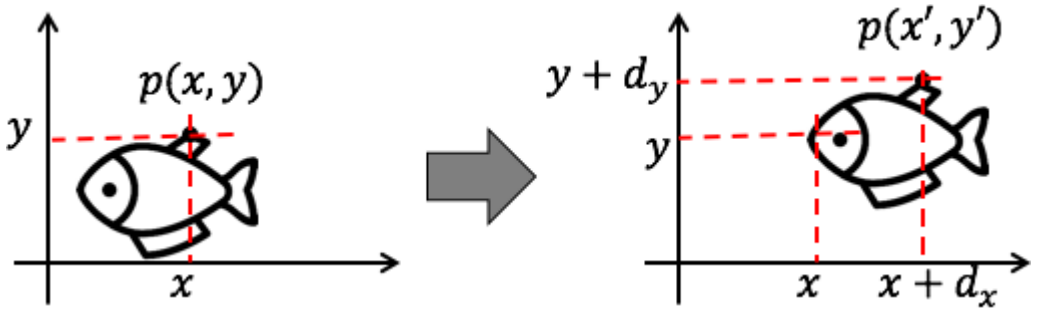
5장 . 기하학적 변환

세부목차

Geometric Transform

❖ Translate

- 영상 이동
  - 이동할 좌표 = 원래 있던 좌표 + 이동할 거리
- 방정식
  - $x' = x + d_x$
  - $y' = y + d_y$
- 행렬식
  - $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x + d_x \\ y + d_y \end{bmatrix} = \begin{bmatrix} 1x + 0y + 1d_x \\ 0x + 1y + 1d_y \end{bmatrix}$



[그림 5-1] 영상 이동을 위한 좌표 계산

Geometric Transform

❖ Translate

- dst = cv2.warpAffine(src, mtrx, dsize [, dst, flags, borderMode, borderValue])
  - src : 원본 영상, NumPy 배열
  - mtrx : 2 x 3 변환 행렬, NumPy 배열, dtype=float32
  - dsize : 결과 이미지 크기, tuple(width, height)
  - flags : 보간법 알고리즘 선택 플래그
    - cv2.INTER_LINEAR : 기본 값, 인접한 4개 픽셀값을 거리 가중치 사용
    - cv2.INTER_NEAREST : 가장 가까운 픽셀 값 사용
    - cv2.INTER_AREA : 픽셀 영역 관계를 이용한 재 샘플링
    - cv2.INTER_CUBIC : 인접한 16개 픽셀 값을 거리 가중치 사용
    - cv2.INTER_LANCZOS4 : 인접한 8개 픽셀을 이용한 란초의 알고리즘
  - borderMode : 외곽 영역 보정 플래그
    - cv2.BORDER_CONSTANT : 고정 색상 값 (999|12345|999)
    - cv2.BORDER_REPLICATE : 가장 자리 복제 (111|12345|555)
    - cv2.BORDER_WRAP : 반복 (345|12345|123)
    - cv2.BORDER_REFLECT : 반사 (321|12345|543)
  - borderValue : cv2.BORDER_CONSTANT 경우 사용 할 색상 값, 기본 값 = 0
  - dst : 결과 이미지, NumPy 배열

Geometric Transform

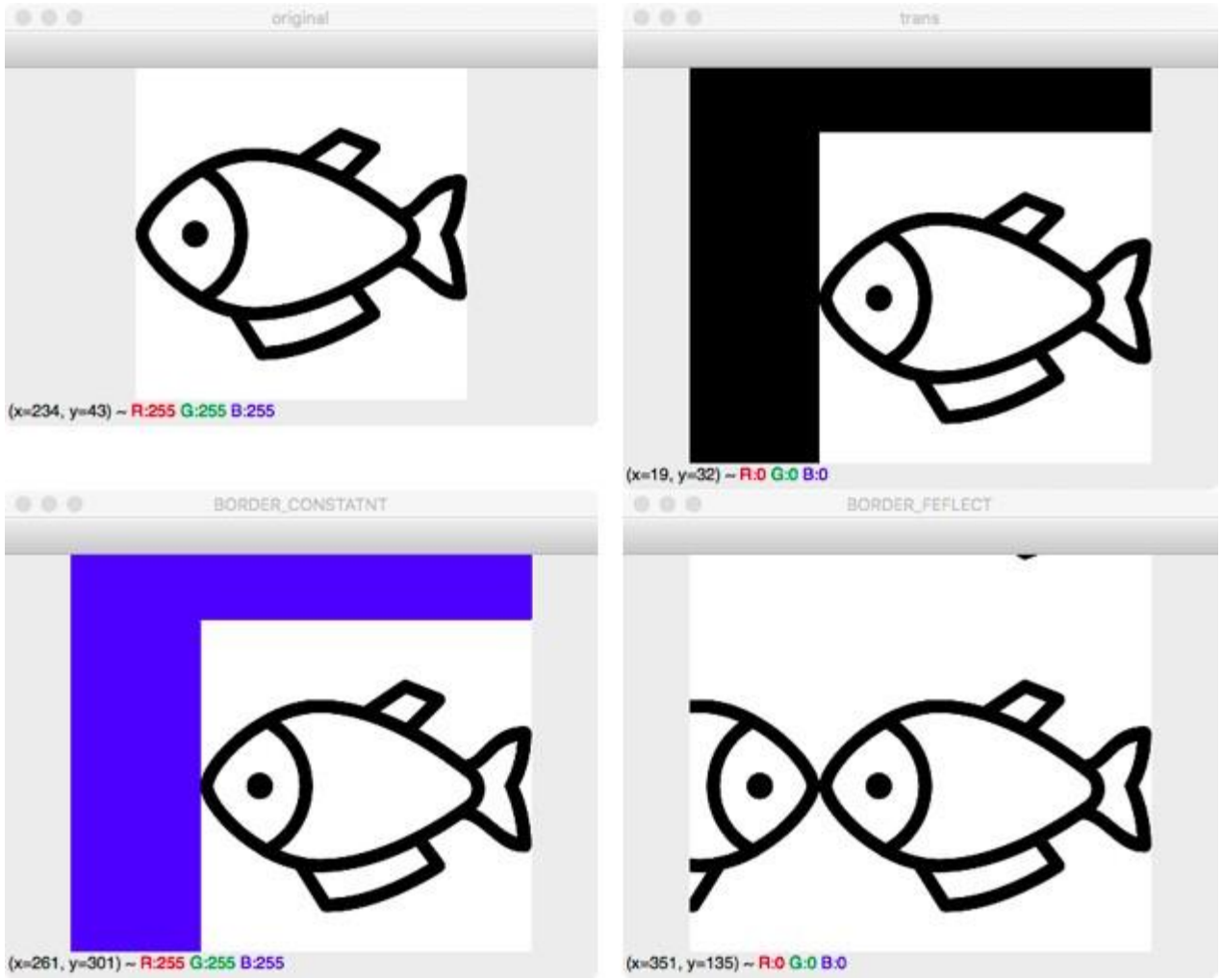
❖ Translate

```
import cv2
import numpy as np
img = cv2.imread('../img/fish.jpg')
rows,cols = img.shape[0:2] # 영상의 크기
dx, dy = 100, 50 # 이동할 픽셀 거리
# ---① 변환 행렬 생성
mtrx = np.float32([[1, 0, dx],
                   [0, 1, dy]])
# ---② 단순 이동
dst = cv2.warpAffine(img, mtrx, (cols+dx, rows+dy))
# ---③ 탈락된 외곽 픽셀을 파랑색으로 보정
dst2 = cv2.warpAffine(img, mtrx, (cols+dx, rows+dy), None, \
                      cv2.INTER_LINEAR, cv2.BORDER_CONSTANT, (255,0,0) )
# ---④ 탈락된 외곽 픽셀을 원본을 반사 시켜서 보정
dst3 = cv2.warpAffine(img, mtrx, (cols+dx, rows+dy), None, \
                      cv2.INTER_LINEAR, cv2.BORDER_REFLECT)
cv2.imshow('original', img)
cv2.imshow('trans',dst)
cv2.imshow('BORDER_CONSTATNT', dst2)
cv2.imshow('BORDER_FEFLECT', dst3)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[예제 5-2] 평행 이동(translate.py)

Geometric Transform

❖ **Translate**



[그림 5-2] [5-1]의 실행 결과

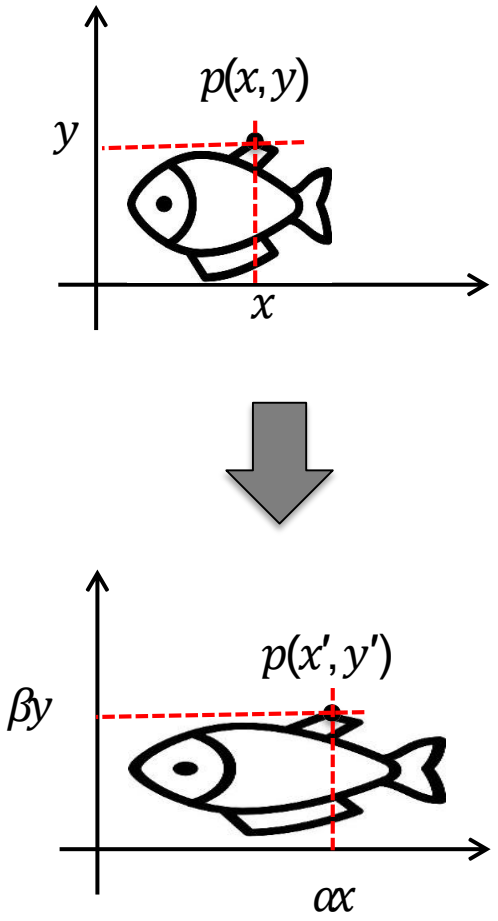
Geometric Transform

❖ Scaling

- 확대/축소 : $\alpha \cdot \beta$배 변환 행렬

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- dst=cv2.resize(src, dsize, fx,fy, interpolation)
  - src : 입력 이미지
  - dsize : 출력 이미지 크기 (w,h)
  - fx, fy : 배율, 생략하면 dsize 적용
  - interpolation : 보간법
    - cv2.INTER_LINEAR :default
    - cv2.INTER_NEAREST
    - cv2.INTER_AREA
    - cv2.INTER_CUBIC
    - cv2.INTER_LANCZOS4
  - dst : 결과 영상



[그림 5-3] 영상 확대/축소 위한 좌표 계산

Geometric Transform

❖ Scaling

- 변환 행렬 Example

```
mport cv2
import numpy as np
img = cv2.imread('../img/fish.jpg')
height, width = img.shape[:2]
# --① 0.5배 축소 변환 행렬
m_small = np.float32([[0.5, 0, 0],
                      [0, 0.5,0]])
# --② 2배 확대 변환 행렬
m_big = np.float32([[2, 0, 0],
                    [0, 2, 0]])
# --③ 보간법 적용 없이 확대 축소
dst1 = cv2.warpAffine(img, m_small, (int(height*0.5), int(width*0.5)))
dst2 = cv2.warpAffine(img, m_big, (int(height*2), int(width*2)))
```

[예제 5-2] 행렬을 이용한 확대와 축소(scale_matrix.py)

Geometric Transform

❖ Scaling

- 변환 행렬 Example

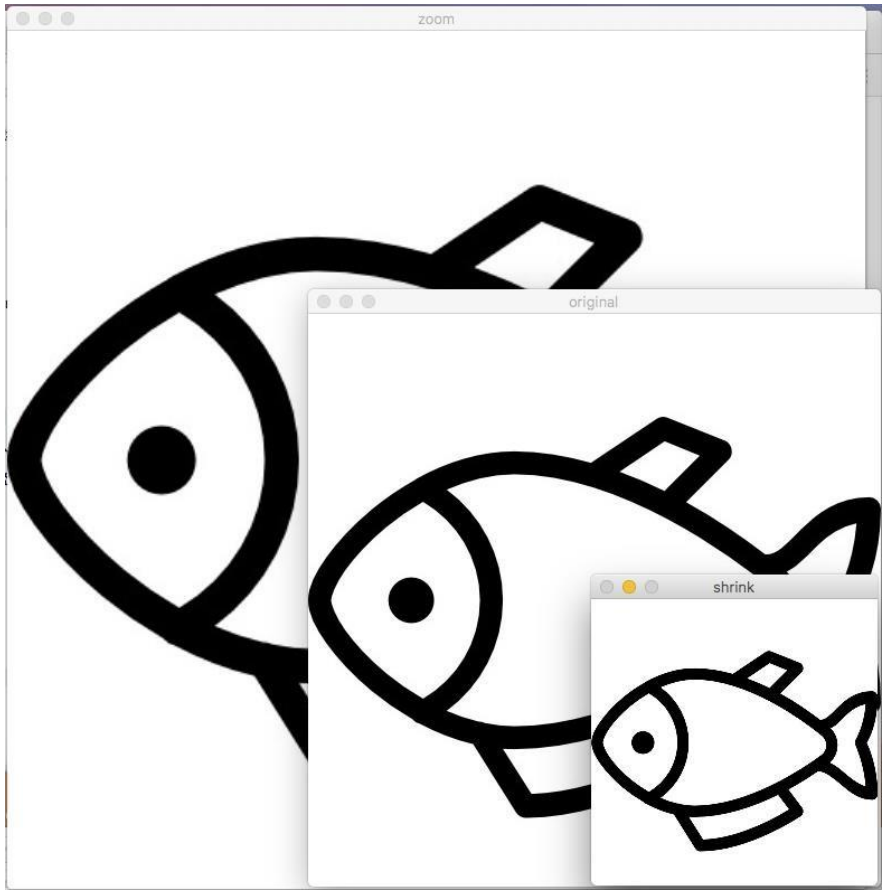
```
# --④ 보간법 적용한 확대 축소
dst3 = cv2.warpAffine(img, m_small, (int(height*0.5), int(width*0.5)), \
                                None, cv2.INTER_AREA)
dst4 = cv2.warpAffine(img, m_big, (int(height*2), int(width*2)), \
                                None, cv2.INTER_CUBIC)

# 결과 출력
cv2.imshow("original", img)
cv2.imshow("small", dst1)
cv2.imshow("big", dst2)
cv2.imshow("small INTER_AREA", dst3)
cv2.imshow("big INTER_CUBIC", dst4)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Geometric Transform

❖ Scaling

- 변환 행렬 Example <결과>



[그림 5-4] [5-2]의 실행 결과

Geometric Transform

❖ Scaling

- cv2.resize 함수 Example

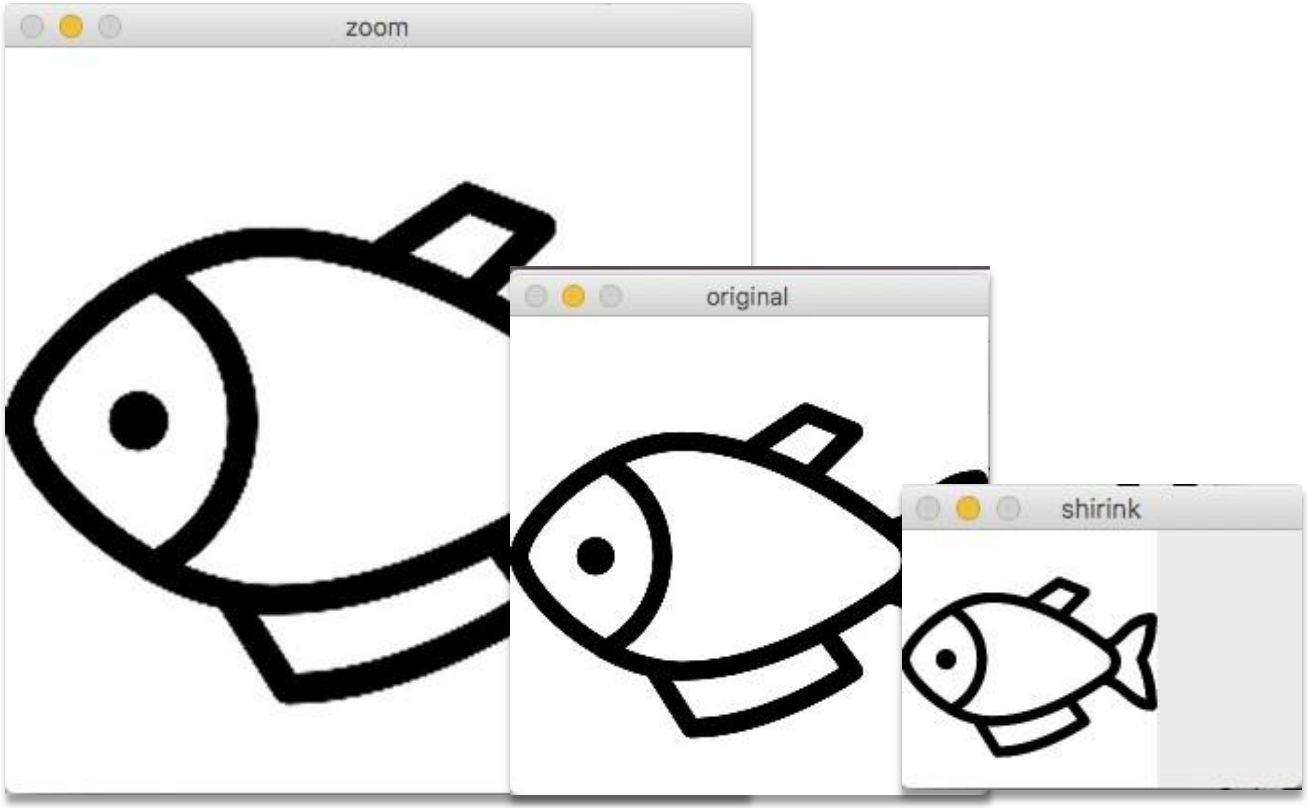
```
import cv2
import numpy as np
img = cv2.imread('../img/fish.jpg')
height, width = img.shape[:2]
#--① 크기 지정으로 축소
#dst1 = cv2.resize(img, (int(width*0.5), int(height*0.5)),\
                                # None, 0, 0, cv2.INTER_AREA)
dst1 = cv2.resize(img, (int(width*0.5), int(height*0.5)), \
                                interpolation=cv2.INTER_AREA)
#--② 배율 지정으로 확대
dst2 = cv2.resize(img, None, None, 2, 2, cv2.INTER_CUBIC)
#--③ 결과 출력
cv2.imshow("original", img)
cv2.imshow("small", dst1)
cv2.imshow("big", dst2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[예제 5-3] cv2.resize() 로 확대와 축소(scale_resize.py)

Geometric Transform

❖ Scaling

- cv2.resize 함수 Example <결과>



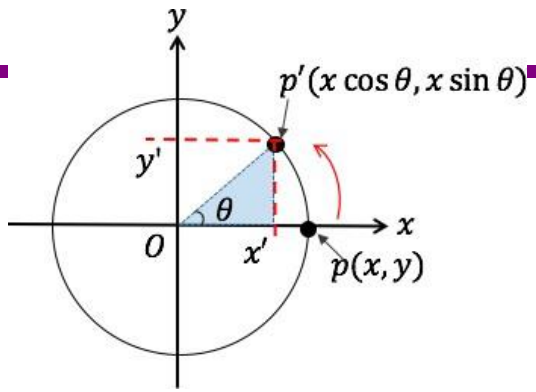
[그림 5-5] [5-3]의 실행 결과

Geometric Transform

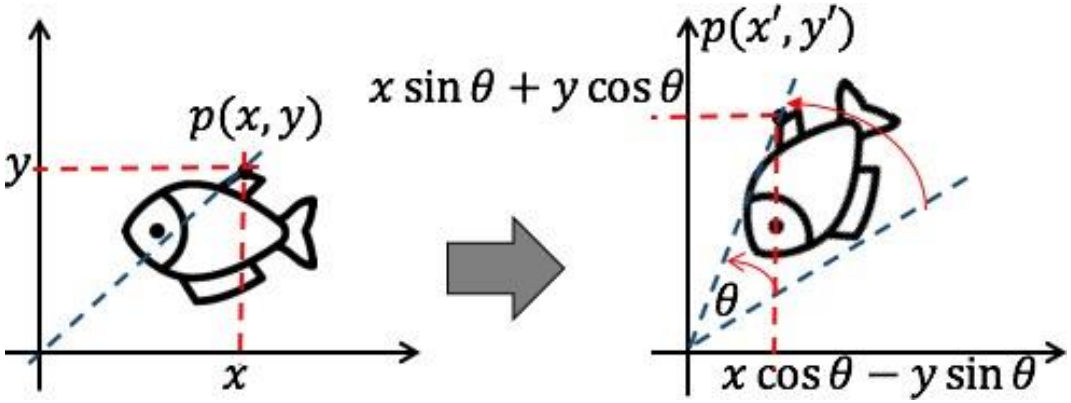
❖ Rotate

- 회전 변환 행렬

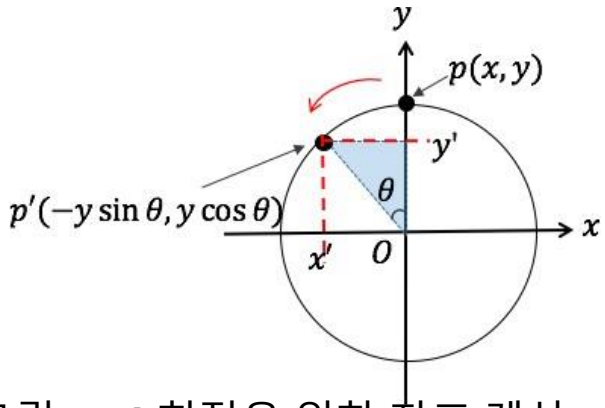
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



[그림 5-6] 회전을 위한 좌표 계산-1



[그림 5-8] 회전을 위한 좌표 계산-3



[그림 5-7] 회전을 위한 좌표 계산-2

Geometric Transform

❖ Rotate

- 변환 행렬

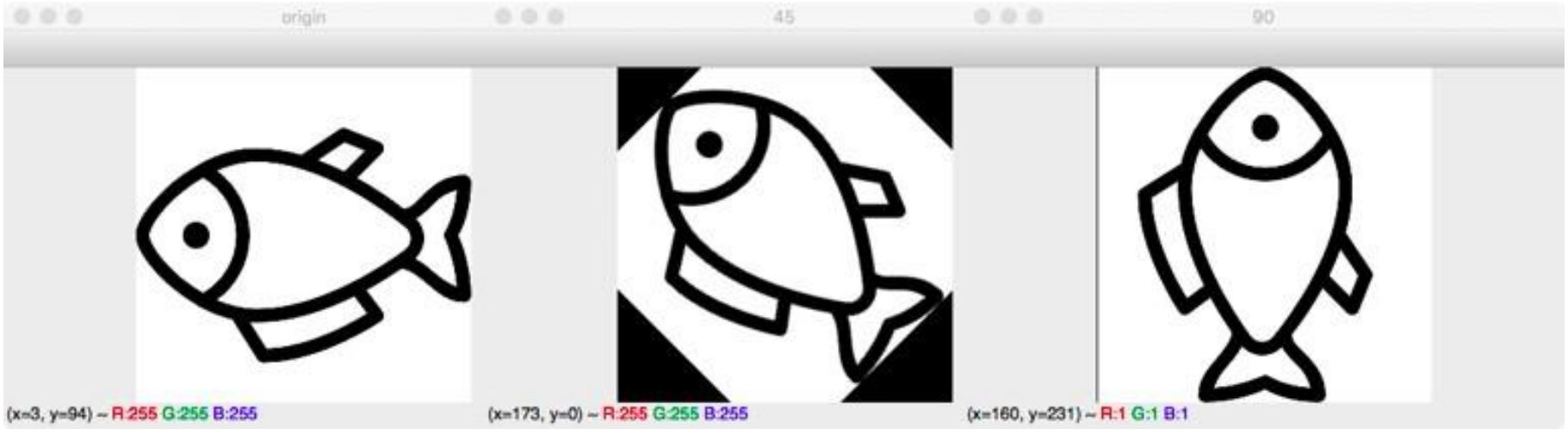
```
import cv2
import numpy as np
img = cv2.imread('../img/fish.jpg')
rows,cols = img.shape[0:2]
# ---① 라디안 각도 계산(60진법을 호도법으로 변경)
d45 = 45.0 * np.pi / 180 # 45도
d90 = 90.0 * np.pi / 180 # 90도
# ---② 회전을 위한 변환 행렬 생성
m45 = np.float32( [[ np.cos(d45), -1* np.sin(d45), rows//2],
                   [np.sin(d45), np.cos(d45), -1*cols//4]])
m90 = np.float32( [[ np.cos(d90), -1* np.sin(d90), rows],
                   [np.sin(d90), np.cos(d90), 0]])
# ---③ 회전 변환 행렬 적용
r45 = cv2.warpAffine(img,m45,(cols,rows))
r90 = cv2.warpAffine(img,m90,(rows,cols))
# ---④ 결과 출력
cv2.imshow("origin", img)
cv2.imshow("45", r45)
cv2.imshow("90", r90)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[예제 5-4] 변환행렬로 회전(rotate_matrix.py)

Geometric Transform

❖ Rotate

- 변환 행렬



[그림 5-9] [5-4]의 실행 결과

Geometric Transform

❖ Rotate

- 회전을 위한 변환 행렬 반환
- 회전 함수, 회전축, scale 지정 가능
- mtrx = cv2.getRotationMatrix2D(center, angle, scale)
  - center : 회전축 중심 좌표, 튜플(x, y)
  - angle : 회전 각도, 60진법
  - scale : 확대 축소 배율

$$\begin{bmatrix} \alpha & \beta & (1-\alpha)\cdot center.x - \beta \cdot center.y \\ -\beta & \alpha & \beta \cdot center.x + (1-\alpha)\cdot center.y \end{bmatrix}$$

$$\alpha = scale \cdot \cos\theta,$$
$$\beta = scale \cdot \sin\theta$$

Geometric Transform

❖ Rotate

- 회전 함수 Example

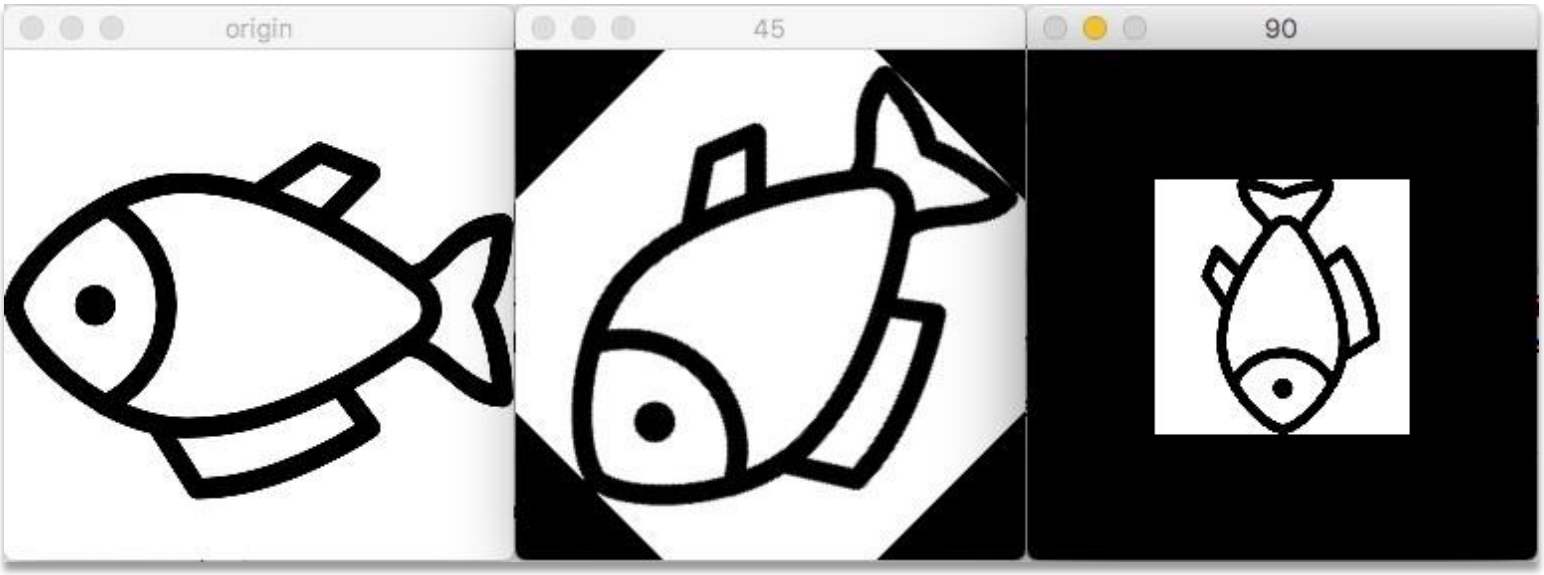
```
import cv2
img = cv2.imread('../img/fish.jpg')
rows,cols = img.shape[0:2]
#---① 회전을 위한 변환 행렬 구하기
# 회전축:중앙, 각도:45, 배율:0.5
m45 = cv2.getRotationMatrix2D((cols/2,rows/2),45,0.5)
# 회전축:중앙, 각도:90, 배율:1.5
m90 = cv2.getRotationMatrix2D((cols/2,rows/2),90,1.5)
#---② 변환 행렬 적용
img45 = cv2.warpAffine(img, m45,(cols, rows))
img90 = cv2.warpAffine(img, m90,(cols, rows))
#---③ 결과 출력
cv2.imshow('origin',img)
cv2.imshow("45", img45)
cv2.imshow("90", img90)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[예제 5-5] 회전 변환행렬 구하기(rotate_getmatrix.py)

Geometric Transform

❖ Rotate

- 회전 함수 Example <결과>



[그림 5-10] [5-5]의 실행 결과

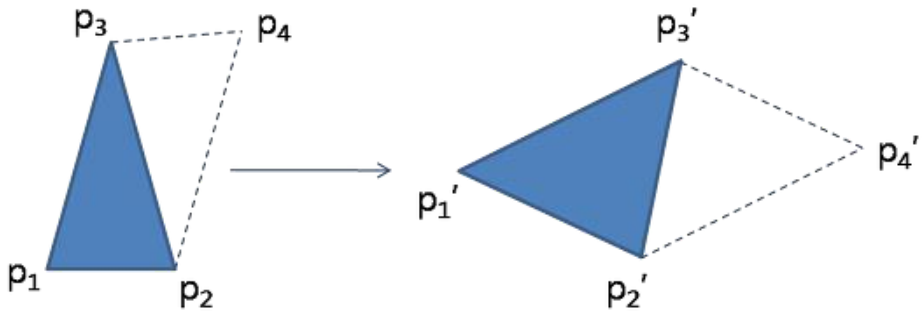
세부목차

1. Translate, Scaling, Rotate
2. **Warping**
3. Lens Distortion
4. Workshop

Warping

❖ Affine Transform

- 선의 평행성을 유지, 이미지 변환
- 이동, 확대, Scale 모두 포함
- 직선, 길이의 비율, 평행성을 보존하는 변환
- 원본 이미지의 3개의 좌표를 결과 이미지 좌표 3개에 매핑
- matrix = cv2.getAffineTransform(pts1, pts2)
  - pts1 : 변환 전 영상의 좌표 3개 , 3 x 2 NumPy 배열(float32)
  - pts2 : 변환 후 영상의 좌표 3개 , pts1과 동일
  - matrix : 변환 행렬 반환, 2 x 3 행렬



Warping

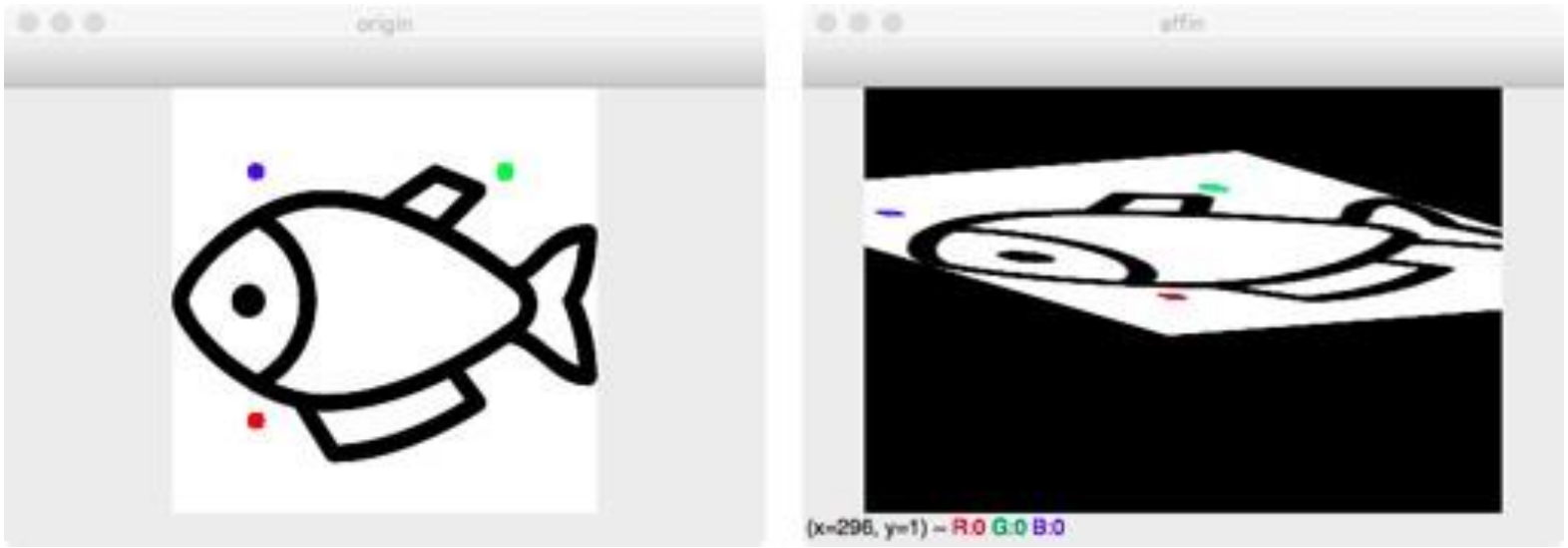
❖ Affine Transform

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
file_name = '../img/fish.jpg'
img = cv2.imread(file_name)
rows, cols = img.shape[:2]
# ---① 변환 전, 후 각 3개의 좌표 생성
pts1 = np.float32([[100, 50], [200, 50], [100, 200]])
pts2 = np.float32([[80, 70], [210, 60], [250, 120]])
# ---② 변환 전 좌표를 이미지에 표시
cv2.circle(img, (100,50), 5, (255,0), -1)
cv2.circle(img, (200,50), 5, (0,255,0), -1)
cv2.circle(img, (100,200), 5, (0,0,255), -1)
#---③ 짝지은 3개의 좌표로 변환 행렬 계산
mtrx = cv2.getAffineTransform(pts1, pts2)
#---④ 어핀 변환 적용
dst = cv2.warpAffine(img, mtrx, (int(cols*1.5), rows))
#---⑤ 결과 출력
cv2.imshow('origin',img)
cv2.imshow('affin', dst)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[예제 5-6] 어핀 변환(getAffine.py)

Warping

❖ Affine Transform



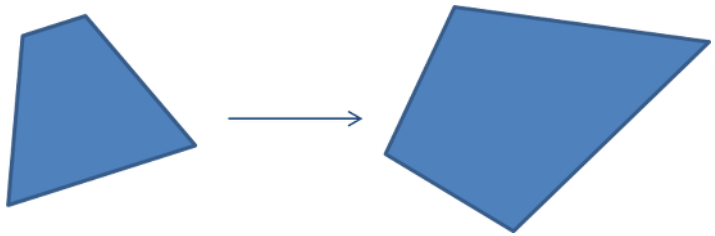
[그림 5-11] [5-6]의 실행 결과

Warping

❖ Perspective Transform

- 원근 변환
- Homography coordinate(동차 좌표) : 3차원 좌표계를 2차원 좌표계로 표시
- 3 x 3 변환 행렬
- 최소 4개의 매칭쌍 좌표
- mtrx = cv2.getPerspectiveTransform(pts1, pts2)
  - pts1 : 변환 이전 영상의 좌표 4개, 4 x 2 NumPy 배열(float32)
  - pts2 : 변환 이전 영상의 좌표 4개, pts1과 동일
  - mtrx : 변환 행렬 반환, 3 x 3 행렬
- dst = cv2.warpPerspective(src, mtrx, dsize [, dst, flags, borderMode, borderValue])
  - 3 x 3 행렬로 원근 변환 적용

$$w\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Warping

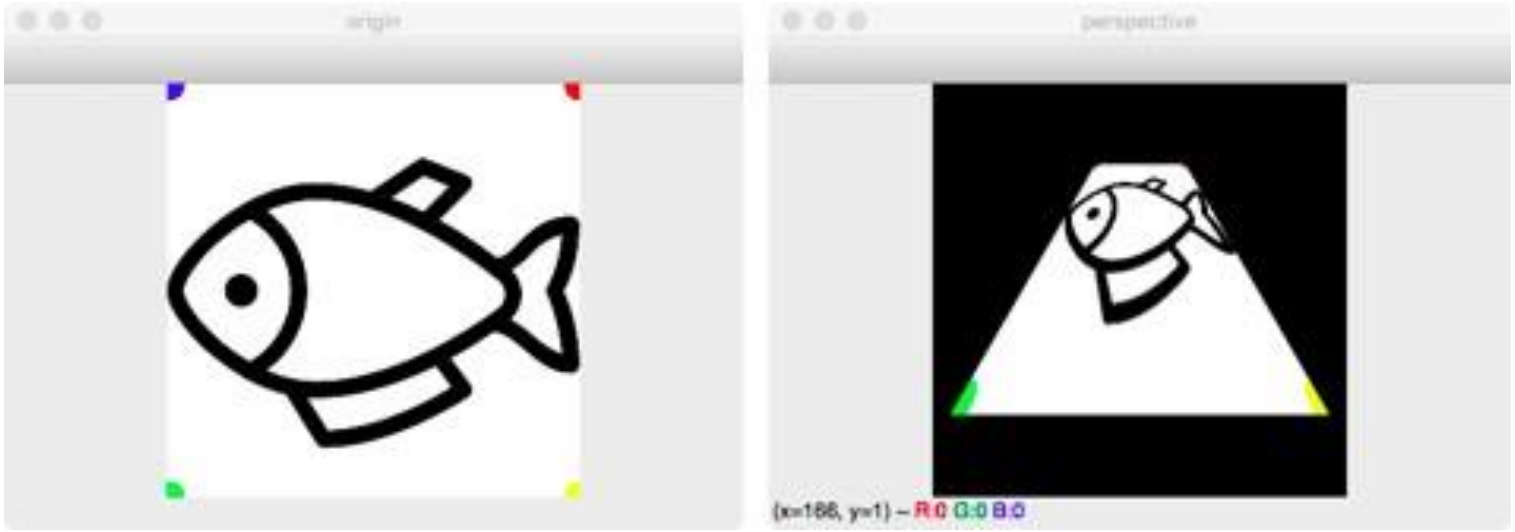
❖ Perspective Transform

[예제 5-7] 원근 변환(perspective.py)

```
import cv2
import numpy as np
file_name = "../img/fish.jpg"
img = cv2.imread(file_name)
rows, cols = img.shape[:2]
#---① 원근 변환 전 후 4개 좌표
pts1 = np.float32([[0,0], [0,rows], [cols, 0], [cols,rows]])
pts2 = np.float32([[100,50], [10,rows-50], [cols-100, 50], [cols-10,rows-50]])
#---② 변환 전 좌표를 원본 이미지에 표시
cv2.circle(img, (0,0), 10, (255,0,0), -1)
cv2.circle(img, (0,rows), 10, (0,255,0), -1)
cv2.circle(img, (cols,0), 10, (0,0,255), -1)
cv2.circle(img, (cols,rows), 10, (0,255,255), -1)
#---③ 원근 변환 행렬 계산
mtrx = cv2.getPerspectiveTransform(pts1, pts2)
#---④ 원근 변환 적용
dst = cv2.warpPerspective(img, mtrx, (cols, rows))
cv2.imshow("origin", img)
cv2.imshow('perspective', dst)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Warping

❖ Perspective Transform



[그림 5-12] [5-7]의 실행 결과

Warping

❖ Perspective Transform

- 문서 스캔 효과 내기 < 1/3 >

```
import cv2
import numpy as np
win_name = "scanning"
img = cv2.imread("../img/paper.jpg")
rows, cols = img.shape[:2]
draw = img.copy()
pts_cnt = 0
pts = np.zeros((4,2), dtype=np.float32)
def onMouse(event, x, y, flags, param): #마우스 이벤트 콜백 함수 구현 ---①
    global pts_cnt # 마우스로 찍은 좌표의 갯수 저장
    if event == cv2.EVENT_LBUTTONDOWN:
        cv2.circle(draw, (x,y), 10, (0,255,0), -1) # 좌표에 초록색 동그라미 표시
        cv2.imshow(win_name, draw)
        pts[pts_cnt] = [x,y] # 마우스 좌표 저장
        pts_cnt+=1
        if pts_cnt == 4: # 좌표가 4개 수집됨
```

[예제 5-8] 마우스와 원근 변환으로 문서 스캔 효과 내기(perspective_scan.py)

Warping

❖ Perspective Transform

- 문서 스캔 효과 내기 < 2/3 >

```
# 좌표 4개 중 상하좌우 찾기 ---②
 sm = pts.sum(axis=1) # 4쌍의 좌표 각각 x+y 계산
 diff = np.diff(pts, axis = 1) # 4쌍의 좌표 각각 y-x 계산
 topLeft = pts[np.argmin(sm)] # x+y가 가장 값이 좌상단 좌표
 bottomRight = pts[np.argmax(sm)] # x+y가 가장 큰 값이 우하단 좌표
 topRight = pts[np.argmin(diff)] # y-x가 가장 작은 것이 우상단 좌표
 bottomLeft = pts[np.argmax(diff)] # y-x가 가장 큰 값이 좌하단 좌표
 # 변환 전 4개 좌표
pts1 = np.float32([topLeft, topRight, bottomRight , bottomLeft])
# 변환 후 영상에 사용할 서류의 폭과 높이 계산 ---③
w1 = abs(bottomRight[0] - bottomLeft[0]) # 상단 좌우 좌표간의 거리
w2 = abs(topRight[0] - topLeft[0]) # 하당 좌우 좌표간의 거리
h1 = abs(topRight[1] - bottomRight[1]) # 우측 상하 좌표간의 거리
h2 = abs(topLeft[1] - bottomLeft[1]) # 좌측 상하 좌표간의 거리
width = max([w1, w2]) # 두 좌우 거리간의 최대값이 서류의 폭
height = max([h1, h2]) # 두 상하 거리간의 최대값이 서류의 높이
```

Warping

❖ Perspective Transform

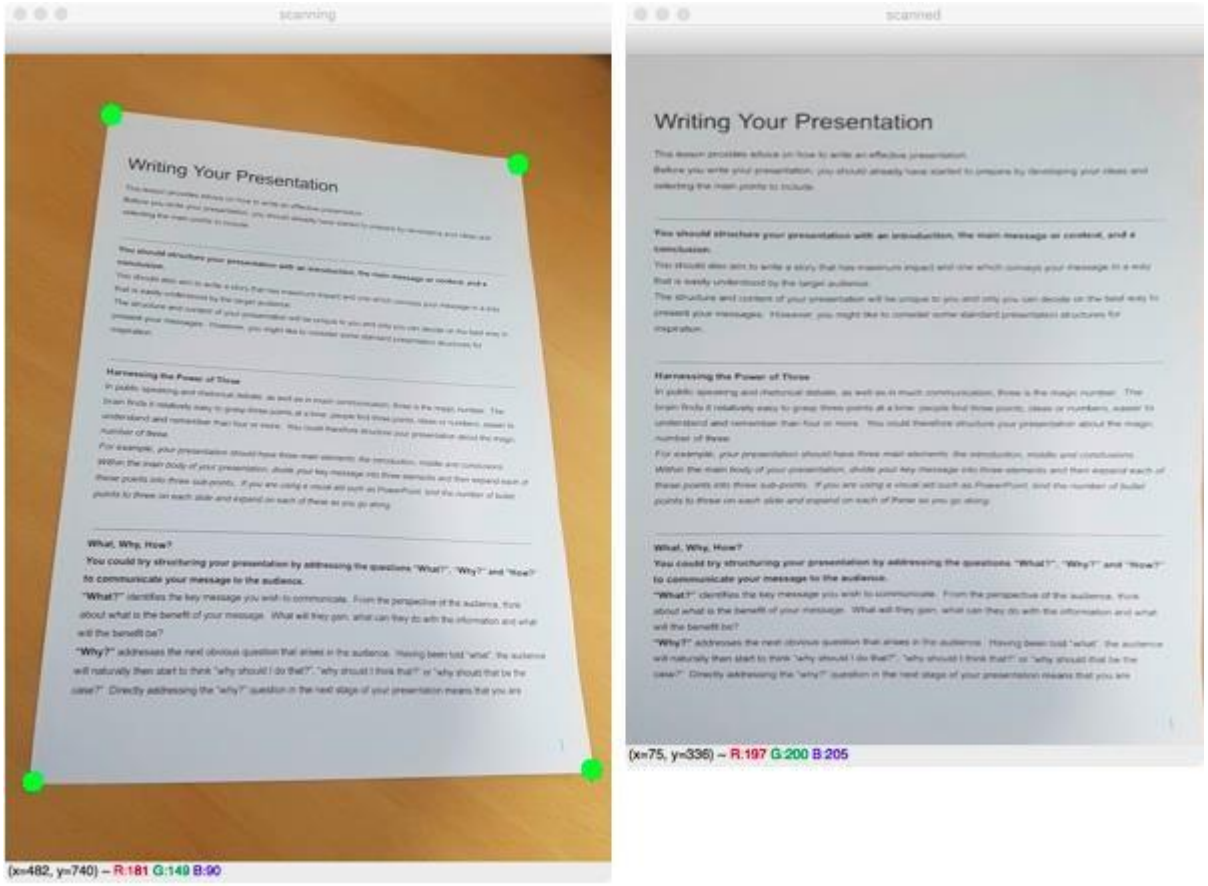
- 문서 스캔 효과 내기 < 3/3>

```
            # 변환 후 4개 좌표
            pts2 = np.float32([[0,0], [width-1,0],
                               [width-1,height-1], [0,height-1]])
            # 변환 행렬 계산
            mtrx = cv2.getPerspectiveTransform(pts1, pts2)
            # 원근 변환 적용
            result = cv2.warpPerspective(img, mtrx, (width, height))
            cv2.imshow('scanned', result)
cv2.imshow(win_name, img)
cv2.setMouseCallback(win_name, onMouse) # 마우스 콜백 함수를 GUI 윈도우에 등록 ---④
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Warping

❖ Perspective Transform

- 문서 스캔 효과 내기 < 결과 >

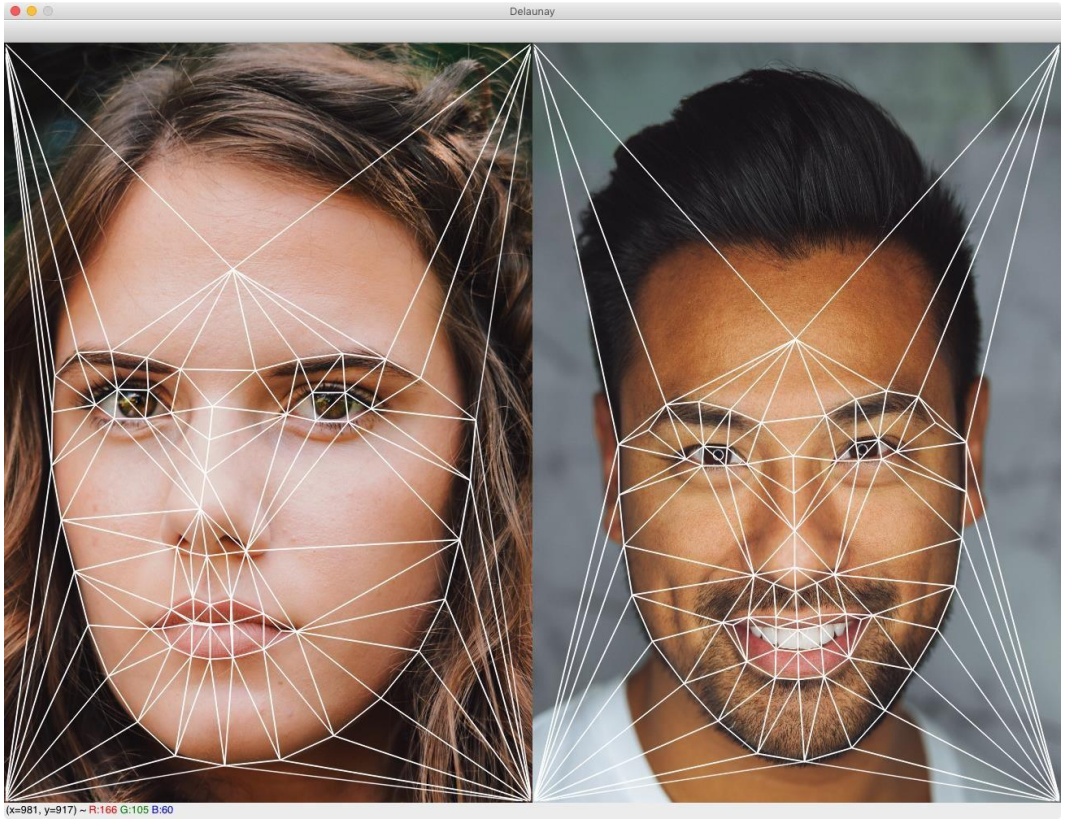


[그림 5-13] [5-8]의 실행 결과

Warping

❖ Triangluar Affine Transform

- 영상은 사각형 형태
- 삼각형 어핀 변환이 필요한 경우
  - 영상 왜곡
  - 영상 Morphing
  - Liquefy



[그림 5-14] 대응점으로 삼각 분할

Warping

❖ Triangluar Affine Transform

- OpenCV 제공 함수 없슴
- 삼각형 어핀 변환이 필요한 경우 절차
  1. 변환 전 삼각형 좌표 3쌍을 정한다.
  2. 변환 후 삼각형 좌표 3쌍을 정한다.
  3. 과정 1의 삼각형 좌표를 완전히 감싸는 외접 사각형 좌표를 구한다.
  4. 과정 3의 사각형 영역을 관심영역으로 지정한다.
  5. 과정 4의 관심 영역을 대상으로 과정 1과 과정 2의 좌표로 변환 행렬을 구하여 어핀 변환한다.
  6. 과정 5의 변환된 관심영역에서 과정 2의 삼각형 좌표만 마스킹 한다.
  7. 과정 6의 마스크를 이용해서 원본 또는 다른 영상에 합성한다.
- x,y,w,h = cv2.boundingRect(pts)pts : 다각형 좌표, 과정 3 필요
  - x, y, w, h : 외접 사각형의 좌표와 폭과 높이
- cv2.fillConvexPoly(img, points, color [, lineType])img : 입력 영상, 과정 6 필요
  - points : 다각형 꼭지점 좌표
  - color : 채우기에 사용할 색상
  - lineType : 선 그리기 알고리즘 선택 플래그

Warping

❖ Triangluar Affine Transform < 1/2>

```
import cv2
import numpy as np
img = cv2.imread("../img/taekwonv1.jpg")
img2 = img.copy()
draw = img.copy()
# 변환 전,후 삼각형 좌표 ---①
pts1 = np.float32([[188,14], [85,202], [294,216]])
pts2 = np.float32([[128,40], [85,307], [306,167]])
# 각 삼각형을 완전히 감싸는 사각형 좌표 구하기 ---②
x1,y1,w1,h1 = cv2.boundingRect(pts1)
x2,y2,w2,h2 = cv2.boundingRect(pts2)
# 사각형을 이용한 관심영역 설정 ---③
roi1 = img[y1:y1+h1, x1:x1+w1]
roi2 = img2[y2:y2+h2, x2:x2+w2]
# 관심영역을 기준으로 좌표 계산 ---④
offset1 = np.zeros((3,2), dtype=np.float32)
offset2 = np.zeros((3,2), dtype=np.float32)
for i in range(3):
    offset1[i][0], offset1[i][1] = pts1[i][0]-x1, pts1[i][1]-y1
    offset2[i][0], offset2[i][1] = pts2[i][0]-x2, pts2[i][1]-y2
```

[예제 5-9] 삼각형 어핀 변환(triangle_affine.py)

Warping

❖ Triangluar Affine Transform < 2/2>

```
# 관심 영역을 주어진 삼각형 좌표로 어핀 변환 ---⑤
mtrx = cv2.getAffineTransform(offset1, offset2)
warped = cv2.warpAffine( roi1, mtrx, (w2, h2), None, \
                                    cv2.INTER_LINEAR, cv2.BORDER_REFLECT_101)
# 어핀 변환 후 삼각형만 골라 내기 위한 마스크 생성 ---⑥
mask = np.zeros((h2, w2), dtype = np.uint8)
cv2.fillConvexPoly(mask, np.int32(offset2), (255))
# 삼각형 영역만 마스킹해서 합성 ---⑦
warped_masked = cv2.bitwise_and(warped, warped, mask=mask)
roi2_masked = cv2.bitwise_and(roi2, roi2, mask=cv2.bitwise_not(mask))
roi2_masked = roi2_masked + warped_masked
img2[y2:y2+h2, x2:x2+w2] = roi2_masked
# 관심 영역과 삼각형에 선 그려서 출력 ---⑧
cv2.rectangle(draw, (x1, y1), (x1+w1, y1+h1), (0,255,0), 1)
cv2.polylines(draw, [pts1.astype(np.int32)], True, (255,0,0), 1)
cv2.rectangle(img2, (x2, y2), (x2+w2, y2+h2), (0,255,0), 1)
cv2.imshow('origin', draw)
cv2.imshow('warped triangle', img2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Warping

❖ Triangluar Affine Transform

[그림 5-15] [5-9]의 실행 결과

세부목차

1. Translate, Scaling, Rotate
2. Warping
3. Lens Distortion
4. **Workshop**

Workshop

❖ Mosaic

- 마우스로 선택한 영역을 모자이크 처리하세요.
- 결과 예시

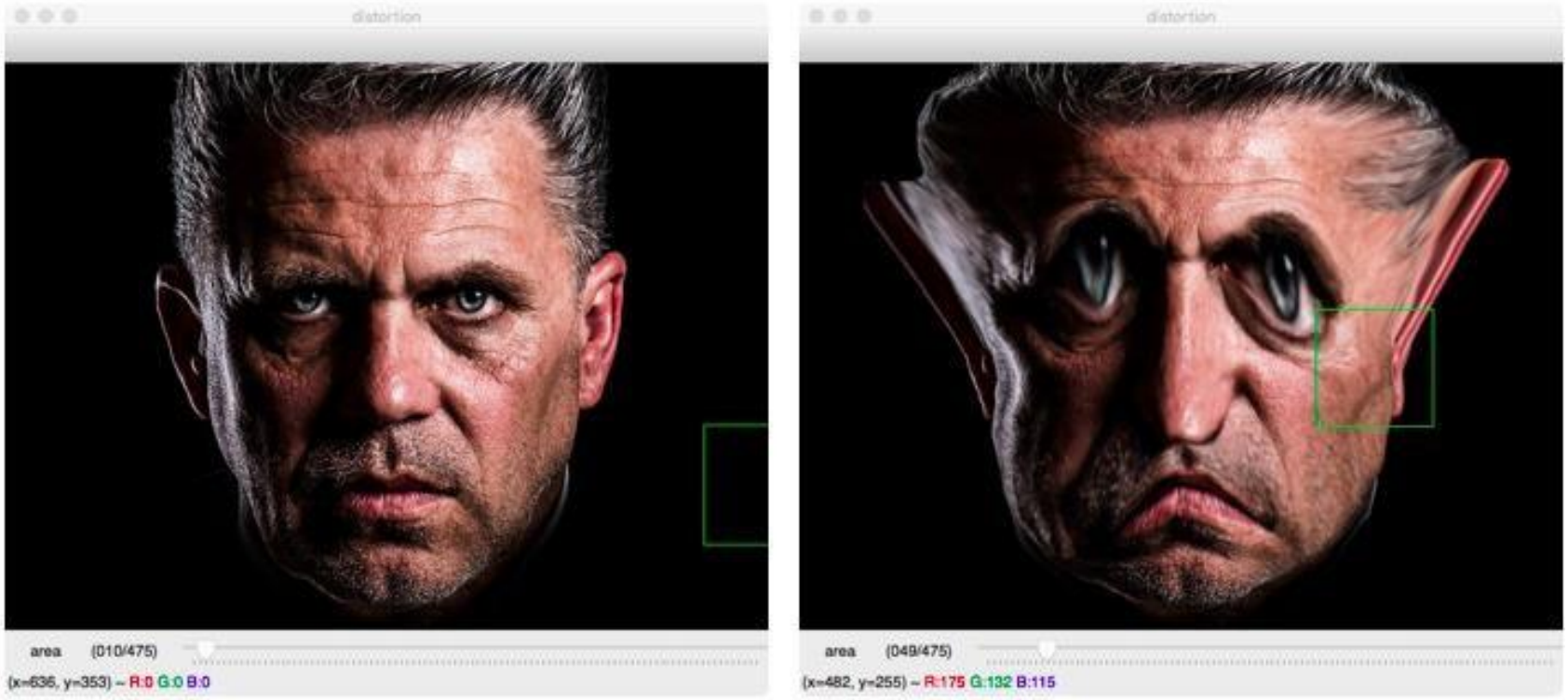
[그림 5-24] 모자이크 처리 예시

- 힌트
  - 선택 영역을 축소했다가 다시 확대하면 선명도가 떨어 집니다.
  - 보간법 중에 cv2.INTER_AREA를 선택하면 바둑판 모양 모자이크가 생성됩니다.

Workshop

❖ Liquefiy Tool

- 포토샵 리퀴파이 도구를 흉내내는 프로그램을 작성해 보세요.
- 마우스로 선택한 부분만 편집합니다.
- 결과 예시

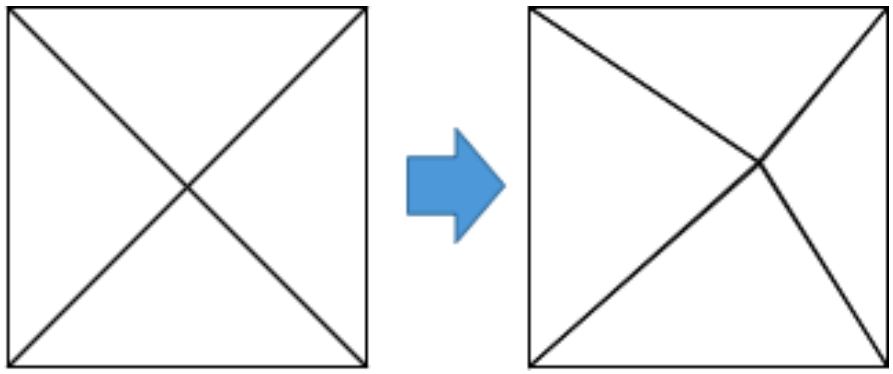


[그림 5-25] 리퀴파이 변환 사례

Workshop

❖ Liquefiy Tool

- 힌트
  - 마우스 좌표를 기준으로 일정한 크기의 사각형 영역을 만든다.
  - 사각형 중심을 기준으로 4개의 삼각형을 만든다.
  - 마우스를 드래그 한 만큼 원래의 중심점에서 이동한 중심점을 구한다.
  - 크기가 달라진 삼각형을 이용해서 삼각형 어핀 변환을 한다.



[그림 5-26] 4개의 삼각형 어핀 변환

Workshop

❖ Distortion Camera

- 마법 거울과 같은 효과를 내는 왜곡 카메라를 만드세요.
- 6개 효과
  - 원본, 좌우 대칭, 상하대칭
  - 물결, 볼록, 오목
- 결과 예시

[그림 5-27] 왜곡 거울 카메라 사례

Workshop

❖ Distortion Camera

- 힌트
  - Lenz Distotion 예제를 참고해서 6개 리매핑 한 결과를 하나의 영상으로 병합합니다
  - 리매핑 좌표는 한번만 만들고 프레임마다 적용만 하면 됩니다.