CMSC733: Project 2 - FaceSwap

Khoi Viet Pham - Gnyana Teja Samudrala
Email: khoi@terpmail.umd.edu - sgteja@terpmail.umd.edu
Use 1 late day

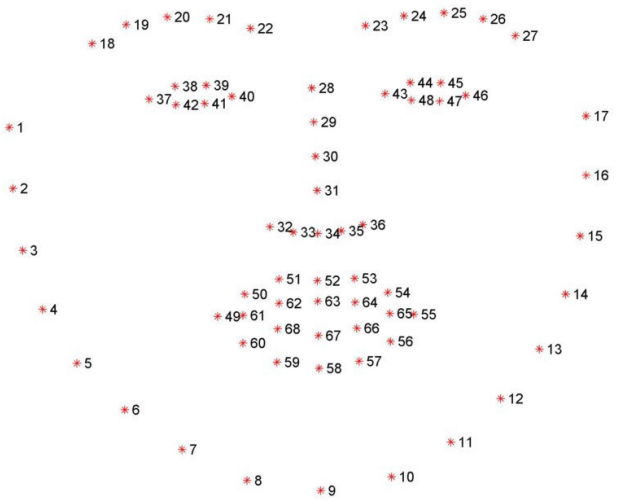

Fig. 1. Order of facial fiducials from dlib library (source: [1])

I. Phase 1: Traditional Approach

In this section, we will present in details our approach to perform Face Swap in 2 scenarios: (1) between two faces in the same video, (2) a face in video with a face in an image. The video is read as frames and operation is perfomed on each frame individually. In the following subsections, we will describe our pipeline and present the output on two frames of each video for every step.

We mainly follow the steps listed in the instructions. The algorithm consists of 3 main steps: (1) Facial Landmarks Detection, (2) Face Warping (using (a) Triangulation or (b) Thin Plate Spline), and (3) Blending.

A. Facial Landmarks Detection

The first step in the algorithm is to detect facial fiducials/landmarks. This is done using the "dlib" library. Using this library we can obtain the location (x, y) of 68 keypoints of the detected face. The points are identified using a pre-trained model. It returns the points in the order shown in fig. 1. The output of this step can be seen from fig. 2 to 5.

B. Warping using Triangulation

In this step, we will use the detected facial landmarks to create the Delaunay Triangles. This algorithm tries to maximize the smallest angle in each triangle, and from this, we can obtain the same triangulation between the source and the target face image. This solves the correspondance problem.

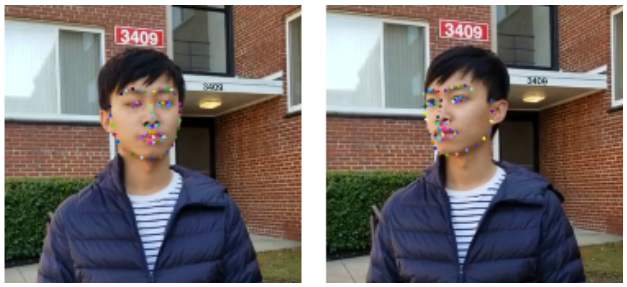

Fig. 2. Facial landmarks detected using dlib library on one frame in video Data1.mp4.

Fig. 3. Facial landmarks detected using dlib library on a face image of Jay Chou (a singer) (source: [2]), which we use to morph with the face in video Data1.mp4.

Fig. 6 to 9 show some examples of the triangulation step performed on some images.

Now we warp the triangles from the target face to the source face. This inverse warping makes sure that there are no spots left untouched. We use barycentric coordinates to make sure that the points lie inside the triangle. We then iterate over all pixels in the target face, use interpolation to find the

Fig. 4. Facial landmarks detected using dlib library on face 1 in video Data2.mp4.

Fig. 5. Facial landmarks detected using dlib library on face 2 in video Data2.mp4.

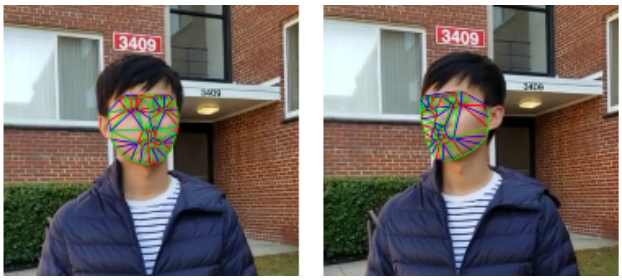

Fig. 6. Triangulation performed on one face in video Data1.mp4.

corresponding pixel and copy its color from the source face. We also would like to discuss our implementation to iterate over all pixels in the target face here.

Our goal is that: for each pixel in the target face, we have to find the triangle in the target face's triangulation that contains it. Simply using two nested for loop, one loop for the pixel and one loop to find its triangle, is not efficient. Let N be the total number of pixels in the face and M be the total number of triangles, this approach has time complexity $O(N \times M)$. Therefore, we propose an algorithm (which we implement as function *triangle_interior_iterator* in file *helper.py*) to efficiently iterate over all face pixels and at the same time know what triangle contains it, with time complexity only $O(N)$. The idea is that: for each triangle, we can find the minimum and maxium y-coordinate of its vertices; then, we use a line-sweep approach to iterate a horizontal line from y_{min} to y_{max}; for each horizontal line position, we can find its intersection points with any 2 sides of the triangle; from these 2 intersection points, we will know the minimum and maximum x-coordinate so that we can iterate over all pixels on this horizontal line (more details can be found in the code). This approach greatly improves the running time of our pipeline.

The output of the warping step can be seen from fig. 10 to 12.

C. Thin Plate Spline

The output of the warping step using Thin Plate Spline instead of Barycentric coordinates can be seen from fig. 13 to 15 .

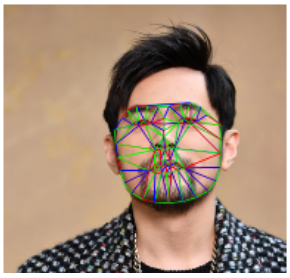

Fig. 7. Triangulation performed on an image of Jay Chou (source: [2]), which is then used to morph with face in video Data.mp4.

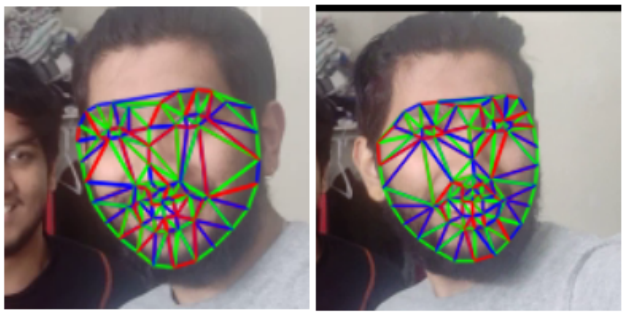

Fig. 8. Triangulation performed on face 1 in video Data2.mp4.

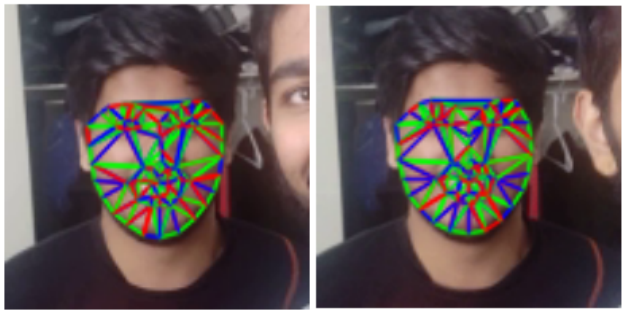

Fig. 9. Triangulation performed on face 2 in video Data2.mp4.

Fig. 10. Warping using Barycentric coordinates performed on face in video Data1.mp4 with the face of Jay Chou.

Fig. 11. Warping using Barycentric coordinates performed on face 1 in video Data2.mp4 with face 2 in the same video.

Fig. 12. Warping using Barycentric coordinates performed on face 2 in video Data2.mp4 with face 1 in the same video.

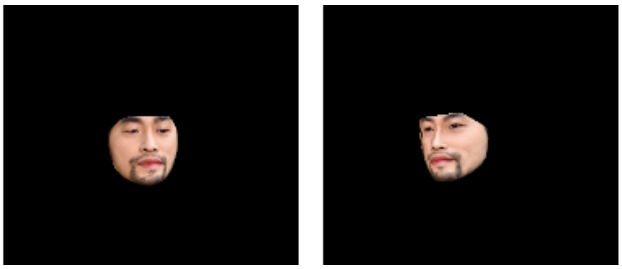

Fig. 13. Warping using Thin Plate Spine performed on face in video Data1.mp4 with the face of Jay Chou.

Fig. 14. Warping using Thin Plate Spine performed on face 1 in video Data2.mp4 with face 2 in the same video.

Fig. 15. Warping using Thin Plate Spine performed on face 2 in video Data2.mp4 with face 1 in the same video.

Fig. 16. Result after blending on one frame in Data1.mp4, warped using triangulation.

D. Blending

The face which is overlapped over the target image is to be blended into it, to make it look natural such as to match the skin tone, lighting conditions etc. We have written our own implementation for Poisson Blending (in the function *insert_face* in *helper.py*). Even though our implementation produced nice results, its running speed is not as efficient as the seamlessClone function which is built in OpenCV (which is obvious, but we still want to try implementing Poisson blending ourselves to get a feeling of how it works). Therefore, we move on to use seamlessClone in our final implementation. We have one interesting observation when using Poisson blending. In order to get more visually appealing result, before applying Poisson blending, it is better to perform erosion (a well-known image processing operation) to reduce the size of the mask a little. This helps the mask totally lie inside the interior of the face, which helps Poisson blending algorithm to better mimic the actual color of the target face. Otherwise, if we do not do this, Poisson blending will mimic the color of the background, which might not reflect correctly the color of the target face.

The final output of this step can be seen in the images from fig. 16 to 19.

II. Phase 2: Deep Learning Approach

We used the code [3] from the paper given [4] in the instructions to obtain the facial fiducials using deep learning. This output of the facial landmarks provided by the network is presented in the images from fig. 20 to 23. This is used


Fig. 17. Result after blending on one frame in Data1.mp4, warped using Thin Plate Spline.

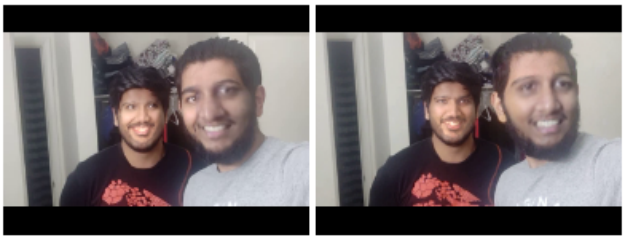

Fig. 18. After doing blending operation on Data2 warped using Triangulation

to perform the face swap by using the Triangulation and Thin Plate Spline algorithm implemented above. The results of the whole deep learning pipeline can be seen from fig. 20 to 37.

III. Results on Test Set

We demonstrate our results on the test set from fig. 38 to 46. We notice several problems with our results on the test set:

- PRNet does not produce correct facial landmarks locations, which result in the face warping incorrectly (fig. 40, fig. 43, and fig. 46).
- Faces not detected in some frames. For example: in video Test2.mp4, when the two people look at each other, their faces are not in frontal pose anymore and thus our frontal face detector dlib failed to detect them; in video Test3.mp4, some frames are very dark and our detector cannot locate the face. One other interesting observation is that: in our implementation, PRNet uses a different face detector which can even find the face that is not in frontal pose; this is opposed to the face detector we use

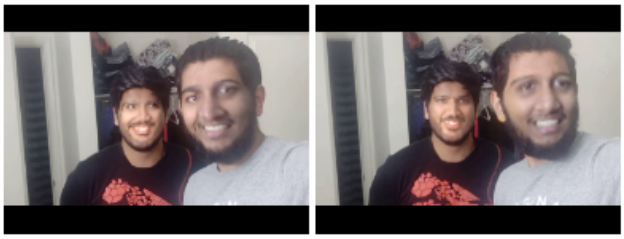

Fig. 19. After doing blending operation on Data2 warped using Thin Plate Spline


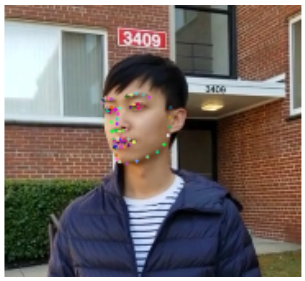

Fig. 20. Facial landmarks detected using PRNet on one frame in video Data1.mp4.

Fig. 21. Facial landmarks detected using PRNet on Image which will be morphed with Data1 face

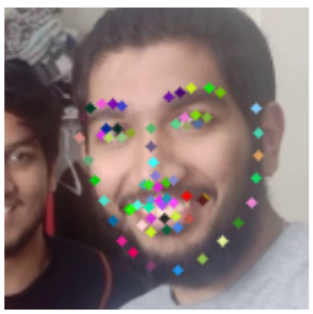
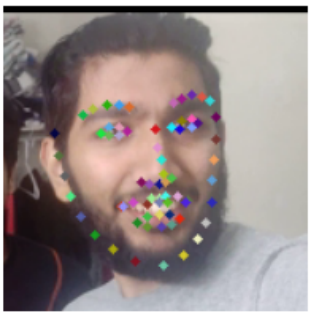

Fig. 22. Facial landmarks detected using PRNet on Data2, face1

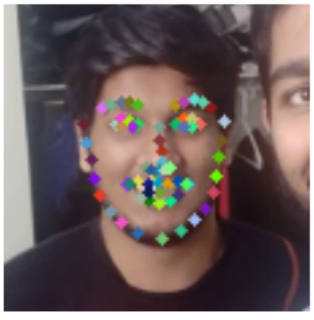
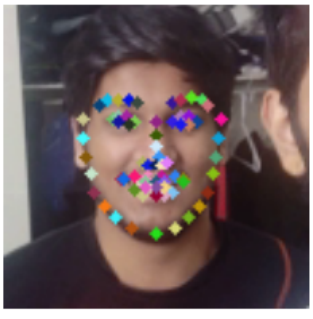

Fig. 23. Facial landmarks detected using PRNet on Data2, face2

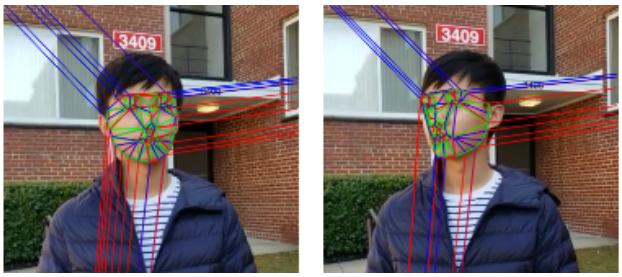

Fig. 24. Triangulation performed on video Data1.mp4 using landmarks from PRNet.

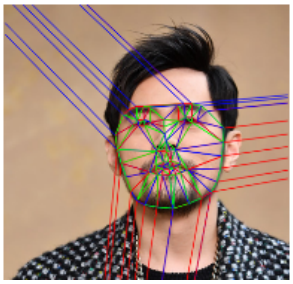

Fig. 25. Triangulation performed on image of Jay Chou which will be morphed with Data1.mp4, using landmarks from PRNet.

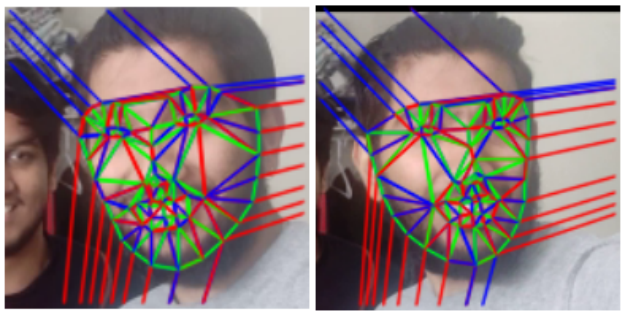

Fig. 26. Triangulation performed on Data2, face1 using landmarks from PRNet.

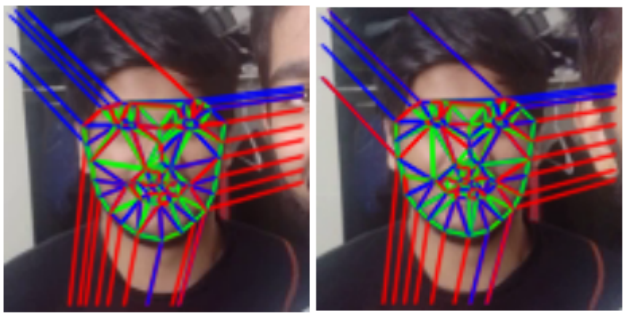

Fig. 27. Triangulation performed on Data2, face2 using landmarks from PRNet.

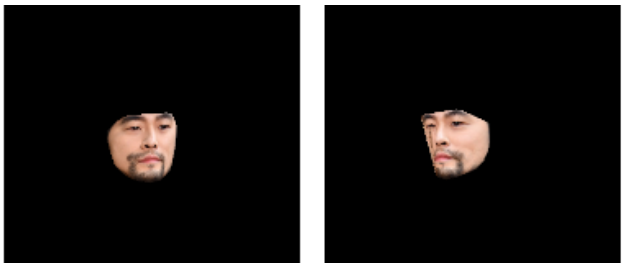

Fig. 28. Triangulation warp using Barycentric coordinates performed on Data1 using landmarks from PRNet.

Fig. 29. Triangulation warp using Barycentric coordinates performed on Data2, face1 using landmarks from PRNet

Fig. 30. Triangulation warp using Barycentric coordinates performed on Data2, face2 using landmarks from PRNet

Fig. 31. Thin Plate Spine performed on Data1 using landmarks from PRNet.

Fig. 32. Thin Plate Spine performed on Data2, face1 using landmarks from PRNet.

Fig. 33. Thin Plate Spine performed on Data2, face2 using landmarks from PRNet.

Fig. 34. After doing blending operation on Data1 warped using Triangulation with landmarks from PRNet.

Fig. 35. After doing blending operation on Data1 warped using Thin Plate Spline with landmarks from PRNet.

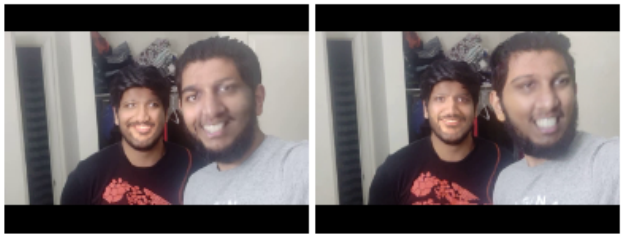

Fig. 36. After doing blending operation on Data2 warped using Triangulation with landmarks from PRNet.

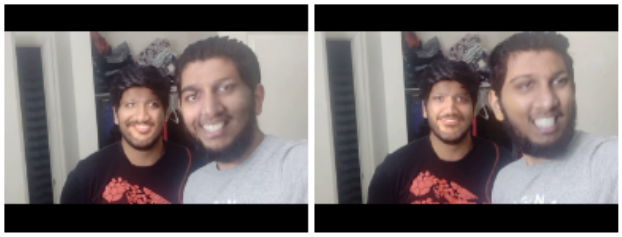

Fig. 37. After doing blending operation on Data2 warped using Thin Plate Spline with landmarks from PRNet.

for triangulation and thin plate spine, which can only find face in frontal pose.

- Thin plate spline often produces artifacts. These might be the results of not applying enough regularization when solving the linear system of equations using least squares.

REFERENCES

[1] I. Jos, "Facial mapping (landmarks) with dlib + python." [Online]. Available: https://towardsdatascience.com/facial-mapping-landmarks-with-dlib-python-160abcf7d672

[2] "Image of jay chou." [Online]. Available: http://www.sohu.com/a/294739527_114988

[3] 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. 38. Results on video Test1.mp4, using triangulation with dlib facial landmarks.

Fig. 39. Results on video Test1.mp4, using Thin Plate Spline with dlib facial landmarks.

Fig. 40. Results on video Test1.mp4, using triangulation with PRNet facial landmarks.

Fig. 41. Results on video Test2.mp4, using triangulation with dlib facial landmarks.

Fig. 42. Results on video Test2.mp4, using Thin Plate Spline with dlib facial landmarks.

Fig. 43. Results on video Test2.mp4, using triangulation with PRNet facial landmarks.

Fig. 44. Results on video Test3.mp4, using triangulation with dlib facial landmarks.

Fig. 45. Results on video Test3.mp4, using Thin Plate Spline with dlib facial landmarks.

Fig. 46. Results on video Test3.mp4, using triangulation with PRNet facial landmarks.

[4] ——, "Joint 3d face reconstruction and dense alignment with position map regression network," *CoRR*, vol. abs/1803.07835, 2018. [Online]. Available: http://arxiv.org/abs/1803.07835