Method

- 這次作業使用 opencv 來做圖片的讀取；使用 numpy 來做矩陣相關運算。
  - Rotate：

(x_new, y_new)是旋轉後的新座標，則其對應的原始座標(x_old, y_old)可透過反向旋轉計算(因為旋轉矩陣是逆時針旋轉)。

```
x_old = (x_new - center_x) * cos_a + (y_new - center_y) * sin_a + center_x
y_old = -(x_new - center_x) * sin_a + (y_new - center_y) * cos_a + center_y
```

  - Warp：

把指定圖片經過矩陣 H 把指定圖片上的點轉換到目標的點上。Homography 變換的關係式為：$\begin{cases} X = h11x + h12y + h13 - h31xX - h32yX \\ Y = h21x + h22y + h23 - h31xY - h32yY \end{cases}$

```
def homography(img_pts, dst_pts):
    A = []
    b = []
    # 4 個點
    for i in range(4):
        x = img_pts[i][0]
        y = img_pts[i][1]
        X = dst_pts[i][0]
        Y = dst_pts[i][1]

        A.append([x, y, 1, 0, 0, 0, -X*x, -X*y])
        b.append(X)
        A.append([0, 0, 0, x, y, 1, -Y*x, -Y*y])
        b.append(Y)

    A = np.array(A, dtype=np.float64)
    b = np.array(b, dtype=np.float64)

    h = np.linalg.solve(A, b)
    # 把h33設為1
    H = np.array([
        [h[0], h[1], h[2]],
        [h[3], h[4], h[5]],
        [h[6], h[7], 1.0]
    ], dtype=np.float64)
    return H
```

我們先利用上面的關係式，透過相對應的四個點來去做計算，並且列出 8 條線性方程，再把他變成 H 的矩陣的形式。

  - Nearest：

```
def nearest(img, x, y):
    x_n = int(round(x))
    y_n = int(round(y))
    x_n = clamp(x_n, 0, img.shape[1]-1)
    y_n = clamp(y_n, 0, img.shape[0]-1)
    return img[y_n, x_n].astype(np.float32)
```

直接將所對應的座標取整再進行取值，當作是最接近的像素值。

- **Bilinear**

```python
def bilinear(img, x, y):
    x1 = int(np.floor(x))
    y1 = int(np.floor(y))
    x2 = x1 + 1
    y2 = y1 + 1

    dx = x - x1
    dy = y - y1

    x1 = clamp(x1, 0, img.shape[1]-1)
    x2 = clamp(x2, 0, img.shape[1]-1)
    y1 = clamp(y1, 0, img.shape[0]-1)
    y2 = clamp(y2, 0, img.shape[0]-1)

    top = (1 - dx) * img[y1, x1].astype(np.float32) + dx * img[y1, x2].astype(np.float32)
    bottom = (1 - dx) * img[y2, x1].astype(np.float32) + dx * img[y2, x2].astype(np.float32)
    val = (1 - dy) * top + dy * bottom
    return val
```

使用周圍四個像素進行加權平均計算內插然後取兩次

- **Bicubic：**

```python
def cubic_interpolate(p0, p1, p2, p3, x):
    a = (-0.5 * p0) + (1.5 * p1) - (1.5 * p2) + (0.5 * p3)
    b = (p0) - (2.5 * p1) + (2.0 * p2) - (0.5 * p3)
    c = (-0.5 * p0) + (0.5 * p2)
    d = p1
    return a*(x**3) + b*(x**2) + c*x + d
```

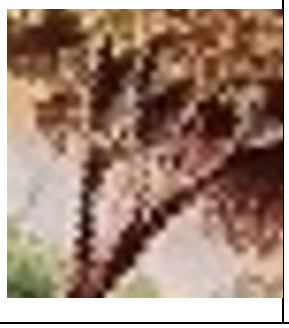
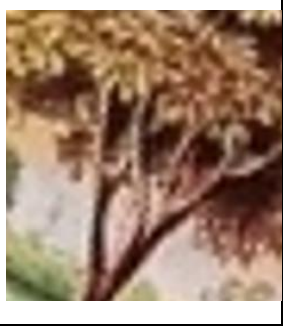
先利用 cubic_interpolate 來做預處理，並且導出公式

```python
# 先對 x 方向做三次插值
col_values = np.zeros((4, img.shape[2]), dtype=np.float32)
for row in range(4):
    col_values[row] = cubic_interpolate(
        pixels[row, 0],
        pixels[row, 1],
        pixels[row, 2],
        pixels[row, 3],
        fx
    )
# 再對 y 方向做三次插值
val = cubic_interpolate(
    col_values[0],
    col_values[1],
    col_values[2],
    col_values[3],
    fy
```

套用投影片上的公式，每個點需要四個相鄰的點來近似曲線，然後先對列做一次再對行做一次，因此需要周圍的 16 個點來獲得近似值。套用上面求得的公式來加速計算。

Result

- Rotate 結果比較：

原圖	nearest	bilinear	bicubic
			

- Warp 結果比較：

nearest	bilinear	bicubic
		

1. 圖片中像素的鋸齒感：nearest > bilinear > bicubic
2. 影像的品質：bicubic > bilinear > nearest

Feedback

這次的作業讓我了解三種基本的插值方法來做運算，並且去動手做了圖片的旋轉和將一張指定的圖片嵌進去另一張圖片裡面。在做 bicubic 的時候.輸出的圖片會有一些奇怪顏色的像素，後來改善不能用 int8 去做，運算結果可能會產生誤差和溢位，後來改成 float32 來做圖片的計算。