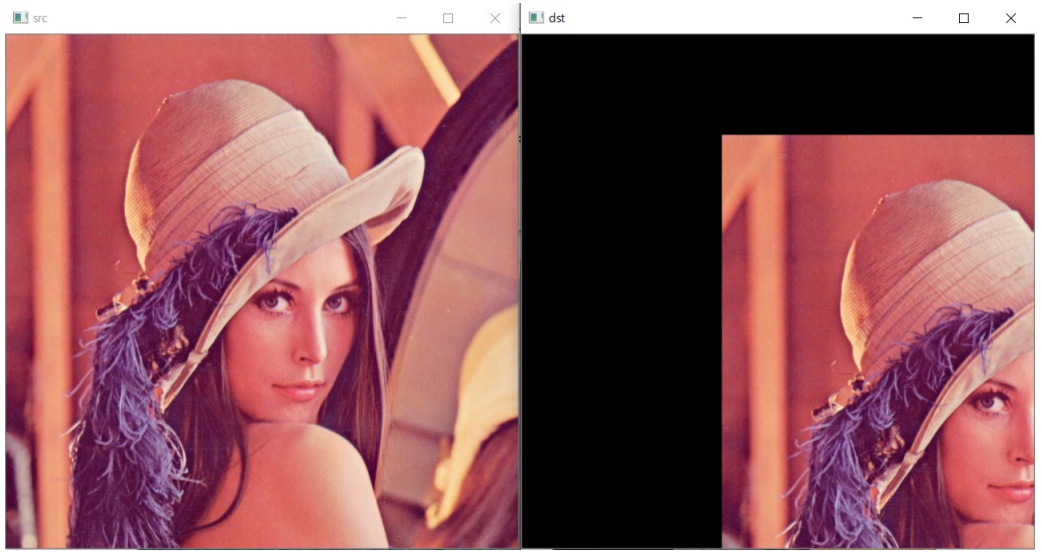
$$\begin{cases} x' = x + a \\ y' = y + b \end{cases} \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2x3 어파인 변환 행렬

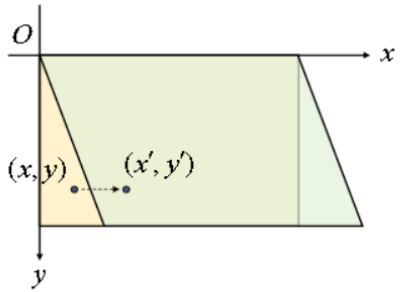
영상의 이동 변환(Translation transformation)

```
aff = np.array([[1, 0, 200],
                [0, 1, 100]], dtype=np.float32)

dst = cv2.warpAffine(src, aff, (0, 0))
```

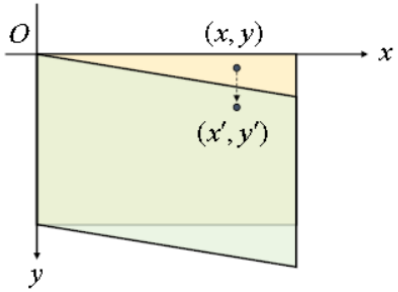


영상의 전단 변환(Shear transformation)

- 층 밀림 변환, X축과 Y축 방향에 대해 각각 정의



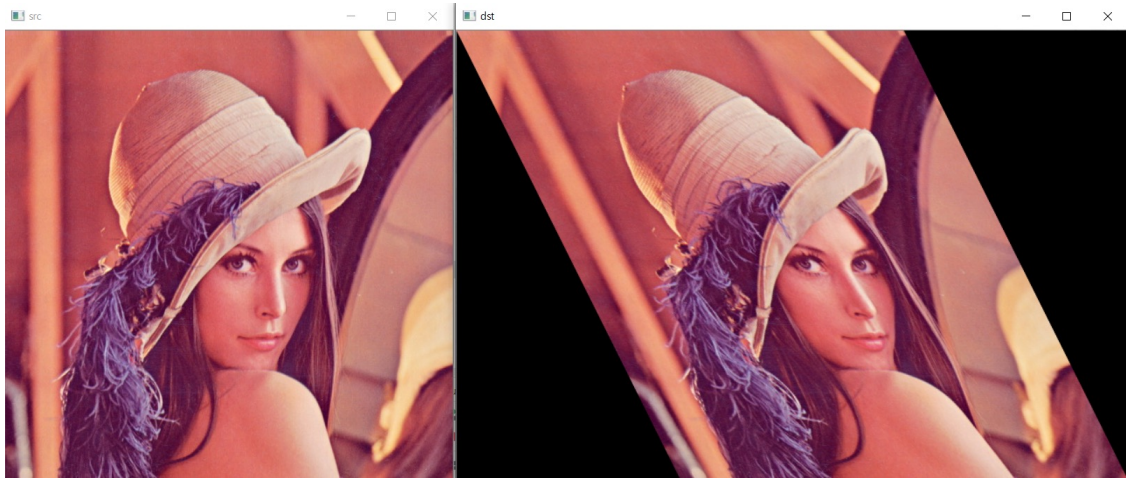
$$\begin{cases} x' = x + my \\ y' = y \end{cases} \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & m & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



$$\begin{cases} x' = x \\ y' = mx + y \end{cases} \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ m & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

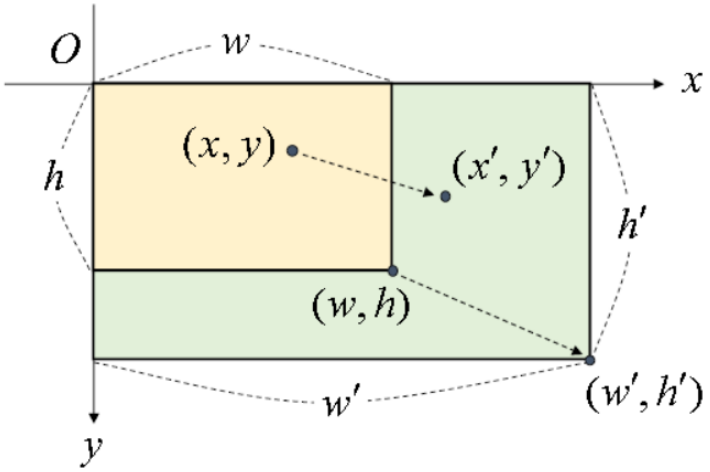
영상의 전단 변환(Shear transformation)

```
aff = np.array([[1, 0.5, 0],
                [0, 1, 0]], dtype=np.float32)

h, w = src.shape[:2]
dst = cv2.warpAffine(src, aff, (w + int(h * 0.5), h))
```



영상의 확대와 축소 (Scale transformation)

- 영상의 크기를 원본 영상보다 크거나 작게 만드는 변환

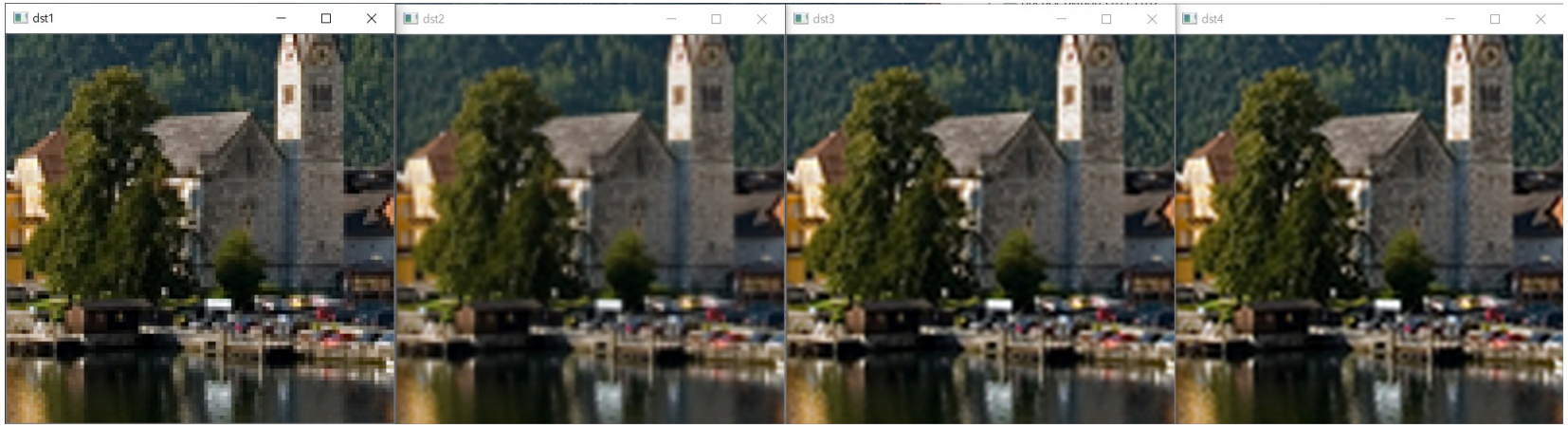


$$\begin{cases} x' = s_x x \\ y' = s_y y \end{cases} \qquad \begin{cases} s_x = w' / w \\ s_y = h' / h \end{cases}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

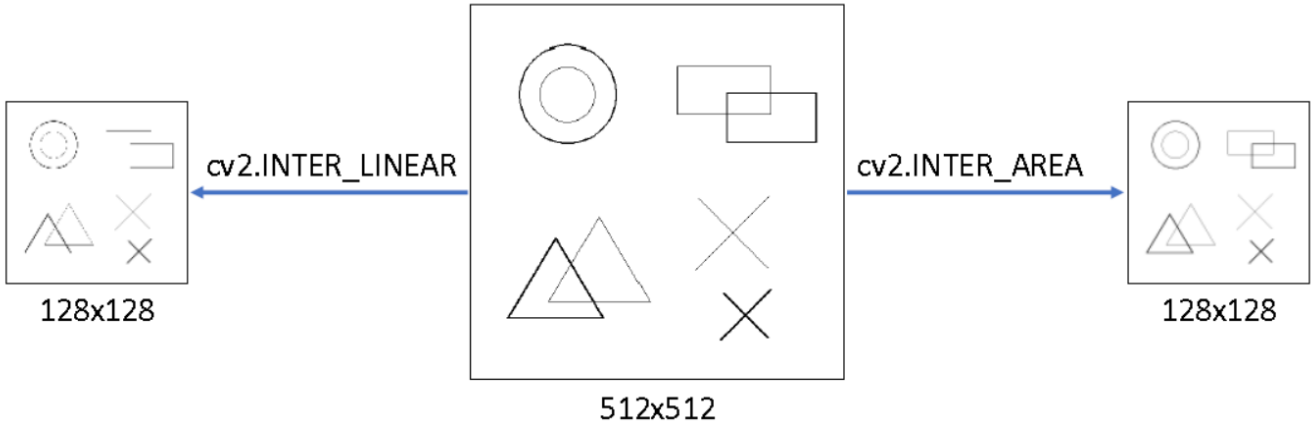
영상의 확대와 축소 (Scale transformation)

```
dst1 = cv2.resize(src, (0, 0), fx=4, fy=4, interpolation=cv2.INTER_NEAREST)
dst2 = cv2.resize(src, (1920, 1280))  # cv2.INTER_LINEAR
dst3 = cv2.resize(src, (1920, 1280), interpolation=cv2.INTER_CUBIC)
dst4 = cv2.resize(src, (1920, 1280), interpolation=cv2.INTER_LANCZOS4)
```



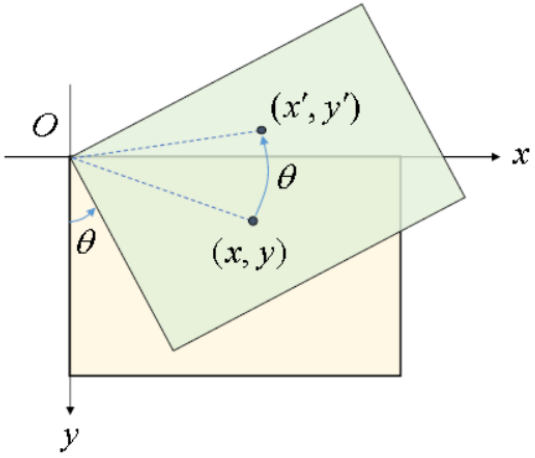
영상의 확대와 축소 (Scale transformation)

- 영상의 축소시 고려할 사항
- 영상 축소시 디테일이 사라지는 경우가 발생(특히, 한 픽셀로 구성된 선)
- 입력 영상을 부드럽게 필터링한 후 축소, 다단계 축소
- OpenCV의 cv2.resize()함수에서는 cv2.INTER_AREA 플래그를 사용



영상의 회전 (rotation transformation)

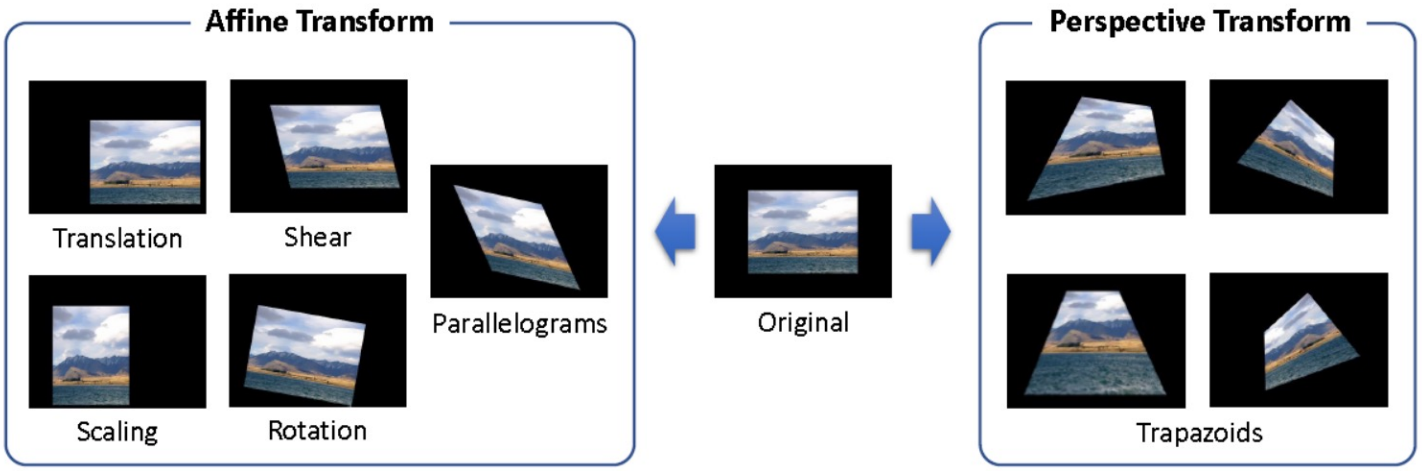
- 영상을 특정 각도만큼 회전시키는 변환(반시계 방향)



$$\begin{cases} x' = \cos\theta \cdot x + \sin\theta \cdot y \\ y' = -\sin\theta \cdot x + \cos\theta \cdot y \end{cases}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Affine Transform VS Perspective Transform



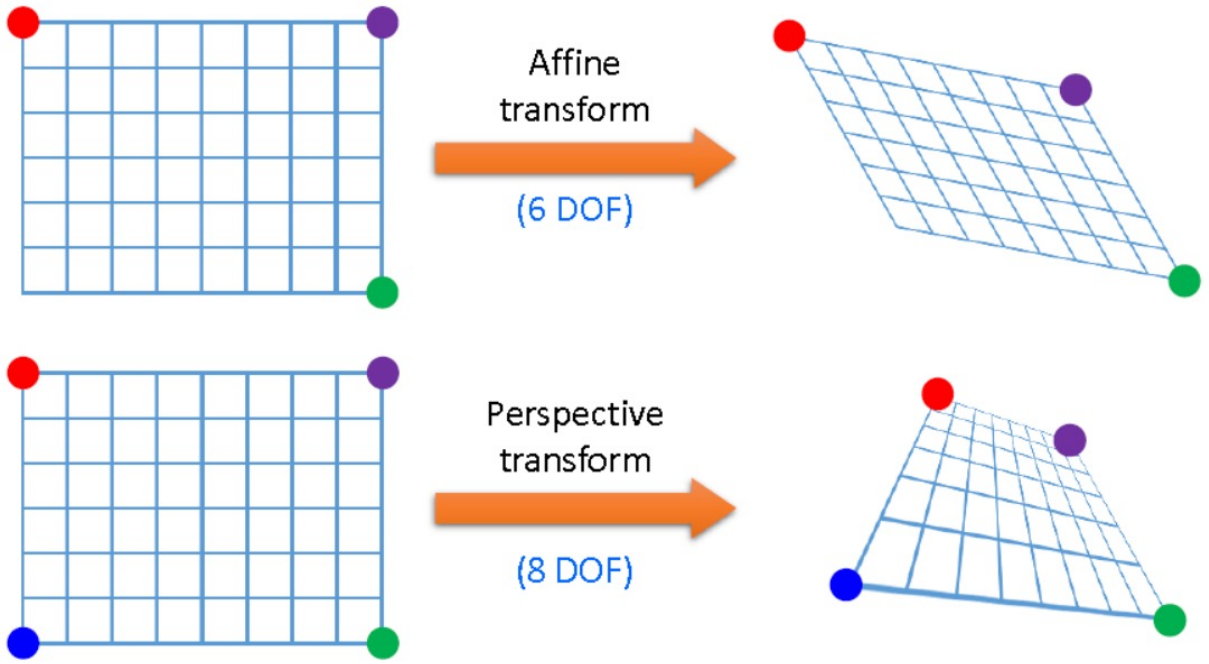
$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

2x3 matrix
(6 DOF)

3x3 matrix
(8 DOF)

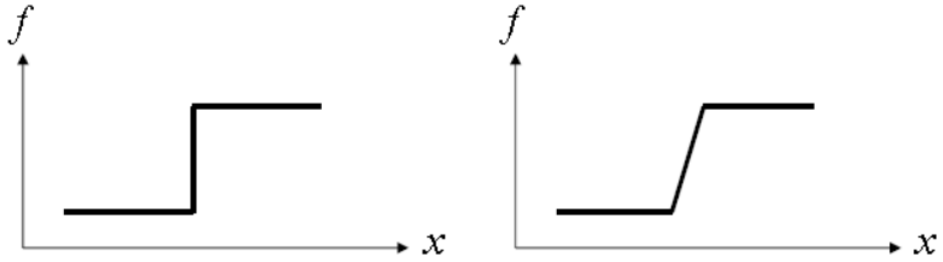
$$\begin{pmatrix} wx' \\ wy' \\ w \end{pmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Affine Transform VS Perspective Transform



에지 검출과 미분

- 에지(edge)
- 영상에서 픽셀의 값이 급격하게 변하는 부분



영상의 미분과 소벨 필터

가로 방향:

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

Prewitt

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

Sobel

-3	0	3
-10	0	10
-3	0	3

Scharr

세로 방향:

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

Prewitt

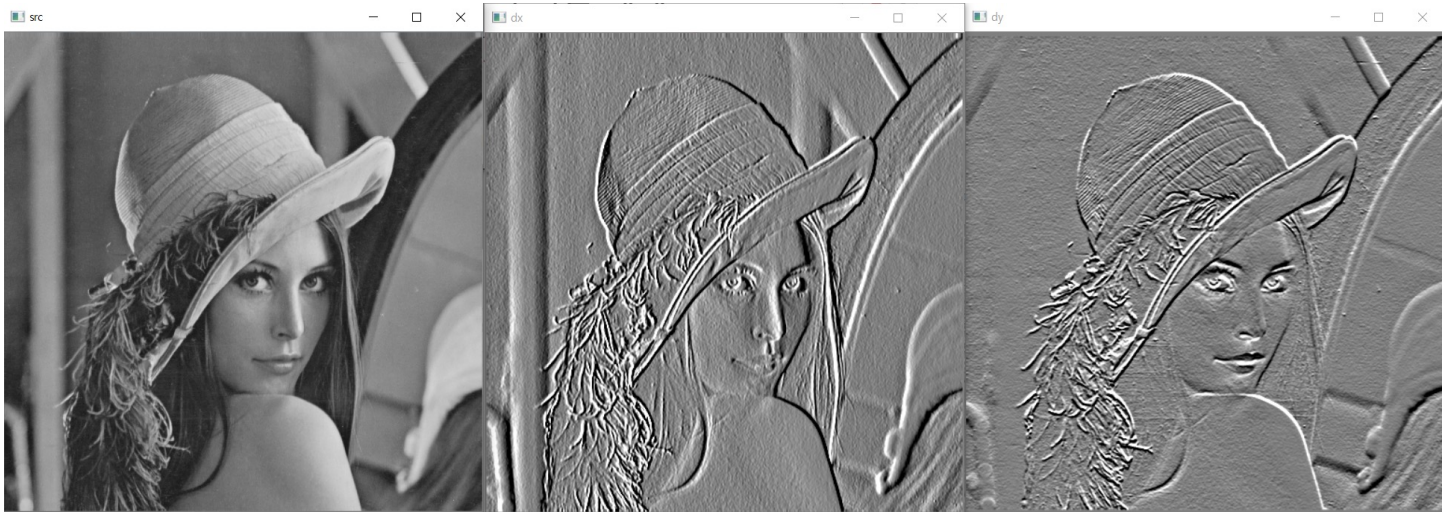
-1	-2	-1
0	0	0
1	2	1

Sobel

-3	-10	-3
0	0	0
3	10	3

Scharr

```
dx = cv2.Sobel(src, -1, 1, 0, delta=128)
dy = cv2.Sobel(src, -1, 0, 1, delta=128)
```



그래디언트

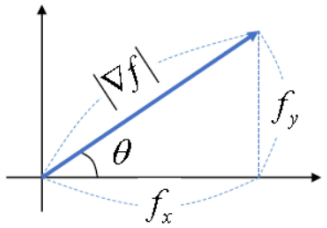
- 영상의 그래디언트

  - 함수 $f(x, y)$를 x축과 y 축으로 각각 편미분(partial derivative)하여 벡터 형태로 표현한 것

$$\nabla f = \begin{bmatrix} f_x \\ f_y \end{bmatrix} = f_x \mathbf{i} + f_y \mathbf{j}$$

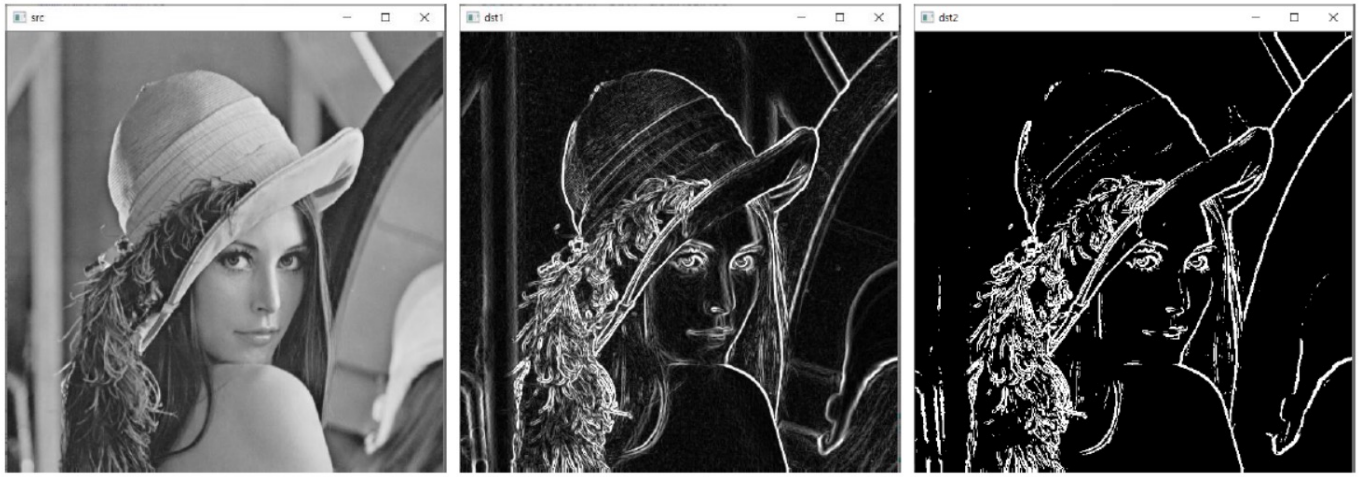
  - 그래디언트 크기: $|\nabla f| = \sqrt{f_x^2 + f_y^2}$

  - 그래디언트 방향: $\theta = \tan^{-1}\left(\frac{f_y}{f_x}\right)$



그래디언트와 에지 검출

```
dx = cv2.Sobel(src, cv2.CV_32F, 1, 0)
dy = cv2.Sobel(src, cv2.CV_32F, 0, 1)

mag = cv2.magnitude(dx, dy)
mag = np.clip(mag, 0, 255).astype(np.uint8)

dst = np.zeros(src.shape[:2], np.uint8)
dst[mag > 120] = 255
```



캐니 에지 검출 단계

- 1.가우시안 필터링 : 잡은 제거 목적
- 2. 그래디언트 계산 : 주로 소벨 마스크 사용

  크기: $\|f\| = \sqrt{f_x^2 + f_y^2}$

  방향: $\theta = \tan^{-1}\left(f_y / f_x\right)$

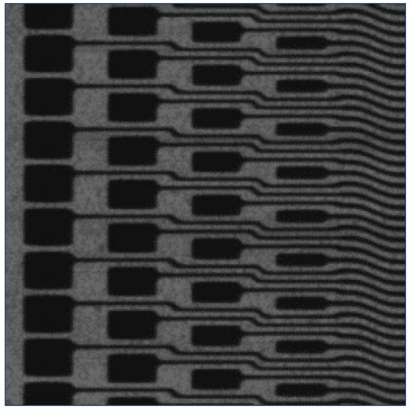
- 3. 비최대 억제(Non maximum suppression)

  하나의 에지가 여러 개의 픽셀로 표현되는 현상을 없애기 위해 그래디언트 크기가 국지적 최대인 픽셀만을 에지 픽셀로 설정

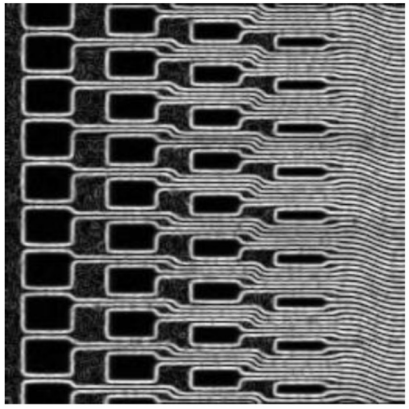
- 4. 히스테리시스 에지 트래킹

  두 개의 임계값을 사용: T_{Low}, T_{High}
  강한 에지: $\|f\| \geq T_{High}$ → 최종 에지로 선정
  약한 에지: $T_{Low} \leq \|f\| < T_{High}$
  → 강한 에지와 연결되어 있는 픽셀만 최종 에지로 선정

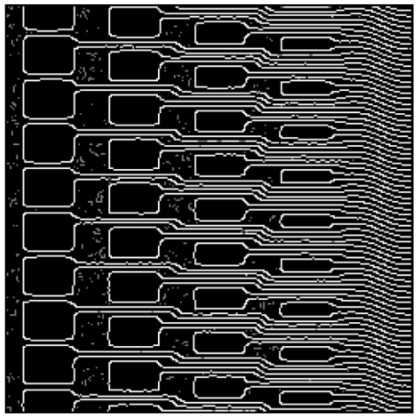
캐니 에지 검출 단계



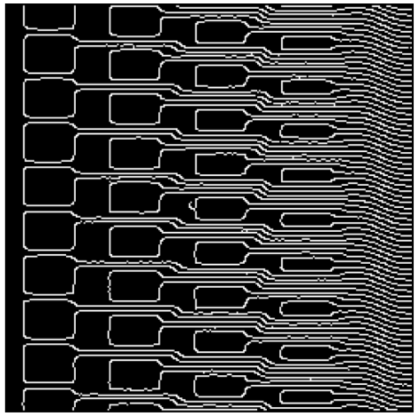
입력 영상



그래디언트 크기



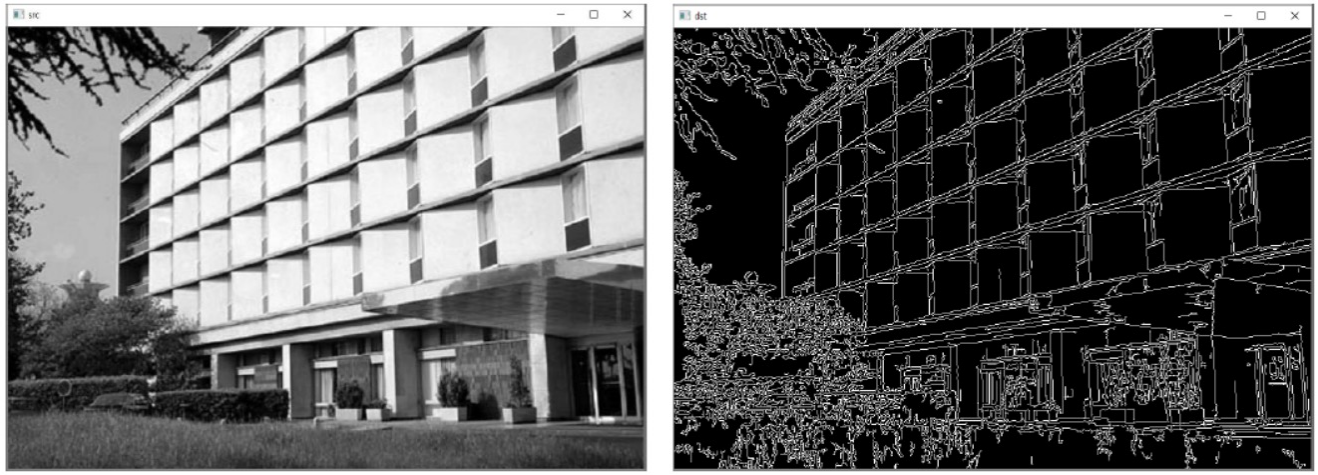
비최대 억제



히스테리시스 에지 트래킹

캐니 에지 검출

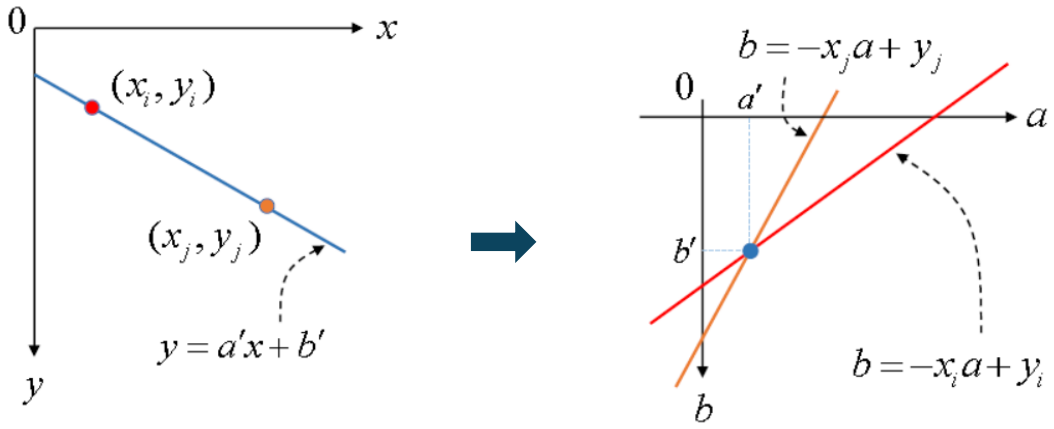
```
dst = cv2.Canny(src, 50, 150)
```



허프변환 : 직선검출

- 2차원 영상 좌표에서 직선의 방정식을 파라미터 공간으로 변환하여 직선을 찾는 알고리즘

$y = ax + b$ → $b = -xa + y$



허프변환 : 직선검출

- 직선의 방정식 $y = ax + b$ 를 사용할 때 y축과 평행한 직선을 표현하지 못하여 극좌표계 직선의 방정식을 사용

$$x\cos\theta + y\ \sin\theta = \rho$$

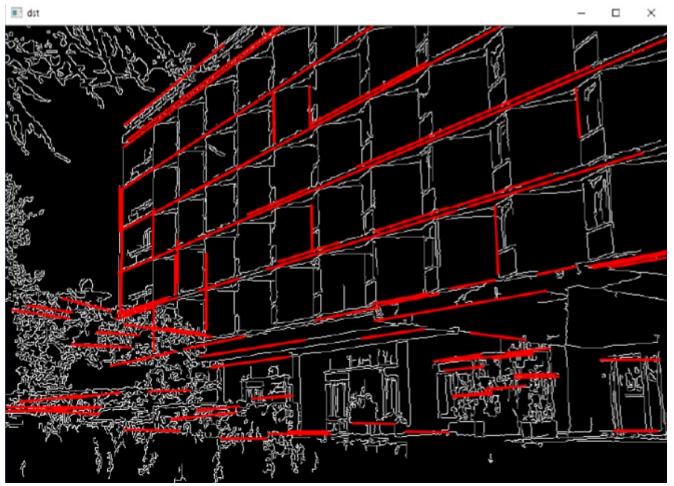
$$\begin{cases} \text{기울기} = -\dfrac{\cos\theta}{\sin\theta} \\ \text{y절편} = \dfrac{\rho}{\sin\theta} \end{cases}$$

$$y = -\frac{\cos\theta}{\sin\theta}x + \frac{\rho}{\sin\theta}$$

허프변환 : 직선검출

- 허프 변환 직선 검출 예제

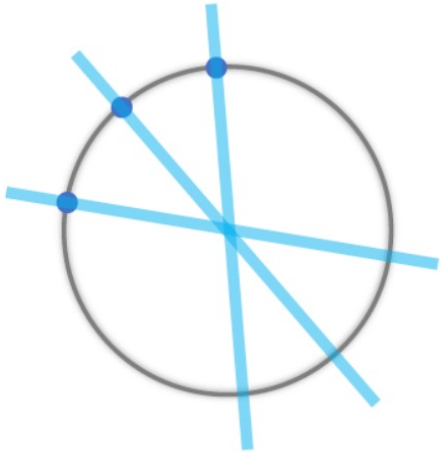
```
edges = cv2.Canny(src, 50, 150)

lines = cv2.HoughLinesP(edges, 1, np.pi / 180., 160,
                        minLineLength=50, maxLineGap=5)
```



허프변환 : 원검출

- 허프 변환을 응용하여 원을 검출

원의 방정식 : $(x-\mathrm{a})^2+(x-\mathrm{b})^2=c^2$ → 3차원 축적 평면



허프변환 : 원검출

- 허프 변환 직선 검출 예제

```
blr = cv2.GaussianBlur(gray, (0, 0), 1.0)
circles = cv2.HoughCircles(blr, cv2.HOUGH_GRADIENT, 1, 50,
                           param1=120, param2=th, minRadius=rmin,
maxRadius=rmax)
```