Face Image Morphing (May 2017)

Conrad Appel[1], Danton Zhao[2], Logan Campbell[3]
Lyle School of Engineering, Computer Engineering[1], Electrical Engineering[2,3]
Southern Methodist University

***Abstract*—In this paper, we propose an implementation of Facial Image Morphing using openCV. Our framework uses the detection of facial features in an image and then warps and adjusts the pixel intensity values determined by a predefined alpha value. Alpha is a value between 0 and 1 that determines the presence of each original image in the morphed image.**

I. Introduction

AT THE simplest level, Image morphing is simply the blending of pixels of image I and J to create M using:

$$M(x,y) = (1-\alpha)I(x,y) + \alpha * J(x,y) \tag{1}$$

$$0 \leq \alpha \leq 1$$

When $\alpha = 0$, M will be identical to I, and when $\alpha = 1$, it will look like J. This simplistic method will blend the two images, but the faces will most likely not be aligned and this will result in a blurry mess of image M

Thus, we must establish pixel correspondence between images I and J. Only then can we calculate the pixel locations in M. The x and y locations in M can be represented with:

$$X_M = (1-\alpha)X_I + \alpha * X_J \tag{2}$$
$$Y_M = (1-\alpha)Y_I + \alpha * Y_J \tag{3}$$

Equation (**??**) is then applied to the shifted pixels to obtain our morphed image.

II. Image Morphing Technique

FIRST, to find each corresponding points in images I and J, we used dlib to detect facial feature points. We then added points to the corners of the images to increase the number of triangles used for Delaunay Triangulation in the following step. Using the corresponding points, we want to perform Delaunay Triangulation on the set of averaged points for the two images. This will give us an indexed list of triangles that we can use to warp the images for morphing.

To warp or blend the two images we must use equations (**??**) and (**??**) to locate the pixels in image M. Then we calculate the affine transform using OpenCVs getAffineTransform. This will map the three corners of a triangle in image I or J to a corresponding triangle in M. Then we must warp all of the pixels inside each triangle to M. This is achieved using the function warpAffine. It must be noted that warpAffine only takes in an image and not a triangle, so we had to create a bounding box for each triangle and then create a triangular mask using fillConvexPoly. Finally after some fine-tuning to hide the seams, the images can be blended using varying values of α.

A. Find Corresponding Points

a) Dlib

i) dlib.get_frontal_face_detector() to find a bounding box over a face

1) Uses sliding window classifier over a histogram of gradients pyramid

B. Delaunay Triangulation

C. Warping Images

III. Analysis

IV. Conclusion

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.
[2] https://github.com/