

數位影像處理

影像特效

08160130 林采葳

08360086 王彥傑

指導老師：賈叢林 老師

目錄

1. 計畫動機

2. 計畫目標

- ① 放射狀像素化
- ② 漣漪特效
- ③ 魚眼特效
- ④ 捻轉特效

2. 像素特效

- ① 模糊特效
- ② 運動模糊
- ③ 放射狀模糊

3. 非真實感繪製

- ① 鉛筆素描
- ② 風格化
- ③ 懷舊特效
- ④ 光照特效
- ⑤ 流年特效

4. 結論 & 希望

計畫 動機

- 為我們整組幾乎沒有接觸過python，所以在程式碼的理解上就需要花比較多時間，先以理解課本的內容為主
-
- 和我們的生活息息相關，很常社群媒體媒體一打開來，就可以看到影像或影片都有套用特效，因此我們最後決定以影像特效為主題。

計畫目標

01

在製作專題的過程中，結合所學的知識與技能做出各種特效。同時也增強自己的技術。

02

在製作的過程中發現還有可以改善的地方，希望藉由製作過程學習相關技術(程式、公式)並且加強我們不足的部分。

03

利用本次專題研究了解影像處理相關的應用與公式，並實際製作。

幾何特效處理步驟

1

計算影像的中心點，作為特效處理的中心，
將影像座標原點由左上角移至影像中心

2

由直角坐標系統轉換成極座標系統

3

進行轉換運算

4

計算出新座標

5

執行影像內插

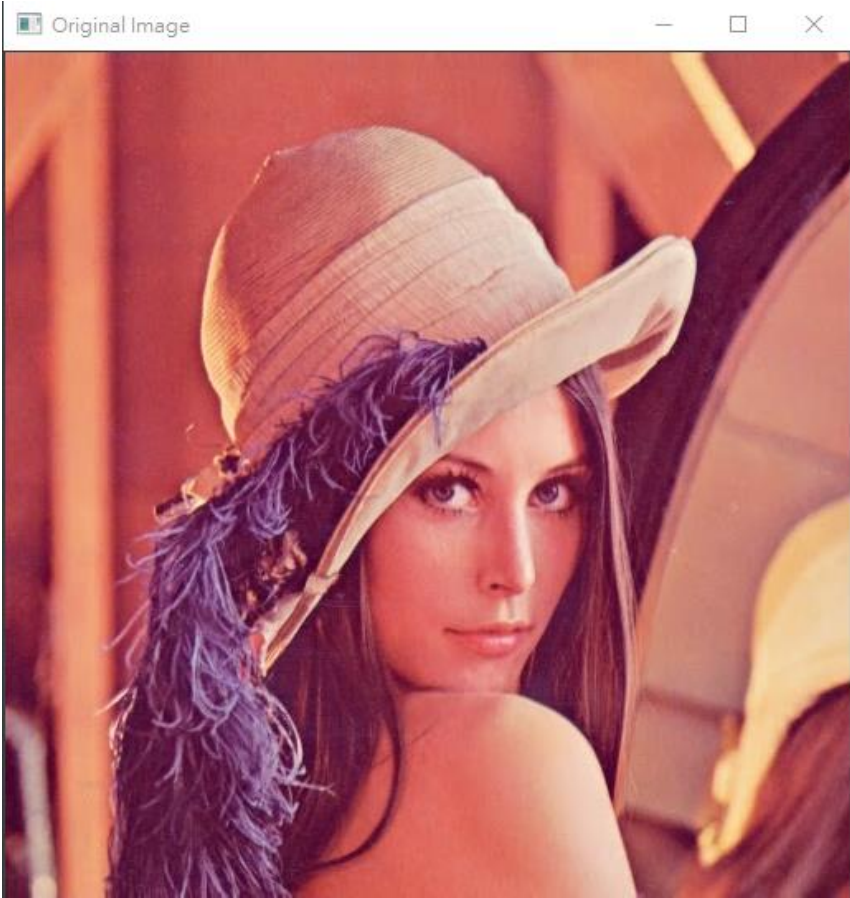
```

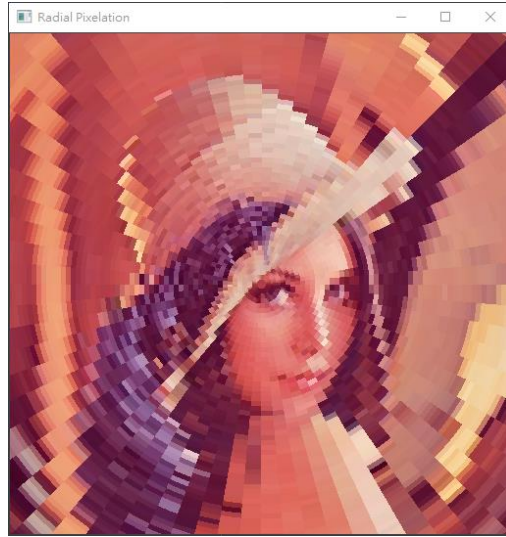
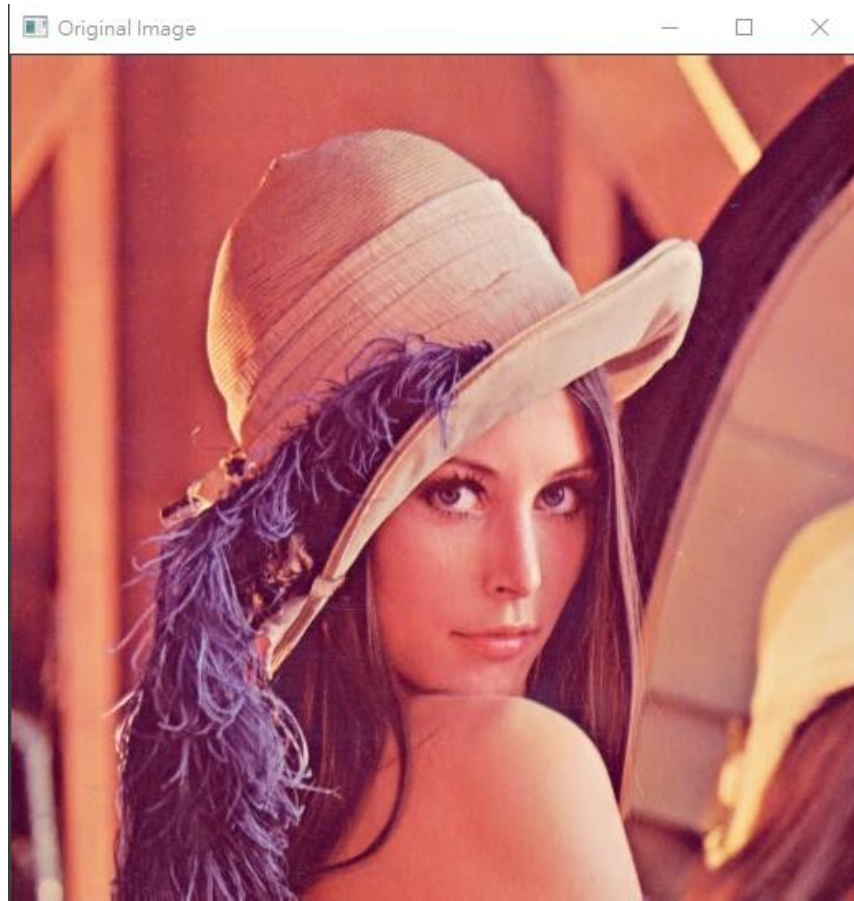
1  import numpy as np
2  import cv2
3
4  def radial_pixelation( f, delta_r, delta_theta ):
5      nr, nc = f.shape[:2]  #取出原始影像大小
6      map_x = np.zeros( [nr, nc], dtype = 'float32' )  # X 對應座標
7      map_y = np.zeros( [nr, nc], dtype = 'float32' )  # y 對應座標
8      x0, y0 = nr // 2, nc // 2  #定位影像中心
9      for x in range( nr ):
10         for y in range( nc ):
11             r = np.sqrt( ( x - x0 ) ** 2 + ( y - y0 ) ** 2 )  #直角座標系和極坐標系的轉換
12             if r == 0: theta = 0  #代表中心點(0,0)
13             else: theta = np.arccos( ( x - x0 ) / r )
14             r = r - r % delta_r  # r % delta_r
15             if y - y0 < 0: theta = -theta  #左半邊因為對稱所以theta減掉
16             theta = theta - theta % ( np.radians( delta_theta ) )
17             map_x[x,y] = np.clip( y0 + r * np.sin( theta ), 0, nc - 1 )
18             map_y[x,y] = np.clip( x0 + r * np.cos( theta ), 0, nr - 1 )
19         g = cv2.remap( f, map_x, map_y, cv2.INTER_LINEAR )  #用remap函式 雙線性內插法
20         return g
21
22  def main( ):
23      img1 = cv2.imread( "lena.bmp", -1 )
24      img2 = radial_pixelation( img1, 5, 5 )
25      cv2.imshow( "Original Image", img1 )
26      cv2.imshow( "Radial Pixelation", img2 )
27      cv2.waitKey( 0 )
28
29  main( )

```

放射狀像素化

與取樣技術相似，
只是取樣的方式變成在極座標上進行



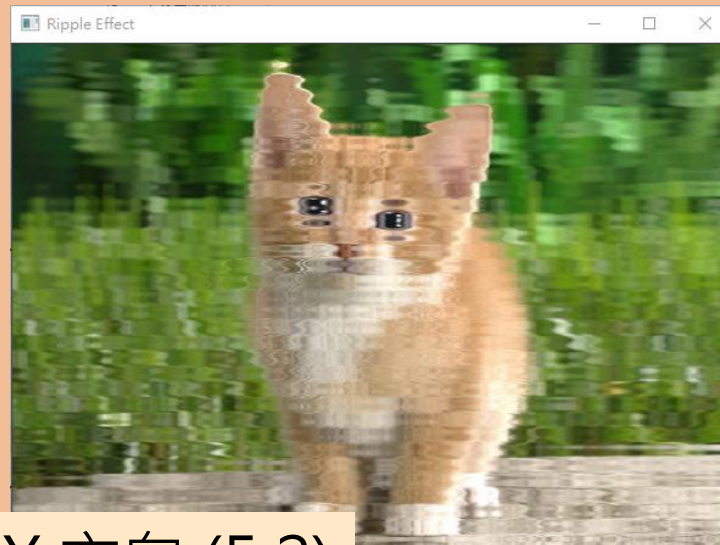


(5,5)

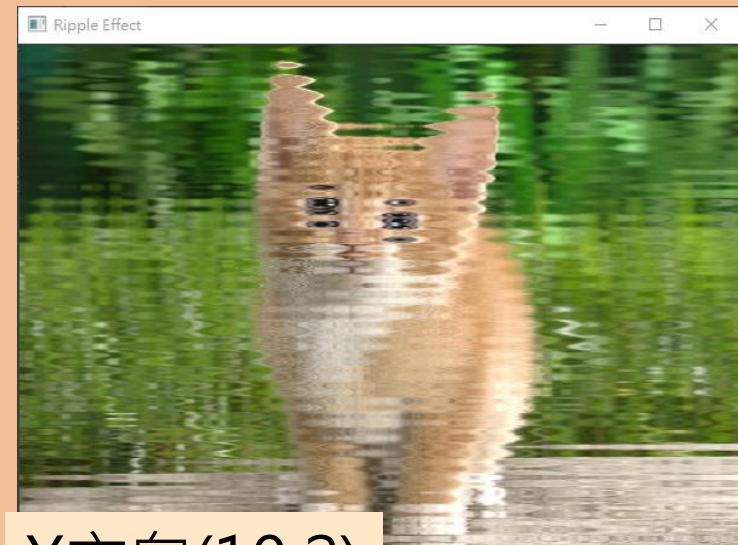
漣漪特效

利用正弦函數作為轉換函數，
來模擬水面上產生漣漪的視覺效果

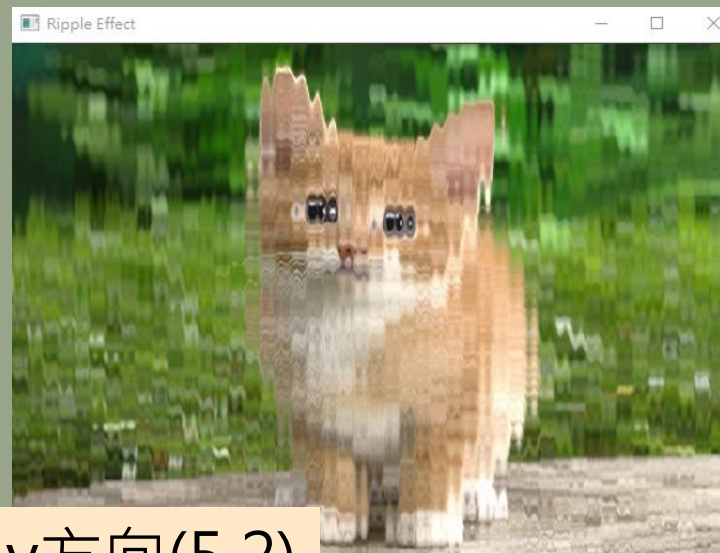
```
1 import numpy as np
2 import cv2
3
4 def ripple_effect( f, method, amplitude, period ):
5     nr, nc = f.shape[:2]
6     map_x = np.zeros( [nr, nc], dtype = 'float32' )
7     map_y = np.zeros( [nr, nc], dtype = 'float32' )
8     x0, y0 = nr // 2, nc // 2
9     for x in range( nr ):
10         for y in range( nc ):
11             if method == 1:      # x-direction
12                 xx = np.clip( x + amplitude * np.sin( x / period ), 0, nr - 1 )
13                 map_x[x,y] = y
14                 map_y[x,y] = xx
15             elif method == 2:    # y-direction
16                 yy = np.clip( y + amplitude * np.sin( y / period ), 0, nc - 1 )
17                 map_x[x,y] = yy
18                 map_y[x,y] = x
19             elif method == 3:    # x & y direction
20                 xx = np.clip( x + amplitude * np.sin( x / period ), 0, nr - 1 )
21                 yy = np.clip( y + amplitude * np.sin( y / period ), 0, nc - 1 )
22                 map_x[x,y] = yy
23                 map_y[x,y] = xx
24             else:                # Radial
25                 r = np.sqrt( ( x - x0 ) ** 2 + ( y - y0 ) ** 2 )
26                 if r == 0: theta = 0
27                 else: theta = np.arccos( ( x - x0 ) / r )
28                 r = r + amplitude * np.sin( r / period )
29                 if y - y0 < 0: theta = -theta
30                 map_x[x,y] = np.clip( y0 + r * np.sin( theta ), 0, nc - 1 )
31                 map_y[x,y] = np.clip( x0 + r * np.cos( theta ), 0, nr - 1 )
32     g = cv2.remap( f, map_x, map_y, cv2.INTER_LINEAR )
33     return g
34
35 def main( ):
36     img1 = cv2.imread( "Cat.bmp", -1 )
37     img2 = ripple_effect( img1, 4, 40, 2 )
38     cv2.imshow( "Original Image", img1 )
39     cv2.imshow( "Ripple Effect", img2 )
40     cv2.waitKey( 0 )
41
42 main( )
```



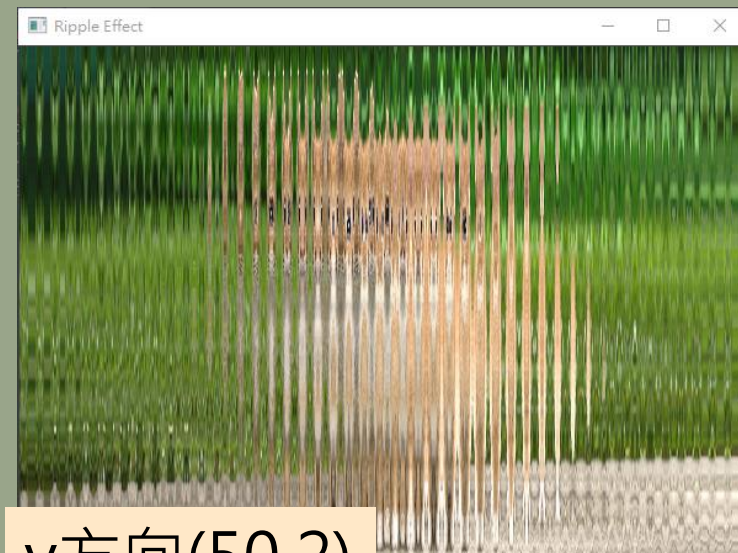
X 方向 (5,2)



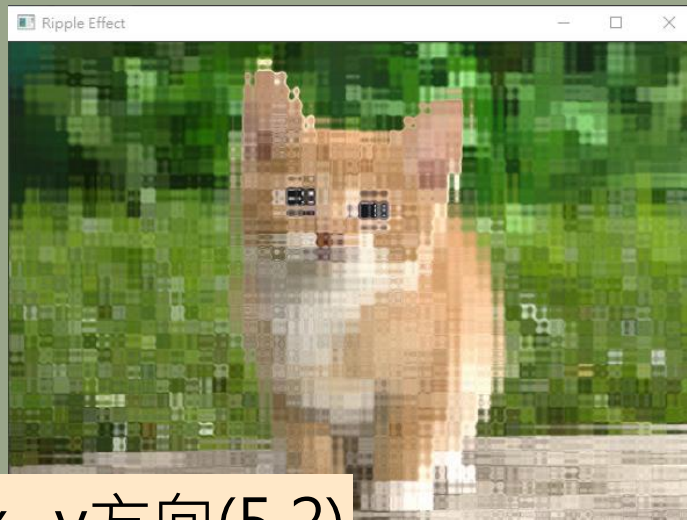
X方向(10,2)



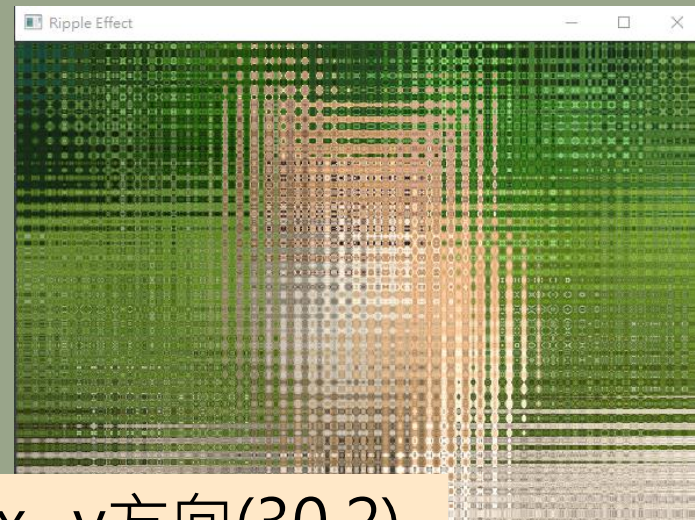
y方向(5,2)



y方向(50,2)



x, y方向(5,2)



x, y方向(30,2)



放射狀(5,2)



放射狀(40,2)

魚眼特效

視角比一般相機廣，可以接近
180°的視角，可以避免監控死
角的問題

但是魚眼相機的焦距非常短，
因此會有桶狀失真的現象，為
了使魚眼鏡頭中央的成像較佳，
因此採用雙立方內插法

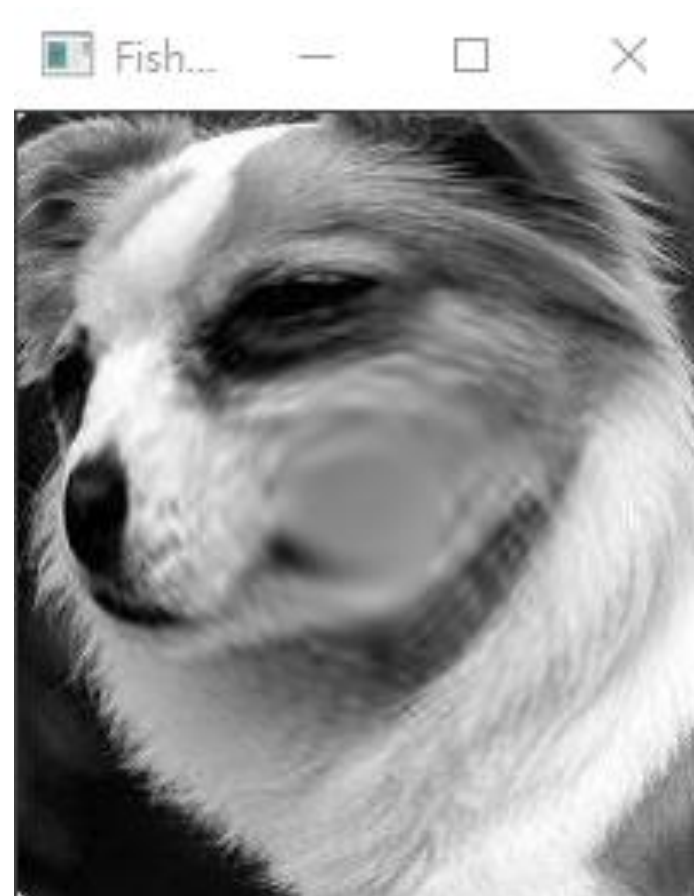
```
1 import numpy as np
2 import cv2
3
4 def fisheye_effect( f ):
5     nr, nc = f.shape[:2]
6     map_x = np.zeros( [nr, nc], dtype = 'float32' )
7     map_y = np.zeros( [nr, nc], dtype = 'float32' )
8     x0, y0 = nr // 2, nc // 2
9     R = np.sqrt( nr ** 2 + nc ** 2 ) / 2 #求出大R
10    for x in range( nr ):
11        for y in range( nc ):
12            r = np.sqrt( ( x - x0 ) ** 2 + ( y - y0 ) ** 2 )
13            if r == 0: theta = 0
14            else: theta = np.arccos( ( x - x0 ) / r )
15            r = ( r * r ) / R #R為最大半徑=對角線的二分之一
16            if y - y0 < 0: theta = -theta
17            map_x[x,y] = np.clip( y0 + r * np.sin( theta ), 0, nc - 1 )
18            map_y[x,y] = np.clip( x0 + r * np.cos( theta ), 0, nr - 1 )
19    g = cv2.remap( f, map_x, map_y, cv2.INTER_CUBIC )
20    return g
21
22 def main( ):
23     img1 = cv2.imread( "Dog.jpg", -1 )
24     img2 = fisheye_effect( img1 )
25     cv2.imshow( "Original Image", img1 )
26     cv2.imshow( "Fisheye Effect", img2 )
27     cv2.waitKey( 0 )
28
29 main( )
```




原始圖像



彩色圖像



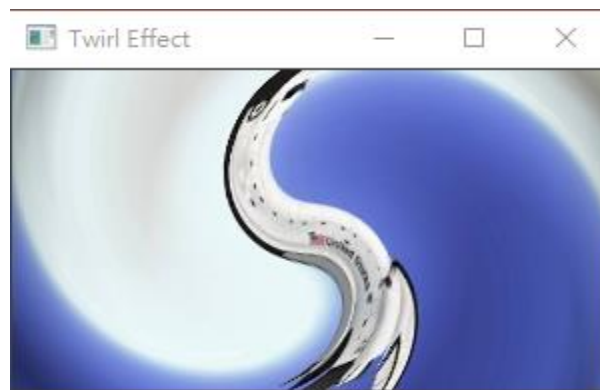
灰階圖像

捻轉特效

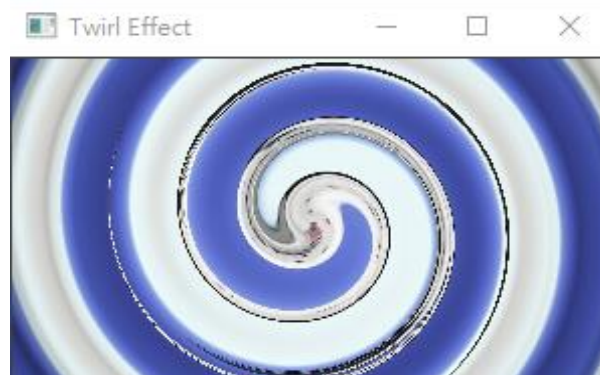
像是咖啡杯中使用湯匙旋轉拉花，利用捻轉參數改變捻轉方向與捻轉的量，所形成視覺上的特殊效果。

因為 r 不變，所以捻轉參數 K 值越小，捻轉的量越大

```
1 import numpy as np
2 import cv2
3
4 def twirl_effect( f, K ):
5     nr, nc = f.shape[:2]
6     map_x = np.zeros( [nr, nc], dtype = 'float32' )
7     map_y = np.zeros( [nr, nc], dtype = 'float32' )
8     x0, y0 = nr // 2, nc // 2
9     for x in range( nr ):
10         for y in range( nc ):
11             r = np.sqrt( ( x - x0 ) ** 2 + ( y - y0 ) ** 2 )
12             if r == 0: theta = 0
13             else:      theta = np.arccos( ( x - x0 ) / r )
14             if y - y0 < 0: theta = -theta
15             phi = theta + r / K # K值越小捻轉越多 可以是正的也可以是負的
16             map_x[x,y] = np.clip( y0 + r * np.sin( phi ), 0, nc - 1 )
17             map_y[x,y] = np.clip( x0 + r * np.cos( phi ), 0, nr - 1 )
18     g = cv2.remap( f, map_x, map_y, cv2.INTER_LINEAR )
19     return g
20
21 def main( ):
22     img1 = cv2.imread( "Space_Shuttle.bmp", -1 )
23     img2 = twirl_effect( img1, 50 )
24     cv2.imshow( "Original Image", img1 )
25     cv2.imshow( "Twirl Effect", img2 )
26     cv2.waitKey( 0 )
27
28 main( )
```



$K = 50$



$K = 10$



$K = 250$



$K = -50$




$K = -10$



$K = -250$

像素特效

基於像素的數學運算，
相關的會牽涉到局部的處理、

影像濾波等等的技術，
來達到視覺上的特殊效果。

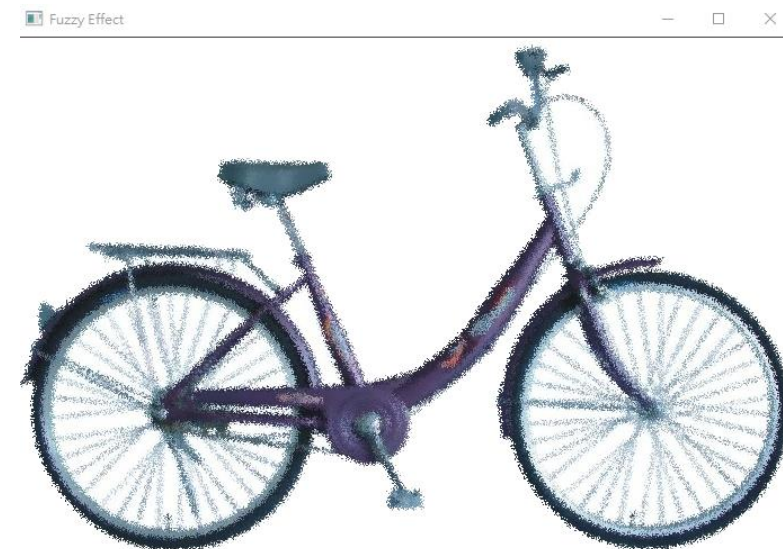
模糊特效

可以產生點狀模糊的視覺效果，
利用選取窗的大小改變模糊的
效果。

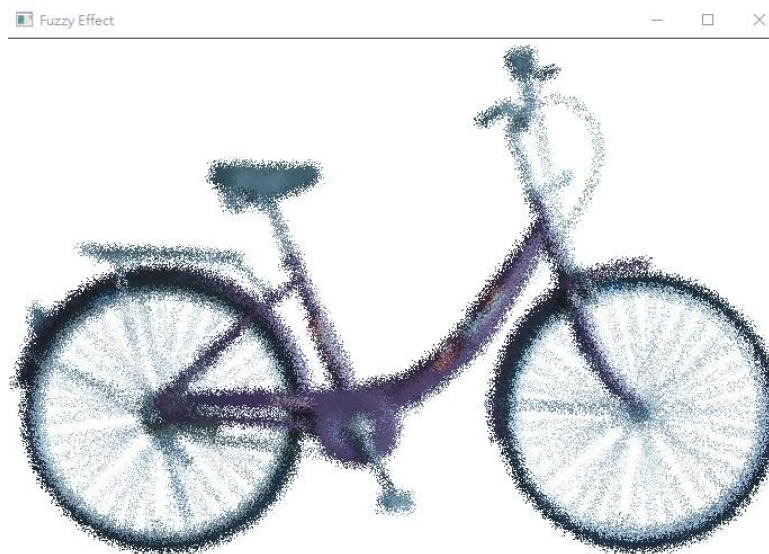
```
1 import numpy as np
2 import cv2
3 from numpy.random import uniform
4
5 def fuzzy_effect( f, W ):
6     g = f.copy( )
7     nr, nc = f.shape[:2]
8     for x in range( nr ):
9         for y in range( nc ):
10             xp = int( x + W * uniform() - W // 2 ) # W為視窗大小 值越大模糊效果越明顯
11             yp = int( y + W * uniform() - W // 2 )
12             xp = np.clip( xp, 0, nr - 1 )
13             yp = np.clip( yp, 0, nc - 1 )
14             g[x,y] = f[xp,yp]
15     return g
16
17 def main( ):
18     img1 = cv2.imread( "Bicycle.bmp", -1 )
19     img2 = fuzzy_effect( img1, 3 )
20     cv2.imshow( "Original Image", img1 )
21     cv2.imshow( "Fuzzy Effect", img2 )
22     cv2.waitKey( 0 )
23
24 main( )
```



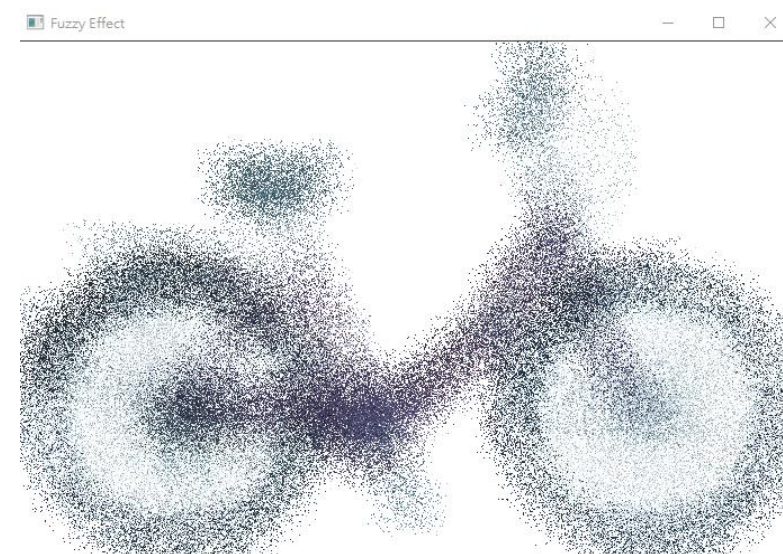
$W=3$



$W=5$



$W=10$



$W=50$

運動模糊

由於物體本身是運動狀態或相機拍攝時產生晃動所產生的特殊效果。

利用核函數中的直線長度與角度，
改變運動量與運動方向。

步驟：

- 定義2D濾波器（一條直線），長度決定運動量，方向決定角度
- 將核函數的系數總和正規化為1
- 使用二維的影像濾波

```
1 import numpy as np
2 import cv2
3
4 def motion_blur( f, length, angle ):
5     nr, nc = f.shape[:2]
6     filter = np.zeros( [ length, length ] ) # W * W
7     x0, y0 = length // 2, length // 2
8     x_len = round( x0 * np.cos( np.radians( angle ) ) )
9     y_len = round( y0 * np.sin( np.radians( angle ) ) )
10    x1, y1 = int( x0 - x_len ), int( y0 - y_len )
11    x2, y2 = int( x0 + x_len ), int( y0 + y_len )
12    cv2.line( filter, ( y1, x1 ), ( y2, x2 ), ( 1, 1, 1 ) ) #白線
13    filter /= np.sum( filter ) #加總=1
14    g = cv2.filter2D( f, -1, filter )
15    return g
16
17 def main( ):
18     img1 = cv2.imread( "Pepper.bmp", -1 )
19     img2 = motion_blur( img1, 20, 0 ) #(影像,像素,角度)
20     cv2.imshow( "Original Image", img1 )
21     cv2.imshow( "Motion Blur", img2 )
22     cv2.waitKey( 0 )
23
24 main( )
```






放射狀模糊

以運動模糊為基礎，模擬成放射狀模糊，調整濾波器的大小，影響放射狀模糊的效果。

濾波器 l 越大，放射狀模糊的效果越明顯

步驟：

假設濾波器為 l ，用來控制放射狀模糊的量

根據 (x,y) 座標，計算相對應的 (r,θ) 座標

根據鄰近像素座標，取平均值輸出

```
1 import numpy as np
2 import cv2
3
4 def radial_blur( f, filter_size ):
5     g = f.copy( )
6     nr, nc = f.shape[:2]
7     x0, y0 = nr // 2, nc // 2
8     half = filter_size // 2
9     for x in range( nr ):
10         for y in range( nc ):
11             r = np.sqrt( ( x - x0 ) ** 2 + ( y - y0 ) ** 2 )
12             if r == 0: theta = 0
13             else: theta = np.arccos( ( x - x0 ) / r )
14             if y - y0 < 0: theta = -theta
15             R = G = B = n = 0
16             for k in range( -half, half + 1 ):
17                 phi = theta + np.radians( k )
18                 xp = int( round( x0 + r * np.cos( phi ) ) )
19                 yp = int( round( y0 + r * np.sin( phi ) ) )
20                 if ( xp >= 0 and xp < nr and yp >= 0 and yp < nc ):
21                     R += f[xp,yp,2]
22                     G += f[xp,yp,1]
23                     B += f[xp,yp,0]
24                     n += 1
25             R = round( R / n )
26             G = round( G / n )
27             B = round( B / n )
28             g[x,y,2] = np.uint8( R )
29             g[x,y,1] = np.uint8( G )
30             g[x,y,0] = np.uint8( B )
31     return g
32
33 def main( ):
34     img1 = cv2.imread( "person.bmp", -1 )
35     img2 = radial_blur( img1, 15 )
36     cv2.imshow( "Original Image", img1 )
37     cv2.imshow( "Radial Blur", img2 )
38     cv2.waitKey( 0 )
39
40 main( )
```

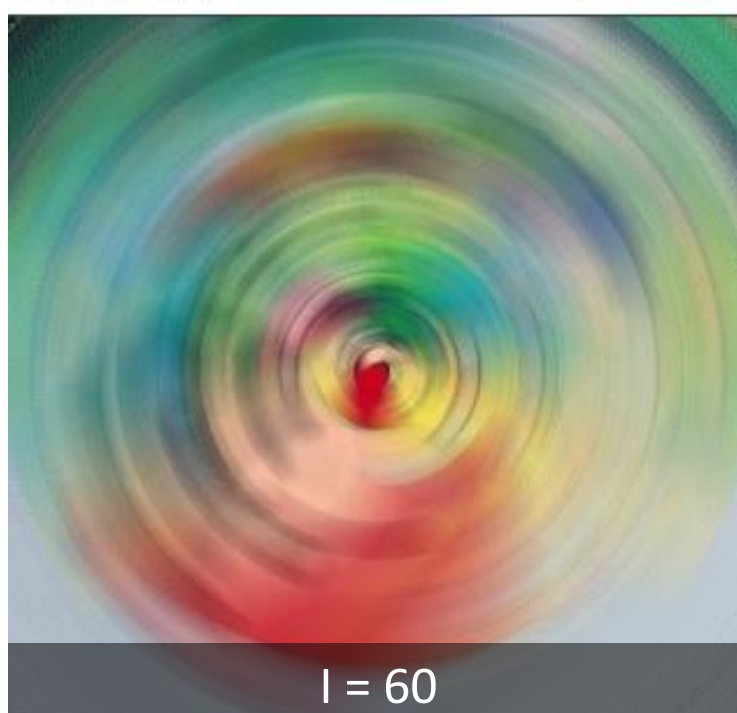

Original Image



Radial Blur



Radial Blur



非 真實感 繪製

微電腦圖學的分支，

目的是模仿藝術家或畫家的繪製風格，

以電腦運算方式模擬，

並自動產生繪製的特殊效果

鉛筆素描

利用鉛筆素描函式，模擬鉛筆

素描的特殊效果

```
1 import numpy as np
2 import cv2
3
4 img = cv2.imread( "Brunch.bmp", -1 )
5 img1, img2 = cv2.pencilSketch( img )
6 cv2.imshow( "Original Image", img )
7 cv2.imshow( "Pencil Sketch 1", img1 )
8 cv2.imshow( "Pencil Sketch 2", img2 )
9 cv2.waitKey( 0 )
```



風格化

利用風格化函式，產生類似卡通的特殊效果

```
1 import numpy as np
2 import cv2
3
4 img1 = cv2.imread( "Brunch.bmp", -1 )
5 img2 = cv2.stylization( img1 )
6 cv2.imshow( "Original Image", img1 )
7 cv2.imshow( "Stylization", img2 )
8 cv2.waitKey( 0 )
```

Original Image



懷舊特效

懷舊特效是將圖像的 RGB 三個向量分別按照一定比例進行處理的結果

```
1  import cv2
2  import numpy as np
3
4  #讀取影像
5  img = cv2.imread('person.bmp')
6
7  #取出原始影像大小
8  nr, nc = img.shape[:2]
9
10 dst = np.zeros((nr, nc, 3), dtype="uint8")
11
12 for i in range(nr):
13     for j in range(nc):
14         B = 0.272*img[i,j][2] + 0.534*img[i,j][1] + 0.131*img[i,j][0] #懷舊特效RGB比例
15         G = 0.349*img[i,j][2] + 0.686*img[i,j][1] + 0.168*img[i,j][0] #懷舊特效RGB比例
16         R = 0.393*img[i,j][2] + 0.769*img[i,j][1] + 0.189*img[i,j][0] #懷舊特效RGB比例
17         if B>255:
18             B = 255
19         if G>255:
20             G = 255
21         if R>255:
22             R = 255
23         dst[i,j] = np.uint8((B, G, R))
24
25
26 cv2.imshow('src', img)
27 cv2.imshow('dst', dst)
28 cv2.waitKey()
29 cv2.destroyAllWindows()
```



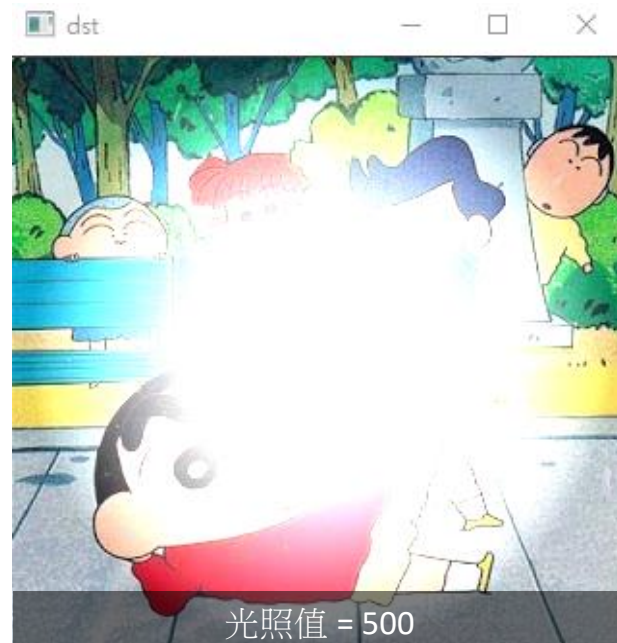
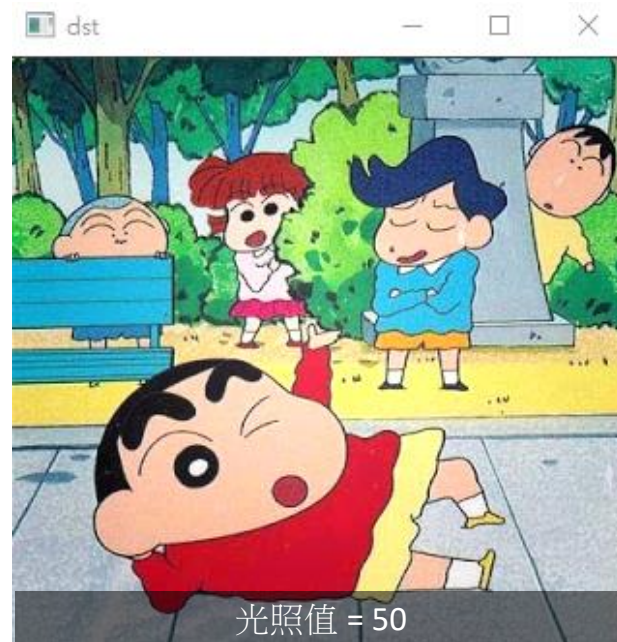

光照特效

為影像存在一個類似光暈的特效

透過計算兩點之間的距離，判斷
此距離和圖像中心圓半徑的大小
關係

中心圓範圍內的灰階值增強，範圍外的灰階值則不變，並結合邊界範圍判斷最後的效果

```
1  import cv2
2  import math
3  import numpy as np
4
5
6  img = cv2.imread('person.bmp')
7
8
9  nr, nc = img.shape[:2]
10
11  x0, y0 = nr // 2, nc // 2  #定位影像中心
12  radius = min(x0, y0)
13
14  strength = 200  #光照強度
15
16
17  dst = np.zeros((nr, nc, 3), dtype="uint8")
18
19
20  for i in range(nr):
21      for j in range(nc):
22          distance = math.pow((y0-j), 2) + math.pow((x0-i), 2)  #計算現在點到光照中心的距離
23          B = img[i,j][0]
24          G = img[i,j][1]
25          R = img[i,j][2]
26          if (distance < radius * radius):
27              result = (int)(strength*( 1.0 - math.sqrt(distance) / radius ))  #按照距離大小計算增強的光照值
28              B = img[i,j][0] + result
29              G = img[i,j][1] + result
30              R = img[i,j][2] + result
31
32              B = min(255, max(0, B))  #判斷邊界 防止越界
33              G = min(255, max(0, G))  #判斷邊界 防止越界
34              R = min(255, max(0, R))  #判斷邊界 防止越界
35              dst[i,j] = np.uint8((B, G, R))
36          else:
37              dst[i,j] = np.uint8((B, G, R))
38
39  cv2.imshow('src', img)
40  cv2.imshow('dst', dst)
41  cv2.waitKey()
42  cv2.destroyAllWindows()
```

流年特效

透過RGB的B像素值開根號並乘上一個參數，將影像轉換為具有年代感或歲月感的特效

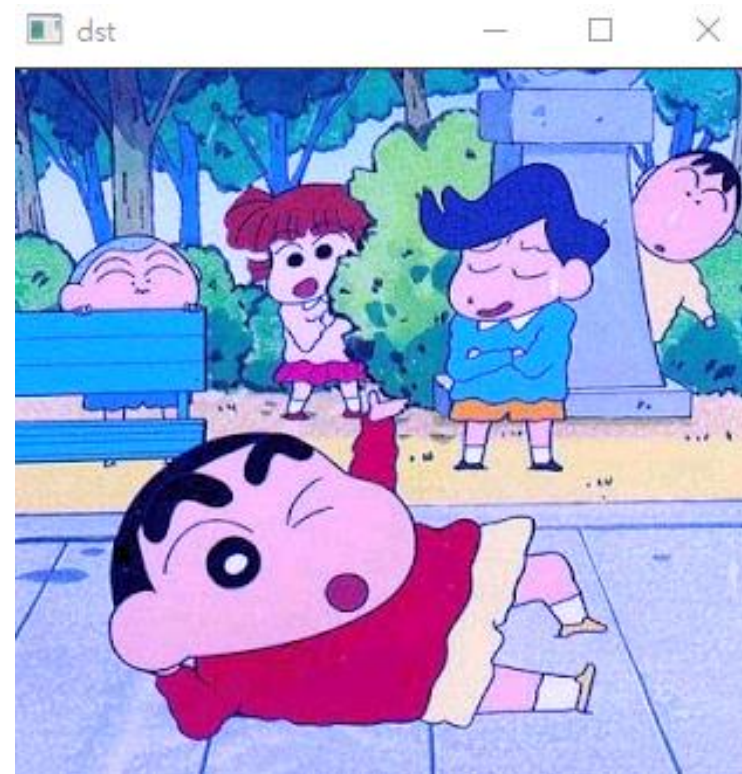
```
1  import cv2
2  import math
3  import numpy as np
4
5
6  img = cv2.imread('person.bmp')
7
8
9  nr, nc = img.shape[:2]
10
11
12  dst = np.zeros((nr, nc, 3), dtype="uint8")
13
14
15  for i in range(nr):
16      for j in range(nc):
17          B = math.sqrt(img[i,j][0]) * 20 #開根號 * 參數(12最剛好)
18          G = img[i,j][1]
19          R = img[i,j][2]
20          if B>255:
21              B = 255
22          dst[i,j] = np.uint8((B, G, R))
23
24
25  cv2.imshow('src', img)
26  cv2.imshow('dst', dst)
27  cv2.waitKey()
28  cv2.destroyAllWindows()
```



原始影像



參數 = 12



參數 = 20

結論 & 希望

01

我們的技術並不是很成熟，所以這次的專題沒有做出類似軟體的介面，把特效串聯再一起。

02

在製作的過程中，在前期理解程式碼的時候，就花費蠻多時間，所以希望在以後把介面、按鈕.....等等的製作完成。

謝謝大家