Real-time Facial Animation on Mobile Devices

Yanlin Weng, Chen Cao, Qiming Hou, Kun Zhou

State Key Lab of CAD&CG, Zhejiang University, Hangzhou, China, 310058

Abstract

We present a performance-based facial animation system capable of running on mobile devices at real-time frame rates. A key component of our system is a novel regression algorithm that accurately infers the facial motion parameters from 2D video frames of an ordinary web camera. Compared with the state-of-the-art facial shape regression algorithm [1], which takes a two-step procedure to track facial animations (i.e., first regressing the 3D positions of facial landmarks, and then computing the head poses and expression coefficients), we directly regress the head poses and expression coefficients. This one-step approach greatly reduces the dimension of the regression target and significantly improves the performance of the tracking process while preserving the tracking accuracy. We further propose to collect the training images of the user under different lighting environments, and make use of the data to learn a user-specific regressor, which can robustly handle lighting changes that frequently occur when using mobile devices.

Keywords: Video tracking, 3D avatars, Facial performance, User-specific blendshapes, Shape regression

1. Introduction

Performance-based facial animation has attracted a lot of research attention in the fields of computer vision and computer graphics. And facial animation techniques based on special equipment (e.g., facial markers and structure light projectors) have achieved great success in film and game production. However, these techniques cannot be adopted by ordinary users who do not have access to these equipment. With the rapid spread of mobile devices, a real-time, single-camera based facial animation system could play an important role in many applications like video chat, social network and online games.

Weise et al. [2] developed a real-time facial animation system that utilizes depth and color information from a Kinect RGBD camera. It, however, can only work in indoor environments. Moreover, none of existing mobile devices integrates a depth camera. Earlier this year, Cao et al. [1] proposed a more practical solution for ordinary users, which only needs a single web camera. Although most mobile devices contains a video camera, there still exist two major issues preventing the system from running on mobile devices efficiently: first, the 3D shape regression algorithm in [1] is still time consuming, significantly limiting the system's performance on mobile devices; second, the limitation of handling dramatic lighting changes becomes very serious on mobile devices which could be frequently used under different environments.

In this paper, we present a facial animation system suitable for mobile devices by making two contributions. First, instead of using a two-step procedure to track facial animations (i.e., first regressing the 3D positions of facial landmarks, and then computing the head poses and expression coefficients.) as in [1], we directly regress the facial motion parameters (i.e., head poses and expression coefficients) from 2D video frames of an ordinary web camera. This one-step approach greatly reduces the dimension of the regression target and significantly improves the performance of the tracking process while preserving the tracking accuracy. Second, we propose to collect the training images of the user under different lighting environments, and make use of the data to learn a user-specific regressor. This regressor can robustly handle lighting changes that frequently occur when using mobile devices.

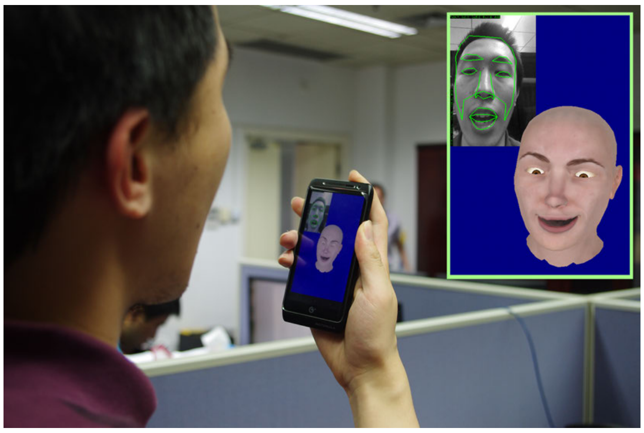
Figure 1: Our facial animation system running at over 30 fps on a smart phone. The insert on the top right is the screenshot of the phone, showing the input video frame, reconstructed landmarks and avatar.

1.1. Related Work

Many facial animation systems work by tracking facial features and use the information derived from these features to animate digital avatars. Tracking algorithms based on special equipments (e.g., markers [3, 4], structure lights [5, 6], and camera arrays [7, 8]) have been widely used in film production where high-fidelity animations are required. While being able to generate high-quality animations, these methods are not suitable for ordinary users due to the requirement of special equipments.

Face tracking/animation based on simple devices, such as an ordinary video camera, has also been explored. According to how facial features are extracted from video, these tracking algorithms can be divided into two main categories: *optimization-based* and *regression-based*. Optimization-based methods always formulate a fitting energy related to the matching error between the features and image appearance. For example, Active Appearance Model (AAM) methods [9, 10] reconstruct the face shape based on an appearance model by minimizing the error between the reconstructed appearance and image appearance. Regression-based methods learn a regression function that maps the image appearance to the target output. Some methods [11, 12] use the parametric models (e.g., AAM) and regress the parameters. Cao et al. [13] directly regress facial landmarks without using any parametric shape models by minimizing the alignment error over training data in a holistic manner. All these algorithms learn a model or a regressor from a large pool of training data, and try to find a general routine to track facial features in any face image. However, such general models or regressors may not generate satisfactory tracking results for images of a specific person, especially for non-frontal faces and exaggerated expressions.

Weise et al. [2] developed a novel facial animation system based on a Microsoft Kinect RGBD camera. They combine the geometry and texture registration with animation priors into a single optimization problem. With the help of the user-specific blendshape model pre-constructed for the user, they formulate the optimization problem as a maximum a posterior estimation in a reduced parameter space. Cao et al. [1] demonstrated comparable tracking results using a single web camera. They developed a novel user-specific 3D shape regressor, which can regress the 3D positions of landmarks directly from 2D video frames. These 3D landmarks are then used to track the facial motion, including the rigid transformation and non-rigid blendshape coefficients, which can be mapped to any digital avatar.

1.2. Overview

We propose to learn a user-specific regressor that can directly regress facial motion parameters from 2D video frames. To learn this user-specific regressor, we need to prepare the training data as in [1] (see Section 2.1). We then construct the training data for the regressor in Section 2.2, and learn a facial motion regressor in Section 2.3. In Section 2.4, at run time, the regressor takes the video frame and the previous frame's facial motion parameters as input and compute the facial motion parameters for the current frame.

In Section 3, we describe two strategies to handle dramatic lighting changes in the system. First, we collect images taken under different lighting environments and train them together to broaden the range of lighting environments that can be handled in our regressor. Second, to deal with the global intensity adjustment caused by the camera white balance, we perform a histogram normalization to normalize all training images and runtime video.

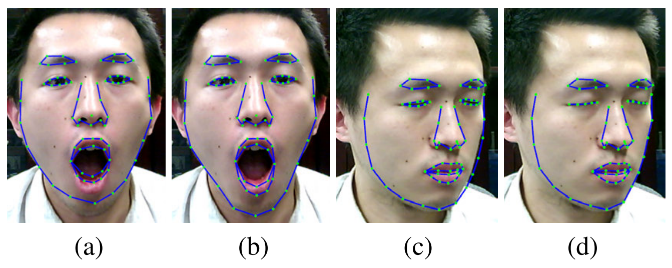

Figure 2: Two captured images with automatically located landmarks (a)(c), and the manually corrected landmarks (b)(d).

2. Facial Motion Regression

Our facial motion regression algorithm is an extension of the 3D facial shape regression algorithm proposed by Cao et al. [1]. Unlike in [1] where the regression target is the 3D facial shape, our regression target is the facial motion parameters that describe the rigid and non-rigid facial motion. We take an image I and an initial guess of its facial motion parameters $\mathbf{x}^c$ as input, iteratively update $\mathbf{x}^c$ in an additional manner, and output the final motion parameters. Like in [1], the regression algorithm consists of four parts: training data preparation, training data construction, training and run-time regression.

2.1. Training Data Preparation

Following [1], we need to collect training data from the user.

Image capturing and labelling. First, we capture 60 images $\{I_i\}$ of the user performing a sequence of predefined facial poses and expressions. Then the 2D facial alignment algorithm described in [13] is used to automatically locate 75 facial landmarks on each image. These landmarks describe the relatively distinctive facial features on the image, such as the face contour, eye boundaries, and mouth boundary. The user can also manually adjust the landmark positions if the automatic detection results are not accurate enough. Two examples of labelled images are shown in Figure 2.

User-specific blendshape generation. Making use of these labelled images and the FaceWarehouse database [14], we extract the user-specific blendshapes $\mathbf{B} = \{B_0, B_1, ..., B_{46}\}$ and obtain the intrinsic matrix $\mathbf{Q}$ of the camera by minimizing the following energy as in [1]:

$$E_t(\mathbf{Q}, \mathbf{w}_{id}, \mathbf{w}_{exp,i}, \mathbf{M}_i) = \sum_{i=1}^{60} \sum_{k=1}^{75} \left\| \Pi_{\mathbf{Q}} \left(\mathbf{M}_i F^{(v_k)} \right) - \mathbf{s}_i^{(k)} \right\|^2, \qquad (1)$$
$$F = C_r \times_2 \mathbf{w}_{id}^T \times_3 \mathbf{w}_{exp,i}^T,$$

where $\Pi_{\mathbf{Q}}$ is the projection operator which transforms the 3D position in camera coordinates to a 2D position in screen coordinates, $\mathbf{M}_i$ is the rigid transformation matrix for each image, F is the reconstructed mesh by tensor contraction, C_r is the core tensor from FaceWarehouse, $\mathbf{w}_{id}$ and $\mathbf{w}_{exp,i}$ are the column vector of identity and expression weights in the tensor (the identity weights are the same for all the images), $\mathbf{s}_i^{(k)}$ is the 2D position of the k-th landmark for image I_i, and v_k is the related vertex index on the mesh.

Following the assumption of the ideal pinhole camera, the intrinsic matrix $\mathbf{Q}$ of the camera can be represented as

$$\mathbf{Q} = \begin{pmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{pmatrix}, \tag{2}$$

where f represents the focal length in terms of pixel, (c_x, c_y) represents the principal point and is the center of the image. Thus f is the only unknown parameter in the matrix.

We solve for all the unknowns $(\mathbf{Q}, \mathbf{w}_{id}, \mathbf{w}_{exp,i}, \mathbf{M}_i)$ using the same approach described in [1]. The expression blendshapes $\{B_i\}$ for the user can be computed as

$$B_i = C_r \times_2 \mathbf{w}_{id} \times_3 (\check{\mathbf{U}}_{exp}\mathbf{d}_i), \quad 0 \leq i < 47, \tag{3}$$

where $\check{\mathbf{U}}_{exp}$ is the truncated transform matrix for the expression mode in FaceWarehouse, and $\mathbf{d}_i$ is an expression weight vector with value 1 for the i-th element and 0 for other elements.

Motion parameter recovery. With the user-specific blendshapes and camera's intrinsic matrix, we can recover the 3D face meshes for all input images. For each image, we minimize the the error between the labeled 2D landmark positions and the projection of the 3D landmark vertices, and solve the facial motion parameters

$$\arg\min_{\mathbf{M},\mathbf{a}} \sum_{l=1}^{75} \left\| \Pi_{\mathbf{Q}} \left(\mathbf{M} \left(B_0 + \sum_{i=1}^{46} \alpha_i B_i \right)^{(v_l)} \right) - \mathbf{q}^{(l)} \right\|^2, \tag{4}$$

where $\mathbf{a} = \{a_1, a_2, ..., a_{46}\}$ is the expression coefficients $\mathbf{M}$ is the rigid transformation matrix. Note that the setup images consist of pre-defined standard expressions. To make the blendshape expression coefficients consistent with pre-defined standard blendshape coefficients $\mathbf{a}^*$, we also add a regularization term to the above energy: $||\mathbf{a} - \mathbf{a}^*||^2$.

We further represent the rigid transformation matrix $\mathbf{M}$ as a $4D$ rotation quaternion vector $\mathbf{R}$ and a $3D$ translation vector $\mathbf{T}$. Finally, for each input image I_i, we concatenate its motion parameters to form a $46+4+3 = 53D$ vector: $\mathbf{x}_i^g = (\mathbf{a}_i^g, \mathbf{R}_i^g, \mathbf{T}_i^g)$.

2.2. Training Set Construction

For each image and the computed motion parameter vector $(I_i, \mathbf{x}_i^g)$, we construct a set of augmented parameter vectors as its initial guess in the regression process. Specifically, for each parameter vector $\mathbf{x}_i^g = (\mathbf{a}_i^g, \mathbf{R}_i^g, \mathbf{T}_i^g)$, the augmented parameter vectors are computed as follows:

- **Translation**: $(\mathbf{a}_i^g, \mathbf{R}_i^g, \mathbf{T}_i^g + \Delta_{ij})$, where translation vectors Δ_{ij}, $1 \leq j \leq 27$, form a $3 \times 3 \times 3$ jittered grid.
- **Rotation**: $(\mathbf{a}_i^g, \mathbf{R}_{i'}^g, \mathbf{T}_i^g + \Delta_{ij})$, where $\{\mathbf{R}_{i'}^g, i' \neq i\}$ cover the rotations of all other input images. For each $\mathbf{R}_{i'}^g$, several random translations Δ_{ij}, $1 \leq j \leq 32$ are applied.
- **All other expressions**: $(\mathbf{a}_{i'}^g, \mathbf{R}_i^g, \mathbf{T}_i^g + \Delta_{ij})$, where $\{\mathbf{a}_{i'}^g, i' \neq i\}$ cover the blendshape coefficients of all other input images. For each $\mathbf{a}_{i'}^g$, several random translations Δ_{ij}, $1 \leq j \leq 10$ are applied.

We denote each augmented parameter vector as $\mathbf{x}_{ij}^c$, combing which we can construct a training tuple as $(I_i, \mathbf{x}_i^g, \mathbf{x}_{ij}^c)$. For description simplicity, we denote all these training data as $\{(I_i, \mathbf{x}_i, \mathbf{x}_i^c)\}$.

2.3. Regressor training

With these N training data $\{(I_i, \mathbf{x}_i, \mathbf{x}_i^c)\}$, we train a motion parameter regression function from $\mathbf{x}_i^c$ to $\mathbf{x}_i$ based on the pixel intensity of image I_i. We use the two-level boosted regression algorithm in [1], as described in Algorithm 1. In the first level, we reconstruct the 3D landmarks from the current motion parameters $\mathbf{x}_i^c$, and sample pixels on image I_i to construct the appearance vector. In the second level, we build a sequence of weak regressors based on this appearance vector, and update the current motion parameters $\mathbf{x}_i^c$ by minimizing the error between $\mathbf{x}_i$ and $\mathbf{x}_i^c$.

Appearance vector generation. We need to randomly generate P points around the face mesh and use them to sample the images. Each point is represented as an index-offset pair $(k_p, \mathbf{d}_p)$, where k_p is the landmark index and $\mathbf{d}_p$ is the offset. In our experiment, we randomly choose a landmark k_p, and determine a random offset as in [1].

For each training data $(I_i, \mathbf{x}_i, \mathbf{x}_i^c)$, we can first reconstruct the 3D landmark positions at the rest pose (i.e., without rotation and translation), $S_i^c = \{\mathbf{s}_{ik}^c\}$, by

$$\mathbf{s}_{ik}^c = \left(B_0 + \sum_{j=1}^{46} \alpha_{ij}^c B_j \right)^{(v_k)}. \tag{5}$$

Based on the positions of these landmarks at rest pose, we generate the P points at the rest pose according to the index-offset pairs $\{(k_p, \mathbf{d}_p)\}$. We then transform these 3D points via the transformation in $\mathbf{x}_i^c$, and then project the transformed points to the image space via the mapping function $\Pi_{\mathbf{Q}}$:

$$\mathbf{u}_p = \Pi_{\mathbf{Q}} \left(\mathbf{R}_i^c \left(\mathbf{s}_{ik_p}^c + \mathbf{d}_p \right) + \mathbf{T}_i^c \right). \tag{6}$$

The image intensity values at $\{\mathbf{u}_p\}$ construct the appearance vector $V_i = App(I_i, \mathbf{x}_i^c, \{(k_p, \mathbf{d}_p)\})$. For each appearance vector V_i, we generate P^2 index-pair features by computing the differences between each pair of elements in V_i.

In the second level, based on the appearance vector V_i obtained in the first level, we construct a set of weak regressors in an additive manner, which minimize the error between the ground truth parameters $\mathbf{x}_i$ and the current parameters $\mathbf{x}_i^c$. To train these weak regressors, we need to efficiently choose the features from the P^2 index-pair features.

Feature selection. We choose the feature that has the highest correlation with the regression target. To select the effective features, we first generate a random $53D$ direction Y. For each training data, we calculate the difference vector by $\delta\mathbf{x}_i = \mathbf{x}_i - \mathbf{x}_i^c$. We then project this difference vector to the random direction Y to produce a scalar. We choose the index-pair with the highest correlation with this scalar.

In the parameter vector $\mathbf{x} = (\mathbf{a}, \mathbf{R}, \mathbf{T})$, the three components are in different spaces with different units. We thus cannot handle them in the same way. We need to carefully choose weights

Algorithm 1 Facial motion regression training

```
Input: $N$ Training data $(I_i, \mathbf{x}_i, \mathbf{x}_i^c)$
Output: Two-level boosted regressor
  /* level one */
  for $t = 1$ to $T$ do
    $\{k_p, \mathbf{d}_p^t\} \leftarrow$ randomly generate $P$ offsets
    for $i = 1$ to $N$ do
      $V_i \leftarrow App(I_i, \mathbf{x}_i^c, \{(k_p, \mathbf{d}_p)\})$
      Compute the $P^2$ feature values for $V_i$
    end for

    /* level two */
    for $k = 1$ to $K$ do
      for $f = 1$ to $F$ do
        $Y_f \leftarrow$ randomly generate a direction
        for $i = 1$ to $N$ do
          $\delta\mathbf{x}_i \leftarrow \mathbf{x}_i - \mathbf{x}_i^c$
          $\delta\mathbf{x}_i' \leftarrow$ scale $\delta\mathbf{x}_i$
          $c_i \leftarrow \delta\mathbf{x}_i' \cdot Y_f$
        end for
        Find the index-pair with the highest correlation with
        $\{c_i\}$, and randomly choose a threshold
      end for
      for $i = 1$ to $N$ do
        Calculate the features in $V_i$ using $F$ index-pairs
        Compare the features to the thresholds and determine
        which bin the data belongs to
      end for
      for $i = 1$ to $2^F$ do
        Compute $\delta\mathbf{x}_{b_i}$ according to Eq. (7)
        for each training data $l \in \Omega_{b_j}$ do
          $\mathbf{x}_l^c \leftarrow \mathbf{x}_l^c + \delta\mathbf{x}_{b_i}$
        end for
      end for
    end for
  end for
```

to reflect their influences on the features. According to our experiments, the weight values listed in Table 1 can generate satisfactory results.

Coefficients **a**	Rotation **R**	Translation **T**
4	8	$1/\sigma$

Table 1: The scaling factors for different components in motion parameters, where σ is the standard deviation of translation in the training data.

Fern construction. We repeat F times to select F different index-pair features. For each index-pair feature, we assign a random threshold value between the minimum and maximum differences of element pairs in the appearance vectors. These features and related thresholds are used to build a primitive regressor called *fern*. In each fern, we calculate the F index-pair features for each training data, compare the feature values with the threshold values, and determine which bin in the index-pair space it belongs to. In each bin b, for the training data falling into it, which we denote them as Ω_b, we try to find an additional offset that minimizes the error between the ground truth parameters and current parameters. Following [1], this additional offset can be calculated as

$$\delta\mathbf{x}_b = \frac{1}{1+\beta/|\Omega_b|}\frac{\sum_{i\in\Omega_b}\left(\mathbf{x}_i - \mathbf{x}_i^c\right)}{|\Omega_b|}, \tag{7}$$

where $|\Omega_b|$ is the number of training data falling into bin b, β is the free shrinkage parameter that helps to overcome the overfitting when the number training in the bin is too small. We take $\delta\mathbf{x}_b$ as the regression output for each bin, and for all the data in the bin, we update their current parameters by $\mathbf{x}^c = \mathbf{x}^c + \delta\mathbf{x}_b$.

As described in [1], we process T stages in the first level and K stages in the second level, iteratively build the primitive regressor, calculate the regression output and update current motion parameters for each training data. In our experiment, we choose $T = 7, P = 300, K = 200, F = 5, \beta = 150$.

Algorithm 2 Runtime regression

```
Input: Previous frame's motion parameters $\mathbf{x}'$, current frame's
image $I$
Output: Current frame's motion parameters $\mathbf{x}$
  $S' \leftarrow RC(\mathbf{x}')$
  Get the parameter vector $\mathbf{x}_s$ in $\{\mathbf{x}_i^g\}$ most similar to $\mathbf{x}'$
  $\mathbf{R}_b \leftarrow \mathbf{R}_s(\mathbf{R}')^{-1}$
  $S'^* \leftarrow= \mathbf{R}_b S'$
  $\{\mathbf{x}_l^*\} \leftarrow$ Get $L$ parameter vectors most similar to $S'^*$ in the
  training data
  for $l = 1$ to $L$ do
    $\mathbf{M}_l^* \leftarrow merge(\mathbf{R}_s, \mathbf{T}_l)$
    $\mathbf{M}_b \leftarrow \mathbf{M}'(\mathbf{M}_l^*)^{-1}$
    $\mathbf{x}_l \leftarrow \mathbf{M}_b \mathbf{x}_l^*$
    for $t = 1$ to $T$ do
      $V_l \leftarrow App(I, \mathbf{x}_l, \{(k_p, \mathbf{d}_p)\})$
      for $k = 1$ to $K$ do
        Get the $F$ index-pairs recorded during training
        Calculate the $F$ appearance feature values for $V_l$
        Use the values to locate its bin $b$ in the fern
        Get $\delta\mathbf{x}_b$ in $b$
        $\mathbf{x}_l \leftarrow \mathbf{x}_l + \delta\mathbf{x}_b$
      end for
    end for
  end for
  $\mathbf{x} \leftarrow$ Compute the median result of $\{\mathbf{x}_l, 1 \le l \le L\}$
```

2.4. Runtime Regression

With the facial motion regressor trained in the last section, we can compute the facial motion parameters for input video frame I in real time. We start from $\mathbf{x}'$, the regression result of the previous frame and find motion parameters similar to $\mathbf{x}'$ from the training set as the initial parameters for regression.

Initial parameters. Similar to the training process, we first reconstruct the 3D landmark vector $S' = \{\mathbf{s}_k'\}$ using the previous frame's parameter vector $\mathbf{x}'$ via

$$\mathbf{s}'_k = \mathbf{R}'\left(B_0 + \sum_{j=1}^{46} \alpha'_j B_j\right)^{(v_k)}. \tag{8}$$

Notice that we omit the translation $\mathbf{T}'$ here, and we denote this reconstructed operator as $S = RC(\mathbf{x})$. For the ground truth motion parameter of each input image $\mathbf{x}_i^g$, we reconstruct the 3D landmark positions via $S_i^g = RC(\mathbf{x}_i^g)$. We then find the landmark vector S_s from $\{S_i^g\}$, which is most similar to S', and take the corresponding parameters as the most similar parameters $\mathbf{x}_s$. To compare two landmark vectors, we directly compute the Euclidean distance between them. We then transform S' to the pose of $\mathbf{x}_s$ via rotation $\mathbf{R}_b = \mathbf{R}_s(\mathbf{R}')^{-1}$. Finally, we compare the rotated landmarks $\mathbf{R}_b S'$ with the reconstructed landmarks $\{S_i^c = RC(\mathbf{x}_i^c)\}$ from all augmented parameters in the training data, find the L most similar ones, and take the related parameters as $\{\mathbf{x}_l^*\}$.

Given that we rotate the landmarks according to $\mathbf{R}_b$ in finding the similar landmarks and omit all translations in choosing similar parameters, we need to transform each similar parameter vector $\mathbf{x}_l^*$ to the pose of the previous frame's parameters $\mathbf{x}'$. We first combine the rotation $\mathbf{R}_s$ in $\mathbf{x}_s$ and the translation in $\mathbf{x}_l^*$, and merge them as a new transformation $\mathbf{M}_l^* = merge(\mathbf{R}_s, \mathbf{T}_l^*)$. Thus the transformation from $\mathbf{x}_l$ to $\mathbf{x}'$ pose is $\mathbf{M}_b = \mathbf{M}'\left(\mathbf{M}_l^*\right)^{-1}$, where $\mathbf{M}'$ is the transformation in $\mathbf{x}'$. Applying the $\mathbf{M}_b$ on $\mathbf{x}_l^*$, we finally find the L initial parameter vectors: $\mathbf{x}_l = \mathbf{M}_b\mathbf{x}_l^*$.

Appearance vector. We take each parameter vector $\mathbf{x}_l$ as the initial parameters in the regressor, and update it through the regressor. In each stage of the first level regression, we construct the appearance vector based on current motion parameters $\mathbf{x}_l$ and the index-offset pairs $\{(k_p, \mathbf{d}_p)\}$.

Fern passing. In the second level of regression, we take the appearance vector V_l, and pass all the ferns in an additional manner. For each fern, we take the recorded F index-pairs and calculate the related features from V_l. These features are compared with related threshold values to locate the bin b among the 2^F bins. We then take the regression output δ_b associated with b and update the parameters via $\mathbf{x}_l = \mathbf{x}_l + \delta\mathbf{x}_b$.

For each initial parameter vector, after passing all ferns in the regressor, we get an updated parameter vector. We finally calculate the median of all the updated parameter vectors $\{\mathbf{x}_l\}$ as the final result $\mathbf{x}$. The runtime regression algorithm is described in Algorithm 2.

Note that to get realistic facial animation, the range of expression coefficients needs to be restricted to between 0 and 1. To do this, we clamp the coefficients to between 0 and 1 each time the parameter vector is updated in the regression process. Such a simple approach can generate satisfactory results in our experiments. And interestingly, the results from our motion parameter regressor are already very smooth and temporally coherent. Although it is possible to consider animation prior in a postprocessing step like in [1], we found it is unnecessary to do this in our experiment.

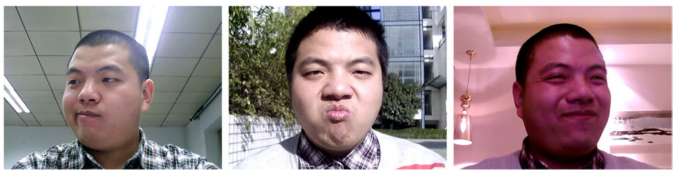
Figure 3: Captured images under different light conditions. From left to right: in the office, outdoor with direct sunshine, and in a hotel with dim light.

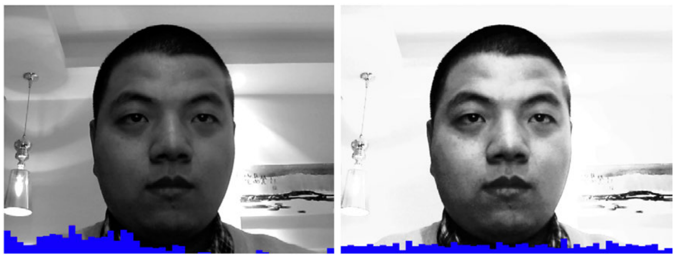
Figure 4: Histogram normalization. Left: before normalization, most pixels are located in the dark region in histogram and the face appears very dark; right: after normalization, the histogram distributes uniformly and the face region becomes brighter.

3. Handling Lighting Changes

Unlike desktop PCs which are used indoors most of the time, mobile devices may be used under different environments. For example, if the user is using our system while walking, the lighting and background are changing all the time. Cao et al. [1] can handle lighting changes to a modest extent, but may fail to generate accurate tracking results if the lighting environment changes dramatically.

Our first strategy is to include training data from multiple environments to train the regressor. We collect the user's setup images under different environments. In our experiments, as shown in Figure 3, we include the images in the office, outdoors with direct sunshine, and in a hotel room with dim light.

Second, mobile cameras often perform white balancing. This changes the overall intensity of the image, making the whole image darker or brighter. Given that we compare the index-pair features acquired from images, which are absolute values related to pixel's intensities, the global adjustment will make the intensity value inconsistent in range. To handle this problem, we additionally perform a histogram normalization on the appearance vectors, in both the training and run-time testing.

An example of histogram normalization is shown in Figure 4. Due to the bright background, white balance makes the face appear dark. Histogram normalization helps to adjust the intensity of the entire face, unify the global intensity of the image, and make the tracking more robust.

4. Experimental Results

We have implemented the system on a PC with an Intel Core i7 (3.5 GHz) CPU, with an ordinary web camera recording 640×480 images at 30 fps. The runtime algorithm runs at over 200 fps on this device, eight times faster than [1]. We also tested the regression algorithm on a Motorola MT788 cell phone, with an Intel Atom 2.0 GHz CPU and Android 4.0 operating system.

RMSE	< 3.5 pixels	< 5.0 pixels	< 6.5 pixels	< 8.0 pixels	Avg. Err. (pixels)
Basic	3.0%	17.4%	29.9%	44.3%	12.12
Basic + AD	11.4%	46.1%	69.5%	80.2%	6.18
Basic + HN	22.2%	46.7%	60.5%	77.8%	7.22
Basic + AD + HN	61.6%	88.0%	97.6%	100.0%	3.31

Table 4: Percentages of frames with RMSE less than given thresholds and the average errors using four different methods,

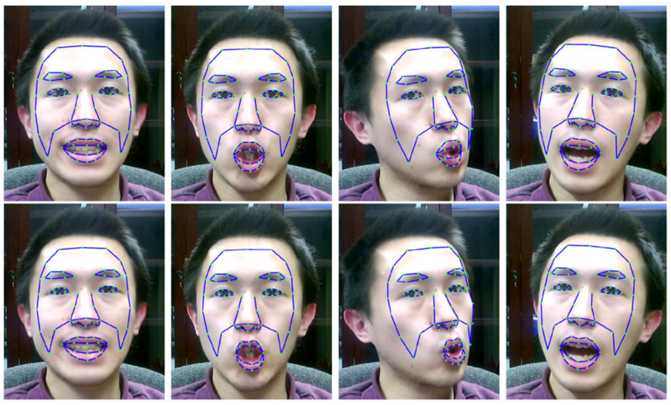
Figure 5: Comparison with Cao et al. [1]. Top: our tracking results. Bottom: results from Cao et al. [1].

Target	2D shape	3D shape	Ours
Dimension	150	225	53
Training (min)	8	10	3.5
Regression (ms)	4.3	6.2	1.8
Fitting (ms)	/	8.2	0.0
Runtime total (ms)	/	**14.4**	**1.8**
Memory (MB)	21.2	32.3	9.17

Table 2: The training timing and memory consumption of different regression algorithms. All results are trained using the same training data.

The performance is still very high, at about 30 fps (Figure 1). Figure 7 shows some other results using our system. Please see the accompanying video[1] for animation demos.

Unlike [13] and [1] which regress face shapes, we directly regress facial motion parameters, which are represented as a $53D$ vector. The dimension of our regression target is thus much lower than the regression targets of [13] ($75 \times 2D$) and [1] ($75 \times 3D$). The efficiency of our algorithm thus comes from several aspects: 1) the reduced dimension of regression target greatly improves the regression efficiency, in both the training and runtime testing; 2) given that we have the rotation $\mathbf{R}$ and translation $\mathbf{T}$ in each iteration, we do not need to calculate the transformation via singular value decomposition (SVD); 3) we do not need to fit facial parameters after the regression, which contains the SVD and BFGS solving procedure in [1]. In Table 2, we compare the timings in training, testing and the memory consumption with different regression targets.

To compare the accuracy of different algorithms, we take a set of video sequences as input, and use the generated regressors to compute the landmark positions (or parameters) respectively. In Table 3, we measure the errors (in pixels) between the screen projection of landmarks obtained by the two regression algorithms and the ground truth positions, which are manually labelled for each frame.

[1]http://gaps-zju.org/publication/2013/mface-divx.avi

RMSE	< 2 px	< 3 px	< 4 px	Avg. Err.(px)
Cao et al. [1]	66.2%	87.1%	97.1%	2.08
Ours	20.1%	65.5%	95.7%	2.93

Table 3: Percentages of frames with RMSE less than given thresholds and the average errors.

Figure 5 shows the comparisons of our algorithm with Cao et al. [1]. It can be seen that our algorithm achieves comparable tracking results. We note that the landmark positions reconstructed from our motion parameters are not as accurate as those generated by Cao et al.'s direct shape regression. Some facial features of our results do not match very well. In practice we found that the animations generated from our motion parameters are visually plausible and resemble the user's facial motion very well. Please see the accompanying video for animation results.

To evaluate the effects of our two strategies for handling lighting changes, we designed four different regressors: the basic regressor trained with images captured under one lighting environment (**basic**), the regressor trained with images captured under tree lighting environment (**basic+AD**), the regressor trained with images captured under one lighting environment using histogram normalization (**basic+HN**), and the regressor trained with images captured under tree lighting environment using histogram normalization (**basic+AD+HN**). We tested the four regressors on a testing video recorded under a new lighting environment which is not used in the training process. From Table 4 and Figure 6, it can be seen that **Basic+HN+AD** can produce the best results in our experiment.

5. Conclusion

We have introduced a novel facial motion regression algorithm and shown that it can be used to generate accurate facial animations in real time even on mobile devices. We also described two strategies to improve the robustness of our algorithm to lighting changes. The whole system provides a practical solution to real-time facial animation on mobile devices.

Acknowledgements

This work was supported by the NSF of China (No. 61003145 and No. 61272305) and the National High-Tech R&D program of China (No. 2012AA011503).

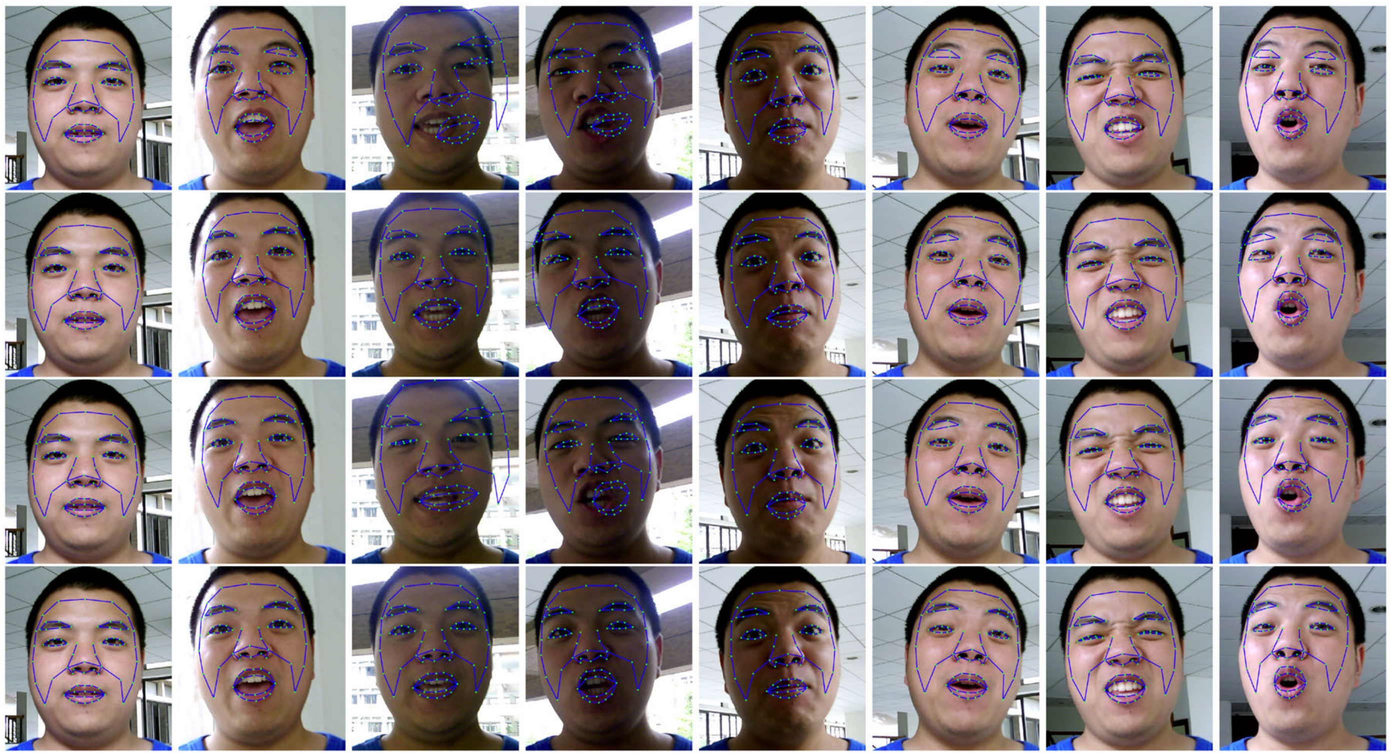

Figure 6: Comparison of different algorithms for dynamic light adaptation. From top to bottom: **Basic**, **Basic+AD**, **Basic+HN**, **Basic+AD+HN**.

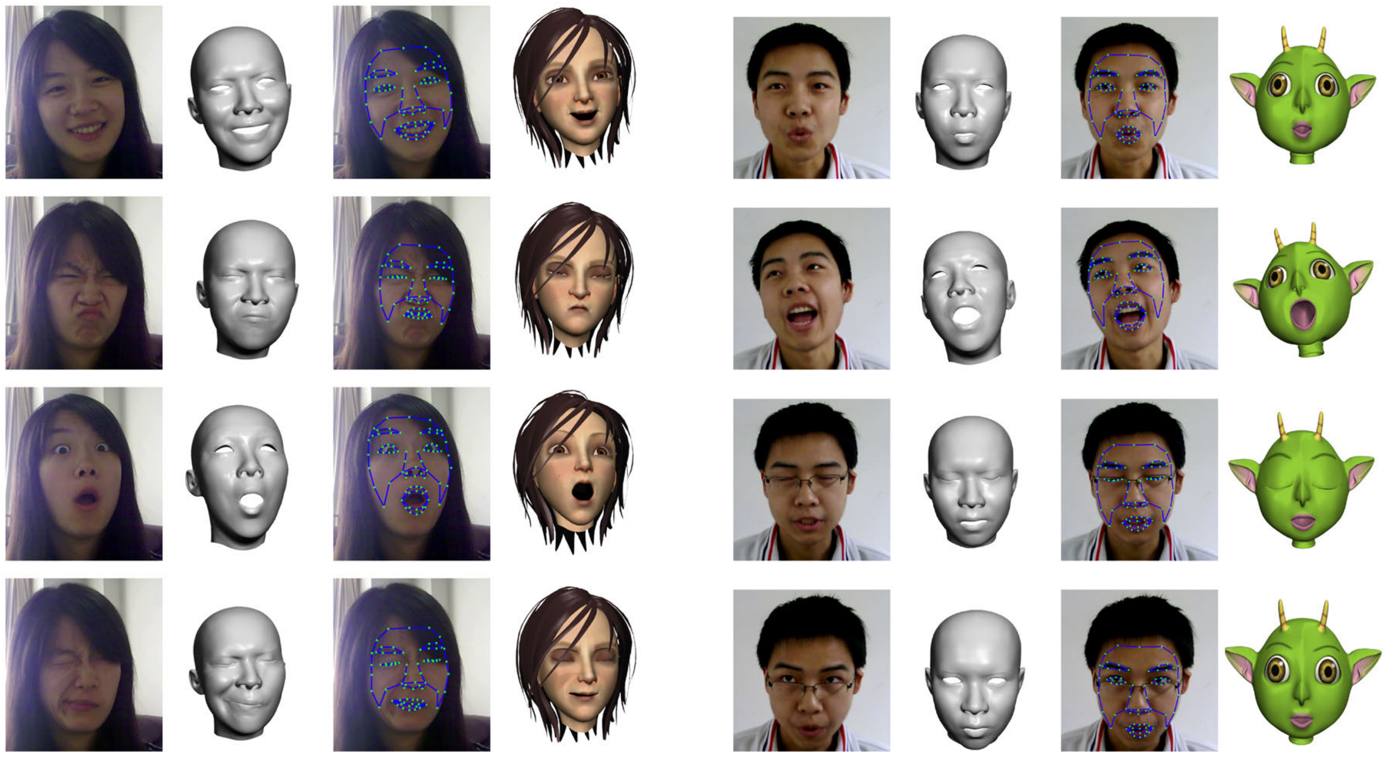

Figure 7: Real-time facial animation results of two examples. For each example, from left to right: input video frame, tracked mesh, reconstructed landmark, driven avatar.

References

[1] C. Cao, Y. Weng, S. Lin, K. Zhou, 3d shape regression for realtime facial animation, ACM Trans. Graph. 1, 2, 3, 4, 5, 6

[2] T. Weise, S. Bouaziz, H. Li, M. Pauly, Realtime performance-based facial animation, ACM Trans. Graph. 30 (4) (2011) 77:1–77:10. 1, 2

[3] L. Williams, Performance driven facial animation, in: Proceedings of SIGGRAPH, 1990, pp. 235–242. 1

[4] H. Huang, J. Chai, X. Tong, H.-T. Wu, Leveraging motion capture and 3d scanning for high-fidelity facial performance acquisition, ACM Trans. Graph. 30 (4) (2011) 74:1–74:10. 1

[5] L. Zhang, N. Snavely, B. Curless, S. M. Seitz, Spacetime faces: high resolution capture for modeling and animation, ACM Trans. Graph. 23 (3) (2004) 548–558. 1

[6] T. Weise, H. Li, L. V. Gool, M. Pauly, Face/off: Live facial puppetry, in: Eurographics/SIGGRAPH Symposium on Computer Animation, 2009. 1

[7] D. Bradley, W. Heidrich, T. Popa, A. Sheffer, High resolution passive facial performance capture, ACM Trans. Graph. 29 (4) (2010) 41:1–41:10. 1

[8] T. Beeler, B. Bickel, P. Beardsley, R. Sumner, M. Gross, High-quality single-shot capture of facial geometry, ACM Trans. Graph. 29 (4) (2010) 40:1–40:9. 1

[9] T. Cootes, G. Edwards, C. Taylor, Active appearance models, ECCV'98 (1998) 484–498. 2

[10] I. Matthews, S. Baker, Active appearance models revisited, International Journal of Computer Vision 60 (2) (2004) 135 – 164. 2

[11] P. Dollár, P. Welinder, P. Perona, Cascaded pose regression, in: Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on, IEEE, 2010, pp. 1078–1085. 2

[12] P. Sauer, T. Cootes, C. Taylor, Accurate regression procedures for active appearance models, in: British Machine Vision Conference, 2011. 2

[13] X. Cao, Y. Wei, F. Wen, J. Sun, Face alignment by explicit shape regression, in: Computer Vision and Pattern Recognition (CVPR), 2012, pp. 2887–2894. 2, 6

[14] C. Cao, Y. Weng, S. Zhou, Y. Tong, K. Zhou, Facewarehouse: a 3d facial expression database for visual computing, Tech. rep. (2012). 2