MP7: Audio-to-Video Animation

ECE 417 Fall 2017

Yuchen Liang

Zixu Zhang

December 12, 2017

1 Introduction

In this MP, we use conduct audio to video animation by applying artificial neural networks (ANNs) to train parameters that map speech signals to video features of facial movement. After we successfully train the ANNs, we will use it to map facial mesh sequences of our test wave. After we generate the mesh for audio sequence, we will use revers affine transformation to wrap images of each frame of the audio, and produce the facial animation.

2 Method

2.1 Artificial Neural Networks (ANNs)

Artificial Neural Networks are network of its element *neurons*, which is able to receive signal input from another neuron, execute some calculation based on the input, and send out signal. ANNs are inspired by human brain system, which is good at learning and performing complex tasks. If we regard the neural networks as functions, it is a powerful estimator for any linear and nonlinear functions.
In side neuron, there are usually three steps: weighting, summing, and activation. The most basic form of neuron's activation is called perceptron, which is a simple binary function defined as

$$f(x) = \begin{cases} 1 & \text{if } W^T X > \theta \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where W is the weight vector, X is the input signal, and θ is some threshold. A typical ANN is composed of an input layer, one or several hidden layers, and one output layer.

2.2 Affine Transformation and Barycentric Coordinates

Affine transformation is able to linearly map points between two frames as following

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_0 \\ b_1 & b_2 & b_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \eta \\ \zeta \\ 1 \end{bmatrix} \tag{2}$$

Assume we have a triangle on a $2D$ plane, and its three corners can be represented as three position vectors $\overrightarrow{x}_1,\ \overrightarrow{x}_2,\ \overrightarrow{x}_3$. In this way, we are able to write the position vector $\overrightarrow{x}$ of any point within or on the triangle as equation (2) or in a matrix form as equation (3)

$$\overrightarrow{x} = \lambda_1 \overrightarrow{x}_1 + \lambda_2 \overrightarrow{x}_2 + \lambda_3 \overrightarrow{x}_3 \tag{3}$$

$$\overrightarrow{x} = \begin{bmatrix} \overrightarrow{x}_1 & \overrightarrow{x}_2 & \overrightarrow{x}_3 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} \tag{4}$$

where, λ follows that $\lambda_1, \lambda_2, \lambda_3 \in [0, 1]$, and $\lambda_1 + \lambda_2 + \lambda_3 = 1$. In this sense, it is easy to see if a given point is within the triangle by checking corresponding λ as

$$\begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} \vec{x}_1 & \vec{x}_2 & \vec{x}_3 \\ 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} \vec{x} \\ 1 \end{bmatrix} \tag{5}$$

Suppose that we have a mesh triangle $\triangle_i$ on the source image, and its three corners $\vec{x}_{i1}$, $\vec{x}_{i2}$, $\vec{x}_{i3}$. On the deformed image, the corresponding mesh triangle is $\tilde{\triangle}_i$, whose corners are $\vec{z}_{i1}$, $\vec{z}_{i2}$, $\vec{z}_{i3}$. Assume that all points between $\triangle_i$ and $\tilde{\triangle}_i$ can be mapped by an affine transformation A. Whereby, we have

$$\begin{bmatrix} \vec{z}_{i1} & \vec{z}_{i2} & \vec{z}_{i3} \end{bmatrix} = A \begin{bmatrix} \vec{x}_{i1} & \vec{x}_{i2} & \vec{x}_{i3} \end{bmatrix} \tag{6}$$

For any point $\vec{x}_i$ with in the source mesh triangle, and its corresponding point $\vec{z}_i$, we can find the following relationship.

$$\vec{z} = A \begin{bmatrix} \vec{x}_1 & \vec{x}_2 & \vec{x}_3 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} \vec{z}_1 & \vec{z}_2 & \vec{z}_3 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} \tag{7}$$

This means that for any points in Barycentric coordinate under affine transformation, its λ will preserve. In this MP, we want to wrap deformed images with given mesh of both source and deformed images. This is performed by the function `wraping.m` with following steps. From line 12 to 17, we find the coordinates of each triangle on both deformed and source images. At the mean while, we also find the centroid of each mesh triangle. Then from line 45 to 59, for each pixel, we try to find which triangle it belongs to. This step is done by first find its 10 nearest neighbors. Since most of mesh triangle has similar size, then we can make the assumption that if the point is in any of of mesh triangle, we should be able to find corresponding centroid with in 10 nearest centroid. We traverse through all those 10 nearest triangles, and to see if we can the point that meet the condition stated above. After we determine which triangle the point belongs to, and corresponding λ in Barycentric coordinate, we are able to obtain its corresponding pixel coordinate in the source images. The grey scale color mapping between images are finised by bilinear interpolation.

2.3 Bilinear Interpolation

Since an image is considered a matrix in computer memory, its coordinates are represented as integers. However, in the transformation described above, non-integer value could emerge in at least one of the source image and the deformed image (usually chosen to be the former for easier calculation). Hence, interpolating the value between pixels is necessary, and in this MP we use bilinear interpolation (Piece-wise linear interpolation), on line 84 and 85 of `warping.m`.

Suppose we need to find the pixel value at (u, v), where u and v are not integers. (If one of them is integer, only one of the following two steps is needed.) Define $u_1 = \lfloor u \rfloor$, $v_1 = \lfloor v \rfloor$, $u_2 = \lceil u \rceil$, $v_2 = \lceil v \rceil$. We first interpolate (linearly) along x-axis on $y = v_1$ and $y = v_2$, so we get

$$I(u, v_1) = I(u_1, v_1) + (u - u_1) * \frac{I(u_2, v_1) - I(u_1, v_1)}{u_2 - u_1} = \frac{I(u_1, v_1) * (u_2 - u) + I(u_2, v_1) * (u - u_1)}{u_2 - u_1}$$

$$= I(u_1, v_1) * (u_2 - u) + I(u_2, v_1) * (u - u_1)$$

since $u_2 - u_1 = 1$, and similarly

$$I(u, v_2) = I(u_1, v_2) * (u_2 - u) + I(u_2, v_2) * (u - u_1)$$

Finally we interpolate along y axis:

$$I(u, v) = I(u, v_1) * (v_2 - v) + I(u, v_2) * (v - v_1) \tag{8}$$