COMP0026 (Image Processing) Coursework II: *Warping*

COMP0026 Team

Tobias Ritschel, Niloy Mitra, Mirghaney Mohamed, Hengyi Wang Daniele Giunchi

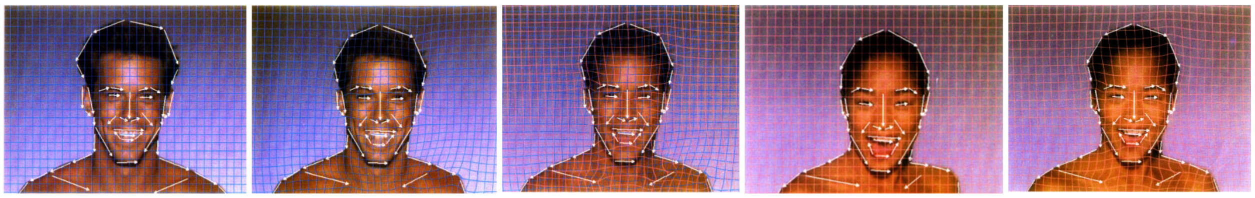
November 28, 2022

You will create a python Jupyter notebook.

The total points for this exercise is **100**.

Please refer to Moodle for the due dates.

For this Coursework, you will implement *Face Morphing* as proposed in Beier and Neely's, "Feature-Based Image Metamorphosis" (SIGGRAPH 1992) and popularized in John Landis' music video for Michael Jackson's "Black or White". Here are some stills from said video:



The algorithm will take as input two portrait photos of different people. This can be photos of yourself and another classmate, but always make sure that you use your own photos or photos that are Public Domain or have a Creative Commons license.

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 $w \in \mathbb{R}$, 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 ($w = 0.00, 0.02, 0.04, \ldots, 1.00$). When $w = 0$ the output image $O(\mathbf{x}), \mathbf{x} \in \mathbb{R}^2$, where $\mathbf{x}$ is the pixel coordinate, will look exactly like the starting image $S(\mathbf{x})$, and when $w = 1$ it will look exactly like the ending image $E(\mathbf{x})$. This can be achieved by

$$O(\mathbf{x}) = (1 - w)S(\mathbf{x}) + wE(\mathbf{x}).$$

However, as you know, this naïve blending will not achieve a plausible transition between the two images – it will introduce unwanted ghosting and artifacts. 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 $S(\mathbf{x})$ and $E(\mathbf{x})$ 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 $S(\mathbf{x})$ and the corresponding "in between" image of $E(\mathbf{x})$ 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 Correspondence (10 points)

Find some corresponding face landmarks between start and end images and display them (**10 points**). This can either 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 a landmark 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.

We 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.

2 Mesh (20 points)

Create a triangulation (**10 points**) and visualise it (**10 points**). 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 2D triangle vertices for each image. The order of the triangles and the vertices should reflect the correspondences between triangles and vertices.

3 Blending with a mesh (20 points)

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. For every pair of corresponding triangles, take the 3 pairs of corresponding vertices and estimate an affine warp (**20 points**). 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. 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. You must build your own function that takes in pairs of corresponding points and returns the parameters of the affine transformation. You can use a built-in function to determine if a point is inside a triangle or not. 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 built in functions from `OpenCV` or any other library for this. You may *not* use the `applyaffine` built-in function. You may *not* use an existing function, to estimate the parameters of the affine transform given a set of corresponding points. You may *not* use the estimated affine transformations to warp the triangles using bilinear interpolation to create the intermediate images.

4 Blending without a mesh (25 points)

An alternative way to morph is "meshless" as prposed by *Schaefer et al.*'s **Image Deformation Using Moving Least Squares**, SIGGRAPH 2006. To this end, you can just interpolate the deformation field itself without making a mesh. Let $\mathbf{x}_{\mathrm{s},i}$ and $\mathbf{x}_{\mathrm{e},i}$ be the 2D start and end position of correspondence i.

Then the deformation vector field $d \in \mathbb{R}^2 \to \mathbb{R}^2$ at position $\mathbf{x}$ is

$$d(\mathbf{x}) = \frac{1}{\sum_i \kappa(||\mathbf{x} - \mathbf{x}_{\mathrm{s},i}||)} \sum_i \kappa(||\mathbf{x} - \mathbf{x}_{\mathrm{s},i}||)(\mathbf{x}_{\mathrm{s},i} - \mathbf{x}_{\mathrm{e},i}) \qquad \text{where} \qquad \kappa(d) = \frac{1}{\alpha d^2}.$$

The idea here is to compute the deformation from every source to every target point and then interpolate it from the neighbors, but without making a mesh, just by averaging all deformations and weighting them with a function κ that weights nearby points more. α can be used to control locality of the warp, and will depend on the number of correspondences you use. The output image is

$$O(\mathbf{x}) = (1 - w)S(\mathbf{x}) + wE(\mathbf{x} + d((\mathbf{x}))),$$

i.e., look-up the source image at the pixel position plus the deformation field at that position.

Implement a version of the approach that implements the deformation directly as explained above (**15 points**) (affine).

Implement the rigid version (**10 points**) as explained in their paper.

5 Capture (25 points)

Apply blending using the parameter w described above (**25 points**). Create a video with all the “in between” images for both methods, meshed and meshless. For instance, you can use `ffmpeg` to create the video.