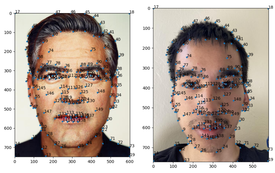
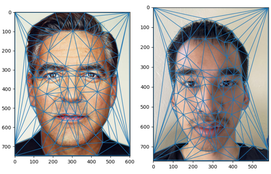
Defining Correspondences

To morph one image to another, I would need to define keypoints in both images such that one image's keypoints map to the other. I used python's ginput function from matplotlib to define the keypoints:

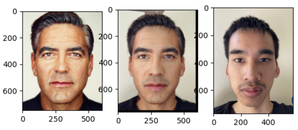


In selecting the keypoints, I had to make sure that close keypoints were not colinear to ensure that there was a triangulation. Also important was ensuring that the resulting triangles well partitioned the image such that no two different elements were in the same triangle. For example, if a triangle corresponding to the first image captured some of the hair and the background while the other did not, a fragment of the hair would morph inconsistantly with the rest of the hair. Finally, keypoints were set in the corners to ensure that the entire image was morphed and not just the head.



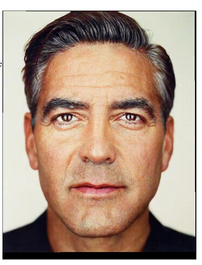
Computing the Midway Face

To get the midway face, the average of the keypoints were first calculated to determine the keypoints on the midway face. Delaunney triangulation gave triangles that indexed into both the keypoints for the midway face along with the keypoints for the two images to be morphed. Using this, the affine transformation for mapping the midway triangle's vertices on each of the two image's triangle's vertices could be computed. This allowed me to pull the rgb pixel values from both the source images, interpolated, to set in the midway image's triangle. As for the details of the implementation, I created an `inverseWarp(source, sourceTri, dest, destTri, color=True)` function which uses the pixel values in the source image located at the sourceTriangle. Using scipy's griddata method, I used this data to interpolate the values for the destination triangle. With skimage's polygon method, I could get the indices for the destination triangle `destRR, destCC = polygon(destTri)` such that `dest[destRR, destCC] = res`, where `res` is the interpolated data. I used the `inverseWarp` function over all the simplices in the triangulation to get two morphs from the original images to the midway face.



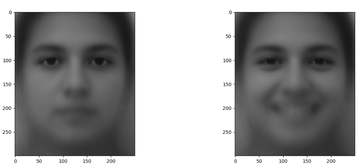
The Morph Sequence

All that is left is to calculate frames between image1 and the midway face and between the midway face and image2 to get a morph sequence from image1 to image2. This was done by first setting the start and target images for the current frame such that the current frame's image was somewhere between the start and target images. Then I used linear interpolation to determine where the keypoints lie in the current frame. This is determined by `warpFrac` which lies between 0 and 1, 0 being the start of the morph and 1 being the end. Then the current keypoints are given by `startKp + warpFrac * (targetKp - startKp)` if `warpFrac < 0.5`. If `warpFrac > 0.5`, I used `2 * warpFrac - 1`. After warping the start and target images to the new keypoints, a dissolve fraction was used to determine the blend of the two results for the current frame in a similar manner to the warp fraction.

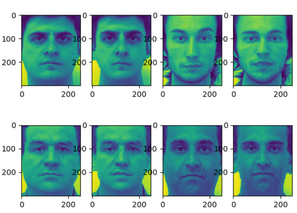


Population Mean Face

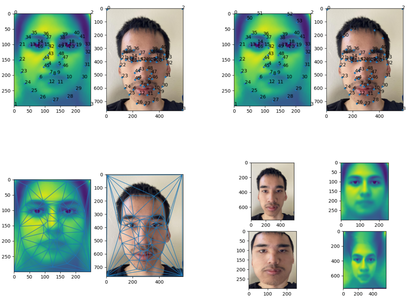
To get the mean face, I used the faces from the FEI database. Getting the mean face is adding all the images and dividing by the number of images.



To morph a given face, f, and its keypoints, f_k, to the average face, I used the same method for computing the midway face. Using the average keypoints computed for the average face, I got data for it by interpolating from the data given on the triangles `f[f_k[simplex]]`. Here are some results, where the images are shown in pairs. The left image is the original and the right is the image morphed to the average:



Using the annotatations for the image, I labelled my face with keypoints in the same order. I noticed that there were some keypoints missing for the top of the forehead, so I added them to get more accurate results. Here are pictures of my face morphed to the average and the average morphed to my face:



Caricature

To get a caricature I extrapolated data from the mean. I took a difference `myface - mean = diff` and for `alpha > 1` or `alpha <0` , I computed `alpha * mean + (1 - alpha) * myface`.