Project 2: Face Swap

Used 1 late day, Aadhya Puttur, Alex Chiluisa

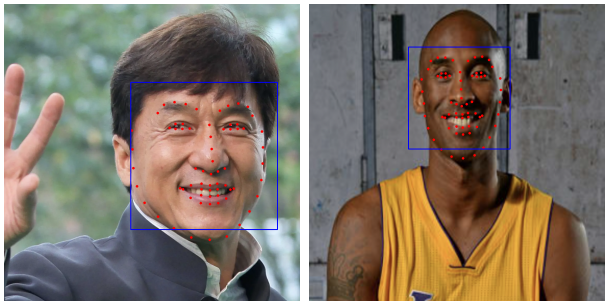
I. Introduction

In this project our goal is to swap faces and warp a face to another. In phase 1 we follow 2 traditional approaches on face swap. The first is a Delaunay Triangulation and the second is using thin plate splines. Then in phase 2 we use a deep learning pipeline to swap faces.

II. Phase1: Traditional Approach

III. Delauney Triangulation

A) Facial Landmark Detection The traditional approach is to first find facial landmarks. Facial landmarks are the detection of important characteristic points on the face. The purpose of this is so we can have a one-to-one correspondence between facial landmarks with two faces. To get landmarks we used the dlib library. In Figure 1 below you will see the landmarks of Kobe Bryant and Jackie Chan along with the bounding box of the face. We obtained 68 landmarks for both images. (0-17) represented the jaw ear to ear, (17-22) represented the left eyebrow, (22-27) represented the right eyebrow, (27-31) represented the line on top of the nose, (36-42) represented the left eye, (42-48) represented the right eye, (48-60) represented the outer part of the lips, (60-68) represented the inner part of the lips. We are planning to warm image (b) to image (a) [Fig 1]. *B) Triangulation Correspondence and Warping* After we



(a) (b)

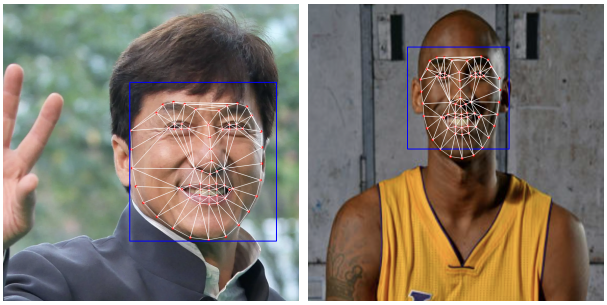
Fig. 1. Facial landmarks and bounding box using dlib with OpenCV

obtained the landmarks, ideally we needed to warp the faces in 3D, although we don't have 3D information to do that. We triangulate using facial landmarks by making assumptions about the 2D image to approximate 3D information. We assume that in each triangle the content forms a plane in 3D. Now we needed to start preparing the images for warping by finding the corresponding features. We do this by using Delaunay Triangulation, which is the process of connecting every two neighboring sites in the Voronoi diagram. We are using this method because it is constructed in O(N log N) time. In order to avoid parts of the face fading into the background, using Delaunay triangulation we can maximize the smallest angle in each triangle.

Step 1: We obtained Delauney triangulation of an image using Subdiv2D and keep track of the landmark points that made each triangle

Step 2: We use the same landmark points that made each triangle to make each triangle on the other image

From these steps above we produce the images below in figure 2 for Delaunay Triangulation After we have found



(a) (b)

Fig. 2. Delaunay Triangulation, finding corresponding points for triangles

triangulation for both images, we want to begin warping Kobe's face to Jackie's face or the destination image to the source face. For this method, we use inverse warping in order to prevent holes. The reason for this is because image interpolation works in two directions, and tries to achieve the best approximation of a pixel's color intensity based on the values of surrounding pixels. There are many different types of interpolation algorithms, although for this we are using spline interpolation. Spline interpolation takes more surrounding pixels into consideration, more than bilinear interpolation. We are using spline because we retain most image information after an interpolation. We are using this method because of the complexity of our problem where there are multiple distortion steps, therefore it is computationally intensive. In Figure 3, you can see the difference between having interpolation and having no interpolation.

The steps necessary to warp Kobe's face into Jackie's are as follows:

A. Chiluisa is with the Department of Robotics Engineering, Worcester Polytechnic Institute, Worcester, MA 01609, USA (e-mail: ajchiluisa@wpi.edu)

A. Puttur is with the Department of Computer Science, Worcester Polytechnic Institute, Worcester, MA 01609, USA (e-mail: aputtur@wpi.edu)

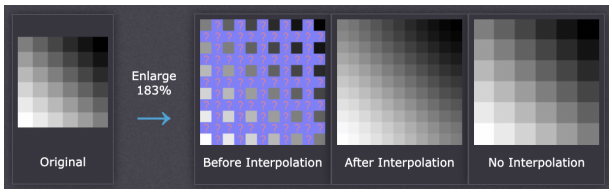


Fig. 3. Deep Image Homography estimation Network Architecture

4) For each triangle in the destination face $\mathcal{B}$, we would compute the Barycentric coordinate, specifically the α, β, γ for each point in each triangle.

$$\begin{bmatrix} \mathcal{B}_{a,x} & \mathcal{B}_{b,x} & \mathcal{B}_{c,x} \\ \mathcal{B}_{a,y} & \mathcal{B}_{b,y} & \mathcal{B}_{c,y} \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Fig. 4. Barycentric Coordinate

The barycentric coordinate is given by these three values. The Matrix on the left-hand side is the coordinates of the 3 corners of each triangle. For each triangle, we will find the α, β, γ for each point in each triangle. Now that we know the corresponding triangles for both images, we need to find the corresponding coordinate points in the area of each triangle. As such, the x and y on the right-hand side of figure 4 represent the x and y values of a point in a triangle represented by the matrix on the right-hand side. Although, firstly we need to compute the α, β, γ, so we will change this equation slightly in order to do that.

$$\begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \mathcal{B}_{\Delta}^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Fig. 5. Barycentric Coordinate compute value of $[\alpha, \beta, \gamma]$

5) Check if the coordinate belongs in a specific triangle

Given the α, β, γ, we can say a specific point x lies inside the triangle if $\alpha \in [0,1]$, $\beta \in [0,1]$, $\gamma \in [0,1]$, and $\alpha + \beta + \gamma \in (0,1]$. This is how we will check if a coordinate belongs in the triangle. However, iterating through a triangle to do this is not really impossible. Instead, we found bounding boxes of each triangle where we were only interested in the coordinate points in the triangle, anything outside the triangle we ignored.

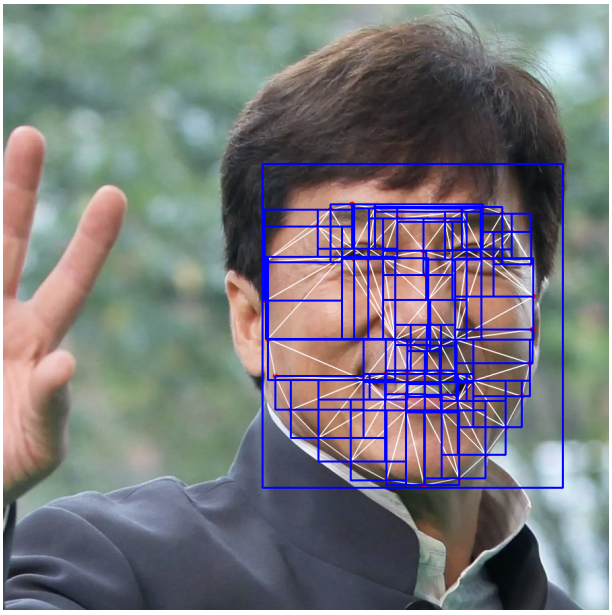


Fig. 6. bounding boxes over every triangle on image

Once we know what α, β, γ are for each coordinate point and for each triangle we can start finding the coordinate points for the other image.

6) User the barycentric equation to compute the corresponding coordinate points (x, y) for each triangle in the destination image $\mathcal{A}$ (Kobe's Face).

$$\begin{bmatrix} x_{\mathcal{A}} \\ y_{\mathcal{A}} \\ z_{\mathcal{A}} \end{bmatrix} = \mathcal{A}_{\Delta} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} \quad \mathcal{A}_{\Delta} = \begin{bmatrix} \mathcal{A}_{a,x} & \mathcal{A}_{b,x} & \mathcal{A}_{c,x} \\ \mathcal{A}_{a,y} & \mathcal{A}_{b,y} & \mathcal{A}_{c,y} \\ 1 & 1 & 1 \end{bmatrix}$$

(a) (b)

$$x_{\mathcal{A}} = \frac{x_{\mathcal{A}}}{z_{\mathcal{A}}} \text{ and } y_{\mathcal{A}} = \frac{y_{\mathcal{A}}}{z_{\mathcal{A}}}$$

(c)

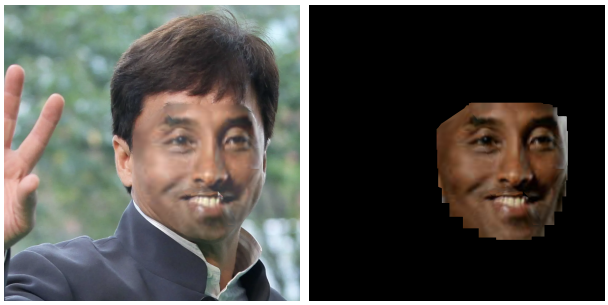
Fig. 7. Computing corresponding pixel coordinates for image $\mathcal{A}$ (destination image)

Same as Figure 4, the $\mathcal{A}$ matrix represents the 3 corners of the corresponding triangles from $\mathcal{B}$[Figure 7(b)]. Now we are using the α, β, γ computed from Figure 5 to compute the $x, y, andz$ coordinates for the destination image [Figure 7(a)]. The coordinate points will have to be in the corresponding triangle in Figure 7b because of the range of α, β, γ numbers we accepted. We then divide the x and y values by the z value, since we only care about the x and y. Therefore, we convert the values to homogeneous coordinates [Figure 7(a)].

7) Lastly, we need to copy back the value of the pixel

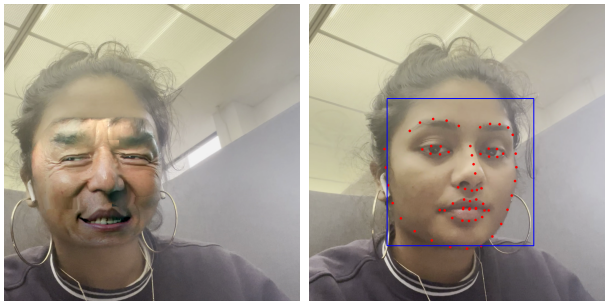
at (x_A, y_A) to the target/destination location using scipy.interpolate.interp2D.

"scipy.interpolate.interp2D" performs a spline interpolation as we discussed previously. With this function, we are able to get the color gradient values for each pixel. We set everything outside of our destination face to black (0,0,0) and we get a warped image. We then take this warped image to place it on the face of the source image (Jackie Chan).



(a) (b)

Fig. 8. Face Swap for the source and destination images to create Kobe Chan



(a) (b)

Fig. 9. Video capture of face swap with Jackie Chan

IV. Face Warping using Thin Plate Spline

Since human faces have complex and smooth shapes, a better alternative for transformation is by using Thin Plate Spline (TPS). TPS allows the mapping of pixel coordinates from the source image to the destination images by following the eq. 1.

$$f(x,y) = a_1 + (a_x)x + (a_y)y + \sum_{i=1}^{p} w_i U(\|(x_i, y_i) - (x,y)\|_1) \quad (1)$$

Where, $U(r) = r^2 log(r^2)$, x and y are the pixel coordinates of the source image and x_i and y_i are the the face features. The goal is to find the matrix of weight (a_1, a_x, a_y, w_i). Following the method processed described on [1], [2], we can find the position of the pixels in the source images using eq. 1 for $f_{x'}(x,y)$ and $f_{y'}(x,y)$. To wrap the images we first detected the landmarks in each face using dlib as we did in triangulation, and get the boxes around each face, crop them, and got their corresponding shifted landmarks. Subsequently, we applied the TPS method to obtain the $f_{x'}(x,y)$ and $f_{y'}(x,y)$ and wrap the image source. Following the process on [3] we warped the cropped source image on the destination image. Finally, we blend the images using the Poisson Image Editing [4]. The results are shown in fig. **??**

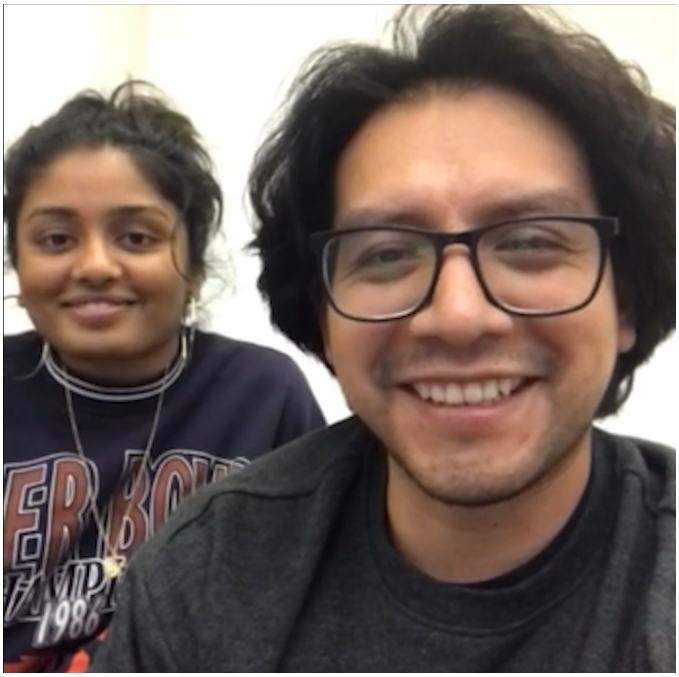


Fig. 10. originalimage

Fig. 11. TPS image

V. Phase2

The PRNet was not working with our computer because of an installation issue. Although, looking at results of how it would compare to our images. We found that dlib had an issue finding our face with face detection when we would

rotate our face or a landmark couldn't be seen. Although with the PRNet you can see the other side of the face when it is rotated and it is able to produce a 3D mask on it.

Fig. 12. Landmarks using PNCC render

REFERENCES

[1] https://khanhha.github.io/posts/Thin-Plate-Splines-Warping/.

[2] F. L. Bookstein, "Principal warps: Thin-plate splines and the decomposition of deformations," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 11, no. 6, pp. 567–585, 1989.

[3] https://github.com/h-gokul/FaceSwap/.

[4] P. Pérez, M. Gangnet, and A. Blake, "Poisson image editing," in *ACM SIGGRAPH 2003 Papers*, 2003, pp. 313–318.