Computer Vision

CVI620

Session 6

Overview

- Point Operations
- Neighborhood Operations
- UpScaling Interpolation

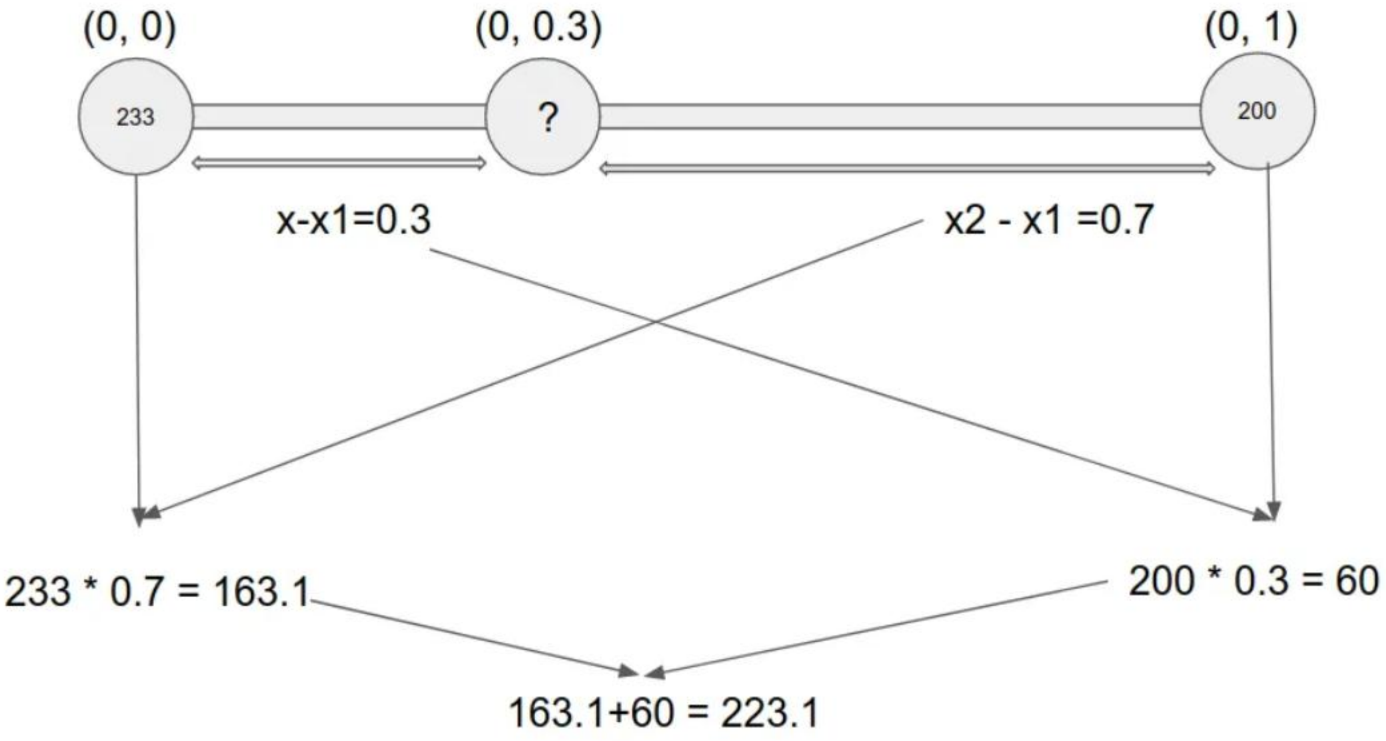
Agenda

- Bilinear Interpolation
- DownScaling Interpolation
- Geometric Transformations
- Affine
- Scale, rotate, reflection, shear, translation
- warpAffine

Bilinear Interpolation

$$I_{new} = \frac{x2 - x}{x2 - x1} * I_1 + \frac{x - x1}{x2 - x1} * I_2$$

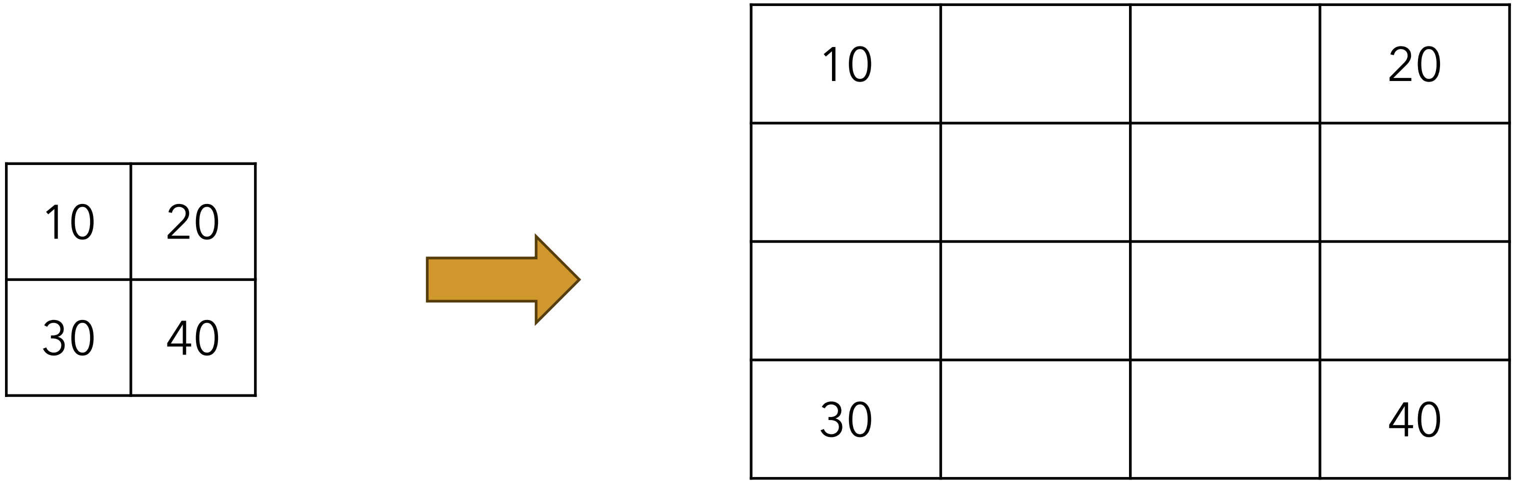
where x1 ≥ x ≥ x2



I1=233, I2=200 and I_new=223.1

Bilinear Interpolation

Initialization and filling pixels are hyperparameters

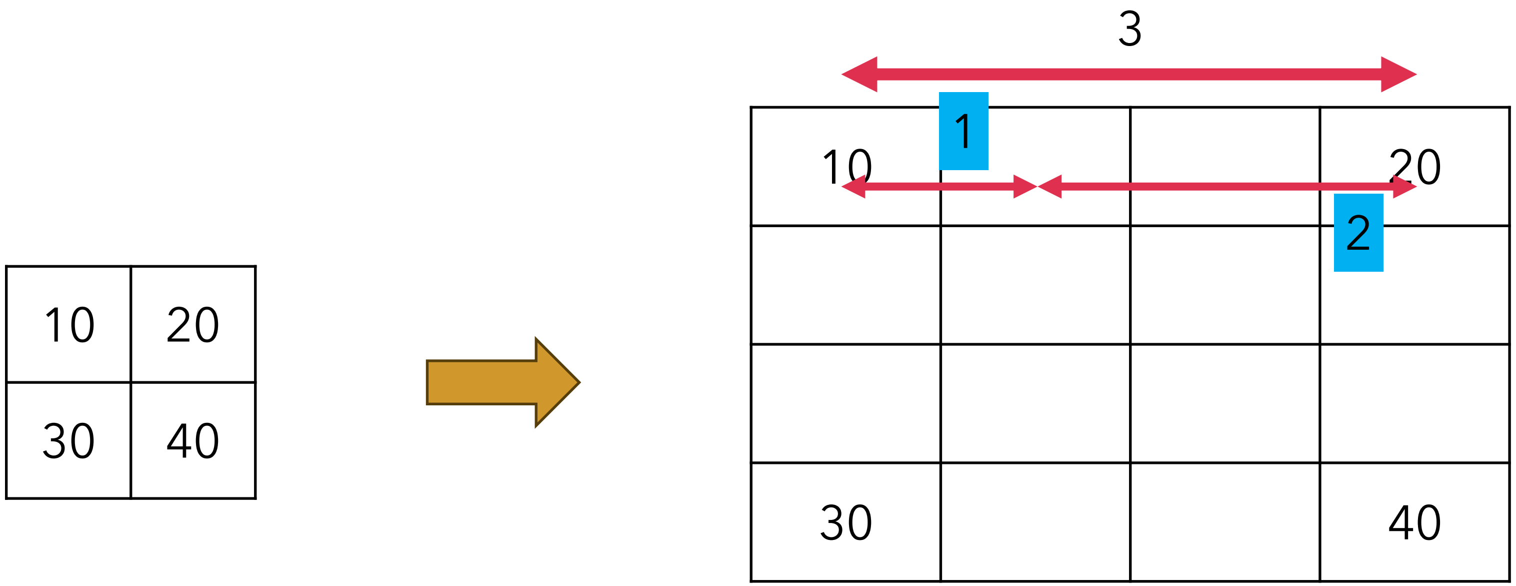


Bilinear Interpolation

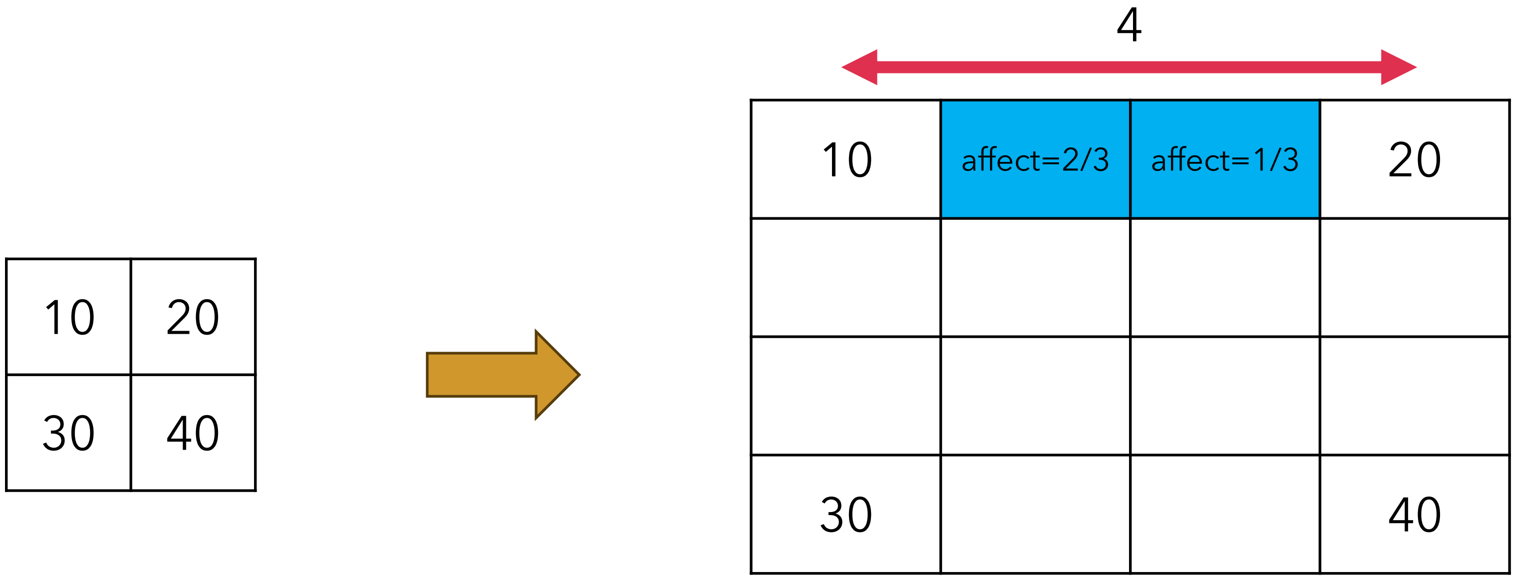
10	20
30	40

	(0,0)	(0,1)	(0,2)	(0,3)
(0,0)	10			20
(0,1)				
(0,2)				
(0,3)	30			40

Bilinear Interpolation

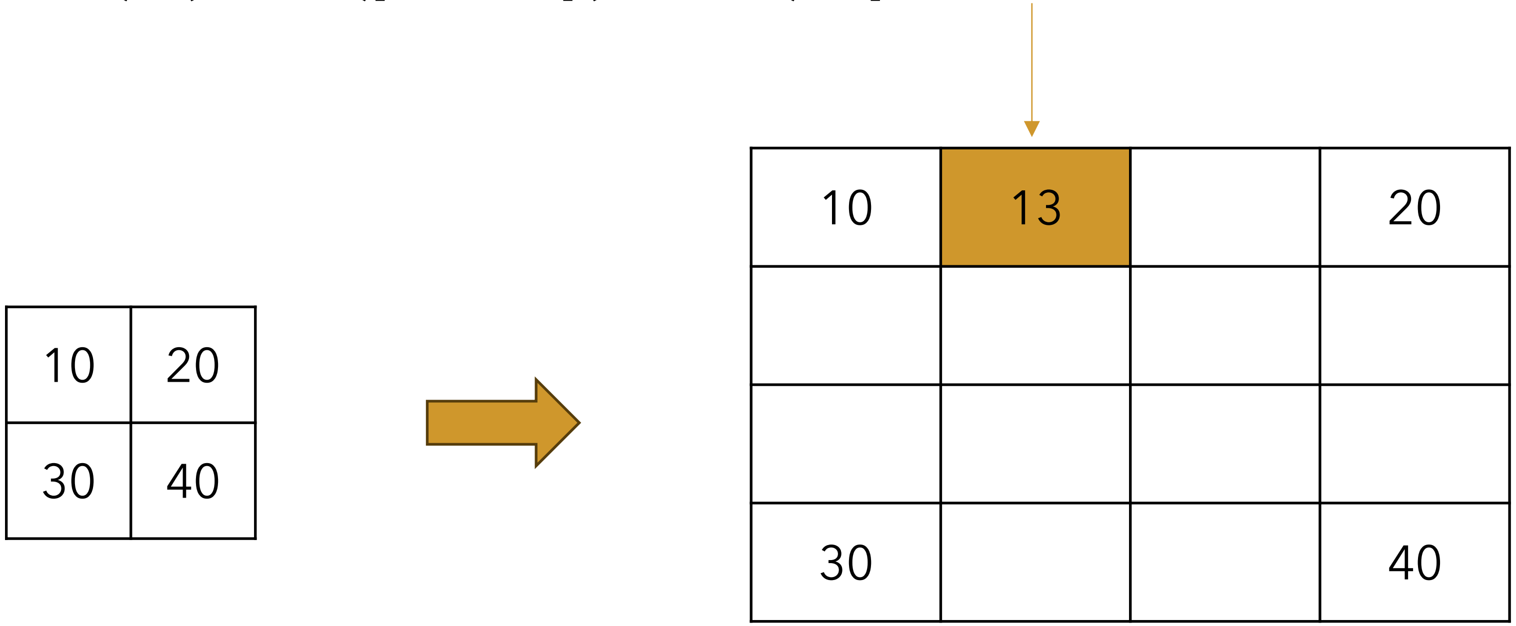


Bilinear Interpolation



Bilinear Interpolation

round [(2/3 * 10) + (1/3 *20)] = round[(6.6 + 6.6) / 2] = 13



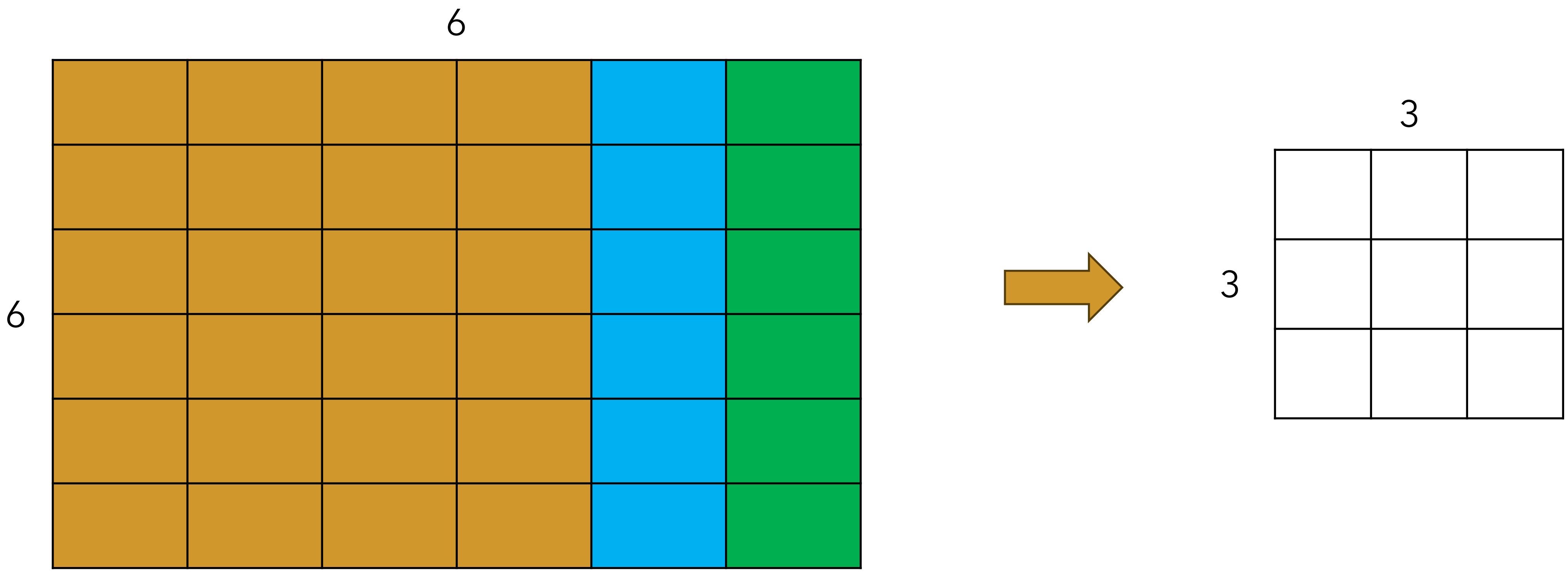
Enumerator	
INTER_NEAREST Python: cv.INTER_NEAREST	nearest neighbor interpolation
INTER_LINEAR Python: cv.INTER_LINEAR	bilinear interpolation
INTER_CUBIC Python: cv.INTER_CUBIC	bicubic interpolation
INTER_AREA Python: cv.INTER_AREA	resampling using pixel area relation. It may be a preferred method for image decimation, as it gives moire'-free results. But when the image is zoomed, it is similar to the INTER_NEAREST method.
INTER_LANCZOS4 Python: cv.INTER_LANCZOS4	Lanczos interpolation over 8x8 neighborhood
INTER_LINEAR_EXACT Python: cv.INTER_LINEAR_EXACT	Bit exact bilinear interpolation
INTER_NEAREST_EXACT Python: cv.INTER_NEAREST_EXACT	Bit exact nearest neighbor interpolation. This will produce same results as the nearest neighbor method in PIL, scikit-image or Matlab.
INTER_MAX Python: cv.INTER_MAX	mask for interpolation codes
WARP_FILL_OUTLIERS Python: cv.WARP_FILL_OUTLIERS	flag, fills all of the destination image pixels. If some of them correspond to outliers in the source image, they are set to zero
WARP_INVERSE_MAP Python: cv.WARP_INVERSE_MAP	flag, inverse transformation For example, linearPolar or logPolar transforms: • flag is **not** set: $dst(\rho, \phi) = src(x, y)$ • flag is set: $dst(x, y) = src(\rho, \phi)$

Downsamlping

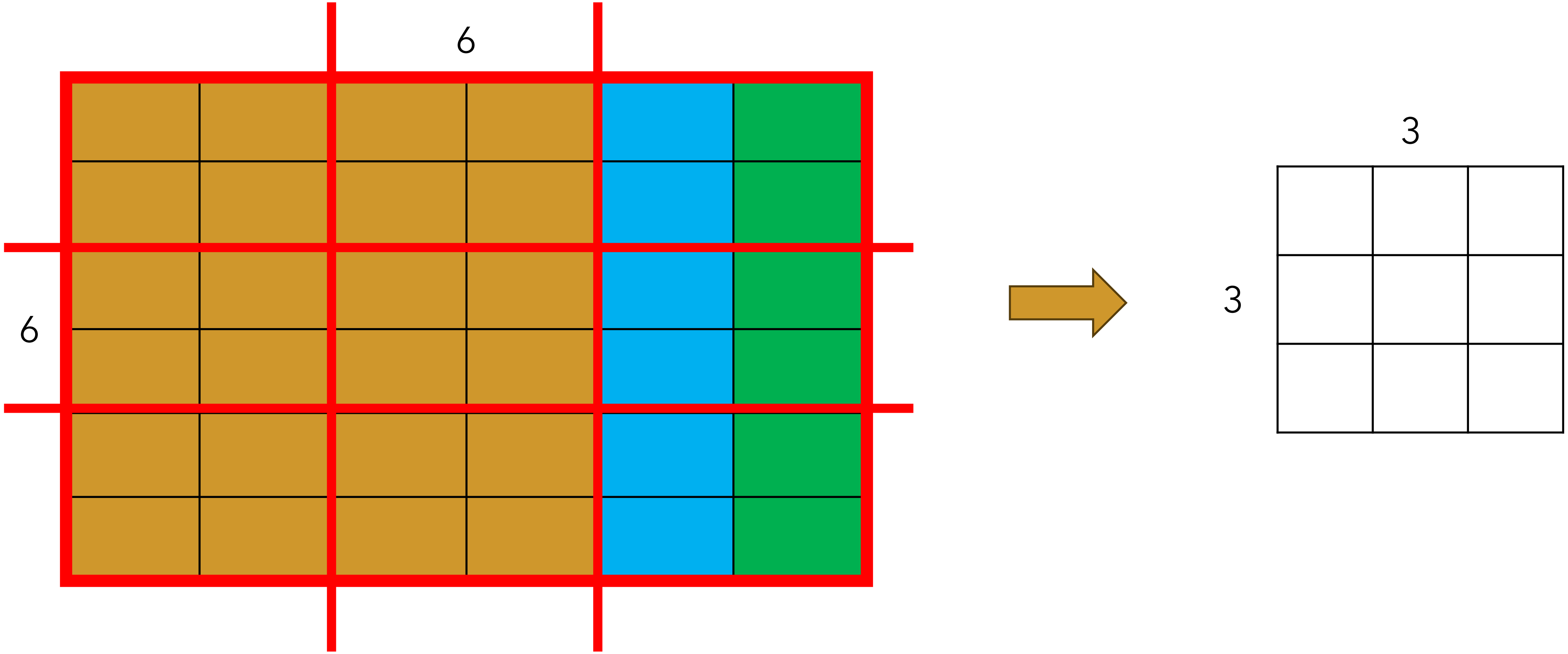
Interpolation is used to estimate the color and intensity values of the new pixels based on the existing pixels.

- Nearest-neighbor
- Bilinear interpolation
- Area interpolation
- Bicubic interpolation
- Lanczos interpolation

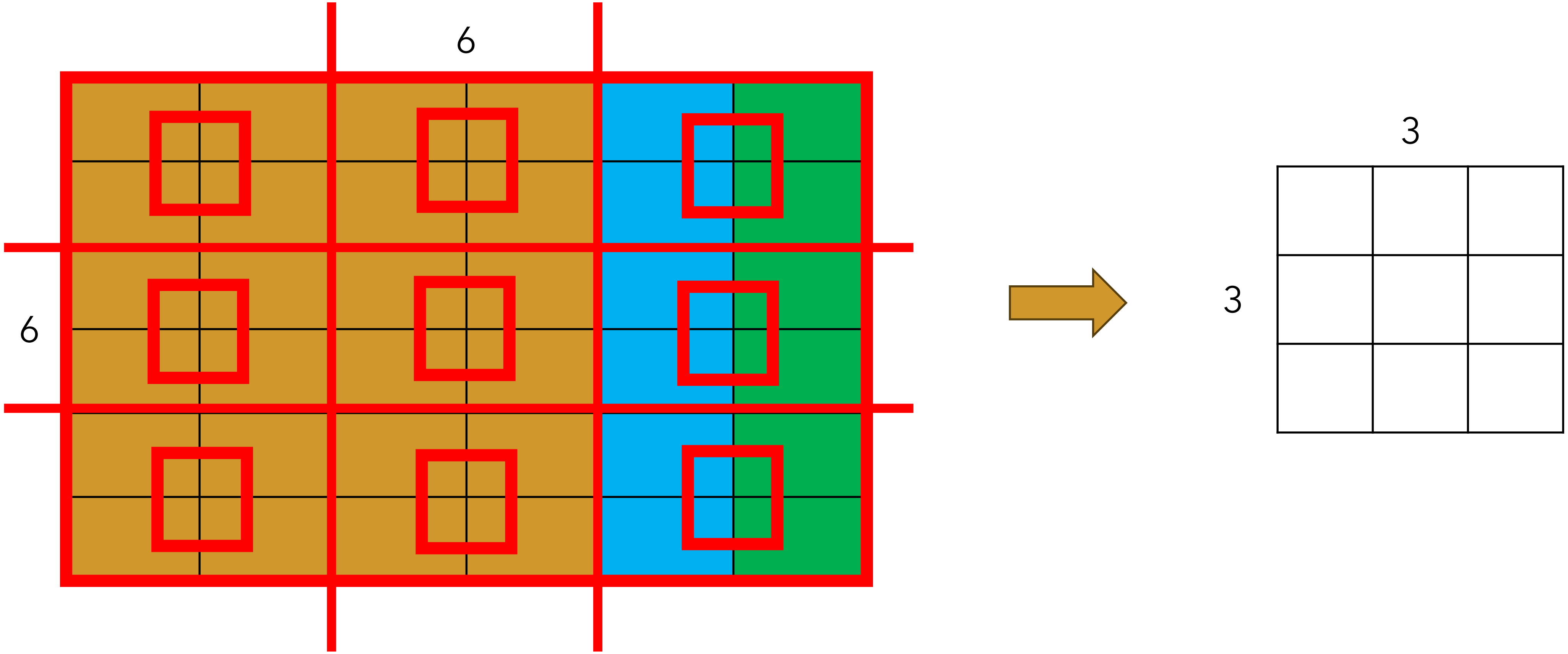
Nearest Neighbor – Max Choice



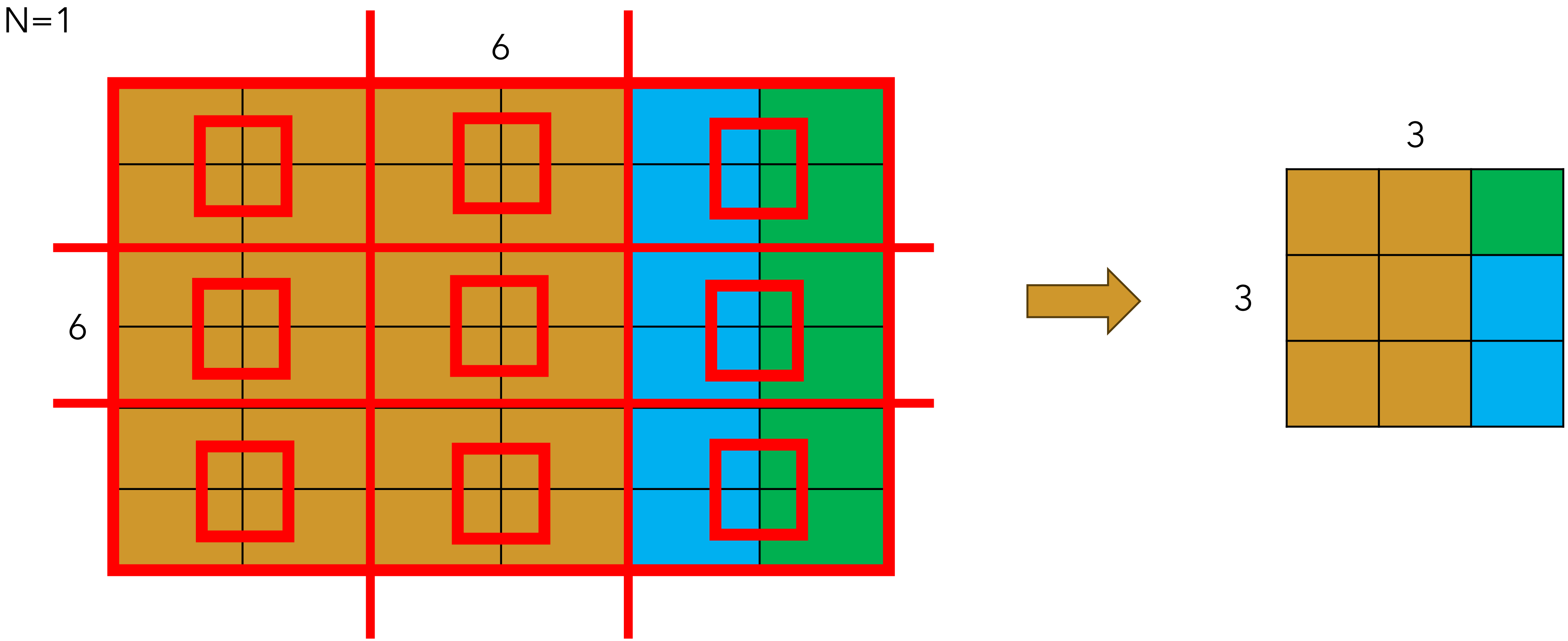
Nearest Neighbor - Max Choice



Nearest Neighbor - Max Choice



Nearest Neighbor - Max Choice



Bilinear Filtering

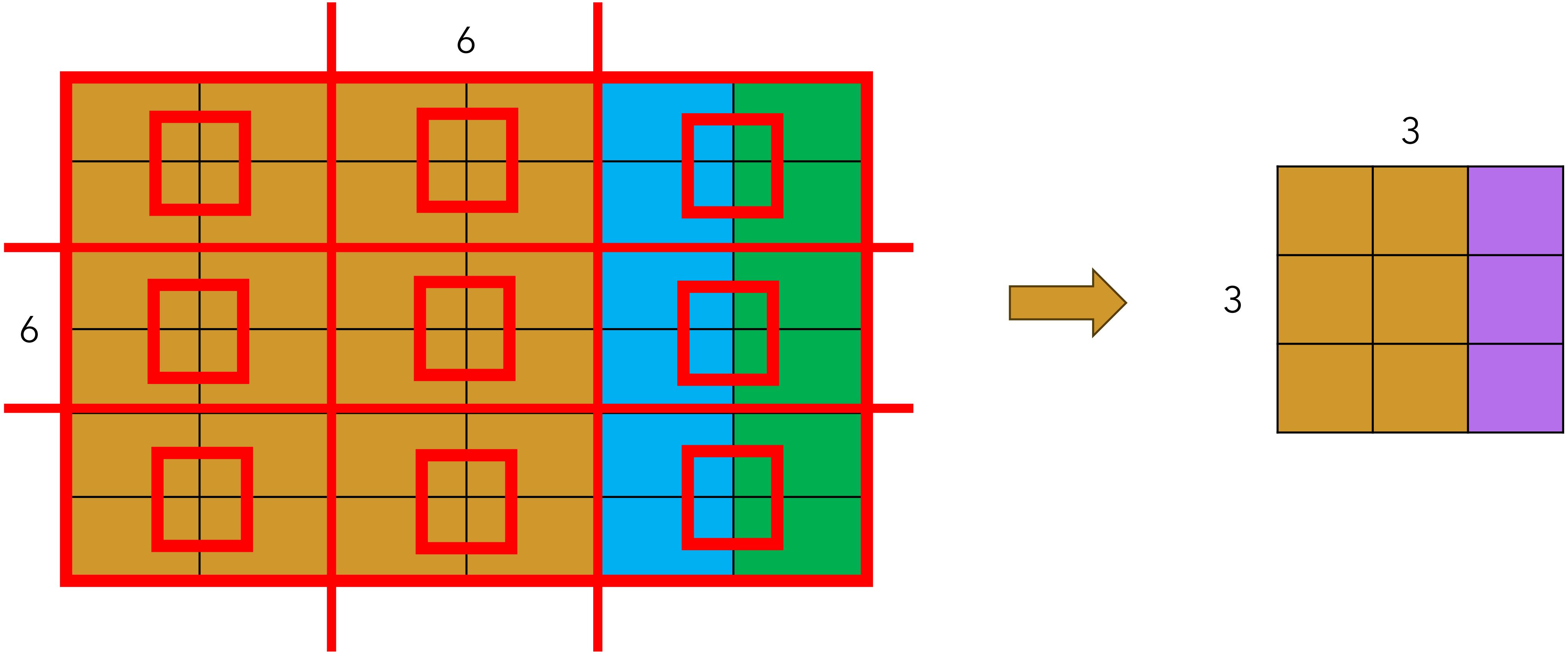
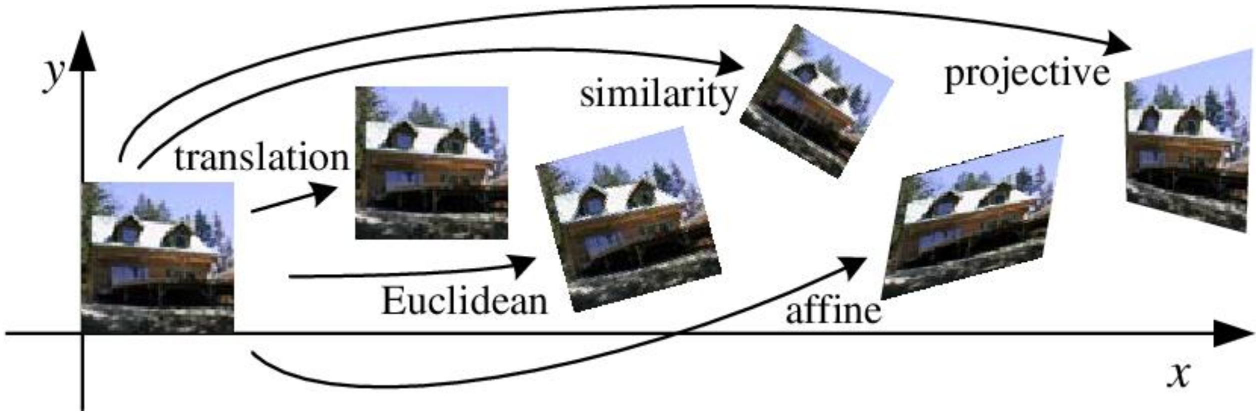
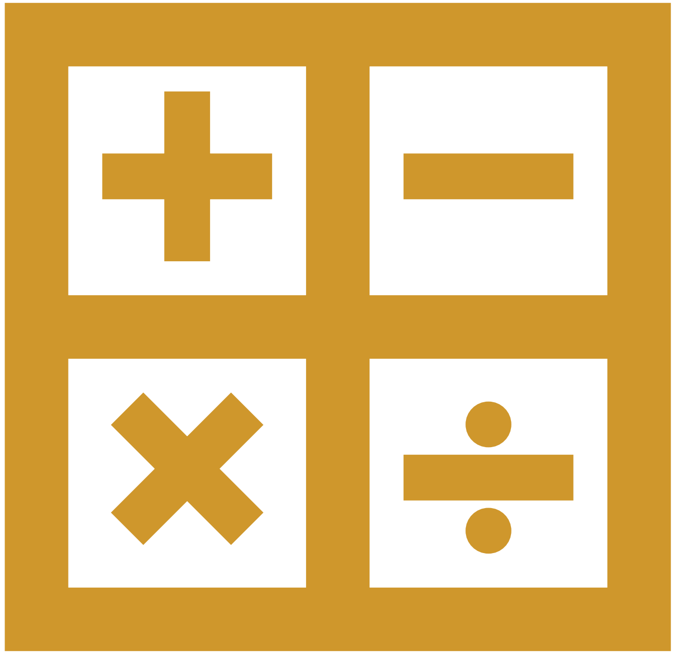


Image Geometric Transformations



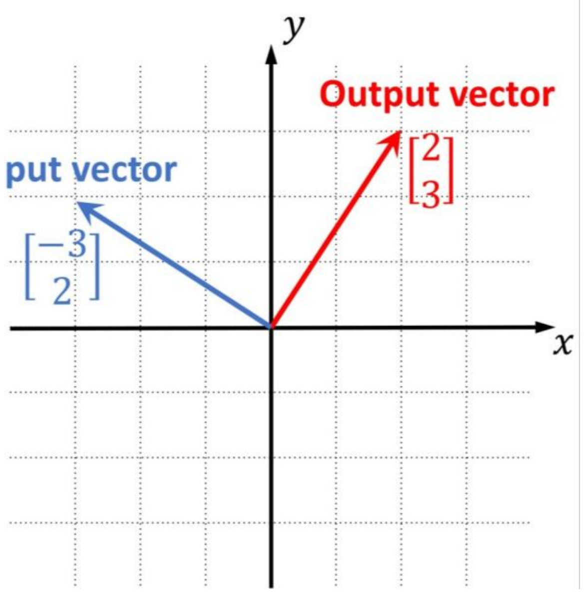
Question?

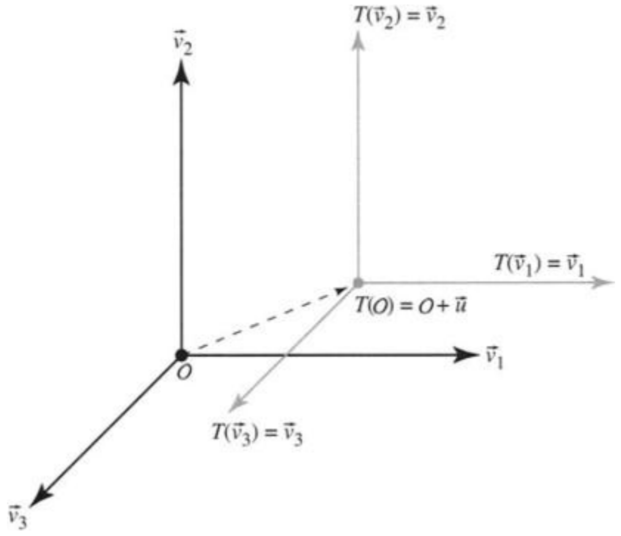
- What do we mean when we had matrix multiplication?
- Why did we learn vector/matrix multiplication/division/addition,…?
- What was the use-case?
- Are we just doing some calculations?



Tip

- If you want to understand any mathematical calculation, imagine it in the space.
- The intuition behind matrix calculations is transformation what do we mean?
- If we are multiplying something to a vector, it means we are changing the vector in the space (this applies to any transformation)
- In transformations, we sometimes change the vector or the space itself

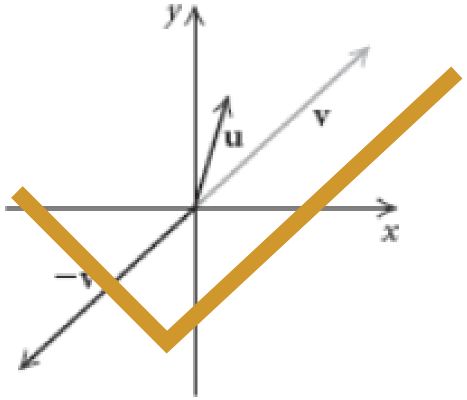




Geometric Transformation

- Mathematical operation that changes the position, size, orientation, or shape of a geometric object in a space

$$Ax = \xcancel{\begin{bmatrix} 6 & 2 & 4 \\ -1 & 4 & 3 \\ -2 & 9 & 3 \end{bmatrix} \begin{bmatrix} 4 \\ -2 \\ 1 \end{bmatrix}} = \begin{bmatrix} 24 \\ -9 \\ 23 \end{bmatrix}$$

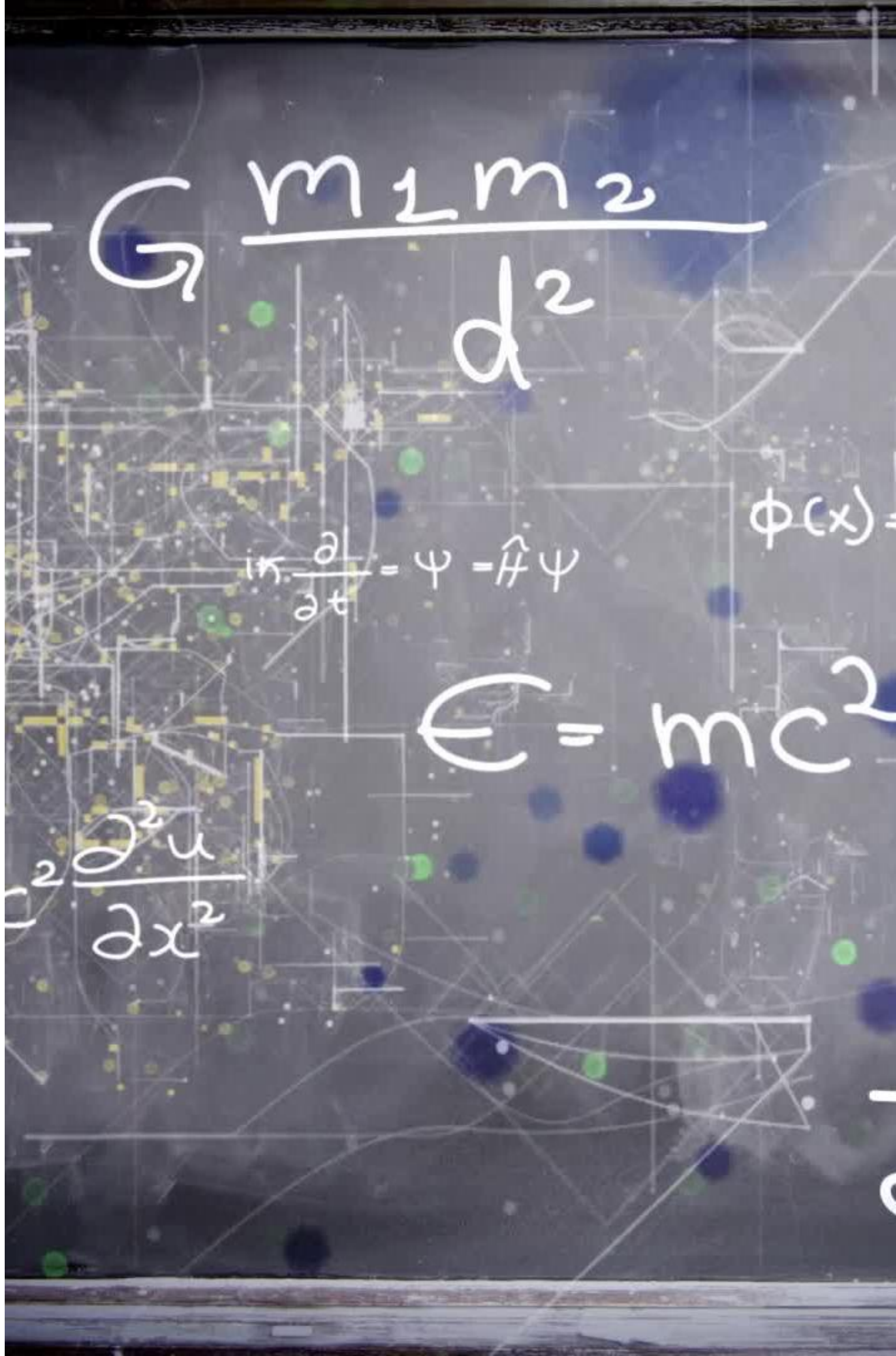


Sketch −v.

Image Transformation

- Most transformations in math and physics can be represented or approximated using matrix operations.
- The same in Computer Vision since images are vectors or matrices

- Rotation: Certain matrices rotate the vector around an origin.
- Shearing: Changes the shape by shifting components of the vector.
- Reflection: Flips the vector across an axis or plane.



Affine Transform

- Linear mapping method that preserves straight lines and ratios of distances
- Position, orientation, and scale can change

- Let's assume we have grayscale transformation

Scale

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Reflection

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Shear

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotation

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

Translation/Shift

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- ***Affine:***

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Translation:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Scale/ Resize:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Rotation:

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

- Shear:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Identity Matrix

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \times$$  $=$ 

Identity Matrix / Horizontal Scale

$$\begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \times$$  $=$ 

Identity Matrix / Horizontal Scale

$$\begin{bmatrix} 6 & 0 \\ 0 & 1 \end{bmatrix} \times$$  $=$ 

Identity Matrix / Horizontal Scale

$$\begin{bmatrix} 1 & 0 \\ 0 & 6 \end{bmatrix} \times$$

$=$

Identity Matrix / Reflection

$$\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix} \times$$

=

Identity Matrix / Horizontal Shear

$$\begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \times$$  $=$ 

Horizontal shear

Identity Matrix / Vertical Shear

$$\begin{bmatrix} 1 & 0 \\ -3 & 1 \end{bmatrix} \times$$

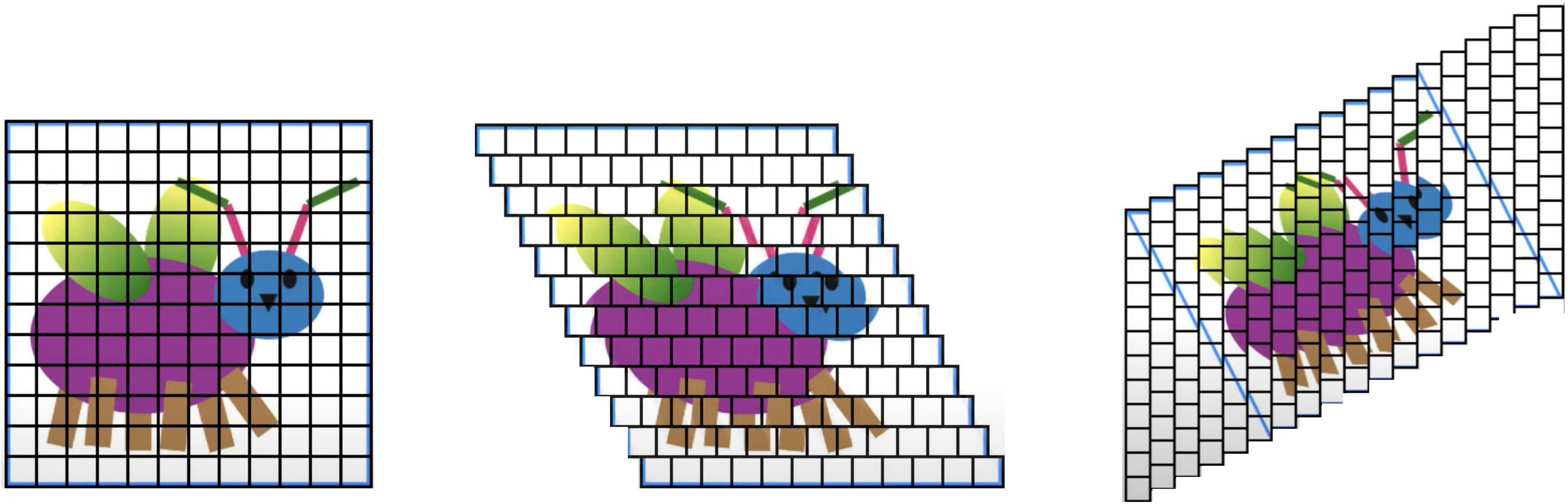
$=$

Identity Matrix / Rotation

Rotation is multiple shears

$$\begin{bmatrix} 1 & -3 \\ 3 & 1 \end{bmatrix} \times$$

$=$



Rotation Matrix

$$\begin{bmatrix} 1 & -a \\ a & 1 \end{bmatrix}$$

Rotation Matrix

degree of rotation

$$\begin{bmatrix} 1 & -\sin \\ \sin & 1 \end{bmatrix}$$

Rotation Matrix

rotation angle and size adjustments

$$\begin{bmatrix} \cos & -\sin \\ \sin & \cos \end{bmatrix}$$

- ***Affine:*** $$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$
- Translation: $$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$
- Scale/ Resize: $$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$
- Rotation: $$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$
- Shear: $$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotate

- The standard 2D rotation matrix for counterclockwise rotation is:

$$R = [[\ \cos\theta \quad -\sin\theta \quad (1-\cos\theta\cdot s)\cdot cx+\sin\theta\cdot s\cdot cy\ \text{ or } 0],$$

$$[\sin\theta \quad \cos\theta \quad (1-\cos\theta\cdot s)\cdot cy-\sin\theta\cdot s\cdot cx\ \text{ or } 0]]$$

- OpenCV uses a clockwise rotation by default, so it flips the sign of

$$R = [[\ \cos\theta \quad \sin\theta \quad (1-\cos\theta\cdot s)\cdot cx-\sin\theta\cdot s\cdot cy],$$

$$[-\sin\theta \quad \cos\theta \quad \sin\theta\cdot cx+(1-\cos\theta\cdot s)\cdot s\cdot cy]]$$

- The third column in OpenCV's 2D rotation matrix represents translation and adjusts the rotated image's position to keep it properly aligned within the output frame.

Rotate

```
import cv2
import numpy as np

image = cv2.imread("Lucy.jpg")

# dimensions of the image
(h, w) = image.shape[:2]

# center of the image
center = (w // 2, h // 2)

# rotation angle in degrees
angle = 45

# the scale (1.0 means no scaling)
scale = 1.0

# rotation matrix
rotation_matrix = cv2.getRotationMatrix2D(center, angle, scale)

# rotate
rotated_image = cv2.warpAffine(image, rotation_matrix, (w, h))

cv2.imshow("Rotated Image", rotated_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Shear

```
import cv2
import numpy as np

image = cv2.imread("image.jpg")

(h, w) = image.shape[:2]

# shear factor
shear_factor_x = 0.5  # horizontal shear
shear_factor_y = 0.2  # vertical shear

# shear matrix
shear_matrix = np.array([
    [1, shear_factor_x, 0],
    [shear_factor_y, 1, 0]
], dtype=np.float32)

# the new width and height after shearing
new_w = int(w + abs(shear_factor_x * h))
new_h = int(h + abs(shear_factor_y * w))

# shear
sheared_image = cv2.warpAffine(image, shear_matrix, (new_w, new_h))

cv2.imshow("Sheared Image", sheared_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```