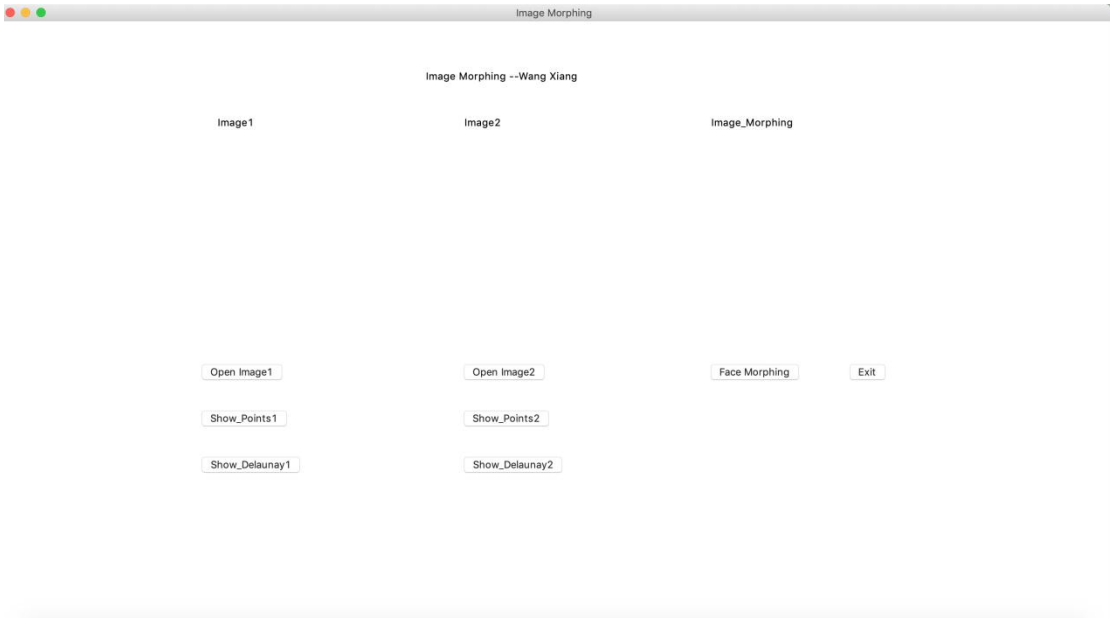
In this Project, I implemented the image morphing by reading key points in the face,Using Delaunay's triangulation to realize face mesh,finally implement face fusion. I designed the GUI,through the GUI, I can upload two face photos, realize the morphing function, and display the key points of the face and the Delaunay's triangulation. It should be noted that I spent most of my time coding image morphing functions. Finally, there are some problems with the GUI. So I wrote the Delaunay's triangulation Delaunay_Practice.py file and the key points Face_feature.py file separated . They all run perfectly. You only need to modify the path of the photo to be operated in the file to run normally. And I described some method through coding comment step by step, which can clearly demonstrate my idea. According to the comment (Begin with # signal), it can let the readers know my ideas easily.

Main Idea

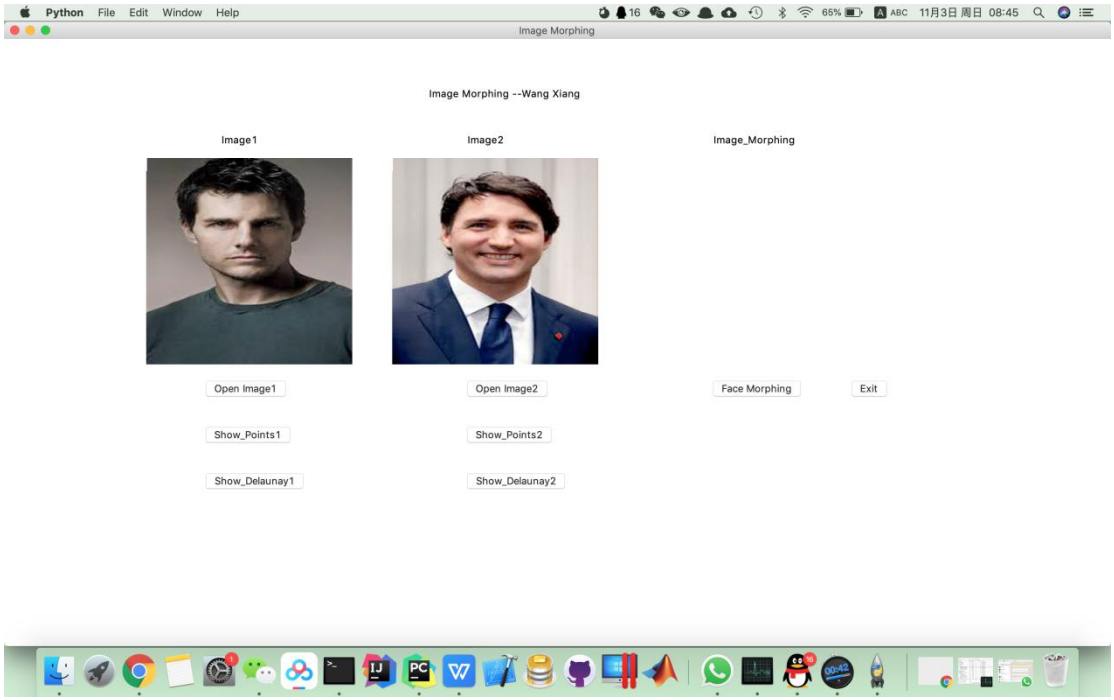
Open image1:tom.jpeg image2:Trudeau.jpeg and then create morphing image

I used Delaunay triangulation, cv2.getAffineTransform method and cv2.warpAffine ,First use the shape_predictor_68_face_landmarks.dat model for 68-point calibration, but the effect is not good, you need to add an additional 8 points, which are the four vertices of the picture and the midpoint of the four sides, and then save the key points to their respective In the CSV file, then use Delaunay triangulation on the feature points, join the point sets into triangles of a certain size, then get the coordinates of the triangle points in each picture, then enter the morph_triangle triangle for triangle transformation, alpha blending, Find a minimum bounding rectangle for each triangle, because the latter operation is warpAffine to operate on an image, it can only operate on rectangles, so I have to calculate the minimum bounding rectangle of each triangle before. Then, the enclosing rectangle is warped by warpAffine, and then the mask operation is used to remove the area other than the triangle. Then take another point again, form a triangle, do the above again, and finally generate the merged point and then generate the merged photo.

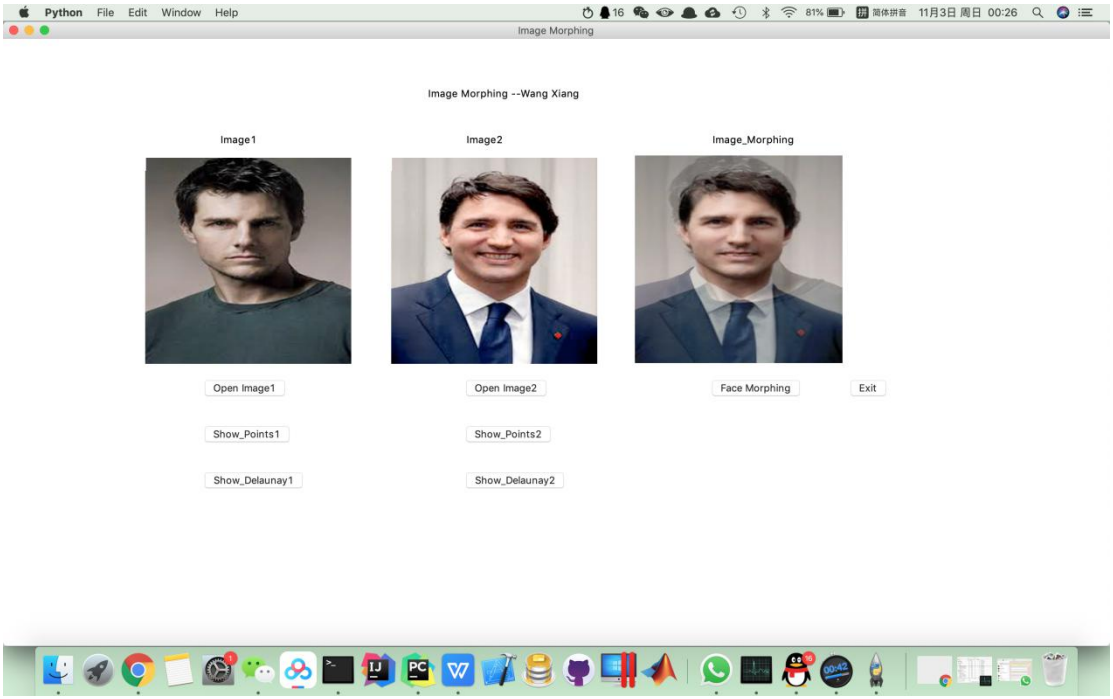
First, I designed a GUI interface. In this interface,Clicking these buttons will upload image, display the image, display Face morphing image and show each image points and delaunay triangulation.



Second I upload two image, and then click face morphing button



Thrid, when click face morphing button,the result can show and methods which i show below



In this function I read the image and resize the size of image,and then go to get_points function to get face key points,and store points into CSV file. and then go to Delaunay triangulation.Use the coordinates of the triangle in each image to calculate Delaunay triangulation

```
def morph_faces(filename1, filename2, alpha=0.5):    # Fusion picture

    img1 = cv2.imread(filename1)
    img2 = cv2.imread(filename2)
    # scr:original image    dsize：the size of output image(img1.shape[1],img1.shape[0])
interpolation：the method of interpolation
    img2 =
cv2.resize(img2,(img1.shape[1],img1.shape[0]),interpolation=cv2.INTER_CUBIC)
    print('img1.shape',img1.shape)
    print('img2.shape',img2.shape)

    # Get the face feature points by the custom get_points method,
    # including the 68 feature key coordinates of the face collected by pointes

    points1 = get_points1(img1)

    # Read in the points from a text file

    #print('pionts1:',len(points1),points1)
    points2 = get_points2(img2)
```

```
    # with open("image2.txt") as file:
    #         for line in file:
    #                 x, y = line.split()
    #                 points2.append((int(x), int(y)))
    #
    #print('pionts2:', len(points2), points2)


    points = (1 - alpha) * np.array(points1) + alpha * np.array(points2)

    p = pd.DataFrame(points)

    with open("image_morph.txt") as file:
        for line in file:
            x, y = line.split()
            p.append((int(x), int(y)))



    p.to_csv('./1.csv',index=False)

    img1 = np.float32(img1)
    img2 = np.float32(img2)
    img_morphed = np.zeros(img1.shape, dtype=img1.dtype)

    # Use Delaunay triangulation on feature points to join point sets into triangles of a
certain size,
    # and the allocation is relatively reasonable in order to present a beautiful
triangulation

    triangles = get_triangles(points)
    for i in triangles:
        x = i[0]
        y = i[1]
        z = i[2]

        tri1 = [points1[x], points1[y], points1[z]]
        tri2 = [points2[x], points2[y], points2[z]]
        tri = [points[x], points[y], points[z]]
        morph_triangle(img1, img2, img_morphed, tri1, tri2, tri, alpha)

    return np.uint8(img_morphed)
```

Get the face points
68-point calibration using Dlib officially trained model
"shape_predictor_68_face_landmarks.dat"
Then use OpenCv for image processing and draw 68 points on the person's face.
#import model
dlib key point detection model (68)

predictor_model = 'shape_predictor_68_face_landmarks.dat'

# Use dlib to get the feature points of the face
def get_points1(image):

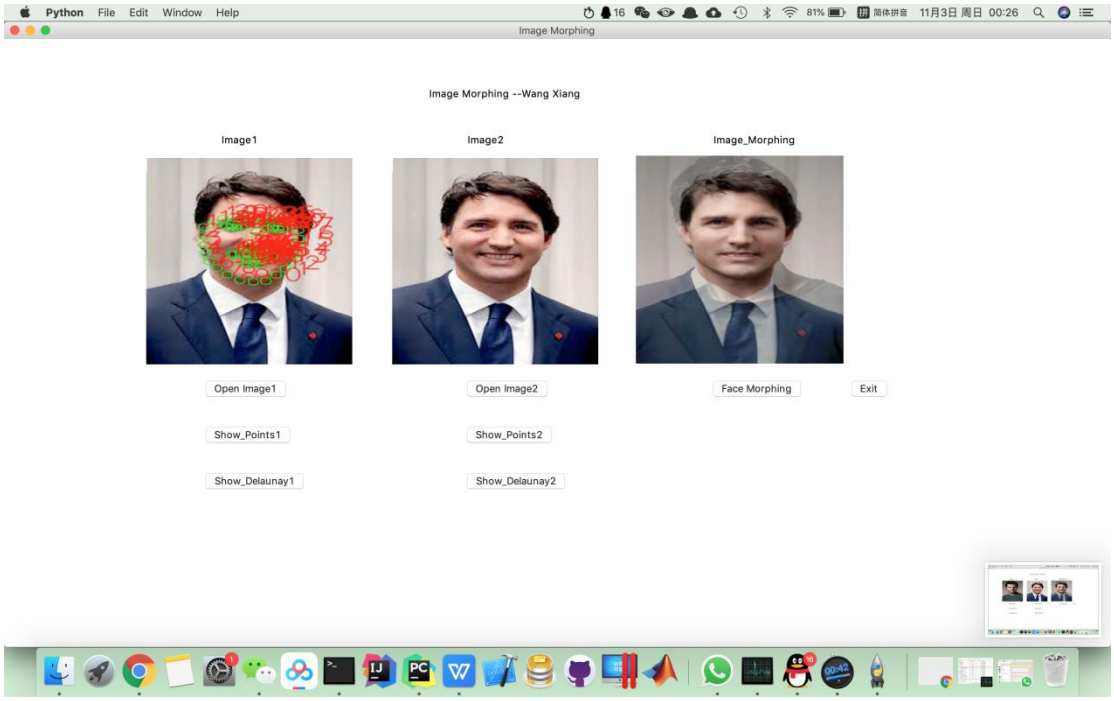
    # Same as face detection, using frontal_face_detector that comes with dlib as a face detector
    face_detector = dlib.get_frontal_face_detector()   #for face detection, extracting the outer rectangle of the face
    # Keypoint extraction requires a feature extractor, and the feature extractor can be used to train the model.
    # #Build a feature extractor using the officially provided model
    face_pose_predictor = dlib.shape_predictor(predictor_model)
    try:
        detected_face = face_detector(image, 1)[0]
    except:
        print('No face detected in image {}'.format(image))
    pose_landmarks = face_pose_predictor(image, detected_face)   # get landmark
    points = []

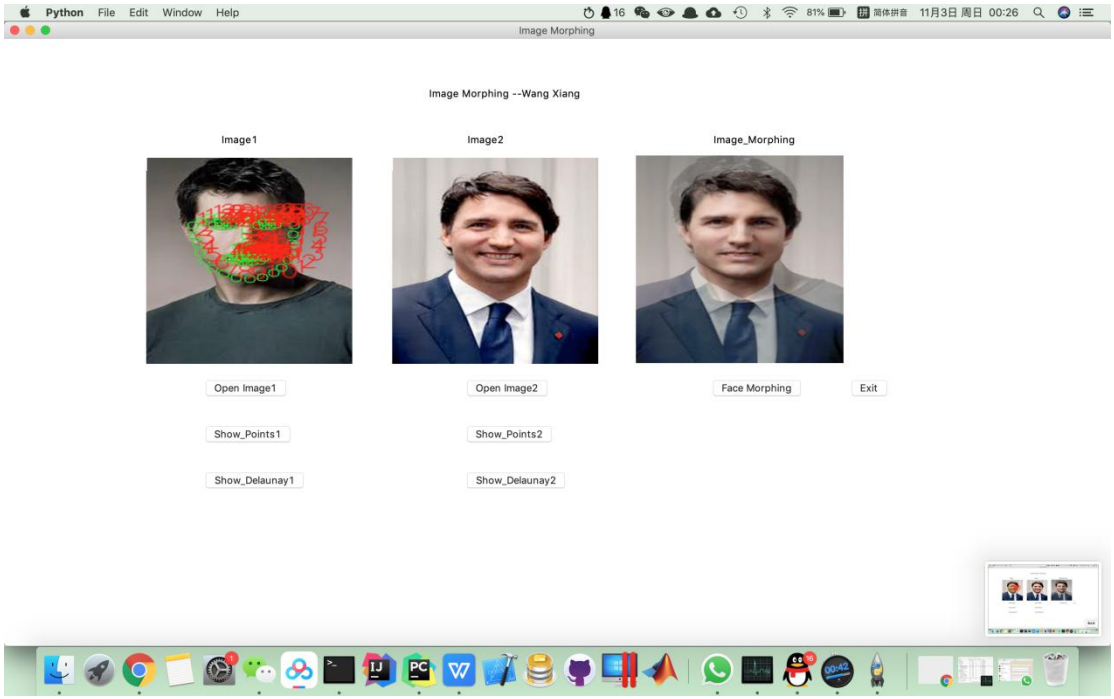
    # points is the coordinates of the 68 key points of the collected face
    for p in pose_landmarks.parts():
        points.append([p.x, p.y])
        p1 = pd.DataFrame(points)
        p1.to_csv('./image1.csv',index=False)

    a=1

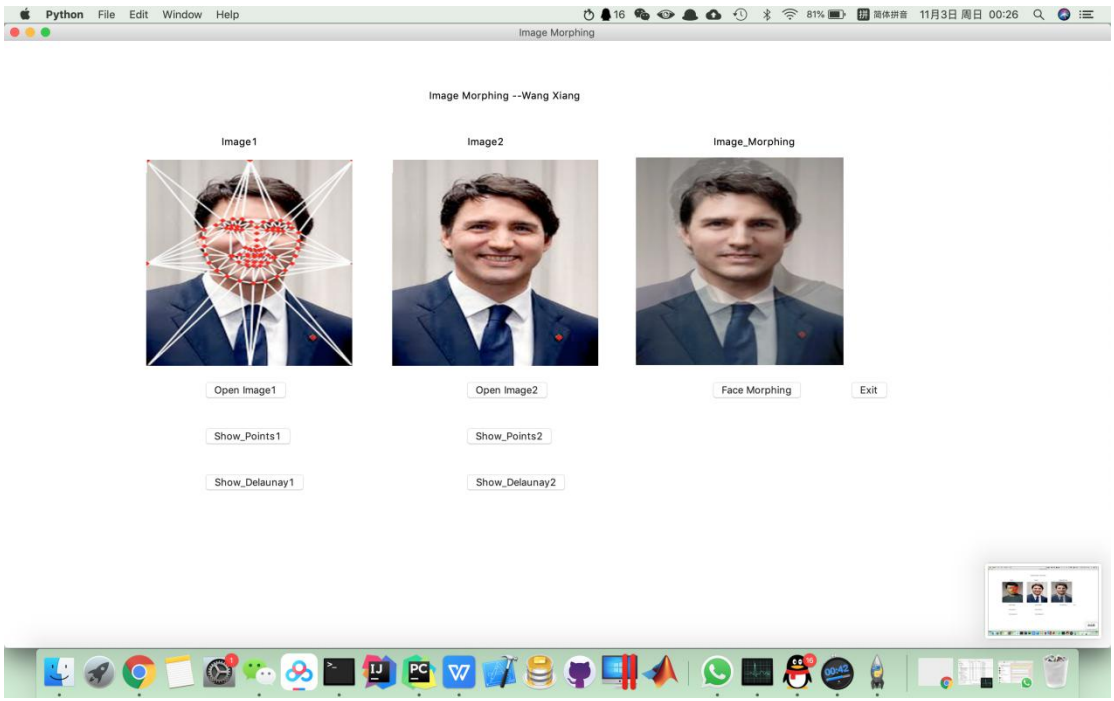
    # Add four vertices and the midpoint of the four sides
    # 1:kweight 0 width
    x = image.shape[1] - 1
    y = image.shape[0] - 1
    points.append([0, 0])
    points.append([x // 2, 0])
    points.append([x, 0])

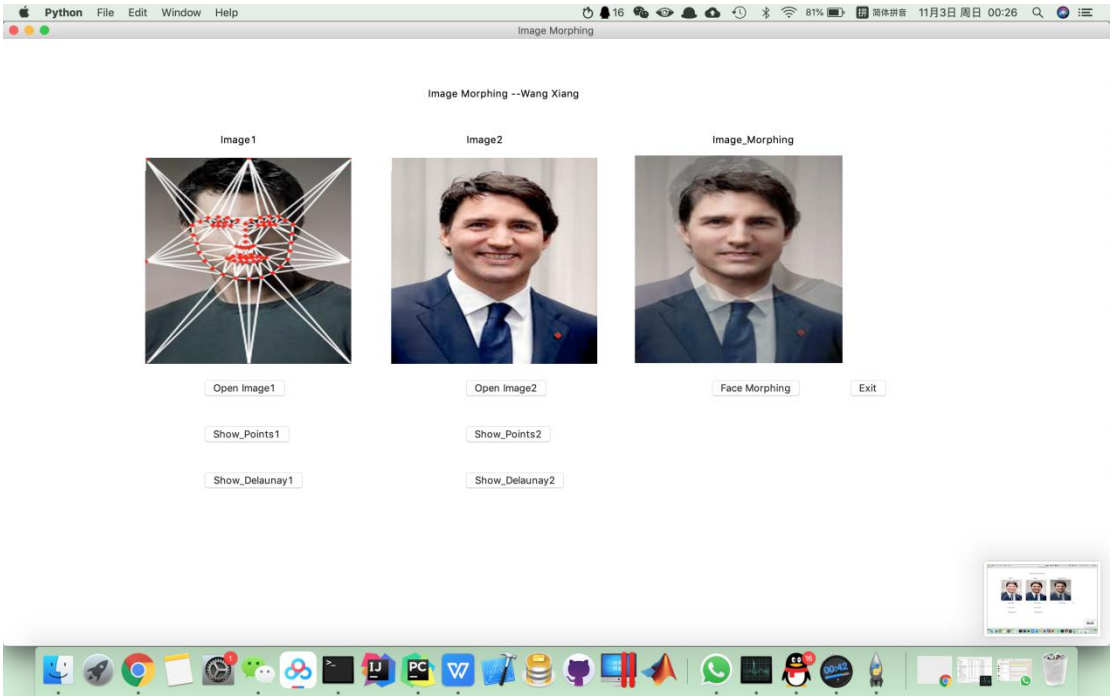
```
points.append([x, y // 2])
points.append([x, y])
points.append([x // 2, y])
points.append([0, y])
points.append([0, y // 2])


p1 = pd.DataFrame(points)
p1.to_csv('./image1.csv',index=False)

return np.array(points)
```





# Use Delaunay triangulation on feature points to join point sets into triangles of a certain size,
# and the allocation is relatively reasonable in order to present a beautiful triangulation
def get_triangles(points):
    p = Delaunay(points)
    return p.simplices






```
def affine_transform(input_image, input_triangle, output_triangle, size):    # affine transformation of the face to determine the position

    # This function uses three pairs of points to calculate the affine transformation, which is mainly used to generate the affine transformation matrix.
    # First parameter: the triangle vertex coordinates of the input image
    # second parameter: the corresponding triangle vertex coordinates of the output image



    warp_matrix = cv2.getAffineTransform(
        np.float32(input_triangle), np.float32(output_triangle))



    # affine transformation of images
    # First: Input image Second parameter: Output image
    # Third parameter: 2×3 transformation matrix
    # Output image size
    # flags=Bilinear interpolation (default method)
    # boundary processing method
```

```
    output_image = cv2.warpAffine(input_image, warp_matrix, (size[0], size[1]), None,
                                  flags=cv2.INTER_LINEAR,
borderMode=cv2.BORDER_REFLECT_101)
    return output_image



def morph_triangle(img1, img2, img, tri1, tri2, tri, alpha):    # Triangle deformation, Alpha
blend


# Calculate the bounding box of the triangle Calculate the minimum rectangle of the
vertical boundary of the outline,
# which is parallel to the upper and lower boundaries of the image
    rect1 = cv2.boundingRect(np.float32([tri1]))    # Find the coordinates of the upper left
corner of tri1, and the length and width of tri1

    rect2 = cv2.boundingRect(np.float32([tri2]))
    rect = cv2.boundingRect(np.float32([tri]))

    tri_rect1 = []
    tri_rect2 = []
    tri_rect_warped = []

    for i in range(0, 3):
        tri_rect_warped.append(
            ((tri[i][0] - rect[0]), (tri[i][1] - rect[1])))
        tri_rect1.append(
            ((tri1[i][0] - rect1[0]), (tri1[i][1] - rect1[1])))
        tri_rect2.append(
            ((tri2[i][0] - rect2[0]), (tri2[i][1] - rect2[1])))

    #   Affine transformation in the bounding box
    img1_rect = img1[rect1[1]:rect1[1] +
                     rect1[3], rect1[0]:rect1[0] + rect1[2]]
    img2_rect = img2[rect2[1]:rect2[1] +
                     rect2[3], rect2[0]:rect2[0] + rect2[2]]

    size = (rect[2], rect[3])
#img1_rect is input_image and tri_rect1 is input_triangle
# and tri_rect_warped is output_triangle
    warped_img1 = affine_transform(
        img1_rect, tri_rect1, tri_rect_warped, size)
```

```
        warped_img2 = affine_transform(
            img2_rect, tri_rect2, tri_rect_warped, size)


        #weighted summation
        #Having a key point is equivalent to having two faces of data.
        # Next we will focus on these key points,
        # The formula code for the fusion is as follows:

        # points = (1 - alpha) * np.array(points1) + alpha * np.array(points2)
        img_rect = (1.0 - alpha) * warped_img1 + alpha * warped_img2


        #function fillConvexPoly fills the interior of the convex polygon
        # first parameter: image template second parameter: pointer array to a single
polygon
        # Third parameter: the vertex of the convex polygon.
        cv2.fillConvexPoly(mask, np.int32(tri_rect_warped), (1.0, 1.0, 1.0), 16, 0)

        # Application template
        img[rect[1]:rect[1] + rect[3], rect[0]:rect[0] + rect[2]] = \
            img[rect[1]:rect[1] + rect[3], rect[0]:rect[0] +
                rect[2]] * (1 - mask) + img_rect * mask
```