

# 影像處理介紹 (使用 Python)

# Python in image processing

影像處理它是計算機視覺的核心部分，在機器人、自動駕駛汽車和物體檢測等許多現實世界的例子中起著至關重要的作用。

影像處理使我們能夠一次轉換和操作數千張圖像，並從中提取有用的特徵。它在幾乎所有領域都有廣泛的應用。

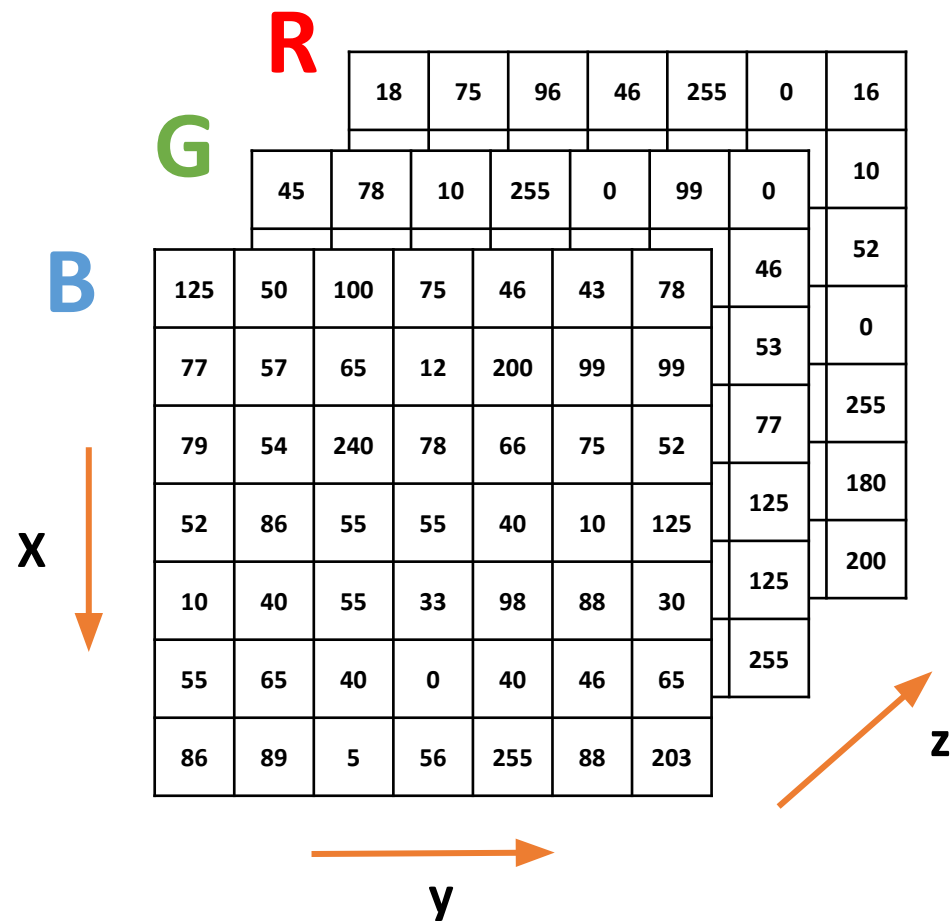
Python在影像處理這一方面有許多的套件可以使用可以非常有效的完成影像處理的任務

我們將在此介紹一些python的基礎和在影像處理中常使用的一些套件

# 圖片的輸入



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



## 圖片的輸出



```
image = image[:, :, ::-1]

# show the image
plt.imshow(image)
plt.show()
```



R

G

B

				125	50	100	75	46	43	78
		45	78	10	255	0	99	0		99
	18	75	96	46	255	0	16	46		52
	86	57	255	20	0	0	10	53		88
	30	203	240	57	66	100	52	77		10
	45	52	89	66	40	10	200	125		65
	33	40	11	42	98	88	255	125		203
	98	65	40	5	40	46	180	255		
	0	255	5	255	255	40	200			

# Pixel 的運算 (Point Processing)

## Point processing

original



$$x$$

darken



$$x - 128$$

lower contrast



$$\frac{x}{2}$$

non-linear lower contrast



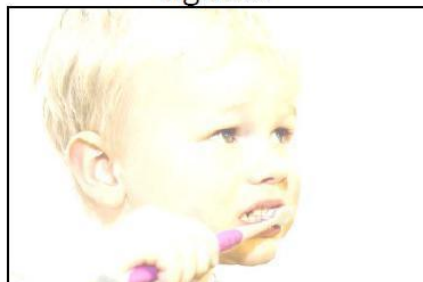
$$\left(\frac{x}{255}\right)^{1/3} \times 255$$

invert



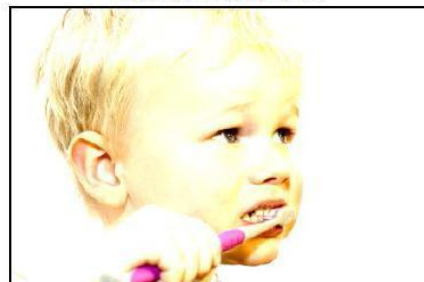
$$255 - x$$

lighten



$$x + 128$$

raise contrast



$$x \times 2$$

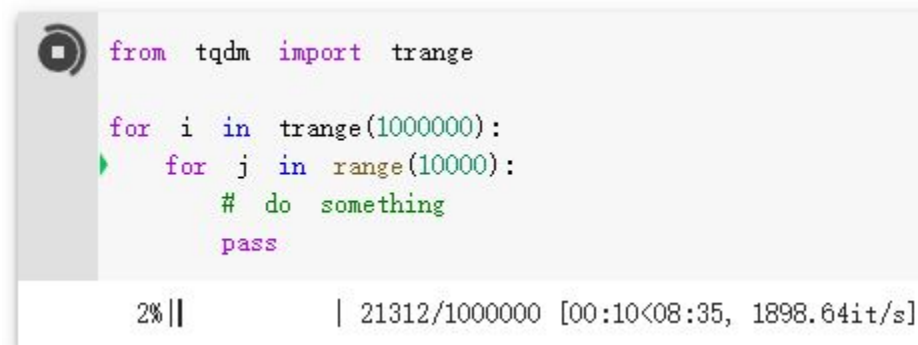
non-linear raise contrast



$$\left(\frac{x}{255}\right)^2 \times 255$$

# Pixel 的運算 (Point Processing)

- Plot 的時候注意矩陣是浮點數 (float) 還是整數 (int) 型態
  - 可以使用 `image.dtype` 查看
  - 可以使用 `image.astype('uint 8')` 指定型態
- Plot 的時候注意是灰階還是彩色, 以及通道順序, 需不需要 `cmap` 參數
- `np. clip()` 可以截斷超出範圍的數值
- `from tqdm import trange`



```
from tqdm import trange

for i in trange(1000000):
    for j in range(10000):
        # do something
        pass
```

2%|| | 21312/1000000 [00:10<08:35, 1898.64it/s]

# Pixel 的運算 (Point Processing)

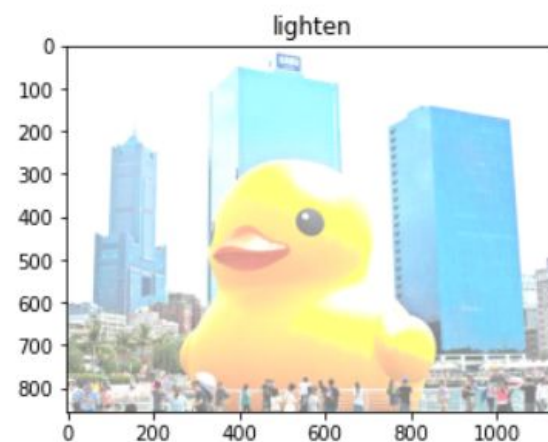
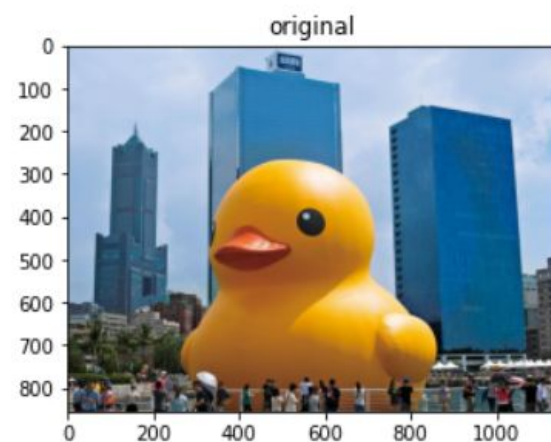
```
▶ image_int = image.astype('int32')
  result_img = np.clip(image_int + 128, 0, 255)

plt.figure(figsize=(10, 20))

plt.subplot(1, 2, 1)
plt.imshow(image_int)
plt.title('original')

plt.subplot(1, 2, 2)
plt.imshow(result_img)
plt.title('lighten')

plt.show()
```





# Pixel 的運算 (Point Processing)



```
# invert  
invert_img = img.copy()  
invert_img = 255 - invert_img
```



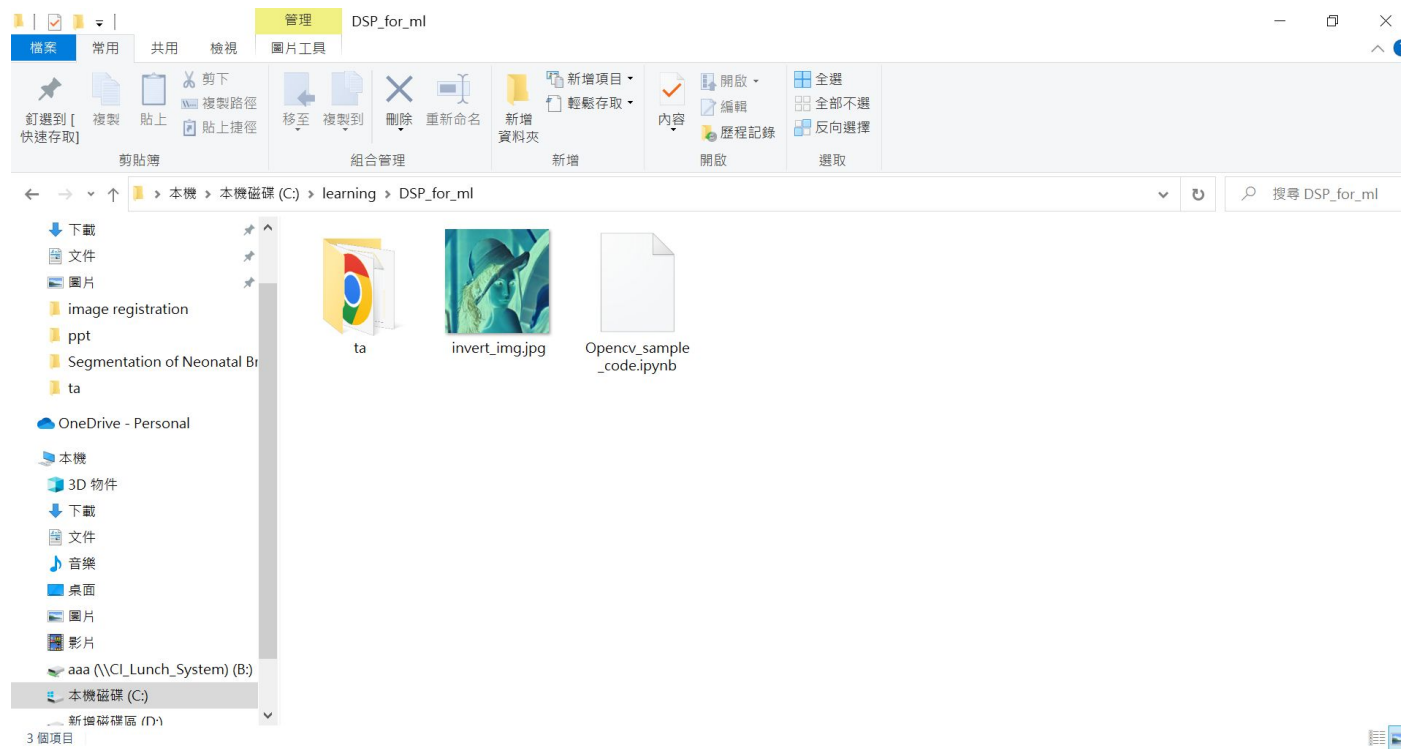
```
# 顯示圖片  
cv2.imshow('My Image', invert_img)  
  
# 按下任意鍵則關閉所有視窗  
cv2.waitKey(0)  
cv2.destroyAllWindows()
```



# 儲存圖片



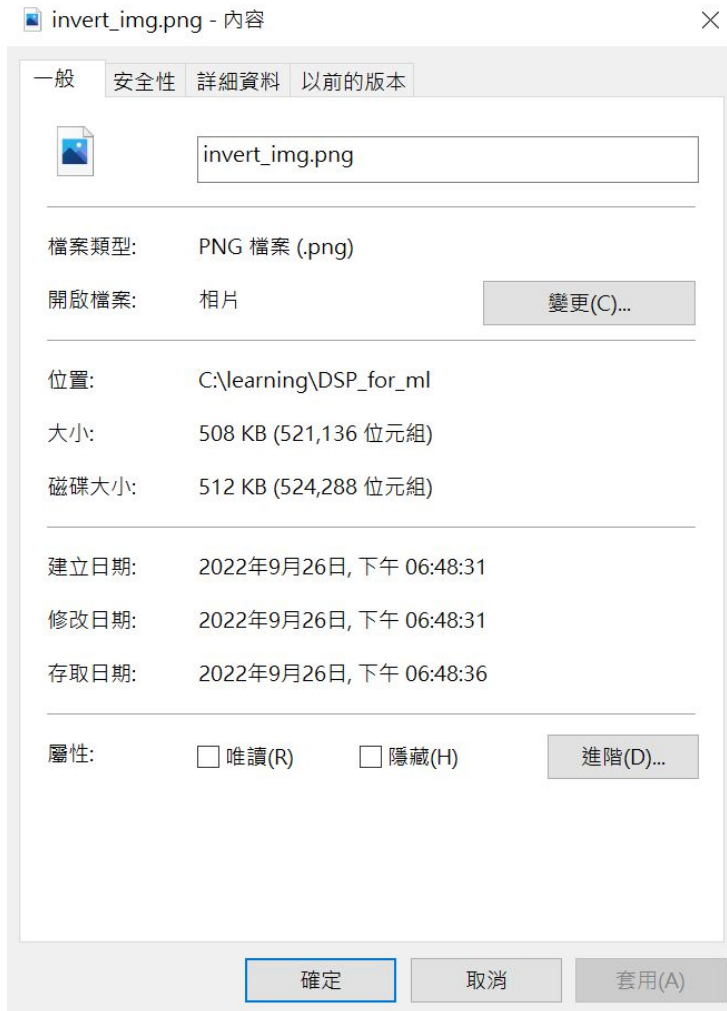
會出現在當前檔案夾生成invert\_img.jpg



# 寫入圖檔

```
cv2.imwrite('invert_img.jpg', invert_img)
```

# JPG VS PNG



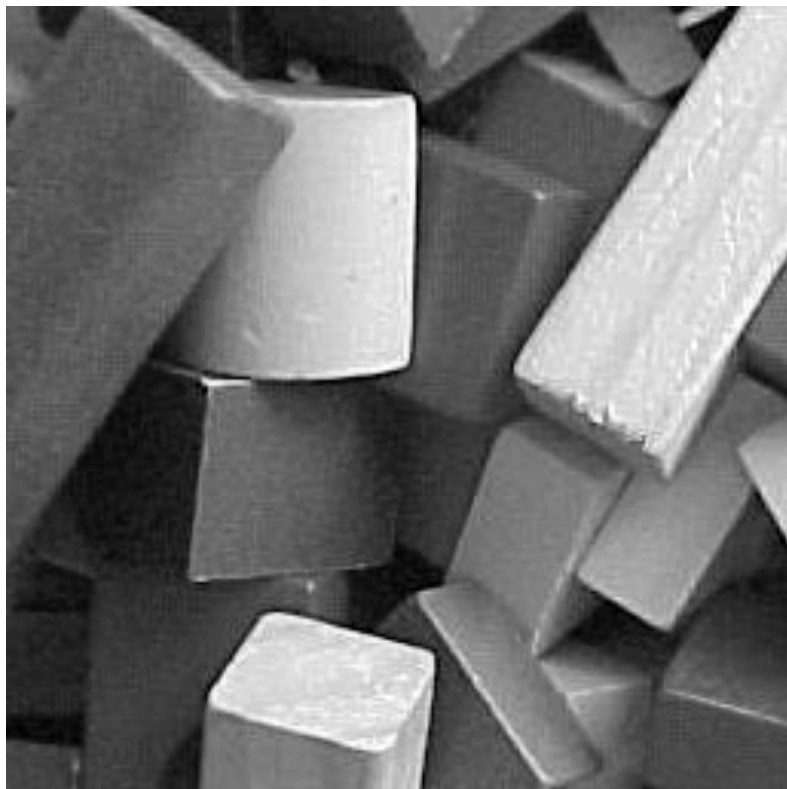
可注意到大小不一樣  
因JPG是有損壓縮,  
因此JPG大小比PNG小

# JPG VS PNG

```
# jpg vs png  
jpg_img = cv2.imread('invert_img.jpg')  
png_img = cv2.imread('invert_img.png')  
  
print("invert_img 與 jpg_img 是否一樣:", np.equal(invert_img, jpg_img).all())  
print("invert_img 與 png_img 是否一樣:", np.equal(invert_img, png_img).all())
```

```
invert_img 與 jpg_img 是否一樣: False  
invert_img 與 png_img 是否一樣: True
```

# Canny Edge Detection



Grayscale or RGB



Binary

# Canny Edge Detection

1. 前處理: 轉灰階, 降噪 (Gaussian filter)
2. 計算每個 pixel 的梯度大小與方向
3. 非極大值抑制: 找出最可能的邊緣
4. 閾值處理: 分出 Strong Edge 和 Weak Edge
5. 連接邊緣: 從 Strong Edge 和 Weak Edge 中確定邊緣

# Canny Edge Detection

計算每個 pixel 的梯度大小與方向

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

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

$$\frac{\partial f}{\partial x} = S_x \otimes f$$

$$\frac{\partial f}{\partial y} = S_y \otimes f$$

$$\nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]$$

gradient

$$\theta = \tan^{-1} \left( \frac{\partial f / \partial y}{\partial f / \partial x} \right)$$

direction

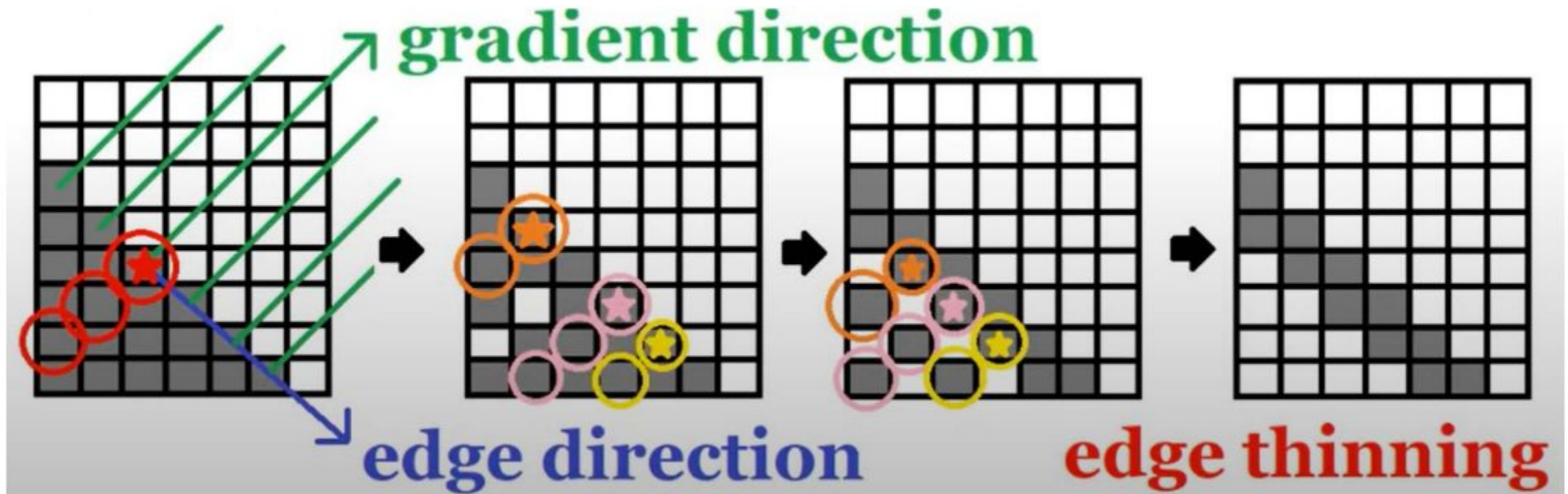
$$\|\nabla f\| = \sqrt{\left( \frac{\partial f}{\partial x} \right)^2 + \left( \frac{\partial f}{\partial y} \right)^2}$$

amplitude



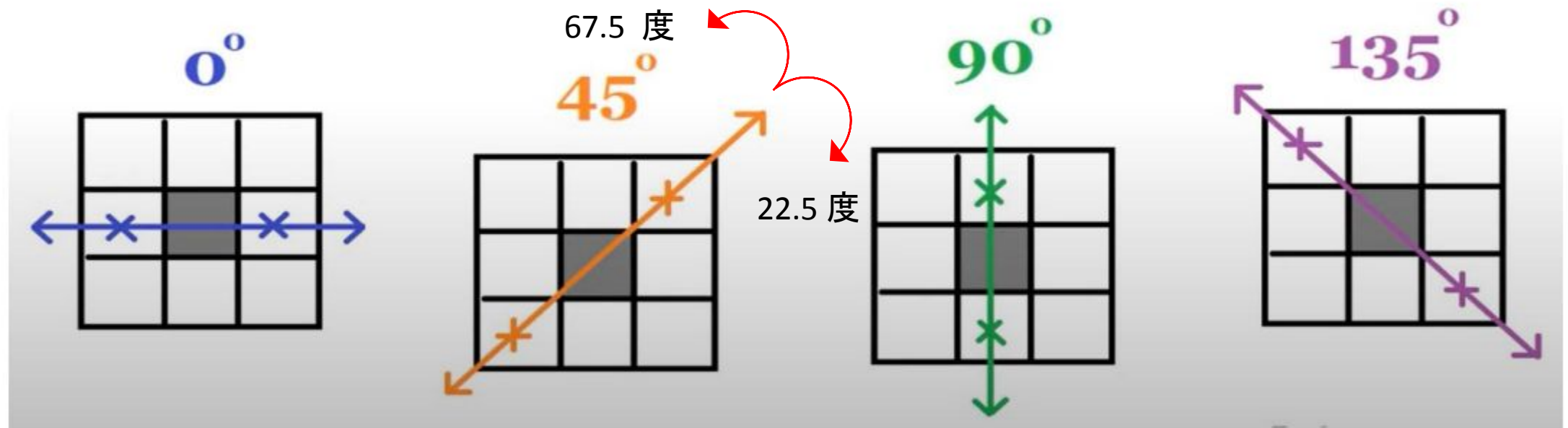
# Canny Edge Detection

非極大值抑制



# Canny Edge Detection

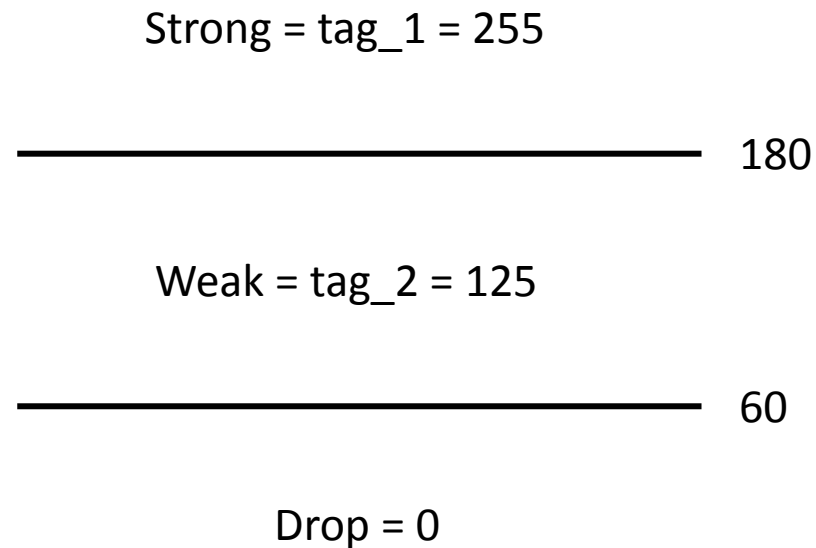
## 非極大值抑制



與沿方向前後兩個點比較，若非最大則消失

# Canny Edge Detection

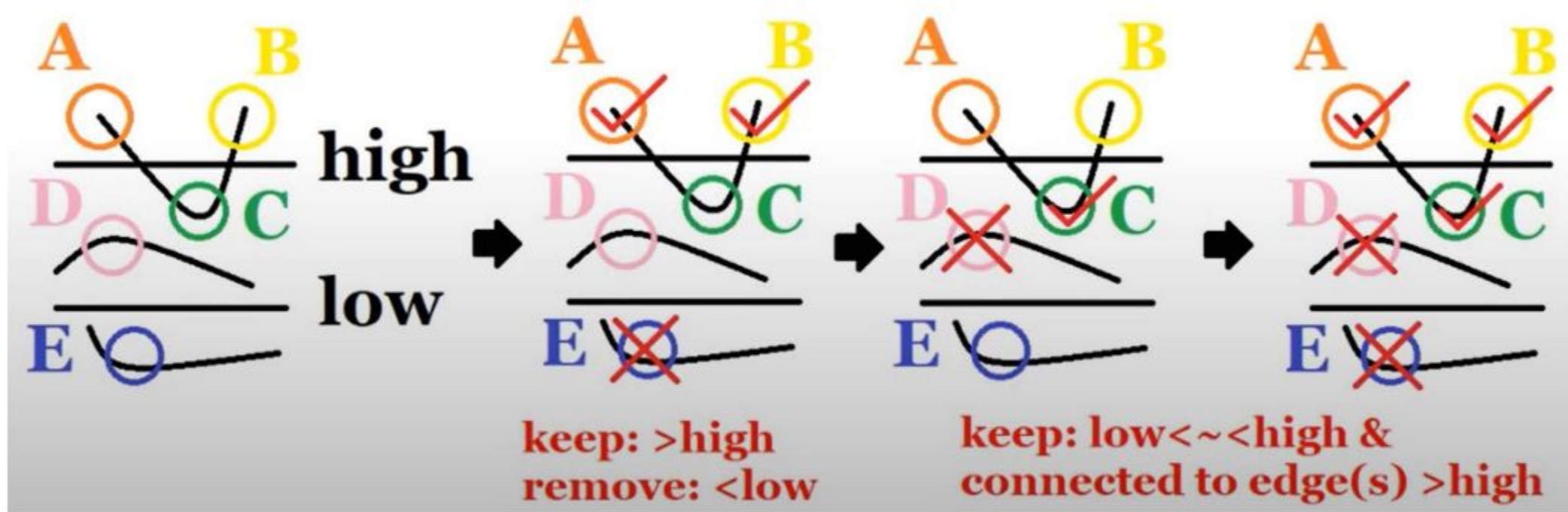
## 閾值處理 & 連接邊緣



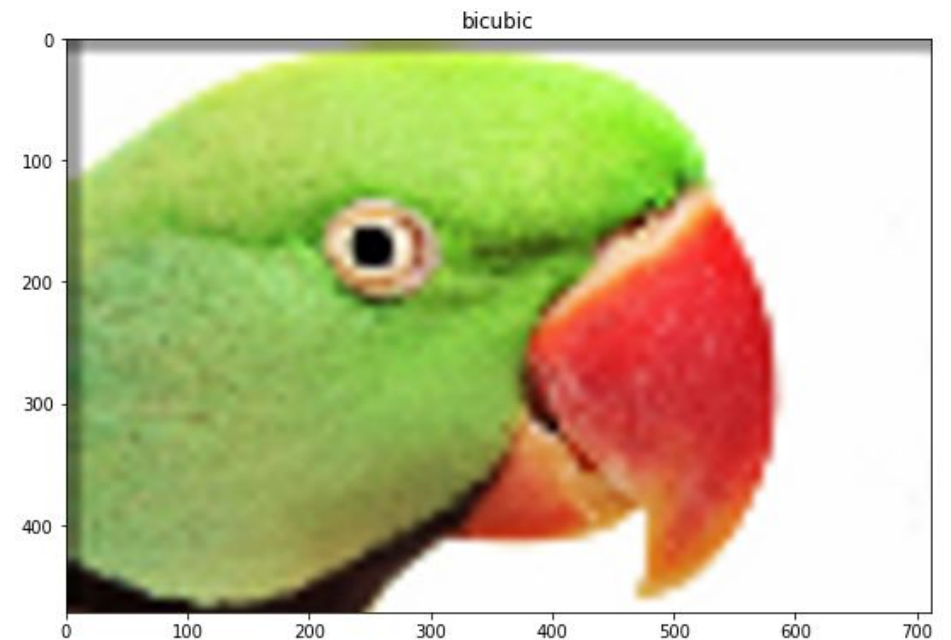
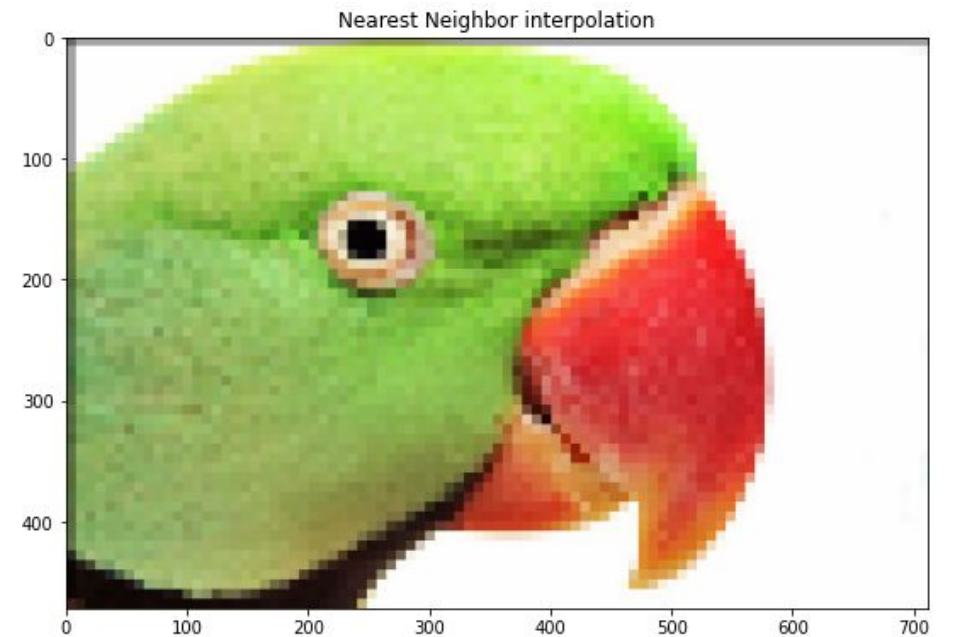
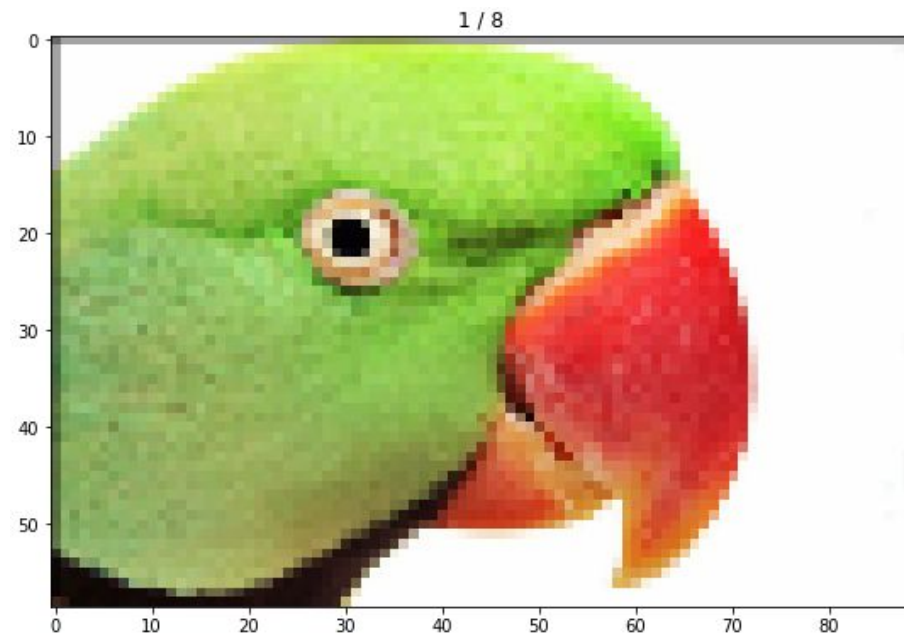
- 從強邊緣出發，看能不能沿著弱邊緣走到另一個強邊緣
- 如果可以，經過的弱邊緣都變成強邊緣
- 從弱邊緣變強邊緣的點也要再做一次上述步驟
- 最後剩下的弱邊緣全部丟掉

# Canny Edge Detection

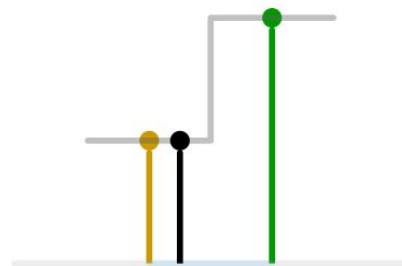
閾值處理 & 連接邊緣



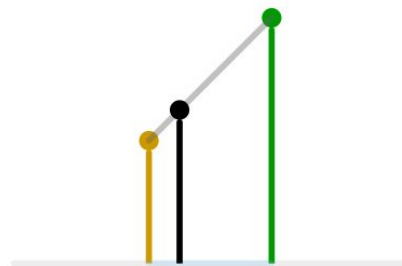
# Bicubic Interpolation



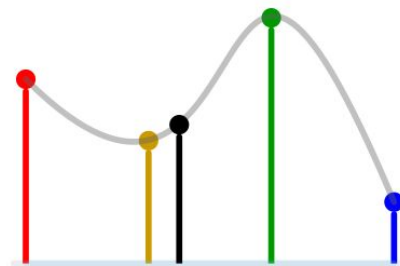
# Bicubic Interpolation



1D nearest-neighbour



Linear



Cubic

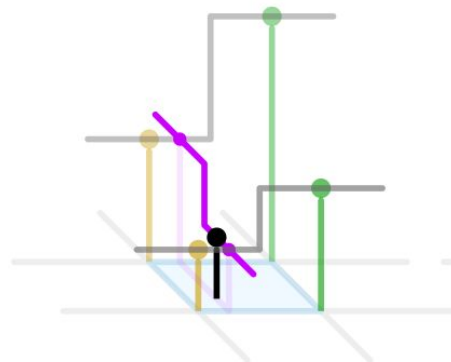
$$f(x) = ax^3 + bx^2 + cx + d$$

$$g(y) = ay^3 + by^2 + cy + d$$

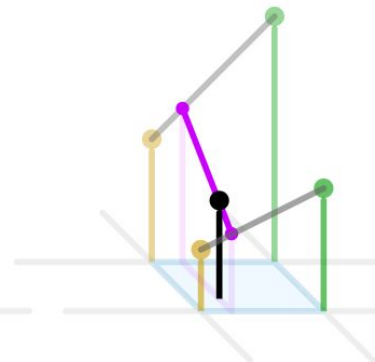
←

$$p(x, y) = \sum_{i=0}^3 \sum_{j=0}^3 a_{ij} x^i y^j.$$

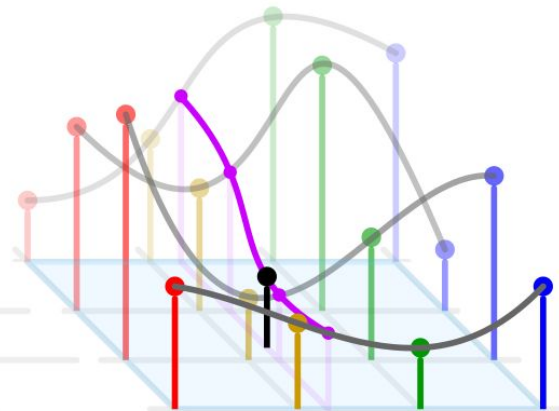
16 coefficients  $a_{ij}$



2D nearest-neighbour



Bilinear



Bicubic



# Bicubic Interpolation

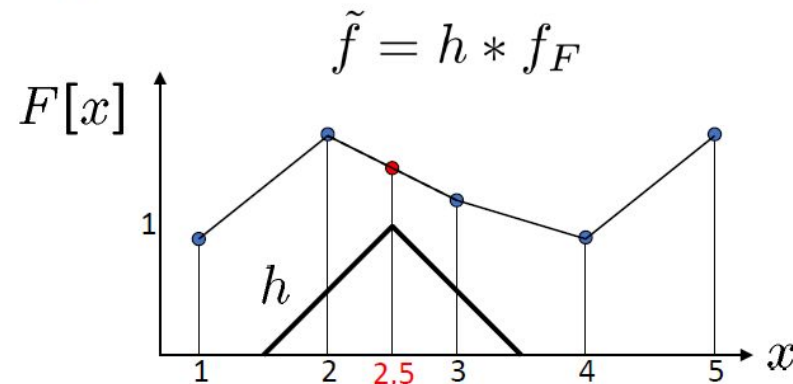
## Image interpolation

- What if we don't know  $f$  ?

- Guess an approximation:  $\tilde{f}$
- Can be done in a principled way: filtering
- Convert  $F$  to a continuous function:

$$f_F(x) = F\left(\frac{x}{d}\right) \text{ when } \frac{x}{d} \text{ is an integer, } 0 \text{ otherwise}$$

- Reconstruct by convolution with a *reconstruction filter*,  $h$



$d = 1$  in this example

# Bicubic Interpolation

找出擬合曲線並計算插值

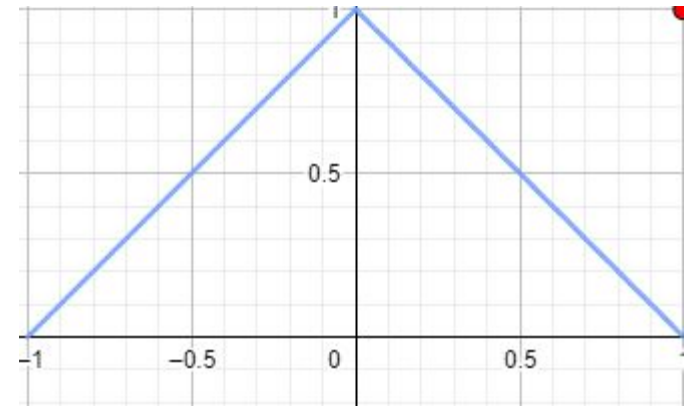
→

使用 Interpolation Kernel 做 Convolution 得到插值



$$f(x) = -0.5x + 3, x = 2.5$$

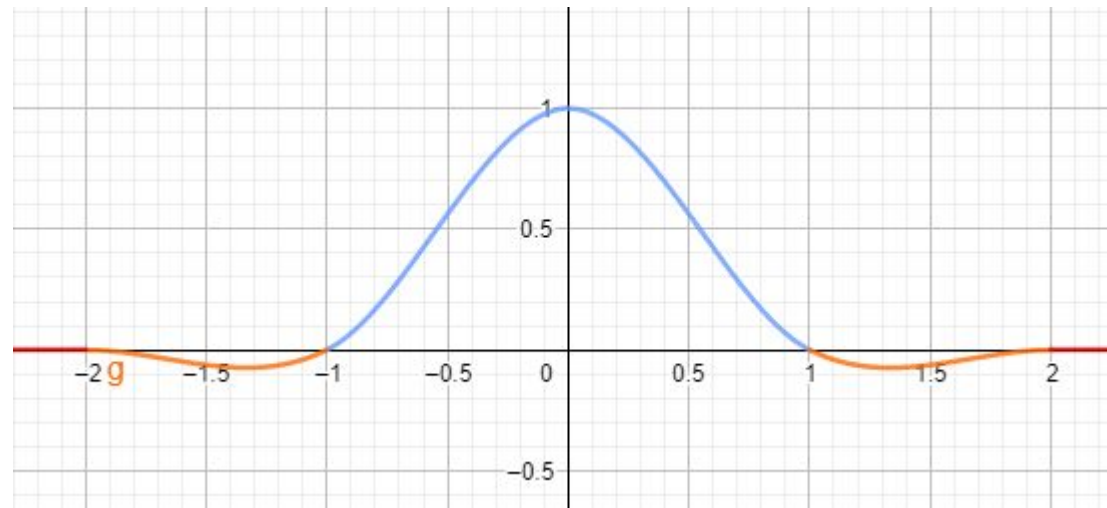
$$\frac{(2 + 1.5)}{2} = 1.75$$



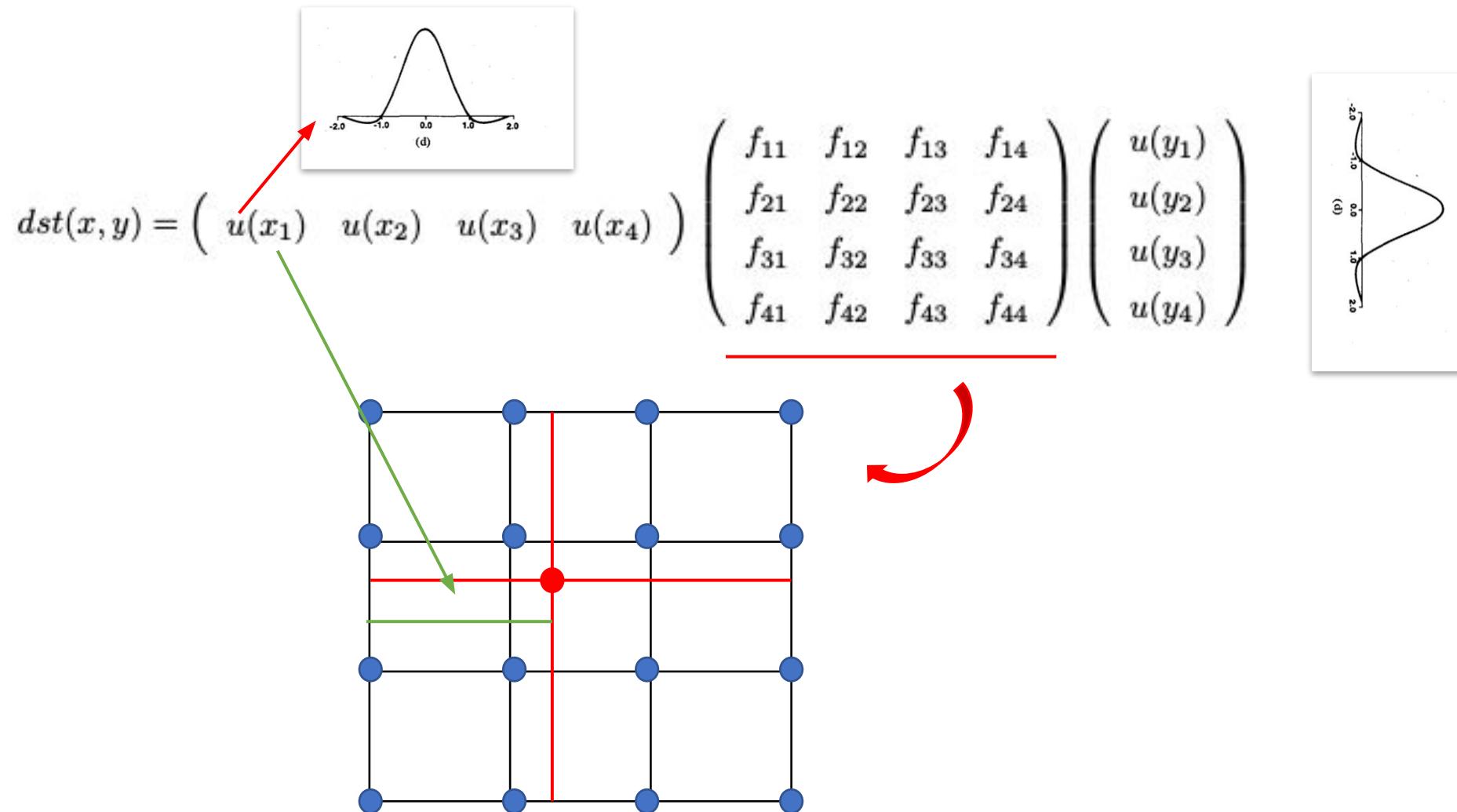
# Bicubic Interpolation

Bicubic 使用的 Interpolation Kernel:  $a = -0.5$  or  $-0.75$

$$W(x) = \begin{cases} (a+2)|x|^3 - (a+3)|x|^2 + 1 & \text{for } |x| \leq 1, \\ a|x|^3 - 5a|x|^2 + 8a|x| - 4a & \text{for } 1 < |x| < 2, \\ 0 & \text{otherwise,} \end{cases}$$



# Bicubic Interpolation



# 作業 – 繳交格式

- 每人繳交兩份 .ipynb 檔案
- 作業名稱: n26102199\_蔡延隆\_HW1.ipynb、  
n26102199\_蔡延隆\_HW2.ipynb
- 留下 cell output (結果)
- 期限: 4 / 7 (五) 23:59
- Colab 連結: [點我](#), [點我](#)

# Office Hour

- 時間: 星期三 14:00 ~ 16:00
- 地點: 92810