CS 166 Final Report

Tyrone McNichols

June 2022

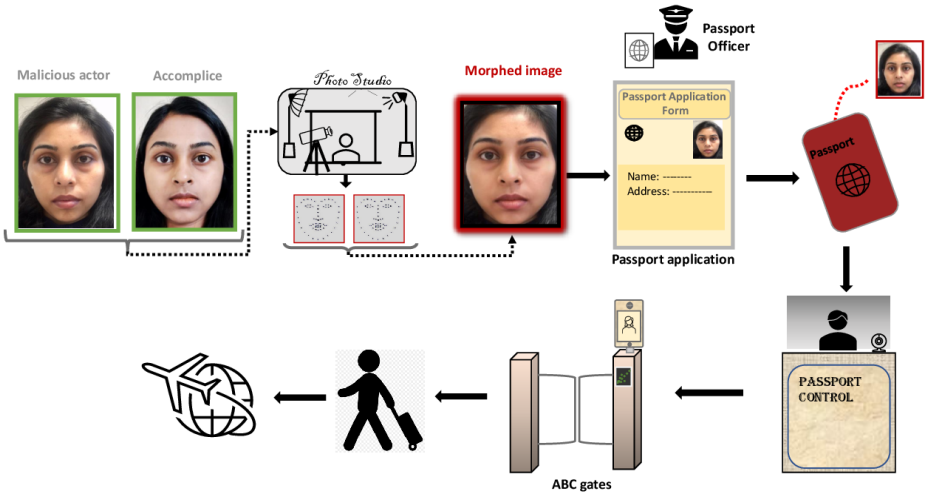

Figure 1: Example Morphing Attack from [Venkatesh et al., 2021]. In this example, two faces are combined to create a fake passport image, allowing malicious parties to navigate past border security.

1 Introduction

In this project, I explore image morphing, that is the process of morphing an image of one face into another. Such tasks have become increasingly relevant in recent years, as with the rise of facial recognition software, the desire to avoid recognition has risen in turn. In particular, the prevalence of morphing attacks has risen. In a morphing attack, the images of two faces are combined and the resulting morphed facial image is then presented during registration as a biometric reference [Scherhag et al., 2022]. An example of such an attack is shown in Figure 1. As a result, a number of scientists are researching the technology to create better morphed image detection. While I will not be investigating detection in this project, I do implement my own image morphing software[1].

[1]The implementation is available at https://github.com/Tyronitar/cs166-final-project.

2 Methods

In this project, I implement landmark-based image morphing. This method identifies corresponding landmarks in both images, aligns them by morphing the images, and then blends the two images [Venkatesh et al., 2021]. Specifically, I implement the method presented in [Beier and Neely, 1992] to transform a source image into a destination image. In this paper, a pair of corresponding lines (one in the source image I_0, one in the destination image I_1) is identified, defining a mapping from one image to the other. This resulting map is applied to all pixels in the source image. This method is then extended to use multiple pairs of lines, where the resulting transformation then becomes a weighted average of the contributions from each pair.

2.1 Algorithm

Given a pair of lines PQ and $P'Q'$ (in the destination I_1 and source images I_0 respectively), we compute the coordinate mapping from coordinates X in the destination image to X' in the source image as follows:

$$u = \frac{(X-P)\cdot(Q-P)}{\|Q-P\|^2} \tag{1}$$

$$v = \frac{(X-P)\cdot(Q-P)_\perp}{\|Q-P\|} \tag{2}$$

$$X' = P' + u\cdot(Q'-P') + \frac{v\cdot(Q'-P')_\perp}{\|Q'-P'\|} \tag{3}$$

where $\perp$ indicates the vector that is perpendicular of the same length. Here, u, v are scalars, while X, X', P, P', Q, Q' are all vectors.

To use multiple lines, we use each pair of lines P_iQ_i to calculate a X_i'. We use this to compute a displacement from the original position $X_i' - X$. Then X' is computed by applying the weighted average of the displacements. The weighting function used is

$$w_i = \left(\frac{\|Q_i - P_i\|_2^p}{a + d(X, P_iQ_i)}\right)^b \tag{4}$$

where $d(X, P_iQ_i)$ indicates the distance between the pixel X and the line. The parameters a, b, p can be altered to change relative effects from various lines. a balances smoothness of the transformation and control of where pixels go. A small value of a means points very close to lines will be very strongly affected by the line, ensuring that those pixels go where the user intends. b determines how weight falls off with distance. p determines how line length is factored into the weight. For my implementation, I used $a = 1$, $b = 1$, and $p = 0.5$. The full algorithm becomes:

1. For each pixel X:
   (a) For each line P_iQ_i:
      i. Calculate X_i' using Equations 1, 2, and 3.
      ii. Compute the displacement $D_i = X_i' - X$.
      iii. Compute the weight w_i using Equation 4.
   (b) Set $X' = X + \frac{\sum_i D_i \cdot w_i}{\sum_i w_i}$
   (c) Set $I_1(X) = I_0(X')$

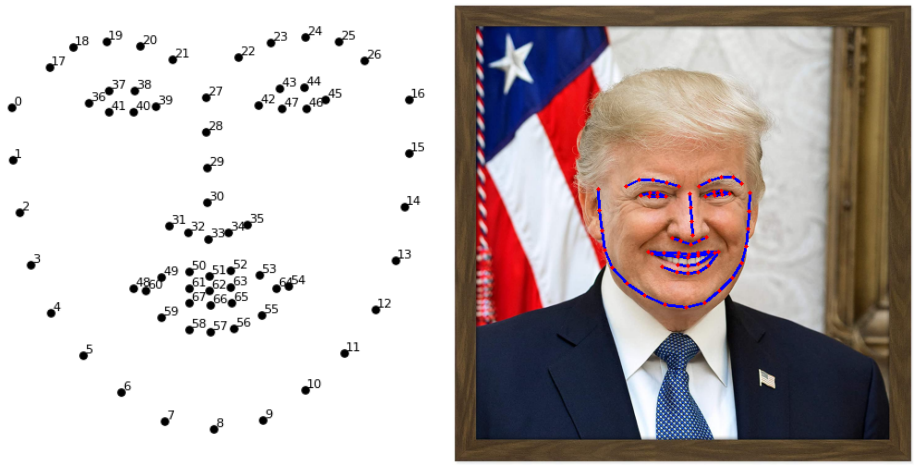

Figure 2: Face landmarks and corresponding feature lines. `dlib` outputs 68 landmarks (Left), which are then used to create the feature lines (Right) by connecting consecutive pairs of points.

2.2 Implementation

To implement the algorithm from [Beier and Neely, 1992], one first needs a way to identify the feature line segments in faces. To that end, I utilized the 68 face landmark detector in `dlib` [King, 2009]. This model outputs 68 points to characterize the jawline, eyes, eyebrows, nose, and mouth of a human face (See Figure 2). Using these points, I created the feature lines by joining consecutive consecutive points corresponding to the same facial feature. For example, for all the jawline points 0 through 16, there is a line segment connecting each consecutive pair. An example using former president Donald Trump is shown in Figure 2.

One issue I ran into was that `dlib`'s landmark detector will only work on human faces it can detect. As a result, undetectable faces, or non-human faces would not work with the program. To accommodate this, I also allowed the user to provide manual annotation indicating the locations of the 68 landmarks in the image. Using this, I was able to combine a human face and non-human face in Figure 4.

To generate a smooth sequence of images morphing from image to another, I interpolate the lines between the source and destination position. Both images are then morphed to the intermediary positions, and then combined by crossfading.

3 Results

An example morph sequence from young Robin Williams to his older self is shown in Figure 3. Additional morph sequence examples are shown in Appendix A. Furthermore, gif versions of all of the morph sequences are available at the github repository[2].

The morph sequences are fairly smooth, with the only substantial artifacts coming from difference in hairstyles. Unfortunately, the [Beier and Neely, 1992] algorithm does not have a way to automatically remove artifacts such as these. Similarly, when subjects are very far removed from one another, or their faces are oriented different ways, artifacts become more pronounced. This is one of

[2]https://github.com/Tyronitar/cs166-final-project

Figure 3: Example morph sequence from a younger Robin Williams to an older one.

the major weakness of my implementation: it's heavily dependent on the images the user chooses. Two people facing different directions, or with wildly different backgrounds will result in ghosting. For the best results, one should use images where the subjects are facing forward with plain backgrounds. For further exploration, one might explore different methods for removing ghosts. For example, using some type of masking to only combine desired regions.

Another limitation of my implementation is that it only works on square images. This was a convenience for the programming, but for further work one could change that with relative ease. It's important however to ensure that the two target images have the same aspect ratio. The images must be resized to the same image before morphing, so if the aspect ratios differ the result will look strange.

My implementation is also much better in performance that the original paper, largely thanks to the improvement in computational power since then. In [Beier and Neely, 1992], they note that generating a frame of size 720x480 takes about 2 minutes per frame, whereas my implementation takes about 25 seconds per frame for a 720x720 image.

References

[Beier and Neely, 1992] Beier, T. and Neely, S. (1992). Feature-based image metamorphosis. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '92, page 35–42, New York, NY, USA. Association for Computing Machinery.

[King, 2009] King, D. E. (2009). Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10:1755–1758.

[Scherhag et al., 2022] Scherhag, U., Rathgeb, C., and Busch, C. (2022). *Face Morphing Attack Detection Methods*, pages 331–349. Springer International Publishing, Cham.

[Venkatesh et al., 2021] Venkatesh, S., Ramachandra, R., Raja, K., and Busch, C. (2021). Face morphing attack generation and detection: A comprehensive survey. *IEEE Transactions on Technology and Society*, 2(3):128–145.

A Additional Morphing Examples

Figure 4: Morph sequence from Barack Obama to a lion

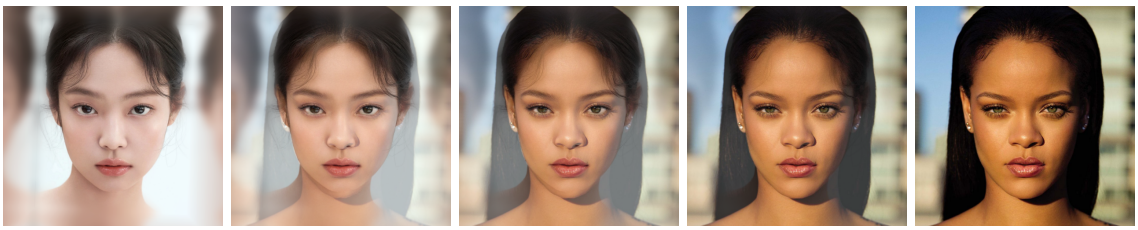

Figure 5: Morph sequence from Jennie Kim to Rihanna

Figure 6: Morph sequence from young Leonardo DiCaprio to his older self

Figure 7: Morph Sequence from Bob Ross to Barack Obama