CIS581 Computer Vision
Project 2, Image Blending and Face Morphing
Due October 13, 3:00pm

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>.zip" and submit it via Canvas. This should include:

- your .m files for the six required Matlab functions
- some demo .m scripts for showing your results
- 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).

Image Blending

When we want to stitch two images together, we will need blending techniques. Ideally each pixel in intersection should have the same intensity in every image, but in reality this is not the case since we are blending two different images. Actually, even if we are blending

images with similar objects or pattern around boundary (such as in panoramas), there are still a number of reasons that can cause the inconsistency: vignetting (intensity decreases towards the edge of the image), parallax effects due unwanted motion of the optical center, radial distortion etc. Thus, a good blending strategy is important.

This part of the project focuses on two-band blending, which is a simplified version of pyramid blending. You will choose two images freely (except images used in slides), define masks and blend them together. There are no restrictions on where to blend, you can blend two images together along a certain image boundary, or blend the left part of one image with the right part of the other image, or even blend a small object into another image (as "Horror Photo" in slides).

You will need to write a Matlab function:

[im_blend, im1_low, im1_high, im2_low, im2_high] = twoBandBlending(im1, im2);

where im1 and im2 are source images (RGB or gray image, not image path), im1_low, im1_high, im2_low, im2_high are the low frequency and high frequency part of the source images respectively, and im_blend is the blended output.

Please make sure to submit your source images with name im1_blending.jpg and im2_blending.jpg, since your mask might not make sense for other images.

Two-band Blending

Two-band blending is a simplification of pyramid blending proposed by Brown and Lowe [**?**]. Two-band-blending uses two bands of the images: high frequency band and low frequency band, blend high frequency part with binary mask, and blend low frequency part with linear mask.

Here is a pipeline you may follow:

(1) Smooth source images with a smoothing kernal to get low pass image I_{low}.

(2) Subtract low pass images from source images to get high pass images.

$$I_{high} = I - I_{low}$$

(3) Generate a binary mask w_{binary} to blend high pass images.

$$I_{blend\_high} = I_{1\_high} \times w_{binary} + I_{2\_high} \times (1 - w_{binary})$$

where $w_{ij} = \{0, 1\}$

(4) Generate a linear mask w_{linear} to blend low pass images.

$$I_{blend\_low} = I_{1\_low} \times w_{linear} + I_{2\_low} \times (1 - w_{linear})$$

where entries of w changes linearly form 1 to 0 (or from 0 to 1) around the blending boundary. You'll need to choose a proper window size that works for your images.

(5) Add high pass image and low pass image together to get complete blended image.

$$I_{blend} = I_{blend\_high} + I_{blend\_low}$$

Note that both images and masks are matrices in Matlab, and "$\times$" here is pixel-wise multiplication (or ".*" in Matlab).

Face Morphing

This part of the 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. Also please check the corresponding test scripts (test_script.m and eval_project2.m)

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 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.

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

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 (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);

   where pts is the point positions (x and y) in the image (B) (px2), and target_value is the corresponding point x *or* y position (px1) in image (A).

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);
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.

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

   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 Task 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.