

Guido van Rossum

Python is an object-oriented, multi-paradigm and cross-platform programming language. It promotes structured, functional and object-oriented imperative programming. It is equipped with strong dynamic typing, automatic garbage memory management and an exception management system.

Digital Imaging Project


Face Swapping



ADRIEN Eunice
MAZARD Quentin



Table of contents

 **Presentation of the project**

 **Objectives to solve**

 **Description**

 **Results**

 **Discussions : what could be done further ?**

Presentation of the project

As part of our courses on digital imaging, we have a project to carry out in order to test and improve our knowledge.

The project, realized in pairs, is a very current project since we chose the face swapping. Indeed, this photo effect is very appreciated by users, especially on facebook, snapchat ...

But what is the face swapping?

This effect allows you to interchange two faces on a photo.

Objectives to solve

The project, quite complex, required a division into several small parts so as not to get lost. Initially, it was necessary to be able to determine the two faces. It is important to define key points such as the eyes, the mouth or the nose.

We used the Dlib library which is a very useful tool. This library, written in C ++, contains complex machine learning algorithms to solve many "real world" problems. What we mean by real problem is, in our case studies, the detection of objects in an image. And more particularly of face for us.

So we have two major problems to solve:

- Detecting the face
- Exchange them

Description

To begin with, in the swapping face, we decided to interchange two faces.

At first, we will transform the images into matrix of 68x2 thanks to "prediction" witch uses the file shape_predictor from the dlib library.

```
def get_matrix(img):  
    array = detection(img,1)  
    return np.matrix([[p.x, p.y] for p in prediction(img, array[0]).parts()])
```

We first define the different key points of the face.

```
points_of_the_contour = list(range(17, 68))  
  
points_of_the_lbrow = list(range(22, 27))  
points_of_the_rbrow = list(range(17, 22))  
  
points_of_the_leye = list(range(42, 48))  
points_of_the_reye = list(range(36, 42))  
  
points_of_the_noise = list(range(27, 35))  
  
points_of_the_mouth = list(range(48, 61))
```

Once the two faces are selected with the we detect the contours with the draw_convex_hull tool.

```
def convex_hull(img, marks, color):  
    marks = cv2.convexHull(marks)  
    cv2.fillConvexPoly(img, marks, color=color)
```

Once this step is done, the program allows the automatic location of 3D markers:

Once the markers are placed, we draw the contours in order to obtain a sort of transposable pattern. To do this, we define the function convex_hull (img, marks, color): which will take into account the previously found markers in place_for_the_mask.

```
def place_for_the_mask(img, img_marks):  
    img = np.zeros(img.shape[:2], dtype=np.float64)  
  
    for group in array_of_the_points:  
        convex_hull(img, img_marks[group], color=1)  
    img = np.array([img, img, img]).transpose((1, 2, 0))  
    img = (cv2.GaussianBlur(img, (11, 11), 0) > 0) * 1.0  
    img = cv2.GaussianBlur(img, (11,11), 0)  
    return img
```

Once the face is trimmed, we "untie" it and transpose it on the second face.



To do this, it is important to compare the marks of the two faces in order to match them without error to misplace the face.

The warp function allows to map an image into an other.

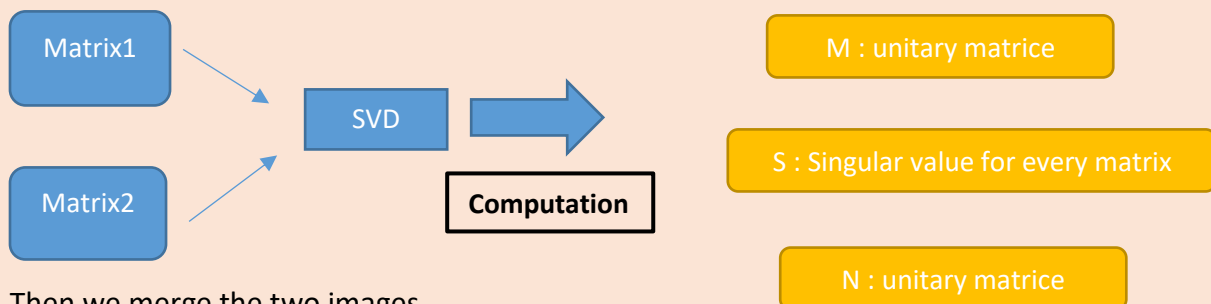
```
def warp(modification, dshape, img):
    newimg = np.zeros(dshape, dtype=img.dtype)
    cv2.warpAffine(img, modification[:2], (dshape[1], dshape[0]), dst=newimg, borderMode=cv2.BORDER_REPLICATE)
    return newimg
```

To do this, we use a function `contouring(points_of_img1, points_of_img2)`: which will have the good alignment from the face of the first image on the second image.

```
def contouring(points_of_img1, points_of_img2):
```

That is why we use the numpy library and the tool "linalg". SVD is the Singular Value Decomposition. It allows to make rotation and translation to have the good alignment.

```
M, S, N = np.linalg.svd(points_of_img1.T * points_of_img2)
```



Then we merge the two images

Then we do the same thing for the second face to finally leave an image with interchanged faces.

The function main allows to have the two images and give the final result.

```
def main():
    img1 = cv2.imread(sys.argv[1], cv2.IMREAD_COLOR)
    img1 = cv2.resize(img1, (img1.shape[1]*1, img1.shape[0]*1))
    out1 = get(img1)

    img1_marks = out1;

    img2 = cv2.imread(sys.argv[2], cv2.IMREAD_COLOR)
    img2 = cv2.resize(img2, (img2.shape[1]*1, img2.shape[0]*1))
    out2 = get(img2)

    img2_marks = out2;

    modification = contouring(img1_marks[list_of_the_points], img2_marks[list_of_the_points])
    place_where_paste = place_for_the_mask(img2, img2_marks)
    face_to_copy = warp(modification, img1.shape, place_where_paste)
    mix = np.max([place_for_the_mask(img1, img1_marks), face_to_copy ], axis=0)
    warp_img2 = warp(modification, img1.shape, img2)

    finalresult = img1*(1.0-mix) + warp_img2*mix

    cv2.imwrite('result.jpg', finalresult)
```

Results

The results we hoped are confirmed, as we can see below.



Discussions : what could be done further ?

The application could probably be better optimized and could know some very appreciable variants also such as a face swapping on a video stream that would allow to constantly exchange the faces.