Feature-Based Image Metamorphosis

Thaddens Beier、Shawn Neely

Siggraph 1992

組員名稱:葉冠宏(108753208)

演算法

$$u = \frac{(X-P)\cdot(Q-P)}{\|Q-P\|^2} \quad (1)$$

$$v = \frac{(X-P)\cdot Perpendicular(Q-P)}{\|Q-P\|} \quad (2)$$

$$X' = P' + u\cdot(Q'-P') + \frac{v\cdot Perpendicular(Q'-P')}{\|Q'-P'\|} \quad (3)$$

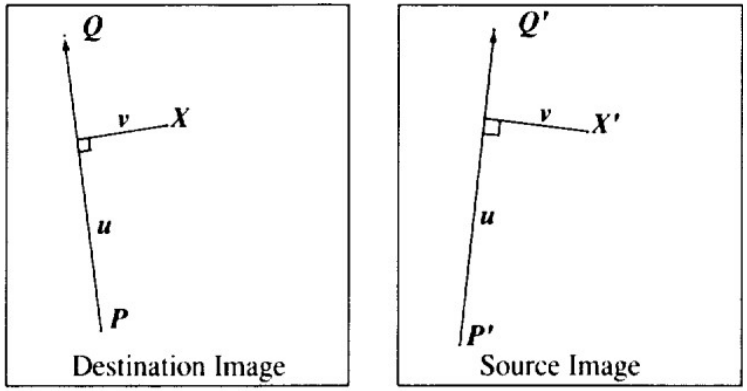


Figure 1: Single line pair

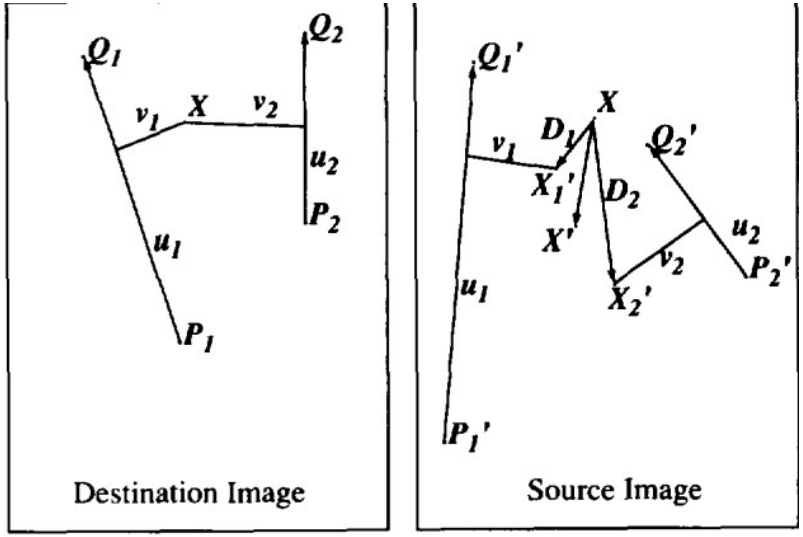


Figure 3: Multiple line pairs

$$weight = \left(\frac{length^p}{(a+dist)}\right)^b$$

For each pixel X in the destination
  $DSUM = (0,0)$
  $weightsum = 0$
  For each line $P_i Q_i$
    calculate u,v based on $P_i Q_i$
    calculate X'_i based on u,v and $P_i'Q_i'$
    calculate displacement $D_i = X_i' - X_i$ for this line
    $dist$ = shortest distance from X to $P_i Q_i$
    $weight = (length^p / (a + dist))^b$
    $DSUM \mathrel{+}= D_i * weight$
    $weightsum \mathrel{+}= weight$
  $X' = X + DSUM / weightsum$
  destinationImage(X) = sourceImage(X')

參數設定

$$weight = \left(\frac{length^{p}}{(a + dist)} \right)^{b}$$

A:1

B:2=> [0.5,2] is the most useful. If it is large,
then every pixel will be affected only by the line nearest

P:0=>suggest [0,1]. if it is zero, then all lines have the same weight. if it is one, then longer lines have a greater relative weight than shorter lines.

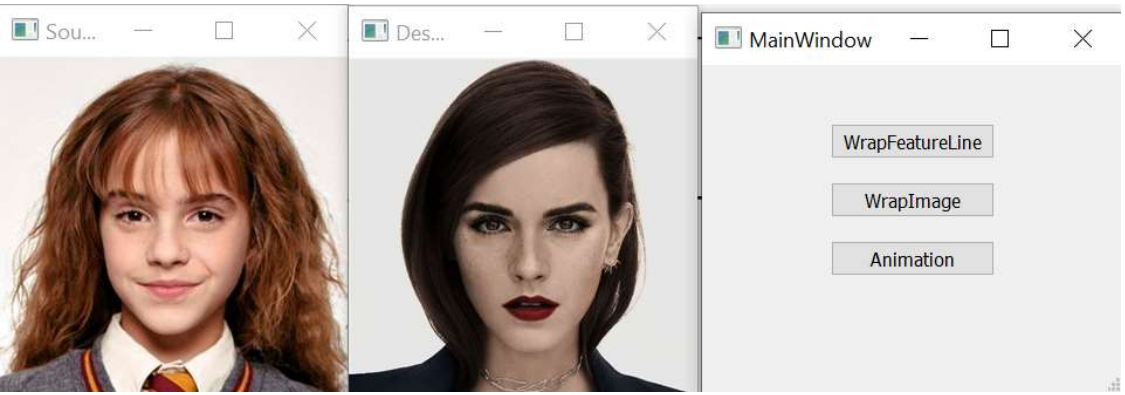
$$x_m = (1 - \alpha)x_i + \alpha x_j$$
$$y_m = (1 - \alpha)y_i + \alpha y_j$$

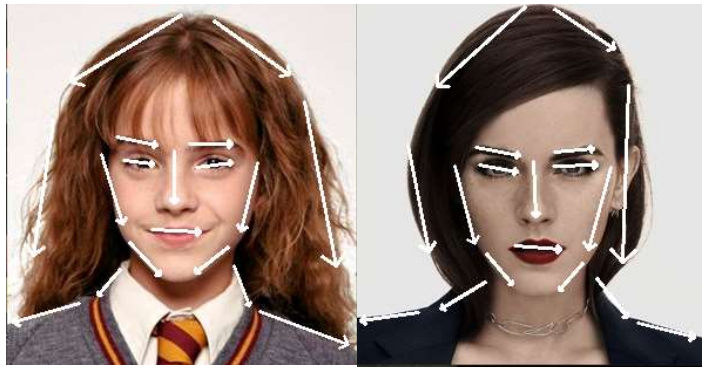
Alpha: 0.5

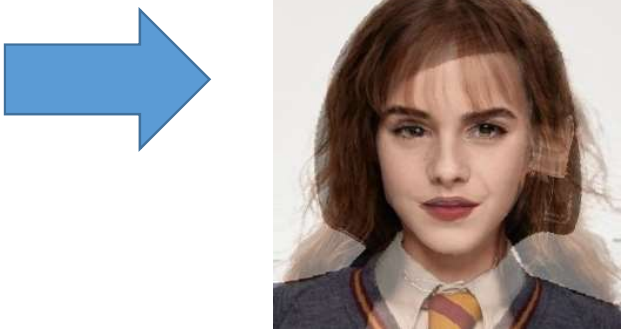
輸入及輸出

- 輸入:兩張image
- 輸出:warped image 和其轉換的動畫

流程圖(標出實作區塊)







關鍵程式碼

```
for i in range(height):
    for j in range(width):
        PSUM = [0, 0]
        u, v, weight = 0, 0, 0
        weightsum = 0.0
        for index in range(len(wrap_line)):
            X = [i, j]
            X_P = np.array(X)-np.array(wrap_line[index].start_point)
            Q_P = wrap_line[index].vector
            u = np.dot(X_P, Q_P)/wrap_line[index].square_length
            V_Q_P = wrap_line[index].perpendicular
            v = np.dot(X_P, V_Q_P)/wrap_line[index].length
            img_Q_P = img_line[index].vector
            V_img_Q_P = img_line[index].perpendicular
            X_new = np.array(img_line[index].start_point)+np.array(u)*np.array(img_Q_P)+np.array(v)*np.array(V_img_Q_P)/img_line[index].length
            if u<0:
                dist = np.sqrt(np.sum(np.square(X_new - np.array(img_line[index].start_point))))
            elif u>1:
                dist = np.sqrt(np.sum(np.square(X_new - np.array(img_line[index].end_point))))
            else:
                dist = abs(v)
            weight = math.pow(math.pow(wrap_line[index].length, 0)/(1+dist), 2)
            PSUM = np.array(PSUM) + np.array(X_new)*weight
            weightsum = weightsum + weight
        map_x[i, j] = ((np.array(PSUM)/weightsum)[0])
        map_y[i, j] = ((np.array(PSUM)/weightsum)[1])
        if map_x[i, j] < 0:
            map_x[i, j] = 0
        elif map_x[i, j] >= height:
            map_x[i, j] = height-1
        if map_y[i, j] < 0:
            map_y[i, j] = 0
        elif map_y[i, j] >= width:
            map_y[i, j] = width-1
        src_wrap[i, j, :] = img[math.floor(map_x[i, j]), math.floor(map_y[i, j]), :]
```

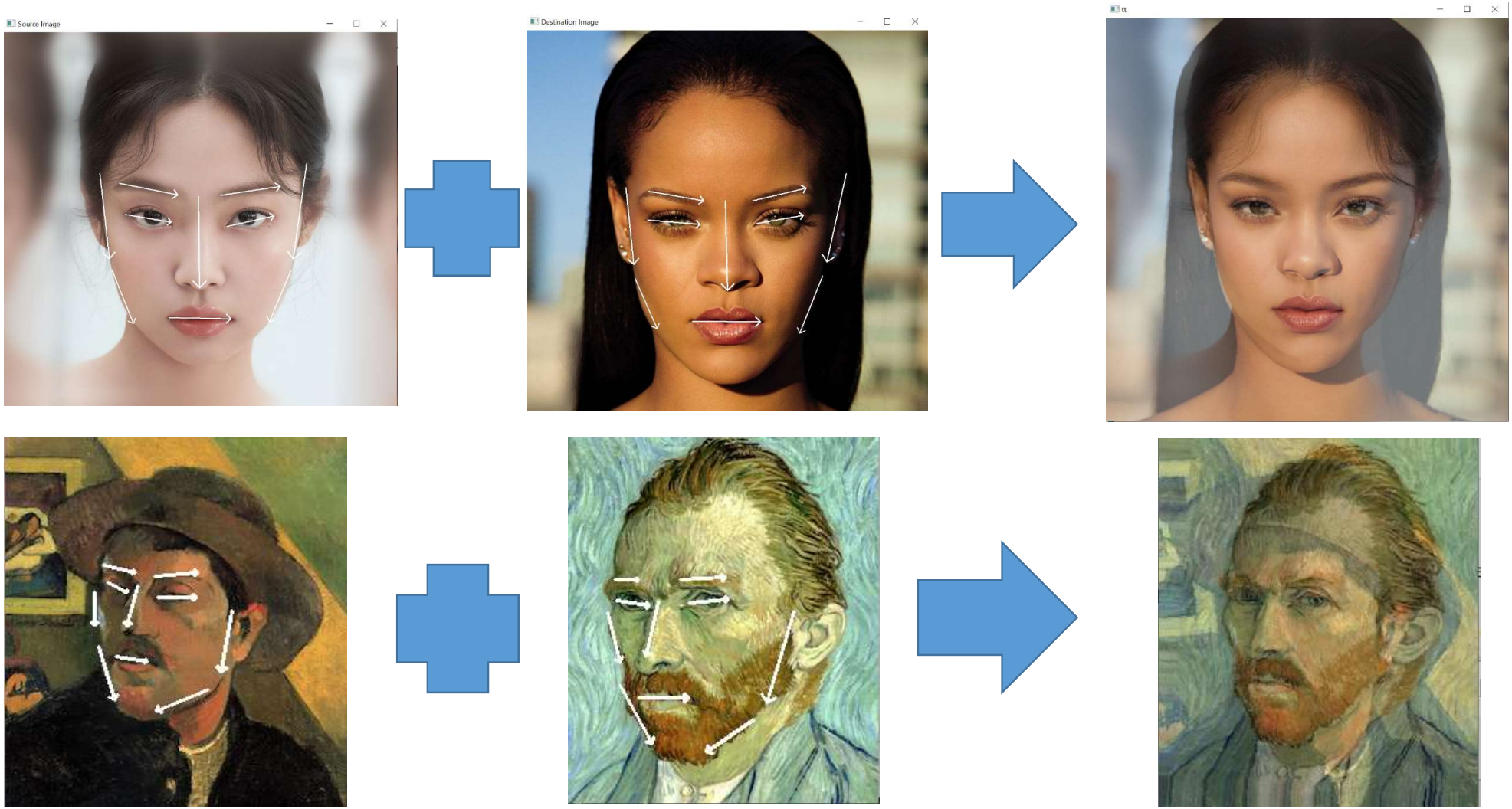
For each pixel X in the destination
    $DSUM = (0,0)$
    $weightsum = 0$
    For each line $P_i Q_i$
        calculate u,v based on $P_i Q_i$
        calculate X'_i based on u,v and $P_i'Q_i'$
        calculate displacement $D_i = X_i' - X_i$ for this line
        $dist$ = shortest distance from X to $P_i Q_i$
        $weight = (length^p / (a + dist))^b$
        $DSUM \mathrel{+}= D_i * weight$
        $weightsum \mathrel{+}= weight$
    $X' = X + DSUM / weightsum$
    destinationImage(X) = sourceImage(X')

實作遇到的挑戰和解決方式

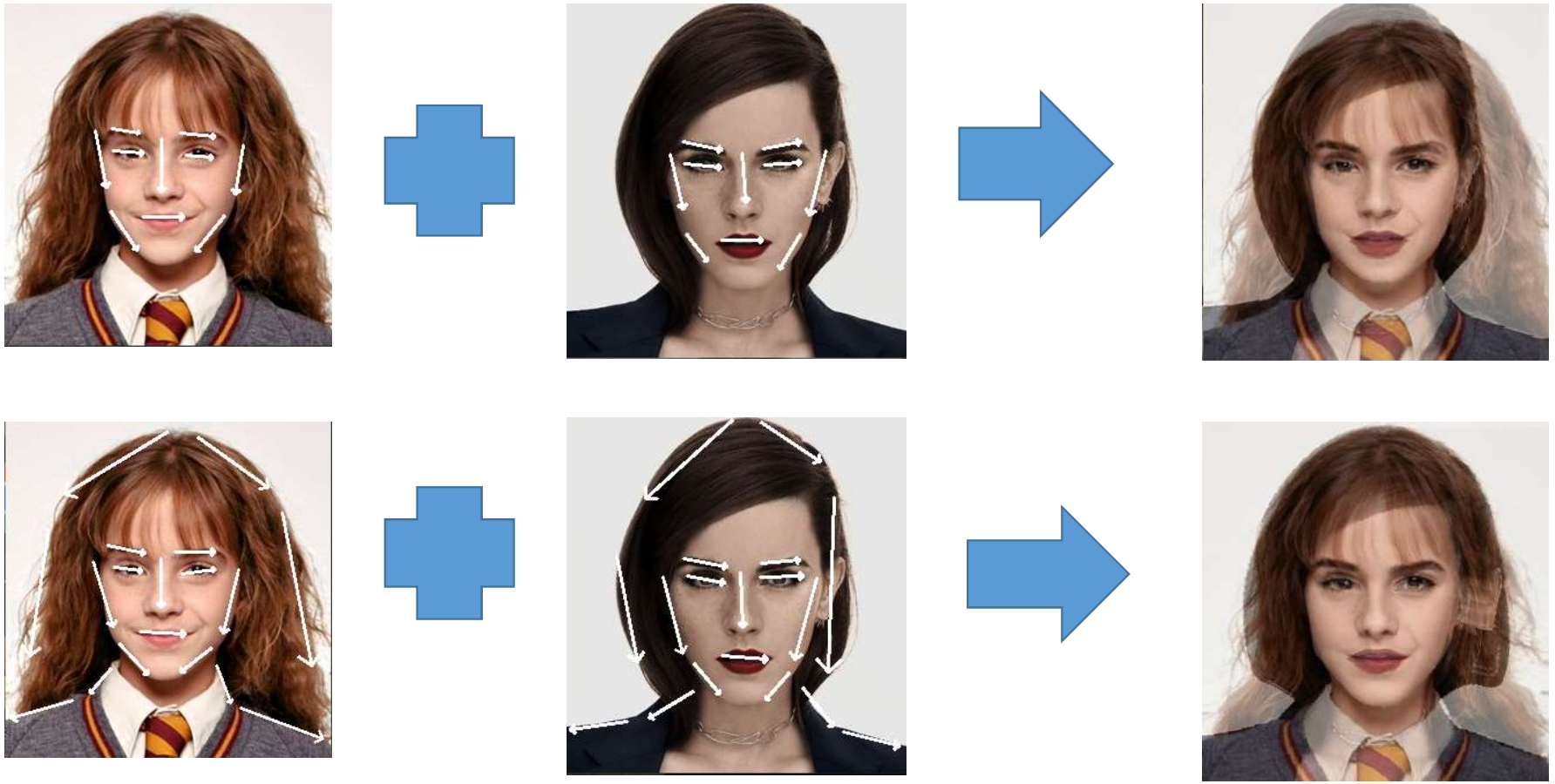
- 遇到的挑戰: UI 介面的產生

- 解決方式:

結果的分析與比較- Few lines



結果的分析與比較-Few lines vs more lines



比較模型:

- 1. Find Point Correspondences using Facial Feature Detection
- =>We can calculate the average of corresponding points in the two sets and obtain a single set of 80 points. On this set of average points we perform Delaunay Triangulation. we have triangle (or region) correspondences.
- 2. Delaunay Triangulation
- 3. Warping images and alpha blending
- =>Find location of feature points in morphed image
- =>Calculate affine transforms
- =>Warp triangles
- =>Alpha blend warped images

$$x_m = (1-\alpha)x_i + \alpha x_j$$
$$y_m = (1-\alpha)y_i + \alpha y_j$$

輸入資料類型

- 輸入:兩張image
- 輸出:warped image 和其轉換的動畫
- 資料集:使用dlib的shape_predictor_68_face_landmarks.dat。

delaunay triangulation-Randomized incremental algorithm

Idea

- insert p_1, then p_2 ... and finally p_n
- suppose we have computed $\mathcal{DT}(P_{i-1})$
- insert $p_i \Rightarrow$ splits a triangle into three
  - find this triangle using conflict lists
    - each non inserted point has a pointer to the triangle in $\mathcal{DT}(P_{i-1})$ that contains it
    - each triangle in $\mathcal{DT}(P_{i-1})$ is associated with the list of all the non–inserted points that it contains
- perform edge flips until no illegal edge remains
  - we only need to perform flips around p_i
  - on average, this step takes constant time
- we have just computed $\mathcal{DT}(P_i)$
- repeat the process until $i = n$

Pseudocode

Algorithm *Insert*(p)
Input: a point p, a set of point P and $\mathcal{T} = \mathcal{DT}(p)$
Output: $\mathcal{DT}(P \cup \{p\})$
1. Find the triangle abc of $\mathcal{DT}(P \cup \{p\})$ containing p
($*$ use reverse pointers from conflict lists $*$)
($*$ abc is chosen to be counterclockwise $*$)
2. Insert edges pa,pb and pc
($*$ it includes conflict lists updates $*$)
3. $SwapTest(ab)$
($*$ pseudocode of this procedure next slide $*$)
4. $SwapTest(bc)$
5. $SwapTest(ca)$

Pseudocode

Algorithm *SwapTest(ab)*
1. **if** ab is an edge of the exterior face
2. **do return**
3. $d \leftarrow$the vertex to the right of edge ab
4. **if** $\text{inCircle}(p, a, b, d) < 0$
5. **do** Flip edge ab for pd
($*$ it includes conflict lists update $*$)
6. $SwapTest(ad)$
7. $SwapTest(db)$

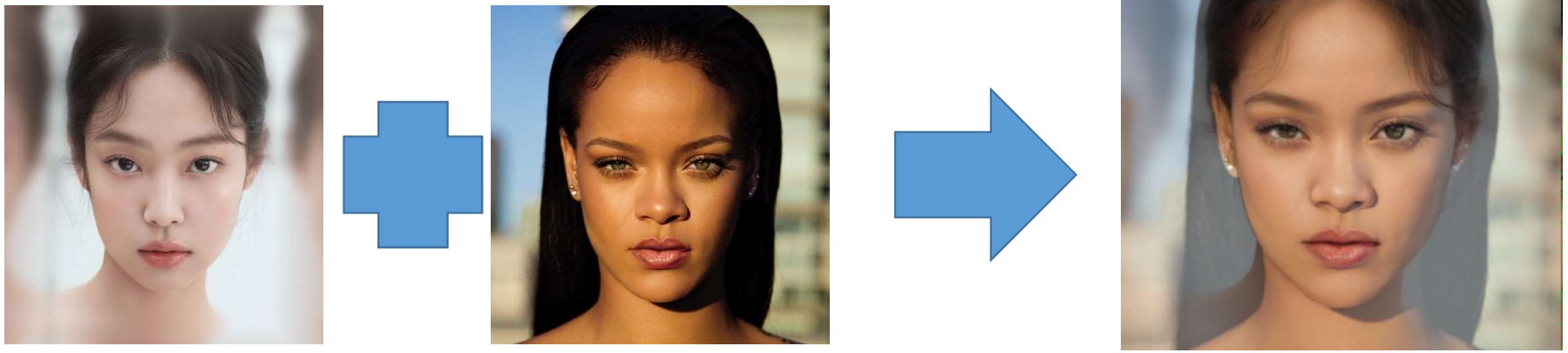
關鍵程式碼

```python
def detectFaceLandmarks(image):
    detector = dlib.get_frontal_face_detector()
    predictor = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')

    height = image.shape[0]
    width = image.shape[1]
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    faces = detector(gray)
    points = []
    for face in faces:
        landmarks = predictor(gray, face)
        for n in range(0, 68):
            x = landmarks.part(n).x
            y = landmarks.part(n).y
            points.append((int(x), int(y)))
    points.append((0, int(0.5 * float(height))))
    points.append((0, 0))
    points.append((int(0.5 * float(width)), 0))
    points.append((width - 1, 0))
    points.append((width - 1, int(0.5 * float(height))))
    points.append((width - 1, height - 1))
    points.append((int(0.5 * float(width)), height - 1))
    points.append((0, height - 1))
    return points
```

```python
def draw_voronoi(img, subdiv):
    (facets, centers) = subdiv.getVoronoiFacetList([])

    for i in range(0, len(facets)):
        ifacet_arr = []
        for f in facets[i]:
            ifacet_arr.append(f)

        ifacet = np.array(ifacet_arr, int)
        color = (random.randint(0, 255), random.randint(0, 255), random.randint(0, 255))

        cv2.fillConvexPoly(img, ifacet, color, cv2.LINE_AA, 0)
        ifacets = np.array([ifacet])
        cv2.polylines(img, ifacets, True, (0, 0, 0), 1, cv2.LINE_AA, 0)
        cv2.circle(img, (round(centers[i][0]), round(centers[i][1])), 1, (0, 0, 0), cv2.LINE_AA, 0)
```

```python
def delaunayTriangulation(img, subdiv, points, delaunay_color, doDraw):
    triangleList = subdiv.getTriangleList()
    size = img.shape
    r = (0, 0, size[1], size[0])
    indexesList = []
    for t in triangleList:

        pt1 = (round(t[0]), round(t[1]))
        pt2 = (round(t[2]), round(t[3]))
        pt3 = (round(t[4]), round(t[5]))
        id1 = points.index(pt1)
        id2 = points.index(pt2)
        id3 = points.index(pt3)

        if rect_contains(r, pt1) and rect_contains(r, pt2) and rect_contains(r, pt3):
            if doDraw:
                cv2.line(img, pt1, pt2, delaunay_color, 1, cv2.LINE_AA, 0)
                cv2.line(img, pt2, pt3, delaunay_color, 1, cv2.LINE_AA, 0)
                cv2.line(img, pt3, pt1, delaunay_color, 1, cv2.LINE_AA, 0)
            indexesList.append((id1, id2, id3))
    return indexesList
```

結果的分析與比較-model2



結論-Warped Line v.s Delaunay Triangulation

Warped Line

Delaunay Triangulation

Future work

- 預期改進的方向:針對不同臉型或角度的人臉，也可以找出方法去做morphing
- 多研究一些近期發表的模型做實作

Reference

- https://learnopencv.com/face-morph-using-opencv-cpp-python/
- https://www.youtube.com/watch?v=rZcdhles6vQ