# ImgProcBasics

Jane Doe

4/9/2022

# Table of contents

# Preface

This is a Quarto book.

To learn more about Quarto books visit https://quarto.org/docs/books.

# 1 Overview

1. Geometric Transformations
2. Linear filtering
3. Image binarization
    1. Histogram equalization
    2. Thresholding
    3. Adaptive thresholding
    4. Otsu's method
4. Morphological operations

# 2 Geometric Transformations

- Rotations
- Affine transformations (skew)
- Perspective transformations

A function $T(\dot{})$ which transforms a pixel's original location $\begin{bmatrix} x \\ y \end{bmatrix}$ into the destination location $\begin{bmatrix} x' \\ y' \end{bmatrix}$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = T\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)$$

## 2.1 Comparison

To Put image

What stays unchanged?

Rotation:

- Lines remain lines, parallel remains parallel, angles are same, distances are same

Affine (skew):

- Lines remain lines, parallel remains parallel, angles change, distances change

Perspective:

- Lines remain lines, angles change, distances change, parallel remains parallel along one direction, converge to a point along other direction

## 2.2 Rotations

Image

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

- This includes a translation with $\begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$.
- Alternatively, we can drop $\begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$, and we can pick the origin $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$ where we want

### 2.2.1 Alternate equation

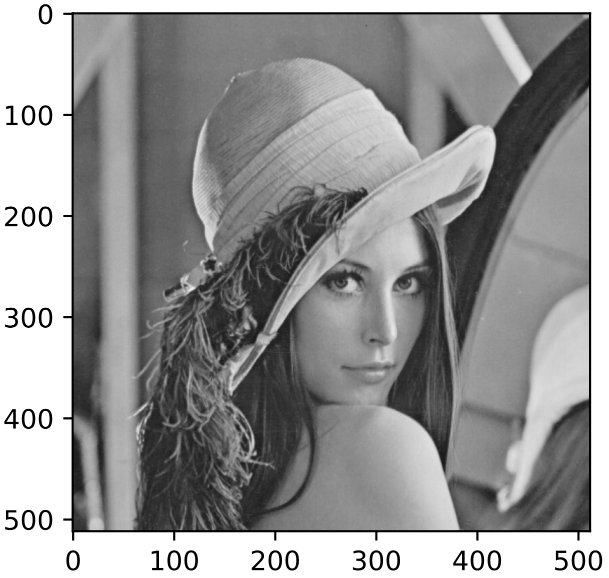
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha & t_1 \\ \sin\alpha & \cos\alpha & t_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

### 2.2.2 Rotation: Sample

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

# Display original
I = cv2.imread('img/lena512.bmp')
plt.imshow(I)
```

```
<matplotlib.image.AxesImage at 0x7f257fdc7e50>
```



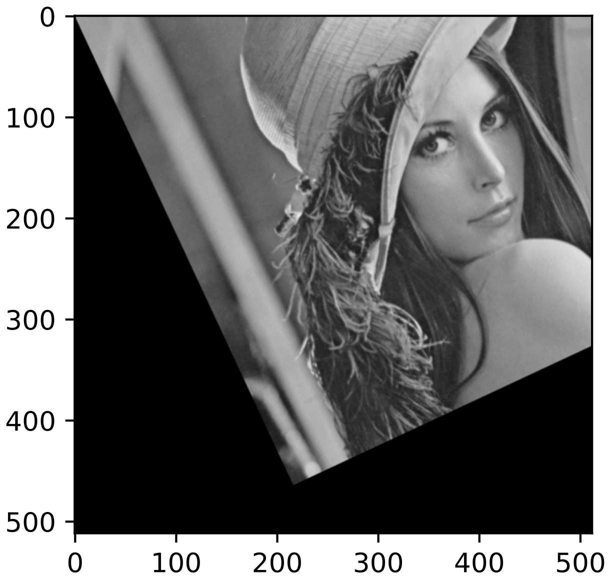
```
# Rotate with OpenCV
angle = 25
#fixed_point = tuple(np.array(I.shape[1::-1]) / 2)
#fixed_point = tuple([50, 50])
fixed_point = tuple([0, 0])

rot_mat = cv2.getRotationMatrix2D(fixed_point, angle, 1.0)
Irot = cv2.warpAffine(I, rot_mat, I.shape[1::-1], flags=cv2.INTER_LINEAR)

print(f"Rotating around point {fixed_point}")
print(f"Rotation matrix is {rot_mat}")
plt.imshow(Irot)
```

```
Rotating around point (0, 0)
Rotation matrix is [[ 0.90630779  0.42261826  0.        ]
 [-0.42261826  0.90630779  0.        ]]
```

```
<matplotlib.image.AxesImage at 0x7f257e762640>
```



## 2.3 Affine transformations (skew)

Image

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

or

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_1 \\ a_{21} & a_{22} & t_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Translation: $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$ ends up at location $\begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$
- 6 unknowns: you need 3 pairs of points to define a skew

## 2.4 Affine transformation: Sample

```
# Define pairs of points
# point = (x, y) ; [511, 0] = top right
points_src = np.float32([[0,0], [511, 0], [0, 511]])
```

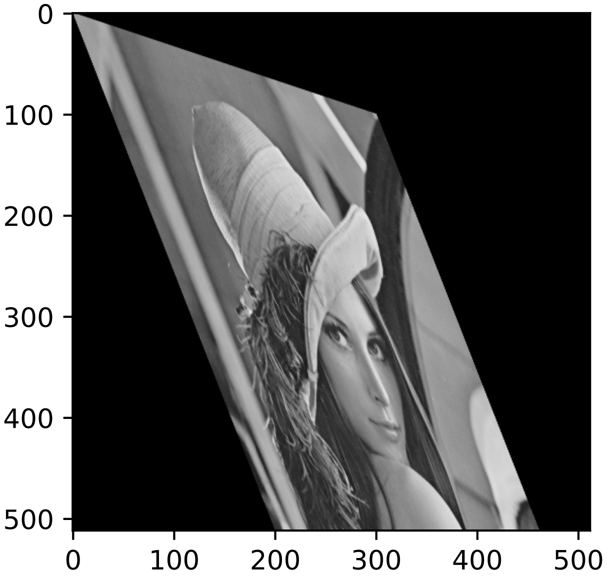
```
points_dst = np.float32([[0,0], [300, 100], [200, 511]])

affine_mat = cv2.getAffineTransform(points_src, points_dst)
Iaff = cv2.warpAffine(I, affine_mat, I.shape[1::-1], flags=cv2.INTER_LINEAR)

print(f"Affine transformation matrix is {affine_mat}")
plt.imshow(Iaff)
```

```
Affine transformation matrix is [[0.58708415 0.39138943 0.        ]
 [0.19569472 1.         0.        ]]
```

```
<matplotlib.image.AxesImage at 0x7f257c6aea90>
```



Example: https://theailearner.com/tag/cv2-getaffinetransform/

## 2.5 Application: Image morphing

1. Define point correspondences
2. Decompose source image in triangles
3. Compute affine transforms and warp every src triangle -> dst triangle
4. Reassemble output image

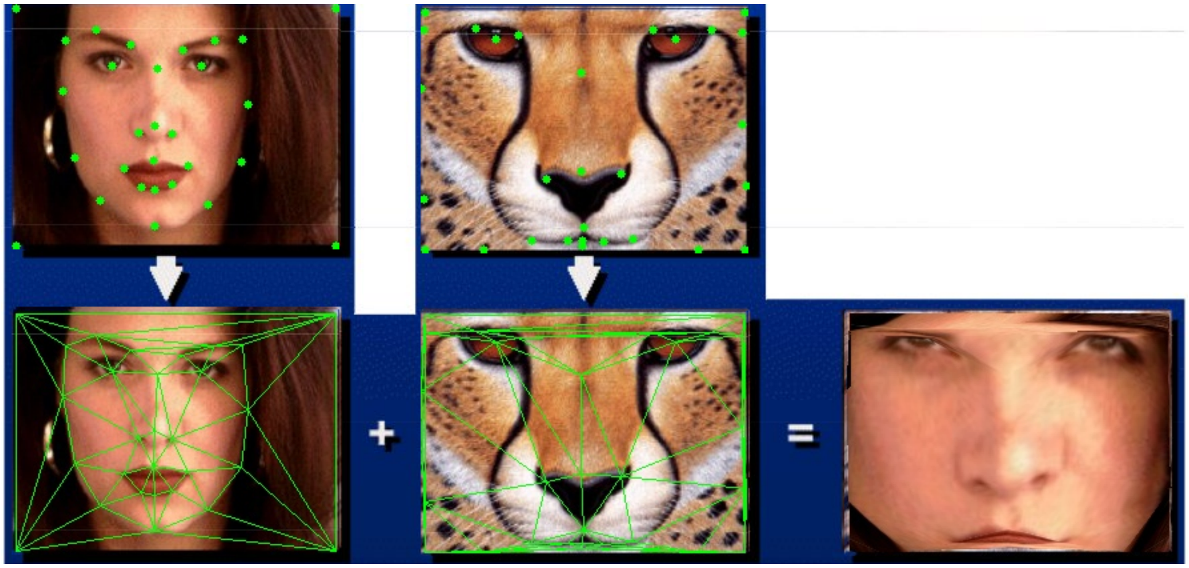


Figure 2.1: Face morphing

Source: https://stackoverflow.com/a/65452859

Also: https://devendrapratapyadav.github.io/FaceMorphing/

## 2.6 Perspective transformations

Image

Step 1:

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_1 \\ a_{21} & a_{22} & t_2 \\ a_{31} & a_{32} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Step 2: divide by $w$:

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} \rightarrow \begin{bmatrix} x'/w \\ y'/w \\ 1 \end{bmatrix}$$

- 8 unknowns: you need 4 pairs of points to define perspective

## 2.7 Perspective transformation: Sample

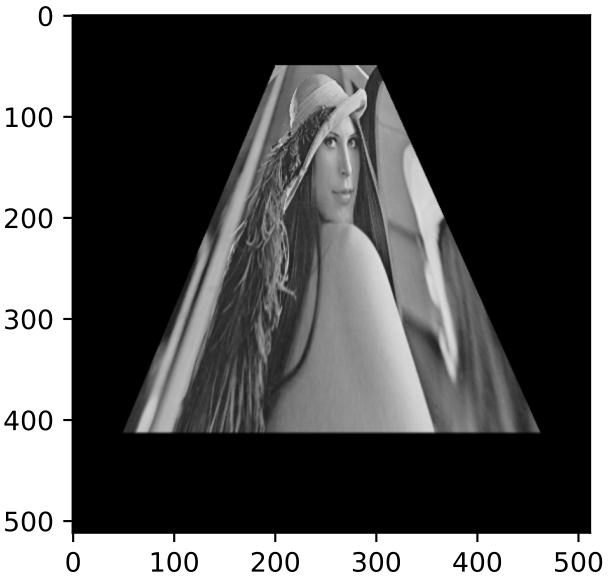
```python
# Define pairs of points
# point = (x, y) ; [511, 0] = top right
points_src = np.float32([[0,0],     [511, 0],  [0, 511],    [511, 511]])
points_dst = np.float32([[200,50], [300, 50], [50, 411], [461, 411]])

persp_mat = cv2.getPerspectiveTransform(points_src, points_dst, cv2.DECOMP_LU)
Ipersp = cv2.warpPerspective(I, persp_mat, I.shape[1::-1], flags=cv2.INTER_LINEAR)

print(f"Perspective transformation matrix is {persp_mat}")
plt.imshow(Ipersp)
```

```
Perspective transformation matrix is [[ 1.95694716e-01 -3.67582289e-01  2.00000000e+02]
 [ 0.00000000e+00  9.78473581e-02  5.00000000e+01]
 [-0.00000000e+00 -1.48080430e-03  1.00000000e+00]]
```

```
<matplotlib.image.AxesImage at 0x7f257c63e520>
```



See here: https://theailearner.com/tag/cv2-getperspectivetransform/

## 2.8 How do these functions work?

1. Find the inverse transformation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \end{bmatrix}$$

2. For each destination location $\begin{bmatrix} x' \\ y' \end{bmatrix}$:

    - Find the source location $\begin{bmatrix} x \\ y \end{bmatrix}$
    - $x$ and $y$ may not be integers, so interpolate

## 2.9 Bilinear interploation

- Interpolate the value based on the 4 neighbors
- Source: Wikipedia
- Image: By Cmglee - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=2140

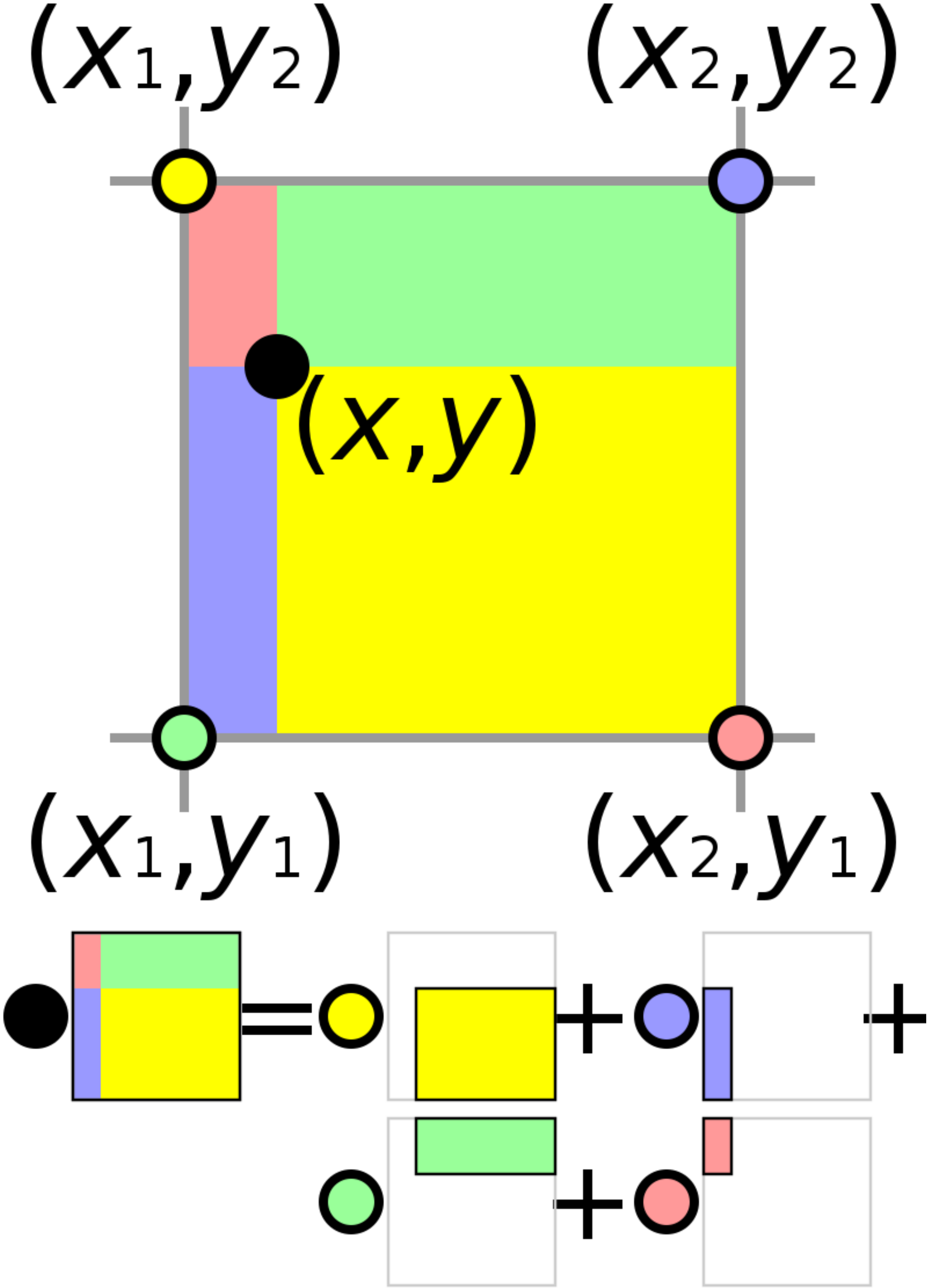


Figure 2.2: Bilinear interpolation