CMSC733 Project 2 Report

Pyone Win
Department of Computer Science
University of Maryland, College Park
College Park, Maryland
Email: pwin17@umd.edu

Nitesh Jha
MEng Robotics
University of Maryland, College Park
College Park, Maryland
Email: niteshj@umd.edu

I. Overview

This project aims to implement a face-swapping pipeline which swaps faces in the video or replaces the face in a video with an image. This is implemented using two approaches: traditional method - which uses Delaunay triangulation and Thin Plate Splines, and the deep learning method - which uses a network PRNet to achieve swapping of faces.

II. Phase 1

In this phase, we would like to present two traditional methods of swapping faces between source image and destination image. The first method is called Triangulation and the second method is called Thin Plate Spline. The algorithm for both methods starts with detecting keypoints (or facial landmarks) of a face in a source image which will be used as point-to-point correspondences to warp a face from a destination image onto this. Another purpose this serves is it reduces the computation required in swapping process. As the correspondence between points in different faces is required, the detected keypoints have to be ordered in the same way. We achieve this using the 'dlib' library which returns the set of landmarks in a fixed way as shown in Fig 1.

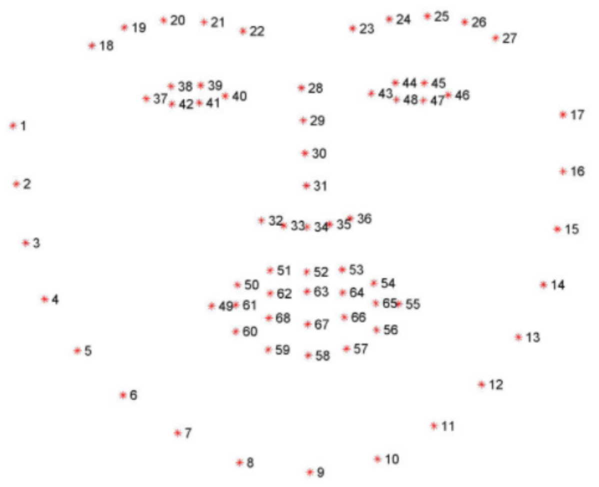

Fig. 1. Order of facial landmarks

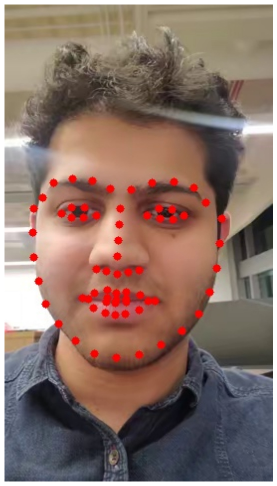

Fig. 2. Destination facial landmarks

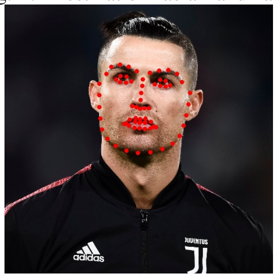

Fig. 3. Source facial landmarks

Fig. 4. Facial landmarks of source and destination

A. Triangulation

Once we obtain the facial landmarks of the two faces to be swapped, the next step is to warp one onto the other. For this, we form triangles between the points such that the circumcircle of any triangle does not include any other point. This method is called Delaunay Triangulation. We used opencv functions Subdiv2D() and getTriangleList() functions for this step. As we do not have a depth information, we made an assumption that these triangles are planar, thus 'simplifying' the face structure into a number of triangular planes. However, we observed that the two generated triangle lists were not ordered and there was no correspondence between them. We countered this leveraging the fact that the 68 landmark points are always

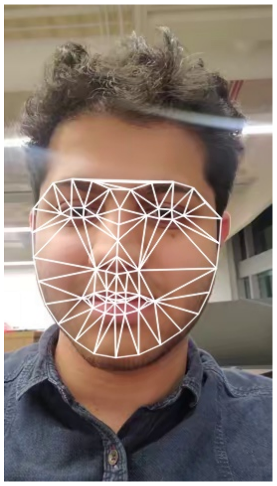
Fig. 5. Destination triangulation

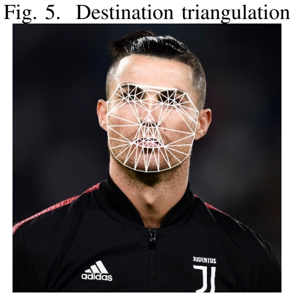
Fig. 6. Source triangulation

Fig. 7. Delaunay triangulation of source and destination

ordered, and therefore, by iterating over every triangle of one list and identifying the corresponding three points in the other triangle list, reordering was done for correspondence. Then, we assumed that all corresponding triangles have an affine transformation between them.

We calculated the barycentric coordinates α, β, γ for pixels inside every triangle, and retained only those which satisfy 1 and 2.

$$\alpha \in (0,1], \beta \in (0,1], \gamma \in (0,1] \tag{1}$$

$$\alpha + \beta + \gamma \in (0,1] \tag{2}$$

The computation for mapping all pixels between the source image and the destination image can be extremely expensive. Thus, for each triangle that we are evaluating, we found the pixels only inside the triangle boundary of the source images we are currently evaluating, hence reducing the computation required significantly.

Then, we calculated pixel locations corresponding to the barycentric coordinates generated from the destination triangle, in source image, and copied these coordinates onto the destination image. On repeating this process for all triangles, we obtained the warping. Note that as we performed inverse warping, we ensured that there were no holes present in the final result as would be in forward warping. The results are shown in Fig 10 After warping, we blend the source face on the warp image using the function cv2.seamlessClone() using the face binary mask and destination face center passed as arguments.

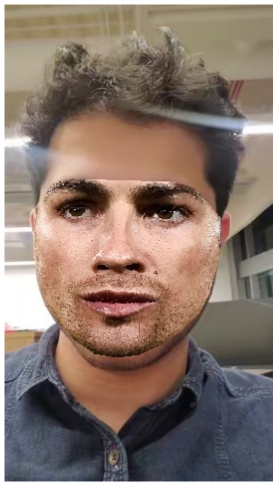
Fig. 8. Unblended: swapped result

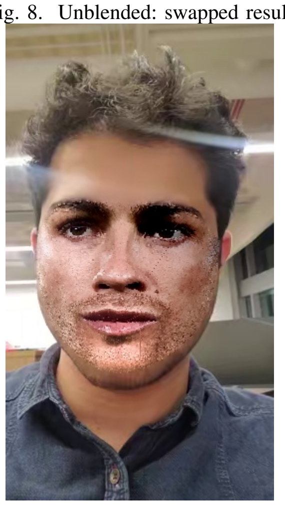
Fig. 9. Blended: swapped result

Fig. 10. Triangulation Warping results

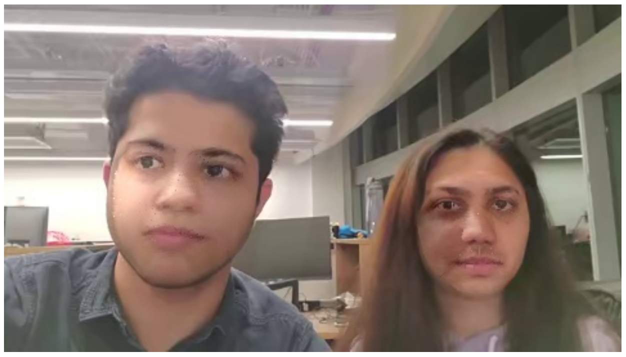
Fig. 11. Face swap within frame

As we have made assumptions of the contents of the face being in a plane to simplify calculations, the results do show some distortion. This assumption would still yield better results if the number of facial landmarks are increased for triangulation, as this would result in the triangular area being smaller, meaning smaller planar structures and hence it would be closer to the actual structure of the face.

B. Thin Plate Splines

As the triangulation warping assumes that the transformations are affine, we can use splines to model and capture the complex structure of the face. For this, we model splines using the form:

$$f(x, y) = a_1 + (a_x)x + (a_y)y + \sum w_i U(||(x_i, y_i) - (x, y)||_1) \tag{3}$$

where

$$U(r) = r^2 log(r^2) \tag{4}$$

The spline parameters w_i are calculated for both x and y separately, hence we obtain two spline parameters. We write this in matrix form as follows:

$$\begin{bmatrix} K & P \\ P^T & 0 \end{bmatrix} * \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ . \\ . \\ . \\ w_p \\ a_x \\ a_y \\ a_1 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ . \\ . \\ . \\ v_p \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{5}$$

We calculated K by iterating over every landmark coordinate (x_i,y_i) and taking difference with every other landmark coordinate of the source image (x_j,y_j). Summing all the terms according to 3 gives K. P is (x_i,y_i,1). v_i is the landmark coordinate of the destination image. We then took the inverse of the aggregate matrix and multiply this with v matrix to obtain x and y parameters w_i. Using these parameters, we multiplied all the pixels of the destination image inside the face (we use cv2.PointPolygonTest to find interior test inside the convex hull) to obtain corresponding locations in the source image. Copying these values to the destination image resulted in the warping of the faces. Results of this are shown in Fig 12 and 13. Final images in both methods were blending using cv2.SeamlessClone function to obtain the final result.

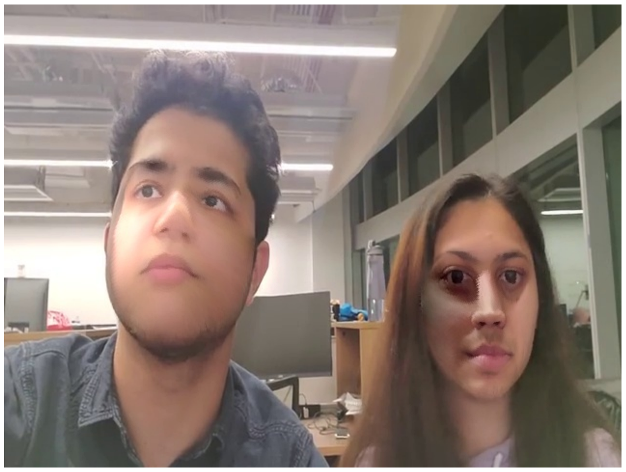

Fig. 12. Thin Plate Spline Result on Custom Input 1

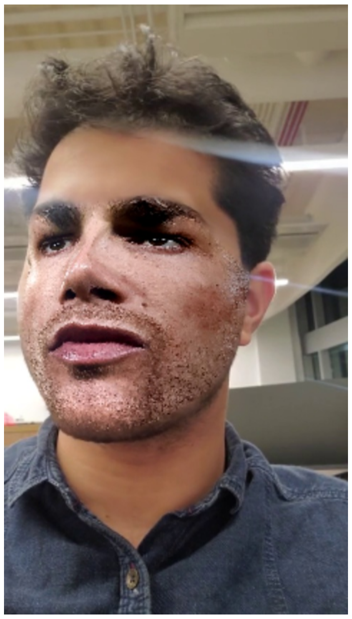

Fig. 13. Thin Plate Spline Result on Custom Input 2

III. Phase 2

For this phase, we are using the baseline code from [1]. They have also provided the trained weight, so we were able to use the model and the trained weights directly. In the original code, firstly, input image's faces were detected using the dlib library, which we were also using from Phase 1. However, one assumption the original authors made was that each image has only one face, or if they had more than one face detected, they always used the first detected one. We modified their algorithm and constructed our own workflow to take two images with one or more faces in each image. Since Project 2 involves two ways of swapping faces, we tailored our approach to match the instructions. The first way is to swap the face in video frame with one still image. The second way is to swap two faces detected in the same video frame.

For the first way, we were able to use the code as is. First, we received the vertices and texture information of the source image. Then, we also got the vertices of the destination image and changed the texture of the destination image with the texture information from the source image. Lastly, we swapped and blended the image using same procedures from Phase 1. For the second way of swapping, we set our source and destination images as the same image. We followed the steps from the first way for both the source and destination images. Here, we first swapped one face (from source face to destination face) and blended it. We, then, used this image to swap from destination face to source face with the texture information we obtained before the first swap.

Since PRNet uses dense face alignment based on the 68 feature points, the outputs looked great even when a person is not facing straight forward. The modern approach yields better results than the traditional approaches mentioned in Phase 1. The results are shown as below.

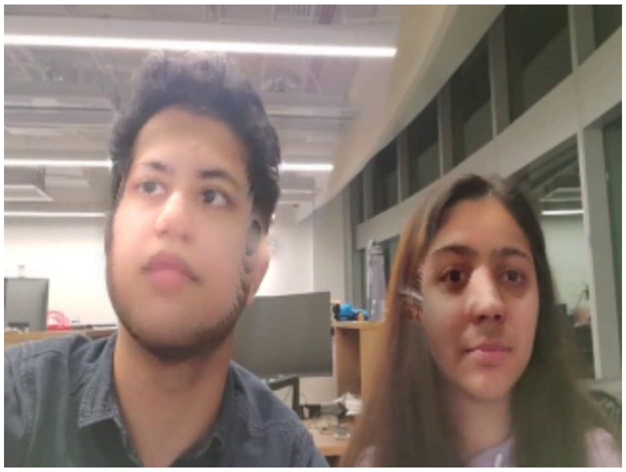

Fig. 14. PRNet Result on Custom Input 1

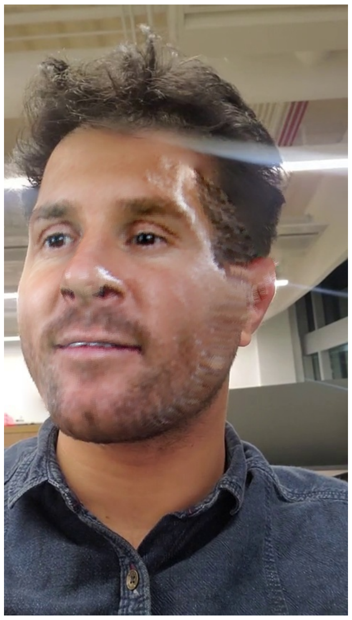

Fig. 15. PRNet Result on Custom Input 2

IV. Test set result

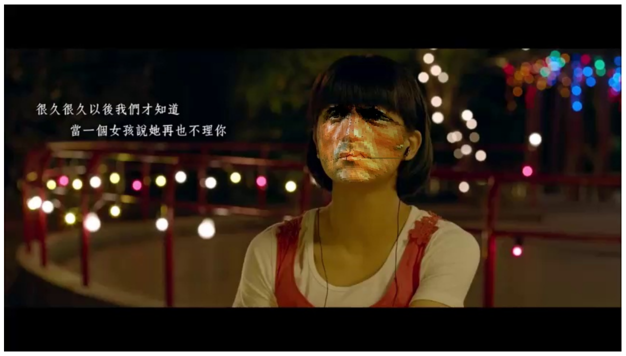

Fig. 16. Triangulation Result on Test Input 1

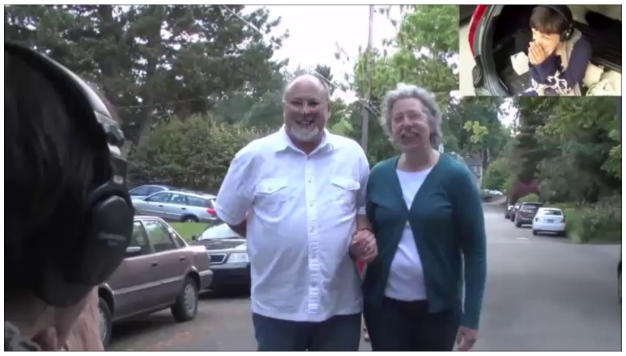
Fig. 17. Triangulation Result on Test Input 2

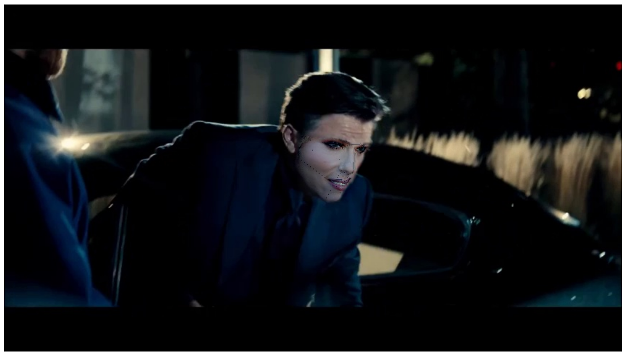
Fig. 18. Triangulation Result on Test Input 3

References

[1] Y. Feng, F. Wu, X. Shao, Y. Wang, and X. Zhou, "Joint 3d face reconstruction and dense alignment with position map regression network," in *ECCV*, 2018.

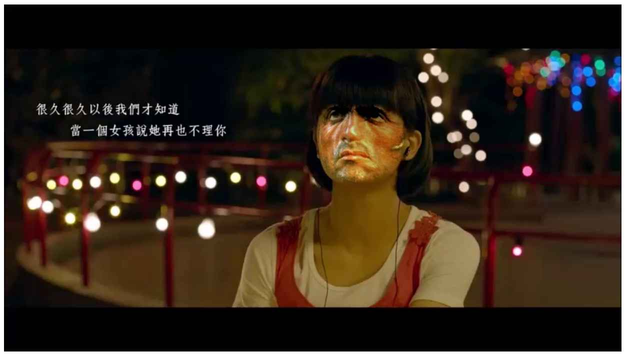

Fig. 19. Thin Plate Spline Result on Test Input 1

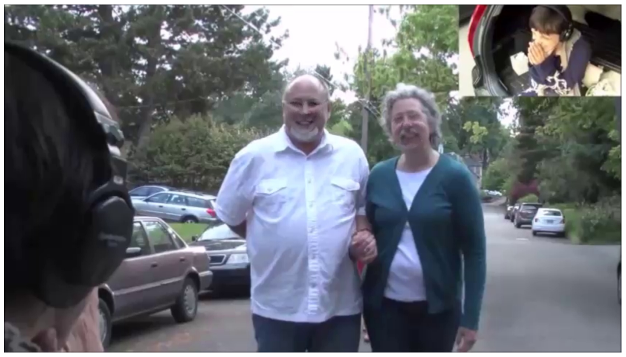
Fig. 20. Thin Plate Spline Result on Test Input 2

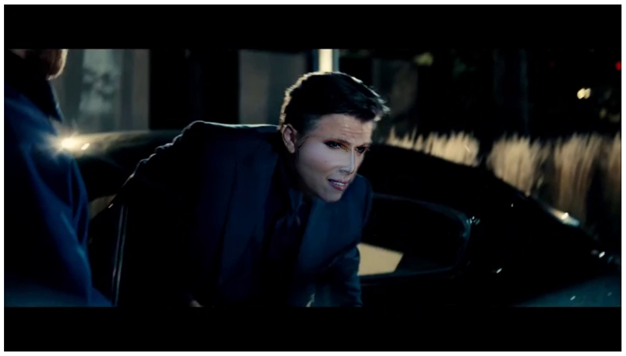
Fig. 21. Thin Plate Spline Result on Test Input 3

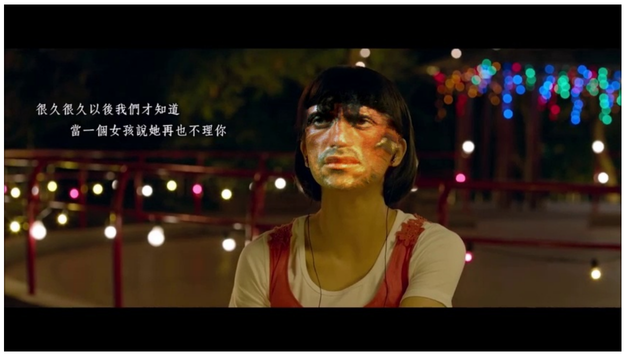

Fig. 22. PRNet Result on Test Input 1

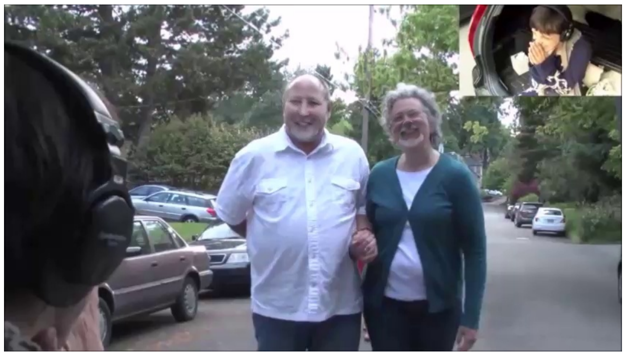

Fig. 23. PRNet Result on Test Input 2

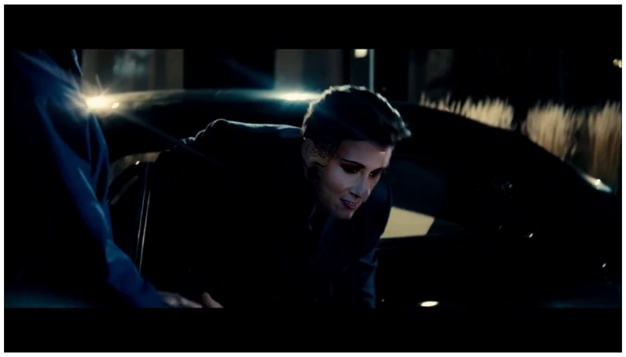

Fig. 24. PRNet Result on Test Input 3