Become Brad Pitt

Chung-Yi Weng, Xuan Luo
Computer Science & Engineering
University of Washington
chungyi@cs.washington.edu, xuanluo@cs.washington.edu

Abstract

In this project, we designed an real-time face reenactment system which enables users to control celebrities' head motion and facial expression by moving his/her own head. We also implemented a transition effect which smoothly morphs one face to another when the user changes to control another celebrity. Our system involves implementing a high-speed tracking module, a puppetry module to control facial expression and a animation module to morph the faces. Experiments show that our system can reenact the head motion and facial expression of target celebrities (e.q. Brad very well in real time.

1 System Overview

Our system is composed of a face tracking module to obtain a bounding box of face, a puppetry module which control the facial expression and a animation module that does the morphing effect. In the following, we will describe the tracking module in Sec. 2, puppetry module in Sec. 3 and animation module in Sec. 4. Finally, experimental results are shown in Sec. 5.

2 Face Tracking Module

We implemented a high-speed tracking with Kernelized Correlation Filters (KCF) [4]. It is a tracking-by-detection algorithm. So at each new frame, it will first detect the best bounding box and train its model by the new data. The strengh of KCF is that it is trained with thousands of sample patchs at each new frame but can achieve 172 FPS on CPU. KCF also has high accuracy and has won the VOT2014 challenge [6]. Its main idea is to speed up its training and detection process by Discrete Fourier Transform (DFT). We will first illustraste the DFT trick with the simplest case of detection using a linear model in Sec. 2.1, and then describe application of this DFT trick in kernelized model as in our implementation, see Sec. 2.2.

2.1 DFT in Detection with a Linear Model

Here we first describe a target detection behaviour KCF wants to approximate and then demonstrates how KCF approximates it.

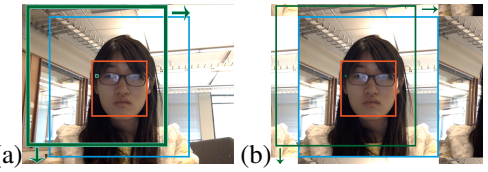

Figure 1: Sliding window on (a) the original image and (b) a periodic image.

The desired detection process is illustrated in Fig. 1 (a). The tracker is given a bounding box of the largest face from face detection [8] at the first frame. It then trains its detection model with samples in the first frame and starts reading video stream. Suppose in a previous frame, the tracker detected the red bounding box (bbox). Then at a new frame, we slide a larger green window across all pixels in the blue bbox. At each window, we extract its feature $\mathbf{z}$ and compute

its detection score by $f(\mathbf{z}) = \mathbf{w}^T\mathbf{z}$.Then we pick the window with the highest score to be the new detected bbox.

Computing a feature and $\mathbf{w}^T\mathbf{z}$ for each sliding window is very computationally expensive. So DFT is used to speed it up.

Note that computing $\mathbf{w}^T\mathbf{z}$ over all sliding windows is actually a convolution. By *Convolution Theorem*, we only needs to compute features for the centered blue box once. And then do a fast DFT, a pointwise product in the frequency domain, and finally an inverse DFT to get spatial convolution response in the spatial domain. This DFT trick can dramatically speed up the detection process. But note that with DFT, we are actually considering a periodic image as in Fig. 1 (b), which is a fair approximation when the window is sufficiently large than the detected bbox.

2.2 DFT in Kernelized Model

The tracking algorithm we implemented uses a kernelized model. Here we will describe how a similar DFT trick can be applied to the detection and training and detection process of a kernelized model to gain remarkable speedup.

Features: With the DFT trick, we only need to compute features for the blue box once as in the linear case. The features used are HOG [3] and LAB color histogram at each cell of the HOG feature. More specifically, we pick 15 LAB color centroids, assign pixels to its nearest LAB centroid in the LAB color space and compute a histogram of the assignment for each cell of the HOG feature. Then we concatenate the HOG and LAB histogram together to form the final features.

Detection with Kernelized Model: As in linear case, the target detection behaviour is to compute detection score $y = f(\mathbf{z})$ for each sliding window, where $\mathbf{z}$ is the features of the window as described above. And we will use DFT to approximate it.

Here we use a gaussian kernel. More specifically,

$$f(\mathbf{z}) = \sum_i \alpha_i K(\mathbf{x}_i, \mathbf{z}), \text{where } K(\mathbf{x}, \mathbf{z}) = exp\left(\frac{1}{\sigma^2}(||\mathbf{x} - \mathbf{z}||^2)\right),$$

$\mathbf{x}_i$'s are the features of sample patches used to train the model, α_i's are parameters in the model. Let $\mathbf{k^{xz}} = (K(\mathbf{x}_1, \mathbf{z}), \ldots, K(\mathbf{x}_n, \mathbf{z}))$. When consider a periodic image, [4] shows that we can use DFT to obtain the following approximation

$$\mathbf{k^{xz}} = exp\left(\frac{1}{\sigma^2}(||\mathbf{x}||^2 + ||\mathbf{z}||^2 - 2\mathcal{F}^{-1}(\hat{\mathbf{x}}^* \odot \hat{\mathbf{z}}))\right),$$

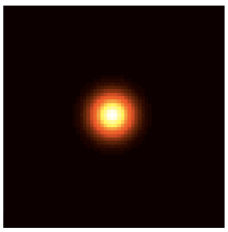

Figure 2: Regression target $\mathbf{y}$ for training: a gaussian map.

where $\mathbf{x}, \mathbf{z}$ are features extracted from the centered blue box in previous training data and the current frame, $\hat{\mathbf{z}} = \mathcal{F}(\mathbf{z})$ is the Fourier transform of $\mathbf{z}$, $\hat{\mathbf{x}}^*$ is the conjugate of the complex Fourier transform of $\mathbf{x}$, $\mathcal{F}^{-1}(\cdot)$ is the inverse Fourier transform and $\odot$ denotes pointwise multiplication. Since we want to compute detection scores for all sliding windows, the vector of all responses in the Fourier domain can be approximated by

$$\hat{\mathbf{f}}(\mathbf{z}) = \hat{\mathbf{k}}^{\mathbf{xz}} \odot \hat{\alpha},$$

where $\hat{\alpha}$ is the Fourier transform of parameters $\alpha = (\alpha_1, \ldots, \alpha_n)$.

Training with Kernelized Model: define obj. describe y. give DFT form. The goal of training is to find parameter α that minimize the squared error over samples $\mathbf{x}_i$ and their regression target y_i,

$$\min_\alpha \sum_i (f(\mathbf{x}_i) - y_i)^2 + \lambda \alpha^T \mathbf{K} \alpha,$$

where kernel matrix $\mathbf{K} = (K(\mathbf{x}_i, \mathbf{x}_j))_{i,j}$ and λ is a regularization parameter that controls overfitting. $\mathbf{y} = y_i$ is defined as a gaussian map as shown in Fig. 2 so that the centered sliding window should have a large detection score. [4] shows that its closed form solution in fourier domain is

$$\alpha = \frac{\hat{\mathbf{y}}}{\hat{\mathbf{k}}^{\mathbf{xx}} + \lambda},$$

where $\mathbf{k^{xx}}$ is defined similar to the $\mathbf{k^{xz}}$ above.

3 Face Puppetry Module

Face puppetry module is responsible for transferring the user's facial movement to the celebrity. In order to achieve the goal, we want to build a mesh both on the user's face (source face) and the celebrity's face (target face). Once we have meshes, we can transfer the movement from the source face to the target face by deforming the meshes. Three components are necessary to meet our goal. The first is to detect facial landmarks and the second is to build a mesh based on facial landmarks. The last is to warp the mesh of the target face to reflect the facial movement of the source face. The flow of face puppetry is shown in Figure.3. Later, we will describe each module sequentially.

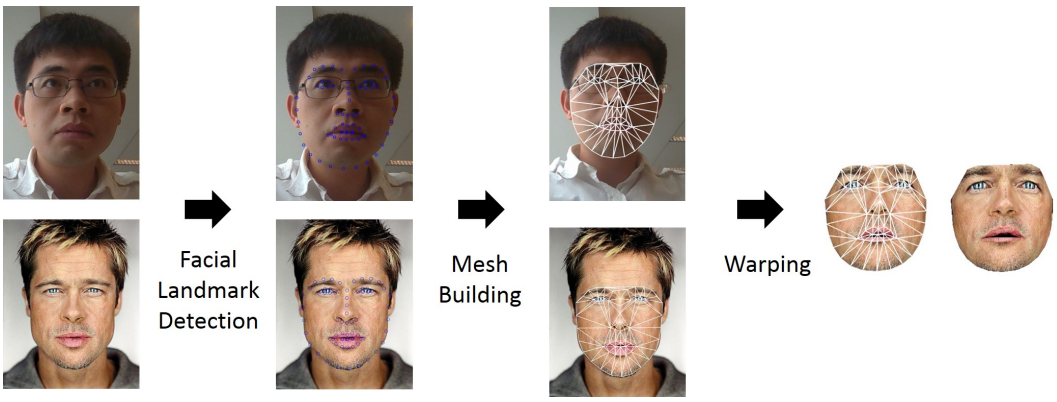

Figure 3: Flow of face puppetry.

3.1 Facial Landmark Detection

Nowadays, the performance of facial landmark detection has quite big improvement both on quality and speed. One of state-of-the-art methods is to solve the problem with a cascade of regression functions. Kazemi et al. [5] propose such kind of algorithm in CVPR 2014.

The regression approach predicts facial shape S in a cascade manner. Beginning with an initial shape S^0, S is progressively refined by estimating a shape increment ΔS. Then, the predicted ΔS is added to S^0 to create S^1, which is the input of next iteration. The process is iterated until a cascade of T regressors. It can be represented as:

$$S^{t+1} = S^t + \Delta S^{(t)} = S^t + r_t(I, S^t)$$

where r_t is a regressor, which predicts an update vector from the image and S^t.

To train each r_t we use the gradient tree boosting algorithm with a sum of square error loss. At each split node in the regression tree we use thresholding the difference between intensities of two pixels as our feature to make an optimal decision. The optimization goal is to find a good feature to split all training samples into left and right nodes by minimize the following error function

$$\sum_{s \in l,r} \sum_{i \in Q_s} \|\mathbf{r}_i - \mu_s\|$$

where Q is the set of the indices of the training examples at a node, $\mathbf{r}_i$ is the residual vector computed from current shape and the ground truth, and μ_s is the mean vector of all residual vectors at the splitted left/right nodes.

After all regressors are trained, we concatenate them all together to construct a cascade. When detecting the facial landmarks of a new face, we start from a predefined initial shape, which is also used in training phase, and then update the shape regressor-by-regrssor in the cascade to get the final shape. A sample result is shown in Figure.3.

3.2 Mesh Building

After getting facial landmarks, we want to build a mesh based on these landmarks. We apply Delaunay triangulation to achieve our goal. It divides the image plane into triangles by connecting input

points (i.e. facial landmarks). We use incremental construction [7] to build the Delaunay triangle mesh. The idea is to re-layout the existed triangles by inserting one point after another. We describe the algorithm as follows.

First, when inserting the first point, we create 3 artificial points "far out" and connect the inserted point to the artificial points to build initial triangles. This step is to make sure the following inserted points would be included in one of existed triangles.

Second, for a new inserted point s, find the triangle Δ that contain s, and replace it with the three triangles resulting from connecting s with all three vertices of Δ, like Figure.4.

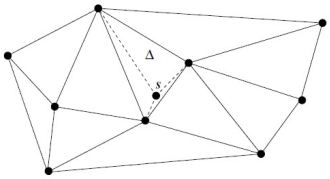

Figure 4: Find the triangle that contains s

Third, as Figure.5 shows, flip the edge in the convex quadrilateral that contains point s if the current split doesn't follow Delaunay property, which is that no point is inside the circumcircle of any triangles.

After all points are inserted, we have divided the plane into Delaunay triangles successfully. You can see one of our results in Figure.3.

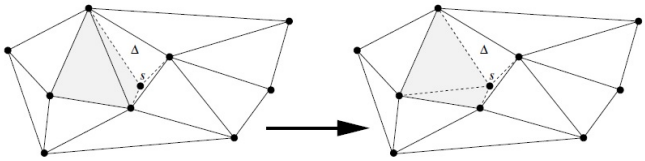

Figure 5: Flip the edge in the convex quadrilateral containing s.

One trick we use here is to first build triangle mesh on the celebrity's face and then propagate the layout to the user's face. It makes the triangle layout identical in both faces, and so we can get their triangle correspondence accordingly.

3.3 Local Warping

After building triangle meshes on both faces, the next step is to warp the celebrity's mesh to the user's mesh in order to transfer the facial movement. Here is our algorithm.

For each corresponding triangle pairs between the two meshes, we compute the corresponding affine matrix and use it to do affine transform for all points located in the celebrity's triangle. We do the same process for all celebrity's triangles. Finally, the celebrity's mesh is deformed to be identical to the user's mesh (that is, the user's facial movement). An warping example is shown in Figure.3.

3.4 Post Processing

Before rendering the puppetry result, we refine it by hollowing the mouth region between the lips of the celebrity, like Figure.6. This post-processing step is in order to remove the apparent artifact resulting from lacking texture in celebrity's mouth region. It significantly improve the final result to make the puppetry more nature.

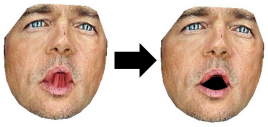

Figure 6: Hollow the region between the lips.

4 Face Animation Module

In our system, we allow users to change the celebrity they want to control. When it happens, we will animate the process by morphing current controlled celebrity's face to next one, as shown in Figure.7.

In [9], it guide us that in order to get good morphing images, we must align the source image and the target image. So we first align the face of

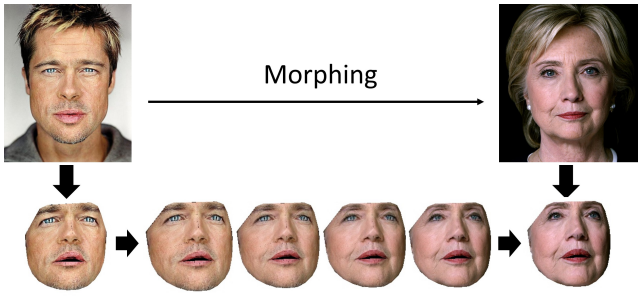

Figure 7: Morphing one celebrity to another.

both celebrities to the same mesh, which is the user's mesh when he/she determines to switch the controlled celerity. After aligning both face images, we will do cross-dissolve on the two images to generate in-between images. The cross-dissolve formulation is as follows:

$$I_{inbetween} = (1-\alpha)I_{source} + \alpha I_{target}$$
$$\alpha = \frac{\text{the index of current frame}}{\text{the number of animated frames}}$$

5 Experimental Results

Implementation Detail We use C++ and OpenCV [2] to implement the project. Also, DLib [1] is used to detect facial landmarks, which offers a good implementation of [5]. The final system runs in real time in the most popular laptops, and it offers lots of joy to the users because of its real-time response.

Results We offer some example puppetry results in Figure.8. You can see the facial movement transfer is quite convincing even we don't build the 3D face model actually. The benefit of using 2D mesh give us big performance gain to make a real-time face reenactment system possible.

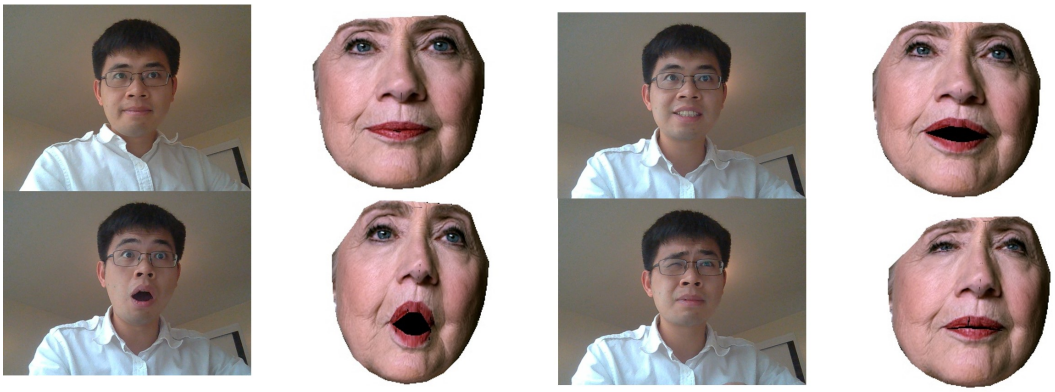

Figure 8: Examples of face puppetry.

References

[1] Dlib c++ library. `http://dlib.net/`.

[2] Opencv: Open source computer vision. `http://http://opencv.org/`.

[3] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[4] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. High-speed tracking with kernelized correlation filters. *CoRR*, abs/1404.7584, 2014.

[5] V. Kazemi and J. Sullivan. One millisecond face alignment with an ensemble of regression trees. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1867–1874, 2014.

[6] F. LIRIS. The visual object tracking vot2014 challenge results.

[7] D. Lischinski. Incremental delaunay triangulation. *Graphics gems IV*, pages 47–59, 1994.

[8] P. Viola and M. J. Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

[9] G. Wolberg. Image morphing: a survey. *The visual computer*, 14(8):360–372, 1998.