**FACE SWAP DELAUNAY**

***Step 1:*** Find landmork points of both images

Use the pre-trained models of dlib or mediapipe library to detect the facial landmark points on source and destination image. In the example, I use model shape\_predictor\_68\_face\_landmarks.dat to illustrate those point

|  |  |
| --- | --- |
|  |  |
| *Landmark point in source image* | *Landmark point in destination image* |

***Step 2:*** Triangulation source and destination image

The reason why we need to divide the face into triangles is that we can not just cut out the face from the source image and put it into the destination image as they have different size and perspective.

Also we can not change it is size and perspective right away because the face would lose the original proportions.

Instead if we split the face into Delaunay triangles, we can simply swap each triangle and in this way it will keep the proportions and also it will match the expressions of the new face, like for example if you smile, close eyes or open the mouth.

\*Requirements on destination image

The triangulation of the destination image needs to have the same patterns of the triangulation of the source image. That means that the connection of the points has to be the same.

So after we do the triangulation of the source image, from that triangulation we take the indexes of the landmark points so that we can replicate the same triangulation on the destination image.

|  |  |
| --- | --- |
|  |  |
| *Delaunay triangles in source image* | *Delaunay triangles in destination image* |

***Step 3:*** Extract and warp triangles

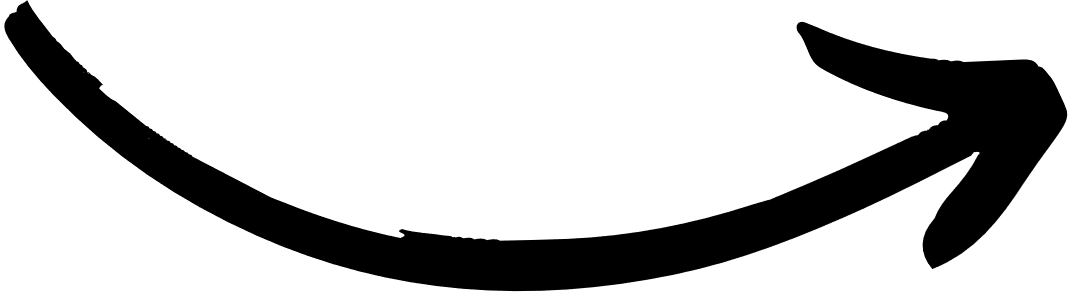
Once we have the triangulation of both faces we take the triangles of the source face and we extract them.

We also need to take the coordinates of the triangles of the destination face, so that we can warp the triangles of the source face to have same size and perspective of the matching triangle on the destination face. This step uses Affine Transformation to warp those triangles.

In Open CV library, it supports Affine Transformation through two function

M = cv2.getAffineTransform (origin\_triangle\_points, target\_triangle\_points)

* Input: Three pairs of corresponding points
* Output: The transformation matrix



warped\_triangle = cv2.warpAffine (origin\_cropped\_rectangle, M, (w,h))

* Input: The transformation matrix
* Output: An image

This function multiply each pixel in the source triangle by that matrix to get the triangle destination.

***Step 4:*** Link the warped triangles together

Once we have cutted and warped all the triangles we need to link them together.

We simply rebuild the face using the triangulation pattern, with the only difference that this time we put the warped triangle.

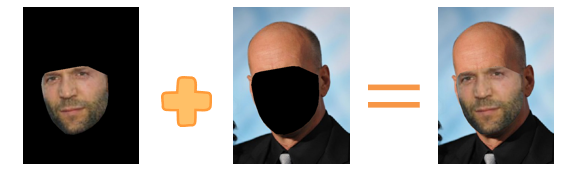
|  |  |
| --- | --- |
|  |  |

After warp each triangle together, we will get the face of the source image as the face of the destination image without changing the original features of the source image such as eyes, nose and mouth.

***Step 5:*** Replace the face on the destination image

The face is now ready to be replaced. We cut out the face of the destination image to make space for the new face.

So we take the new face, and the destination image without face and we link them together.



***Step 6:*** Seamless Cloning

It is easy to see that the skin color of these two people has a big difference, so we need to adjust the skin color so that the final image looks the most natural.

On Opencv we have a built in function called “ cv2.seamlessClone ” that does this operation automatically.

