Project2: FaceSwap
RBE549

Karter Krueger
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA 01609
Email: kkrueger2@wpi.edu

Tript Sharma
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA, 01609
Email: tsharma@wpi.edu

I. Phase 1: Traditional Approach

We use traditional computer vision algorithms to swap two faces in a given image or video frame. We explored two approaches to achieve this: (a) Delaunay Triangulation (b) Thin Plate Splines.

To replace a face, we must first detect the faces using facial fiducials which are distinctive feature points on the face. We detect these using the opencv dlib library which detects 68 points on the face as seen in Fig. 1.

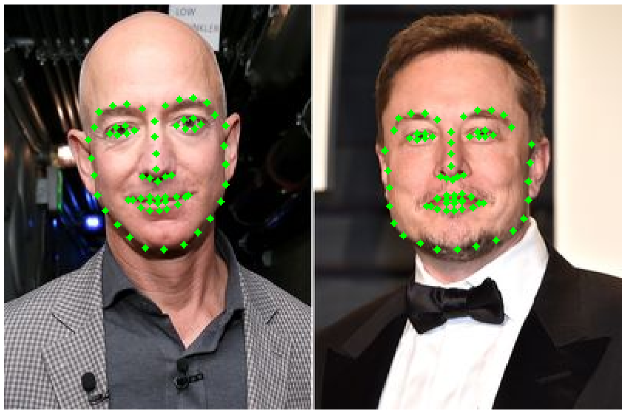
Fig. 1. DLib fiducial points plotted on two faces

In the next step, the two methods of triangulation and TPS diverge down their respective paths.

A. Delaunay Triangulation

Using the fiducial points, we generate a Delaunay triangulation, which is the dual of a Voronoi diagram. The Delaunay method triangulates the points in a specific way such that triangles are optimized to maximize the smallest angle within each triangle to form a consistent triangulation, as seen in Fig. 2. We take the triangulation of $Face_A$ and project it onto the matching fiducials points of $Face_B$.

Once we have triangle meshes for both faces, we can warp the section of face of each triangle to the corresponding facial triangle in the opposite face. We perform the warp by inverse warping of interpolated values to prevent there being undefined sections of the face that would create black spots or artifacts.

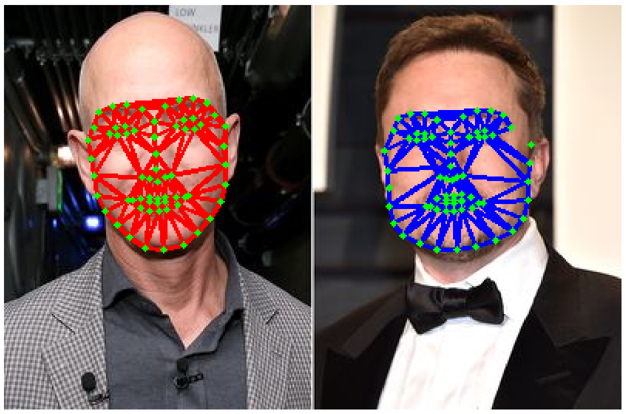
Fig. 2. Delaunay triangulation of the facial fiducial points

We inverse warp a triangle by converting to Barycentric coordinates, applying the inverse of the transformation matrix, and converting back to (x,y) coordinates from Barycentric as shown below.

For each triangle, we first convert all (x, y) coordinates in the image to Barycentric coordinates $[\alpha, \beta, \gamma]$ using the three corners a, b, c of triangle B, provided as their (x, y) values in the below matrix.

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

To solve for Barycentric coordinates, we invert the B matrix and multiply by the (x, y) values in homogeneous form.

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

After converting all points to Barycentric coordinates, we check which converted points lie inside the boundaries of the triangle by checking that the following conditions are met:

$$\alpha \in [0, 1], \beta \in [0, 1], \gamma \in [0, 1], (\alpha + \beta + \gamma) \in [0, 1].$$

We then convert the Barycentric coordinates of triangle B to homogeneous coordinates of triangle A using the following.

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

Homogeneous coordinates are then converted back to frame coordinates.

$$x_A = \frac{x_A}{z_A}, y_A = \frac{y_A}{z_A}$$

Lastly, `scipy.interpolate.interp2d` is used to interpolate a value from intermediate Face A positions $(x_A, y_A$ to the new target pixel in Face B. The final result of the Delaunay triangulation-based method is shown below in Fig. I-A.

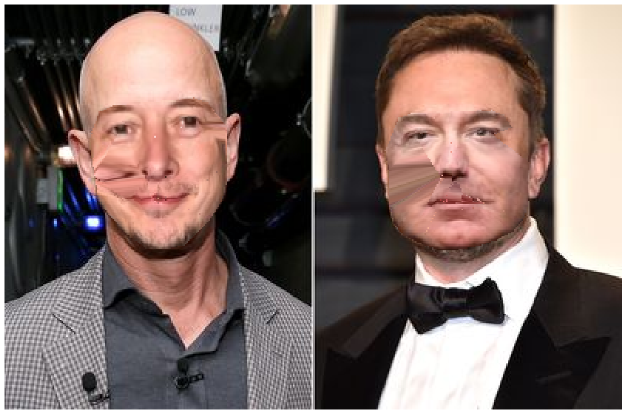

Fig. 3. Full face-swap operation using Delaunay triangulated method

Additional results with another set of faces is shown below.

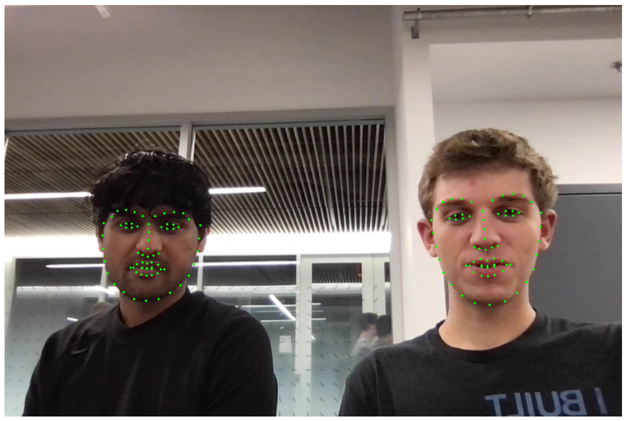

Fig. 4. Dlib face points

B. Thin Plate Spline

The results from Subsection I-A show how triangulation can miss some triangles from being warped. This partial swap can be overcome by fitting a thin plate spline to smoothly map the

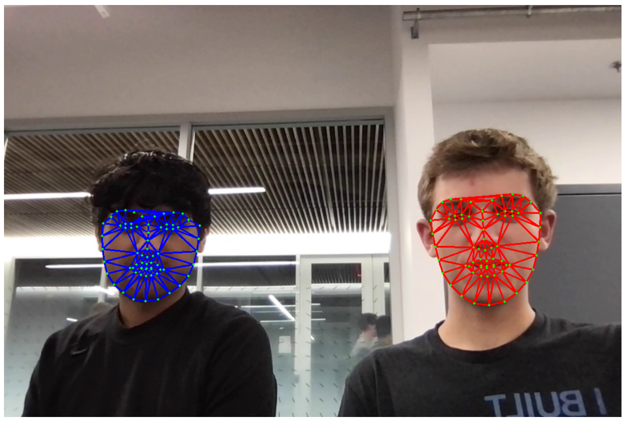

Fig. 5. Delaunay triangles

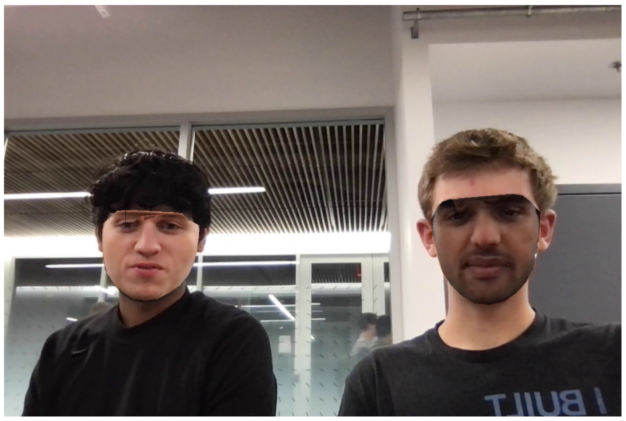

Fig. 6. Full face-swap operation using Delaunay triangulated method

points of one face to the other face. We map feature points in $Face_B$ to corresponding points in $Face_A$ to perform an inverse warping. To do this, we first calculate the coefficients necessary for the non-linear/spline part along with the affine part. This involves using a Radial Basis Function (RBF) to determine the "energy" required to bend between two points separated by a distance r as follows

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

The weights and affine variables are solved a system as follows.

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

Here K gives us the relationship between all the features points in $Face_A$ while $P_i = (x_i, y_i, 1)$. Once we have the weights and affine parameters, we use them in the $f(x,y)$ function to get the new points for $Face_A$ as seen in the equation.

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

where, (x,y) are points contained within Face B, determined by a convex hull that surrounds all feature points of Face B. These points are used to swap the two faces where the color at each location is determined using `scipy.interpolate.interp2d` for a smooth image.

The final result of the TPS-based face-swap method is shown in Fig. 7.

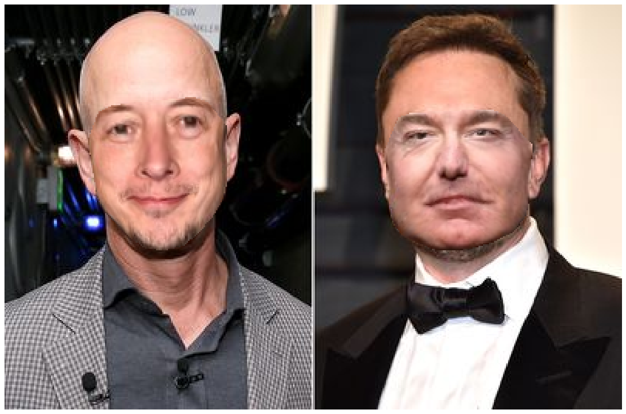

Fig. 7. TPS-based full face-swap of two faces

C. Blending

We perform blending using the Poisson image blending method, which is implemented in the `seamlessClone` OpenCV function. We blend using the original image, the replaced face, and a mask of the face region. The blending result is shown in Fig. 8.

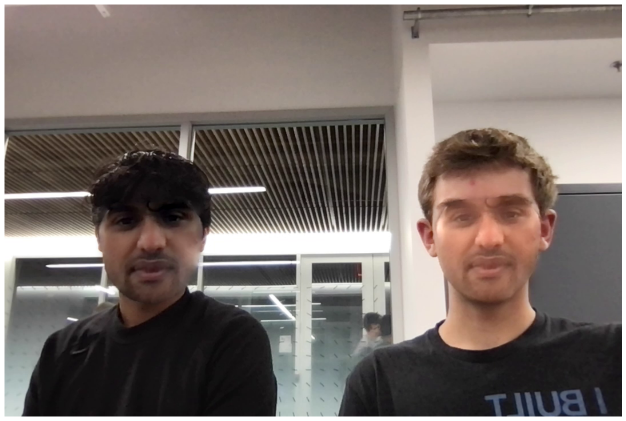

Fig. 8. Blending result of 2 faces with TPS

D. Motion Filtering

We filtered the motion with a low-pass filter method that performed a rolling average of locations for the 68 points between frames. (this was not used in the video due to lower framerate for smaller filesize)

E. Failures

You will notice the triangulation sometimes fails to match in the same way which causes distortion. We believe a better filtering method, such as a Kalman Filter would help this issue.

II. Deep Learning Approach

Phase 2 of the project is tasked with swapping the faces using a deep network to detect facial features. We used an existing implementation of the network called "Face Alignment in Full Pose Range: A 3D Total Solution." The network outputs 3d facial fiducials, a 3d facial mesh, and the normal vectors for each point on the mesh.

A. Implementation

Our implemented solution uses the facial fiducials from the network in a similar way as the features from Phase1, which describe certain parts of the face. We then feed these points into the TPS method of Phase1 to perform the full face swap. Results of the PRNet-based face swap are shown below in Fig. 10.

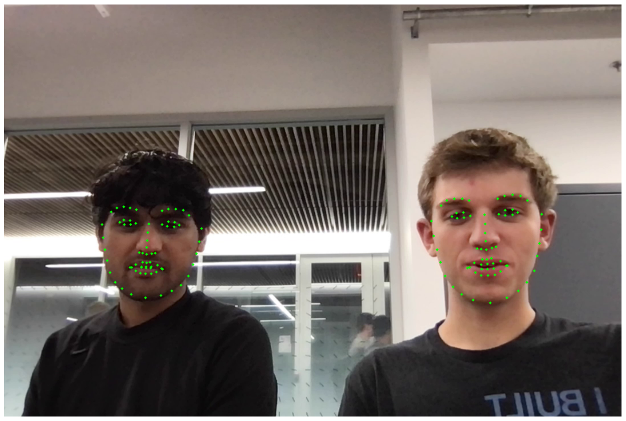

Fig. 9. Feature Points from PRNet

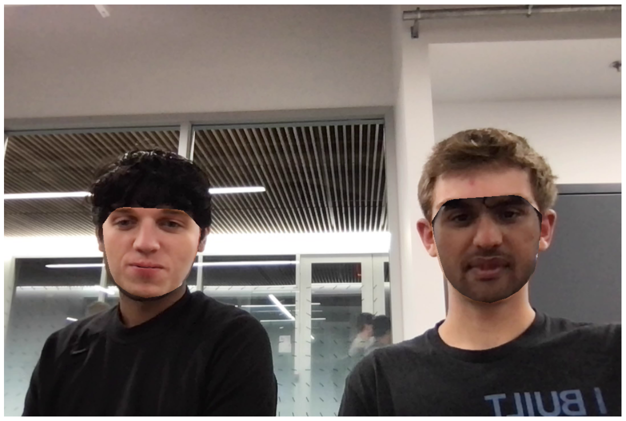

Fig. 10. Face swap using PRNet to generate fiducial points for TPS-based swap

We also had other ideas to further utilize the mesh outputs of the deep network.

B. *Extra Idea 1*

The first idea was to perform inverse ray tracing to first find the points of the 3d mesh that are visible from the camera perspective by back-projecting the rays from the 3D points to the camera center to find the intersecting pixel coordinate. Using the masks generated by the two faces on the camera plane, we then find the mask of the overlapping regions of visible face to determine which portions of the face meshes can be swapped (parts that are both visible by the camera). We then find the fiducial points along the edges of the masked meshes that intersect with a curve drawn between the facial fiducials from the network. An illustration of this idea is shown in Fig. 11.

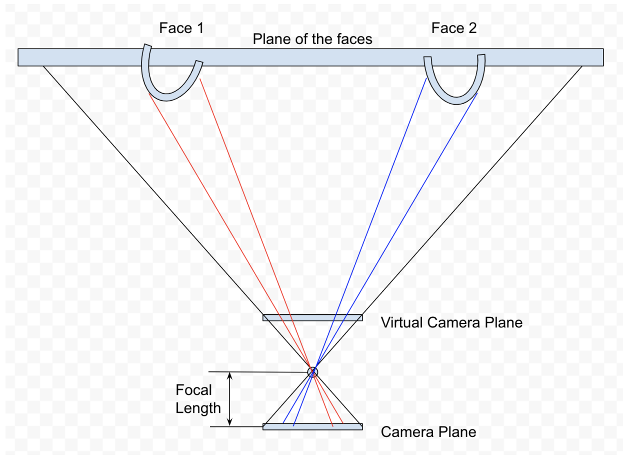

Fig. 11. Using inverse ray-tracing to find overlapping face mesh points

C. *Extra Idea 2*

Align the faces using the normal-vector masks, by aligning a distinguishable features of all faces - the nose. The nose has a unique shape of normal vectors pointing in almost all directions, and as it is the furthest forward features of a facial mask. The nose vectors can then be averaged on both faces to align their poses and then generate a mesh grid across the rest of the face centered at the nose that spans to the edges. The mesh grids of the two faces can be sparse of only 100 points, then those points can be passed into TPS as normal for a face swap operation.