Face Morphing using OpenCV

Atul Gadhiya
Technische Hochschule Deggendorf Cham,
Germany
atul.gadhiya@stud.th-deg.de

***Abstract*—This project explores the implementation of a face morphing algorithm, leveraging computer vision techniques. The algorithm seamlessly transforms one face into another, demonstrating the potential of image processing in creating realistic morphing effects.**

***Index Terms*—Face-Morphing, Python(OpenCV) Libraries, Triangulation.**

I. Introduction

Image morphing was first used extensively in the movie Willow using a technique developed at Industrial Light and Magic. A scene from the movie is shown below.

The idea behind Image Morphing is rather simple. Given two images I and J we want to create an in-between image M by blending images I and J. The blending of images I and J is controlled by a parameter that is between 0 and 1 (0 ¡= alpha ¡= 1). When alpha is 0, the morph M looks like I, and when is 1, M looks exactly like J. Naively, you can blend the images using the following equation at every pixel (x,y).

M(x,y) = (1 - alpha)I(x,y)+ alphaJ(x,y)

However, using the above equation to generate a blend between the image of Secretary Hillary Clinton and Senator Ted Cruz with set to 0.5, you will get the following terrible results.

The resulting image is disturbing, but it also screams a solution at you. It begs you to somehow align the eyes and the mouth before blending the images. You get similar disturbing results when you try to blend two different political ideologies without first aligning the minds, but I digress.

So, to morph image into image I into J We need to first establish pixel correspondence between the two images. In other words, for every pixel (xi, yi) in image I, we need to find it's corresponding pixel (xj, yj) in image J. Suppose we have magically found these correspondences, we can blend the images in two steps. First, we need to calculate the location (xm, ym) of the pixel in the morphed image. It is given by the following equation 1

xm = (1 - alpha)xi + alpha xj
ym = (1 - alpha)xj + alpha yj

Second, we need to find the intensity of the pixel at using the following equation 2

M(xm,ym) = (1 - alpha)I(xi, yi) + alpha J(xj, yj)

That's it. We are done. Now, Let's go and vote for Trump. Kidding! Just like Trump, I left out some important details. Finding a corresponding point in image J for every pixel in image I is about as difficult as building a 10 ft wall between the United States and Mexico. It can be done, but it is expensive and not really necessary.

But it is very easy to find a few point correspondences. For morphing two dissimilar objects, like a cat's face and a human's face, we can click on a few points on the two images to establish correspondences and interpolate the results for the rest of the pixels. We will next see how Face Morphing is done in detail, but the same technique can be applied to any two objects.

The project utilizes facial landmark detection, triangulation, image warping, and alpha blending to achieve smooth transitions between source and target faces. Python, along with OpenCV and NumPy libraries, is employed for the implementation.

II. Software/Package Requirement

Python(version 3.11)

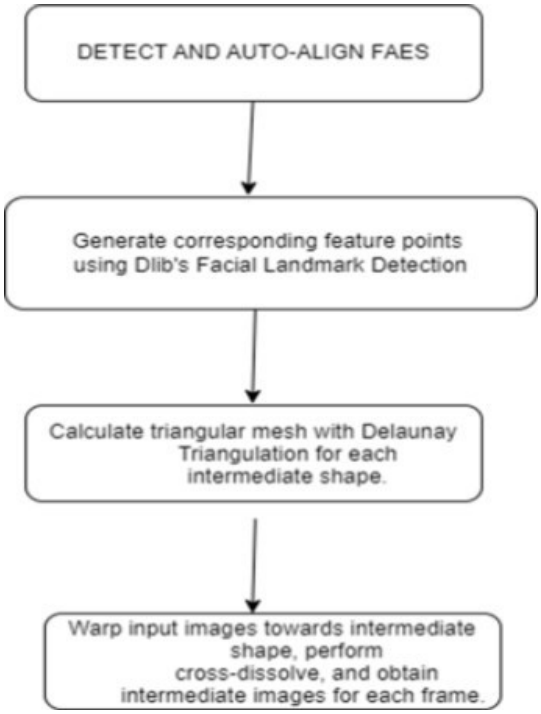
OpenCV

NumPy

dlib library

Matplotlib (for visualization)

III. Flowchart



IV. Step-by-Step Instructions:

Morphing two faces can be done using the following steps. For simplicity, we will assume the images are of the same size, but it is not a necessity.

A. Find Point Correspondences using Facial Feature Detection

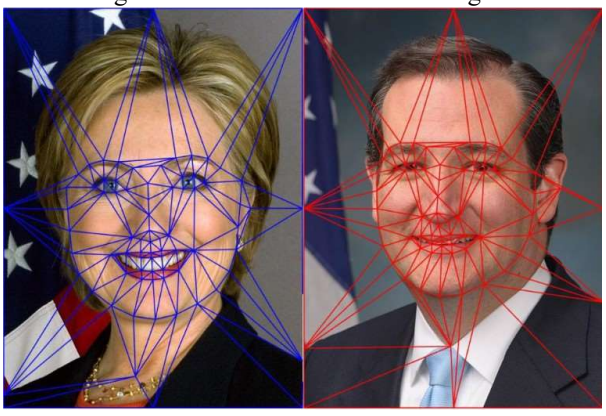
Let's start by obtaining corresponding points. First, we can get a lot of points by automatically (or manually) by detecting facial feature points. I used dlib to detect 68 corresponding points. Next, I added four more points (one on the right hand side ear, one on the neck, and two on the shoulders). Finally, I added the corners of the image and half way points between those corners as corresponding points as well. Needless to say, one can add a few more points around the head and neck to get even better results, or remove the manually clicked points to get slightly worse (but fully automatic) results.

B. Delaunay Triangulation

From the previous step we have two sets of 80 points — one set per image. We can calculate the average of corresponding points in the two sets and obtain a single set of 80 points. On this set of average points we perform Delaunay Triangulation. The result of Delaunay triangulation is a list of triangles represented by the indices of points in the 80 points array. In this particular case the triangulation produces 149 triangles connecting the 80 points. The triangulation is stored as an array of three columns. The first few rows of the triangulation is shown below.

Triangulation Points
38 40 37
35 30 29
38 37 20
18 37 36
33 32 30
...

It shows that points 38, 40 and 37 form a triangle and so on. The triangulation is shown on the two images below.



C. Warping images and alpha blending

We are now in a position to intelligently blend the two images. As mentioned before, the amount of blending will be controlled by a parameter . Create a morph using the following steps.

Find location of feature points in morphed image : In the morphed image , we can find the locations of all 80 points (xm, ym) using equation (1).

Calculate affine transforms : So we have a set of 80 points in image 1, another set of 80 points in image 2 and a third set of 80 points in the morphed image. We also know the triangulation defined over these points. Pick a triangle in image 1 and the corresponding triangle in the morphed image and calculate the affine transform that maps the three corners of the triangle in image 1 to the three corners of the corresponding triangle in the morphed image. In OpenCV, this can be done using getAffineTransform . Calculate an affine transform for every pair of 149 triangles. Finally, repeat the process of image 2 and the morphed image.

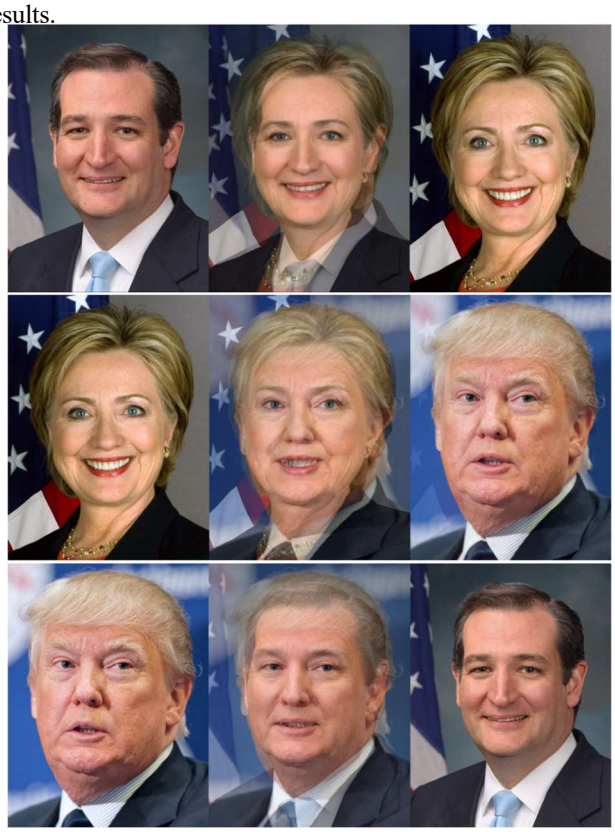
Warp triangles : For each triangle in image 1, use the affine transform calculated in the previous step to transform all pixels inside the triangle to the morphed image. Repeat this for all triangles in image 1 to obtain a warped version of image 1. Similarly, obtain a warped version for image 2. In Open CV this is achieved by using the function warp-Affine. However, warp-Affine takes in an image and not a triangle. The trick is to calculate a bounding box for the triangle, warp all pixels inside the bounding box using warp-Affine, and then mask the pixels outside the triangle. The triangular mask is created using fillConvexPoly. Be sure to use blend Mode BORDER_REFLECT_101 while using warp-Affine. It hides the seams better than Secretary Clinton hides her emails.

Alpha blend warped images : In the previous step we obtained warped version of image 1 and image 2. These two images can be alpha blended using equation (2), and this is your final morphed image. In the code I have provided warping triangles and alpha blending them is combined in a single step.

V. Face Morphing Results

The results of applying the above technique are shown below. The image in the center is a 50% blend of the image on the left and the right. The video on the top of this page shows an animation with different alpha values. Animation is a cheap trick that hides a lot of flaws in a morph; Senator Ted Cruz would love it.

Most facial features are very well aligned. The part of the image outside the face is not so well aligned because we have fewer corresponding points in that region. One can manually add additional points to fix misalignment's and get better results.



VI. Summary

This document delves into the implementation of a face morphing algorithm, utilizing computer vision techniques to seamlessly transform one face into another. The process begins with a brief history of image morphing, dating back to its extensive use in the movie "Willow." The core concept involves blending two images to create an in-between image, controlled by a parameter. However, the document emphasizes the need to align features such as eyes and mouth before blending images to avoid disturbing results. Establishing pixel correspondence between the two images is crucial, and while finding a corresponding point for every pixel is challenging, it's easier to find a few point correspondences, especially for morphing dissimilar objects. The software/package requirements for face morphing include Python (version 3.x), OpenCV, NumPy, dlib library, and Matplotlib for visualization.

The step-by-step instructions for morphing two faces involve finding point correspondences using facial feature detection, followed by Delaunay triangulation to obtain a single set of points and warping images. The process includes calculating

affine transforms, warping triangles, and alpha blending the warped images to generate the final morphed image. The resulting morphed images showcase well-aligned facial features, while the document acknowledges the need for additional points to fix misalignment's in the non-facial regions. The project also mentions the potential application of the same technique to morph any two objects, not limited to human faces.

In summary, the document provides a comprehensive overview of the face morphing algorithm, emphasizing the importance of pixel correspondence and feature alignment to achieve realistic morphing effects. It also outlines the necessary steps and software requirements for implementing the algorithm, offering insights into the intricacies of image processing and computer vision techniques. The practical application of the technique and the potential for extending it to morphing different objects further enrich the scope of the project, highlighting its versatility beyond facial morphing.

REFERENCES

[1] https://www.researchgate.net/publication/280929662_Image_Morphing_A_Literature_Study.

[2] https://www.researchgate.net/publication/270018791_Comparative_Study_of_Triangulation_based_and_Feature_based_Image_Morphing.

[3] https://ijcatr.com/archives/volume3/issue11/ijcatr03111010.pdf.

[4] https://www.researchgate.net/publication/369905336_Image_Morphing_Techniques_A_Review.