CIS581: Computer Vision and Computational Photography
Project 2: Image Morphing
Due: October. 15, 2019 at 11:59pm

Instructions

- This is an **individual** project. 'Individual' means each student must hand in their **own** answers, and each student must write their **own** code in the homework. It is admissible for students to collaborate in solving problems. To help you actually learn the material, what you write down must be your own work, not copied from any other individual. You **must** also list the names of students (maximum two) you collaborated with.
- You **must** submit your code online on Canvas. We recommend that you can include a README.txt file to help us execute your code correctly. Please place your **code**, **resulting images and videos** into the top level of a single folder (no subfolders please!) named `<Pennkey>_Project2.zip`
- Your submission folder should include the following:
  - your .m or .py scripts for the required functions.
  - .m or .py scripts for generating the face morphing video.
  - any additional .m files with helper functions you code.
  - the images you used.
  - .avi files generated for each of the morph methods in face morphing.
- This handout provides instructions for two versions of the code: MATLAB and Python. You are free to select **either one of them** for this project.
- Feel free to create your own functions as and when needed to modularize the code. For MATLAB, ensure that each function is in a separate file and that all files are in the same directory. For python, add all functions in a helper.py file and import the file in all the required scripts.
- **Start early!** If you get stuck, please post your questions on Piazza or come to office hours!

Overview

This project focuses on image morphing techniques. You will produce a "morph" animation of your face into another person's face. **This is mandatory**. You can also morph your face into anything else that you wish. Use your creativity here!

You will need to generate 60 frames of animation. You can convert these image frames into a .avi movie.

A morph is a simultaneous warp of the image shape and a cross-dissolve of the image colors. The cross-dissolve is the easier part; controlling and doing the warp is the harder 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.

The triangulation used in Task 2 can be computed in any way you like, or can even be defined by hand. A Delaunay triangulation is a good choice. Recall that you need to generate only one triangulation and use it on both the sets of points. We recommend computing the triangulation at the midway shape (i.e. mean of the two point sets) to lessen the potential triangle deformations.

1 Defining Correspondences

Goal First, you will need to define pairs of corresponding points on the two images by hand (In general: The more the points, the better the morph).

For MATLAB, cpselect is recommended. For Python, ginput is recommended. There may be other functions that help you implement this functionality.

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

- (INPUT) im1: H1 $\times$ W1 $\times$ 3 matrix representing the first image.
- (INPUT) im2: H2 $\times$ W2 $\times$ 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.

2 Image Morph Via Triangulation

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

[morphed_im] = morph_tri(im1, im2, im1_pts, im2_pts, warp_frac, dissolve_frac)

- (INPUT) im1: H1 $\times$ W1 $\times$ 3 matrix representing the first image.
- (INPUT) im2: H2 $\times$ W2 $\times$ 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 frame's shape warping parameter.
- (INPUT) dissolve_frac: 1 $\times$ M vector representing each frame's cross-dissolve parameter.
- (OUTPUT) morphed_im: Data structure containing output images.
  - In MATLAB, it should be an M elements cell array, where each element is a morphed image frame.
  - In Python, it should be a 4 dimensional NumPy array, with dimensions being Num $\times$ H $\times$ W $\times$ 3, where Num is the number of images.

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 explained in class, this computation is best done in `reverse`:

- For each pixel in the target intermediate shape(called shape B from this point on), determine which triangle it falls inside. You could use inbuilt functions for this, or implement your own barycentric coordinate check. **Recommended functions for MATLAB and Python will be posted on Piazza shortly.**
- Compute the barycentric coordinate for each pixel in the corresponding triangle. Recall, the computation involves solving the following equation:

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

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 $\begin{pmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ 1 & 1 & 1 \end{pmatrix}$ and its inverse only once per triangle.

- Compute the corresponding pixel position in the source image: using the barycentric equation(eq. 1), but with three corners of the same triangle in the source image $\begin{pmatrix} a^s_x & b^s_x & c^s_x \\ a^s_y & b^s_y & c^s_y \\ 1 & 1 & 1 \end{pmatrix}$, 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 homogeneous coordinate (x^s, y^s, z^s) to pixel coordinates by taking $x^s = x^s/z^s, y^s = y^s/z^s$.
- 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 Test and Submission

- Use different kinds of images. Face to face morphing is mandatory but get creative! Morph objects to objects, even face to objects. Creative morphs will receive the honor of public recognition on Piazza.
- We have provided a test script for MATLAB and Python for this project. Extract the contents of the test script to the same directory as your functions and run `Test_script.m/py` in MATLAB/Python. When grading, we'll be calling your functions in the same manner, so make sure they work as you'd expect on the sample in the test script.
- Collect all your source code files and test images into a folder named as `<Pennkey>_Project2`. Zip this folder and submit it to Canvas. Any break in this rule will lead to a failure in the test script. Only submit codes pertaining to your language of implementation. For example: If you choose to do the project in Python, do not submit the MATLAB folder containing the MATLAB starter codes.