# CMSC733: Project 2 FaceSwap

Varun Asthana
vasthana@umd.edu

Saumil Shah
sshah293@terpmail.umd.edu

Use 1 Late Day

## I. Phase 1a: Face Swap with Delaunay Triangulation

This section provides the details of the traditional approach to perform the face-swapping of 2 input faces. Input faces can be from 2 different images, from an image and from a video (frame) or 2 faces from a single video (frame). 4 main steps involved are: a) Detect a face in a given image, b) Generate corresponding Delaunay triangles, c) Warping of one face onto another, and color blending.

### A. Detect a Face

The very first step is to detect a face in the given image and find facial landmarks. For landmark detection we have used the *dlib* library, which detects 68 key-points for a front face. The detector used does not perform well with a side face pose. A pre-trained model is used to detect the landmarks, namely *"shape_predictor_68_face_landmarks.dat"*. The detected landmarks are always stored in a particular order, shown in Fig 1. Sample outputs for 68 frontal-face landmarks are shown in Fig 2.

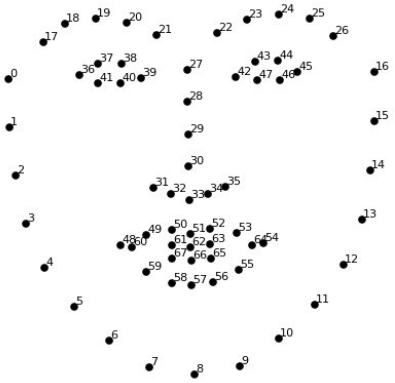


Fig. 1: Ordered 68 frontal-face landmarks

### B. Generate Corresponding Delaunay Triangles

Next step is to have a methodology that can help to transform one face into another by considering the 3D features, when only 2D image is available. This is done by generating a triangular mesh by considering the 68 landmarks as the vertices. An assumption is made that the content in each triangle is planar. Here the triangles were obtained by drawing the dual of the Voronoi diagram, called *Delaunay Triangulation*. We used the *getTriangleList()* function in *cv2.Subdiv2D* class of OpenCV to generate

Fig. 2: 68 frontal-face landmarks detection

the Delaunay Triangles. Sample outputs for the generated triangles using the 68 landmarks are shown in Fig 3.

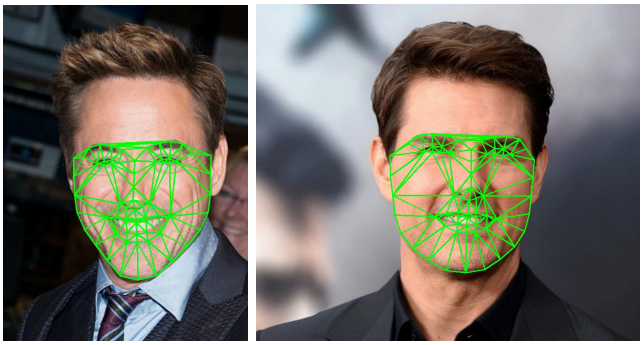


Fig. 3: Delaunay triangles

To perform the face-swapping, we aim to warp data from source image to target image. This step is to be done in parts for each triangle (affine transformation). To successfully do this, we should know that data from a particular triangle in the source image will be used to replace data of which triangle in the target image. Thus we require a correspondence in the Delaunay Triangles in both the images. But since the Delaunay triangulation tries the maximize the smallest angle in each triangle, it is not necessary that we get the same triangular mesh in both the images (due to different face structure and face pose). This is shown in Fig 4.

To overcome this issue and have the similar triangular mesh (to achieve correspondence), we explored the unique property of the output of the 68 landmarks, i.e. they are always ordered in a defined pattern. Hence Delaunay trian-

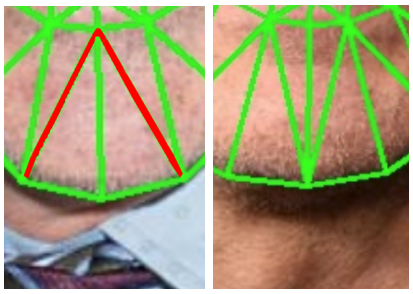


Fig. 4: No correspondence in the obtained triangular mesh

gulation was done on only source image using the *getTriangleList()* function. All the 3 vertices of each triangle were compared with the landmark data of the source image to find which points were used to form triangles. A corresponding triangle was drawn in the target image using the same set of landmarks of the target image. Output samples for this methodology is shown in Fig 5.

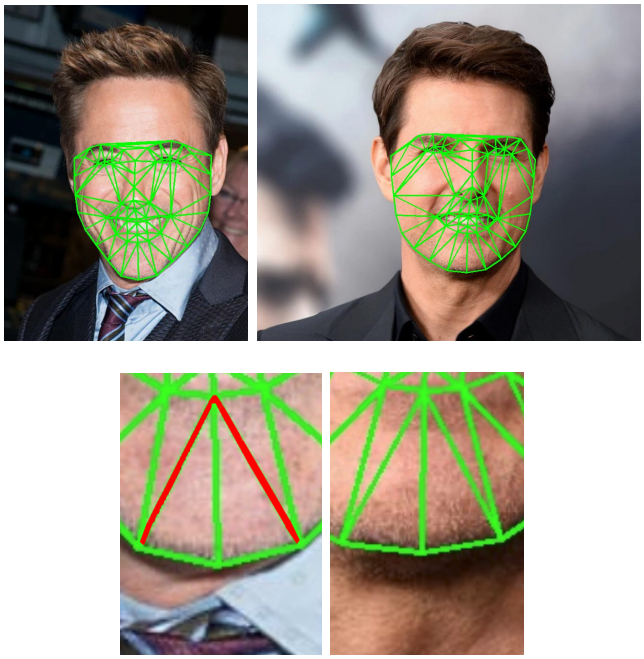


Fig. 5: Corresponding Delaunay triangles

### C. Warp a Face onto Another and Color Blending

In parallel to the generation of corresponding triangle in the target image, we would simultaneously warp the content of the source image onto the target image. We used the inverse-warping technique by utilizing *Barycentric* coordinates. Since currently we only have the 3 vertices of 2 triangles (one in target image and one in source image), we need to find all the coordinates that lie inside the target image triangle to perform inverse warping. Barycentric coordinates provide an efficient way to transform all the data for a particular triangle pair at once (using matrix representation). To further reduce to operating time, we generated a bonding box around the triangle using *cv2.boundingRect()* function and used this patch of the target image to find all the points inside the triangle. The conditions used to check if a point lies inside the triangle are-

$$\alpha \in [0,1] \quad \beta \in [0,1] \quad \gamma \in [0,1]$$

where $(\alpha, \beta, \gamma)$ is the triangle vertex in the Barycentric coordinate system (1)

Once the corresponding coordinates in the source image are calculated, we used the *scipy.interpolate.interp2d()* function to get the pixel values in the source image for each color channel, and then copied the data onto the target image. Output of the swapped face is shwon in Fig 6.

Fig. 6: Swapped face

For color blending we utilized the *cv2.seamlessClone* function. After warping we obtained the common area in the swapped image and the original target image i.e. the convex hull of the 68 landmark points of the target image. We then applied the function only over the common area by generating a binary mask, and swapped image was blended with the original target image. The output is shown in Fig 7.

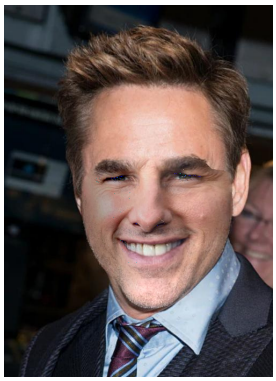


Fig. 7: Color blended swapped face

## II. Phase 1b: Face Swap with Thin Plate Spline

In the previous section we saw how can we divide a 3D face structure into 2D meshes and reconstruct it using 2D

affine transformation on each triangular mesh. This doesn't always work and doesn't give consistent results. One more approach can be used using spline fitting. With thin plate spline we can create a spline curve passing through the boundary of the facial structure and warping it according to the target image where we want to place that face. Warping of a thin plate spline is represented in a mathematical form as below.

$$f(x,y) = a_1 + (a_x)x + (a_y)y + \sum_{i=1}^{p} w_i U\left(||(x_i,y_i) - (x,y)||_1\right) \tag{2}$$

$$where, U(r) = r^2 \log(r^2 + \epsilon), \epsilon = 0.00001 \tag{3}$$

Writing the same equation in the matrix form can drastically reduce the time complexity of the algorithm. Same equation can be written in the matrix format as shown below. $\epsilon$ is necessary to prevent the condition of $log(0)$. Lambda can be anything near to zero. Here we have taken a random value of $0.00001$.

$$\begin{bmatrix} K & P \\ P^T & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_p \\ a_x \\ a_y \\ a_1 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_p \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4}$$

$$\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_p \\ a_x \\ a_y \\ a_1 \end{bmatrix} = \left( \begin{bmatrix} K & P \\ P^T & 0 \end{bmatrix} + \lambda I(p+3, p+3) \right)^{-1} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_p \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{5}$$

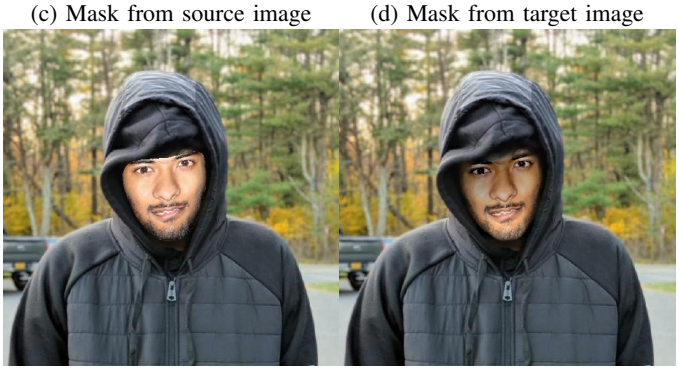
Value of K depends on the points to be warped and the facial fiducial points with respect to points should be warped. We know the initial 68 facial landmarks on both source and target images. We can use this values to solve the Eq 5. We can now use the weight values calculated from this, to warp all the points in the source image to target image. To find the weights, we need to take the inverse of the matrix and sometimes it may happen that inverse is not possible because of singular matrix. To tackle this problem, a small term $\lambda$ is inserted in the Eq 5. Usually value of $\lambda$ is very close to zero and we have used $\lambda = 1 \times 10^{-8}$.

Entire procedure to swap a face using thin plate spline can be explained as below.

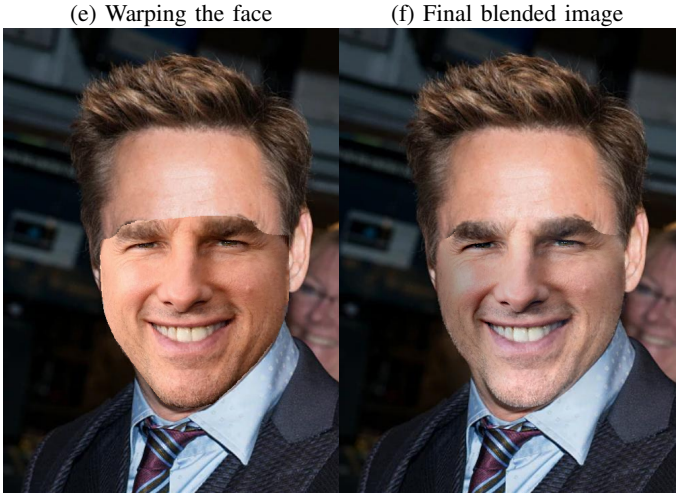
- Get the 68 face fiducials using dlib on both the source image and the target image.
- Use those 68 landmarks to create a mask consisting just the face like shown in Fig 8c and Fig 8d. Use Eq 4 to find the weights.

(a) Source image (b) Target image

(c) Mask from source image (d) Mask from target image

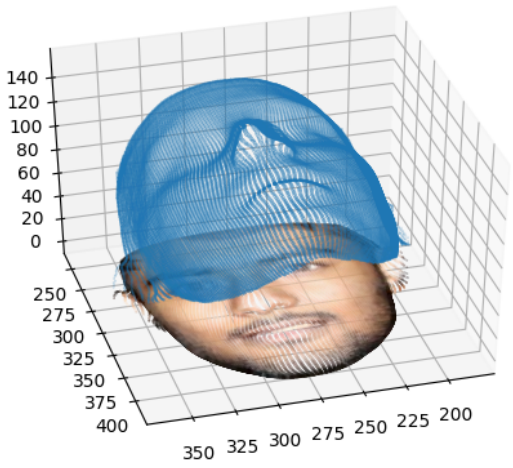


(e) Warping the face (f) Final blended image

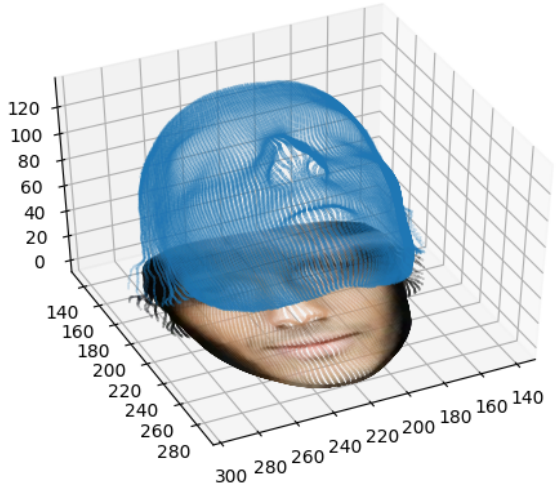


(g) Warping the face (h) Final blended image

Fig. 8: Swapping a face using thin plate spline




(a) Source image




(b) Target image

Fig. 9: 3D mesh obtained for an image using PRNet model

- Generate a bounding box on the convex hull of the target image's mask, and use the above weights to inverse warp all the point from the source image onto the target image. Warped image is shown in Fig 8g.
- Use *cv2.seamlessClone* to blend the warped face into the target image. Only the points within the mask of the source image are reflected in the final output. Final result with the swapped face is shown in Fig 8h.

## III. Phase 2: 3D Face Mesh using Deep Learning

In reference to the approach used by the authors of [1], we have used their code directly, provided at [2], for this phase.

The code in the file *demo.py* provides 2 main outputs that are being used by us to perform the face swapping. These outputs are- 1) 3D data for 68 key-points on a face, and 2) 3D mesh data for points inside the face boundary (including the forehead). Output of the 3D mesh data for the entire face is shown in Fig 9. Slight modifications were done in the file *demo.py* to use these data (2D data of (x,y) by ignoring the z-axis data) with our previous approach of TPS. We also modified the TPS code, such that the weights are calculated using the 68 face landmarks while the warping is done using a face mask which covers the entire face including forehead area. The sample mask of the source image used for warping is shown in Fig 10. The final output of the swapped face is shown in Fig 11. Instructions on how to use the modified *demo.py* and *tps.py* is provided separately in the README.md provided with the code.

## IV. Results

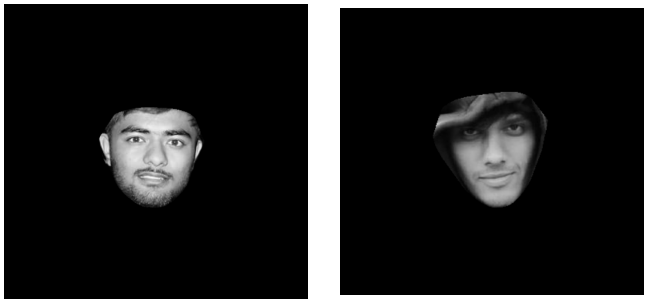
We were able to implement all the methods in the defined manner with acceptable performance. A comparison of the output for the 3 methods, namely traditional approach with Delaunay triangulation, traditional approach with TPS, and deep learning approach with TPS, is shown in Fig 11,Fig 12 and Fig 13.

Few observations from the outputs of all the 3 approaches are-

- From Fig 11 it can be noted that at the mouth area, Delaunay and Deep Learning methods produced good result, while the traditional TPS method have some deformations. This is due to the difference in the source and target image. The source image have open mouth (with teeths visible) and the target image has closed mouth. Since triangulation method warps the entire triangle formed by the facial landmarks hence output appears to be smooth. While with TPS since a spline is fitted at the landmarks, the difference in the position of the landmarks on upper and lower lips caused to have pixels copied from the teeth area in the source image during warping.
- Since we have used the forehead area also from the source image in deep learning method during warping, it can be seen from Fig 11 that it may cause an issue if the target image has its forehead area occluded. But since the TPS is formed using the 68 landmarks, and in a way extrapolated for the forehead area, so it is warped properly but causes issue while performing color blending (shades of gray blended from the original target image).
- Fig 12 shows good result of face swap from deep learning method, compared to both traditional approaches. This is strongly visible near the right eye. Suspected reason for this is that in traditional approach, due to face pose we have limited information and thus cannot be warped efficiently. While in deep learning approach, forehead is also considered this leads to more area near the eye and output looks better after color blending.

## V. Test Result Discussions

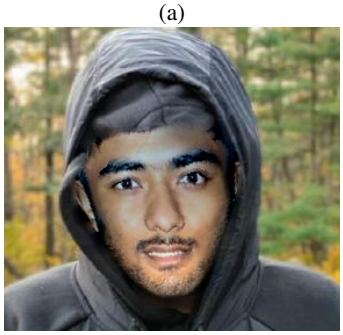
We have successfully executed all the 3 test cases.



(a) Source mask (b) Target mask

Fig. 10: Warping mask used in Deep Learning method

(a) (b)

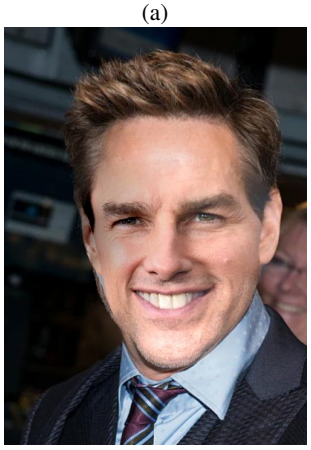


(c)

Fig. 11: Swapping face using (a) traditional approach with Delaunay triangulation . (b) traditional approach with TPS, (c) deep learning approach with TPS

- For test case 1 we swapped the face in a video with a face from a static image. Face from Rambo.jpg was transferred onto the face of a girl in the video. We have used both methodologies of Delaunay triangulation and the Thin Plate Spline to this test case. Results were good with both the algorithms. Due to very prominent texture features of the face in the Rambo.jpg, the swapped face can be distinguished easily even after color blending. We also observed some flickering due to the variation in position pf the detected face landmarks in each frame of the video. In the source image, mouth of Rambo is closed and teeth information is not present. In the video, when girl's mouth is open, swapped face will have blank pixels at the open mouth location. Sample results can be seen in Fig 14.
- For test case 2, we had to swap two faces within a video and it was performed successfully. Dlib was not able to

(a) (b)



(c)

Fig. 12: Swapping face using (a) traditional approach with Delaunay triangulation . (b) traditional approach with TPS, (c) deep learning approach with TPS

(a)

(b)

(c)

Fig. 13: Swapping face (a) original image (from custom video) . (b) traditional approach with TPS (similar result with triangulation, (c) deep learning approach with TPS

(a) Using Delaunay Triangulation (b) Using Thin Plate Spline

Fig. 14: Swapping face in video for test case 1

(a) Using Delaunay Triangulation (b) Using Thin Plate Spline

Fig. 15: Swapping face in video for test case 3

find both the faces in every frame, especially in side face position. Sample output is shown in Fig 16.

- For test case 3, we were provided with a video and had to swap the face in the video with a face in a static image of Scarlett.jpg. This case was different from test case 1 in the sense that the video provided here was dark and most of the time frontal face position was not present. These conditions made it difficult to swap the two face properly. Swapping with Delaunay triangulation was more stable than the swapping with Thin Plate Spline method. Absence of the frontal face created much more flickering in the later method. Sample output of the this is shown in Fig 15.

All the videos of test results can be found here.
`https://drive.google.com/drive/folders/1qrbH84ziI4b4HyK-ch1x3Vr0L1aGgunc?usp=sharing`

Fig. 16: Swapping 2 faces in one video

## REFERENCES


[1] Joint 3D Face Reconstruction and DenseAlignment with Position Map RegressionNetwork, ao Feng†, Fan Wu, Xiaohu Shao, Yanfeng Wang, Xi Zhou, 2018,

[2] https://github.com/YadiraF/PRNet

[3] https://drive.google.com/drive/folders/1qrbH84ziI4b4HyK-ch1x3Vr0L1aGgunc?usp=sharing