CIS581 Computer Vision
Project 2, Face Morphing
Due October 18, 3:00pm

Instructions. This is an individual assignment. All matlab functions should follow the names and arguments stated in the problems in order for them to run properly with the grading script. A test script will be provided shortly that will call your functions to ensure they will run inside the grading script. To submit the assignment, upload a zip file containing all your code via Canvas.

Turn-in

Zip your files into a folder named `PennKey_Project2.zip` and submit it via Canvas.
This should include:

- your .m files for the five required Matlab functions
- some demo .m scripts for generating your face morphing video
- any additional .m files with helper functions for your code
- the images you used
- the two .avi files generated for each morph methods in face morphing

All the files should be in the top level of the ZIP file (no subfolders, please).

Overview

This project focuses on image morphing techniques. You will produce a "morph" animation of your face into another person's face. You can pick anyone (or any object) you prefer.

You will need to generate 60 frames of animation. You can convert these image frames into a .avi movie. This can be done using the VideoWriter functionality in Matlab.

A morph is a simultaneous warp of the image shape and a cross-dissolve of the image colors. The cross-dissolve is the easy part; controlling and doing the warp is the hard part. The warp is controlled by defining a correspondence between the two pictures. The correspondence should map eyes to eyes, mouth to mouth, chin to chin, ears to ears, etc., to get the smoothest transformations possible.

For the triangulation used for Task 2. You can compute it any way you like, or even define it by hand. A Delaunay triangulation (see Matlab `delaunay`) is a good choice. Recall you need to generate only one triangulation and use it on both the point sets. We recommend computing the triangulation at the midway shape (i.e. mean of the two point sets) to lessen the potential triangle deformations.

Notice that, you need to implement not only the five functions decried bellow, but also the full script to generate face morphing video using triangulation model and Thin Plate Spline model.

1 Defining Correspondences

First, you will need to define pairs of corresponding points on the two images by hand (the more points, the better the morph, generally). The simplest way is probably to use the "cpselect" tool or write your own little tool using ginput and plot commands (with hold on and hold off).

Create the function:

`[im1_pts, im2_pts] = click_correspondences(im1,im2)`

(INPUT) `im1`: $H_1 \times W_1 \times 3$ matrix representing the first image.
(INPUT) `im2`: $H_2 \times W_2 \times 3$ matrix representing the second image.
(OUTPUT) `im1_pts`: $N \times 2$ matrix representing correspondences coordinates in first image.
(OUTPUT) `im2_pts`: $N \times 2$ matrix representing correspondences coordinates in second image.

2 Image Morph Via Triangulation

You need to write a function that produces a warp between your two images using point correspondences:

`morphed_im = morph(im1, im2, im1_pts, im2_pts, warp_frac, dissolve_frac)`

(INPUT) `im1`: $H_1 \times W_1 \times 3$ matrix representing the first image.
(INPUT) `im2`: $H_2 \times W_2 \times 3$ matrix representing the second image.
(INPUT) `im1_pts`: $N \times 2$ matrix representing correspondences in the first image.
(INPUT) `im2_pts`: $N \times 2$ matrix representing correspondences in the second image.
(INPUT) `warp_frac`: $1 \times M$ vector representing each frames' shape warping parameter.
(INPUT) `dissolve_frac`: $1 \times M$ vector representing each frames' cross-dissolve parameter.
(OUTPUT) `morphed_im`: M elements cell array, each element is a morphed image frame.

In particular, images im1 and im2 are first warped into an intermediate shape configuration controlled by `warp_frac`, and then cross-dissolved according to `dissolve_frac`. For interpolation, both parameters lie in the range [0,1]. They are the only parameters that will vary from frame to frame in the animation. For your starting frame, they will both equal 0, and for your ending frame, they will both equal 1.

Given a new intermediate shape, the main task is to map the image intensity in the original image to this shape. As we explained in class, this computation is best done in *reverse*:

1. for each pixel in the target intermediate shape(called shape B from this point on), determine which triangle it falls inside. You could use Matlab `tsearchn` for this, or implement your own barycentric coordinate check.

2. compute the barycentric coordinate for each pixel in the corresponding triangle. Recall, the computation involves solving the following equation:

$$\begin{bmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{1}$$

   where a, b, c are the three corners of triangle, (x, y) the pixel position, and α, β, γ are its barycentric coordinate. Note you should only compute the matrix $\begin{bmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ 1 & 1 & 1 \end{bmatrix}$ and its inverse only once per triangle. Also, in this case, DO NOT use `tsearchn` for computing the barycentric coordinate.

3. compute the cooresponding pixel position in the source image: using the barycentric equation(eq. 1), but with three corners of the same triangle in the source image

$\begin{bmatrix} a^s_x & b^s_x & c^s_x \\ a^s_y & b^s_y & c^s_y \\ 1 & 1 & 1 \end{bmatrix}$, and plug in same barycentric coordinate (α, β, γ) to compute the pixel position (x^s, y^s, z^s). You need to convert the homogenous coordinate (x^s, y^s, z^s) to pixel coordinates by taking $x^s = x^s/z^s, y^s = y^s/z^s$.

4. copy back the pixel value at x^s, y^s the original (source) image back to the target (intermediate) image. You can round the pixel location (x^s, y^s) or use bilinear interpolation.

3 Image Morph Via Thin Plate Spline

Implement the same function as in Task 2 except using Thin Plate Spline model.

For this part, you need to compute thin-plate-spline that maps from the feature points in the intermediate shape (B) to the corresponding points in original image (A). Recall you need two of them - one for the x coordinate and one for the y. A thin plate spline has the following form:

$$f(x, y) = a_1 + a_x \cdot x + a_y \cdot y + \sum_{i=1}^{p} w_i U(||(x_i, y_i) - (x, y)||), \tag{2}$$

where $U(r) = -r^2 log(r^2)$.

We know there is some thin-plate spline (TPS) transform that can map the corresponding feature points in image (B) back to image (A), using the same TPS transform we will transform all of the pixels of image (B) into image (A), and copy back their pixel value.

You need to write three Matlab functions:

1. Thin-plate parameter estimation:

   `[a1,ax,ay,w] = est_tps(ctr_pts, target_value)`

   (INPUT) `ctr_pts`: $N \times 2$ matrix, each row representing corresponding point position (x, y) in second image.
   (INPUT) `target_value`: $N \times 1$ vector representing corresponding point position x or y in first image.
   (OUTPUT) `a1`: double, TPS parameters.
   (OUTPUT) `ax`: double, TPS parameters.
   (OUTPUT) `ay`: double, TPS parameters.
   (OUTPUT) `w`: $N \times 1$ vector, TPS parameters.

Recall the solution of the TPS model requires solving the following equation:

$$\begin{bmatrix} K & P \\ P^T & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \cdots \\ w_p \\ a_x \\ a_y \\ a_1 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ \cdots \\ v_p \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{3}$$

where

$$K_{ij} = U(||(x_i, y_i) - (x_j, y_j)||), \tag{4}$$

$v_i = f(x_i, y_i)$, and ith row of P is $(x_i, y_i, 1)$. K is a matrix of size p by p, and P is a matrix of size p by 3. In order to have a stable solution you need to compute the solution by:

$$\begin{bmatrix} w_1 \\ w_2 \\ \cdots \\ w_p \\ a_x \\ a_y \\ a_1 \end{bmatrix} = inv(\begin{bmatrix} K & P \\ P^T & 0 \end{bmatrix} + \lambda * I(p+3, p+3)) \begin{bmatrix} v_1 \\ v_2 \\ \cdots \\ v_p \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{5}$$

where $I(p+3, p+3)$ is an identity matrix of size $p+3$, (eye() in matlab), and $\lambda \geq 0$ (usually close to zero).

You need to compute two TPS, one by plugging in the x-coordinates as v_i, and one by plugging in the y-coordinates as v_i.

2. `morphed_im = morph_tps(im_source, a1_x, ax_x, ay_x, w_x, a1_y, ax_y, ay_y, w_y, ctr_pts, sz)`

    (INPUT) `im_source`: $H_s \times W_s \times 3$ matrix representing the source image.
    (INPUT) `a1_x, ax_x, ay_x, w_x`: the parameters solved when doing `est_tps` in the x direction.
    (INPUT) `a1_y, ax_y, ay_y, w_y`: the parameters solved when doing `est_tps` in the y direction.
    (INPUT) `ctr_pts`: $N \times 2$ matrix, each row representing corresponding point position (x, y) in target image.
    (INPUT) `sz`: 1×2 vector representing the target image size (H_t, W_t).
    (OUTPUT) `morphed_im`: $H_t \times W_t \times 3$ matrix representing the morphed image.

In this step, you need transform all the pixels in image (B) by the TPS model, and read back the pixel value in image (A) directly. The position of the pixels in image A is generated by using equation 2.

3. `morphed_im = morph_tps_wrapper(im1, im2, im1_pts, im2_pts, warp_frac, dissolve_frac)`

   (INPUT) `im1`: $H_1 \times W_1 \times 3$ matrix representing the first image.
   (INPUT) `im2`: $H_2 \times W_2 \times 3$ matrix representing the second image.
   (INPUT) `im1_pts`: $N \times 2$ matrix representing correspondences in the first image.
   (INPUT) `im2_pts`: $N \times 2$ matrix representing correspondences in the second image.
   (INPUT) `warp_frac`: $1 \times M$ vector representing each frames' shape warping parameter.
   (INPUT) `dissolve_frac`: $1 \times M$ vector representing each frames' cross-dissolve parameter.
   (OUTPUT) `morphed_im`: M elements cell array, each element is a morphed image frame.

   This is a wrapper for your TPS morphing, in which `est_tps` and `morph_tps` functions will be called. This function is meant to provide a similar interface as that in triangular morphing. It will be called by our evaluation script.

   Note that the `morphed_im` returned in 3.2 is the morph respective to a single source image, so you will need to apply this to both source images and cross-dissolve the results to obtain the final morphed image.