COMP0026 (Image Processing) Coursework II: *Warping*

COMP0026 Team

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

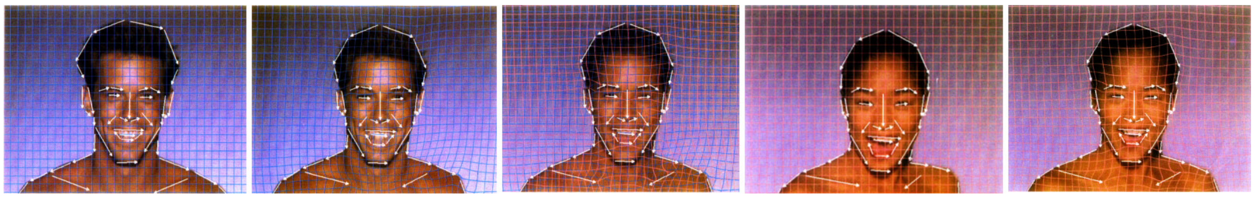
December 21, 2022

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 a simplified version of *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 ($w = 0.00, 0.02, 0.04, \dots, 1.00$). 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.

For this task, you can use a pre-trained facial landmark detector, e.g. `dlib`, which provides 68 facial landmarks on the eyes, nose, mouth, eyebrows, 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. For this task, you are allowed to use *NumPy* built-in functions, however, *OpenCV* built-in functions are not allowed to use for estimating affine transform.

Estimating the color of each point by mapping the corresponding triangles with the estimated affine transform and using bilinear interpolation. Remember for each intermediate image you need two sets of affine transform, and weight the bilinearly interpolated color from the source and target images using w. The bilinear interpolation should be achieved by inverse warping instead of forward warping. For this task, you are *not* allowed to use *OpenCV* built-in functions for bilinear interpolation.

4 Blending without a mesh (20 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

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

The idea here is to compute the warping 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 prefers nearby points more. α can be used to control the 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.

To do that you can use the landmarks from the previous task as correspondences - you can also add some corner points - and your deformation field should deform each point to its corresponding point. Then you can use the κ to weigh the deformation of the nearby points. Play a bit with the value of α and remember the more points you have the more confident you are in the deformation of the corresponding points.

In case your images are quite far from each other you can consider modifying the equation a bit and trying and deforming both the source and target to get a slightly better result. In that case, you can use an interpolated version of the correspondences using the same w. The output image is

$$O(\mathbf{x}) = (1 - w)S(\mathbf{x} + d_1((\mathbf{x}))) + wE(\mathbf{x} + d_2((\mathbf{x}))),$$

Where d_1, d_2 are in the form of $d(x)$, but the corresponding points are different. d_1 takes the corresponding points from the source to the interpolated points with w. While d_2 takes the interpolated corresponding points with w to the target points.

Implement a version of the approach that implements the deformation directly as explained above (**20 points**) (affine).

~~Implement the rigid version (points 5) 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` or `cv2.VideoWriter` to create the video.