Project 2: Face Swap

Radha Saraf
Email: rrsaraf@wpi.edu

Sai Ramana Kiran Pinnama Raju
Email: spinnamaraju@wpi.edu

***Abstract*—The aim of this project is to implement an end-to-end pipeline to swap faces in a video. We explore both classical and deep learning methods to this end.**

Irrespective of the method however, the strategy or the pipeline in both the cases remain same and is shown in the figure 1

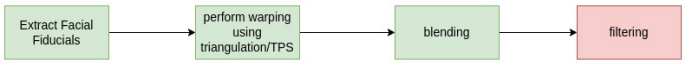


Fig. 1. Face swap pipeline

I. Phase I: Classical Approach

One of the main challenges in face swapping is to find a function to "map" each pixel in one face to each pixel in the other face. For this mapping we first find a set of landmarks and their correspondences in the target faces. We used `dlib` library for the same and obtained facial fiducials as shown in the figure 5

After obtaining the fiducials, our next step is to perform the mapping. A naive approach for this wouldve been is to map pixel by pixel in 3D space. This approach although simple, wouldve given us pretty good results. However, this approach is computationally intensive and wouldnt work well in real time applications. Hence, we simplify this problem of computationally complexity by approximating the facial structure in 2D space. We achieve this using the following 2 methods,

1) Delaunay Triangulation
2) Thin Plate Splines [1]

A. Face swapping using delaunay triangulation

The strategy for delaunay triangulation is as follows,

1) take landmarks obtained in the previous step
2) construct triangles using the landmarks
3) construct a convex hull around the face
4) inverse warp each pixel within triangle using barycentric coords and affine transformation
5) warp the faces

Following subsequent subsections discuss on some of the observations, challenges and finally results of this approach in the swapping faces

1) Inconsistent Landmarks: One of the initial challenge we faced during this was that some of the landmarks obtained from the above library are not unique. Some of the points are representing the same points, because of which there were some problems in the pipeline downstream. A simple problem was that there were singular matrices while finding barycentric coordinates. To avoid this issue we ran a unique check after the fiducials are obtained

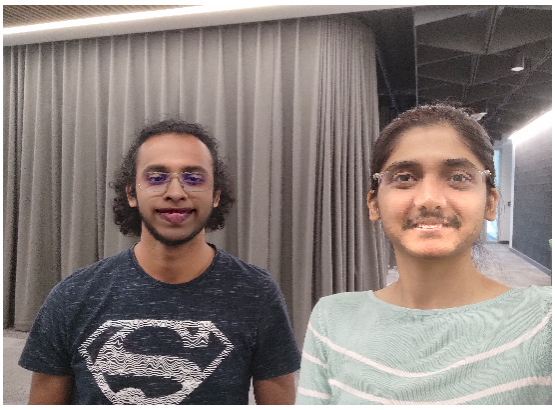
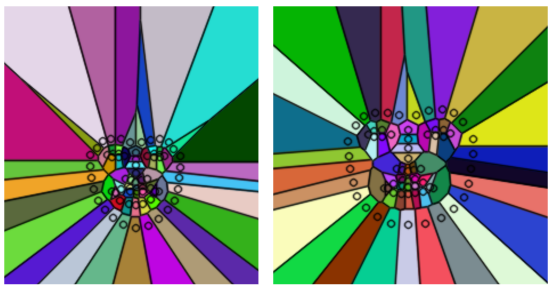


Fig. 2. Output image after blending using TPS

Fig. 3. Face Swap using Triangluation after accouting for the numerical inaccuracies in the inverse calculation

2) Finding triangle correspondences: In order to construct the triangulation using the landmarks we use the dual of Voronoi diagram. Vornoi diagram of our team can be seen in the figure 4. One can easily identify that dissimilar voronoi diagrams by paying attention to the number of "horns" in the voronoi output. Theoretically, for similar faces, we could obtain the approximately same voronoi diagram and hence same triangulation using this formulation. However, in reality faces are hardly similar.

To further simplify this, we first obtained the triangulation on one face patch and applied the same triangulation on



a. face 1 with 3 horns b. face 2 with 1 horn

Fig. 4. Voronoi diagram due to dissimilar faces

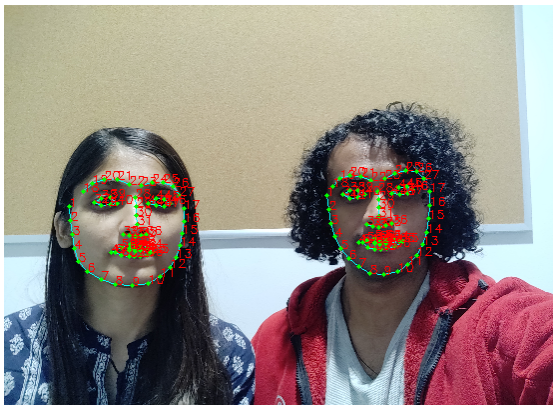


Fig. 5. Face Fiducials

the other face patch using the landmark correspondences. Essentially, since we know the landmark correspondences we *forced* the triangle correspondences across the faces using one as a reference. Figure 6 shows the triangulation obtained on both the faces. The colors, order and structure of the triangles across the face patches are identical in this version.

3) Inverse warping and Numerical inaccuracies: Once we have the triangle correspondences between the faces, we just need to apply the affine transformation from source image to the destination image giving us a *forward warp* of the patch.

Fig. 6. Correct Triangulation by force

Fig. 7. Missing Data in triangulation method

However, this approach has following 2 issues

1) Holes in the destination image
2) destination pixels might be subpixel values instead of pixel indices

We mitigate these 2 issues using inverse warping of the patch. i.e we first construct the destination grid on which we need to *warp to*. Since we have a concrete destination indices we reverse map these indices to the source image and get the pixel values from these coordinates. If they fall on the subpixel values in the source image, we interpolate the values and get these pixels hence mitigating both of the above issues

One of the ways to achieve this inverse warping of the pixels within the triangle is to calculate "Barycentric Coordinates" of pixels in the destination image and use these coordinates to find the pixel correspondencies in the source image triangles. These coordinates are essentially factors while representing the point within triangle as a linear combination of the triangle vertex. Hence, for this reason, every barycentric coordinate obtained for a point within a triangle **must** and will satisfy the conditions mentioned in equations 1. Once we get this coordinate we can map destination image pixel to source image pixel.

$$\begin{aligned} \alpha &\in [0,1] \\ \beta &\in [0,1] \\ \gamma &\in [0,1] \\ \alpha+\beta+\gamma &\in (0,1] \end{aligned} \tag{1}$$

However, following the steps above, resulted in the figure 7. The figure without blending shows the holes despite the inverse warp on both of the faces. The reason for this is due to the numerical inaccuracies in the inverse calculation of the barycentric coordinate on the destination image. Essentially, these inaccuraccies in the inverse matrix have not been taken into account initially when the mapping was performed giving rise the aforementioned face swap. We later took this into account by adding a small epsilon $1e-6$ to the conditions mentioned in the equations 1 that need to be satisfied by the barycentric coordinates resulting in the figure 3

4) Observations: This method although simple to understand and implement, has 2 issues

- Artificial texture: the resulting images from this method look planar and seem very artificial in nature. It lacks the texture the original images has.
- Speed: This method is amazingly slow and cannot be implemented in real time and definitely not in edge devices. It took around 15sec to perform face swapping on an intel i7 10 core desktop. This could be because of latencies in python and the method overall

B. Face swapping using thin plate splines(TPS)

Triangulation assumes that we are doing affine transformation on each triangle. This is not the best way to do warping since the human face isn't planar. Some of the triangles are such that the 3 vertices are all in different planes in 3d space. The triangulation for these triangles isn't great and consequently the warping results aren't perfect either. Since the face has a very complex and smooth shape, we use thin plate splines to approximate its variations.

We compute a TPS that maps from the feature points in one face to the corresponding feature points in the other face. Also, we need to do this for two splines, one for the x coordinate and one for the y.

The strategy for warping using TPS is as follows. The workflow assumes we are warping face one to two.

1) Obtained fiducial landmarks from dlib for both faces.
2) Model landmarks in face one as a parametric spline of the landmarks in face two.
3) Solve for the model and find the parameters.
4) Use the parameters to warp every point in a rectangle around face one to face two.
5) Grab a convex hull around the face for better face fit.
6) Repeat for face two.

The initial result from this vanilla workflow turned out pretty bad as shown in figure 8. The patch is broken in places probably as a consequence of the incorrect warping.

We thought this was the case because we were warping all the pixels within a rectangle around the face but using only the facial landmarks to get the parameters for warping. So, we included the rectangle corners and the centers of the rectangle sides as additional features in the fiducial landmarks set. This helped. It got us rid of the missing patch as seen in figure 9. But it was still nowhere close to perfect.

With everything else in place correctly, the radial basis function was one of the potential reasons why the warping could have gone wrong. We started testing out different ideas here. At first we changed the log function to bases 10 and 2 with not much difference in the result. We used an L2 norm in place of the L1 norm but that also wasn't promising and so we tried changing the function itself and went with the classic gaussian as the first choice. The differences in these RBFs can be seen visually in figure 10.

The results of changing the RBF to gaussian were quite promising as can be seen in figures 11 and 12 for the two faces.

The very evident next step from these results was to make the warping fit to the face since the rectangle looked very unnatural. To overcome this, we found the convex hull of the facial landmarks and created a binary mask from it to only consider the warped points within this mask. As can be seen in figures 13 and 14, we get better face fit for the warpings using the convex hull trick.

1) Observations: One of the main observation of TPS compared to triangulation method is that it's fast! Every step of the pipeline is well vectorized and results are easily evident. On the same platform as earlier, it took around 0.5sec to produce the output. It is still not close to real time but it is sufficiently good to take it forward for optimization and apply it for non critical applications.

Another observation is that, it is comparatively very smooth and has a nice texture. However, it is still not stable and has a wavy output in the videos. This could be because of the choice of gaussian RBF. Need to experiment with better RBF models

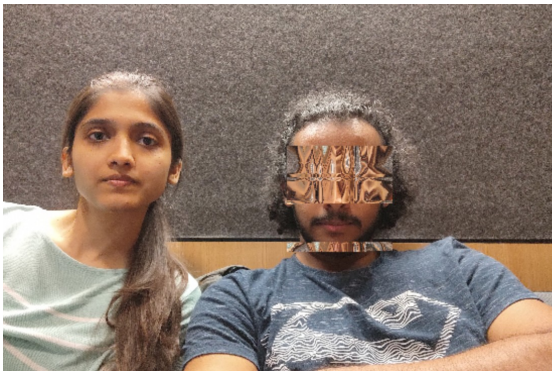


Fig. 8. Initial results: TPS

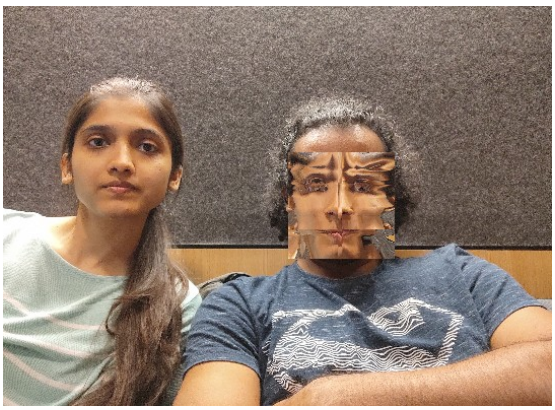


Fig. 9. On incorporating centers of rect sides

II. Blending

After swapping the faces, we need to blend the images to make it look more natural to the scene. We used OpenCV's poisson blending available in `seamlessClone` functionality. To this end, we took a step back and instead of modifying the

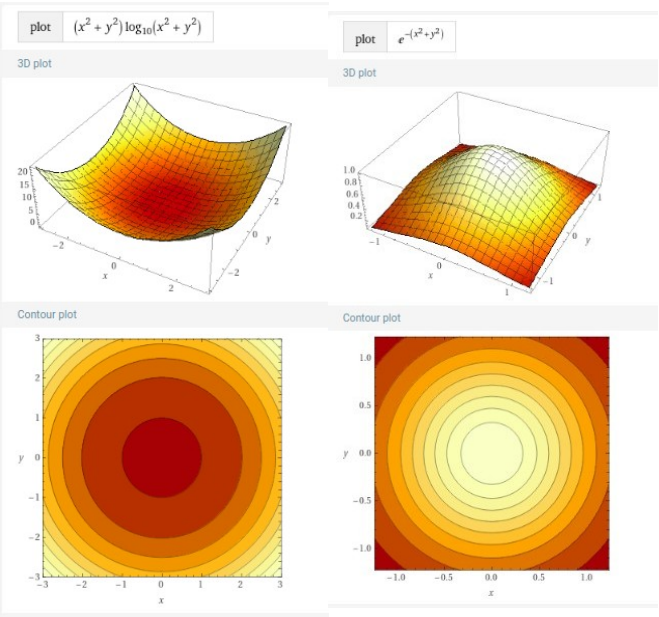


Fig. 10. Log and Gaussian as Radial Basis Functions

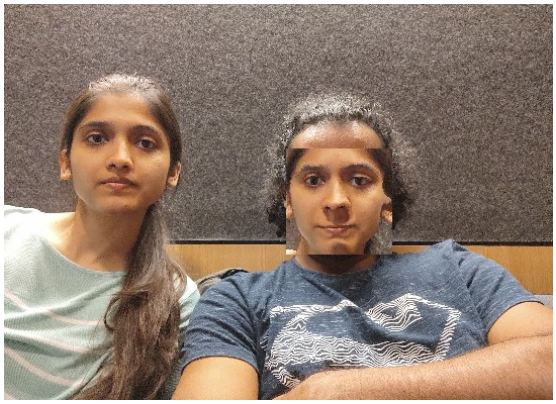


Fig. 11. Gaussian RBF on face 1

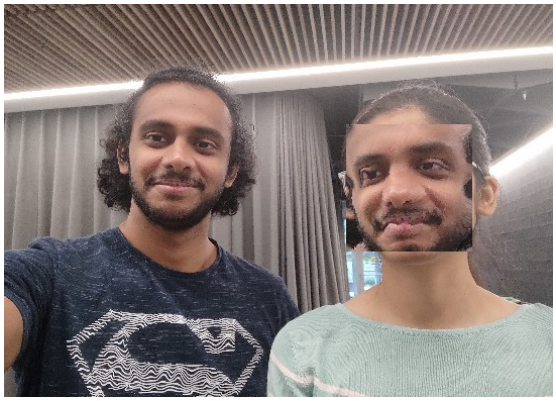


Fig. 12. Gaussian RBF on face 2

Fig. 13. On incorporating convex hull for both faces 1

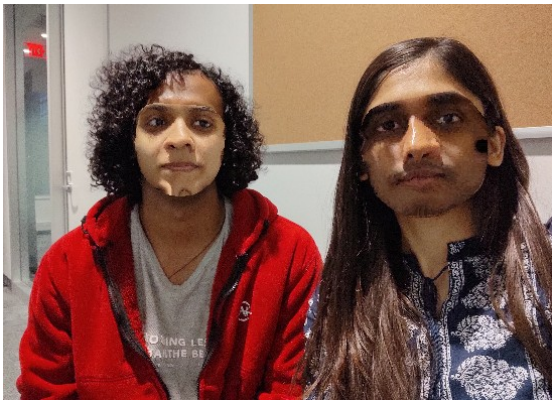
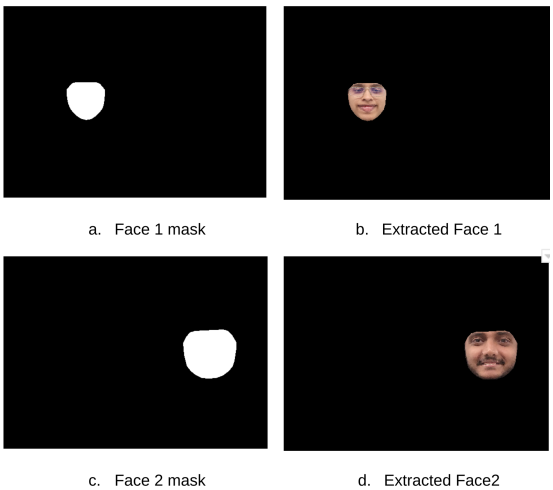


Fig. 14. On incorporating convex hull for both faces 2

images directly, we created masks of the images that needs to blended together. Figure 15 shows the inputs to the blending operation and 16 shows the actual frame and finally 2 shows the output the frame.



a. Face 1 mask b. Extracted Face 1

c. Face 2 mask d. Extracted Face2

Fig. 15. Blending Inputs

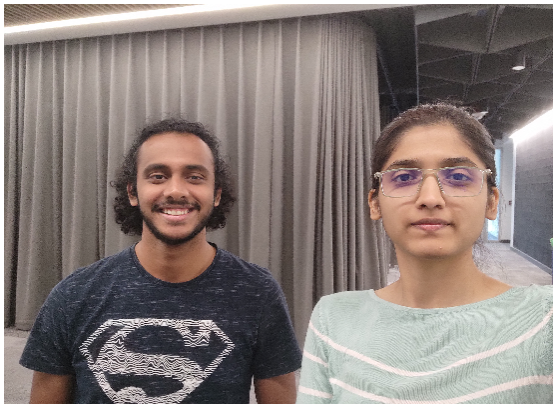


Fig. 16. Input image before blending

REFERENCES

[1] F. L. Bookstein, "Principal warps: thin-plate splines and the decomposition of deformations," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 11, no. 6, pp. 567-585, June 1989, doi: 10.1109/34.24792.