

# Linear Algebra and Applications

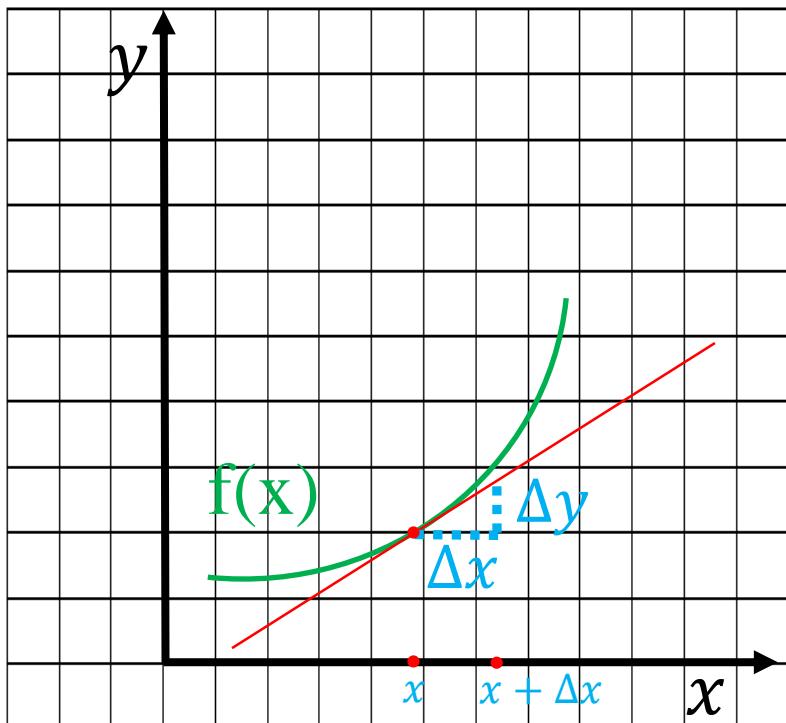
Quang-Vinh Dinh  
Hong-Phuc Nguyen

# Outline

- Derivative/Gradient
- Interpolation
- Vector and Matrix
- Cosine Similarity
- Integral

# Derivative and Applications

## Đạo hàm cho hàm liên tục

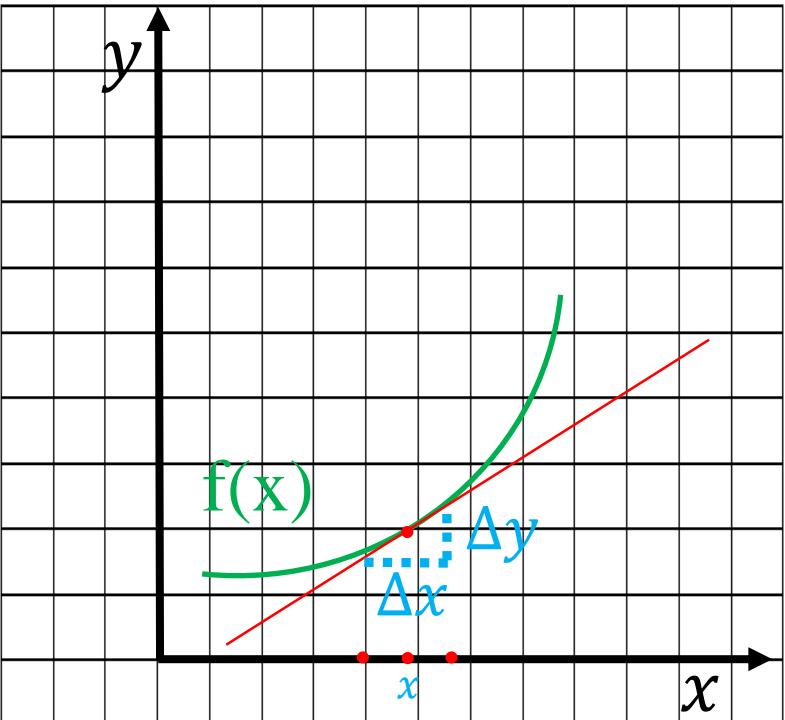


$$\text{Đạo hàm} = \frac{\text{Thay đổi theo } y}{\text{Thay đổi theo } x} = \frac{\Delta y}{\Delta x}$$

$$\frac{d}{dx} f(x) = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

Δx cần tiến về 0 để  
đường tiếp tuyến tiến  
về hàm f(x) trong vùng  
lân cận tại x

## Áp dụng cho hàm rời rạc



Đạo hàm =  $\frac{\text{Thay đổi theo } y}{\text{Thay đổi theo } x} = \frac{\Delta y}{\Delta x}$

$$\frac{d}{dx} f(x) = \lim_{\Delta x \rightarrow 0} \frac{f\left(x + \frac{\Delta x}{2}\right) - f\left(x - \frac{\Delta x}{2}\right)}{\Delta x}$$

32	30	45	36	160	156	155	170
----	----	----	----	-----	-----	-----	-----

$$\Delta x = 2$$

$$\frac{d}{dx} f(x) = \frac{f(x+1) - f(x-1)}{2} = \frac{156 - 36}{2} = 60$$



-1	0	1
----	---	---

x-derivative filter

# Derivative and Applications

Tính đạo hàm trung bình theo hướng x

1
2
1

weighted  
average

-1	0	1
----	---	---

x-derivative

-1	0	1
-2	0	2
-1	0	1

Sobel for x direction

Tính đạo hàm trung bình theo hướng y

1
0
-1

y-derivative

1	2	1
---	---	---

weighted  
average

1	2	1
0	0	0
-1	-2	-1

Sobel for y direction

# Derivative and Applications

## Ứng dụng đạo hàm cho edge detection



Edge  
detection  
→

Sobel-X



$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I$$



Sobel-Y

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

# Derivative and Applications

## ❖ Code

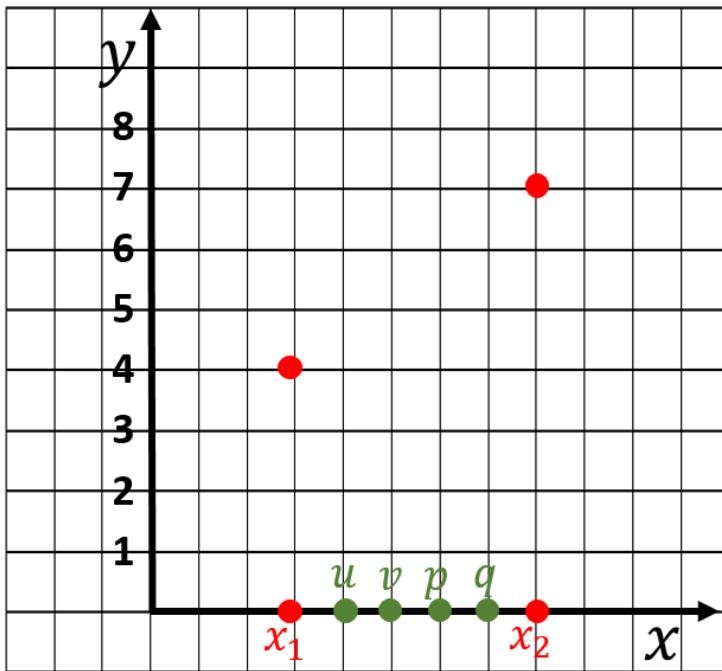
```
1 import numpy as np
2 import cv2
3
4 # read image and convert to grayscale
5 img1 = cv2.imread('vn.jpeg', 0)
6
7 # compute sobel-x
8 sobelx = cv2.Sobel(img1, cv2.CV_64F, 1, 0)
9
10 # compute sobel-y
11 sobely = cv2.Sobel(img1, cv2.CV_64F, 0, 1)
12
13 # save results
14 cv2.imwrite('vn_edge_x.jpg', sobelx)
15 cv2.imwrite('vn_edge_y.jpg', sobely)
16 cv2.imwrite('vn_grayscale.jpg', img1)
```

# Outline

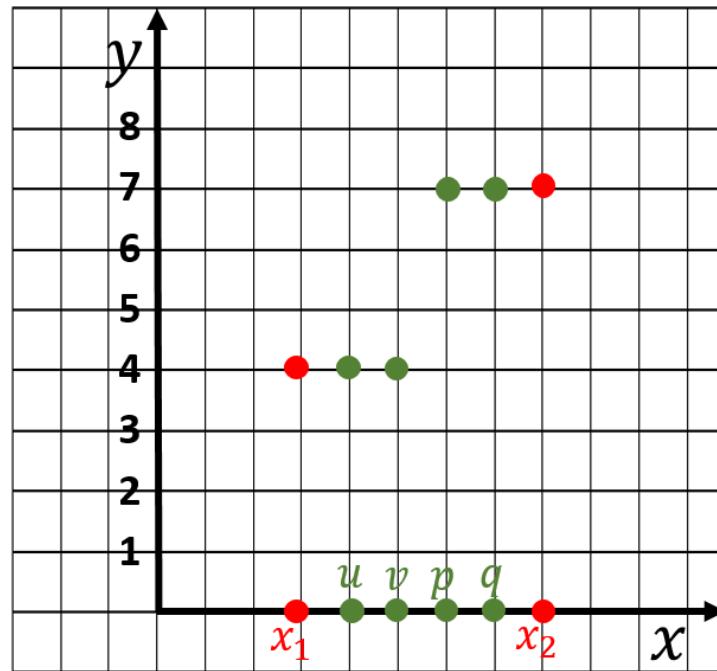
- Derivative/Gradient
- Interpolation
- Vector and Matrix
- Cosine Similarity
- Integral

# Interpolation

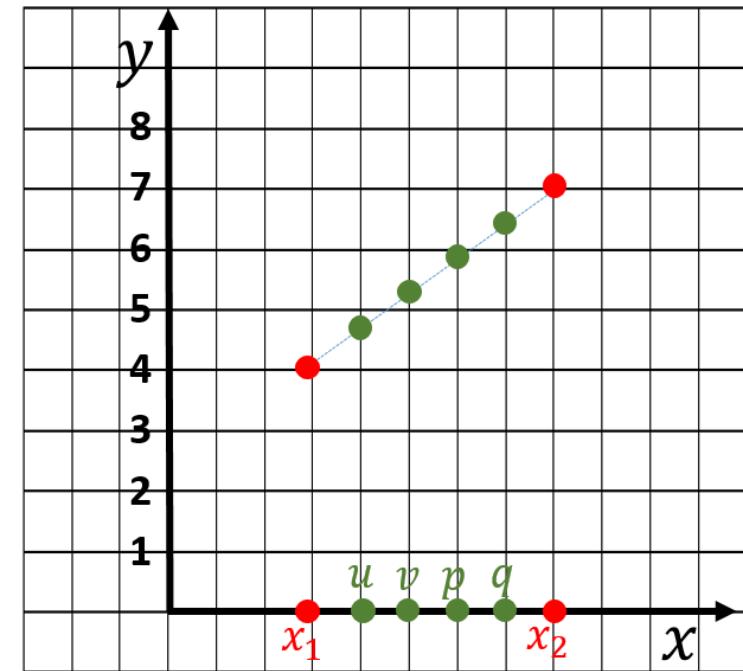
## ❖ Two simplest techniques



Tìm giá trị cho các vị trí  $u$ ,  $v$ ,  $p$  và  $q$



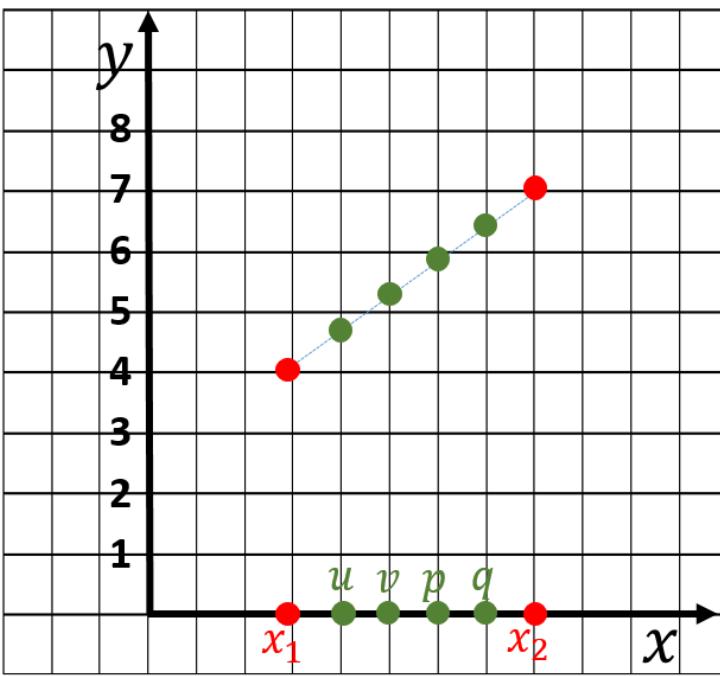
Nearest neighbor: Tính khoảng cách  
đến  $x_1$  và  $x_2$ , và lấy giá trị của  $x$  gần hơn



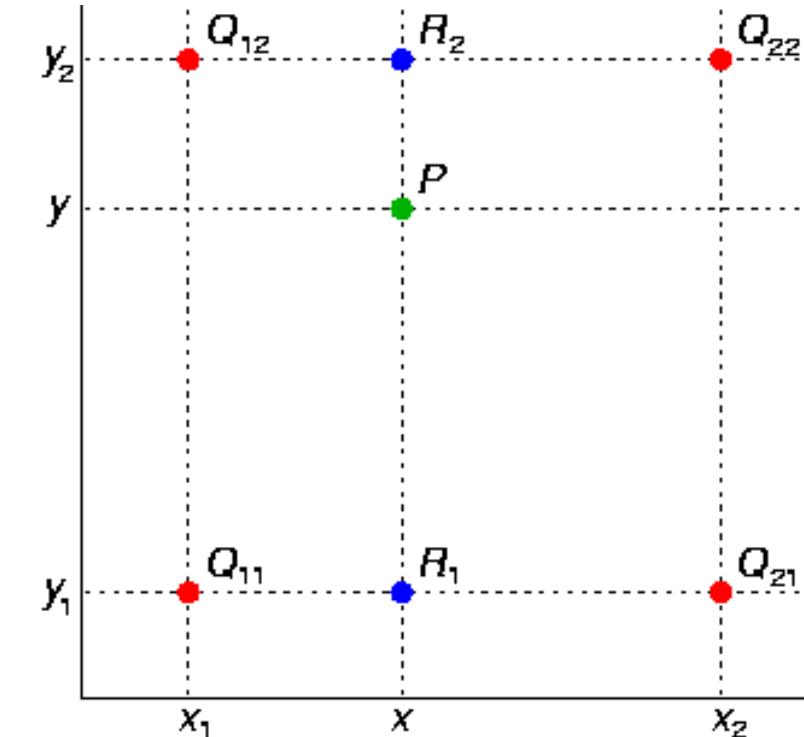
Nội suy theo hàm tuyến tính

# Interpolation

## ❖ Linear and bilinear interpolation



Nội suy theo hàm tuyến tính



[http://en.wikipedia.org/wiki/Bilinear\\_interpolation](http://en.wikipedia.org/wiki/Bilinear_interpolation)

# Interpolation

## ❖ Image upsampling

Phóng to ảnh (upsampling/super-resolution) có thể giải quyết bằng phép nội suy; tìm giá trị cho các điểm ảnh mới ở ảnh phóng to, dựa vào thông tin từ ảnh gốc.



Ảnh gốc



Ảnh phóng to dùng nearest neighbor



Ảnh phóng to dùng hàm tuyến tính

# Interpolation

## ❖ Image upsampling



Ảnh gốc



Ảnh phóng to dùng hàm tuyến tính



Advanced Method

# Interpolation

## ❖ Code

```
1 import numpy as np
2 import cv2
3
4 # read a color image
5 img = cv2.imread('image.jpg')
6
7 # get meta-data of the image
8 height, width, channels = img.shape
9
10 # new dimension
11 new_dim = (width*4, height*4)
12
13 # upsampling the image
14 resize_nearest = cv2.resize(img, new_dim, interpolation=cv2.INTER_NEAREST)
15 resize_bilinear = cv2.resize(img, new_dim, interpolation=cv2.INTER_LINEAR)
16
17 # save results
18 cv2.imwrite('resize_nearest.jpg', resize_nearest)
19 cv2.imwrite('resize_bilinear.jpg', resize_bilinear)
```

# Outline

- Derivative/Gradient
- Interpolation
- Vector and Matrix
- Cosine Similarity
- Integral

# Vector & Matrix

## Vector

$n$  is a natural number

$\mathcal{R}$  is a set of real numbers

$\vec{v}$  has a length of  $n$  and contain real numbers

$$\vec{v} \in \mathcal{R}^n$$

$$\vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \in \begin{bmatrix} \mathcal{R} \\ \mathcal{R} \\ \mathcal{R} \end{bmatrix} = \mathcal{R}^3$$

## Matrix

Matrix A has the shape of rectangle

Has  $m$  rows and  $n$  columns

Use capital letter

$$A \in \mathcal{R}^{m \times n}$$

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \in \begin{bmatrix} \mathcal{R} & \mathcal{R} \\ \mathcal{R} & \mathcal{R} \\ \mathcal{R} & \mathcal{R} \end{bmatrix} = \mathcal{R}^{3 \times 2}$$

# Vector Operations

## Addition

$$\vec{v} = \begin{bmatrix} v_1 \\ \dots \\ v_3 \end{bmatrix} \quad \vec{u} = \begin{bmatrix} u_1 \\ \dots \\ u_3 \end{bmatrix}$$

$$\vec{v} + \vec{u} = \begin{bmatrix} v_1 \\ \dots \\ v_3 \end{bmatrix} + \begin{bmatrix} u_1 \\ \dots \\ u_3 \end{bmatrix} = \begin{bmatrix} v_1 + u_1 \\ \dots \\ v_3 + u_3 \end{bmatrix}$$

```
1 def add_vectors(vector1, vector2):
2     """
3         Add corresponding elements between two vectors
4         vector1 and vector2 are with list type
5         output is a vector (list)
6     """
7
8     return [v1+v2 for v1, v2 in zip(vector1, vector2)]
9
10 # Test case
11 vector1 = [1, 2, 3]
12 vector2 = [4, 5, 6]
13
14 output = add_vectors(vector1, vector2)
15 print(output)
```

[5, 7, 9]

# Vector Operations

## Subtraction

$$\vec{v} = \begin{bmatrix} v_1 \\ \dots \\ v_n \end{bmatrix} \quad \vec{u} = \begin{bmatrix} u_1 \\ \dots \\ u_n \end{bmatrix}$$

$$\vec{v} + \vec{u} = \begin{bmatrix} v_1 \\ \dots \\ v_n \end{bmatrix} - \begin{bmatrix} u_1 \\ \dots \\ u_n \end{bmatrix} = \begin{bmatrix} v_1 - u_1 \\ \dots \\ v_n - u_n \end{bmatrix}$$

## Multiply with a number

$$\alpha \vec{u} = \alpha \begin{bmatrix} u_1 \\ \dots \\ u_n \end{bmatrix} = \begin{bmatrix} \alpha u_1 \\ \dots \\ \alpha u_n \end{bmatrix}$$

## Length of a vector

$$\|\vec{u}\| = \sqrt{u_1^2 + \dots + u_n^2}$$

# Matrix Operations

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & \dots & b_{1n} \\ \dots & \dots & \dots \\ b_{m1} & \dots & b_{mn} \end{bmatrix}$$

## Addition

$$A + B = \begin{bmatrix} (a_{11} + b_{11}) & \dots & (a_{1n} + b_{1n}) \\ \dots & \dots & \dots \\ (a_{m1} + b_{m1}) & \dots & (a_{mn} + b_{mn}) \end{bmatrix}$$

## Subtraction

$$A - B = \begin{bmatrix} (a_{11} - b_{11}) & \dots & (a_{1n} - b_{1n}) \\ \dots & \dots & \dots \\ (a_{m1} - b_{m1}) & \dots & (a_{mn} - b_{mn}) \end{bmatrix}$$

# Matrix Operations

## Multiplication

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & \dots & b_{1k} \\ \dots & \dots & \dots \\ b_{n1} & \dots & b_{nk} \end{bmatrix}$$
$$A \in \mathcal{R}^{m \times n} \quad B \in \mathcal{R}^{n \times k}$$

$$C = AB$$

$$c_{11} = \sum_{l=1}^n a_{il} b_{lj}$$

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \\ a_{31}b_{11} + a_{32}b_{21} & a_{31}b_{12} + a_{32}b_{22} \end{bmatrix}$$

# Matrix Operations

## Multiplication

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \quad \vec{x} = \begin{bmatrix} x_1 \\ \dots \\ x_n \end{bmatrix}$$

$$A \in \mathcal{R}^{m \times n}$$

$$C = A\vec{x}$$

$$c_{11} = \sum_{l=1}^n a_{il}x_l$$

## Example

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 \\ a_{21}x_1 + a_{22}x_2 \\ a_{31}x_1 + a_{32}x_2 \end{bmatrix}$$

# Matrix Operations

## Multiplication

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & \dots & b_{1k} \\ \dots & \dots & \dots \\ b_{n1} & \dots & b_{nk} \end{bmatrix}$$

$A \in \mathcal{R}^{m \times n}$        $B \in \mathcal{R}^{n \times k}$

$$C = AB$$

$$c_{11} = \sum_{l=1}^n a_{il} b_{lj}$$

```
1 def matrix_multiplication(matrix1, matrix2):
2     """
3         This function does the multiplication between two matrices.
4         #columns of matrix1 == #rows of matrix2
5     """
6     matrix1_nrows = len(matrix1)
7     matrix1_ncols = len(matrix1[0])
8
9     matrix2_nrows = len(matrix2)
10    matrix2_ncols = len(matrix2[0])
11
12    # tạo matrix kết quả
13    result = [[0]*matrix2_ncols for i in range(matrix1_nrows)]
14
15    for i in range(matrix1_nrows):
16        for j in range(matrix2_ncols):
17            for k in range(matrix2_nrows):
18                result[i][j] += matrix1[i][k] * matrix2[k][j]
19
20    return result
21
22    # test case
23    # 3x3 matrix
24    matrix1 = [[1, 2, 3],
25                [4, 5, 6],
26                [7, 8, 9]]
27
28    # 3x4 matrix
29    matrix2 = [[1, 1, 2, 1],
30                [1, 2, 1, 1],
31                [1, 1, 1, 2]]
32
33    result = matrix_multiplication(matrix1, matrix2)
34    print(result[0])
35    print(result[1])
36    print(result[2])
```

[6, 8, 7, 9]  
[15, 20, 19, 21]  
[24, 32, 31, 33]

# Matrix Operations

## Transpose

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}$$

$$A^T = \begin{bmatrix} a_{11} & \dots & a_{m1} \\ \dots & \dots & \dots \\ a_{1n} & \dots & a_{mn} \end{bmatrix}$$

## Example

$$\begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 \\ \beta_1 & \beta_2 & \beta_3 \end{bmatrix}^T = \begin{bmatrix} \alpha_1 & \beta_1 \\ \alpha_2 & \beta_2 \\ \alpha_3 & \beta_3 \end{bmatrix}$$

# Matrix and Vector

## Phép nhân giữa ma trận và vector

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

$$Ax = b$$

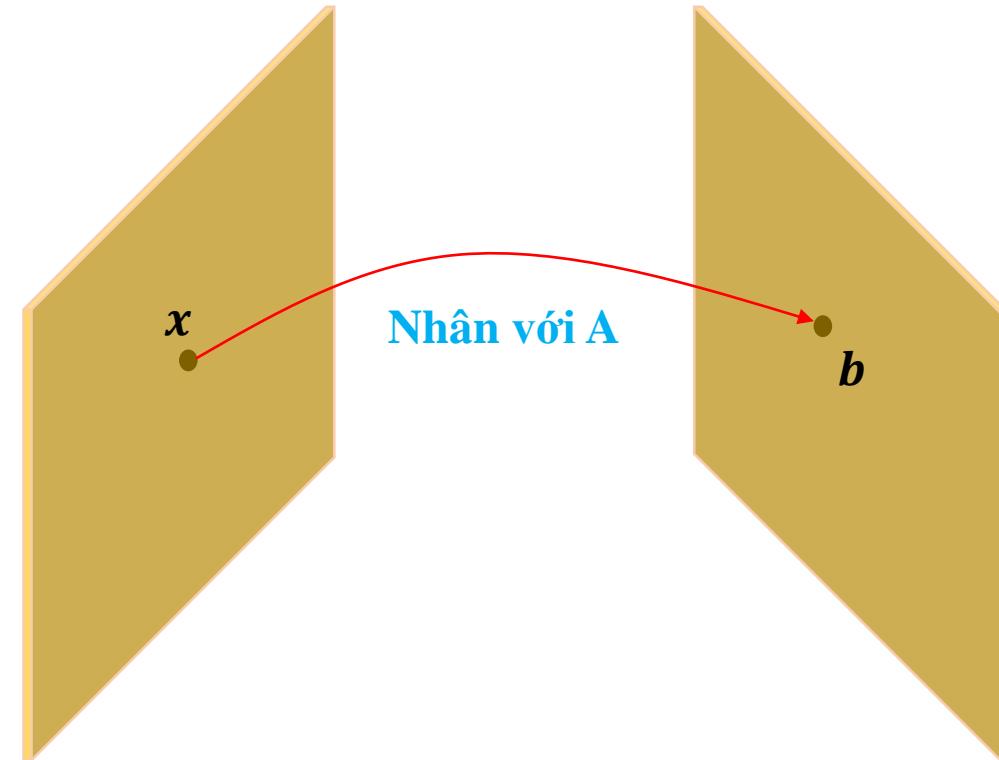
$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 \\ a_{21}x_1 + a_{22}x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

Ma trận A biến đổi/dịch chuyển x sang b

Giá trị từng phần tử của b là tổ hợp tuyến tính của tất cả các phần tử của x

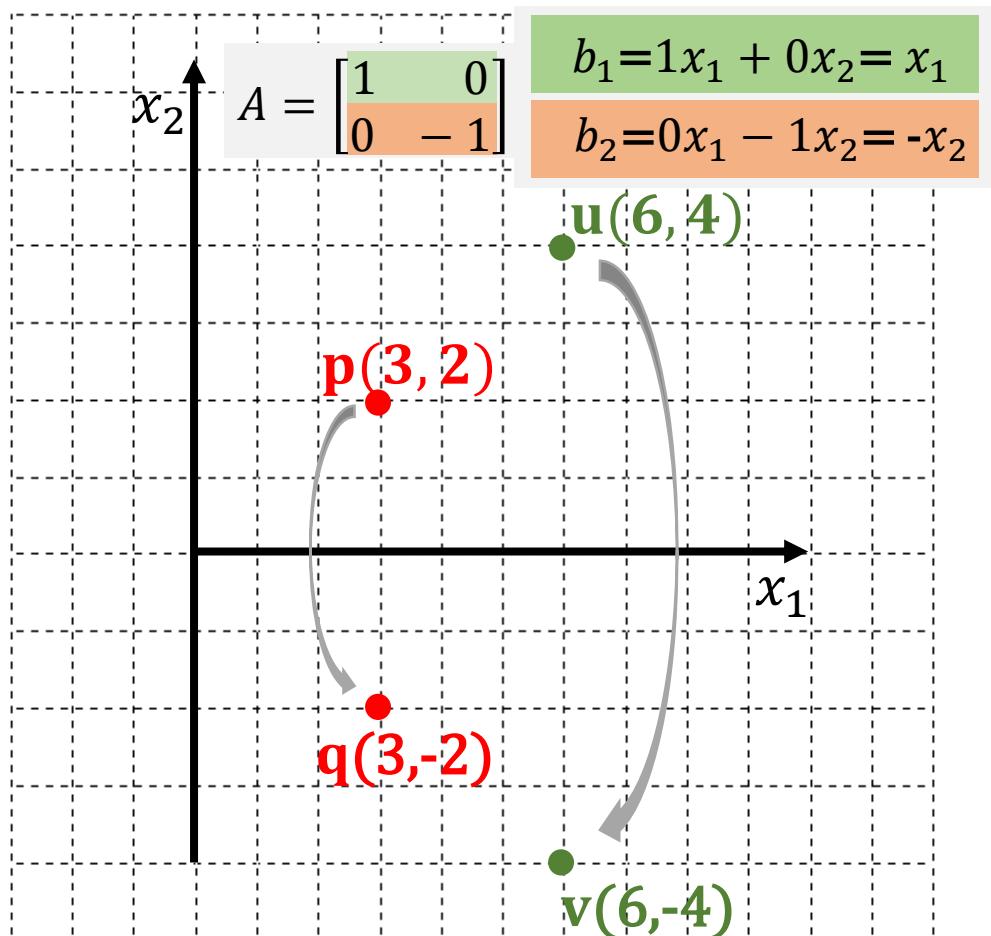
$$b_1 = a_{11}x_1 + a_{12}x_2$$

$$b_2 = a_{21}x_1 + a_{22}x_2$$



# Matrix and Vector

Ma trận A dịch chuyển điểm x đối xứng qua trục  $x_1$



Ứng dụng lật ảnh đối xứng qua trục ngang

Giá trị màu (red, green, blue) của điểm  $p$   
 Các thuộc tính của pixel  $p(x_1, x_2, r, g, b)$   
 Tọa độ điểm  $p$

Dịch chuyển pixel  $(x_1, x_2)$  theo  $A = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$



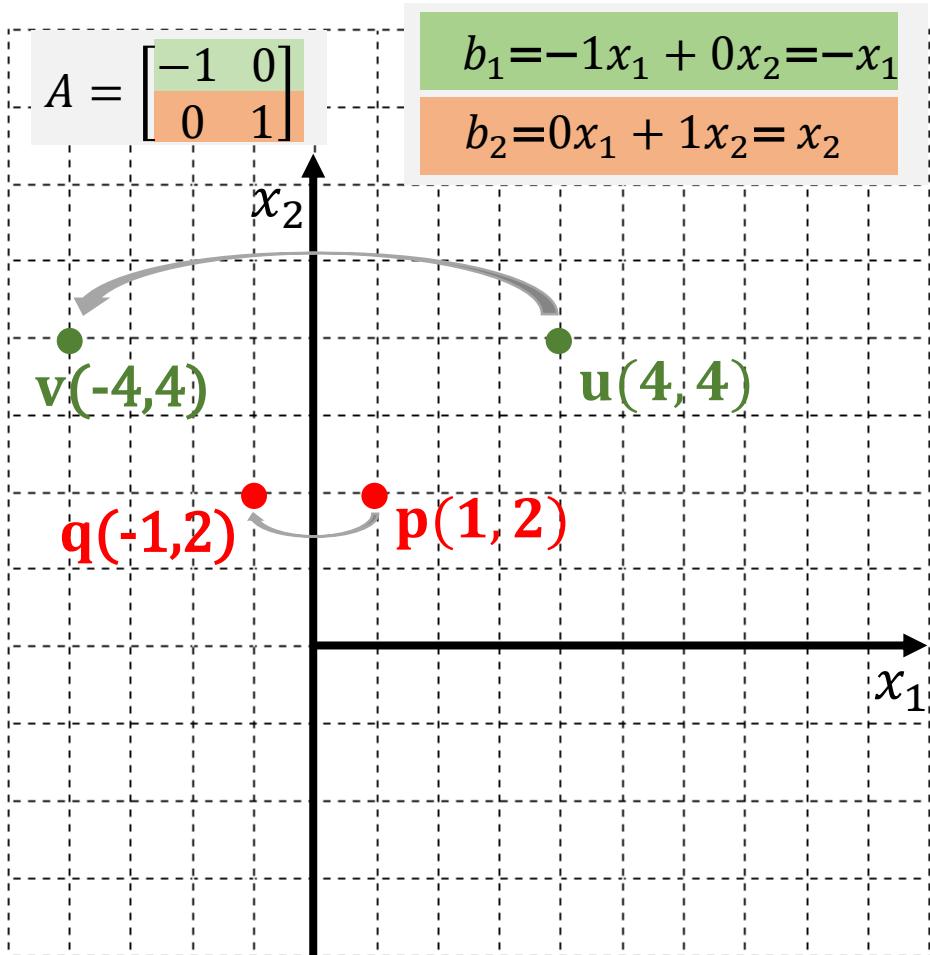
Ảnh gốc



Ảnh kết quả

# Matrix and Vector

Ma trận A dịch chuyển điểm x đối xứng qua trục  $x_2$



Ứng dụng lật ảnh đối xứng qua trục đứng

Các thuộc tính của pixel  $p(x_1, x_2, r, g, b)$

Giá trị màu (red, green, blue) của điểm  $p$

Tọa độ điểm  $p$

Dịch chuyển pixel  $(x_1, x_2)$  theo  $A = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$



Ảnh gốc



Ảnh kết quả

# Background Subtraction

Background



coming image



# Background Subtraction

Background



Coming image

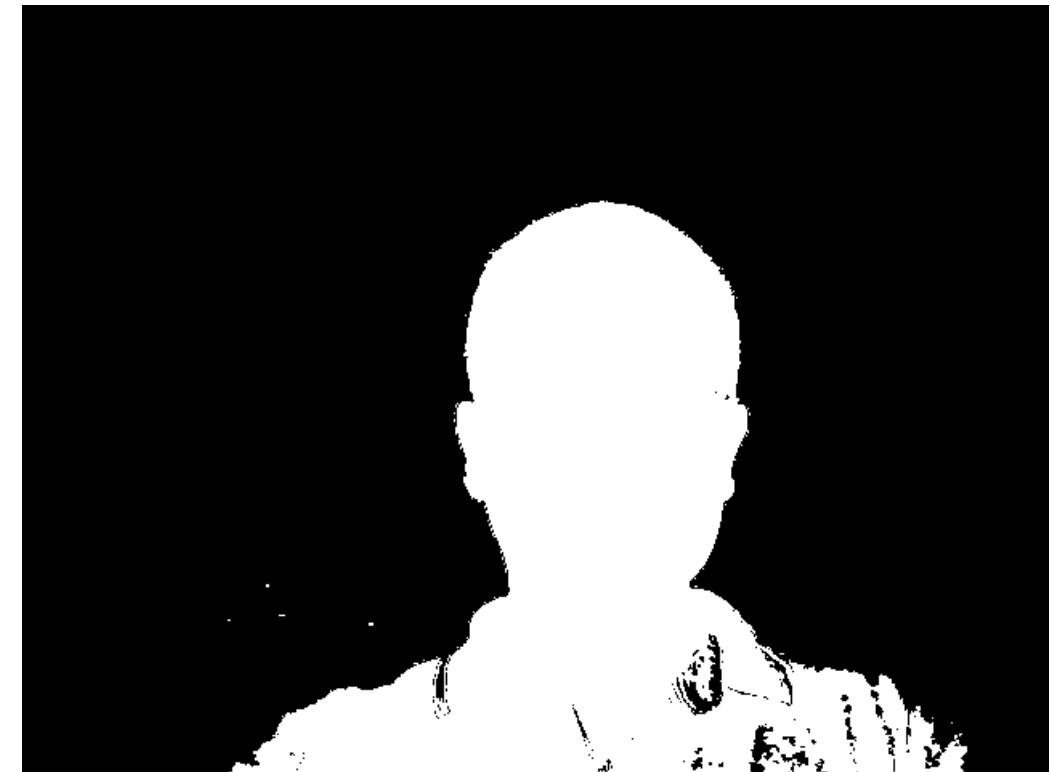


Background Subtraction

# Background Subtraction



Subtracted Background



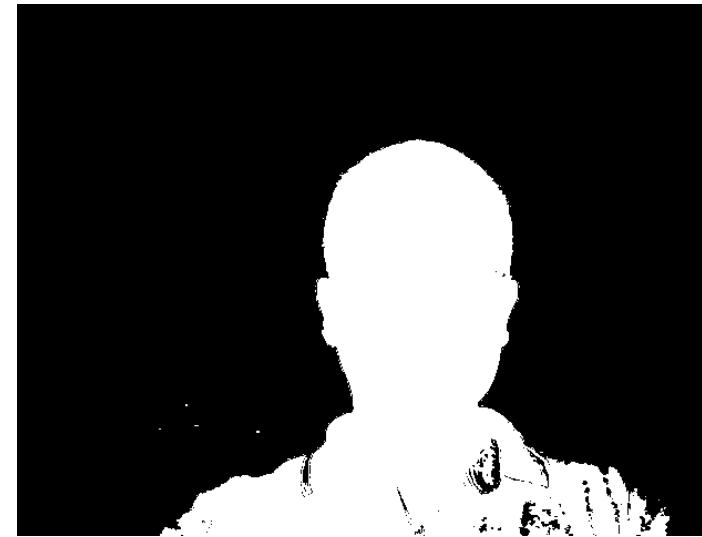
Mask

# Background Subtraction

Background



Coming image

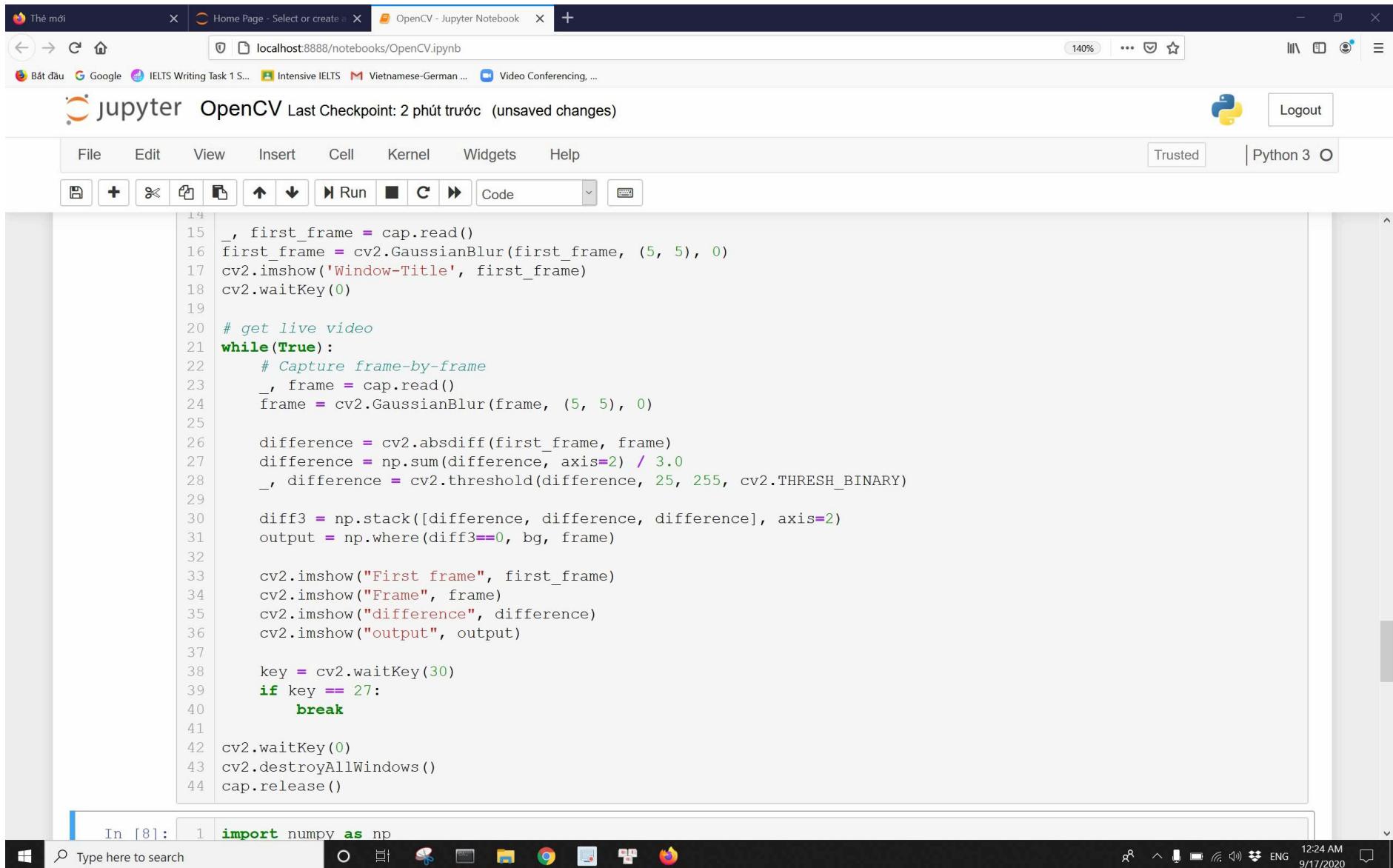


Mask



FakeBackground

# Background Subtraction



The screenshot shows a Jupyter Notebook interface with the title "jupyter OpenCV Last Checkpoint: 2 phút trước (unsaved changes)". The notebook contains the following Python code for background subtraction:

```
14
15 _, first_frame = cap.read()
16 first_frame = cv2.GaussianBlur(first_frame, (5, 5), 0)
17 cv2.imshow('Window-Title', first_frame)
18 cv2.waitKey(0)
19
20 # get live video
21 while(True):
22     # Capture frame-by-frame
23     _, frame = cap.read()
24     frame = cv2.GaussianBlur(frame, (5, 5), 0)
25
26     difference = cv2.absdiff(first_frame, frame)
27     difference = np.sum(difference, axis=2) / 3.0
28     _, difference = cv2.threshold(difference, 25, 255, cv2.THRESH_BINARY)
29
30     diff3 = np.stack([difference, difference, difference], axis=2)
31     output = np.where(diff3==0, bg, frame)
32
33     cv2.imshow("First frame", first_frame)
34     cv2.imshow("Frame", frame)
35     cv2.imshow("difference", difference)
36     cv2.imshow("output", output)
37
38     key = cv2.waitKey(30)
39     if key == 27:
40         break
41
42 cv2.waitKey(0)
43 cv2.destroyAllWindows()
44 cap.release()
```

The code imports numpy and cv2, reads a video file, applies Gaussian blur to the first frame, and then enters a loop to capture frames, calculate the absolute difference between the first frame and each subsequent frame, apply a threshold to create a binary mask, stack the mask three times to create a 3D array, and finally display the original frame, the difference, and the output frame. It also handles user input to break the loop.

# Outline

- Derivative/Gradient
- Interpolation
- Vector and Matrix
- Cosine Similarity
- Integral

# Vector Operations

## Dot product

$$\vec{v} = \begin{bmatrix} v_1 \\ \dots \\ v_n \end{bmatrix} \quad \vec{u} = \begin{bmatrix} u_1 \\ \dots \\ u_n \end{bmatrix}$$

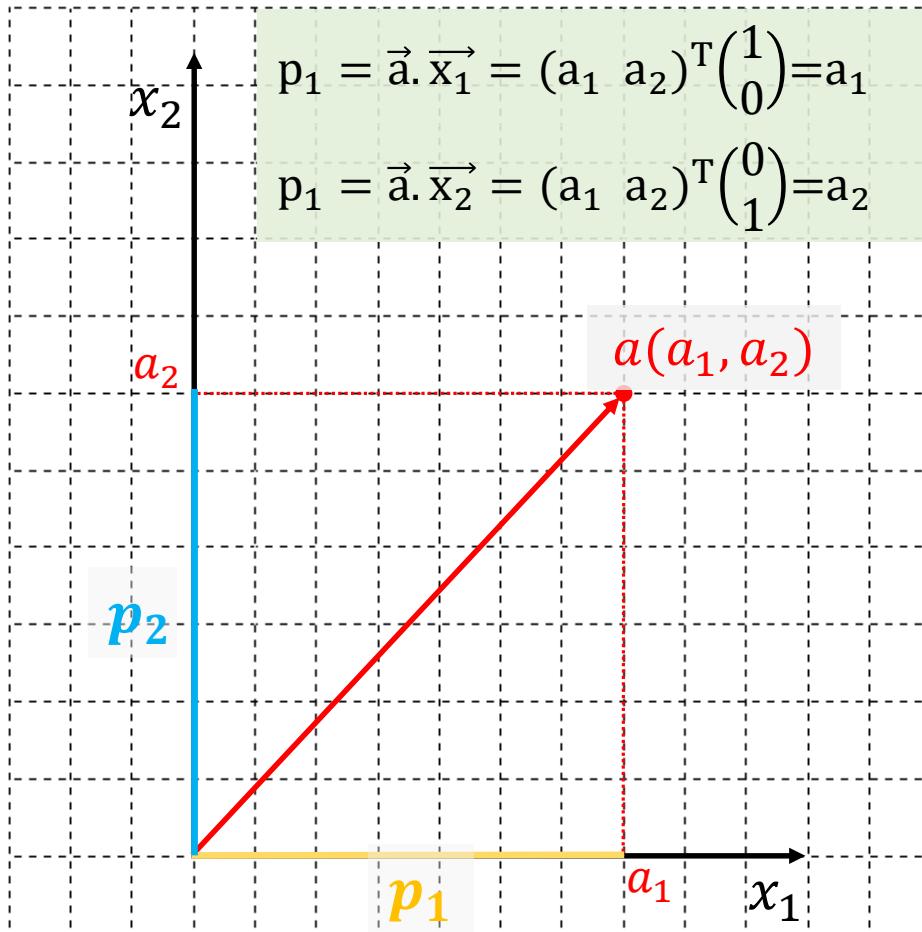
$$\vec{v} \cdot \vec{u} = v_1 \times u_1 + \dots + v_n \times u_n$$

```
1 def dot_product(vector1, vector2):
2     """
3         Compute dot product between two vectors
4         Output is a floating-point number
5     """
6
7     return sum([v1*v2 for v1, v2 in zip(vector1, vector2)])
8
9 # test case
10 vector1 = [1, 2, 3]
11 vector2 = [2, 3, 4]
12
13 output = dot_product(vector1, vector2)
14 print(output)
```

20

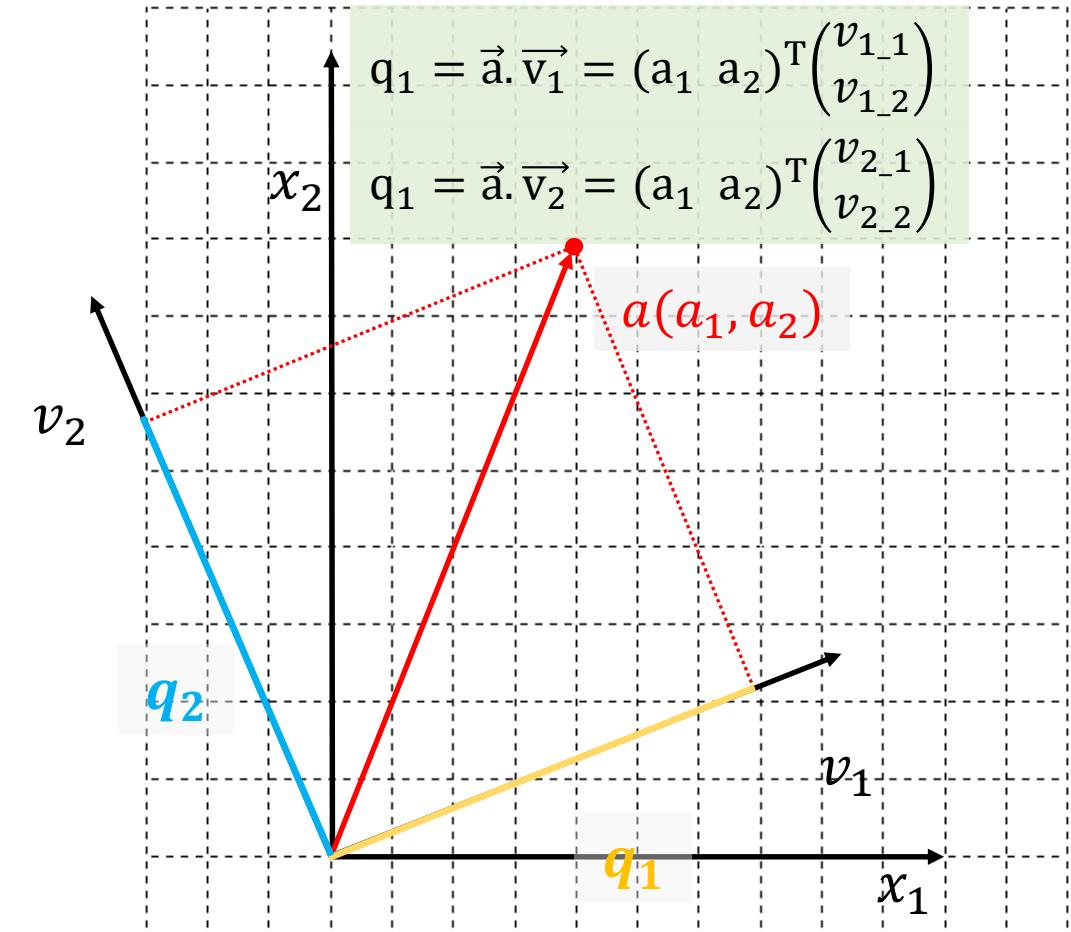
# Dot Product

$$\vec{x}_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \vec{x}_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$



Tìm độ dài hình chiếu của  $\vec{a}$  lên  $\vec{x}_1$  và  $\vec{x}_2$

$$\vec{v}_1 = \begin{pmatrix} v_{1\_1} \\ v_{1\_2} \end{pmatrix} \quad \vec{v}_2 = \begin{pmatrix} v_{2\_1} \\ v_{2\_2} \end{pmatrix}$$



Tìm độ dài hình chiếu của  $\vec{a}$  lên  $\vec{v}_1$  và  $\vec{v}_2$

# Dot Product

Mục đích: Đưa ma trận Q về dạng UΣ sao cho các vector (cột) trong U có độ dài bằng 1.

Công thức

Matrix decomposition

$$\begin{pmatrix} a & c \\ b & d \end{pmatrix} = \begin{pmatrix} \frac{a}{\sqrt{a^2 + b^2}} & \frac{c}{\sqrt{c^2 + d^2}} \\ \frac{b}{\sqrt{a^2 + b^2}} & \frac{d}{\sqrt{c^2 + d^2}} \end{pmatrix} \begin{pmatrix} \sqrt{a^2 + b^2} & 0 \\ 0 & \sqrt{c^2 + d^2} \end{pmatrix}$$

Ví dụ

$$\begin{pmatrix} 3 & 2 \\ 4 & 0 \end{pmatrix} = \begin{pmatrix} \frac{3}{\sqrt{3^2 + 4^2}} & \frac{2}{\sqrt{2^2 + 0}} \\ \frac{4}{\sqrt{3^2 + 4^2}} & 0 \end{pmatrix} \begin{pmatrix} \sqrt{3^2 + 4^2} & 0 \\ 0 & \sqrt{2^2 + 0} \end{pmatrix}$$
$$= \begin{pmatrix} \frac{3}{5} & 1 \\ \frac{4}{5} & 0 \end{pmatrix} \begin{pmatrix} 5 & 0 \\ 0 & 2 \end{pmatrix}$$

Cách tính tương tự khi Q có kích thước khác

# Dot Product

## Singular Value Decomposition

Tìm độ dài hình chiếu của  $\vec{a}$  lên  $\vec{v}_1$  và  $\vec{v}_2$

$$q_1 = \vec{a} \cdot \vec{v}_1 = (a_1 \ a_2)^T \begin{pmatrix} v_{1\_1} \\ v_{1\_2} \end{pmatrix}$$

$$q_2 = \vec{a} \cdot \vec{v}_2 = (a_1 \ a_2)^T \begin{pmatrix} v_{2\_1} \\ v_{2\_2} \end{pmatrix}$$

Giả sử  $V$  là ma trận trực giao

$$AV = Q$$

$$A = QV^{-1} = QV^T \quad \text{chuyển về } V$$

$$A = QV^{-1} = U\Sigma V^T \quad \text{tách } Q$$

Viết lại dạng ma trận cho ngắn gọn

$$\begin{aligned} \vec{a} \cdot V &= (a_1 \ a_2)^T \begin{pmatrix} v_{1\_1} & v_{2\_1} \\ v_{1\_2} & v_{2\_2} \end{pmatrix} \\ &= (q_1 \ q_2) \end{aligned}$$

Nếu có thêm  $\vec{b}$

$$\begin{pmatrix} A \\ \vec{a}_1 & \vec{a}_2 \\ b_1 & b_2 \end{pmatrix}^T \begin{pmatrix} v_{1\_1} & v_{2\_1} \\ v_{1\_2} & v_{2\_2} \end{pmatrix} = \begin{pmatrix} q_{1a} & q_{2a} \\ q_{1b} & q_{2b} \end{pmatrix}$$

Ma trận  
các điểm

Ma trận  
các trục

Ma trận độ  
dài hình chiếu

Tổng quát

$$A = U\Sigma V^T$$

$A$  là ma trận  $n \times d$ , có  $n$  điểm và mỗi điểm có  $d$  phần tử

$U$  là ma trận  $n \times n$  chứa độ dài hình chiếu, trong đó các vector cột có chiều dài bằng 1

$\Sigma$  là ma trận đường chéo ( $n \times d$ ), xác định độ dài của các vector

$V$  là ma trận  $d \times d$  chứa các trục

# Dot Product

Ứng dụng SVD cho background removal



# Vector Operations

## Hadamard product

$$\vec{v} = \begin{bmatrix} v_1 \\ \dots \\ v_n \end{bmatrix} \quad \vec{u} = \begin{bmatrix} u_1 \\ \dots \\ u_n \end{bmatrix}$$

$$\vec{v} \odot \vec{u} = \begin{bmatrix} v_1 \\ \dots \\ v_n \end{bmatrix} \odot \begin{bmatrix} u_1 \\ \dots \\ u_n \end{bmatrix} = \begin{bmatrix} v_1 \times u_1 \\ \dots \\ v_n \times u_n \end{bmatrix}$$

```
1 def Hadamard_product(vector1, vector2):  
2     '''  
3         Compute Hadamard product between two vectors  
4         Output is a vector  
5     '''  
6     return [v1*v2 for v1, v2 in zip(vector1, vector2)]  
7  
8 # test case  
9 vector1 = [1, 2]  
10 vector2 = [3, 4]  
11  
12 output = Hadamard_product(vector1, vector2)  
13 print(output)
```

[3, 8]

# Cosine similarity

**Cosine similarity (cs) được dùng để đo mức độ giống nhau/tương đồng giữa hai vector**

**Gọi  $\vec{x}$  và  $\vec{y}$  là hai vector, cs được tính như sau**

$$cs(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \|\vec{y}\|} = \frac{\sum_1^n x_i y_i}{\sqrt{\sum_1^n x_i^2} \sqrt{\sum_1^n y_i^2}}$$

**Tính chất 1:**  $cs(\vec{x}, \vec{y}) = cs(a\vec{x}, b\vec{y})$

$$\begin{aligned} cs(a\vec{x}, b\vec{y}) &= \frac{a\vec{x} \cdot b\vec{y}}{\|a\vec{x}\| \|b\vec{y}\|} = \frac{\sum_1^n a x_i b y_i}{\sqrt{\sum_1^n a^2 x_i^2} \sqrt{\sum_1^n b^2 y_i^2}} \\ &= \frac{ab \sum_1^n x_i y_i}{\sqrt{a^2 \sum_1^n x_i^2} \sqrt{b^2 \sum_1^n y_i^2}} \\ &= \frac{\sum_1^n x_i y_i}{\sqrt{\sum_1^n x_i^2} \sqrt{\sum_1^n y_i^2}} = cs(\vec{x}, \vec{y}) \end{aligned}$$

**Ví dụ:**  $\vec{x} = [4, 2, 1, 2]^T$

$$\vec{y} = [1, 2, 2, 0]^T$$

$$\vec{u} = 2\vec{x} = [8, 4, 2, 4]^T$$

$$\vec{v} = 3\vec{y} = [3, 6, 6, 0]^T$$

$$\begin{aligned} cs(\vec{x}, \vec{y}) &= \frac{4*1+2*2+1*2+2*0}{\sqrt{4^2+2^2+1^2+2^2} \sqrt{1^2+2^2+2^2+0}} \\ &= \frac{10}{\sqrt{25}\sqrt{9}} = \frac{10}{15} = 0.67 \end{aligned}$$

$$\begin{aligned} cs(\vec{u}, \vec{v}) &= \frac{8*3+4*6+2*6+4*0}{\sqrt{8^2+4^2+2^2+4^2} \sqrt{3^2+6^2+6^2+0}} \\ &= \frac{60}{\sqrt{100}\sqrt{81}} = \frac{60}{90} = 0.67 \\ &= cs(\vec{x}, \vec{y}) \end{aligned}$$

# Cosine similarity

## ❖ Code

$$CS(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \|\vec{y}\|} = \frac{\sum_1^n x_i y_i}{\sqrt{\sum_1^n x_i^2} \sqrt{\sum_1^n y_i^2}}$$

```
1 import math
2
3 def cosine_similarity(vector1, vector2):
4     """
5         Compute dot product between two vectors
6         Output is a floating-point number
7     """
8
9     sumxy = sum([v1*v2 for v1, v2 in zip(vector1, vector2)])
10    sumxx = sum([v1*v2 for v1, v2 in zip(vector1, vector1)])
11    sumyy = sum([v1*v2 for v1, v2 in zip(vector2, vector2)])
12
13    return sumxy/math.sqrt(sumxx*sumyy)
14
15 # test case
16 vector1 = [5, 3, 2, 7]
17 vector2 = [2, 9, 4, 1]
18
19 output = cosine_similarity(vector1, vector2)
20 print(output)
```

0.552005787925351

# Stereo Matching

## Stereo Matching / Binocular Depth Estimation



Hình từ camera trái



Hình từ camera phải



Stereo camera

Thuật toán  
stereo matching



Hình chiều sâu  
cho hình trái

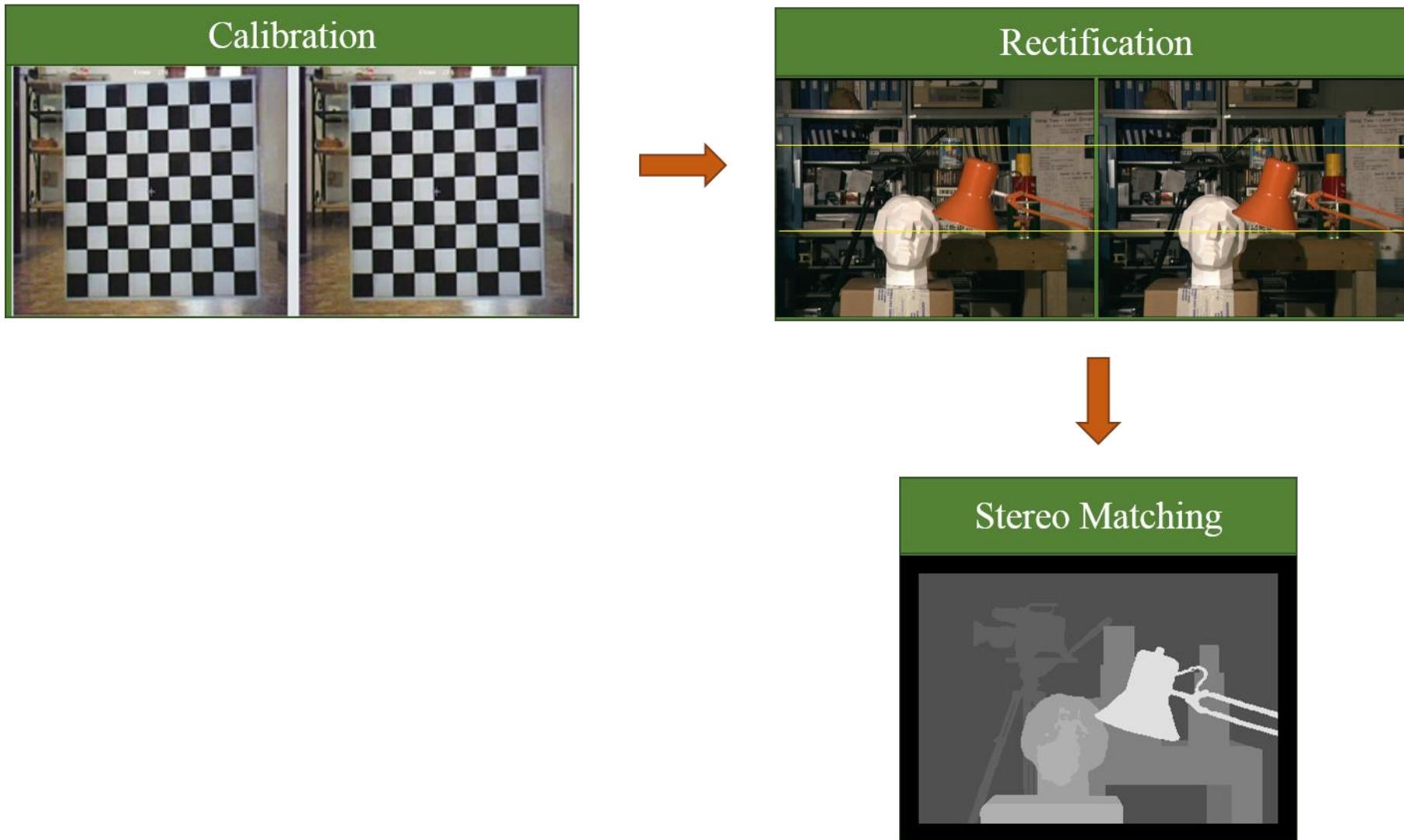


Hình chiều sâu  
cho hình phải

### Ứng dụng

- Đo khoảng cách các tới các object trong hình
- Xây dựng mô hình 3D
- Thông tin cho các ứng dụng khác: detection, tracking, ...

# Stereo Matching



# Stereo Matching

- ❖ Left and right images of a stereo pair



Left Image

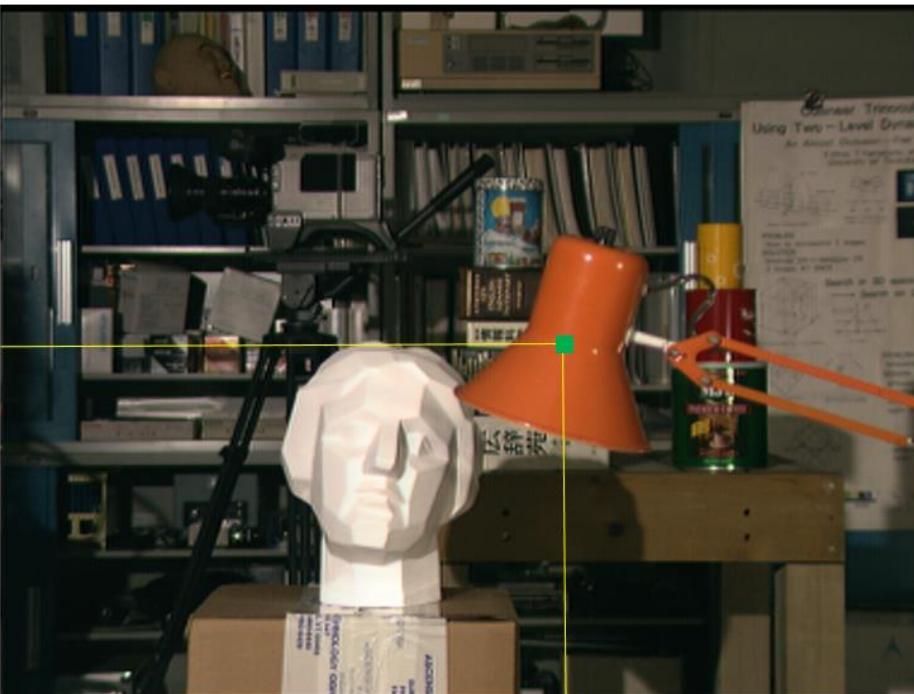


Right Image

# Stereo Matching

$p = (234, 140)$

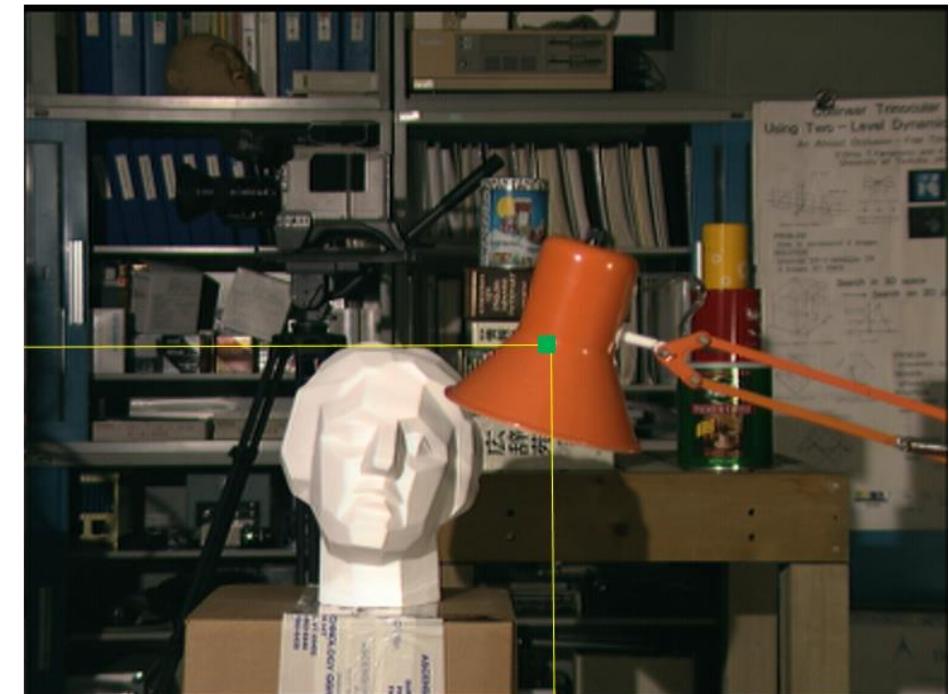
$y_p = 140$



$x_p = 234$

$q = (220, 140)$

$y_q = 140$

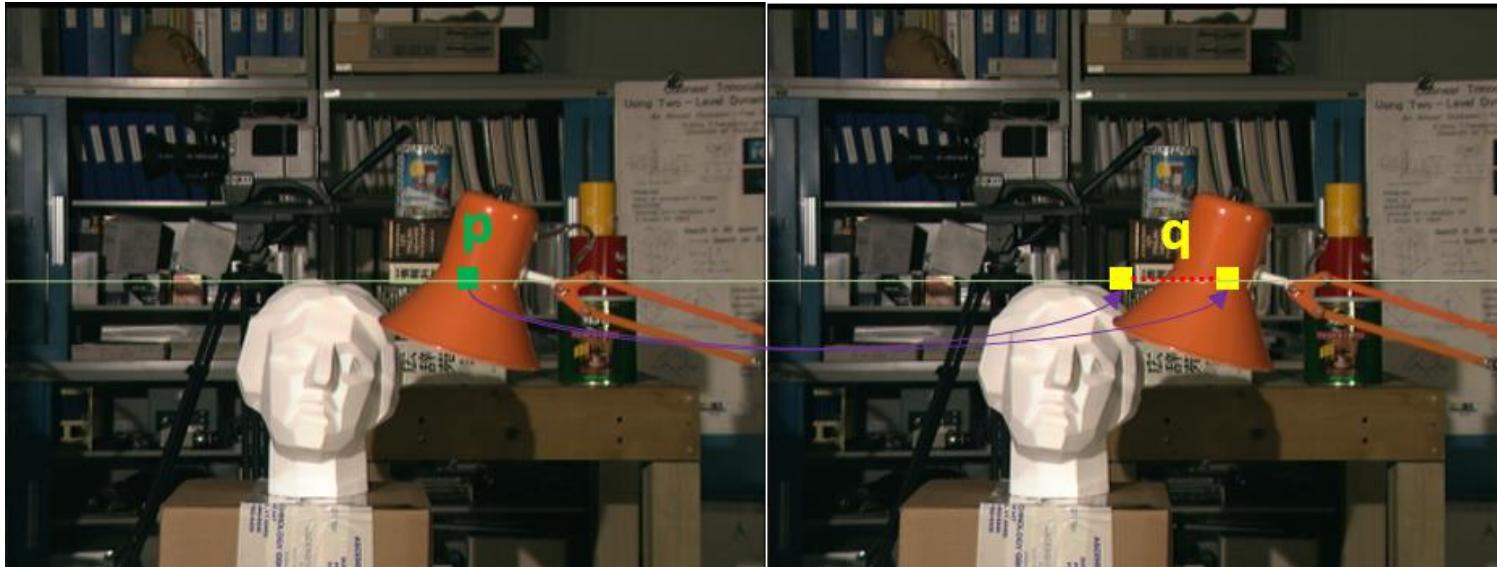


$x_q = 220$

$$\text{disparity}_q = x_p - x_q = 14$$

# Stereo Matching

## ❖ Method 1



Left Image

Right Image

$$d_p = \arg \min_{d \in D} (C(p, q))$$

where  $C(p, q) = (L(p) - R(q))^2$

and  $q = (x_p - d, y_p)$

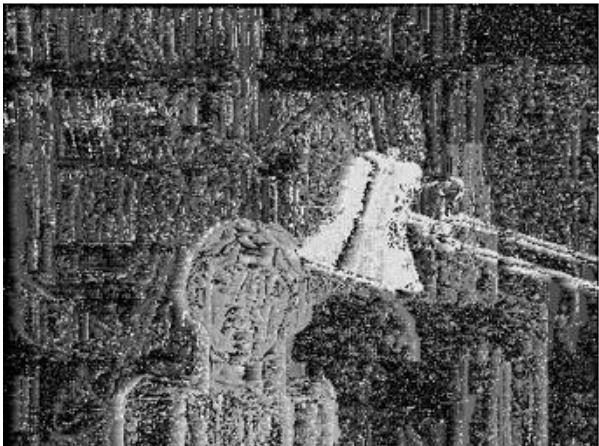
# Stereo Matching



Left Image



Right Image



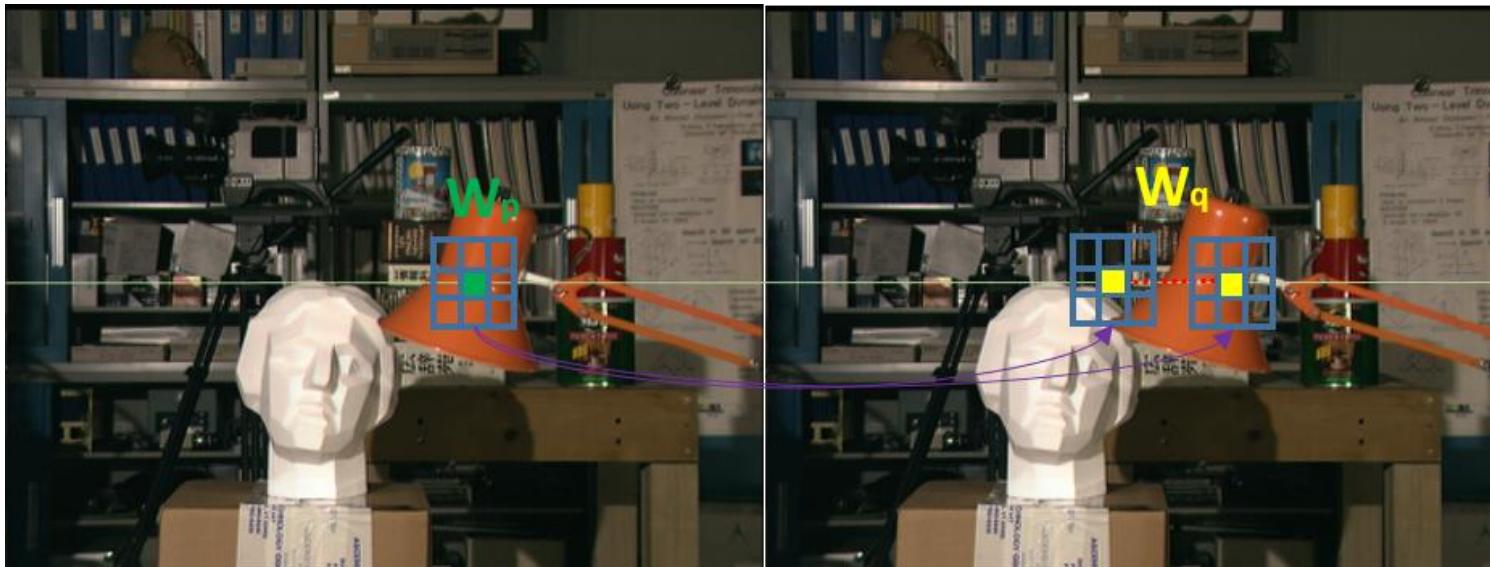
Disparity Map



Ground Truth

# Stereo Matching

## ❖ Method 2



Left Image

Right Image

$$d_p = \arg \min_{d \in D} (C(p, q))$$

$$\text{where } C(p, q) = \sum_{(u, v) \in (W_p, W_q)} (L(u) - R(v))^2$$

$$\text{and } q = (x_p - d, y_p)$$

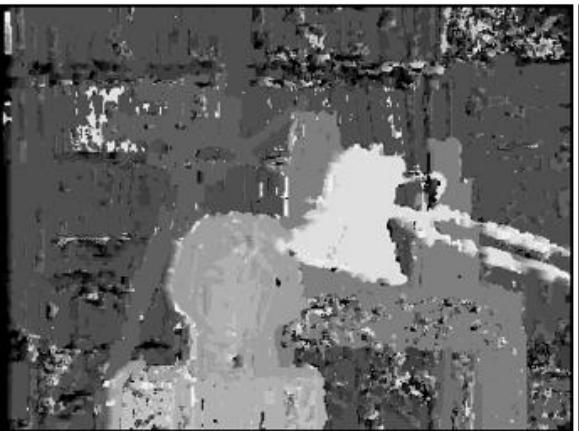
# Stereo Matching



Left Image



Right Image



Disparity Map



Ground Truth

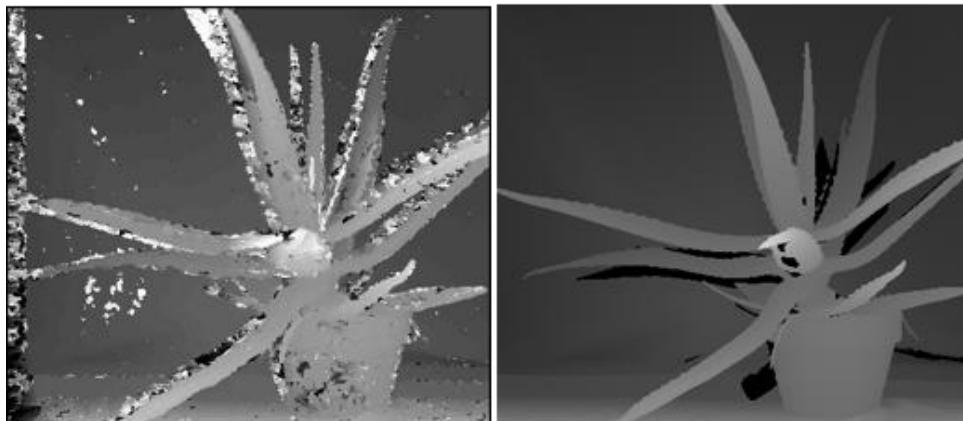
# Stereo Matching

## ❖ Aloe stereo pair



Left Image

Right Image



Disparity Map

Ground Truth

# Stereo Matching

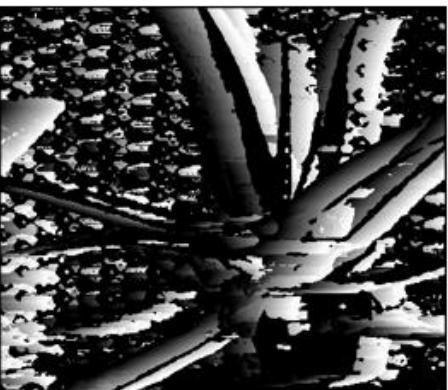
## ❖ Aloe stereo pair



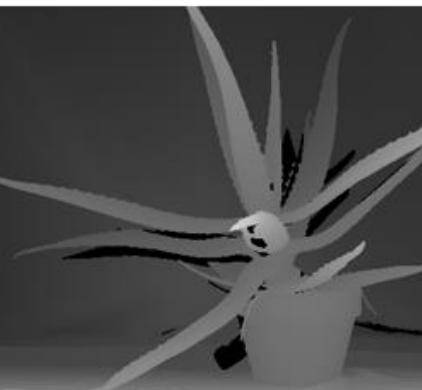
Left Image



Right Image



Disparity Map



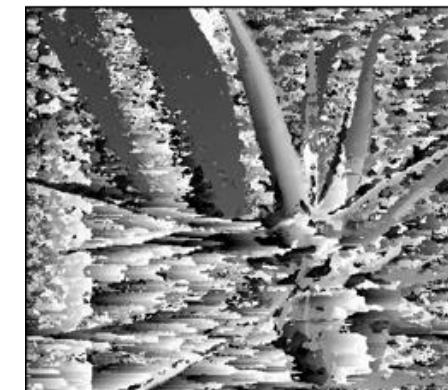
Ground Truth



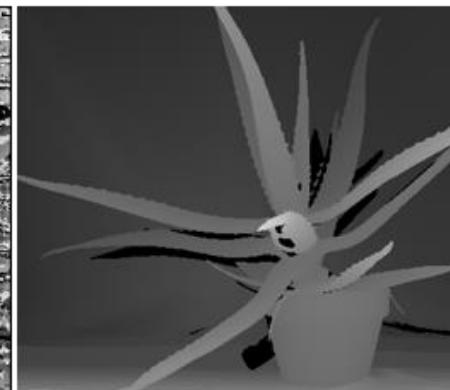
Left Image



Right Image



Disparity Map

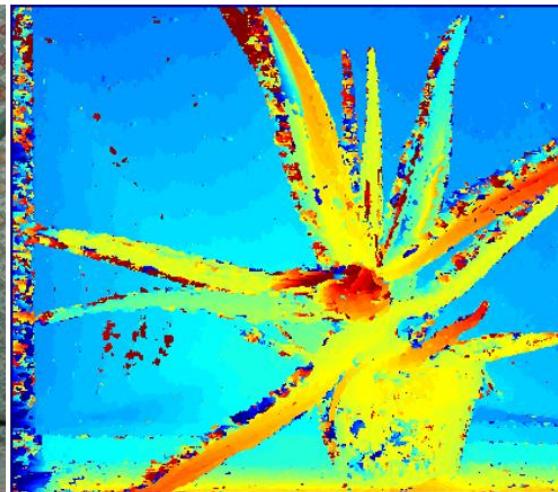


Ground Truth

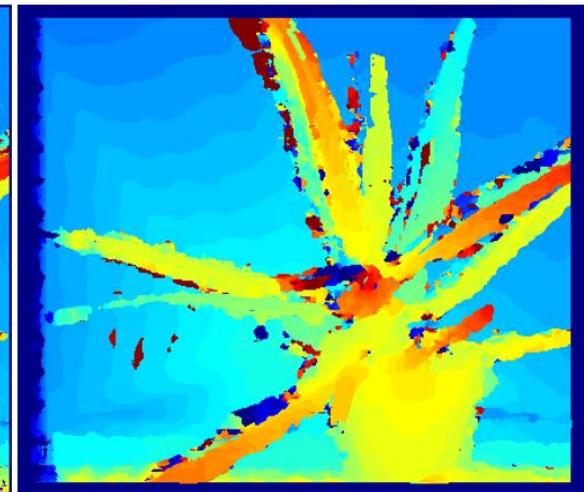
## Ứng dụng tìm chiều sâu (depth estimation)



Ảnh stereo có cùng độ sáng



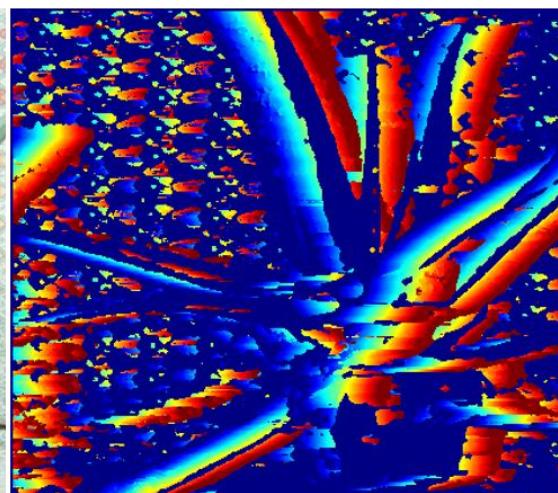
Absolute difference



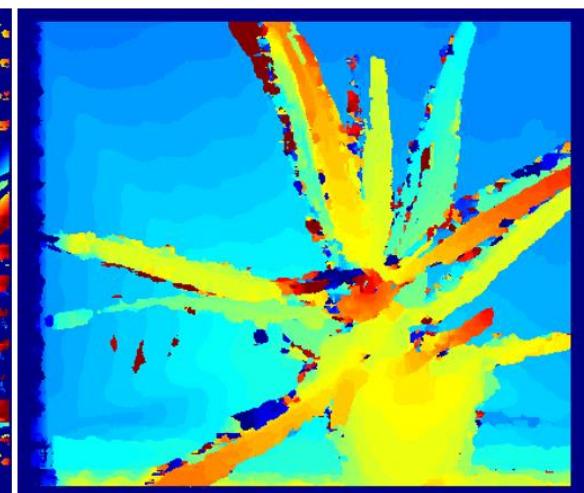
Cosine Similarity



Ảnh stereo khác độ sáng



Absolute difference



Cosine Similarity

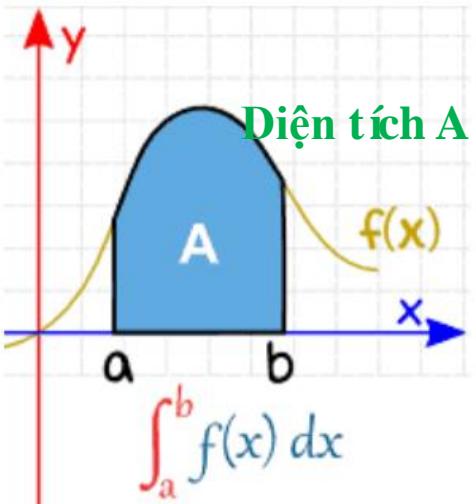
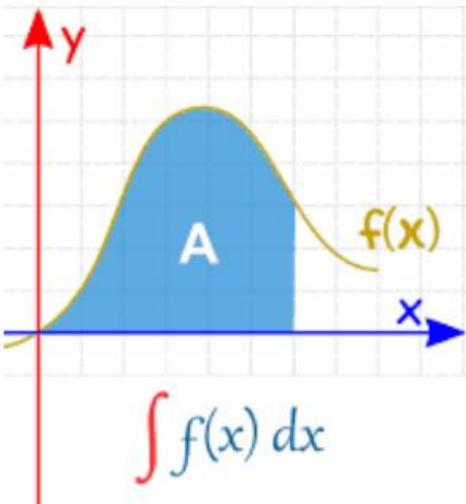
Cosine similarity hoạt động ổn định khi ảnh stereo thay đổi độ sáng

# Outline

- Derivative/Gradient
- Interpolation
- Vector and Matrix
- Cosine Similarity
- Integral

# Integral

## Công thức



<https://www.mathsisfun.com/calculus/integration-introduction.html>

$$F(a) = \int_{-\infty}^a f(a) dx$$

Diện tích A

$$F(b) = \int_{-\infty}^b f(b) dx$$

$$A = F(b) - F(a)$$

## Áp dụng cho hàm rời rạc (1D)

$f(x)$	1	8	5	7	3	5	8	3	2	9
$x$	0	1	2	3	4	5	6	7	8	9

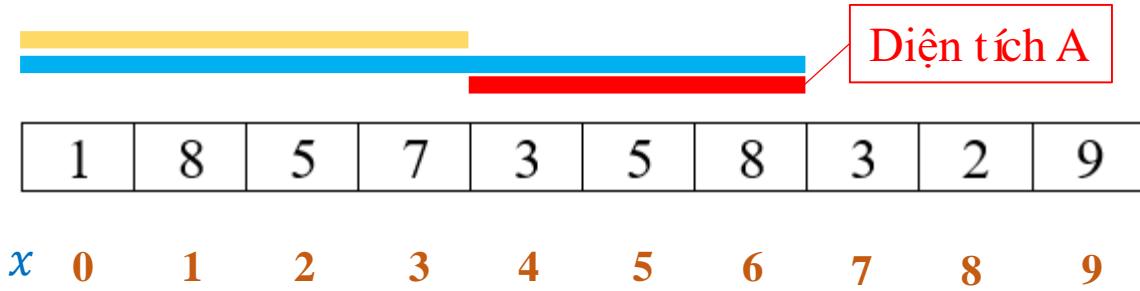
$$\begin{aligned} F(3) &= \sum_{x \leq 3} f(x) = f(0) + f(1) + f(2) + f(3) \\ &= 1 + 8 + 5 + 7 = 21 \end{aligned}$$

$$F(6) = \sum_{x \leq 6} f(x) = 1 + 8 + 5 + 7 + 3 + 5 + 8 = 37$$

$$A = F(6) - F(3) = \sum_{4 \leq x \leq 6} f(x) = 3 + 5 + 8 = 16$$

# Integral

## Áp dụng cho hàm rời rạc (1D)



$$F(3) = \sum_{x \leq 3} f(x) = f(0) + f(1) + f(2) + f(3) \\ = 1 + 8 + 5 + 7 = 21$$

$$F(6) = \sum_{x \leq 6} f(x) = 1 + 8 + 5 + 7 + 3 + 5 + 8 = 37$$

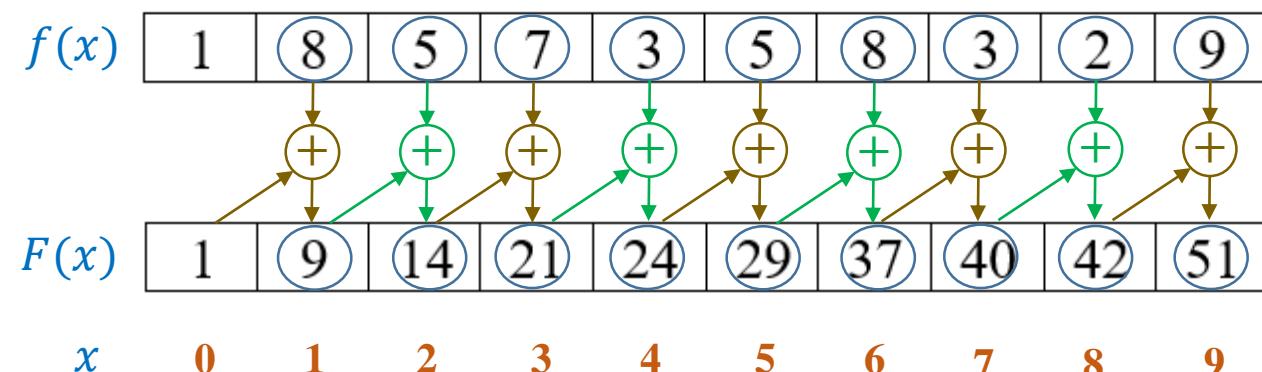
$$A = F(6) - F(3) = \sum_{4 \leq x \leq 6} f(x) = 3 + 5 + 8 = 16$$

## Tính chất

$$F(x) = f(x) + F(x - 1)$$

$$F(7) = f(7) + F(6) = 3 + 37 = 40$$

Xây dựng integral array dùng tính chất  $F(x) = f(x) + F(x - 1)$



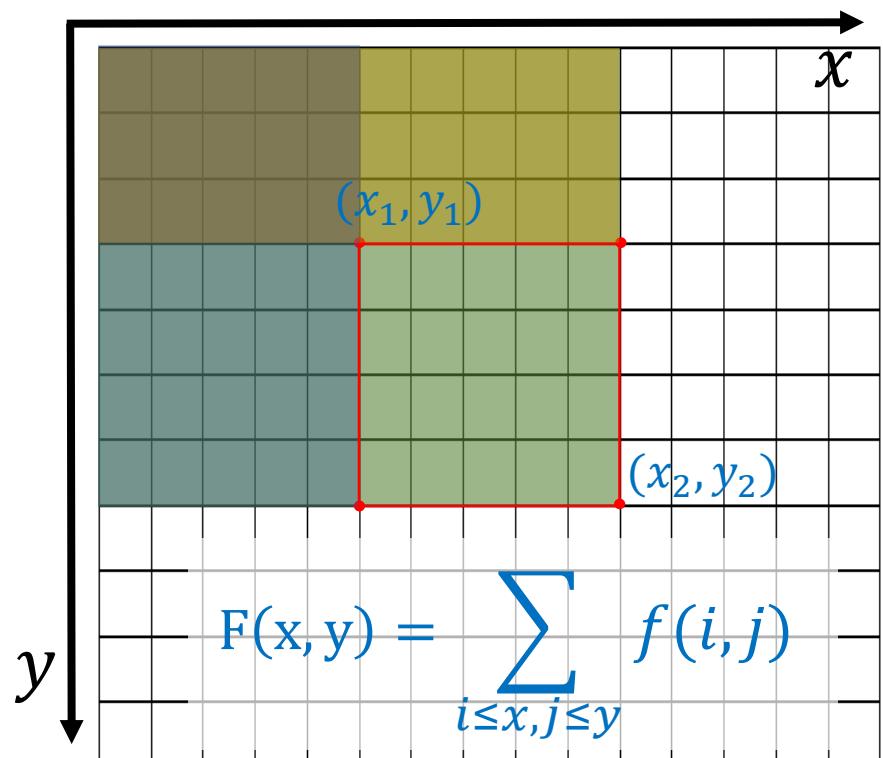
Tính tổng với độ phức tạp ~ O(1)

$$\sum_{a \leq x \leq b} f(x) = F(b) - F(a - 1)$$

$$\sum_{4 \leq x \leq 6} f(x) = F(6) - F(3) = 37 - 21 = 16$$

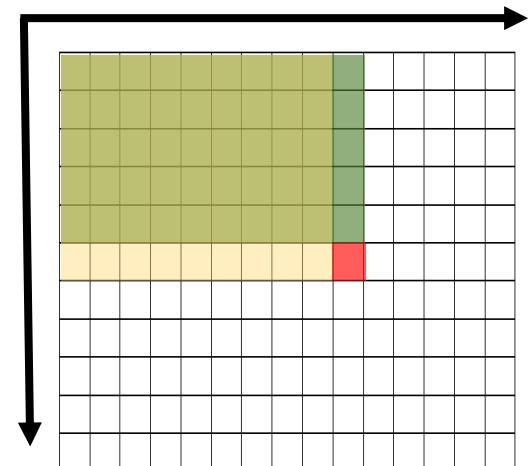
# Integral

Áp dụng cho hàm rời rạc (2D)



Xây dựng integral array với tính chất

$$\begin{aligned} F(x,y) &= f(x,y) + \\ &F(x-1,y) + \\ &F(x,y-1) - \\ &F(x-1,y-1) \end{aligned}$$



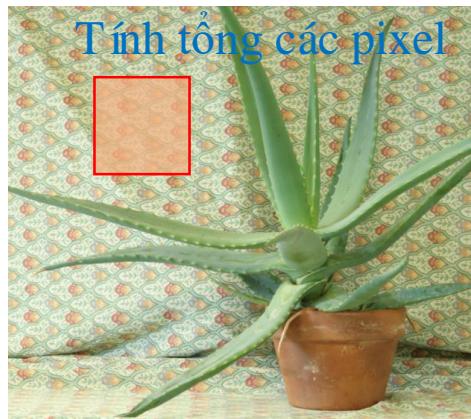
Diện tích A

$$A = \sum_{\substack{x_1 \leq i \leq x_2 \\ y_1 \leq j \leq y_2}} f(i,j) = F(x_2, y_2) - F(x_2 - 1, y_2) - F(x_2, y_2 - 1) + F(x_2 - 1, y_2 - 1)$$

# Ứng dụng cho computer vision

Tính tổng

$$cs(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \|\vec{y}\|} = \frac{\sum_1^n x_i y_i}{\sqrt{\sum_1^n x_i^2} \sqrt{\sum_1^n y_i^2}}$$



Cách tính cosine similarity  
thông thường

Time = ~36 seconds

Dùng integral image

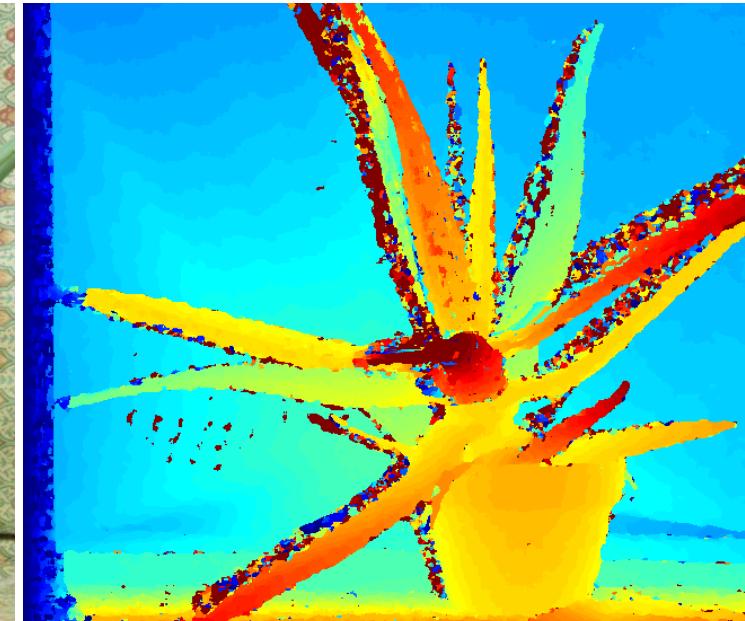
Time < 1 second



Ảnh trái



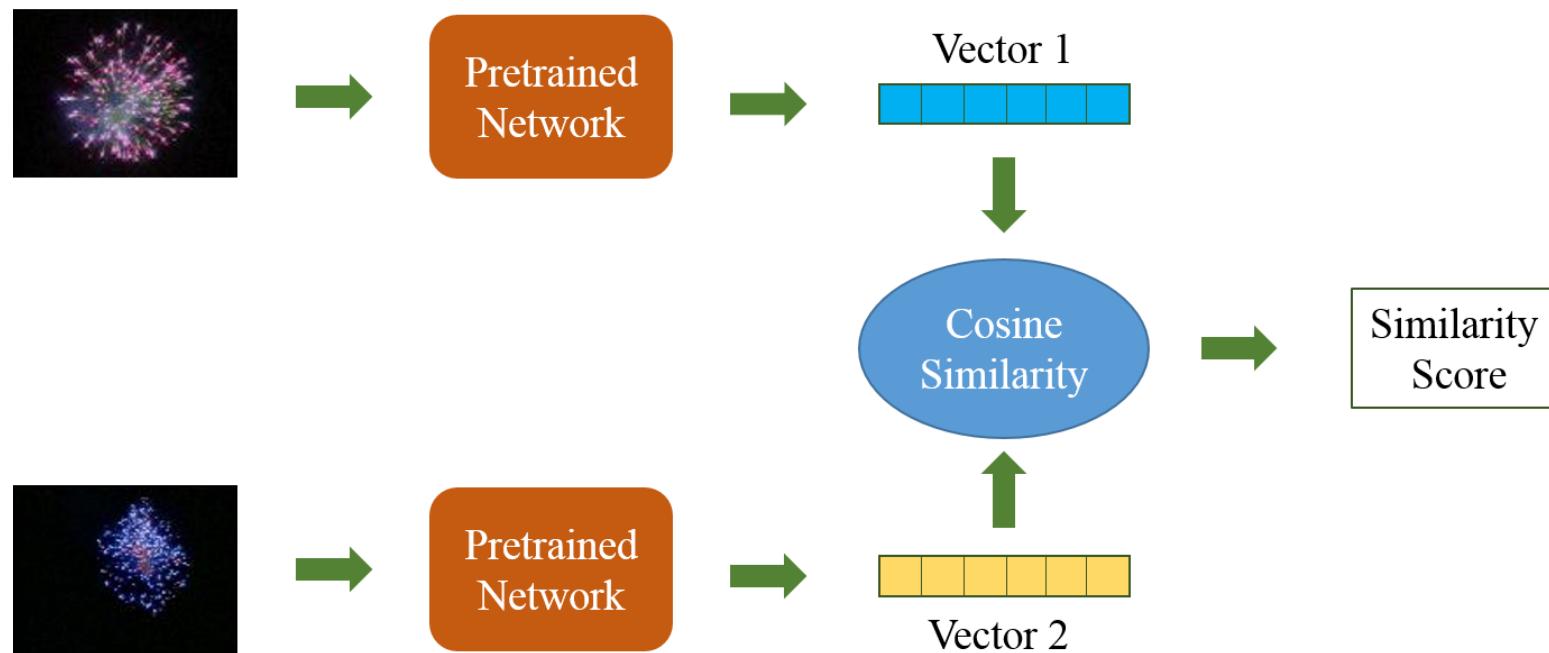
Ảnh phải



Ảnh chiều sâu cho ảnh trái  
dùng cosine similarity

# Discussion

Ứng dụng Cosine Similarity để tính mức độ giống nhau giữa hai hình



## Image Retrieval

Query 2



Query 1

= 2 \*



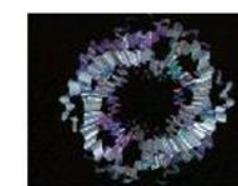
Query 2



Query 1

Image  
Retrieval

Image  
Database



Result

Hai hình query cho cùng kết quả như nhau

**Query 1**



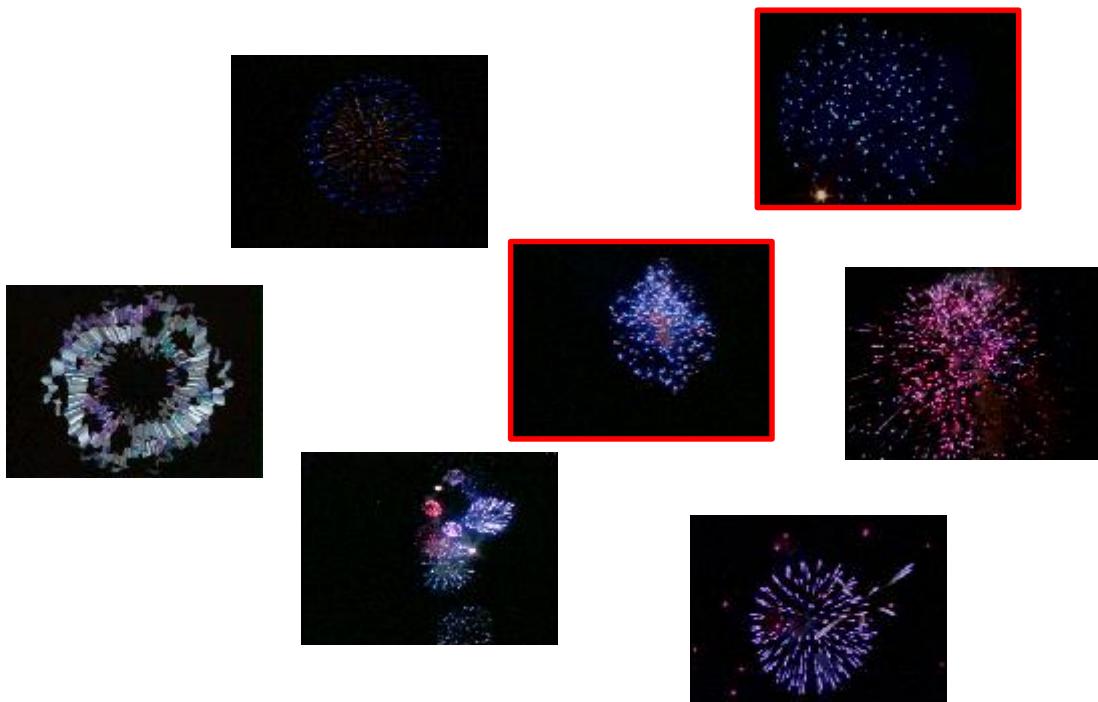
**Query 2**



**Query 1**

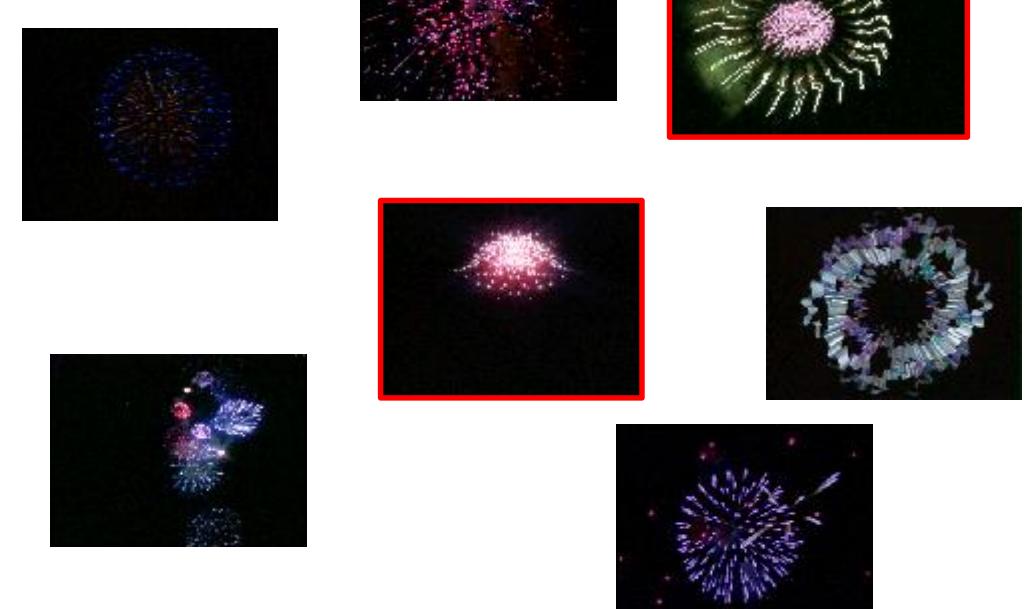


= 50 +



**Result 1**

**Result 1 ≠ Result 2**



**Result 2**

# Summary

- Derivative/Gradient
- Interpolation
- Vector and Matrix
- Cosine Similarity
- Integral

Thank you!