Multimedia (Lab 06)

Spring, 2020

Department of Software
Yong Ju Jung (정용주)

Summary

- In this lab, you will learn about
  - Image interpolation techniques
  - Image warping by Affine transformations
  - Image warping by Perspective transformations

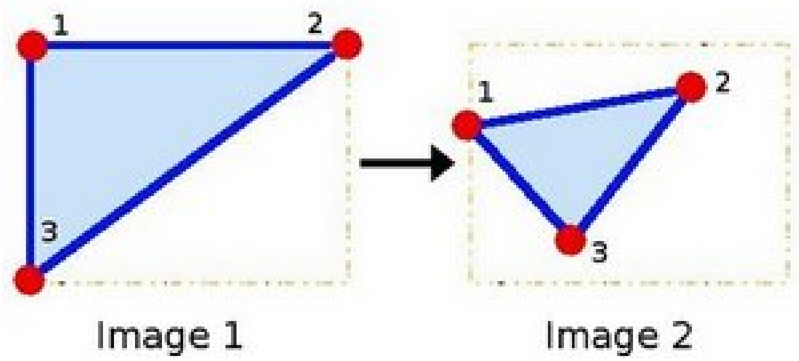
[Lab 06-1] Image warping

parallel, straight line 유지.

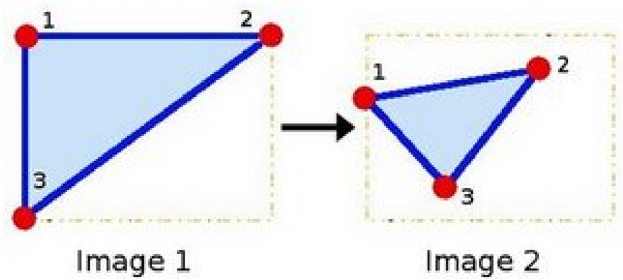
- Apply **affine transformations**

How to get an Affine transform matrix?

- The information for an Affine transform can come as a geometric relation between points.
  - See the below figure where the points 1, 2 and 3 (forming a triangle in image 1) are mapped into image 2.
  - If we find the Affine Transformation with these 3 points, then we can apply this found relation to the whole pixels in the image.



- Given a 3 corresponding points between (x,y) and (x', y'):
  - (1, 1) → (1, 3)
  - (6, 1) → (4, 2)
  - (1, 5) → (3, 4)



- How can we obtain the Affine (M) matrix?
  - Put these points in a matrix form: r = Mp
    - Now, r and p are 3x3 matrix.
    - Also, M is a 3x3 matrix.
  - What is the inverse of p?
  - Solve for M by multiplying p^{-1} by r.

$$M = \begin{bmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \\ 0 & 0 & 1 \end{bmatrix}$$

- To obtain the M matrix, we can simply use an OpenCV library:
  - warp_mat = **getAffineTransform**(srcTri, dstTri);

Exercise

- Loads an image
- Applies an Affine Transform to the image.
  - As noted in previous slides, this transform is obtained from the relation between three points.
    - warp_mat = **getAffineTransform**(srcTri, dstTri);
  - We use the function warpAffine for that purpose.
    - **warpAffine**(src, warp_dst, warp_mat, warp_dst.size());
- Display the input and output images

```
Point2f srcTri[3];
Point2f dstTri[3];

Mat warp_mat( 2, 3, CV_32FC1 );
Mat src, warp_dst;

/// Load the image
src = imread( argv[1], 1 );

/// Set the dst image the same type and size as src
warp_dst = Mat::zeros( src.rows, src.cols, src.type() );
```

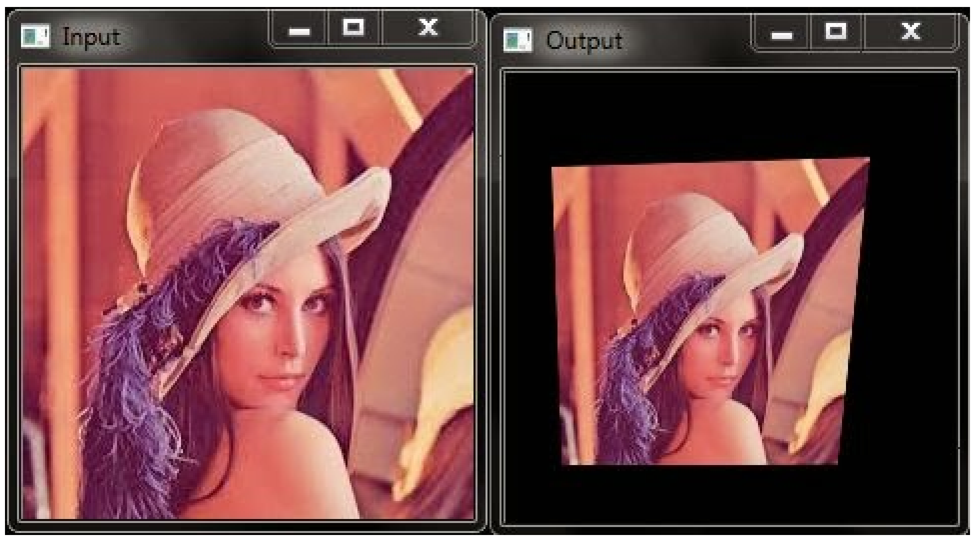
001은 무시해서 2*3이고 연산할 때는 001을 고려해준 것

```
/// Set your 3 points to calculate the  Affine Transform
srcTri[0] = Point2f( 0,0 );
srcTri[1] = Point2f( src.cols - 1, 0 );
srcTri[2] = Point2f( 0, src.rows - 1 );

dstTri[0] = Point2f( src.cols*0.0, src.rows*0.33 );
dstTri[1] = Point2f( src.cols*0.85, src.rows*0.25 );
dstTri[2] = Point2f( src.cols*0.15, src.rows*0.7 );

/// Get the Affine Transform
warp_mat = getAffineTransform( srcTri, dstTri );
```

```
/// Apply the Affine Transform just found to the src image
warpAffine( src, warp_dst, warp_mat, warp_dst.size() );
```

[Lab 06-2] Image warping

- Apply **perspective transformations**



Perspective Transform in OpenCV

- Mat **getPerspectiveTransform**(InputArray src, InputArray dst)
  - Calculates a perspective transform from four pairs of the corresponding points.

$$M = \begin{bmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \\ a_5 & a_6 & 1 \end{bmatrix}$$

- void **warpPerspective**(InputArray src, OutputArray dst, InputArray M, Size dsize, int flags=INTER_LINEAR, int borderMode=BORDER_CONSTANT, const Scalar& borderValue=Scalar())
  - Applies a perspective transformation to an image.
  - Parameters:
    - **src** – input image.
    - **dst** – output image that has the size dsize and the same type as src .
    - **M** – 3 x 3 transformation matrix.
    - **dsize** – size of the output image.
    - **flags** – combination of interpolation methods (INTER_LINEAR or INTER_NEAREST).
    - **borderMode** – pixel extrapolation method (BORDER_CONSTANT or BORDER_REPLICATE).
    - **borderValue** – value used in case of a constant border; by default, it equals 0.

```
// Input points
Point2f inputQuad[4];
// Corresponding points
Point2f outputQuad[4];

// warp Matrix 3*3으로 수정.
Mat warp_mat( 2, 4, CV_32FC1 );
Mat input, output;

//Load the image
input = imread( "lena.jpg", 1 );
output = Mat::zeros( input.rows, input.cols, input.type() );

// These four pts are the sides of the rect box used as input (from top-left in clockwise order)
inputQuad[0] = Point2f( -30,-60 );
inputQuad[1] = Point2f( input.cols+50,-50);
inputQuad[2] = Point2f( input.cols+100,input.rows+50);
inputQuad[3] = Point2f( -50,input.rows+50  );

// The 4 points where the mapping is to be done , from top-left in clockwise order
outputQuad[0] = Point2f( 0,0 );
outputQuad[1] = Point2f( input.cols-1,0);
outputQuad[2] = Point2f( input.cols-1,input.rows-1);
outputQuad[3] = Point2f( 0,input.rows-1  );

// Get the Perspective Transform Matrix i.e. lambda
warp_mat = getPerspectiveTransform( inputQuad, outputQuad );

// Apply the Perspective Transform just found to the src image
warpPerspective(input, output, warp_mat, output.size() );
```

Interpolation

- **[Lab 06-3]** Nearest neighbour interpolation
  - Inputs
    - input.png / 512×512
  - Outputs
    - Display the scaled image which is interpolated from the input data
    - Save it as "result.png" imwrite()함수로 저장.
  - Write program using nearest neighbour interpolation
  - Test it by using input.png with x and y scaling factors

```
Lab05-1.exe input.png result.png 1.5 1.5
```

스케일 1.5배,

Interpolation

- [**Lab 06-4**] Bilinear interpolation
  - Extend Lab 05-1 to use bilinear interpolation
  - Use 2D separable interpolation and boundary mirroring
- [**Lab 06-5**] Bicubic interpolation
  - Extend Lab 05-2 to use bicubic interpolation

- Compare outputs for various scaling factors
  - 2.0 / 2.0
  - 1.5 / 1.5
  - 3.0 / 2.0

Resize using OpenCV Library

- Compare your result with the one obtained by using OpenCV library
  - resize(src, dst, Size(), 1.5, 1.5, INTER_NEAREST);
  - resize(src, dst, Size(), 1.5, 1.5, INTER_LINEAR);
  - resize(src, dst, Size(), 1.5, 1.5, INTER_CUBIC);

Enumerator	
INTER_NEAREST	nearest neighbor interpolation
INTER_LINEAR	bilinear interpolation
INTER_CUBIC	bicubic interpolation