An Introductory Algorithm to Face-Morphing Techniques through Affine and Bilinear Interpolation

By Connie Xu and Jayson Wu
MAT204 Spring 2020

I: INTRODUCTION AND MOTIVATION

Through our research on applications of linear algebra, we found multiple realms of uses, including but not limited to neuroscience, finance, and computer vision. After discussing and reviewing possible projects using Principal-Component Analysis, we decided we wanted to shift our attention towards today's online craze with social media trends, most notably those on the growing popular social media platform TikTok. TikTok is a video-sharing social media service that focuses primarily on short videos containing dances, lip-syncing, and comedy sketches. However, what interested us the most was this one new feature on TikTok that allowed people to morph their faces into other faces by simply inserting different pictures of people. The "Morph" photo template, as it is referred to on TikTok, allows users to insert 2-5 photos of anyone and proceeds to make a slideshow that starts with the first photo, morphs into the second photo, then to the third, and so on. The "Morph" template can take any type of background and identifies the face to morph automatically, allowing quick and easy results of what a morphed version of you or your friends would look like.

II: THEORY

We decided to tackle the same issue in our project, attempting to design from scratch an algorithm that can morph two photos together. After discussing the problem, we realized a few difficulties to overcome. First, the positions of the face pictures may not be the same, and therefore we need to define some type of algorithm to account for the differences in positions. Secondly, the face structures may not be the same, and a morphing of a longer face onto a smaller face would result in a possible image with four eyes, two mouths, and two noses. These concerns need to be addressed in our algorithm and after sketching out a basic algorithmic structure, we decided to split the project into three main parts: Feature-Finding, Interpolation, and Cross-Dissolving. Feature-finding refers to the identification of important features of the face, including the eyes and the mouth, so that the morphing may become more accurate. We used two methods of interpolation. The first is affine interpolation, which defines the mapping of one triangle onto another, and the second is bilinear interpolation, which defines the mapping of one quadrilateral onto another. We use these algorithms to map the eyes and mouth of one face onto the next face. Finally, cross-dissolving is a simple technique used to combine the pixels of two images into one, retaining a portion of the original color from both starting images. Through these three steps, we outline an introductory algorithm that can morph two images into one regardless of position within a picture.

An example of face morphing: Picture retrieved from en.wikipedia.org/wiki/Morphing

1: FEATURE-FINDING

We define the important features in a picture to be the two eyes and the mouth of a face. We must first pre-process an image to allow for more accurate results, which for our algorithm consists of centering the face on the x-axis of the picture and removing the noisy background. We were not able to automate the pre-processing step because that would require implementing a face-finder, which we felt was outside our coding abilities.

Secondly, we wish to identify the eyes. We propose that the eyes are the most complex portion of a face, so we define an energy function that finds the change in color for each pixel, having high values for big changes and small for mostly stagnant colors. Using this energy function, we loop through the picture and calculate the area with the largest complexity, also providing priority to pixels that are closer to the center of the picture. After finding the first eye, we can find the second eye through a reflection across the vertical line that splits the face in half, and then we store the points.

Then, to identify the mouth, we use an algorithm involving the RGB color scale. The mouth is often the area with the highest red color component, coupled with the lowest green color component. In other words, the mouth is the area of the face that most closely resembles the color red. After finding the mouth, we store the left and right endpoints of the mouth as well.

Finally, we store the edges of the picture as important points, as these can be helpful in the interpolation step.

Feature-Finding Diagram:

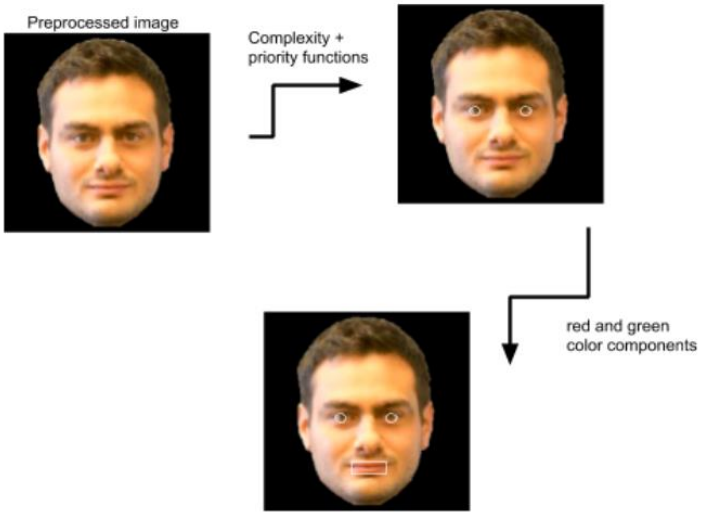
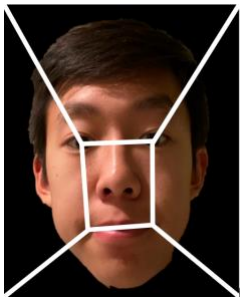


Image taken from:
https://www.math.princeton.edu/people/theodore-drivas

2: INTERPOLATION



We define 5 different quadrilaterals based on the four feature points. Our algorithm then morphs each quadrilateral from the first image into its proper shape on the second image through two different types of interpolations: affine interpolation and bilinear interpolation.

2a: Affine Interpolation

Affine interpolation serves to map a triangle to another triangle. Each of the 5 quadrilaterals indicated above can be split into two triangles, and separate transformations can be applied to each triangle to map the pixels. A transformation is affine if it can be expressed as the sum of a linear transformation and a translation: $T(u,v) = L(u,v) + w$.

Then, given a set of 3 coordinates (u_1, v_1), (u_2, v_2), (u_3, v_3), and a set of target points (x_1, y_1), (x_2, y_2), (x_3, y_3), the affine transformation can be expressed as

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} u_i \\ v_i \end{bmatrix} + \begin{bmatrix} \mathrm{e} \\ f \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \text{ where } L(u,v) = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \text{ and } w = \begin{bmatrix} \mathrm{e} \\ f \end{bmatrix}.$$

We then multiply and sum the matrices to get the following system of equations:

$$\begin{cases} au_1 + bv_1 + \mathrm{e} = x_1 \\ au_2 + bv_2 + \mathrm{e} = x_2 \\ au_3 + bv_3 + \mathrm{e} = x_3 \end{cases} \qquad \begin{cases} cu_1 + \mathrm{d}v_1 + f = y_1 \\ cu_2 + \mathrm{d}v_2 + f = y_2 \\ cu_3 + \mathrm{d}v_3 + f = y_3 \end{cases}$$

$$\begin{bmatrix} u_1 & v_1 & 1 \\ u_2 & v_2 & 1 \\ u_3 & v_3 & 1 \end{bmatrix}\begin{bmatrix} a \\ b \\ \mathrm{e} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \begin{bmatrix} u_1 & v_1 & 1 \\ u_2 & v_2 & 1 \\ u_3 & v_3 & 1 \end{bmatrix}\begin{bmatrix} c \\ d \\ \mathrm{f} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

Then, we solve for the values a, b, c, d, e, f by row-reducing the two matrices, and now we have an affine transformation with which we can use to map any other points within the quadrilateral to map them to their new location.

2b: Bilinear Interpolation

Bilinear interpolation is another linear interpolation approach which serves to map a quadrilateral to another quadrilateral. A transformation for a point (u, v) in the unit square $[0,1] \times [0,1]$ to a point in the quadrilateral defined by the points $A, B, C,$ and D can be computed like so:

$$T(u,v) = \begin{bmatrix} 1-u & u \end{bmatrix}\begin{bmatrix} A & B \\ C & D \end{bmatrix}\begin{bmatrix} 1-v \\ v \end{bmatrix}.$$

For our algorithm, we begin with a point in the quadrilateral $ABCD$, find the corresponding coordinates in the unit square, and then apply the transformation to those unit square coordinates to get its corresponding point in the quadrilateral $EFGH$.

Given point (p_x, p_y) in quadrilateral $ABCD$, we solve for u and v using this equation:

$$\begin{bmatrix} 1-u & u \end{bmatrix}\begin{bmatrix} A & B \\ C & D \end{bmatrix}\begin{bmatrix} 1-v \\ v \end{bmatrix} = \begin{bmatrix} p_{\mathrm{x}} \\ p_{\mathrm{y}} \end{bmatrix}.$$

Then, we solve for (x, y) in quadrilateral $EFGH$ by plugging u and v into this equation:

$$\begin{bmatrix} 1-u & u \end{bmatrix}\begin{bmatrix} E & F \\ G & H \end{bmatrix}\begin{bmatrix} 1-v \\ v \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}.$$

3: CROSS-DISSOLVE

While the image morphs to its proper location, we need to cross-dissolve the second image into the first image so that once the morphing is complete, the second image is the one shown on screen. Cross-dissolving can be achieved using this linear equation:

$$c = (1 - t)c_1 + tc_2$$

where t represents the t-th frame of the morph, c_1 represents the color from the morphing image, and c_2 represents the color from the destination image.

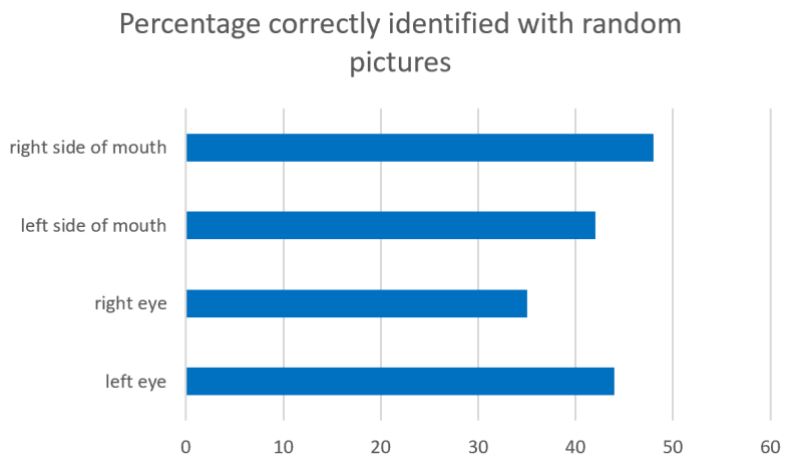
III: RESULTS

Morphing results of when t = 0, ¼, ½, ¾, 1 using affine interpolation.

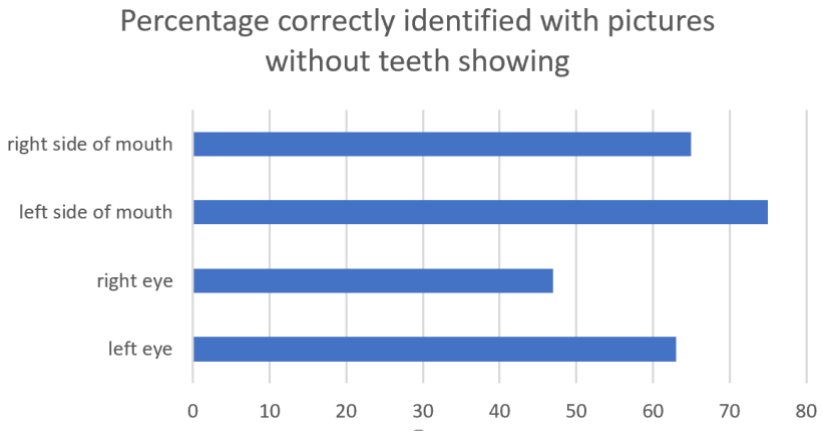
IV: ANALYSIS

1: Feature-finding Algorithm

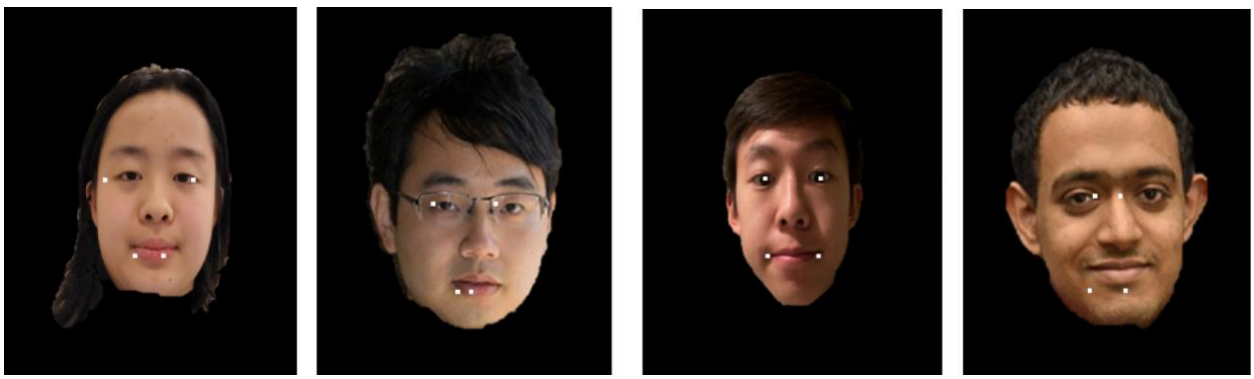
For our analysis, we want to test the accuracy of our feature finding algorithm. We ran our feature finding algorithm on about 100 random pictures from TigerBook and determined whether the feature finder has correctly identified the mouth or the eyes. A correct identification means that the point identified lies within the eye/mouth specified. For example, we still considered the identification correct if it identified a point within the eye, even if it was not the center of the eye. We then calculated the percentage of times the algorithm has correctly identified each of the following features: left eye, right eye, left side of mouth, right side of mouth. The graph is as shown:



One observation as we tested the feature finding algorithm was that the algorithm performed significantly better when the face was not smiling, more specifically when the face did not have teeth showing. The accuracy for the algorithm was recalculated for the pictures without teeth showing, yielding the following results:

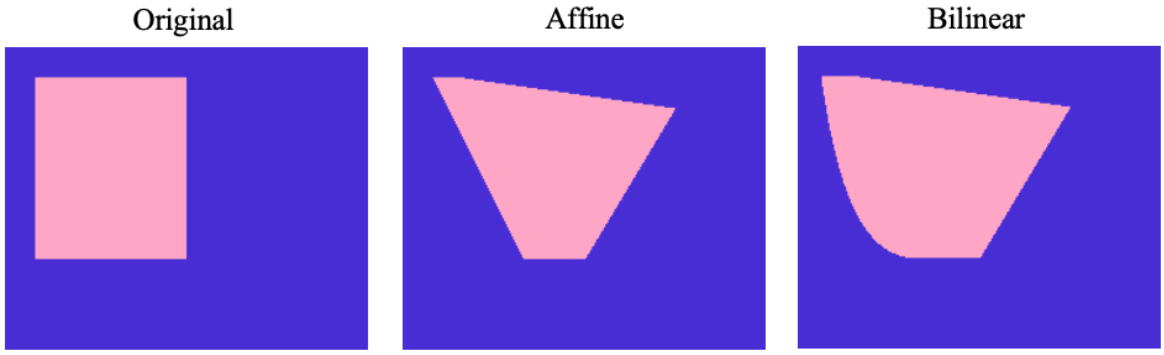


Examples of the feature-finding algorithm:



2: Interpolation Algorithm

It is difficult to distinguish the effects of the type of transformation on the morph when cross-dissolving between photos of human faces. However, there are clear differences. While affine interpolation maps straight lines to other straight lines, bilinear interpolation maps straight lines to straight lines as well as curved lines. Here is an example of a random morph, using both affine and bilinear approaches:



Generally speaking, the morph results are better when we use bilinear interpolation because the morphs are more streamline and dynamic due to the possibility for curves.

V: CONCLUSIONS AND LIMITATIONS

This paper provides an introductory algorithm and approach to the face-morphing technique that TikTok has popularized. Through feature-finding, interpolation, and cross-dissolving, we are able to approximate the behavior of the face-morphing algorithm, although it still contains many limitations that prevent it from providing a perfect face morph. First, the feature finding algorithm that we created is still heavily unreliable and fails if the image has certain properties (if teeth is showing or if glasses are present). These inconsistencies can be explained through the implementation of the feature finding algorithm, which relies on the assumption that the eyes are the most complex area of the face and the mouth is the area that most closely resembles the color red. These dependencies hinder the algorithm heavily and make the preprocessing step ever more important. To improve the algorithm, perhaps a different approach to identifying the features of a face would be more desirable; a machine learning approach or using a face-finder algorithm could possibly be strong improvements to the feature detection algorithm. Second, the face morphing would be more accurate if more feature points could be identified, including possibly the ears, sides of the face, and nose. The more feature points that are identified, the more accurate the face morph will become, as each feature point is matched to the next. Finally, third limitation is the reliance on the preprocessing step, which once again is a byproduct of the algorithmic structure for the feature-finding step because noisy backgrounds distort the color complexity and hence make the algorithm miss the eyes and mouth. For further steps, we can explore a projective interpolation algorithm which, like affine interpolation, maps straight lines to other straight lines, but instead for quadrilaterals, and compare its results with the previous two interpolation methods.

VI: REFERENCES

Jonas Gomes et al. "Warping and morphing of graphical objects ", Morgan Kaufmann Publishers (1999).

Yu-li, and Yi-Wen. *A Study on Face Morphing Algorithms*, ccrma.stanford.edu/~jacobliu/368Report/index.html.

Martin Bichsel, "Automatic Interpolation and Recognition of Face Images by Morphing", citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.53.2535&rep=rep1&type=pdf

“Weibo Fu.” Department of Mathematics, Princeton University, www.math.princeton.edu/people/weibo-fu

“Mohan Swaminathan.” Department of Mathematics, Princeton University, www.math.princeton.edu/people/mohan-swaminathan

“Theodore Drivas.” Department of Mathematics, Princeton University, www.math.princeton.edu/people/theodore-drivas

“Morphing.” *Wikipedia*, Wikimedia Foundation, 4 May 2020, en.wikipedia.org/wiki/Morphing.