Announcement

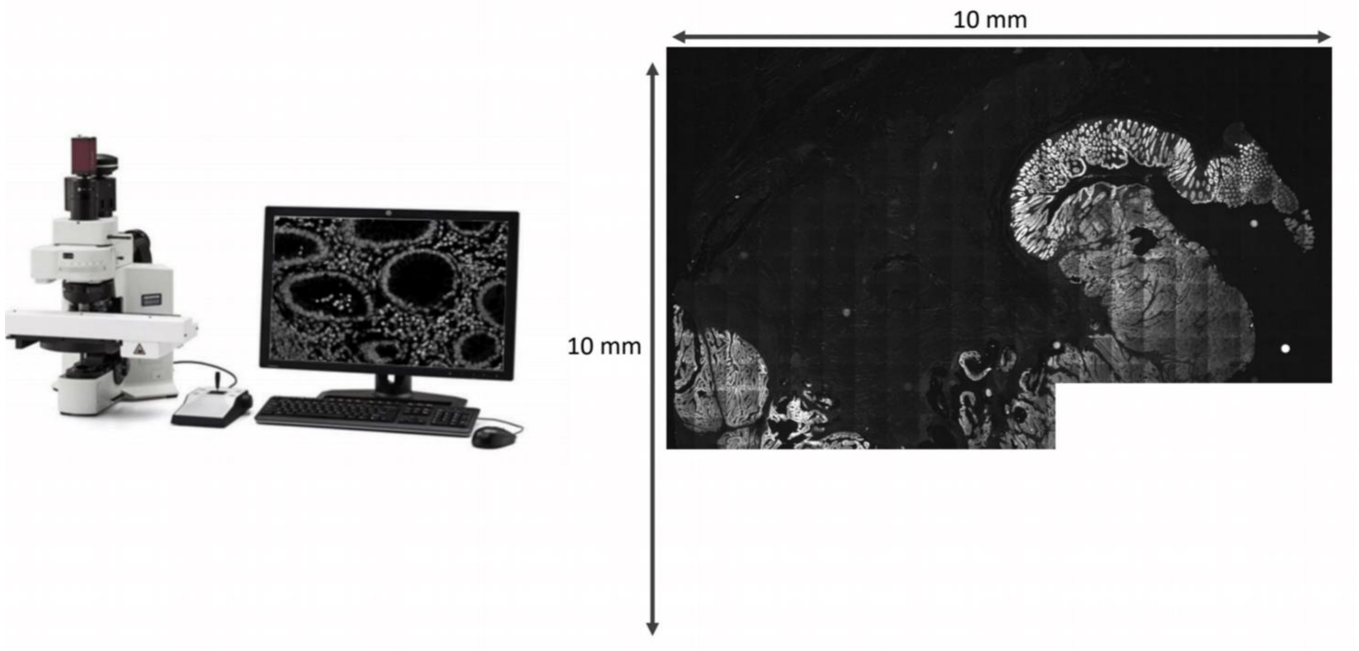
2/11/2025 Tuesday

- Final project
  - Details out
  - Keep milestones in mind

- Image transformation

Motivation

Panoramic scene



Field-of-view is smaller than sample

Example

- Two images with overlapping regions

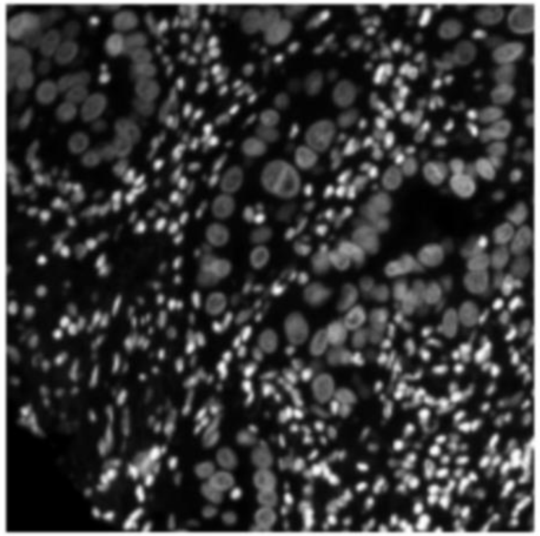


Image 1

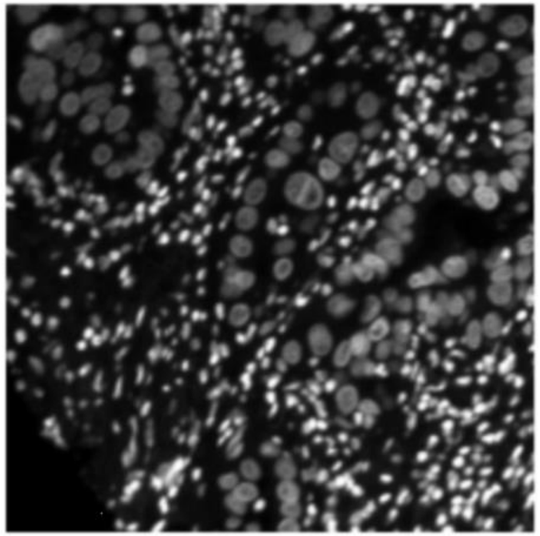
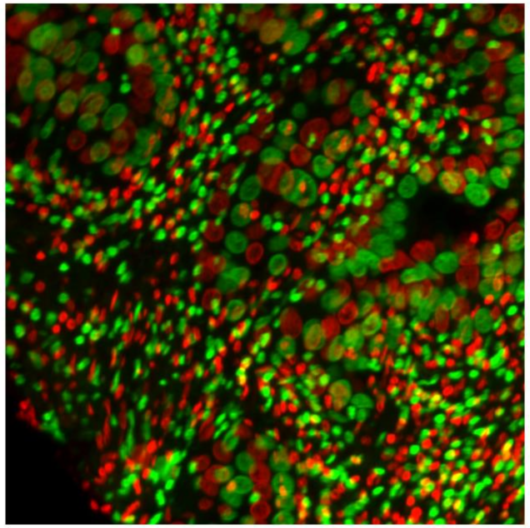
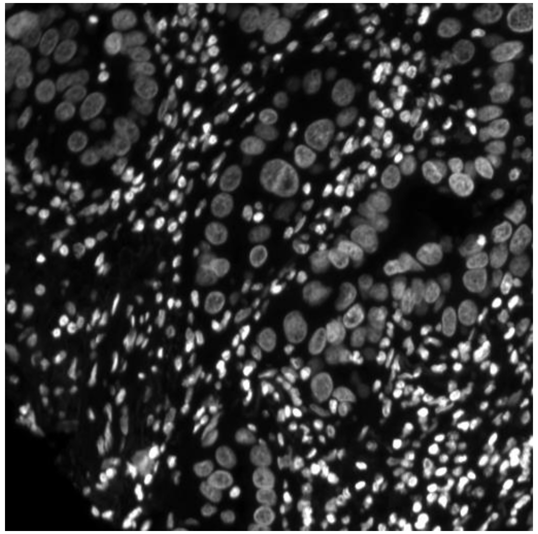


Image 2



Vis 1: RGB



Vis 2: Gif motion

```
plt.subplot(121)
plt.imshow(img1, cmap='gray');plt.axis('off')
plt.subplot(122)
plt.imshow(img2, cmap='gray');plt.axis('off')
```

Better visualizations

Naive plot: looks the same

Example

- Goal: Register two images

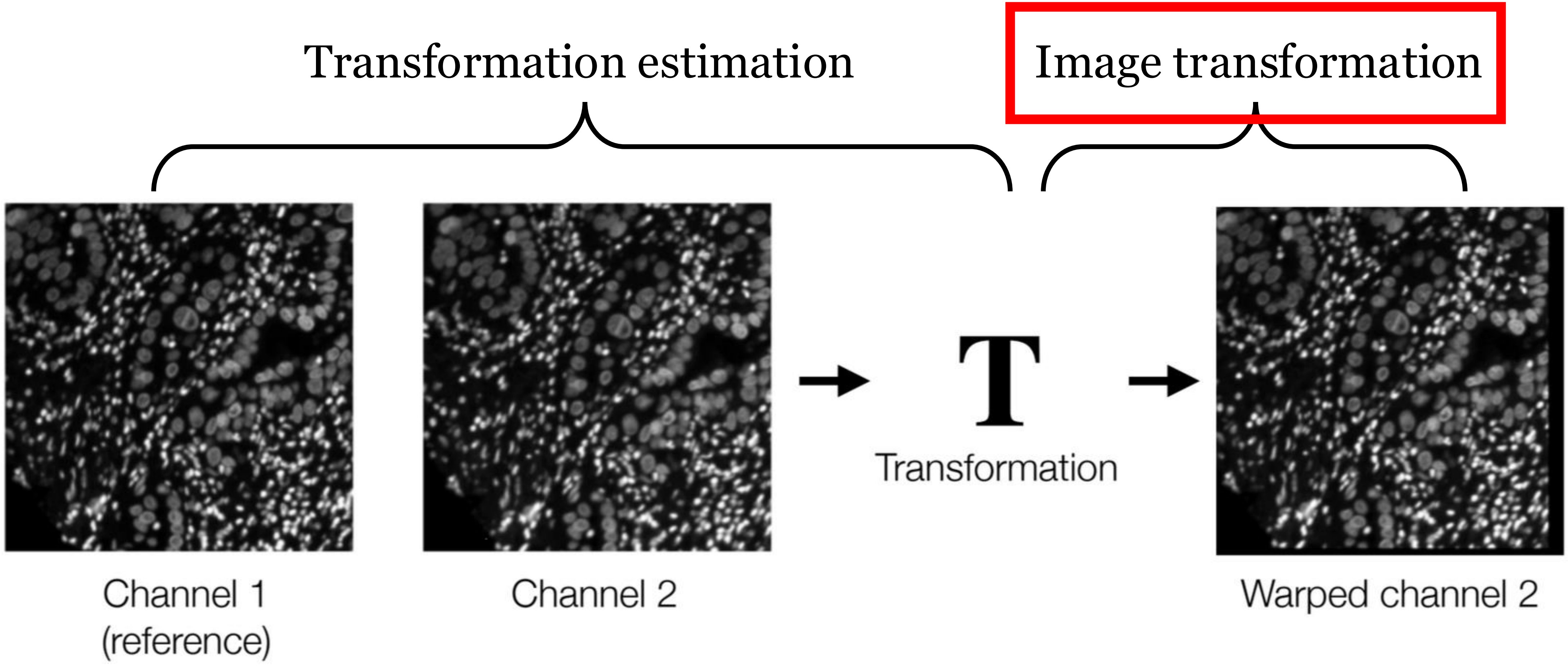
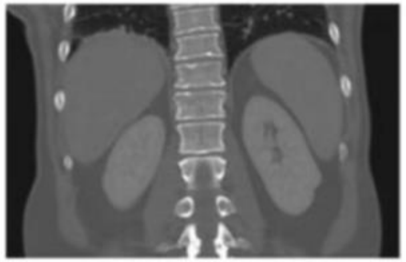


Image transformation

• Image representation: matrix → 2D points

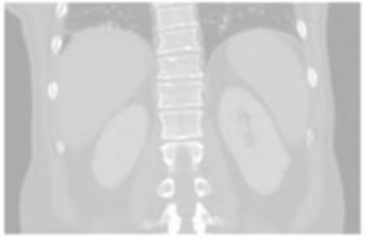


Image

Filtering
(same position, different value)

83	100	240
22	239	159
143	242	5

Matrix



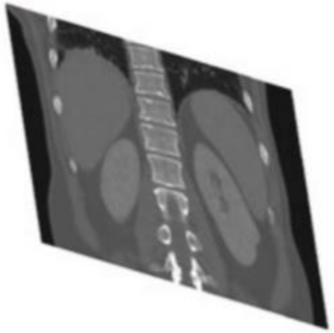
Change appearance

Transformation
(different position, same value)

$(x_i, y_i, I_i = I[x_i, y_i])$

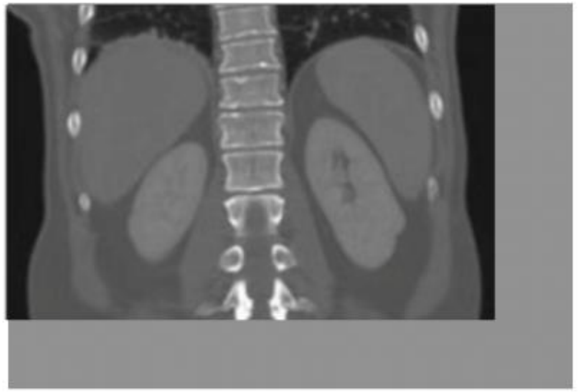
0, 0, 83
1, 0,100
2, 0, 240
0, 1, 222
1, 1, 239
2, 1, 159

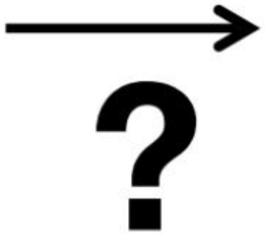
Point cloud

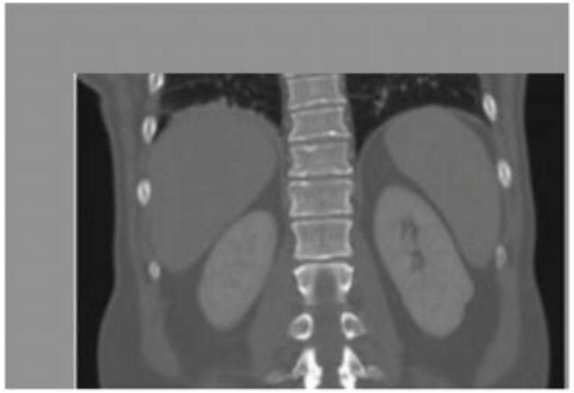


Change shape

I. How to represent the deformation?



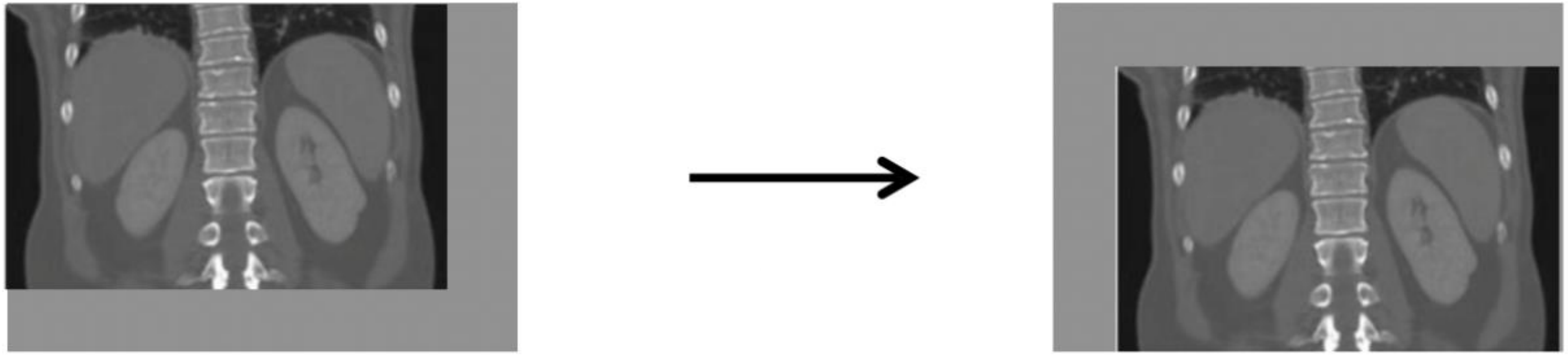




$(x_i, y_i, I_i = I[x_i, y_i])$	$(x_i, y_i, I_i = I[x_i, y_i])$
0, 0, 83	2, 3, 83
1, 0,100	3, 3,100
2, 0, 240	4, 3, 240
0, 1, 222	2, 4, 222
1, 1, 239	3, 4, 239
2, 1, 159	4, 4, 159

How to represent the deformation?

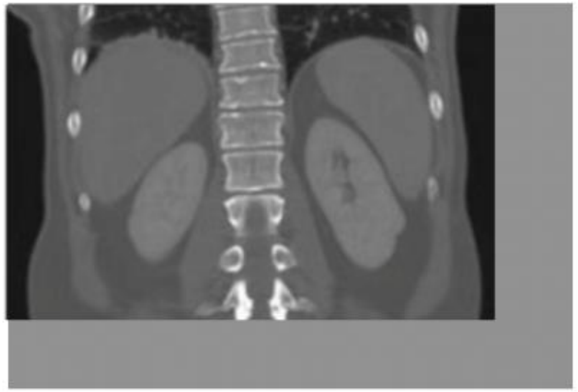
- Method 1: dense deformation field

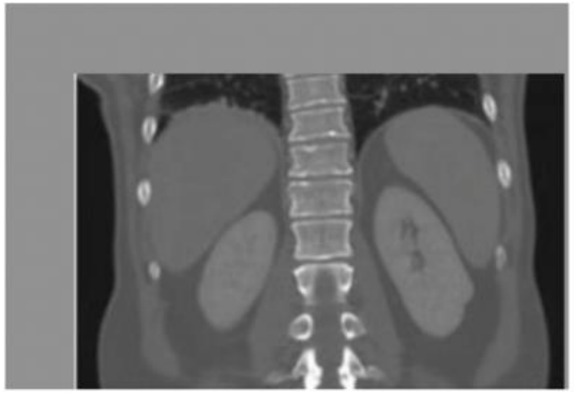


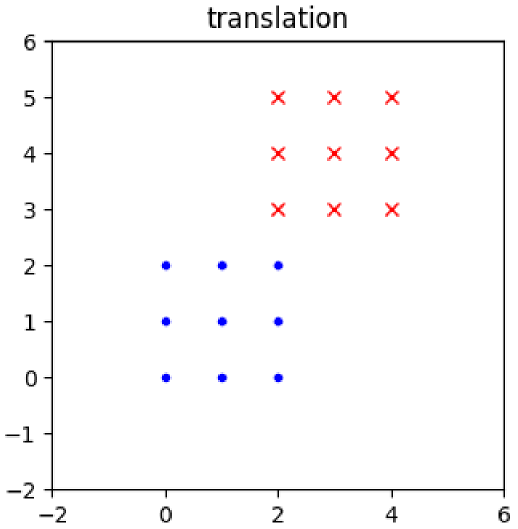
$(x_i, y_i, I_i = I[x_i, y_i])$	$(x_i, y_i, \Delta x_i, \Delta y_i)$	$(x_i, y_i, I_i = I[x_i, y_i])$
0, 0, 83	0, 0, 2, 3	2, 3, 83
1, 0,100	1, 0, 2, 3	3, 3,100
2, 0, 240	2, 0, 2, 3	4, 3, 240
0, 1, 222	0, 1, 2, 3	2, 4, 222
1, 1, 239	1, 1, 2, 3	3, 4, 239
2, 1, 159	2, 1, 2, 3	4, 4, 159

How to represent the deformation?

- Method 2: linear transformation = matrix





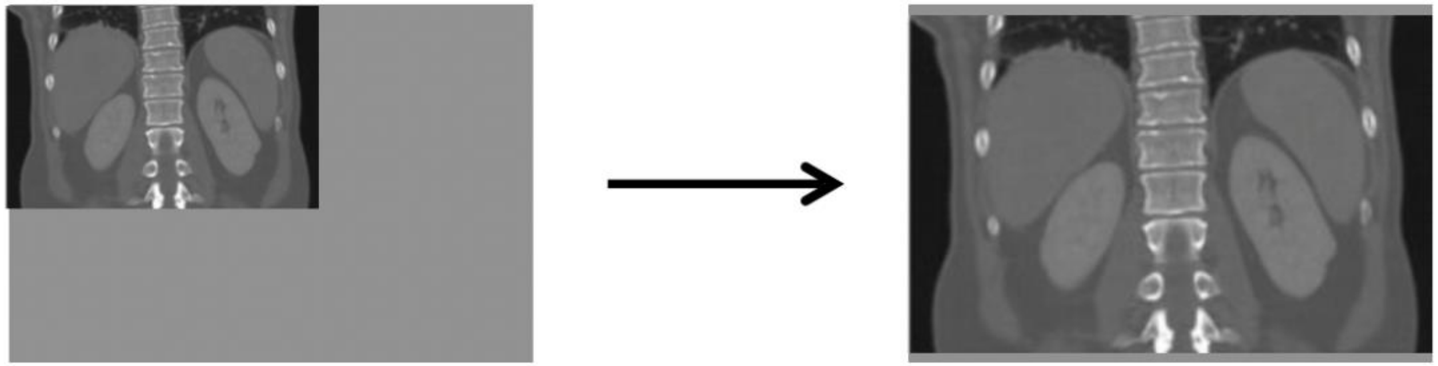


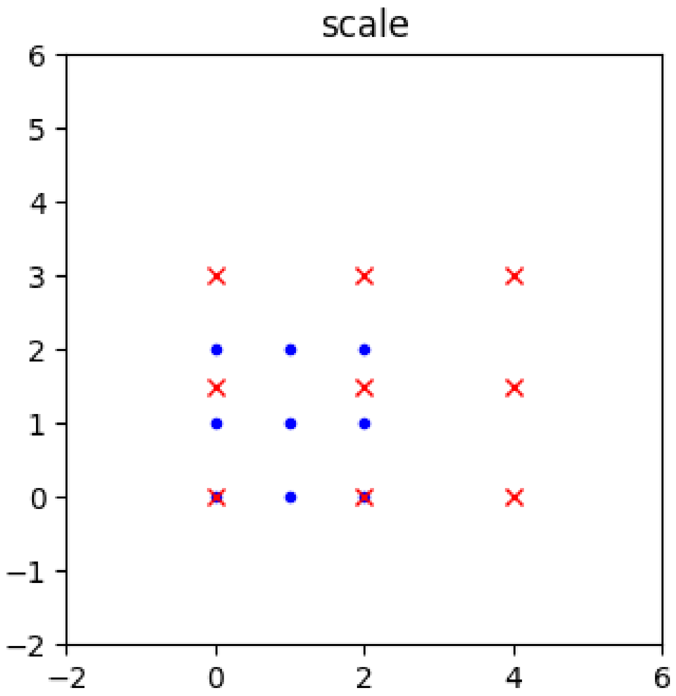
Move to bottom-right by (2,3) pixel

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \rightarrow \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 + a \\ y_1 + b \end{bmatrix} = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

e.g., (0, 0, 83) → (a, b, 83)

Example: image scale

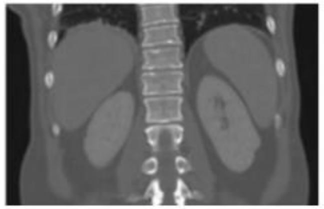


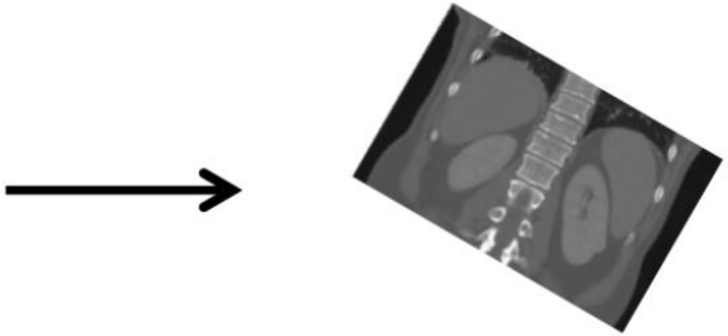


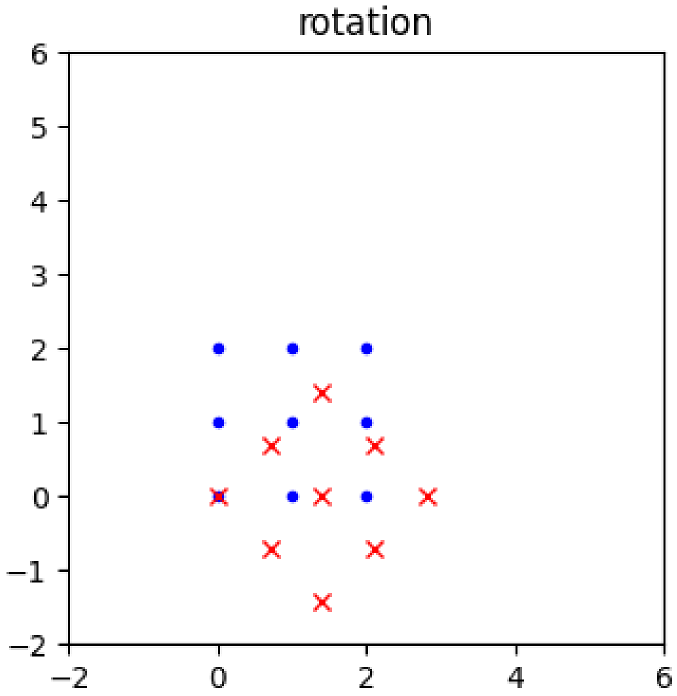
$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \rightarrow \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} ax_1 \\ by_1 \end{bmatrix} = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

e.g., (1, 2, 83) → (a, 2b, 83)

Example: image rotation



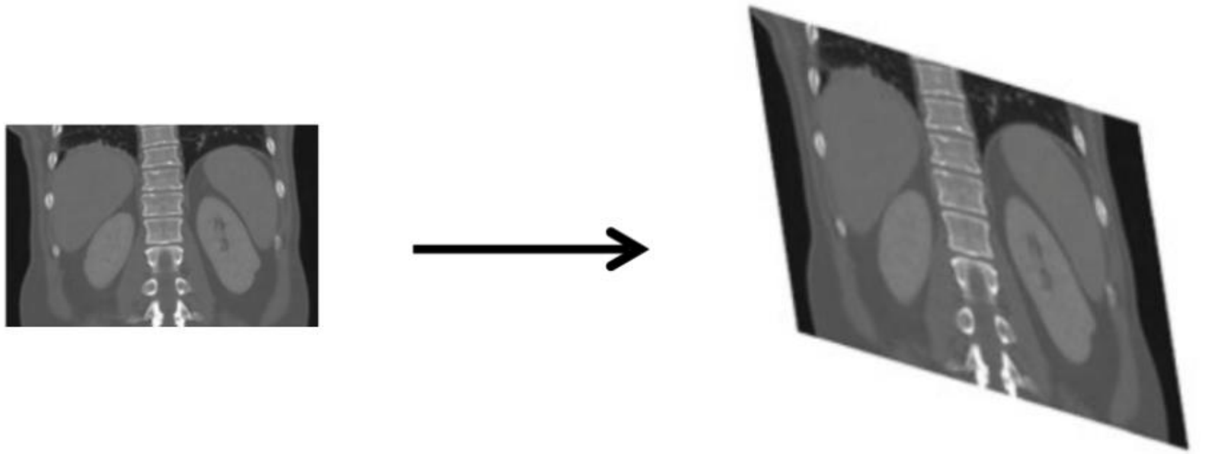


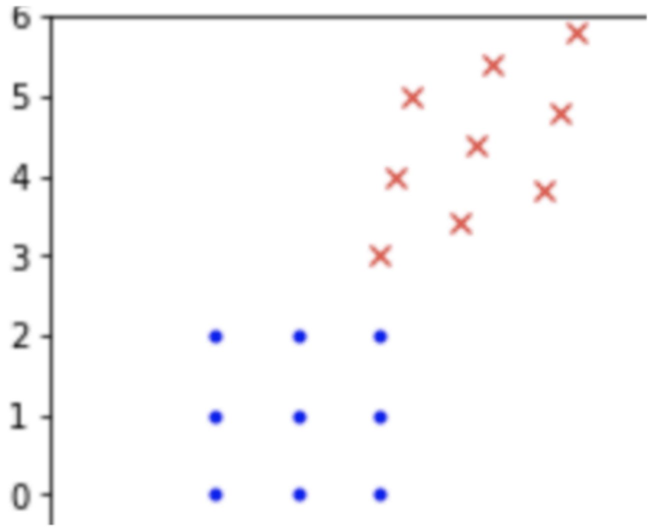


$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \rightarrow \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

e.g., (1, 0, 83) → ($\cos(\alpha)$, $-\sin(\alpha)$, 83)

Example: affine (combine all together)



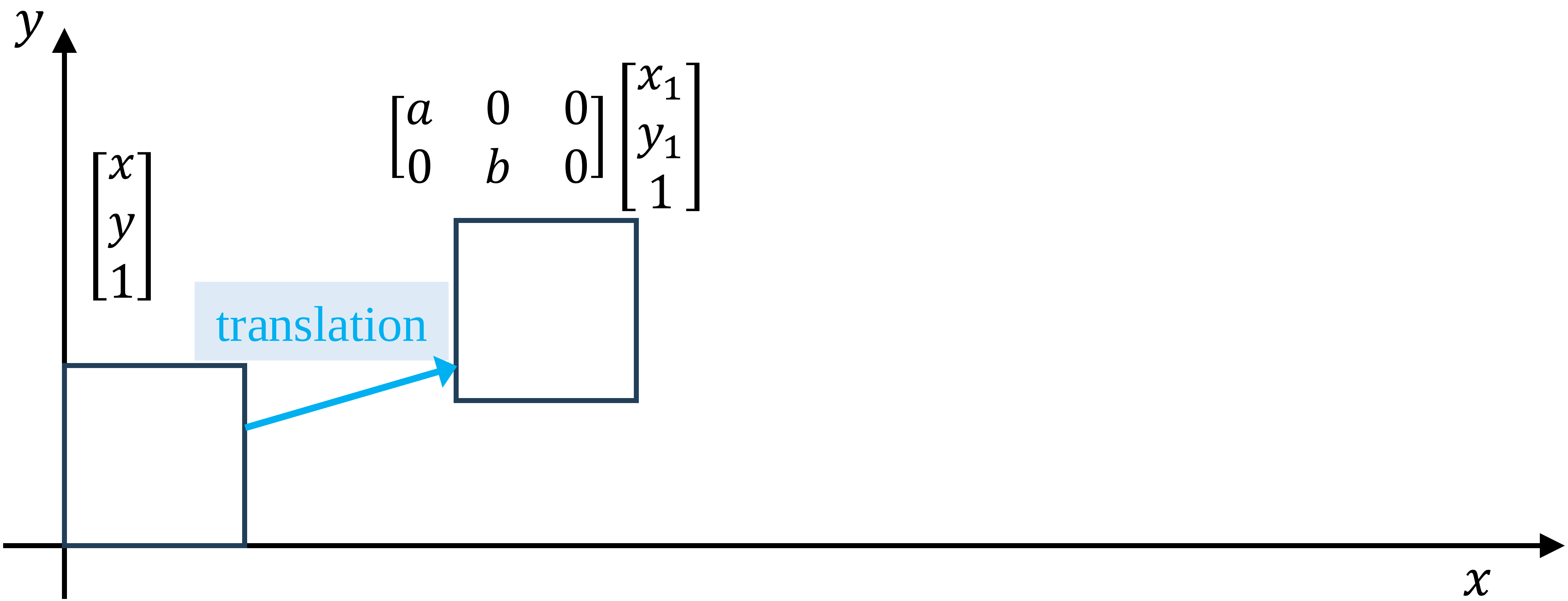


$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \rightarrow \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} ax_1 + by_1 + c \\ dx_1 + ey_1 + f \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

e.g., (1, 0, 83) → (a+c, d+f, 83)

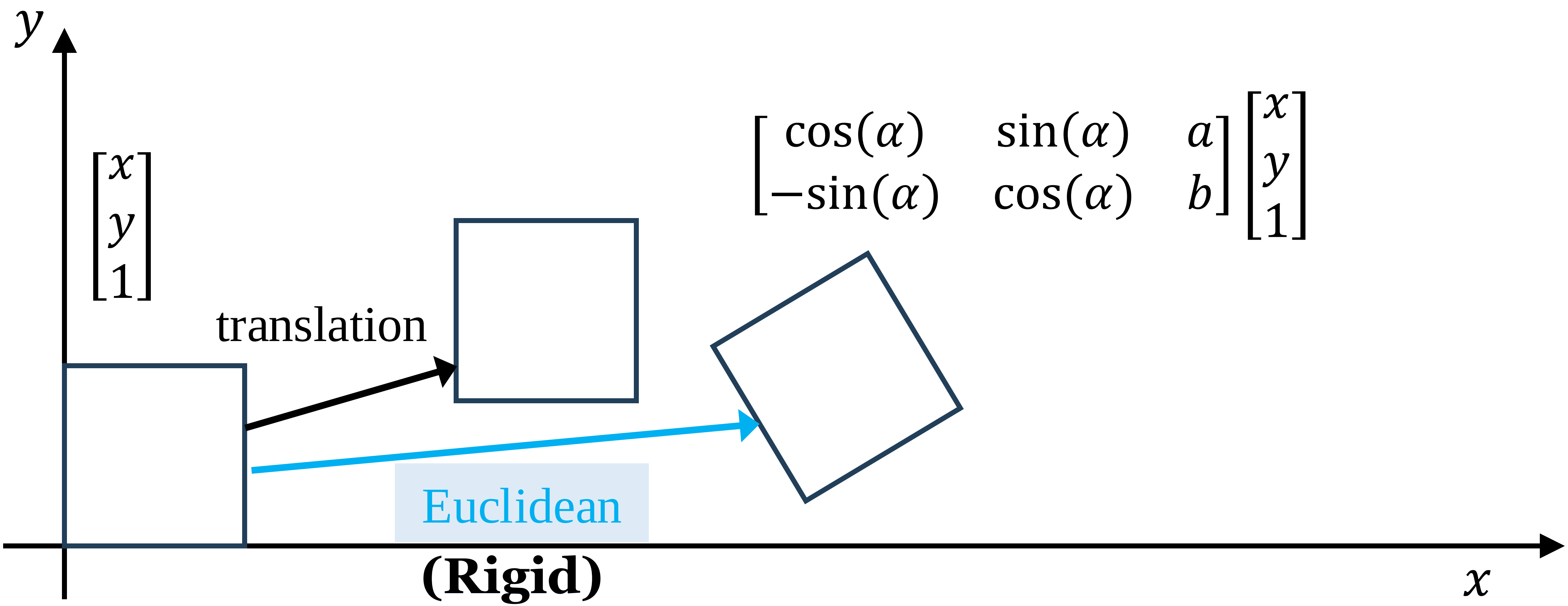
Affine transformation

- Starting from translation



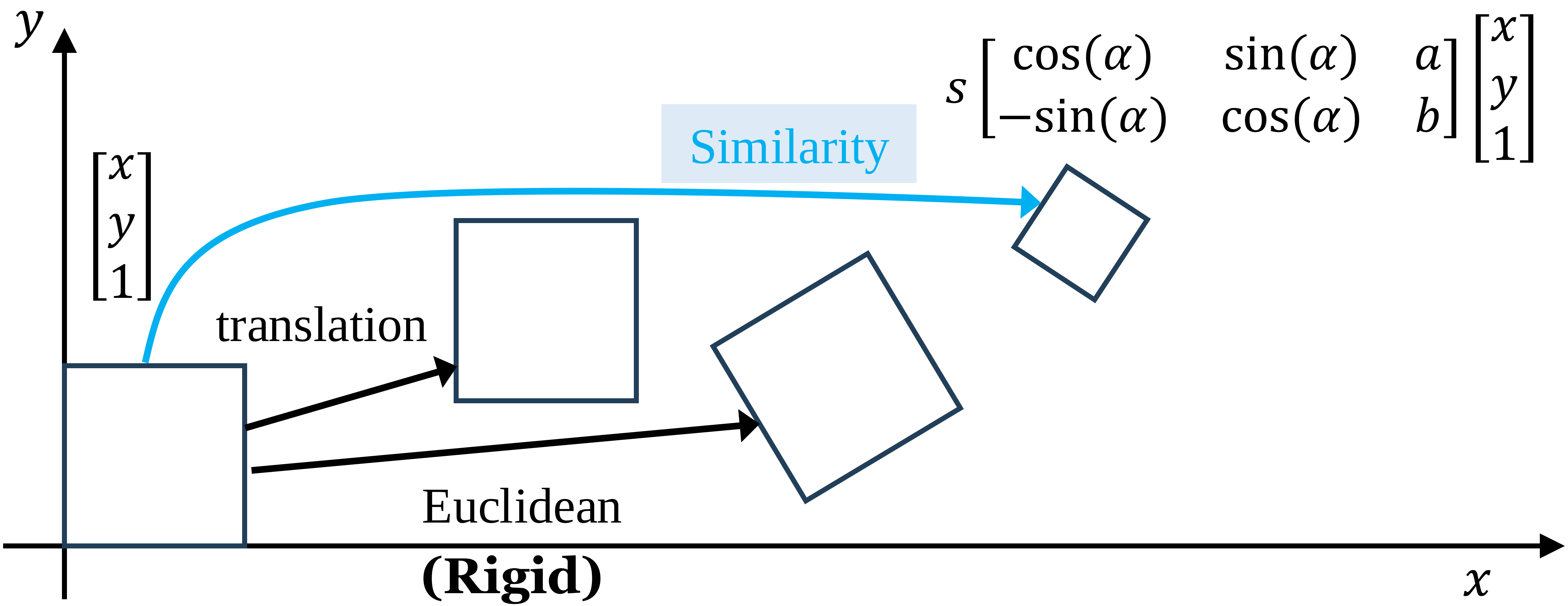
Affine transformation

- Rigid = (translation, rotation)



Affine transformation

- Similarity = (translation, rotation, **uniform** scaling)



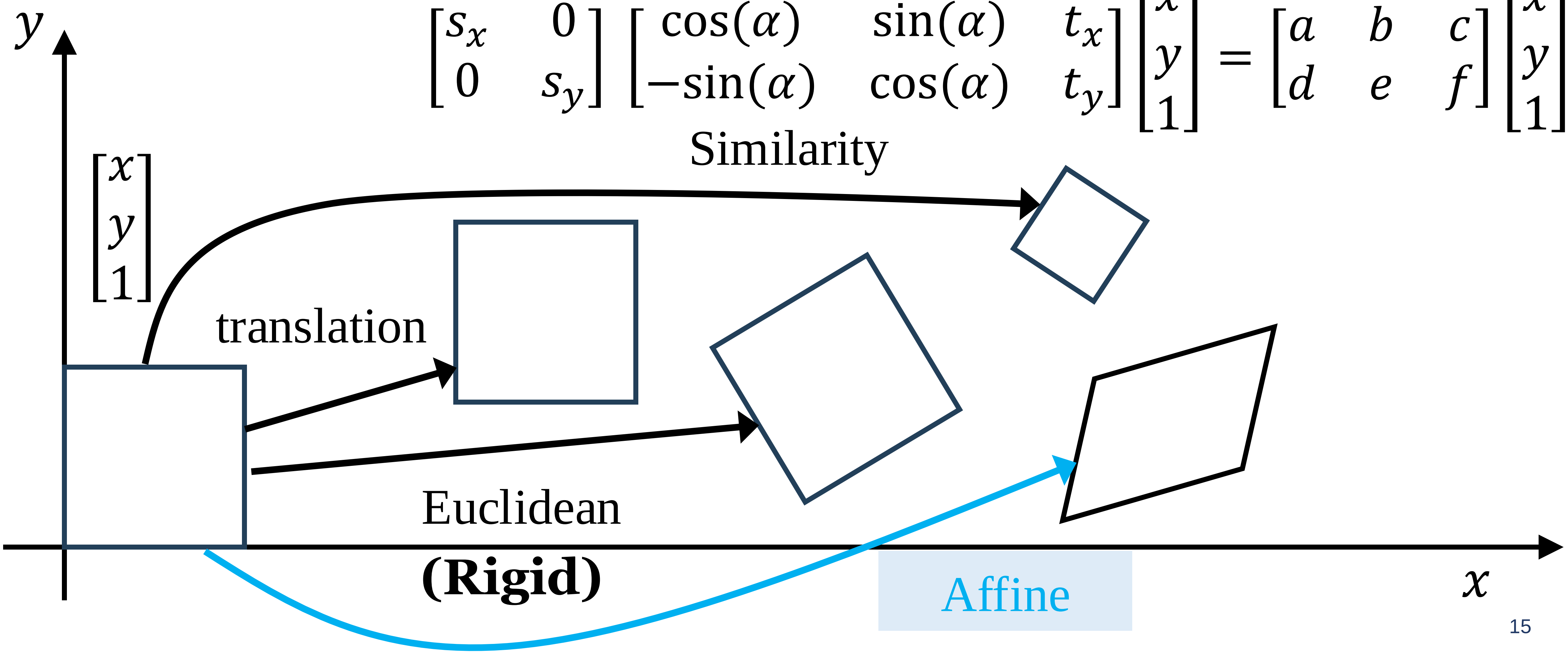
$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$s\begin{bmatrix} \cos(\alpha) & \sin(\alpha) & a \\ -\sin(\alpha) & \cos(\alpha) & b \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

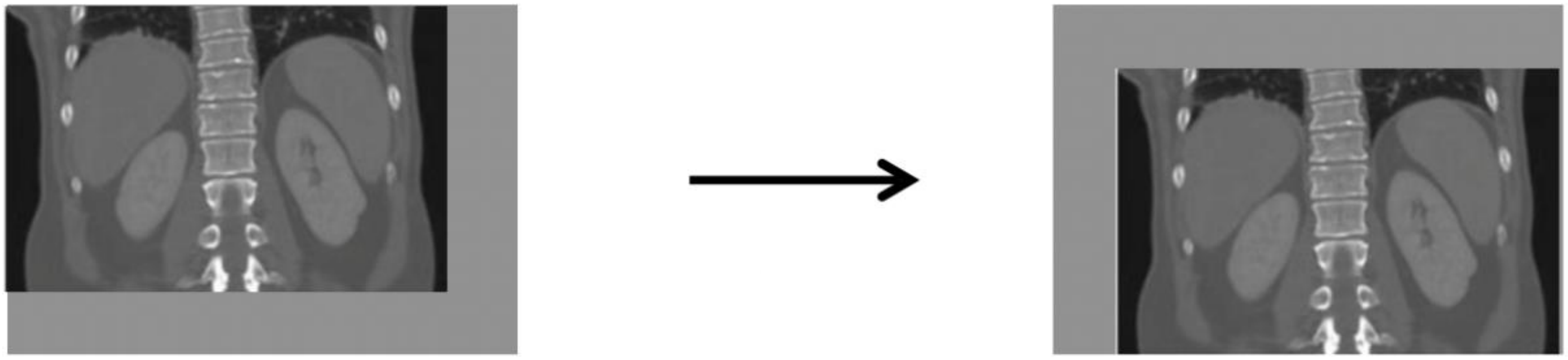
Affine transformation

- Affine = (translation, rotation, scaling)

$$\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & t_x \\ -\sin(\alpha) & \cos(\alpha) & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



II. How to deal with non-integer output points?



Move to bottom-right by (0.8, 0.2) pixel

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \rightarrow \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 + 0.8 \\ y_1 + 0.2 \end{bmatrix}$$

e.g., (0, 0, 83) → (0.8, 0.2, 83)

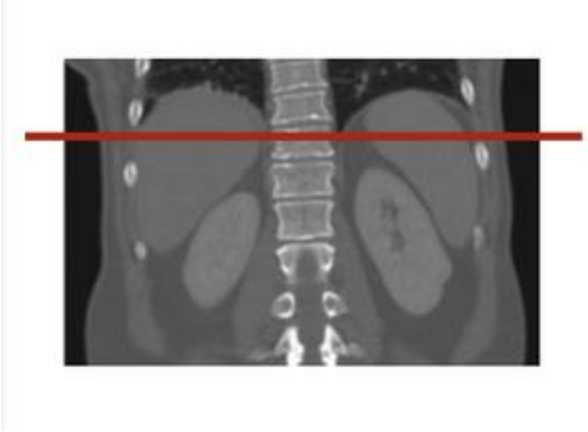
83	100	240	128
22	239	159	128
143	242	5	128
128	128	128	128

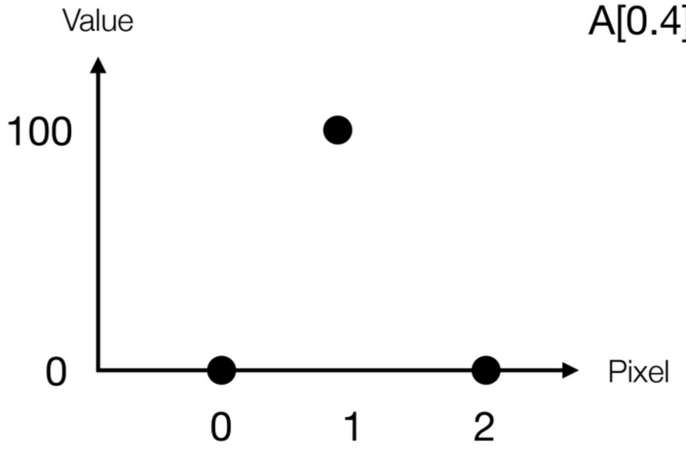
→ ???

Toy example: 1D case

A[0] = 0, A[1] = 100

A[0.4] = ??

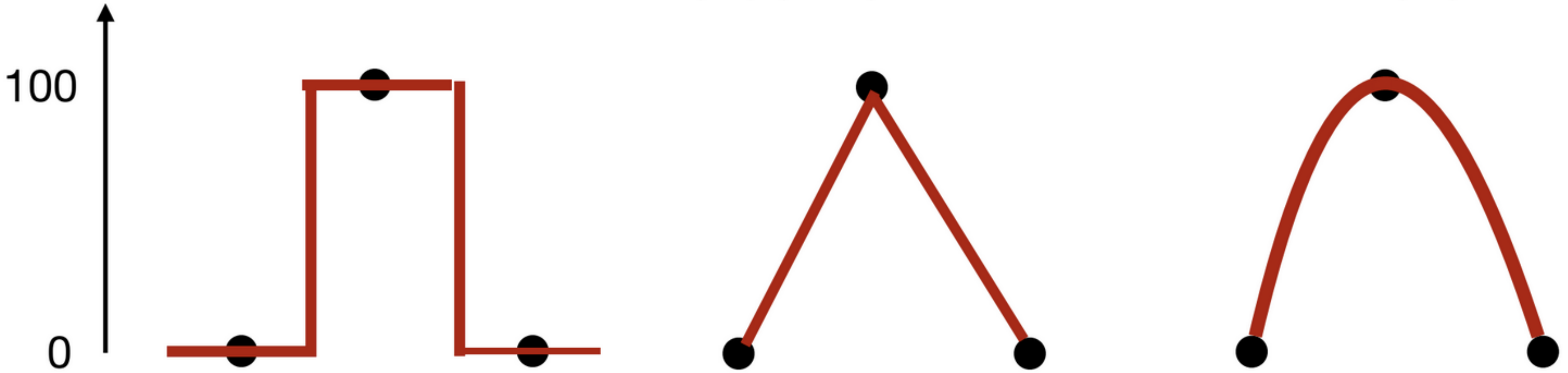




- Solution: interpolation (discrete -> continuous function)

(0,0), (1,100) => 100x

-100x(x-2)

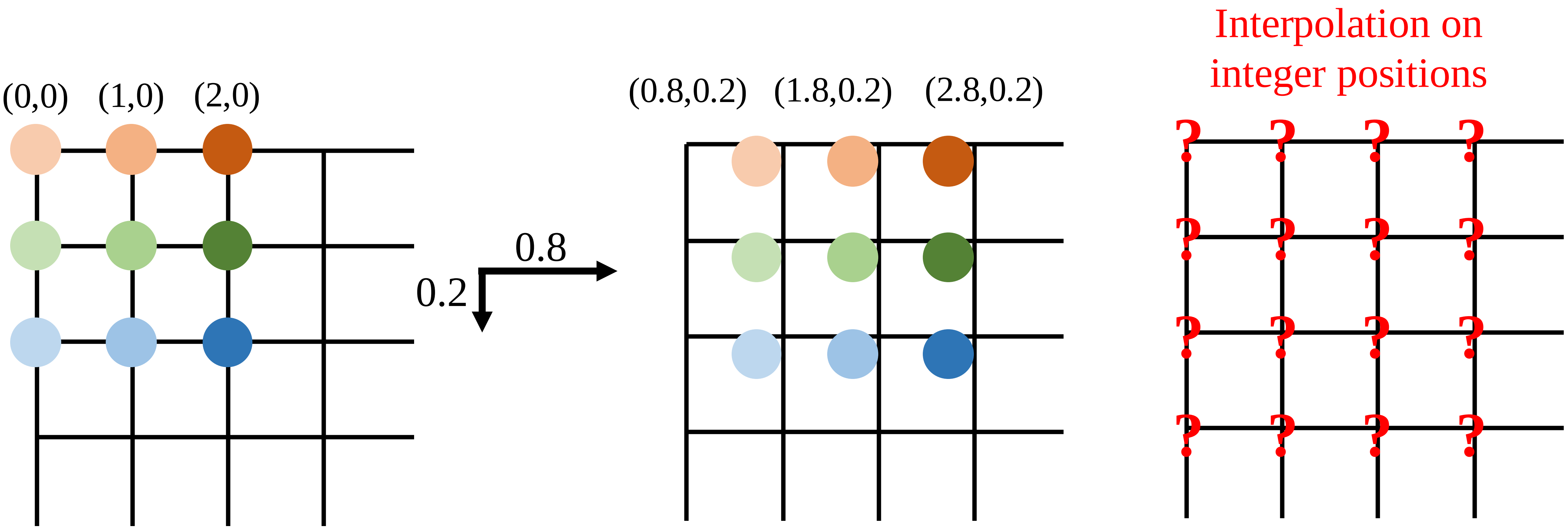


Step function: A[0.4] = 0
(Nearest neighbor)

Linear function: A[0.4] = 40
(Linear)

Quadratic function: A[0.4] = 64
(Quadratic)

Forward mapping



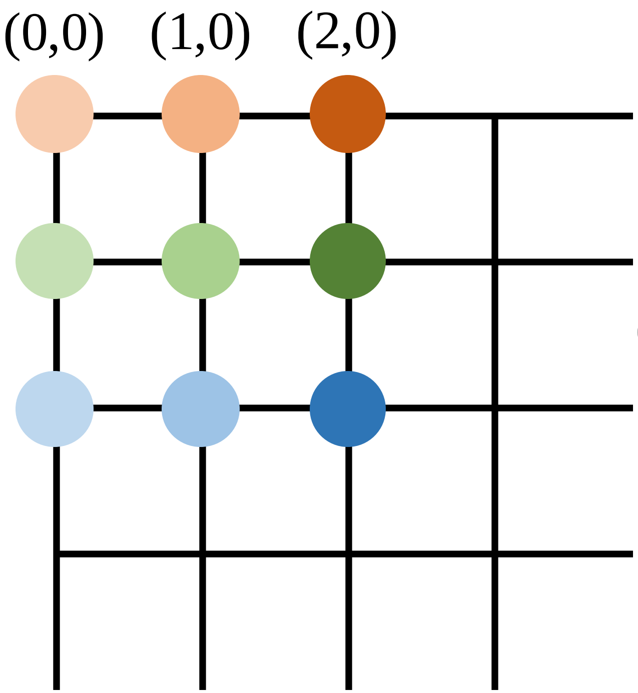
Input image (x_i, y_i)

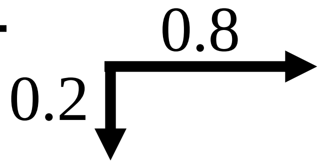
Translated image

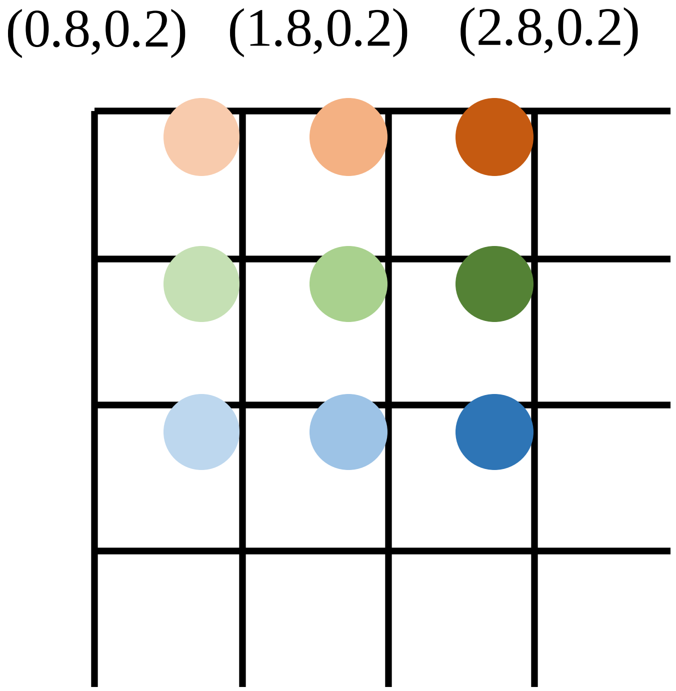
Matrix rendering

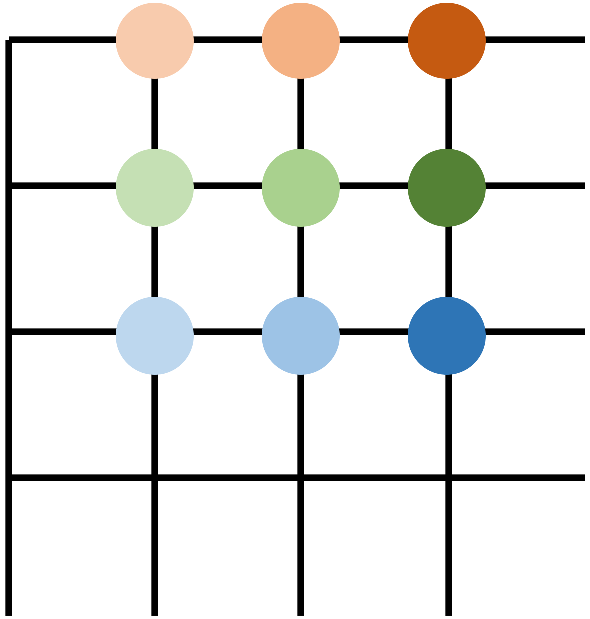
Method 1: nearest neighbor

Problem: same as translating (1, 0)







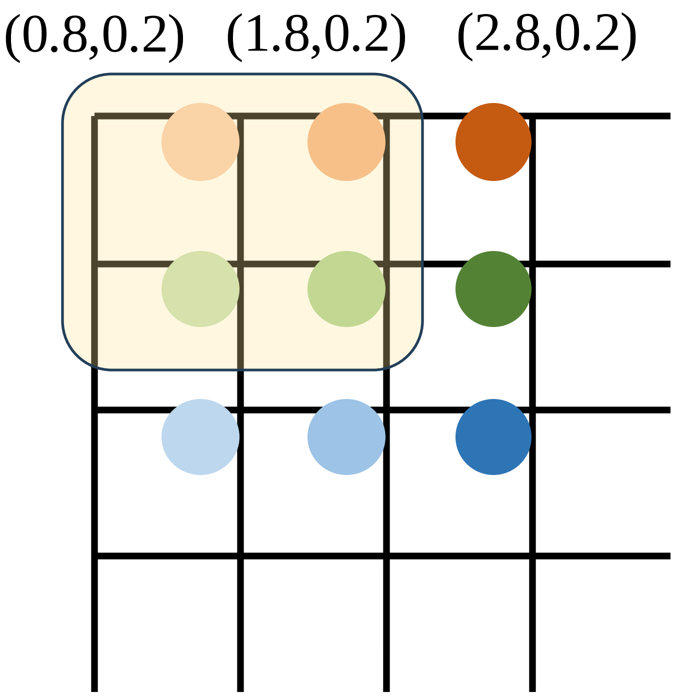


Input image
(x_i, y_i)

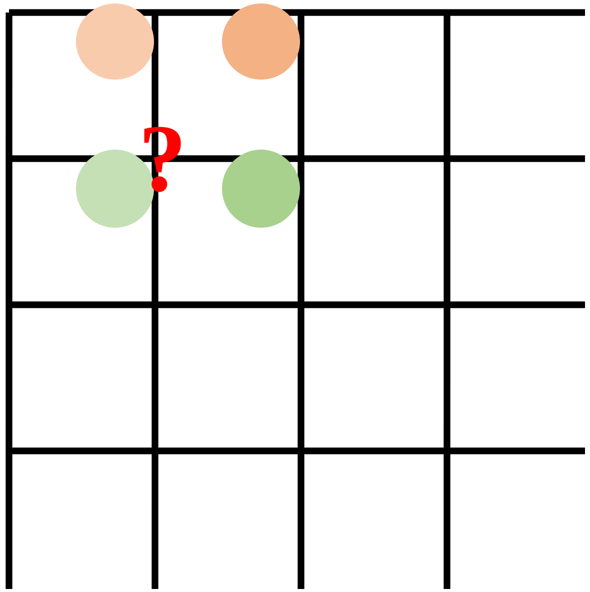
Translated image

Matrix rendering
(Ignore the border for now)

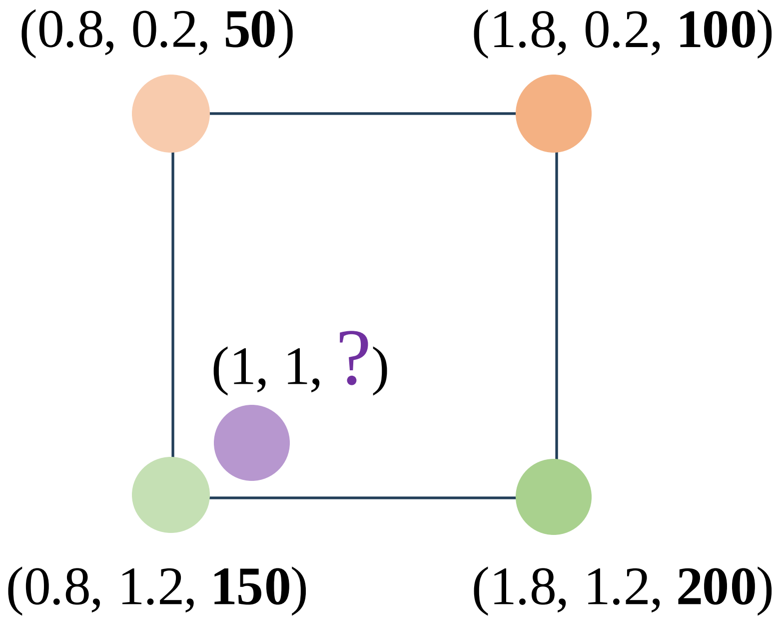
Method 2: bilinear



Translated image

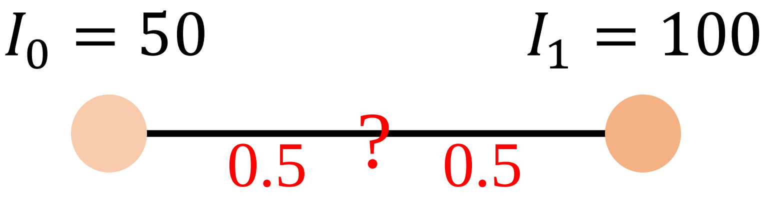


Zoom to one pixel



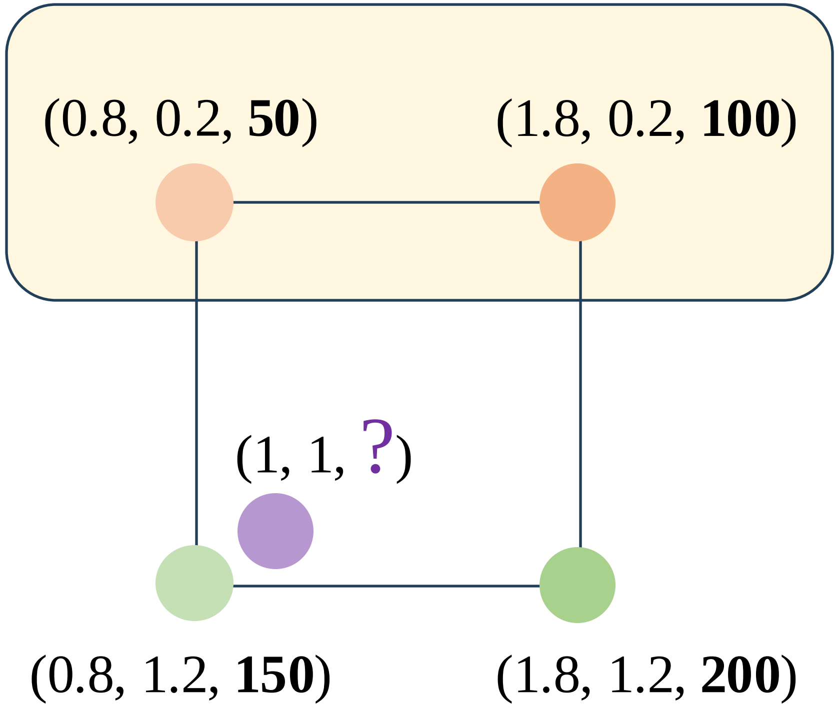
Goal: interpolate pixel value at (1, 1)

Method 2: bilinear



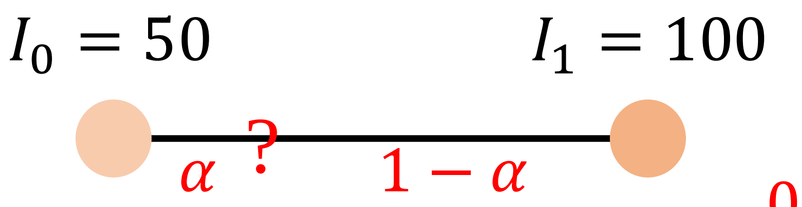
What's the value in the middle?

(1.3, 0.2, 75): take the average



Goal: interpolate pixel value at (1, 1)

Method 2: bilinear

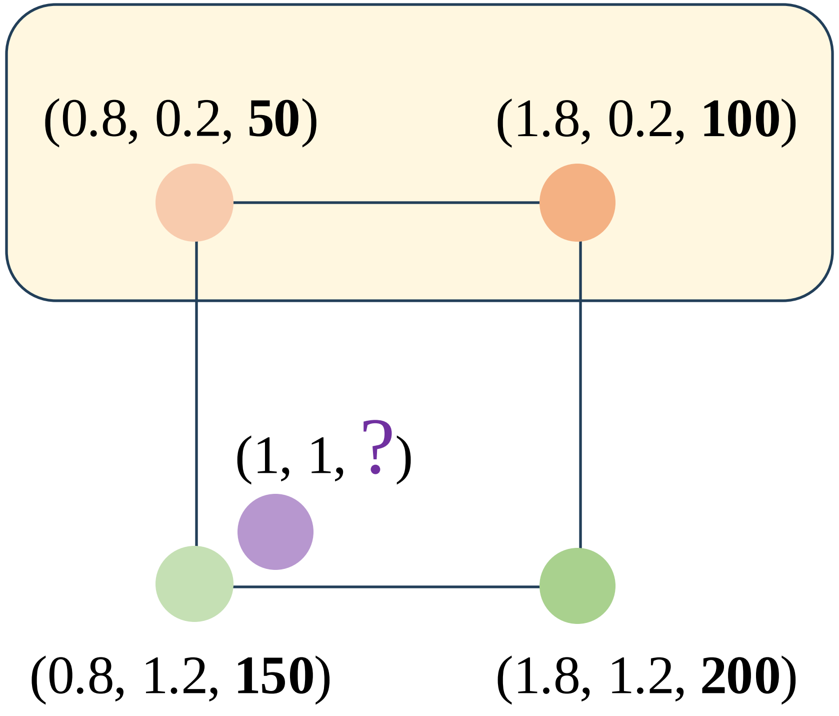


$0 \leq \alpha \leq 1$

What's the value in
any position?

$$I = I_1 * \alpha + I_0 * (1 - \alpha)$$

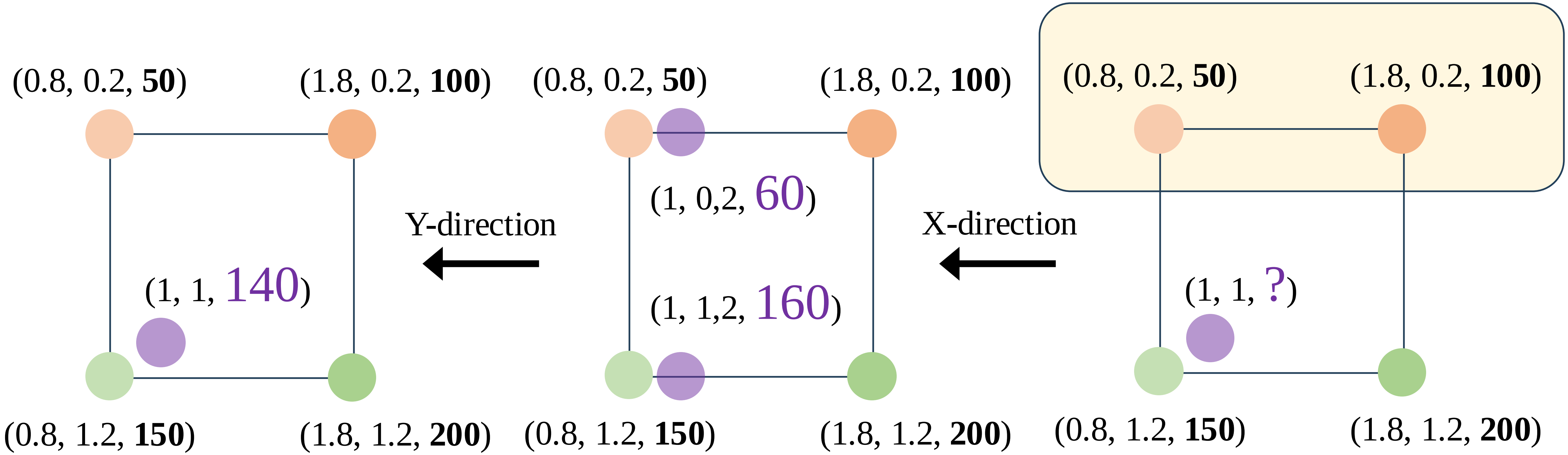
When $\alpha = 0$, $I = I_0$
When $\alpha = 1$, $I = I_1$



Goal: interpolate
pixel value at (1, 1)

Method 2: bilinear

$$I = I_1 * \alpha + I_0 * (1 - \alpha)$$



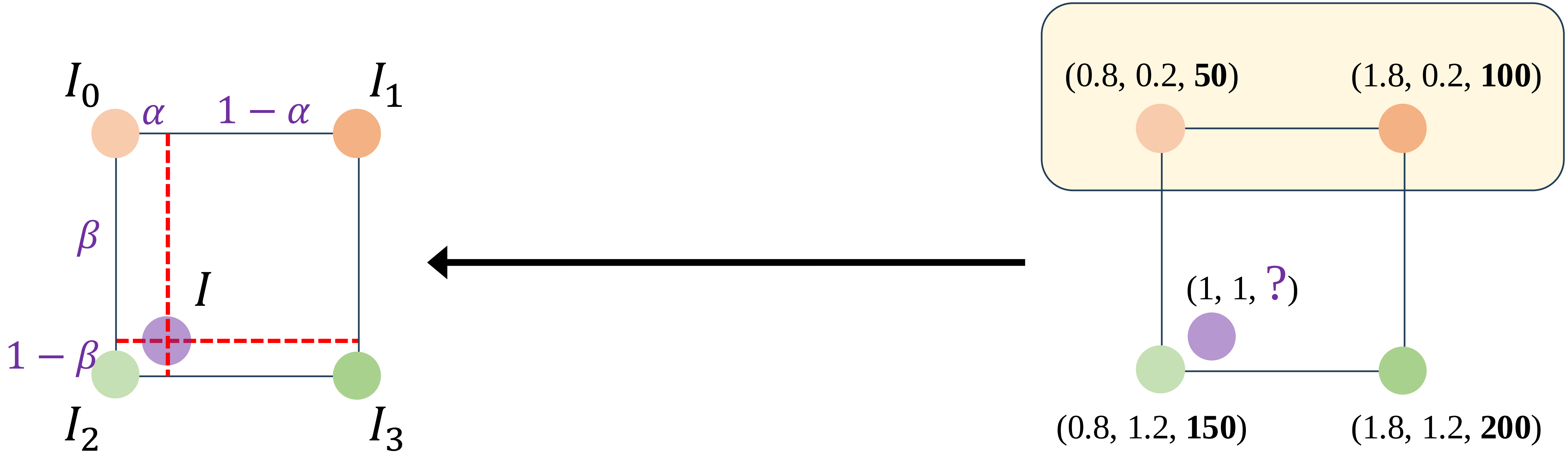
$60 * 0.2 + 160 * 0.8 = 140$

$100 * 0.2 + 50 * 0.8 = 60$

$200 * 0.2 + 150 * 0.8 = 160$

Goal: interpolate pixel value at (1, 1)

Method 2: bilinear



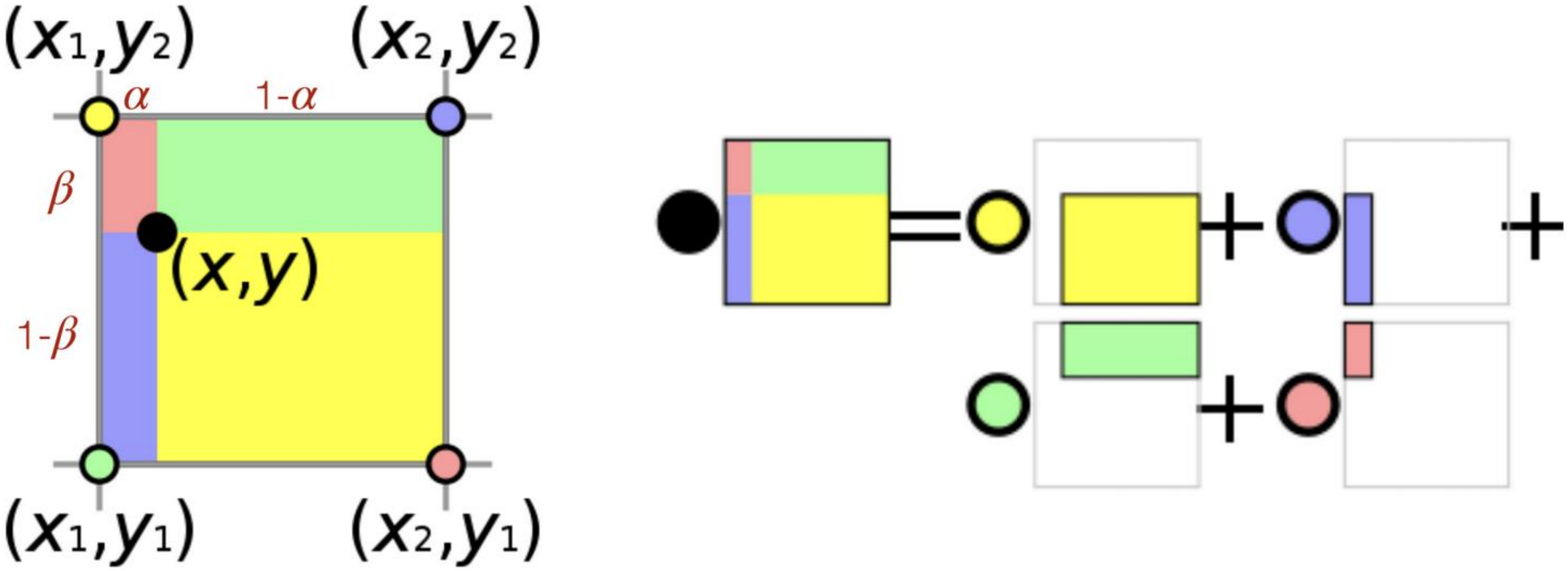
$$I = \big(I_1 * \alpha + I_0 * (1-\alpha)\big) * (1-\beta) + \big(I_3 * \alpha + I_2 * (1-\alpha)\big) * \beta$$
$$= (1-\alpha)(1-\beta)I_0 + \alpha(1-\beta)I_1 + (1-\alpha)\beta I_2 + \alpha\beta I_3$$

Goal: interpolate pixel value at (1, 1)

Linear combination of four corner values

Method 2: bilinear

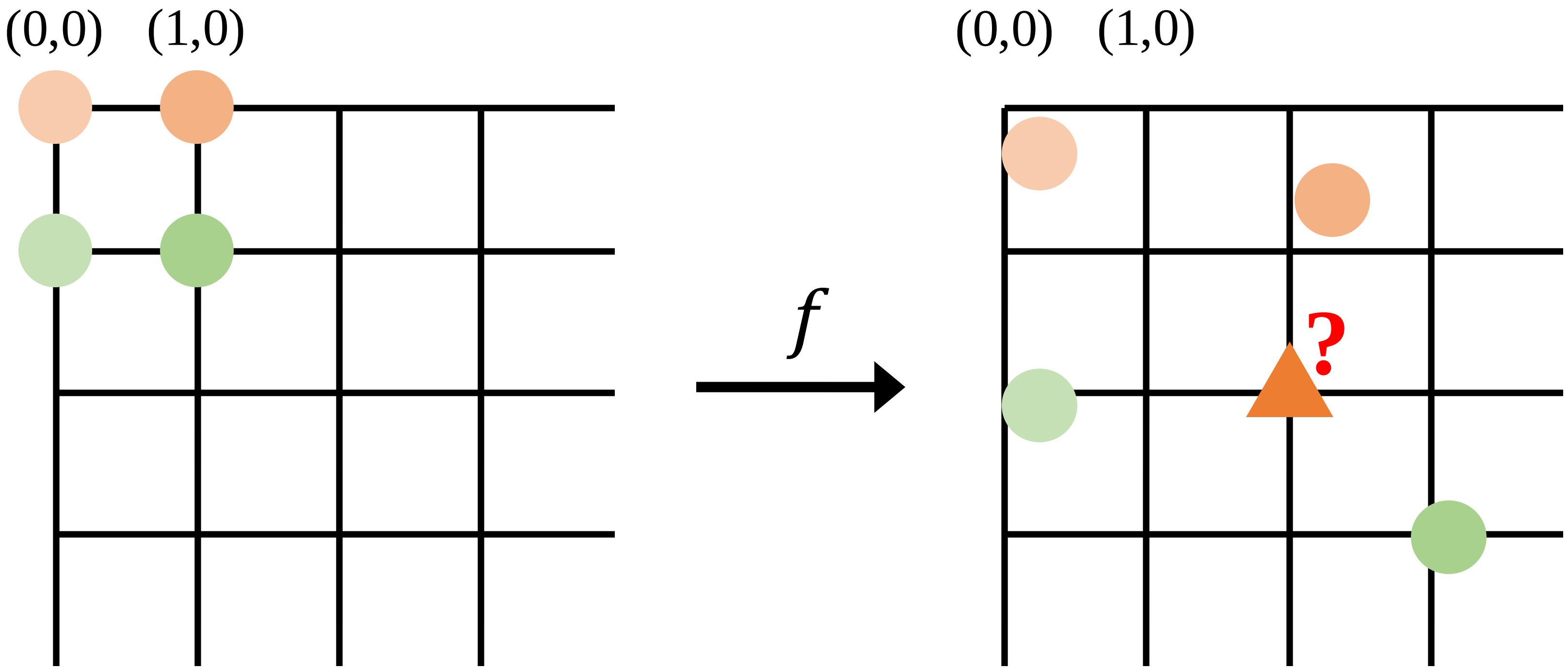
$$I = (1-\alpha)(1-\beta)I_0 + \alpha(1-\beta)I_1 + (1-\alpha)\beta I_2 + \alpha\beta I_3$$



weighted average of 4 corners

What about arbitrary transformation?

- Not a square region: how to do bilinear?

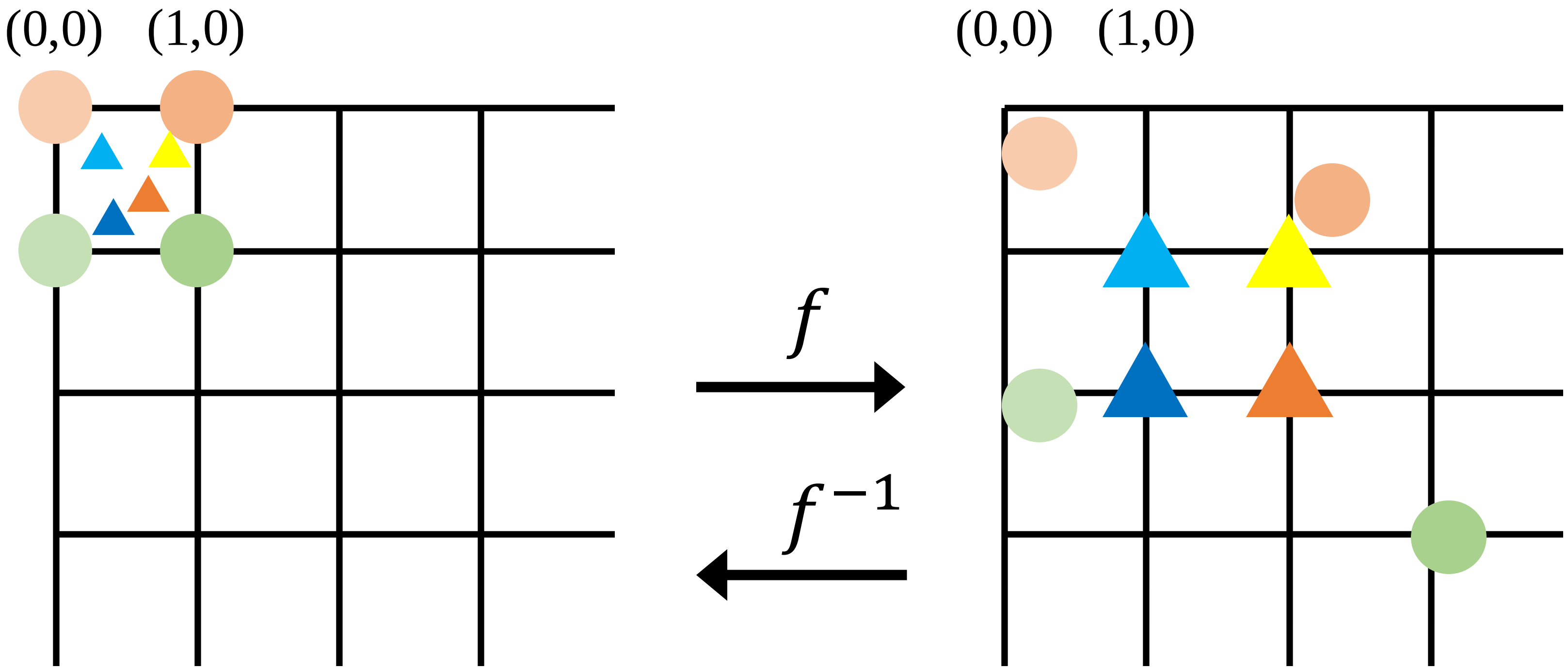


Input image
(x_i, y_i)

Transformed image

Backward mapping

- Back to a square region: easy to do bilinear



Input image
(x_i, y_i)

Transformed image

Summary: image transformation

- What: point cloud $(x_i, y_i, I_i = I[x_i, y_i])$

0, 0, 83
1, 0,100
2, 0, 240
0, 1, 222
1, 1, 239
2, 1, 159

- How: point transformation (***linear** transformation*)

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \to \begin{bmatrix} ax_1 + by_1 + c \\ dx_1 + ey_1 + f \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

- How: Image rendering (*interpolation*)

$$\begin{bmatrix} x \\ y \end{bmatrix} \to \begin{bmatrix} x' = f_0(x, y \\ y' = f_1(x, y) \end{bmatrix}$$

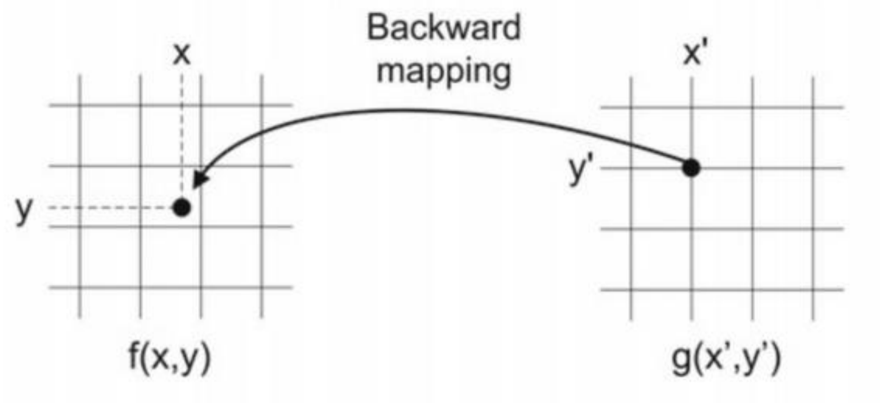
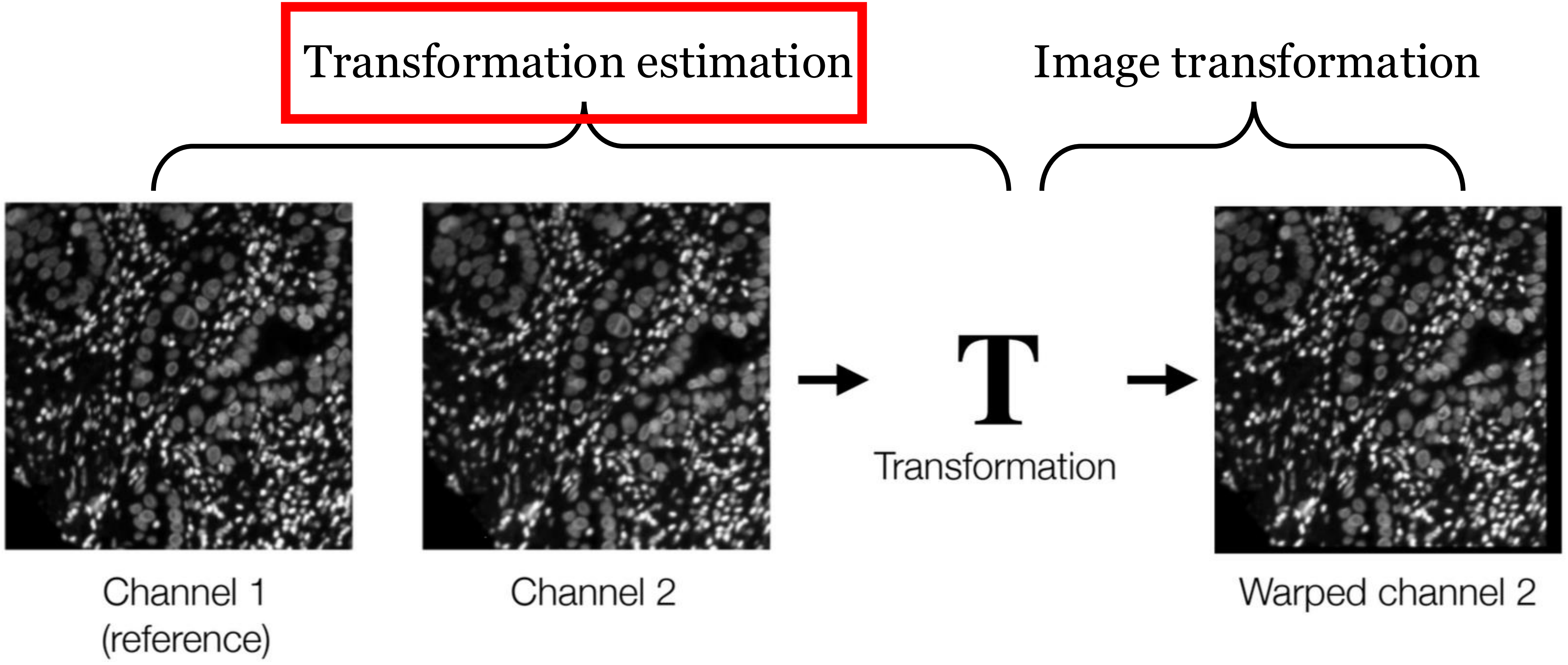


Image registration

- Goal: Register two images



Transformation estimation

- Method 1: intensity-based approach for translation
- Im1 + Im2 → T

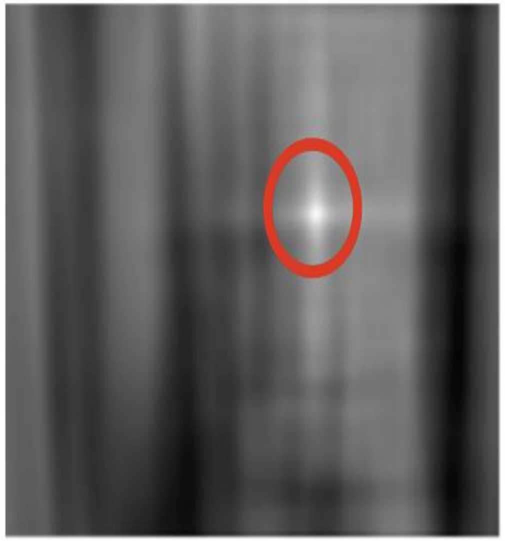
Im1 Im2

Template → matches (normalized cross-correlation)

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

Im1

Im2 patch



For each im1 patch, normalized dot-product

Matches → transformation

Im1

Im2 patch

Im1: matched position (x_1, y_1)

Im2: cropped position (x_2, y_2)

First move to (0, 0), then (x_1, y_1)

Warp Im2

$$T: \begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x - x_2 + x_1 \\ y - y_2 + y_1 \end{bmatrix}$$

Translation is not enough

Im1

Im2

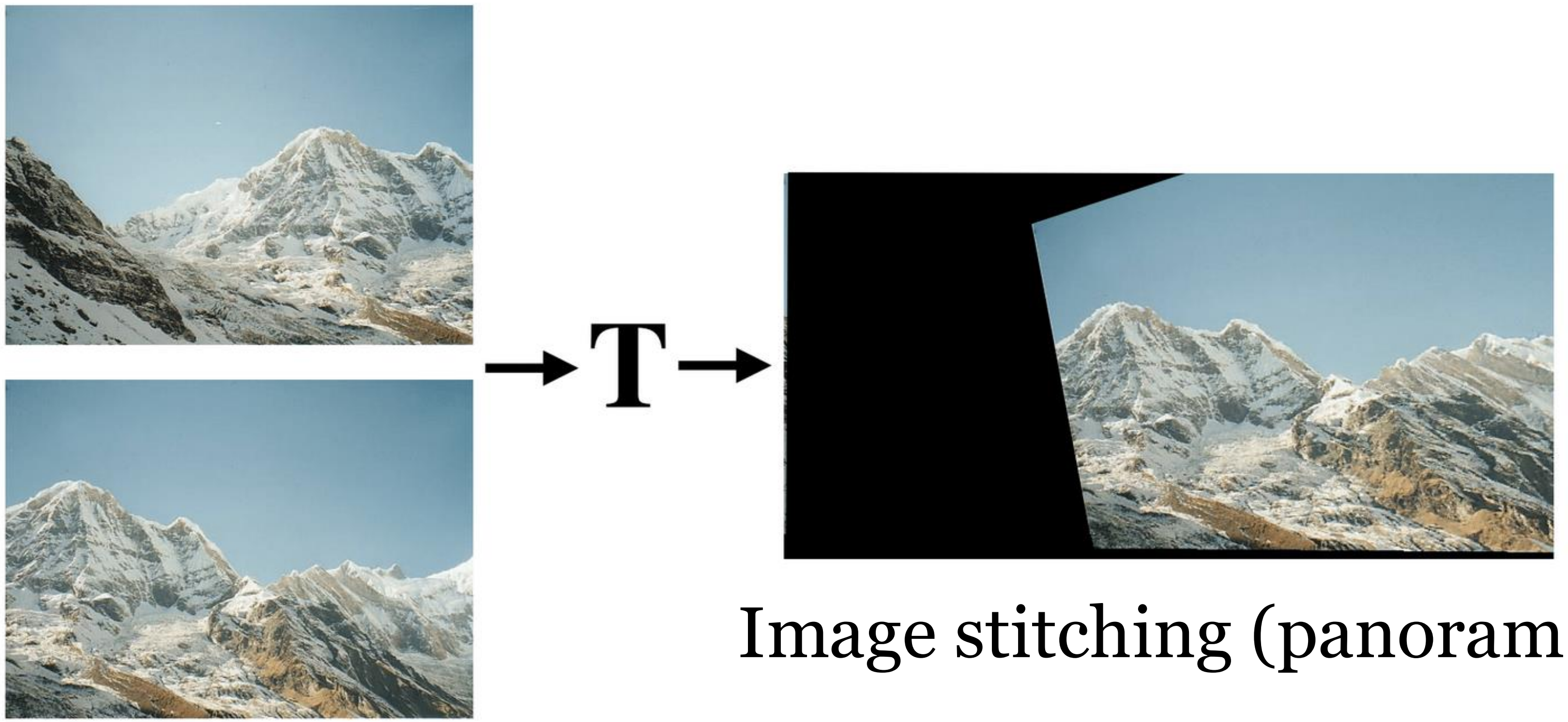


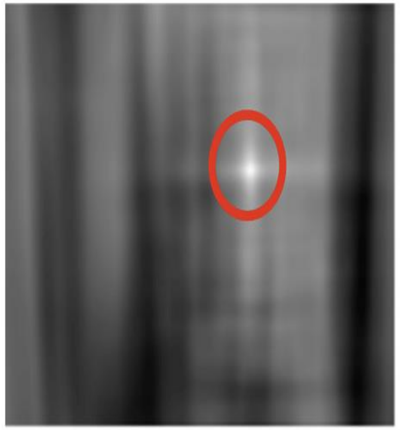
Image stitching (panorama)

Raw images are hard to match
Different size, orientation, lighting, brightness, etc.

Method 2: feature-based approach

1. What to match 2. How to match (NCC) 3. Match → transformation

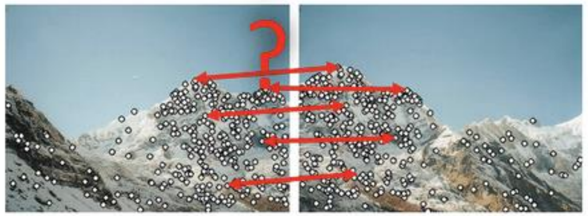
Intensity-based

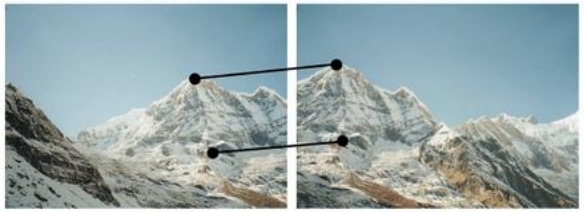


Translation

$$T: \begin{bmatrix} x \\ y \end{bmatrix} \to \begin{bmatrix} x - x_2 + x_1 \\ y - y_2 + y_1 \end{bmatrix}$$

Feature-based





Find key points

Many matches

Find inliers and
$\{(x, y), (x', y')\}_K \to T$