Real-time 3D Neural Facial Animation from Binocular Video

CHEN CAO, Facebook Reality Labs, USA
VASU AGRAWAL, Facebook Reality Labs, USA
FERNANDO DE LA TORRE, Facebook Reality Labs, USA
LELE CHEN, Facebook Reality Labs and University of Rochester, USA
JASON SARAGIH, Facebook Reality Labs, USA
TOMAS SIMON, Facebook Reality Labs, USA
YASER SHEIKH, Facebook Reality Labs, USA

Offline face model fitting

Fig. 1. We present a real-time high-fidelity 3D facial animation method. In the offline face estimation step (a-e), given an input image (a) and a person-specific deep appearance model (PS-DAM), we estimate the model parameters with an analysis-by-synthesis method and track the avatar (d). We propose a new illumination model (i.e., gain and bias map (e)) to relight the avatar (b) and to match the input image (a). We show the $L_2$ error between (a) and (b) as the reconstruction error in (c). In the online (real-time) facial animation step (f-j), we have a pair of images as input (f,h) and we can accurately track the face and drive the 3D avatar (g,i). The animated 3D avatar across different views is shown in (j).

We present a method for performing real-time facial animation of a 3D avatar from binocular video. Existing facial animation methods fail to automatically capture precise and subtle facial motions for driving a photo-realistic 3D avatar "in-the-wild" (i.e., variability in illumination, camera noise). The novelty of our approach lies in a lightweight process for specializing a personalized face model to new environments that enables extremely accurate real-time face tracking anywhere. Our method uses a pre-trained high-fidelity personalized model of the face that we complement with a novel illumination model to account for variations due to lighting and other factors often encountered in-the-wild (e.g., facial hair growth, makeup, blemishes). Our approach comprises two steps. First, we solve for our illumination model’s parameters by applying analysis-by-synthesis on a short video recording. Using the pairs of model parameters (rigid, non-rigid) and the original images, we learn a regression for real-time inference from the image space to the 3D shape and texture of the avatar. Second, given a new video, we fine-tune the real-time regression model with a few-shot learning strategy to adapt the regression model to the new environment. We demonstrate our system’s ability to precisely capture subtle facial motions in unconstrained scenarios, in comparison to competing methods, on a diverse collection of identities, expressions, and real-world environments.

CCS Concepts: · Computing methodologies → Motion capture; Mixed / augmented reality.

Additional Key Words and Phrases: Facial animation, Facial modeling, Differentiable rendering, Augmented reality

ACM Reference Format:

1 INTRODUCTION

Markerless facial motion capture has been a long-standing computer graphics and computer vision problem over the last three decades.
In many applications, such as animating characters in movies or systems to promote a sense of social presence in AR/VR, driving avatars from video with high-degree of photo-realism is required. In these scenarios, a major challenge is to be able to transfer subtle expressions (e.g., perfect eye contact, lip-chewing, tongue movement, blinking) in real-time under a variety of lighting conditions.

There are two main technical challenges: first, we need to correctly decouple rigid (i.e., 3D head pose) and non-rigid motion (i.e., facial expression). Second, we need to render the user’s appearance in the avatar. To decouple the rigid/non-rigid motion, a necessary component is a precise facial tracking/alignment mechanism that achieves sub-pixel accuracy. Precise and dense facial feature tracking is challenging in natural scenarios due to sensor noise and variability in lighting. Lighting changes always dominate the marginal information for detail facial information. If we try to decouple the illumination from an image, we usually destroy the information required for subtle facial motions. To render user’s appearance in the avatar, in the early days, there were a variety of face tracking methods that use no-model or only shape model for facial animation (e.g., [Decarlo and Metaxas 2000]), but these methods failed catastrophically to recover important facial details such as the interior of the mouth or blinking of the eyes. Alternatively, parametric face models such as Active Appearance Models, Morphable Models, or Deep Appearance Models jointly model the 2D/3D shape and appearance allowing a more precise registration/alignment between the image and the model. In addition, these methods perform tracking by reconstruction with the additional benefit of having a latent code for reconstructing the appearance, providing an elegant solution to rendering the subject’s appearance.

Parameterized face models can be learned across users leading to generic models, or within a subject creating a person-specific (PS) model. Generic models typically achieve robustness to lighting or user variability. However, these generic models do not provide accurate reconstruction for new subjects resulting in a loss of facial expression detail, making them not suitable for our scenario of interest. On the other hand, PS models can achieve the necessary accuracy in tracking and reconstruction. For these reasons, the PS model is typically used in the movie industry (e.g., The curious case of Benjamin Button). However, these models are typically run off-line and not perform well in untrained situations (e.g., different lighting). In this work, we present a lightweight process for specializing a personalized face model to new environments that enables extremely accurate real-time face tracking anywhere.

Our system performs a robust and efficient recovery of rigid and non-rigid motion from sequences with fixed illumination conditions using binocular video. It relies on a person-specific deep appearance model (PS-DAM) [Lombardi et al. 2018], which is learned in a multi-camera capture stage with fixed illumination. Although PS-DAM can generate high fidelity face renderings in real-time, accurately estimating its parameters in unconstrained settings remains challenging due to differences in illumination conditions and facial changes (e.g., facial hair, makeup, glasses). Finally, the entire system, including inference and rendering, needs to work in real-time to support interactive applications such as telepresence.

Our method has two main steps: learning and fine-tuning. In the first step (off-line face estimation in Fig. 1 left), we record several videos of a user and register their face using analysis-by-synthesis [Yuille and Kersten 2006]. In order to compensate for illumination changes and other factors, we introduce a parameterized illumination model and employ a ‘coarse-to-fine’ fitting strategy to simultaneously estimate its parameters along with those of the face model, to avoid converging to poor local minima effectively. We use the resulting pairs of images and parameters to learn a real-time direct regression model. In the second step, we apply this regression model to estimate 3D pose and texture in real-time for a new environment.

To compensate for differences between this environment and the environment where the regressor was trained in, we employ few-shot learning domain-adaptation [Zakharov et al. 2019] which fine-tunes our model from only a few images completely unsupervised. Our results show the effectiveness of this approach for driving a hyper-realistic 3D face model in unconstrained scenarios compared with competing state-of-the-art techniques (see Fig. 1 right).

In summary, this paper makes the following contributions:

- A real-time system for high-fidelity facial animation from binocular video. The system can track subtle expressions in uncontrolled environments (i.e., varying lighting).
- An neural illumination model to compensate from the domain mismatch between the face model and in-the-wild image. We use a coarse-to-fine fitting strategy to achieve better visual minima.
- A few-shot learning approach to fine-tune the pre-trained regression (encoder) to a new environment, in an unsupervised manner.

2 RELATED WORK

Research on face tracking and facial animation has a long history in computer vision, graphics and machine learning [Kleinh et al. 2015; Zollhöfer et al. 2018]. In the following, we focus on the most related techniques and categorize the prior work into parametric face modeling, real-time face tracking, lighting adaptation for face tracking, and non-parametric avatar animation.

2.1 Parametric face modeling

Active Appearance Models (AAMs) [Cootes et al. 2001; Jing Xiao et al. 2004; Matthews and Baker 2004; Tzimiropoulos et al. 2013] and 3D morphable models (3DMM) [Blanz et al. 2003; Blanz and Vetter 1999] have been used extensively to model the space of facial appearance and geometry variation. This family of modeling methods is used to register faces using analysis-by-synthesis, and are often complemented with blendshape models to parameterize the space of expression variation [Lewis et al. 2014]. Extensions have focused on building multilinear models of identity and expression [Cao et al. 2013b; Vlasic et al. 2005], creating very large datasets to span the space of identity variation and texture [Booth et al. 2018], region-based models [Tena et al. 2011], joint shape and motion models [Li et al. 2017], and others [Gerig et al. 2018; Huber et al. 2016].

Recently, deep networks have been used to increase the modeling power of linear and multilinear models [Bagaudinov et al. 2018; Tran et al. 2019; Tran and Liu 2018]. In particular, [Lombardi et al. 2018] propose a Deep Active Appearance (DAM) model based on variational autoencoders [Kingma and Welling 2013] to capture photorealistic geometry and view-dependent appearance of human faces. In this work, we use the DAM model of [Lombardi et al. 2018]
to represent avatar appearance, and build an encoder suitable for real-time animation in unconstrained illumination.

2.2 Real-time face tracking

Real-time face tracking has been widely studied in the past decades [Zollhöfer et al. 2018]. Early real-time methods relied on tracking sparse facial features [Baltrušaitis et al. 2012; Chai et al. 2003; Saragih et al. 2011] as control signals for the face. Animation fidelity improvements have been achieved by increasing the tracking accuracy and density. For example, [Weise et al. 2011] used an RGBD sensor to build and track user-specific blendshape model and drive non-photorealistic avatars via expression retargeting. Subsequent work [Bouaziz et al. 2013; Chen et al. 2013; Li et al. 2013] using RGBD sensors showed that generic blendshape models can be adapted, corrected, or deformed to more accurately track a specific user.

In parallel, monocular RGB-only face tracking performance was improved by using data-driven regression of face landmarks or 3D shape [Cao et al. 2014, 2013a; Kazemi and Sullivan 2014; Xiong and De la Torre 2013] trained on large, annotated face datasets. This approach is robust to varying lighting conditions and unconstrained capture conditions. However, this robustness comes at cost to expression fidelity, and these models are still far from achieving the quality of offline facial performance capture.

To improve fidelity for face tracking, landmarks are also combined with dense appearance-based energies in optimization frameworks [Thies et al. 2016, 2018]. Appearance-based alignment via model fitting has a long history with AAMs and extensions [Jing Xiao et al. 2004; Kahraman et al. 2007; Matthews and Baker 2004], but building the AAM training set often requires capturing additional data in the target environment. Alternatively, [Cao et al. 2015] add face details by additionally regressing expression-dependent wrinkles. [Casas et al. 2015] use dynamic textures to improve photo-realism, and [Nagano et al. 2018] improve realism by using a GAN network to synthesize dynamic textures from a single input image. [Cao et al. 2018] add optical flow constraints and a rigidity prior to stabilize the tracking.

Recent work in monocular face tracking has also explored direct regression of face model parameters using CNNs. For example, [Kim et al. 2017] use a large, synthetically generated set of images to train the CNN regressor. [Tran et al. 2017] use 3DMM fitting to generate the training data, while [Tewari et al. 2017] use differentiable rendering to train an image encoder that produces expression, albedo, and lighting parameters using self-supervision. Recent work on face tracking for VR headsets uses specialized training data generation methods, but also uses CNN-based regression of face parameters [Lombardi et al. 2018; Olszewski et al. 2016; Wei et al. 2019]. A related method to ours is that of [Laine et al. 2017], which achieves high-quality real-time face tracking using direct CNN image regression. There, the training data is built using an offline multiview face tracker, and tracking is limited to the lighting conditions and head pose observed in the capture stage. Our method combines sparse face landmark detection with a CNN-based face parameter regressor in texture-space to estimate detailed expression and pose parameters, and the realtime component is adaptable to novel lighting environments.

Real-time face tracking has also been used for photorealistic face reenactment [Thies et al. 2015, 2016, 2018], where only the face portion of a pre-recorded video is controllable. These techniques were later extended to provide upper-body control by using a 3D proxy [Thies et al. 2018] or using neural rendering to inpaint the areas surrounding the face [Kim et al. 2019, 2018]. While producing photorealistic results, these models can only produce a specific viewpoint and cannot be rendered from arbitrary directions.

2.3 Lighting adaptation for face trackers

Achieving dense, high-quality face tracking beyond coarse facial landmarks usually requires estimating or compensating for lighting conditions at evaluation time. Most existing face tracking methods use spherical harmonics (SH) lighting due to the speed and low degrees of freedom required to estimate them [Sengupta et al. 2018; Tewari et al. 2017; Thies et al. 2016]. [Valgaerts et al. 2012] estimate a spherical harmonics-based lighting environment for offline facial performance capture. [Fyffe et al. 2014] assume an HDR probe of the target environment is available, and use skin reflectance estimates from a light-stage capture to render and match using optical flow. [Yoon et al. 2019] achieve in-the-wild face tracking of a DAM model by training a regressor using self-supervised domain adaptation while ensuring temporal consistency. [McDonagh et al. 2016] train a person-specific regressor for facial performance capture to be robust to lighting changes by synthetically generating a large set of rendered images with varying synthetic lighting. In contrast, we learn from a small set of training sequences and use few-shot learning to adapt the regressor at test time.

2.4 Non-parametric Avatars

Non-parametric approaches rely on more sophisticated sensors to directly capture and transmit a 3D representation of the capture volume. Early realtime volumetric approaches, such as [Dou et al. 2017, 2016; Orts-Escolano et al. 2016], use several high-speed RGBD sensors to reconstruct and transmit full-body 3D avatar as meshes and textures. Progress in reducing the sensing requirements has been made by incorporating machine learning techniques to learn to fill holes produced by incomplete sensing or generate higher quality textures. In particular, [Martin-Brúalla et al. 2018] showed full-body reconstructions with 16 active stereo cameras, and upper body reconstructions using a single stereo pair. This method uses a set of training views to learn a neural re-renderer that takes as input partial depth reconstructions and renders higher quality images. This was extended in [Pandey et al. 2019] to allow for full body volumetric capture using a single RGBD sensor, but requires several additional viewpoints during training for viewpoint generalization.

3 OVERVIEW

In this section, we introduce the hardware we are using to capture data (§3.1), and provide an overview of our algorithm in §3.2.

3.1 Hardware and data capture

Hardware Setup. All of the data used in this work was collected using the setup depicted in Fig. 2 (a). The setup consists of a custom binocular webcam, which we refer to as Facestar, used to capture video, a desktop mount for the device, and a PC with custom capture software to record frames from the Facestar.
We use this custom binocular camera to have better control of the camera settings, such as exposure time, white balance, resolution etc. The Facestar device (Fig. 3) has 2x OmniVision OV2312 cameras with a 100 mm baseline. The selected f/2.0 lenses provide a 35 degree and 43 degree HFOV and VFOV respectively. The lenses were focused to approximately 0.5m away, which was empirically determined to be a common desktop-mount distance for our participants.

A custom camera controller is used inside the Facestar to simultaneously capture 2 MP (1300 x 1600) images from both RGB-IR cameras at 60 FPS and send them to the connected PC over USB 3.0. Custom software on the PC is used to perform debayering of the RGB pixels; the IR channel is discarded. The debayered images are encoded into a video using OpenCV. When necessary, single channel audio was recorded separately using a cell phone, using the built-in camera application.

**Data Capture Procedure.** Each participant was asked to capture themselves six times, ideally using two different lighting configurations in each of three different backgrounds. It was recommended that each participant simply rotate the Facestar twice, in 90 degree increments, to obtain 3 different backgrounds, and modify the lighting configuration by turning lamps on or off on either side of their face, though participants could choose other configurations if they were more convenient. One frame from each of the six captures for a participant is shown in Fig. 2 (a). At each frame we can get a pair of images \( \{I_v\}_{v \in [0,1]} \). Each capture lasted about 8 minutes, during which the participant would follow along with a video showing a variety of facial expressions and sentences to read aloud, similar to the procedure in [Lombardi et al. 2018].

---

**4 FACE ESTIMATION**

Our approach assumes the availability of a pre-trained person-specific parametric face model of mesh and appearance (PS-DAM). Given this model, this section describes how to produce realistic avatar animation by fitting the model to a binocular video with unmatched lighting conditions. Fig. 4 shows the overview of our proposed face estimation method.

**4.1 Illumination Invariant Face Model**

Given an input image pair, \( \{I_v\} \), our goal is to estimate the full state of the face, comprising the rigid head pose and facial expression parameters. In this work, we use the Deep Appearance Model (DAM) proposed by [Lombardi et al. 2018] as the parametric face model. DAM generates mesh and view-dependent texture as a function of...
To fully describe the appearance effects brought by illumination variations such as specularities and texture changes due to imprecise calibration frames. This is performed jointly with the registration process to infer the face parameters in those frames as described in the §4.2. Given the capacity of the illumination model, the neural network can easily overfit to these frames and exhibit cheating behavior, where the lighting model also compensates for miss-registration instead of illumination effects only. This phenomenon has been observed previously in other works that use neural networks for domain adaptation [Schwartz et al. 2020; Wei et al. 2019]. In our work, we found that simply restricting the resolution of the network outputs and using the coarse mesh as input, instead of the full expression code, largely avoids overfitting when sufficiently good initialization is provided, as described next.

4.2 Off-line Face Model fitting

Given the rigid head pose \( [r, t] \), the generated mesh \( M \) and transformed texture \( \hat{T}^o \), we can project the face mesh to the original image space using the intrinsic camera parameters \( \Pi_o \), and rasterize it to render a relighted avatar:

\[
M^o = \mathcal{P} \left( M, r, t \mid \Pi_o \right)
\]

\[
\hat{I}^o = \mathcal{R} \left( M^o, \hat{T}^o \right)
\]

where \( \mathcal{P}_{\Pi_o} \) is the projection operator based on the camera intrinsic matrix \( \Pi_o \), \( \mathcal{R} \) is the rasterrizer operator and \( M^o \) is the projected face mesh in screen space. In this way we can render the relighted avatar \( \hat{I}^o \) in the original image space, and formulate the analysis-by-synthesis image loss:

\[
\mathcal{L}_{\text{im}}(\mathbf{p}, \phi) = \sum_{v} \frac{1}{\lambda} \left( \| \hat{I}^o - \hat{I}^o \|_1 + \lambda \text{lap} \| \Delta I^o - \Delta \hat{I}^o \|_1 \right)
\]

where \( \mathbf{p} = [r, t, z] \) corresponds to the vector of face parameters, \( \Delta \) is the image Laplacian operator and \( \lambda_{\text{lap}} \) is the weight of image Laplacian loss, and we choose \( \lambda_{\text{lap}} = 0.2 \) in our experiments.

The previous cost function, Eq. 5, defines a non-linear optimization problem that is prone to local minima. To provide a good initialization for the optimization, for each input image \( I^o \), we detect 96 2D face landmarks \( \{L^o_k\} \), that correspond to face features such as mouth corner, nose tip or face contour. For each landmark, \( k \), we can find its corresponding vertex index on the face mesh denoted as \( v_k \), and calculate the \( L_2 \) distance between 2D face landmark and its corresponding mesh vertex projection:

\[
\mathcal{L}_{\text{land}}(\mathbf{p}) = \sum_{v,k} \left\| L^o_k - M^o_{v_k} \right\|^2_2
\]

where \( M^o \) is the projected face mesh in screen space calculated by Eq.4.

At each step during the optimization, given the current estimated face parameters \( \mathbf{p}_k \) and a lighting model \( G_{\phi} \), we can render the relighted avatar as \( \hat{I}^o_k \) using Eq. 4. We then calculate the dense optical flow between \( \hat{I}^o_k \) and input image \( I^o \), and map the dense optical flow to the projected mesh vertices \( M^o_k \) in screen space through bilinear interpolation, annotated by \( D^o = [d^o_k] \). We can formulate the optical flow loss as the \( L_2 \) distance between the current projection of the face mesh \( M^o \) and the flow-predicted location \( M^o + D^o \):

\[
\mathcal{L}_{\text{flow}}(\mathbf{p}) = \sum_{o} \left( \| M^o - M^o_k - D^o_k \| \right)^2
\]

With these formulated losses Eq. 5 - 7, we can optimize the face parameters of all frames and the lighting model parameters by
Algorithm 1 Off-line face model fitting

**Input:** captured images \( \{I^t\} \), detected 2D face landmarks \( \{L_k^t\} \), intrinsic camera parameters \( \Pi_0 \).

**Output:** face parameters \( p = [r, t, z] \), network parameters \( \phi \) of the lighting model \( G_\phi \).

**Process:**

1. **Step 1:** \( p \leftarrow \arg \min_p L_{\text{land}} \)

2. **Step 2:** Fix \( p \), then: \( \phi_1 \leftarrow \arg \min_{\phi_1} L_{\text{im}} \)

3. **Step 3:** Calculate the optical flow, fix \( \phi_1 \) then:
   \[
   p \leftarrow \arg \min_p L_{\text{im}} + \lambda_{\text{land}}L_{\text{land}} + \lambda_{\text{flow}}L_{\text{flow}}
   \]

4. **Step 4:** Calculate the optical flow, then:
   \[
   p, \phi_1 \leftarrow \arg \min_p,\phi_1 L_{\text{im}} + \lambda_{\text{land}}L_{\text{land}} + \lambda_{\text{flow}}L_{\text{flow}}
   \]

5. **Step 5:** Calculate the optical flow, fix \( \phi_1 \), then:
   \[
   p, \phi_h \leftarrow \arg \min_p,\phi_h L_{\text{im}} + \lambda_{\text{land}}L_{\text{land}} + \lambda_{\text{flow}}L_{\text{flow}}
   \]

 solución the following problem:

\[
\min_{\{p, \phi_h\}} \sum_t L_{\text{im}} + \lambda_{\text{land}}L_{\text{land}} + \lambda_{\text{flow}}L_{\text{flow}}, \tag{8}
\]

where \( p_t \) is the face parameter vector at frame \( t \), \( \lambda_{\text{land}} \) and \( \lambda_{\text{flow}} \) are used to control the weights of landmark loss and optical flow loss and are chosen as \( \lambda_{\text{land}} = 1.0 \) and \( \lambda_{\text{flow}} = 3.0 \) in our experiments. These optimized face parameters, are regarded as the groundtruth parameters in training the encoder for \( \S5 \), and are denoted as \( p^* = [r^*, t^*, z^*] \).

The implementation details in solving the problem Eq. 8 will be discussed in §4.3.

4.3 Implementation details

**Off-line fitting steps.** We solve the optimization problem in Eq. 8 in several steps, which are listed in Algorithm 1. At Step 1, we only minimize the landmark loss formulated by Eq. 6 to obtain the face parameter \( p \) of each frame. Then we fix \( p \) and optimize the lighting model parameters at Step 2. We use a coarse-to-fine scheme to avoid overfitting. That is, we first optimize a low-resolution version of the lighting model parameters \( \phi_1 \) at this step, by minimizing the image loss described as Eq. 5. Then at Step 3, with the optimized \( \phi_1 \) from Step 1 and \( \phi_1 \) from Step 2, we can render the relit avatar \( \hat{I}^t \) using Eq. 4, and calculate the dense optical flow between \( \hat{I}^t \) and input image \( I^t \). Then we formulate the optical flow loss as Eq. 7. We fix the lighting model, and optimize the per-frame face parameters \( p \) by minimizing the image loss, landmark loss and optical flow loss. At Step 4, we jointly optimize \( \phi_1 \) and \( p \) to further minimize the loss. Finally, at Step 5, we fix the low-resolution lighting model’s parameters \( \phi_1 \), and jointly optimize \( p \) and high-resolution lighting model parameters \( \phi_h \), to refine the results. At Step 4 and Step 5, we similarly render the relit avatar, calculate dense optical flow and formulate the optical flow loss.

**Coarse-to-fine lighting model.** Our lighting model is designed to describe the appearance effects brought by illumination differences between the DAM and the real data. However in practice, the lighting model can easily be overfitted to reconstruct the facial expressions from the captured images, which causes the “cheating” effects. For example, in Algorithm 1, as the estimated face parameters based on 2D face landmarks from Step 1 are not accurate, optimizing the lighting model in Step 2 can easily overfit the model to reconstruct the relit avatar \( \hat{I}^t \) matching input image \( I^t \), but with wrong facial expressions. To mitigate this problem, we propose a coarse-to-fine lighting model, shown in Fig. 5. At Step 2, we only optimize the low-resolution lighting model \( G_{\phi_1} \), which outputs \( 32 \times 32 \) gain and bias maps. We then up-sample them using bicubic interpolation to match the texture resolution, \( 1024 \times 1024 \). The output low-resolution gain and bias maps help to avoid overfitting, and obtain accurate facial expression codes in Step 3 and Step 4. Then, at Step 5, given the current estimate of face parameters, we fix the low-resolution lighting model but optimize another high-resolution lighting model \( G_{\phi_h} \), which outputs a \( 256 \times 256 \) gain and bias maps, to refine the details of the face. The experiments (Fig. 10) show that this coarse-to-fine lighting model helps to recover correct facial expression, while also reconstructing the captured lighting well.

5 REAL-TIME FACE ANIMATION

In the previous section, we proposed an offline method to fit a DAM model to arbitrary input video. In this section, we describe a direct regressor that allows real-time inference, based on the training data generated by the offline method. Fig. 6 shows the pipeline for real-time face animation. We first run a coarse mesh tracking algorithm on the input images (§5.1). These tracked meshes, and the unwarped textures are taken as input of an encoder to regress the target face parameters (§5.2). The regressed expression code, together with the camera view vector, are fed into the DAM to extract the face mesh and texture. Finally, the mesh, texture and rigid head pose are used to render the 3D avatar. In order to compensate for different illuminations in test videos, we use a few-shot learning strategy to adapt our pre-trained encoder (§5.3).

5.1 Coarse mesh tracking

To enable a real-time coarse mesh tracking, we build a linear PCA (Principal Components Analysis) model based on the tracked meshes from [Lombardi et al. 2018], annotated by \( \{\tilde{A}, \tilde{A}_1, \tilde{A}_2, ..., \tilde{A}_m\} \), where \( \tilde{A} \) is the average face and \( \tilde{A}_1, \tilde{A}_2, ..., \tilde{A}_m \) are the principal components basis. Denoting a PCA coefficient vector as \( a = [a_1, a_2, ..., a_m] \) we can generate coarse face mesh as the linear combination of the PCA basis: \( \tilde{M} = \tilde{A} + \sum_j a_j \tilde{A}_j \).

At each frame, for each image \( I^t \) from Facestar, we firstly detect 2D face landmarks \( \{L_k^t\} \) using a real-time generic detector [Xiong and De la Torre 2013]. Similar to the landmark loss defined in Eq. 6, we can transform the face mesh \( M \) using rigid head pose \( [r, t] \) and project it to image space using the camera intrinsic matrix \( \Pi_0 \), and penalize the \( L_2 \) distance between the 2D landmarks and their corresponding mesh vertices’ projection:

\[
M^o = P(M, r, t | \Pi_o),
\]

\[
L_{\text{track}}(r, t, a) = \sum_{k \in \mathcal{V}} \left\| L_k^o - M_k^o \right\|^2. \tag{9}
\]

We use non-linear least squares minimization following [Cao et al. 2018] to optimize \( L_{\text{track}} \), and obtain the coarse mesh parameters \( [r, t, a] \) of each frame. We implement the optimization using the Ceres solver [Agarwal et al. 2010] to achieve the real-time performance at runtime.
We then define the encoder, which takes the coarse mesh \( \psi \) and \( \delta t \) into the images based on Eq. 1-4, and define a loss directly in image space as Eq. 5. In our experiments, we firstly optimize \( \psi \) with 200 epochs by minimizing the parameter loss Eq. 11, and then further optimize \( \psi \) with 50 epochs by minimizing the image loss Eq. 5.

### 5.2 Encoder training

For each input image pair \( \{I^0, I^1\} \), with the tracked coarse mesh from §5.1 and the corresponding groundtruth face parameters \( p^* \) from §4.2, we can now train an encoder.

We firstly warp the input image \( I^0 \) into the texture space with the projected coarse mesh \( M^0 \), to get the captured textures \( T^0 \). We then define the encoder, which takes the coarse mesh \( M \) and unwarped texture \( \{T^0\} \) as input, and outputs the expression code and rigid head pose increments:

\[
\delta r, \delta t, z \leftarrow E_{\psi}(M, \{T^0\}),
\]

where \( \psi \) are the network parameters of the encoder. The rigid head pose increments are simply applied to the estimated head pose from coarse mesh tracking: \( r + \delta r, t + \delta t \). We expect these refined head pose and regressed expression codes to match the groundtruth parameters, thus we define a parameter loss by calculating the \( L_2 \) distance between the regressed results and the groundtruth:

\[
L_{\text{param}}(\psi) = \lambda_r ||r + \delta r - r^*||^2 + \lambda_t ||t + \delta t - t^*||^2 + \lambda_z ||z - z^*||^2,
\]

where \( \lambda_r, \lambda_t \) and \( \lambda_z \) are weights to balance the units of rotation, translation and expression code, and we choose \( \lambda_r = 1e^{-2}, \lambda_t = 1e^{-2} \) and \( \lambda_z = 10 \) in our experiments.

Additionally, with the regressed face parameters and the lighting model \( G_\phi \) obtained from §4.2, we can render the relighted avatar into the images based on Eq. 1-4, and define a loss directly in image space as Eq. 5. In our experiments, we firstly optimize \( \psi \) with 200 epochs by minimizing the parameter loss Eq. 11, and then further optimize \( \psi \) with 50 epochs by minimizing the image loss Eq. 5.

### 5.3 Few-shots learning adaptation

The environments and the lighting of the test scenarios may be different from the one of training data, thus directly applying the pre-trained encoder to data from new environments may lead to biased results. This section describes a few-shot learning strategy to adapt the encoder to new environments.

Fig. 7 shows the network architecture of the encoder, which consists of three branches. The first branch is a mesh branch, which takes the coarse mesh \( M \) as input. We sample 274 vertices from mesh \( M \) and construct a position vector using the 3D locations of these vertices, which is transformed using several fully-connected layers to get the latent vector. The second branch is a texture branch, which takes the unwarped textures \( \{T^0\} \) as input, down-samples it to \( 256 \times 256 \), and encodes it using seven convolutional layers, with each layer followed by a BlurPooling layer [Zhang 2019]. The output is reshaped to a 4096 dimensional vector, and passed to several fully-connected layers to get a latent vector. This vector is concatenated with the latent vector from the mesh branch, from which the expression code \( z \) is regressed. Rigid head pose increments \( \delta r \) and \( \delta t \) are similarly regressed from the texture branch. To allow for few-shot learning of new environments in the encoder, inspired by [Zakharov et al. 2019], we add an embedder branch. The embedder branch also takes unwarped textures \( \{T^0\} \) as input. We use similar convolution layers in the texture branch, followed by two fully-connected layers to get adaptive parameters \( \{\mu, \sigma\} \), which are applied to adaptive instance normalization layers in the texture branch. In general, we aim to learn the embedder branch such that the output vector \( \{\mu, \sigma\} \) contains the video specific lighting information.

At runtime, with input test video of a novel lighting environment, we first uniformly sample \( K \) frames from the first 30 seconds of the test video. We use the off-line face estimation methods described in §4.2 to estimate the face parameters for these frames. To prevent the lighting model \( G_\phi \) from overfitting on these limited frames, we select a pre-trained lighting model trained on one of this user’s training videos, and fine-tune it in Step 2 and Step 5 in Algorithm 1, instead of starting from scratch. We then define the parameter loss as Eq. 11 and image loss as Eq. 5, and adapt the pre-trained encoder. Experiments show that after 50 epochs, the adapted encoder can work well on the remaining frames of the test video.
6 RESULTS

In this section we describe the experiment setup and the timing of the system in §6.1, and presents a qualitative evaluation of the off-line face animation method in §6.2. §6.3 describes an ablation study on different components of our algorithms. Finally, we compare our methods with state-of-the-art methods in §6.4.

6.1 Experiment setup

To build a user’s PS-DAM, following [Lombardi et al. 2018], we use a large multi-camera capture apparatus which contains 40 machine vision cameras synchronously capturing images at 1334 × 2048 / 90 fps, and 460 white LED lights to promote uniform illumination. The user is asked to make a predefined set of 122 facial expressions and recite a set of 50 phonetically balanced sentences. This whole capture process takes about 1 hour to complete.

Facestar is then used to capture user’s facial performance under six different in-the-wild environmental conditions. In each capture the user is asked to follow a video demonstrating a variety of facial expressions and sentences to recite. Each capture takes about 8 minutes to complete, for a total recording time of 48 minutes.

We then build PS-DAM based on the captured data in the multi-camera apparatus and estimate the face parameters in each Facestar captured video. We used the parameters of four video sequences for training the encoder. The entire preprocessing step takes about 3 days to finish on a GPU cluster. Once the PS-DAM and encoder are trained, a short sequence (less than 30 seconds) of the user’s face performance is collected in a new target environment, which is used to fine-tune the encoder. In under 1 hour processing on 4 GPUs (Table 3) the user can drive her/his avatar in real-time in the new target environment.

6.2 Qualitative Evaluation

We used Facestar to capture seven users’ facial performance under six different environmental conditions (42 videos in total). For each video sequence, we estimate the face parameters using the method described in §4.2, that provides our ground-truth. We then used the parameters of four video sequences of an user as training data to train the encoder, and the remaining two videos for testing.

Off-line face model fitting. Fig. 1 and Fig. 17 shows the offline face model fitting on different users under different environments. Our method can estimate accurate face parameters, including the rigid head pose and non-rigid facial expression code, which can animate the avatar (b) with the same facial expression as the input image (a), demonstrating the quality of the facial expression transfer, including eyelids, gaze, lip shape, wrinkles, teeth and tongue. Our proposed lighting model effectively generates gain and bias maps (e) which models the illumination changes due to different lighting conditions, head pose and facial expressions, which help to relight the avatar (b) to pixel-wise match the input image. Though the PS-DAM does not model glasses, our methods can still handle the case with glasses on (Fig. 17 3rd row), by encoding the glasses into the gain and bias map.

Our method can also be applied to other consumer RGBD cameras (i.e. Intel RealSense, iPhoneXS etc.) to achieve a fitted face model with similar quality as with using Facestar. In Fig. 8 we show the results of estimating the face parameters from RGBD images from an iPhoneXS. Besides the losses defined in Eq. 8, we also compute the $L_1$ error between the rendered depth map (Fig. 8 (g)) and captured depth map (Fig. 8 (f)), which helps to achieve stable rigid alignment in place of the multi-view images in Facestar.

Real-time facial animation. For each user, we took her/his four videos sequences to train the encoder (Fig. 7). The other two video sequences are used as test data. We uniformly sampled $K$ frames from the first 30 seconds of the test videos, and adapt the encoder following a few-shot learning strategy described in §5.3. We then applied this adapted encoder to other frames of the test videos. The output of this process is shown in Fig. 1 and Fig. 18. With the input image from the binocular camera, each person can drive her/his own avatar to generate the photo-realistic avatar animation, covering wide range of facial motions. With the view-conditioned DAM representation [Lombardi et al. 2018], we can also visualize the 3D avatars from different views, which allows our methods to be
used in AR/VR applications. We implemented the facial animation methods on a Windows machine with 64 cores CPU and 2 Nvidia GTX 2080 GPUs, which allows us to drive the avatar at 45 fps. Please see our supplementary video for a live demo.

6.3 Ablation Study
This section performs an ablation study of different components of the algorithms.

**Robust loss function without a lighting model.** We proposed a novel lighting model to fill the domain mismatch between DAM and in-the-wild video. Can we ignore this domain mismatch and define a robust loss function to fit the face model parameters directly? In this ablation study, we defined a domain-independent perceptual loss in addition to other losses defined in Eq. 5 - Eq. 7. We passed both the rendered avatar and input image into the VGG19 network ([Simonyan and Zisserman 2014]), got the output features, and calculated the $L_1$ loss of the corresponding features. Fig. 9 shows the results of face model fitting by optimizing these losses. While the perceptual loss help to recover the semantic expression to some extent, the facial motions of the fitted avatar (Fig. 9 middle row) have a clear gap to the input image (Fig. 9 top row). As a comparison, our method used the pixel-wise image loss, which can help to recover precise facial expressions (Fig. 9 bottom row).

**Coarse-to-fine lighting models.** The key component of the off-line face model fitting is the lighting model. A good lighting model is critical to recover the illumination differences between DAM and in-the-wild image, and have an accurate estimate of the face parameters. We choose a coarse-to-fine network architecture as the lighting model (Fig. 5), and compared with other architectures.

To get the ground-truth face parameters, we recorded facial performance data in a calibrated multi-view light-stage consisting of 40 vision cameras synchronously capturing images at $1334 \times 2048$ / 90 fps and a total of 460 white LED lights. We flash a group of LEDs (at most 10) at each frame, and turn on all LEDs every 3rd frame to get fully lit images. These full-lit images have the same lighting conditions as the DAM, so we can directly define the image loss without applying any lighting adaptation and get ground-truth face parameters on these full-lit images. These face parameters are then interpolated to in-between frames, to get the ground-truth face parameters for frames with non-uniform lighting conditions.

For testing, we choose frames with one particular lighting condition, and run our off-line face model fitting pipeline to fit the face parameters and reconstruct the input image. We will compare
the $L_2$ image loss between the relighted avatar and input image (Reconstruction Error), and the $L_2$ loss between the avatar rendered using estimated parameters, and the ground-truth rendered avatar (Avatar Error). This avatar loss reflects the accuracy of the fitted face parameters.

The simplest lighting model is a single gain and bias value. The lighting model takes rigid head pose, face mesh and camera viewpoint vector as input, outputs single gain and bias value, which are then expanded to the gain and bias maps (Single value). We also try to optimize a single, frame-independent gain and bias maps at $1024 \times 1024$ resolution (Single map). On the other hand, we tested lighting models similar to our low-resolution lighting model, but with different output resolution: $8 \times 8$, $128 \times 128$, $512 \times 512$. The output of these lighting models are up-sampled to $1024 \times 1024$ resolution.

Table 1 shows the errors using different lighting models in our pipeline, and Fig. 10 shows one example frame using different lighting models. The input face image has non-uniform lighting. Single gain/bias values are insufficient to model this lighting change, which results in both wrong reconstructed avatar (large avatar error) and poor reconstruction (large reconstruction error). The single map lighting model can lower the reconstruction loss as it has different gain/bias values at different pixels. However, as the lighting will change due to different head pose and facial expressions, single map cannot model the lighting variability across different frames. For lighting models with different output resolutions, the one with larger resolution ($128 \times 128$ or $512 \times 512$) has too much capacity, which will get good reconstruction (smaller reconstruction errors), but it fails to estimate accurate face parameters, and results in wrong reconstructed avatar (large avatar error). We call this effect as "Cheating" effect. As a comparison, in our strategy, we first used a low-resolution lighting model with $32 \times 32$ output, to estimate the face parameters in Step 2 - 4 of Algorithm 1. Then in Step 5, we add another high-resolution lighting model with $256 \times 256$ output, to refine the face details. Our coarse-to-fine lighting model gives the most accurate face parameters (lowest avatar error) and the best reconstruction (lowest reconstruction error).

Two views vs. single view. We used Facestar, the binocular cameras to simultaneously capture an image pair from two different views at each frame, which are taken as input in all our experiments in this paper. To evaluate the importance of these two views input,
Fig. 11. Ablation study on few-shots adaptation. Directly apply the pre-trained encoder to the test video will fail in tracking accurate facial motions. With more frames from test video in adapting the encoder will help to get more accurate avatars, but brings more computation time. In our experiments, we find $K = 256$ can already give satisfying results.

Table 2. Ablation study on using different lighting conditions in training data, and embedder branch.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Avatar Error</th>
<th>Decrease</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 lightings w/o embedder</td>
<td>5.24</td>
<td>NA</td>
</tr>
<tr>
<td>2 lightings</td>
<td>4.19</td>
<td>20.0%</td>
</tr>
<tr>
<td>4 lightings w/o embedder</td>
<td>3.16</td>
<td>NA</td>
</tr>
<tr>
<td>4 lightings</td>
<td>2.74</td>
<td>13.3%</td>
</tr>
</tbody>
</table>

Table 3. Using different numbers of frames to adapt the pre-trained encoder.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Avatar Error</th>
<th>Adaptation Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-trained encoder</td>
<td>4.19</td>
<td>NA</td>
</tr>
<tr>
<td>$K = 64$</td>
<td>3.11</td>
<td>4 mins</td>
</tr>
<tr>
<td>$K = 128$</td>
<td>2.92</td>
<td>8 mins</td>
</tr>
<tr>
<td>$K = 256$</td>
<td>2.78</td>
<td>15 mins</td>
</tr>
<tr>
<td>$K = 512$</td>
<td>2.69</td>
<td>31 mins</td>
</tr>
<tr>
<td>$K = 1024$</td>
<td>2.68</td>
<td>59 mins</td>
</tr>
</tbody>
</table>

**Different lightings in training data.** We used training data with different lighting conditions to train the encoders, and apply the encoder to test video with another lighting condition, and compute the $L_2$ avatar error. Tab. 2 shows the comparisons between two lighting conditions and four lighting conditions. As expected, the more variability that we have in the training data, in general, the better the model will be able to generalized.

**Embedder branch in encoder.** To evaluate the embedded branch in our encoder architecture, we removed the last convolution layer in the texture branch from the original encoder, and train it using the same training data. We then applied this encoder to another test video sequence and compared the $L_2$ avatar errors. Tab. 2 shows the comparisons with/without the embedded branch in the encoder. For the encoder trained using data with 2 lighting conditions, adding the embedded will help to decrease the avatar error by 20.0%, and 13.3% for the case with 4 lighting conditions. Fig. 11 (b)(c) show one example frame of animated avatar from the input image. Our encoder with the embedded branch (c) help to recover better mouth shape than the one without the embedded branch (b).
Fig. 13. Comparisons with Lookingood [Martin-Brualla et al. 2018] on data captured in light-stage. With input image pair (b) from Facestar, we estimate the depth using vision stereo algorithm, and construct the RGBD data (d). Lookingood takes this RGBD as input, infer the dense, complete volumetric avatar representation (e). Avatar generated by our method (g) contains more face details, and has lower reconstruction error (h) than Lookingood (f). Please notice that our methods only replies on the input images (a-b), not the RGBD input (d).

Fig. 14. Comparisons with Lookingood [Martin-Brualla et al. 2018] on in-the-wild video. The re-rendering network of Lookingood is trained on light-stage data, which can not be generalized well to in-the-wild video (c). Our method gives satisfying results (d) thanks to the proposed lighting model.

Selection of $K$ in adapting the encoder. At runtime, the more frames we used from the test data to fine-tune the pre-trained encoder the better we are able to adapt the encoder to test video; however, the more frames we select the more expensive is the computation. To study the selection of $K$, we first used two videos of one user to train an encoder. Then we uniformly sample $K$ frames from the first 30 seconds of the test video, estimated the face parameters using method from §4.2, and adapted the encoder. Finally, we applied the adapted encoder to other frames of the test video, and compared the $L_2$ avatar error to the ground-truth avatars. Tab. 3 shows the avatar errors and adaptation times with different $K$. When $K$ is increased from 512 to 1024, the avatar errors are very close already. Fig. 11 shows one example frame which is not used to adapt the encoder. Though the error is decreased a little bit, but it is hard to tell the differences from the visual results after $K = 256$. Considering the performance, we choose $K = 256$ in all our experiments.

6.4 Comparisons

This section compares our methods with other state-of-the-art facial animation methods.

Non-parametric avatars. First, we compared our method to a recently non-parametric avatar method “Lookingood” [Martin-Brualla et al. 2018]. Their system takes real-time, low-quality RGBD data of user facial performance as input, which may suffer from artifacts such as holes and noise in the rendering, and use a deep architecture to perform completion, super resolution and denoising in real-time. To collect the training data to train this system, similar to what we do in ablation study, we put the Facestar in a light-stage containing multiple cameras, to capture user’s facial performance from Facestar and light-stage cameras. The images captured by light-stage cameras are used to build a volumetric avatar of the user. With an input image pair from the Facestar, at each frame, we find the pixel correspondence between the paired images, and estimate the depth using vision stereo algorithm. Based on these data, we can train a machine learning framework called neural re-rendering, to infer the dense, complete volumetric representation (Fig. 13(e)) from noisy, low-quality RGBD data (Fig. 13(d)). As a comparison, we used the images captured by light-stage cameras to build the DAM [Lombardi et al. 2018], and used our method to drive the avatar animation from Facestar images (Fig. 13(g)).
While [Cao et al. 2018] can recover right expression in specific ex-
pressions with accurate rigid head pose and facial expressions (d). Directly
applying the Looking-Good model, trained on light-stage data, on
these in-the-wild image and the estimated RGBD data (b) cannot
give lower reconstruction error than Lookingood.

Our proposed lighting model helps to fit the face model to data
captured under different environments, so that we can not only
reconstruct avatars for video captured inside the light-stage, but
in-the-wild video. Fig. 14 shows examples of our method being
applied to the in-the-wild input images (a), and reconstruct avatar
with accurate rigid head pose and facial expressions (d). Directly
applying the Looking-Good model, trained on light-stage data, on
these in-the-wild image and the estimated RGBD data (b) cannot
get satisfied results (c).

Real-time landmark-based facial animation. In recent years, marker-
less, real-time face tracking methods based on 2D images are widely
explored and used in many gaming and AR applications. A major
approach is to detect the 2D face landmarks, then use these 2D land-
marks to track the 3D face. To enhance the temporal coherence and
rigid stability, [Cao et al. 2018] also used optical flow as constraints
during optimization. We extend their method to binocular cameras
input, using 2D facial landmarks and optical flow as constraints to
optimize the rigid head pose and expression code of DAM. Fig. 15 (b)
shows several example frames. These sparse landmarks and optical
flow are not able to describe complete and subtle facial expressions.
While [Cao et al. 2018] can recover right expression in specific ex-
pressions with distinctive face geometry (Frame #3), their method
cannot reconstruct correct expression for other frames, especially
the details on eyes (Frame #1), teeth (Frame #2) and lip shape (Frame
#3,#4) and gazes (Frame #6).

Fig. 13 shows two example frames from the test video comparing
Lookingood [Martin-Brulla et al. 2018] with our method. From
the visual avatar results (e-g), both methods can reconstruct an
accurate 3D avatar from input images (a-b). However, our results
contain more facial details such as the face pores, while Lookingood
gives over smoothed results. We take the image captured from one
light-stage camera as the reference image. project the avatar into
the image, and calculate the \(L_2\) reconstruction error inside the face
region, shown as (f-h). We can see our method produces lower
reconstruction error. We calculated the reconstruction error on
the whole video sequence with \( \sim 7K \) frames, and get the average
reconstruction error shown as Table 4, which shows that our method
can give lower reconstruction error than Lookingood.

Table 5. Comparisons with [Yoon et al. 2019] and [Li et al. 2018]

<table>
<thead>
<tr>
<th>Methods</th>
<th>Reconstruction Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Yoon et al. 2019]</td>
<td>6.81</td>
</tr>
<tr>
<td>[Li et al. 2018]</td>
<td>7.44</td>
</tr>
<tr>
<td>Ours</td>
<td>2.69</td>
</tr>
</tbody>
</table>

Other lighting adaptation for face tracking. We compared our
method to other methods that use different lighting adaptation
schemes for face tracking. [Yoon et al. 2019] proposed I2ZNet, which
tracks the DAM by training a regressor using self-supervised do-
main adaptation. I2ZNet learns a single 1-by-1 convolution layer
at the output texture of DAM as the color transformation, which
helps to correct the white-balance between the two domains. We
first trained the I2ZNet on the light-stage data which are used to
construct DAM, and adapted the model on in-the-wild data in a
self-supervised way. Fig. 16 (b) shows the results. Their proposed
lighting adaptation model (i.e. the 1-by-1 convolution layer) fails
when using challenging lighting conditions, which results in poor
quality avatars with artifacts.

On the other hand, we also compared our method with the com-
mon lighting model used in previous work, spherical harmonics
(SH). We replaced the lighting model with SH in our system. We
followed the environment map prediction methods proposed by [Li
et al. 2018], to train a network to predict the first nine SH coefficients
for each color channel from the input texture. These SH coefficients,
together with the normal map of face mesh extracted from DAM,
are used to relight the face texture. We then render the relighted
avatar using these relighted texture, and calculate the image loss
between the relighted avatar with input image. This image loss can
be minimized to obtain the optimal rigid head pose and expression
code. Fig. 16 (c) shows the results. The SH lighting model can only
recover the low-frequency lighting differences, thus the optimized
avatar based on this lighting model contain mismatch facial details,
such as blur mouth lip and wrong eye lids. Our methods can re-
construct an avatar with precise facial expressions, and get lower
reconstruction error (Fig. 16(d)) thanks to our coarse-to-fine lighting
model. Table 5 also shows the reconstruction error using different
lighting adaptation methods, while our method can give the lowest
reconstruction error.
We propose a real-time method to drive hyper-realistic avatars from work. Finally, our method is still a person specific solution. To make our system more practical to users, we will explore a general method based on a few-shot learning regressor, which can be quickly adapted to test scenarios by observing a few frames of new environments.

There exists some limitations of our method. First, we assume the lighting conditions during user’s capture will not change dramatically, which is not true in some cases. In the future, we will explore an online adaptation strategy to adapt the encoder. Second, we now only model the face animation, but not the hair, neck or shoulder. We may extend our method to upper body animation in the future work. Finally, our method is still a person specific solution. To make our system more practical to users, we will explore a general method to drive the high-fidelity avatar animation.

REFERENCE


ACM Trans. Graph., Vol. 40, No. 4, Article 87. Publication date: August 2021.
Fig. 17. Examples of off-line face model fitting on different users under different lighting conditions. With input image (a), our algorithm can decouple the accurate rigid head pose and non-rigid facial expression, and reconstruct the avatar with the same facial motions (d). The proposed lighting model can produce the gain and bias map (e), which reflects the lighting condition, head pose and facial expressions. These gain and bias map help to relight the avatar (b) which is pixel-wise matching the input image. We visualize the $L_2$ error between (a) and (b) as the reconstruction error in (c). Our method can handle different head poses or different lighting conditions, and well reconstruct both exaggerated and subtle facial expressions.
Fig. 18. Examples of real-time facial animations for different persons. With input image (a) our method can track 3D face and drive the avatar animation (b). Thanks to the view-dependent DAM representation, we can visualize the 3D avatar from different views (c). Our real-time facial animation method can handle different lighting conditions, recover expression to a wide range of motions. Our methods can well handle user with glasses on (bottom row).


