Project 2: FaceSwap

Yu Shen
M.S. Computer Science
University of Maryland
College Park, MD
Email: yushen@cs.umd.edu

Raghav Nandwani
M. Engg. Robotics
University of Maryland
College Park, MD
Email: raghav15@terpmail.umd.edu

***Abstract*—The goal of this project is to implement the function of replacing one face with another. We introduce the entire process step by step, including facial landmark detection, face warping, blending, and motion filtering, and show the results of our experiments.**

I. Phase 1: Traditional Approach

In this section, we will introduce the process of face swapping step by step, including facial landmark detection, face warping, blending. The overall process is shown in Fig.1. Specifically, in the warping step we will introduce two methods, Triangulation and Thin Plate Spline. After dealing with each independent frame, we use motion filtering to make the video look smoother.

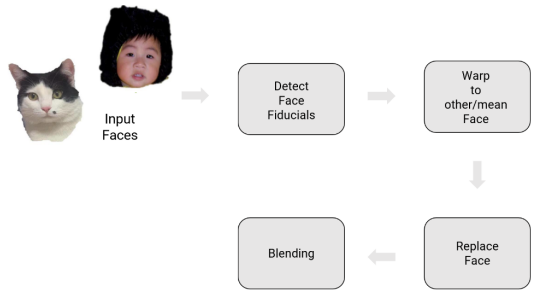

Fig. 1: Overall process of face warping.

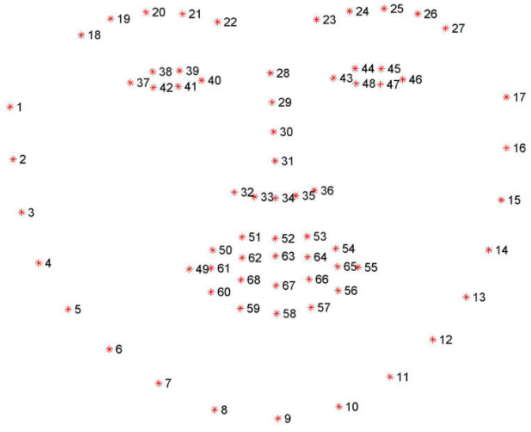

Fig. 2: Facial landmark model of 68 points.

A. Facial Landmark Detection

The first step of face warping is detecting the faces. Here we use *dlib* library to do the face detection. Specifically, the pretrained facial landmark detector inside the dlib library is used to estimate the location of 68 landmarks, which is shown in Fig.2.

An example of facial landmark detecting result is shown in Fig.3, which listed the landmarks on the face in order.

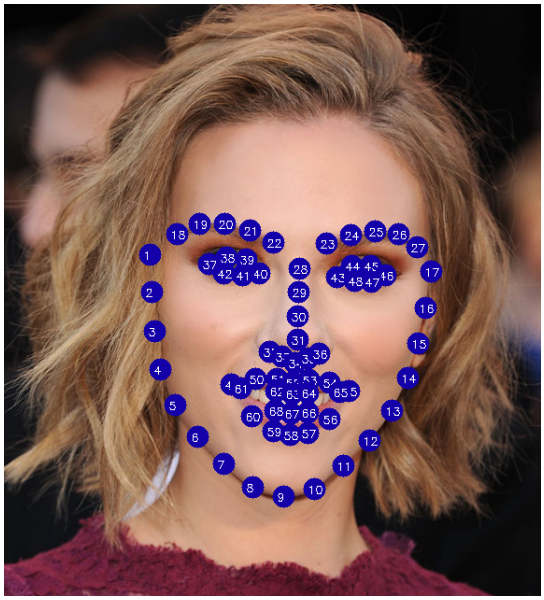

Fig. 3: Facial landmark model of 68 points.

B. Face Warping

In this section, we will show how to warp the face to another with two different methods, Triangulation and Thin Plate Spline (TPS).

1) Triangulation: Now the task is to warp the face in 3D, without actually having a 3d information. So to do that we applied triangulation technique using the facial landmarks as corners of that triangle. considering each triangle will form a plane. So an effective and efficient way of doing this is Delaunay Triangulation [5] as this can be constructed in $(\mathcal{O}(n \log n))$ time. using *scipy.spatial.Delaunay* function of *scipy* library. Firstly we tried triangulation but the triangulation

output was different as same triangles weren't corresponding to the same features. To compensate we perform triangulation on source image i.e. and stored the indexes and made triangles using those indexes in the target image so that we get the corresponding triangles with corresponding feature points in that image, like shown in Fig.4

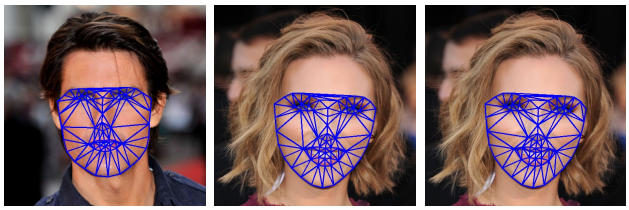

Fig. 4: (From left to right) Triangulation of source image, target image and target image after index matching

The output of the Triangulation in shown in the Fig.5 and Fig.6.

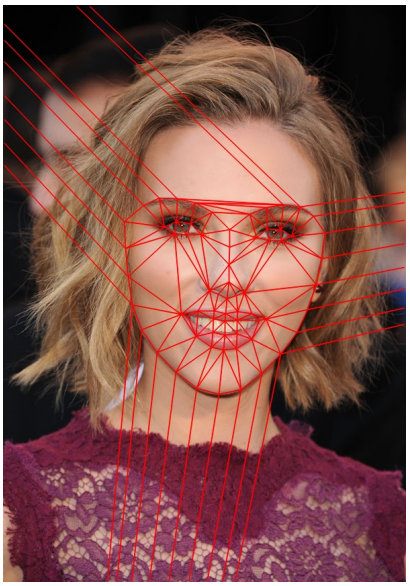

Fig. 5: Delaunay triangulation output of source image using the 68 landmark points

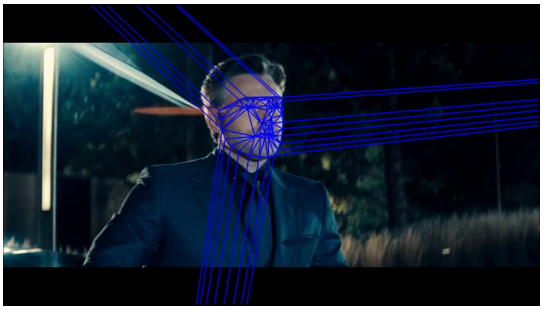

Fig. 6: Delaunay triangulation output of target image using the 68 landmark points

To extract triangle coordinates we used *getTriangleList()* function of *cv2.Subdiv2D*

$$\begin{bmatrix} \alpha_a & \alpha_b & \alpha_c \\ \beta_a & \beta_b & \beta_c \\ \gamma_a & \gamma_b & \gamma_c \end{bmatrix} = \begin{bmatrix} \mathcal{B}_{a,x} & \mathcal{B}_{b,x} & \mathcal{B}_{c,x} \\ \mathcal{B}_{a,y} & \mathcal{B}_{b,y} & \mathcal{B}_{c,y} \\ 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ 1 & 1 & 1 \end{bmatrix}$$

After this for every column check weather the point lies inside triangle or not using the following condition

- $\alpha \in [0,1], \beta \in [0,1]$ and $\gamma \in [0,1]$
- $\alpha + \beta + \gamma \in [0,1]$

Using that find the corresponding coordinates in the source image with target image's corresponding triangle coordinates.

$$\begin{bmatrix} x_{\mathcal{A}} \\ y_{\mathcal{A}} \\ z_{\mathcal{A}} \end{bmatrix} = \mathcal{A}_{\Delta} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix}$$

where $\mathcal{A}_{\Delta}$ is

$$\mathcal{A}_{\Delta} = \begin{bmatrix} \mathcal{A}_{a,x} & \mathcal{A}_{b,x} & \mathcal{A}_{c,x} \\ \mathcal{A}_{a,y} & \mathcal{A}_{b,y} & \mathcal{A}_{c,y} \\ 1 & 1 & 1 \end{bmatrix}$$

Converting the value of $\begin{bmatrix} x_A & y_A & z_A \end{bmatrix}$ homogeneous coordinates to 2d coordinates as

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

Interpolating the values of the pixel(x_A, y_A) to the target coordinates using *scipy.interpolate.interp2d*. Similarly doing this for all the triangles, the warped image looks like Fig.7.

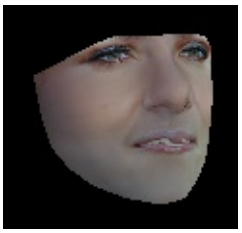

Fig. 7: Source Image warped after Triangulation and Blending

Fig. 8: Triangulation Output on Testset 3

Then the face is placed on the target frame using masking and bitwise operation (*cv2.bitwise_and()*) to generate Fig.8.

Fig. 9: Face warping using TPS. Images from left to right: source face, target face, replaced face. The replaced face has nearly the same landmark positions as the target face, while has nearly the same appearance as the source face.

Fig. 10: Face blending. Images from left to right: target face, replaced face, blended face. The blended face is more seamless than the replaced face.

2) Thin Plate Spline: Thin plate splines (TPS) are a spline-based technique for data interpolation and smoothing. Here we use the following equation to model the mapping function of landmark positions from on one face to another.

$$f(x,y) = a_1 + (a_x)x + (a_y)y + \sum_{i=1}^{p} w_i U(\|(x_i,y_i) - (x,y)\|_2)$$

Where $U(r) = r^2 log(r^2)$

Specifically, we use this function twice to map the landmark positions from one face to the x and y coordinate of landmark positions on another face, respectively.

After we get the mapping functions, we use inverse warping to replace the original face with the new one.

An example result of this step is shown in Fig.9. We can see the replaced face has nearly the same landmark positions as the target face, while has nearly the same appearance as the source face.

C. Blending

Since the brightness of the source face and target face may be different, it may look wired if we replace the face directly. Therefore we use several blending methods to make it look nature.

To make the brightness of replaced face looks similar with the target face, we remap the distributions of pixel values on the source face to the distributions of pixel values on the target face. Here we simply use a linear remapping function to align the expectation and range of the two distributions.

In addition, we use *Poisson Blending* to make the border of replaced face more seamless. Specifically, we use *cv2.seamlessClone* function to achieve this goal.

A sample result of the blending process is shown in Fig.10. The blended face is more seamless than the replaced face.

D. Motion Filtering

We use motion filter to make the blended face in a video more stable. Here we use a simple mean filter to reduce the amount of flickering. Let $p^{(t)}$ to be the landmark position in time t, and $p^{(t)}_{filtered}$ to be filtered position of landmark in time t, we use the following equations to get the $p^{(t)}_{filtered}$ for $t > 1$.

$$p^{(t)}_{filtered} = (1-\alpha)p^{(t-1)}_{filtered} + \alpha p^{(t)}$$

Where $p^{(1)}_{filtered} = p^{(1)}$ and $\alpha = 0.8$.

II. Experiment Results

In this section, we show the results of 3 test scenes.

1) Scene 1: Replace the face in Test1.mp4 by the face in Rambo.jpg.
2) Scene 2: Swap faces in Test2.mp4.
3) Scene 3: Replace the face in Test3.mp4 by the face in Scarlett.jpg.

A. Scene1

We show the result of replacing the face in Test1.mp4 by the face in Rambo.jpg, by the methods of Triangulation and Thin Plate Spline. The source face and target face is shown in Fig.11. The results of different methods is shown in Fig.12.

This case is relatively easy, the landmark detectior can nearly detect faces on all the frames. We show the results of one frame with different methods here. The full results can be found in the supplementary videos.

Fig. 11: Source face and target face of one sample frame in Scene 1.

Fig. 12: Sample Results of Scene 1 with methods of Triangulation (top), TPS (middle) and PRNet (bottom).

B. Scene2

We show the result of swapping faces in Test2.mp4. The source face and target face is shown in Fig.13. The results of different methods is shown in Fig.14.

Fig. 13: One sample frame in Scene 2.

Fig. 14: Sample Results of Scene 2 with methods of Triangulation (top) and TPS (bottom).

C. Scene3

We show the result of replacing the face in Test3.mp4 by the face in Scarlett.jpg. The differences between Scene 3 and Scene 1 are, Scene 1 has nearly constant brightness on face, while Scene 3 contains face in dark and there's flash in the scene, which is quite difficult. The source face and target face is shown in Fig.15. The results of different methods is shown in Fig.16.

REFERENCES

[1] Patrick Perez, Michel Gangnet† and Andrew Blake, *Poisson Image Editing*,Microsoft Research UK

[2] Tsung-Yi Lin, Genevieve Patterson, Matteo R. Ronchi, *COCO-Common Object in Context*,http://cocodataset.org/home

[3] Daniel DeTone, Tomasz Malisiewicz, Andrew Rabinovich, *Deep Image Homography Estimation*

[4] Ty Nguyen, Steven W. Chen, Shreyas S *Unsupervised Deep Homography: A Fast and Robust Homography Estimation Model*

Fig. 15: Source face and target face of one sample frame in Scene 3.

Fig. 16: Sample Results of Scene 3 with methods of Triangulation (top), TPS (middle), and PRNet(bottom).

[5] Satya Mallick *Delaunay Triangulation and Voronoi Diagram using OpenCV* https://www.learnopencv.com/delaunay-triangulation-and-voronoi-diagram-using-opencv-c-python/