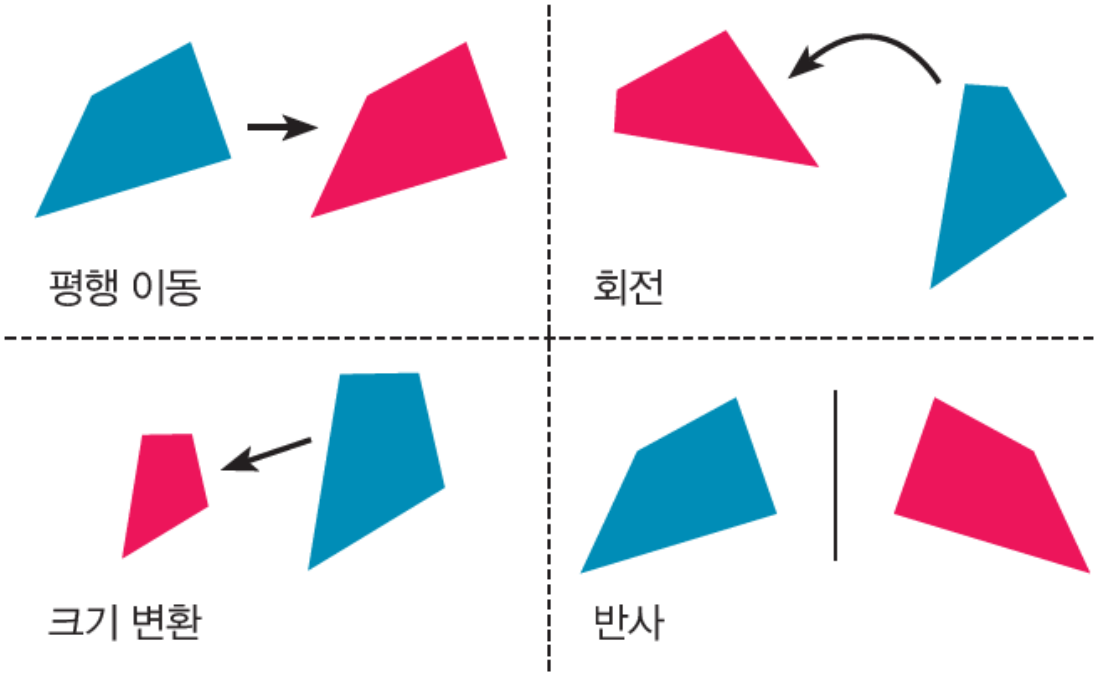
7장 기하학적 변환

(Geometric Transformation)

기하학적 변환

- 영상을 이동하거나 영상의 모양을 변형하는 처리



Forward Mapping

- 순방향 사상
  - 입력영상의 좌표에 해당하는 해당 목적영상의 좌표를 찾아서 화소값을 옮김

$$I(x, y) \rightarrow O(x', y')$$

$$x' = x + T_x$$
$$y' = y + T_y$$

평행이동

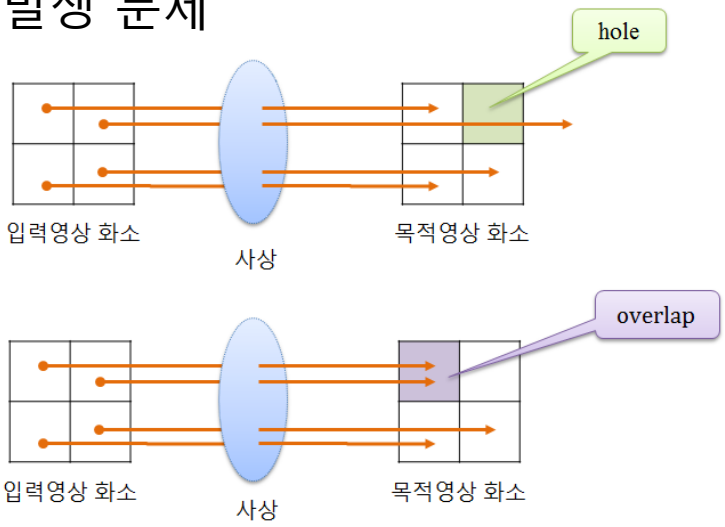
$$x' = x \cdot S_x$$
$$y' = y \cdot S_y$$

크기변환

$$x' = x \cdot \cos\theta - y \cdot \sin\theta$$
$$y' = x \cdot \sin\theta - y \cdot \cos\theta$$

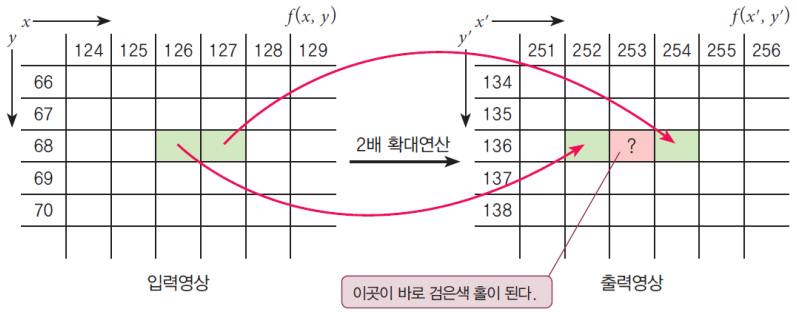
회전

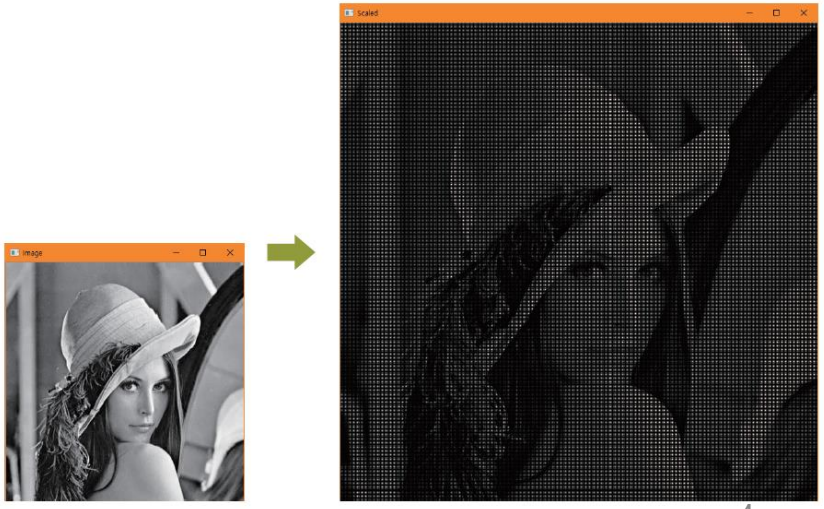
- Hole 이나 overlap 발생 문제



Forward Mapping

```
int main()
{
    Mat src = imread("d:/lenna.jpg", IMREAD_GRAYSCALE);
    Mat dst = Mat::zeros(Size(src.cols*2, src.rows*2), src.type());

    for (int y = 0; y < src.rows; y++) {
        for (int x = 0; x < src.cols; x++) {
            const int newX = x * 2;
            const int newY = y * 2;
            dst.at<uchar>(newY, newX) = src.at<uchar>(y, x);
        }
    }
    imshow("Image", src);
    imshow("Scaled", dst);
    waitKey(0);
    return 1;
}
```



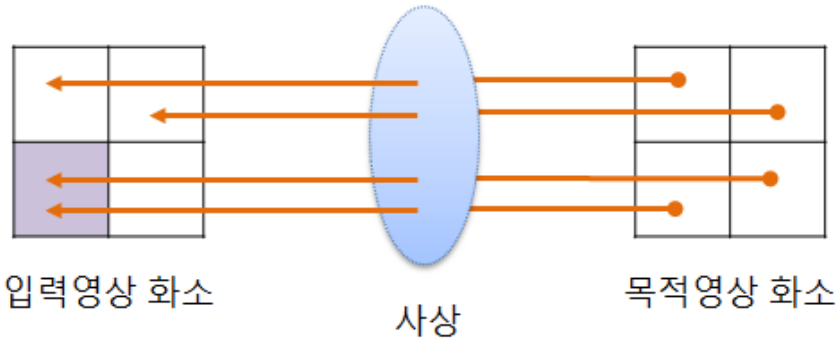


Reverse Mapping

- 역방향 사상
  - 목적영상의 좌표에 해당하는 입력 영상의 좌표를 찾아서 화소값을 옮김

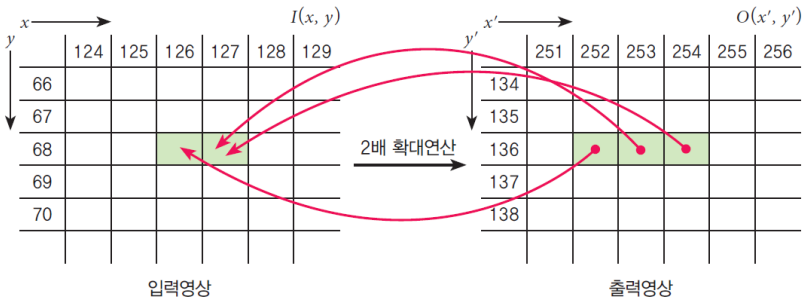
$$O(x, y) \leftarrow I(x', y')$$

  - 홀이나 오버랩은 발생하지 않음
  - 입력영상의 한 화소를 목적영상의 여러 화소에서 사용하게 되면, 결과 영상의 품질 저하



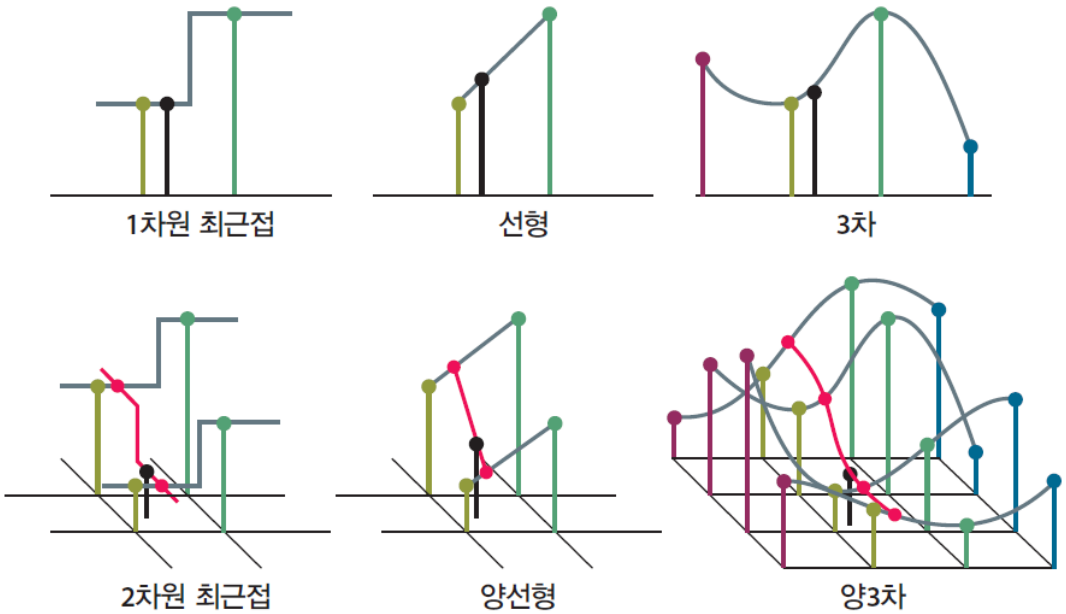
Reverse Mapping

```
int main()
{
    Mat src = imread("d:/lenna.jpg", IMREAD_GRAYSCALE);
    Mat dst = Mat::zeros(Size(src.cols*2, src.rows*2), src.type());

    for (int y = 0; y < dst.rows ; y++) {
        for (int x = 0; x < dst.cols; x++) {
                const int newX = x / 2.0;
                const int newY = y / 2.0;
                dst.at<uchar>(y, x) = src.at<uchar>(newY, newX);
        }
    }
    imshow("Image", src);
    imshow("Scaled", dst);
    waitKey(0);
    return 1;
}
```



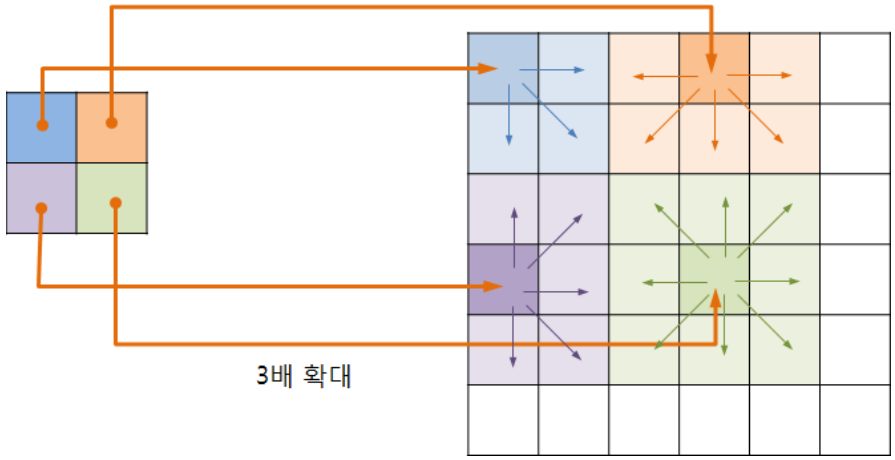
Interpolation

- 보간법
  - 주변 화소의 값을 사용하여 홀의 화소값 결정
  - 홀의 화소를 채우고, 오버랩이 되지 않게 화소 배치
- 종류



최근접 보간법 (nearest neighbor interpolation)

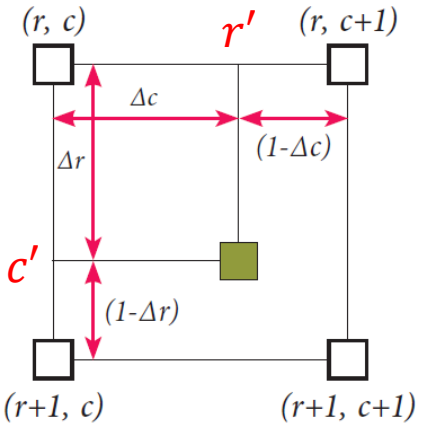
- 변환된 위치와 가장 가까운 화소값을 사용하는 방법



- 확대비율이 커지면, 모자이크 현상 혹은 경계부분에서 계단현상 발생

양선형 보간법 (Bilinear interpolation)

- 이미 알고 있는 4개의 인접 화소의 값을 이용 c′cccc′



$$
\begin{aligned}
O(r', c') = \mathbf{I}(r, c) \cdot (1 - \Delta r) \cdot (1 - \Delta c) \\
+ \mathbf{I}(r + 1, c) \cdot \Delta r \cdot (1 - \Delta c) \\
+ \mathbf{I}(r, c + 1) \cdot (1 - \Delta r) \cdot \Delta c \\
+ \mathbf{I}(r + 1, c + 1) \cdot \Delta r \cdot \Delta c
\end{aligned}
$$

양선형 보간법

```
float Lerp(float s, float e, float t) {
   return s + (e - s) * t;
}

float Blerp(float c00, float c10, float c01, float c11, float tx, float ty) {
    return Lerp(Lerp(c00, c10, tx), Lerp(c01, c11, tx), ty);
}

float GetPixel(Mat img, int x, int y)
{
    if (x > 0 && y > 0 && x < img.cols && y < img.rows)
        return (float)(img.at<uchar>(y, x));
   else
        return 0.0;
}
```

```
int main()
{
    Mat src = imread("d:/lenna.jpg", IMREAD_GRAYSCALE);
    Mat dst = Mat::zeros(Size(src.cols*2, src.rows*2), src.type());

    for (int y = 0; y < dst.rows ; y++) {
        for (int x = 0; x < dst.cols; x++) {
            float gx = ((float)x) / 2.0;
            float gy = ((float)y) / 2.0;
            int gxi = (int)gx;
            int gyi = (int)gy;
            float c00 = GetPixel(src, gxi, gyi);
            float c10 = GetPixel(src, gxi + 1, gyi);
            float c01 = GetPixel(src, gxi, gyi + 1);
            float c11 = GetPixel(src, gxi + 1, gyi + 1);
            int value = (int)Blerp(c00, c10, c01, c11, gx - gxi, gy - gyi);
            dst.at<uchar>(y, x) = value;
        }
}
```

최근접보간법 vs. 양선형 보간법

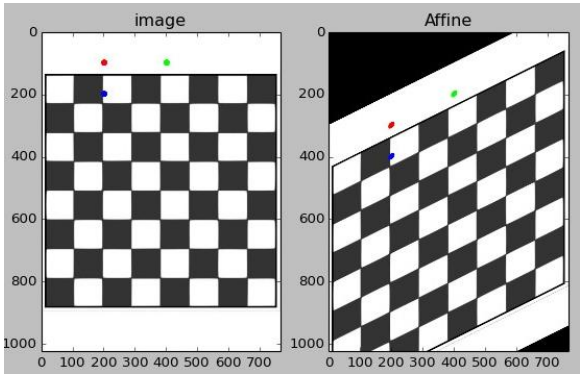
입력영상

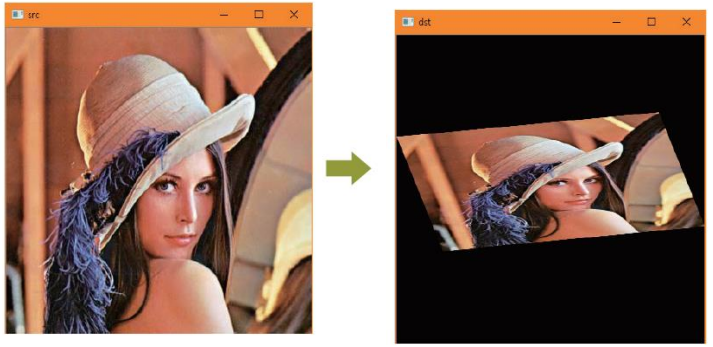
확대

계단 현상

Affine Transformation

- 어파인 변환
  - 선의 직선성과 평행성이 유지되는 영상의 기하학적 변환
  - 회전, 크기변환(확대/축소), 평행이동, 반사 등





Affine Transformation

- 변환 행렬
  - 2 x 3

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

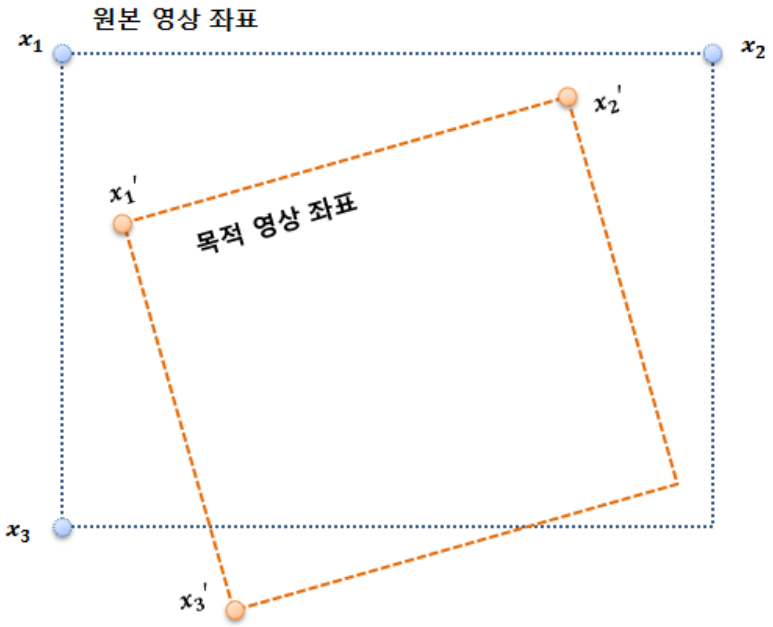
(ex) 회전 $\begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \end{bmatrix}$

(ex) 크기변경 $\begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \end{bmatrix}$

(ex) 평행이동 $\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$

Affine Transformation

- 3점을 이용한 변환행렬 계산
  - 입력영상의 좌표 3개(x_1 , x_2 , x_3)와 목적영상에서 상응하는 좌표 3개(x'_1, x'_2, x'_3)를 알면 → 어파인 행렬 구성 가능



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

✓ 미지수: 6개

✓ 점3개 ☞ 방정식 6개

Affine Transformation

- OpenCV 함수

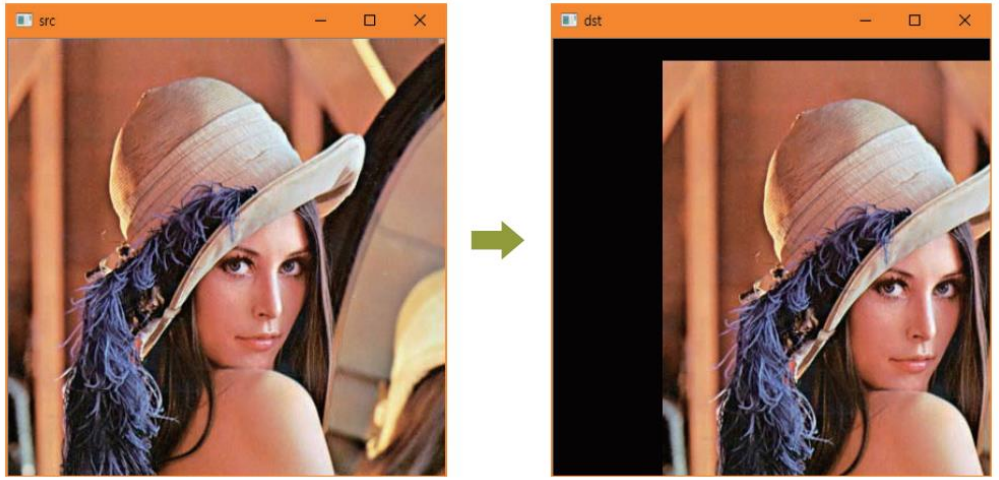
함수 및 인수 구조	
void warpAffine(InputArray src, OutputArray dst, InputArray M, Size dsize, int flags = INTER_LINEAR, int borderMode = BORDER_CONSTANT, const Scalar& borderValue = Scalar())	
Mat getAffineTransform(InputArray src, InputArray dst)	
Mat getAffineTransform(const Point2f src[], const Point2f dst[])	
Mat getRotationMatrix2D(Point2f center, double angle, double scale)	
함수 및 인수	**설명**
void warpAffine()	입력영상에 어파인 변환을 수행해서 반환한다.
• InputArray src	입력영상
• OutputArray dst	반환영상
• InputArray M	어파인 변환 행렬
• Size dsize	반환 영상의 크기
• int flags	보간 방법
• int borderMode	경계지정 방법
Mat getAffineTransform()	3개의 좌표쌍을 입력하면 어파인 변환 형렬을 반환한다.
• InputArray src	입력영상 좌표 3개 (행렬로 구성)
• OutputArray dst	목적영상 좌표 3개 (행렬로 구성)
• Point2f src[]	입력영상 좌표 3개 (배열로 구성)
• Point2f dst[]	목적영상 좌표 3개 (배열로 구성)
Mat getRotationMatrix2D()	회전 변환과 크기 변경을 수행할 수 있는 어파인 행렬을 반환한다.
• Point2f center	회전의 중심점
• double angle	회전 각도, 양수 각도가 반시계 방향 회전 수행
• double scale	변경할 크기

INTER_NEAREST = 0, // 최근접 보간법
INTER_LINEAR = 1, // 양선형 보간법
INTER_CUBIC = 2, // 3차 보간법
INTER_AREA = 3,
INTER_LANCZOS4 = 4,
INTER_MAX = 7,
WARP_FILL_OUTLIERS = 8,
WARP_INVERSE_MAP = 16

Affine Transformation - 평행이동

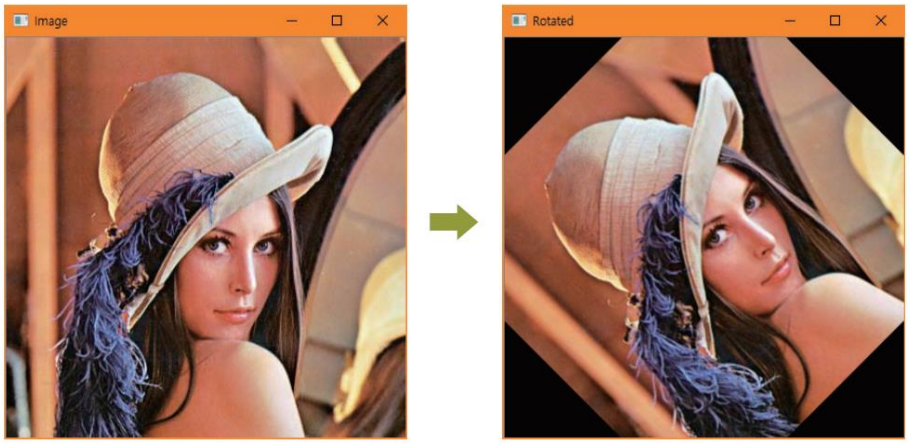
```
int main()
{
    Mat src = imread("d:/lenna.jpg", IMREAD_COLOR);
    Mat dst = Mat();

    int tx = 100;
    int ty = 20;

    Mat tmat = (Mat_<double>(2, 3) << 1, 0, tx, 0, 1, ty);

    warpAffine(src, dst, tmat, src.size());

    imshow("src", src);
    imshow("dst", dst);
    waitKey(0);
    return 1;
}
```

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$



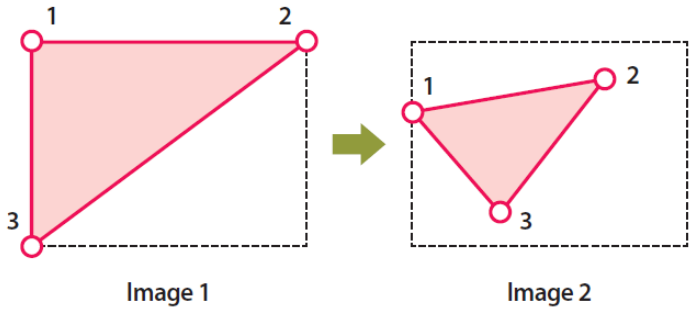
Affine Transformation - 회전

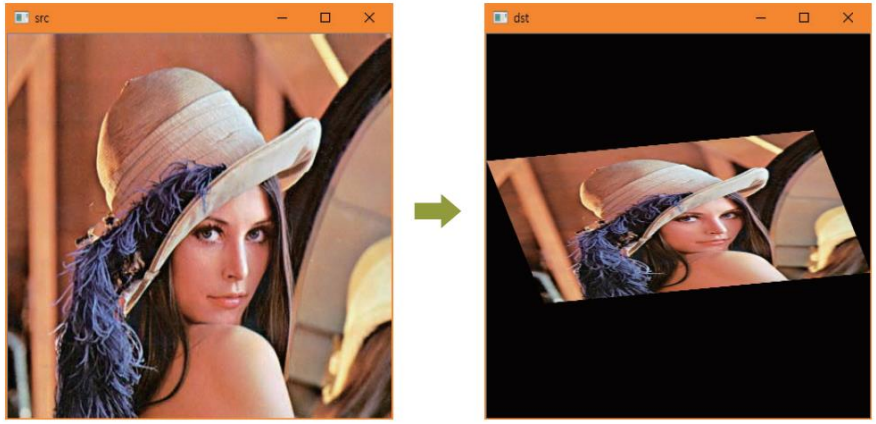
```
int main()
{
    Mat src = imread("d:/lenna.jpg", IMREAD_COLOR);
    Mat dst = Mat();
    Size dsize = Size(src.cols, src.rows);

    Point center = Point(src.cols / 2.0, src.rows / 2.0);
    Mat M = getRotationMatrix2D(center, 45, 1.0);

    warpAffine(src, dst, M, dsize, INTER_LINEAR);

    imshow("Image", src);
    imshow("Rotated", dst);
    waitKey(0);
    return 1;
}
```



Affine Transformation - 3점 이용 변환

```
int main()
{
    Mat src = imread("d:/lenna.jpg", IMREAD_COLOR);
    Point2f srcTri[3];
    Point2f dstTri[3];
    Mat warp_mat(2, 3, CV_32FC1);

    Mat warp_dst;
    warp_dst = Mat::zeros(src.rows, src.cols, src.type());
    srcTri[0] = Point2f(0, 0);
    srcTri[1] = Point2f(src.cols - 1.0f, 0);
    srcTri[2] = Point2f(0, src.rows - 1.0f);
    dstTri[0] = Point2f(src.cols*0.0f, src.rows*0.33f);
    dstTri[1] = Point2f(src.cols*0.85f, src.rows*0.25f);
    dstTri[2] = Point2f(src.cols*0.15f, src.rows*0.7f);
    warp_mat = getAffineTransform(srcTri, dstTri);
    warpAffine(src, warp_dst, warp_mat, warp_dst.size());

    imshow("src", src);
    imshow("dst", warp_dst);
    waitKey(0);
    return 1;
}
```





Affine Transformation – 크기 변환

- 크기 변환 함수

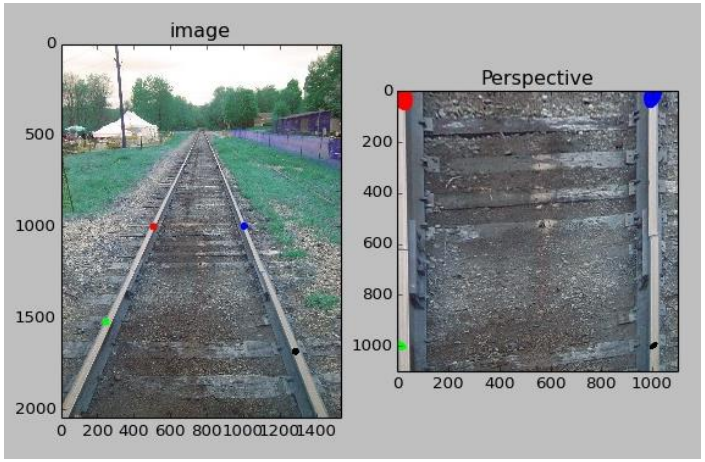
```
resize (src, dst, dsize, fx = 0, fy = 0, interpolation = cv.INTER_LINEAR )
```

매개 변수	설명
src	입력 영상
dst	출력 영상
dsize	출력 영상 크기. 0이면 다음과 같이 계산된다. dsize=Size(round(fx*src.cols), round(fy*src.rows)
fx	x축 상의 배율. 만약 0이면 (double)dsize. width/src.cols로 계산된다.
fy	y축 상의 배율. 만약 0이면 (double)dsize. height/src.rows로 계산된다.
interpolation	보간법

```
int main()
{
    Mat src = imread("d:/lenna.jpg", IMREAD_COLOR);
    Mat dst = Mat();

    resize(src, dst, Size(), 2.0, 2.0);

    imshow("Image", src);
    imshow("Scaled", dst);
    waitKey(0);
    return 1;
}
```

Perspective Transformation

- 원근 변환 / 투시 변환
  - 선의 직선성만 유지되는 영상의 기하학적 변환

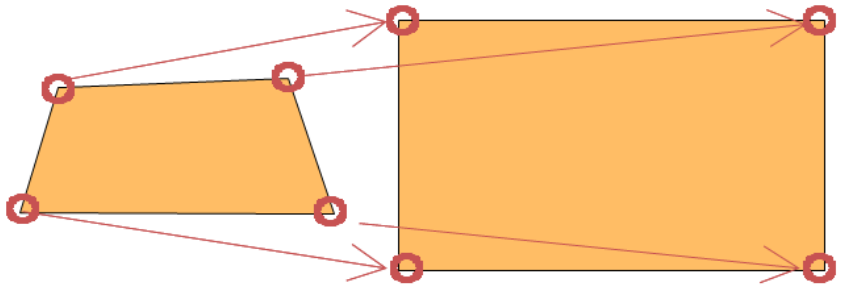


Perspective Transformation

- 변환행렬
  - 3 x 3 행렬

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

  - 4 점을 이용한 변환 행렬 계산

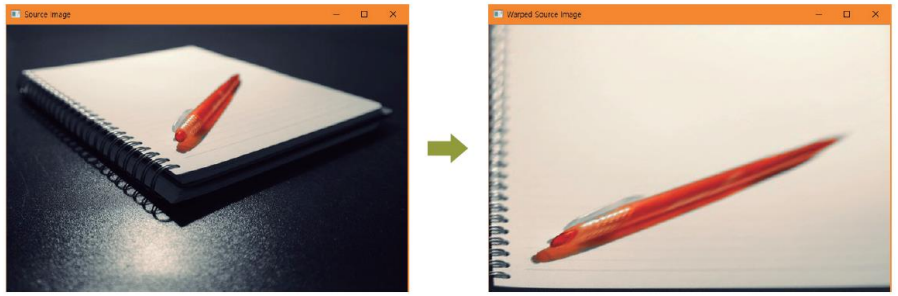


Perspective Transformation

- OpenCV 함수
  - cv::getPerspectiveTransform() 함수
    - 4개의 좌표쌍으로부터 원근변환 행렬을 계산
  - cv::warpPerspective() 함수
    - 원근변환 행렬에 따라서 원근변환 수행

Perspective Transformation

```
int main()
{
    Mat src = imread("d:/book.jpg");

    Point2f inputp[4];
    inputp[0] = Point2f(30, 81);
    inputp[1] = Point2f(274, 247);
    inputp[2] = Point2f(298, 40);
    inputp[3] = Point2f(598, 138);

    Point2f outputp[4];
    outputp[0] = Point2f(0, 0);
    outputp[1] = Point2f(0, src.rows);
    outputp[2] = Point2f(src.cols, 0);
    outputp[3] = Point2f(src.cols, src.rows);

    Mat h = getPerspectiveTransform(inputp, outputp);

    Mat out;
    warpPerspective(src, out, h, src.size());

    imshow("Source Image", src);
    imshow("Warped Source Image", out);

    waitKey(0);
}
```



HW

1. 사용자로부터 3개의 점을 받아서 어파인 변환하는 프로그램을 작성하라. 마우스를 이용하여 점들의 초기위치와 변환 후의 위치를 입력할 수 있도록 한다.

2. 사용자로부터 4개의 점을 받아서 원근 변환하는 프로그램을 작성하라. 마우스를 이용하여 점들의 초기위치와 변환 후의 위치를 입력할 수 있도록 한다.