# CIS581 Computer Vision
# Project 2, Face Morphing
# Due October 15, 4:30pm

## Details

This project is to be done individually. You may discuss how to perform the necessary transforms with classmates, however, you are expected to write your own code and not share code with others.

## Turn-in

Zip your files into a folder named "2_<penn user name>" and email it to cis581.fall2013@gmail.com This should include .m files for the matlab functions and the .avi files generated.

## 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 imageframes into a .avi movie. In Matlab, this can be done using the Matlab function "avifile".

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.

## Task 1: DEFINING CORRESPONDENCES AND TRIANGULATION

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).

You need to write a Matlab function:

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

where im1_pts and im2_pts ( both n-by-2 matrices of (x,y) locations) defines corresponding points in two images.

Now, you need to provide a triangulation of these points that will be used for morphing. You can compute a triangulation any way you like, or even define it by hand. A Delaunay triangulation (see Matlab dalaunay) 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.

## Task 2: IMAGE MORPH VIA TRIANGULATION

You need to write a function:

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

that produces a warp between im1 and im2 using point correspondences defined in im1_pts and im2_pts (which are both n-by-2 matrices of (x,y) locations) and the triangulation structure tri.. The parameters warp_frac and dissolve_frac control shape warping and cross-dissolve, respectively. 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 the this shape. As we explained in the 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 "tsearch" 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 $\alpha, \beta, \gamma$ 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.

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 $\begin{bmatrix} a_x^s & b_x^s & c_x^s \\ a_y^s & b_y^s & c_y^s \\ 1 & 1 & 1 \end{bmatrix}$, and plug in same barycentric coordinate $(\alpha, \beta, \gamma)$ 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.

## Task 3: IMAGE MORPH VIA THIN PLATE SPLINE

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

For this project, 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 (SPT) transform that can map the corresponding feature points in image (B) back to image (A), using the same SPT transform we

will transform all of the pixels of image (B) into image (A), and copy back their pixel value.

You need to write 2 Matlab functions:

1. Thin-plate parameter estimation:

   [a1,ax,ay,w] = est_tps(pts, target_value);

   where pts is the point positions (x and y) in the image (B) (nx2), and target_value is the corresponding point x *or* y position in image (A).

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

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

   where

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

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

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

   where $I(n+3, n+3)$ is an identity matrix of size $n+3$, (eye() in matlab).

   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_tsp(im_source, a1_x, ax_x, ay_x, w_x, a1_y, ax_y, ay_y, w_y, ctr_pts, sz);
   where
   a1_x, ax_x, ay_x, w_x are the parameters solved when doing est_tps in the x direction.
   a1_y, ax_y, ay_y, w_y are the parameters solved when doing est_tps in the y direction.

ctr_pts are the control points used (the second and third columns of P)
sz is the desired size for morphed_im stored as [rows, cols] when the image sources are of difference sizes

This is rather straightforward 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.