Image Processing COMP0026
Coursework 1
Deadline 4pm 27th November 2023

For Coursework 1 you will implement Face Morphing. You will create a python Jupyter notebook. The assessment will be oral, you will only need to submit your code and the results.

The algorithm will take as input two portrait photos of different people (for instance, you could take a photo of yourself and another classmate). It will then create *"in between"* images by warping the pixels from the start image to the end image, while blending the colours.

To do this we use a parameter w, that will gradually vary between 0 and 1. For instance, if you think of using 50 *"in between"* images, then the parameter will take 50 values between 0 and 1 (w=0.00, 0.02, 0.04, … 1.00). When w=0 the output image O(x,y) will look exactly like the starting image S(x,y), and when w=1 it will look exactly like the ending image E(x,y). This can be achieved by using the following equation.

$$O(x, y) = (1 - w)\, S(x,y) + w\, E(x,y)$$

However, as you know, this naive blending will not achieve a smooth transition between the two images -- it will introduce unwanted ghosting and artefacts. As well as blending the images using the weighting factor w, you need to warp the images to bring them into correspondence. So you need to warp images S(x,y) and E(x,y) towards each other, also in 50 steps. To do this the main problem to solve is to find a correspondence for every pixel in every "in between" image of S(x,y) and the corresponding "in between" image of E(x,y) which amounts to finding their pixel coordinates in both images.

Although this sounds like a difficult task, it can be solved easily by breaking it up into smaller sub-problems i.e. warping small triangles using affine warps, instead of applying a single warp to the entire image.
Your algorithm should follow these steps.

1. Find some corresponding face landmarks between start and end images and display them. This can be done manually, or using an automatic facial landmark detector, to find salient points such as on the eyebrows, corners of the eyes etc… An example of such a detector is dlib which provides 68 facial landmarks on the eyes, nose, mouth, eye-brows and jaw. You could even add a few more points manually. The more points the better the final result (but remember that they need to be corresponding points i.e. you need to click on the same *semantic* point on both faces). I would recommend that you debug on a smaller number of points and once you are sure your algorithm works add more to increase the quality. You will get the same grade if you do this

manually or with a landmark detector. This is not the important part of the coursework.
2. Create a triangulation and visualise it. Again, you can do this manually or you can use the standard Delaunay Triangulation algorithm to do it automatically. Your output should be a list of triangle vertices for each image. The order of the triangles and the vertices should reflect the correspondences between triangles and vertices.
3. Create the intermediate image coordinates in all *"in between"* images for all the vertices of all triangles by linearly interpolating between the start and end positions.
4. For every pair of corresponding triangles, take the 3 pairs of corresponding vertices and estimate an Affine warp. You will need to solve a linear system of equations to estimate the parameters of the affine warp. You must write the code for this function yourself. Do not use a built-in function for this.
5. Map all corresponding points between the triangles.This means, find the new coordinates for all the points in the triangle (before you only knew the position of the 3 vertices, with the affine transformation you can find ALL the correspondences!). Remember to use the inverse warp instead of the forward warp and use bilinear interpolation. You must write the code for this function yourself. Do not use a built-in function for this.
6. Apply blending using the parameter w described above.
7. Create a video with all the *"in between"* images. For instance, you can use *ffmpeg* to create the video.

- The deadline for your code + results submission is 27th November.
- We will mark courseworks by an oral assessment. Instructions will follow soon.

Plagiarism/collusion will NOT BE TOLERATED. Write your own code. Do not work in groups. There will be implementations online -- do not use them, do not plagiarise them, do not copy them. As the assessment is oral it is highly likely that we will find out if you have plagiarised or worked with others.

Make sure that you use your own photos or photos that are Public Domain or have a Creative Commons license.

CLARIFICATION About Coursework 1

The coursework requires you to estimate AFFINE TRANSFORMATIONS between corresponding triangles.

You may NOT use built in functions from OpenCV or any other library for this. You must build your own function that takes in pairs of corresponding points and returns the parameters of the Affine Transformation.

Some students are asking if you can use a built-in function to determine if a point is inside a triangle or not. That's OK, you can use a built-in function for that.

The TWO main things that I do NOT want you to use built in functions for are:

1. Estimating the parameters of the AFFINE TRANSFORM given a set of corresponding points. You must write your own function to do this.
2. Using the estimated affine transformations to warp the triangles using bilinear interpolation to create the intermediate images.

You may use built in functions for matrix operations such as: multiplying matrices, computing the inverse of a matrix, transpose etc…

You may NOT use the "applyaffine" built-in function.