2017 IEEE 12th International Conference on Automatic Face & Gesture Recognition

Rapid Synthesis of Massive Face Sets for Improved Face Recognition

Iacopo Masi[1], Tal Hassner[2,3], Anh Tuấn Trần[1] and Gérard Medioni[1]
[1] Institute for Robotics and Intelligent Systems, USC, CA, USA
[2] Information Sciences Institute, USC, CA, USA
[3] The Open University of Israel, Israel

***Abstract*— Recent work demonstrated that computer graphics techniques can be used to improve face recognition performances by synthesizing multiple new views of faces available in existing face collections. By so doing, more images and more appearance variations are available for training, thereby improving the deep models trained on these images. Similar rendering techniques were also applied at test time to align faces in 3D and reduce appearance variations when comparing faces. These previous results, however, did not consider the computational cost of rendering: At training, rendering millions of face images can be prohibitive; at test time, rendering can quickly become a bottleneck, particularly when multiple images represent a subject. This paper builds on a number of observations which, under certain circumstances, allow rendering new 3D views of faces at a computational cost which is equivalent to simple 2D image warping. We demonstrate this by showing that the run-time of an optimized OpenGL rendering engine is slower than the simple Python implementation we designed for the same purpose. The proposed rendering is used in a face recognition pipeline and tested on the challenging IJB-A and Janus CS2 benchmarks. Our results show that our rendering is not only fast, but improves recognition accuracy.**

I. INTRODUCTION

Over the last few years face recognition capabilities made extraordinary progress. Much of this improvement can be attributed to the success of *deep learning* based methods which learn discriminative face representations from massive training sets. These methods make an underlying assumption that by collecting large enough training sets, deep networks will have sufficient examples of both inter-class and intra-class appearance variations. From these variations, networks can learn to produce features which amplify subject identity and suppress other, confounding appearance variations.

Unfortunately, this underlying assumption often does not hold in practice. Even huge data sets, such as the CASIA WebFace collection [2], demonstrate strong biases towards frontal facing faces and, on average, offer only few example images for each subject [3]. Consequently, the intra-class appearance variations offered by these sets are limited. To address this problem, a number of recent methods use computer graphics techniques to enrich existing face sets by synthesizing new views of the face images they contain. Thus, an existing face set is inflated to many times its size by introducing additional intra-subject appearance variations. Pose variations in particular (e.g., Fig. 1, bottom) were shown to provide a substantial boost to the quality of the networks that were trained on these augmented sets.

Similar rendering techniques were also applied when matching face images: To reduce appearance variations due

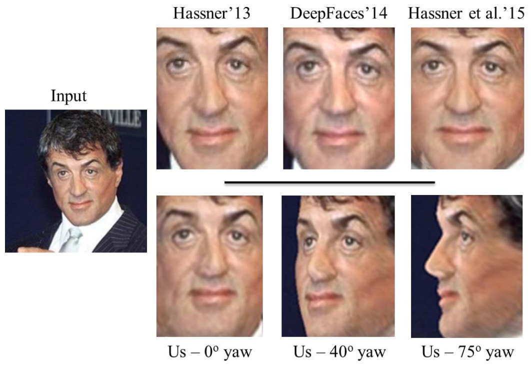

Fig. 1: *Rendered (synthesized) new views of a real face.* Input face image (left) is rendered to frontal views by previous work (top) and to multiple views by our fast method (bottom). Results in top row taken from [1].

to different poses, faces were rendered to common 3D views. The rendered faces then shared the same 3D pose and were therefore easier to compare in order to determine identity.

A remaining concern with these rendering techniques, however, is that they can be computationally expensive. When applied to augment huge face collections, processing can requires days. At test time, where current deep systems utilize graphical processing units (GPU) to expedite feature extraction, rendering new views to align the faces, even using optimized code, can quickly become a bottleneck.

This paper addresses these concerns by describing a simple and highly efficient method for rendering faces in multiple views (e.g., Fig. 1, bottom). We follow the recent work of [1], [3], [4], [5] and [6] by using generic 3D face models in order to produce new views. They showed that by doing so, they obtain the same quality of rendered views yet more stable results than others who estimated 3D facial structures prior to alignment and rendering (e.g., [7], [8], [9]). We explain why an important benefit of this approach, previously overlooked, is its computational simplicity.

Specifically, this paper offers the following contributions. **(1)** We explain how new views of faces can be synthesized for the same computational price as standard image warping, by using generic 3D face models and fixed target poses. **(2)** We compare the runtime of this approach with existing face rendering techniques. We further evaluate the influence of our approach on face recognition performance using the challenging IJB-A and Janus CS2 benchmarks showing it obtains state of the art results. **(3)** Previous related work

978-1-5090-4023-0/17 $31.00 © 2017 IEEE
DOI 10.1109/FG.2017.76

rendered faces without their background. Our method renders faces with backgrounds allowing us to introduce pooling of features across 3D face poses.

II. RELATED WORK

Many previous attempts were made to synthesize new views of faces appearing in single images. In the past, most of these efforts aimed at visualization and graphics applications. To our knowledge, these methods were only recently successfully applied for face recognition.

Early methods for synthesis of new facial views attempted to first estimate the 3D shape of the face. These varied in the importance they attributed to the accuracy of the reconstruction, when compared to ground truth facial shape. Statistical methods, such as the seminal Morphable-Models based methods [10], [11], [12] used example 3D face shapes to span the space of facial 3D geometries, expressions and textures. These methods, however, assumed controlled viewing conditions, restricting their use in real-world settings.

By assuming Lambertian reflectance and constrained lighting conditions, shape from shading based methods such as [13] demonstrated remarkably detailed 3D reconstructions which could then be used to generate new views. These methods also assume that the faces are cleanly segmented from their background. All these assumptions make them likewise unsuitable for our purposes.

Recent examples of new view synthesis for face recognition include the work of [14] and its application to face alignment and recognition in [7]. Both of these methods relax the requirement for an accurate 3D reconstruction, and instead describe a method for coarse 3D face shape estimation from unconstrained images. These reconstructions were used by [7] to produce presumably aligned forward facing faces for recognition.

Most relevant to our work are the recent methods of [1], [3], [4] and [6]. Hassner *et al.* [1], in particular, showed that attempting to estimate 3D facial shapes prior to rendering new 3D views for face recognition can actually be inferior to simply using a generic 3D face shape. Though counter-intuitive, this claim was supported by qualitative examples showing that faces aligned using a single generic face are qualitatively similar to those produced by estimating 3D shape. This was quantitatively verified by nearly matching the recognition accuracy of [7], *with a much simpler, non-deep pipeline and without using millions of training images*. Their work was later extended to multiple 3D views by [4] and multiple generic shapes in [3] in both cases, showing improved performances using deep learning.

Our work follows theirs, but focusing here on the mechanics of the new view generation method. Specifically, we show that beyond the advantages they report for the use of generic shapes, an important advantage not previously noted by others, is its potentially far cheaper computational cost and much simpler rendering process. These benefits are particularly important considering the massive numbers of images that are processed in contemporary face data sets and the complexity of modern face recognition systems.

III. STANDARD FACE RENDERING

Existing methods used to synthesize new views of faces from single images involve two standard and well understood steps, known in computer graphics as *texture mapping* and *ray casting / rasterization*. We next provide a cursory overview of these steps. More details are available in any standard computer graphics textbook [15].

A. Overview of face rendering

Texture mapping of an image $\mathbf{I}$ (i.e., the input image of a face viewed in unconstrained settings) onto a 3D surface $\mathcal{F} \subset \mathbb{R}^3$ is the process of assigning every 3D surface position $\mathbf{P} = (X, Y, Z) \in \mathcal{F}$ with a location $\mathbf{q} = (u, v)$ in the image (where image coordinates are often normalized to the range of $u, v \in [0, 1]$). The shape $\mathcal{F}$ is assumed to be a 3D face shape, which may have been estimated from the input image of the face, $\mathbf{I}$, or is a predetermined, generic face.

Following texture mapping, $\mathcal{F}$ is projected to the desired view, $\mathbf{J}$. To this end, the output view's camera matrix, $\mathbf{M}^J = \mathbf{K}^J \left[\mathbf{R}^J \ \mathbf{t}^J\right]$, is manually specified in order to set the desired output viewpoint (e.g., frontal view for face *frontalization* [1]). This includes setting the intrinsic camera parameters in $\mathbf{K}^J$, and the 3D rotation and translation in $\mathbf{R}^J \ \mathbf{t}^J$, respectively, both in the coordinate frame of the 3D shape. The matrix $\mathbf{M}^J$ is then used to intersect the rays emanating from $\mathbf{J}$'s center of projection, passing through each of its pixels, $\mathbf{p}_i = (x_i, y_i) \in \mathbf{J}$, and the surface of $\mathcal{F}$. Each such intersection is a 3D point $\mathbf{P}_i = (X_i, Y_i, Z_i) \in \mathcal{F}$. Following texture mapping, these 3D points are linked to locations $\mathbf{q}_i = (u_i, v_i)$ in the input face image, $\mathbf{I}$. Thus, the pixel $\mathbf{p}_i$ in the output image is assigned intensity values by sampling $\mathbf{I}$ at its determined $\mathbf{q}_i$.

B. Texture mapping by face pose estimation

Texture mapping of the input face image $\mathbf{I}$ to the face shape $\mathcal{F}$ is performed automatically. To this end, a facial landmark detection method is used to locate k landmarks in the face image $\mathbf{I}$. This process is agnostic to the particular landmark detection method; In [1] the supervised descent method [16] was used whereas our implementation uses the method of [17]. Regardless, for each of the detected landmarks $\mathbf{q}_i \in \mathbb{R}^2$ we assume corresponding landmarks, $\mathbf{P}_i \in \mathbb{R}^3$ specified once on the 3D surface $\mathcal{F}$ (index i indicates the same facial landmark in $\mathbf{I}$ and the 3D shape $\mathcal{F}$.)

Given the corresponding landmarks $\mathbf{q}_i \leftrightarrow \mathbf{P}_i$ we use PnP [18] to estimate extrinsic camera parameters for the input image $\mathbf{I}$. Unlike [3], we assume a fixed intrinsic camera matrix $\mathbf{K}^I$, estimating only the rotation and translation matrices $\mathbf{R}^I \ \mathbf{t}^I$, in the 3D model's coordinate frame. We thus obtain a perspective camera model mapping the 3D face shape $\mathcal{F}$ to the input image so that $\mathbf{q}_i \sim \mathbf{M}^I \ \mathbf{P}_i$ where $\mathbf{M}^I = \mathbf{K}^I \ \left[\mathbf{R}^I \ \mathbf{t}^I\right]$ is the estimated camera matrix for the input view. Hence, matrix $\mathbf{M}^I$ can be used to map any point on the 3D surface onto the input image, thereby providing the desired texture map.

IV. Rapid Rendering with Generic Faces

In light of the importance of rendering as a key step in many computer graphics applications, tremendous efforts were dedicated to expediting this process, speeding up state of the art rendering engines to fractions of a second, even for complex 3D scenes. Specialized computer hardware – namely graphical processing units (GPU) – was also developed for this purpose. Our work is tangential to these efforts. Specifically, we show that by assuming a generic 3D shape and fixed desired output poses, a great deal of the effort required to render new facial views can be performed at preprocessing, and by so doing, substantially reduce the effort required to produce new views and the complexity of the system required for rendering.

A. Precomputing output projections

One of the most time consuming steps in the process described in Sec. III is ray casting: computing the locations of intersections between the rays passing through each output pixel and the surface of the face. Due to potential self occlusions, this process may further involve methods such as Z-buffering or binary space partitioning in order to determine visibility of the 3D shape at each output pixel [15].

As previously mentioned, these steps can be expedited using specialized hardware, optimized code and various approximation methods. We note that when a fixed generic face shape and output view are used, *these steps only need to be performed once*, at preprocessing. Subsequent face synthesis using the same shape and pose, can skip this step, and require the same computational effort as standard image warping in 2D for texture mapping (Sec. III-B).

Specifically, during preprocessing we use a standard rendering engine to perform ray casting of a generic face shape $\mathcal{F}$ onto a desired output view $\mathbf{J}$. Doing so, we store for each output 2D pixel location $\mathbf{p}_i \in \mathbf{J}$ the 3D coordinates $P_i \in \mathcal{F}$ projected onto that pixel (i.e., the 3D location of the surface point $\mathbf{P}_i$ visible at $\mathbf{p}_i$). This information is stored in a lookup table $\mathbf{U}$, simply define as:

$$\mathbf{U}(\mathbf{p}_i) = \mathbf{P}_i \tag{1}$$

In practice, $\mathbf{U}$ is stored as an $N \times M \times 3$ matrix, where N and M are the dimensions of the output view and the last dimension indexes the X, Y and Z coordinates of the 3D point projected onto each pixel.

B. Rendering with precomputed projections

Given an input image $\mathbf{I}$ containing a face in unconstrained settings, we render it to a desired new view using $\mathbf{U}$ as follows (see Fig. 2 for a Python code example)[1].

[1]See project at: www.openu.ac.il/home/hassner/projects/augmented_faces

```
import numpy as np
import cv2
U_bar = np.vstack((U, \
        np.ones((1, threedee.shape[1]))))
q_bar = MI * U_hat
q = np.divide(q_bar[0:2, :], \
        np.tile(q_bar[2, :], (2,1)))
#idx are indices of q inside of the image
q=q[idx]
synth = warpImg(I, N, M, q, idx)
def warpImg(I, N, M, q, idx):
  J = np.zeros((N*M, 3))
  #fast warping
  pixels = cv2.remap(I, \
  np.asarray( q[0,:] ).astype('float32'),\
  np.asarray( q[1,:] ).astype('float32'),\
  cv2.INTER_CUBIC)
  #copy the interpolated pixel back ( Eq. (1) )
  J[idx,:] = pixels
  J = J.reshape(( N, M, 3), order='F')
  J = J.astype('uint8')
  return J
```

Fig. 2: Python code snippet for 3D rendering.

We first estimate the 3D pose of the face, as described in Sec. III-B. This provides a camera matrix $\mathbf{M}^I$ associating 3D points on the surface of $\mathcal{F}$ with pixels in $\mathbf{I}$. Let

$$\bar{\mathbf{q}} = \mathbf{M}^I \bar{\mathbf{U}}, \tag{2}$$

where $\bar{\mathbf{U}}$ is matrix $\mathbf{U}$ reshaped to a $4 \times (NM)$ matrix where the columns are the 3D points stored in $\mathbf{U}$, in homogeneous notation. Matrix $\bar{\mathbf{q}}$ is then a $3 \times (NM)$ matrix with columns representing the 2D projections of these 3D points onto $\mathbf{I}$, also in homogeneous coordinates.

An output view $\mathbf{J}$ can then be produced simply by sampling image $\mathbf{I}$ using $\bar{\mathbf{q}}$ (following conversion to Euclidean coordinates). Sampled intensities are mapped back to the output view $\mathbf{J}$ by using the correspondence between columns in $\bar{\mathbf{q}}$, columns in $\bar{\mathbf{U}}$, and (x, y) pixel locations in $\mathbf{U}$.

Run-time: This process includes precisely the same steps as standard inverse warping [19]. Compared with, for example, the 2D warping regularly performed in real time on even cellphone devices, the only difference is in applying a 3×4 camera matrix transformation to homogeneous 3D coordinates, rather than a 3×3 projective transformation.

C. Preparing generic 3D heads and backgrounds

In [3], ten generic 3D shapes $\mathcal{S} = \{\mathcal{F}_1, ..., \mathcal{F}_{10}\}$, from the Basel Face Set [20], were used to model head shape variations. These 3D faces are aligned with each other and so 3D landmarks required for pose estimation (Sec. III-B) need only be selected once; selected landmarks are then automatically transferred to all other models. Beyond simplifying facial landmark selection for pose estimation,

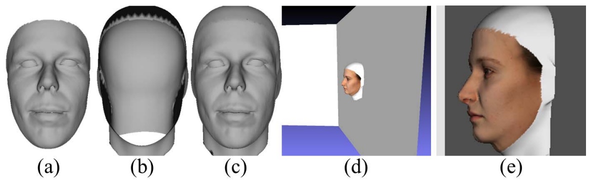

Fig. 3: Preparing generic 3D models: Head added to a generic 3D face along with two planes for background.

this also allows segmenting faces from their background and selecting eye regions to avoid cross-eyed results [1].

These models, however, only represent the facial region of the head. This is presumably why [3] rendered only partial head views without backgrounds and why [7] and [1] only used tight bounding boxes around the center of the face.

Fortunately, rendering full heads and backgrounds can naturally be included in the process described in the previous sections. This is performed by stitching the ten 3D models to an additional, generic 3D structure containing head, ears and neck and adding a plane representing a flat background.

The process of combining 3D faces to 3D heads is described in Fig. 3. We use the generic 3D head from [21]. We removed its facial region and exchange it with the 3D faces from [20]. To allow blending with different 3D faces (e.g., Fig. 3(a)), varying in sizes and shapes, we maintain an overlap belt with radius r = 2cm (Fig. 3(b)). Given an input 3D face (Fig. 3(a)), we merge it onto the head model using soft boundary blending: We first detect points on the overlap region of the face model. Each point $\mathbf{X}$ is then assigned a soft blending weight w:

$$w = \frac{1}{2} - \frac{1}{2}\cos\left(\frac{\pi d}{r}\right), \tag{3}$$

where d is the distance to the boundary. Next, $\mathbf{X}$ is adjusted to the new 3D position $\mathbf{X}'$ by:

$$\mathbf{X}' = w\mathbf{X} + (1-w)\mathbf{P}_X, \tag{4}$$

where $\mathbf{P}_X$ is the closest 3D point from the head. The result is a complete 3D face model (Fig. 3(c)) . Ostensibly, an alternative to this method would be to scan new models, with complete 3D heads. The method described above was chosen in order to minimize the differences between our recognition system and the one used by [3], including the use of the same 3D face shapes from in their system.

To additionally preserve the background, we simply add two planes to the 3D model: one positioned just behind the head and another, perpendicular plane, on its right. This second plane is used to represent the background when the input face is rendered to a profile view, in which case the first plane is mapped to a line. Fig. 3(d) shows the models we produced using one of the 3D face shapes from [20]. Fig. 3(e) shows the rendered view of this generic face from the profile pose used by our system.

V. FACE RECOGNITION

A. Training with rendered face images

We use the proposed face rendering technique in a face recognition pipeline. We employ a system similar to the one recently proposed by [3]. Their system uses a single Convolutional Neural Network (CNN) trained on both real face images and their rendered views.

We use the VGGNet, off-the-shelf deep models of [22], originally trained on the ImageNet, large scale image recognition benchmark (ILSVRC) [23]. We fine-tuned this CNN on our training data, which included the CASIA WebFaces [2], augmented by aligning the faces using similarity transform (i.e., *in-plane alignment*) and rendering them to the three yaw angles with a 3D generic face selected randomly from the ten prepared in Sec. IV-C.

During training, the network learns an embedding function that maps an input image $\mathbf{I}$ (used here to represent both real and rendered views) to a subject label: $f : \mathbf{I} \mapsto c$, where $c \in \mathcal{C}$ and $\mathcal{C}$ represents the set of identities in CASIA. Standard Soft-Max loss was used for the optimization, using the ground-truth CASIA identity labels. All other training parameters were similar to those reported by [3].

B. Face matching pipeline

Face representation: At test time we use only part of the learned function $f : \mathbf{I} \mapsto c$, without relying on the closed set of CASIA subjects, $\mathcal{C}$. The trained CNN is instead used to extract features for open-set recognition.

It is well known [3], [4] that the penultimate layer of a CNN retains high level information, useful at test time as a discriminative feature representation. For this reason, we drop the classification layer from the network and use its *fc7* response after ReLu activation as our face representation, $\mathbf{x}$.

This face representation $\mathbf{x}$ is further specialized to the target benchmark by applying cheap, unsupervised Principal Component Analysis (PCA) learned from the training splits of the test benchmark. Power normalization is then applied to the PCA projected features. This step is widely used in Fisher-Vector encoding schemes to improve their representation power. Finally, the similarity of two faces, $s(\mathbf{x}_1, \mathbf{x}_2)$, is taken to be their correlation score.

Pooling across Synthesized Views: Going beyond the system described in [3], we test also pooling of features obtained from image $\mathbf{I}$ and its rendered views. Specifically, where [3] separately matched the input image, its in-plane aligned view

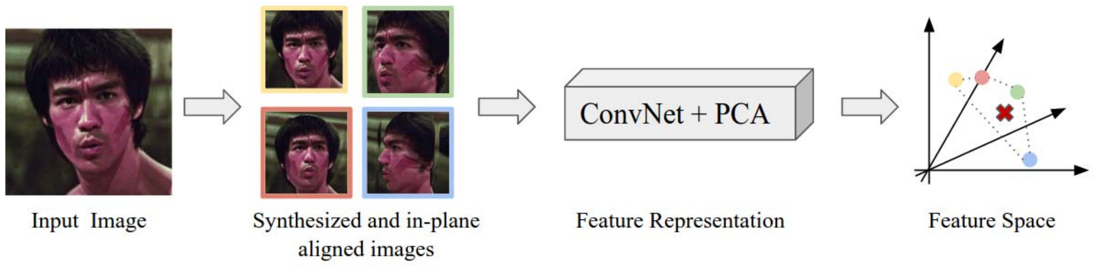

Fig. 4: Face representation: given an input image (left), it is rendered to novel viewpoints as well as in-plane aligned. These are all encoded by our CNN. The CNN features are then pooled by element wise average, obtaining the final representation.

and its rendered views, we extract CNN features for all these views and then pool them together using element-wise average. This process is visualized in Fig. 4.

The idea here follows similar techniques suggested by, e.g., [24]. Unlike them we use the pooled features directly, rather than train subsequent networks to process them. Although the idea of pooling multiple CNN responses over transformation is not new [22] and is extensively done in deep learning framework such as Caffé [25] by sampling random patches and averaging their features, we do it by exploiting the face domain and explicitly pool over synthetically generated out-of-plane transformations.

The rationale for this approach is that the CNN is trained to produce features which should be invariant to confounding factors (e.g., pose, illumination, expression, etc.) instead emphasizing subject identity. In practice, these confounding factors affect the representation produced by the CNN. These effects can be considered noise, which can sometimes overpower the subject's identity (the signal) in the deep feature representation. By pooling together features obtained by multiple, synthetically applied identity preserving transformations, we aim to suppress this noise and amplify the signal. As we later show, this process proved beneficial especially at very low false alarm rates (FAR).

Recognition with face templates: In some cases, subjects are described with a set of images and these images can originate from different sources or media files. This is the case in the recently released JANUS benchmarks for unconstrained face recognition [26]. These benchmarks represent a subject using *templates*; that is, a set of media files from heterogeneous sources, including still images, frames sampled from a video, or multiple videos. Following pooling of the rendered views of each test image (Sec. V-B), our system performs *media pooling* of the template features.

Specifically, a template is *flattened* by first averaging the representations obtained from video frames, independently for each video (i.e., *video pooling* [3]). Following this, a video is represented by a single feature vector, no matter the number of frames it originally contained. Next, a final representation for the template is obtained by pooling together the features from all images and all videos. Here, again, element-wise average is performed providing a single, feature representation per template. Note that this is the same as performing a single weighted average taking into account each source. Following this, we evaluate the similarity between two template as we did when comparing two images, by simply taking the correlation of two features.

VI. EXPERIMENTAL RESULTS

A. Comparing rendering methods

We compare our rendering approach to a number of recent methods proposed for rendering new views of face images and used for face recognition [1], [3], [5], [6]. The rendering method used by both [6] and [3] is not publicly available; we implemented it ourselves with the help of its authors. All run-times are reported on an Intel Core i7-4820K CPU @ 3.70GHz (4 cores), 32GB RAM and nVidia Titan X GPU.

We report run-times as well as other relevant aspects of these rendering systems: their support of multiple pose rendering (rather than, e.g., only frontalization); use of multiple 3D shapes; implementation programming language; dependency on OpenGL; and finally, public availability.

Our report is summarized in Tab. I. For a fair comparison, we measure the average time (Avg. Rend. Time) required by each method for rendering alone; time spent on pose estimation (and landmark detection) was excluded. Speeds were measured averaging over $2,000$ rendered views.

Our method, despite its implementation in high level, un-optimized Python code, can generate a rendered view in 14.27ms. The more recent method of [3], which uses optimized and compiled OpenGL code, requires more time (46.12ms) due to the repeated ray casting it performs and which we instead perform only once at preprocessing. Overall our method has a speed-up of about $\times 3.7$.

Method	**Avg. Rend. Time** (ms)	**Multi-Pose**	**Multi-Shape**	**Implement.**	**Avoid OpenGL**	**Pub. Aval.**
CVPRW13 [5]	N/A	Yes	No	Blender	No	No
CVPR15 [1]	66.33±9.51	No	No	MATLAB	Yes	Yes
CVPR16 [6]*	46.12±18.45	Yes	No	Optimized C++	No	No
ECCV16 [3]*	46.12±18.45	Yes	Yes	Optimized C++	No	No
Our renderer	14.27±1.98	Yes	Yes	Python	Yes	Yes

TABLE I: Overview of render times and various properties of recent face rendering methods. * Our own implementation.

Metrics →	Ver. (TAR@FAR)			Ide. (Rec.Rate)		
ConvNet: VGG19 ↓	1%	0.1%	0.01%	R1	R5	R10
Background and head vs. empty background						
Us, (no context)	85.2	68.0	44.2	90.5	96.0	97.5
Us, (context)	86.1	71.1	47.0	91.5	96.5	97.7
Pooling across Synth.						
Us, (real+our rend.)	86.1	71.1	47.0	91.5	96.5	97.7
Us, (pool.synth.)	86.0	70.9	49.0	91.4	96.4	97.5
Impact of incremental training						
Us, incremental	88.8	75.0	56.4	92.5	96.6	97.4

TABLE II: Comparison of various pipeline components on the IJB-A benchmark: showing the impact of face context, pooling across synthesized images and incremental training.

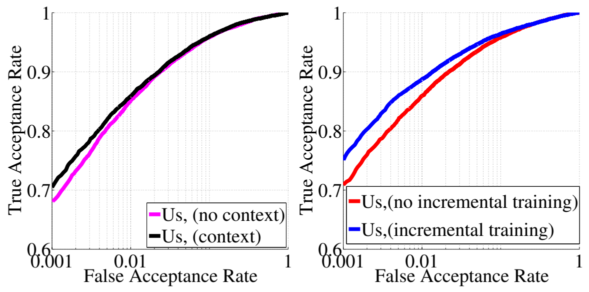

Fig. 5: ROC curves on the IJB-A verification for (left) the use of context; (right) incremental training. See also Tab. II.

To appreciate the significance of this, synthesizing novel views using multiple 3D shapes required our method less than four hours on a cluster with 96 cores for all the faces in the CASIA WebFace collection [2] (about 500,000 facial images). This is the entire, end-to-end run time.

B. Analysis of effects on face recognition

To assess the effects of our rendering we evaluated the face recognition pipeline described in Sect. V. Tab. II reports recognition performances with different experimental settings, showing the impact each has on performances. In particular we tested (1) the effect of rendering the background and entire head with the models prepared in Sec. IV-C vs. using only the face and a blank background; (2) the effect of pooling across synthesized images, as we proposed in Sec. V-B; and (3) incremental CNN training both with and without the background.

The importance of background: To our knowledge, previous face recognition methods for synthesis of new facial views, did not render the background [3], [4], [6], or else did not use it in their tests [1], [7]. We test the effect background and the head around the face both have on performance, by rendering our faces with and without the background plane and the head regions prepared in Sec. IV-C. To this end, our entire pipeline (training and testing) was executed with and without the background and head. Evident from Tab. II and Fig. 5 (left) is that the background improves verification rates at the low FAR rates by $\sim 3\%$.

Pooling rendered views: We test the impact of the pooling step of Sec. V-B. This pooling avoids matching rendered and real images separately, as done by [3], and so simplifies matching. Doing so also improves accuracy: Tab. II shows that despite the simpler matching scheme, accuracy improves. This is particularly evident in the low FAR of 0.01% where pooling adds 2% to the accuracy.

Incremental training: Although rendering faces with their context performs better than without, we next test if *combining both* can achieve even better results. We begin with a network trained on in-plane aligned images and rendered views without background. Once training saturated, we resumed training using rendered views for the same images, but now produced with backgrounds. Matching used faces rendered along with their context and pooled across views.

Surprisingly, the results in Tab. II show this network to perform better than all other combinations, improving at the very low FAR=0.01% by $\sim 7\%$ and $\sim 4\%$ at FAR=0.1%. Recognition rates also improve across all ranks. This may be due to a gradual adaptation of the network to faces: Networks trained with rendered views without backgrounds, were initialized with weights obtained by training on a different domain (ImageNet). By training the network incrementally, the final network was initialized using images from the target problem domain, albeit missing their context.

Comparison with state-of-the-art: We compare our best

	CS2					**IJB-A**				
Methods	Ver. (TAR@FAR)		Ide. (Rec.Rate)			Ver. (TAR@FAR)		Ide. (Rec.Rate)		
Metrics	1%	0.1%	R1	R5	R10	1%	0.1%	R1	R5	R10
COTS [26]	58.1	37	55.1	69.4	74.1	–	–	–	–	–
GOTS [26]	46.7	25	41.3	57.1	62.4	40.6	19.8	44.3	59.5	–
OpenBR [27]	–	–	–	–	–	23.6	10.4	24.6	37.5	–
Fisher Vector [28]	41.1	25	38.1	55.9	63.7	–	–	–	–	–
Wang *et al.* [29]	–	–	–	–	–	73.2	51.4	82.0	92.9	–
Chen *et al.* [30]	64.9	45	69.4	80.9	85.0	57.3	–	72.6	84.0	88.4
Deep Multi-Pose [4]	89.7	–	86.5	93.4	94.9	78.7	–	84.6	92.7	94.7
Pooling Faces [31]	87.8	74.5	82.6	91.8	94.0	81.9	63.1	84.6	93.3	95.1
PAMs [6]	89.5	78.0	86.2	93.1	94.9	82.6	65.2	84.0	92.5	94.6
Face Sp. Aug. [3]	92.6	82.4	89.8	95.6	96.9	88.6	72.5	90.6	96.2	97.7
Swami S. *et al.* [32]	–	–	–	–	–	87.1	76.6	92.5	–	97.8
Swami S. *et al.* [33]	–	–	–	–	–	79	59	88	95	–
Chen *et al.* [30]	92.1	78	89.1	95.7	97.2	83.8	–	90.3	96.5	97.7
Swami S. *et al.* + TPE [32]	–	–	–	–	–	90.0	81.3	93.2	–	97.7
Us	93.9	86.1	92.3	96.4	97.3	88.8	75.0	92.5	96.6	97.4

TABLE III: Performance analysis on JANUS CS2 and IJB-A respectively for verification (ROC) and identification (CMC). The methods of [33], [30] and [32] all perform supervised training on all of the test benchmark's training splits.

performing pipeline with published results on the JANUS CS2 and IJB-A benchmarks (Tab. III). Our report separates methods according to the type of training applied to the target benchmark: our method used computationally cheap, unsupervised PCA whereas others used supervised techniques, trained on each of the ten training splits.

Compared to methods which, like us, did not use the training splits for supervised training, our method achieves state of the art results. It moreover falls only slightly behind methods which explicitly adapt to the test domain by performing supervised training on each training split of the test benchmarks. This implies that by careful rendering, we effectively introduce many of the appearance variations encountered in the target domain.

Qualitative results: Fig. 6 provides a few qualitative rendered results. The effect of strong facial expressions and our use of a static, generic 3D model is shown in Fig. 6(b). Typical failures due to poor facial landmark detection are additionally provided in Fig. 6(c).

VII. CONCLUSIONS

This paper discusses practical aspects of rendering new views for faces appearing in unconstrained views. It shows that by using generic 3D faces and rendering to fixed views, much of the computational effort required to render faces can be performed at preprocessing. This is significant, as it allows for rapid generation of huge face sets for providing CNNs with much needed appearance variations. The simplicity of our method makes it easy to implement and embed in face recognition pipelines and port to different platforms. To these contributions we add also a number of technical novelties (e.g., pooling across rendered 3D views, incremental training). Taken together, these provide recognition performances comparable to methods which require computationally expensive domain adaptation by supervised training on the target domain.

ACKNOWLEDGMENTS

This research is based upon work supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via IARPA 2014-14071600011. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purpose notwithstanding any copyright annotation thereon.

REFERENCES

[1] T. Hassner, S. Harel, E. Paz, and R. Enbar, "Effective face frontalization in unconstrained images," in *Proc. Conf. Comput. Vision Pattern Recognition*, 2015.

[2] D. Yi, Z. Lei, S. Liao, and S. Z. Li, "Learning face representation from scratch," *arXiv preprint arXiv:1411.7923*, 2014.

[3] I. Masi, A. T. an Trãn, T. Hassner, J. T. Leksut, and G. Medioni, "Do We Really Need to Collect Millions of Faces for Effective Face Recognition?" in *European Conf. Comput. Vision*, 2016.

[4] W. AbdAlmageed, Y. Wu, S. Rawls, S. Harel, T. Hassner, I. Masi, J. Choi, J. Leksut, J. Kim, P. Natarajan, R. Nevatia, and G. Medioni, "Face recognition using deep multi-pose representations," in *Winter Conf. on App. of Comput. Vision*, 2016.

[5] I. Masi, G. Lisanti, A. Bagdanov, P. Pala, and A. Del Bimbo, "Using 3D models to recognize 2D faces in the wild," in *Proc. Conf. Comput. Vision Pattern Recognition Workshops*, 2013.

[6] I. Masi, S. Rawls, G. Medioni, and P. Natarajan, "Pose-Aware Face Recognition in the Wild," in *Proc. Conf. Comput. Vision Pattern Recognition*, 2016.

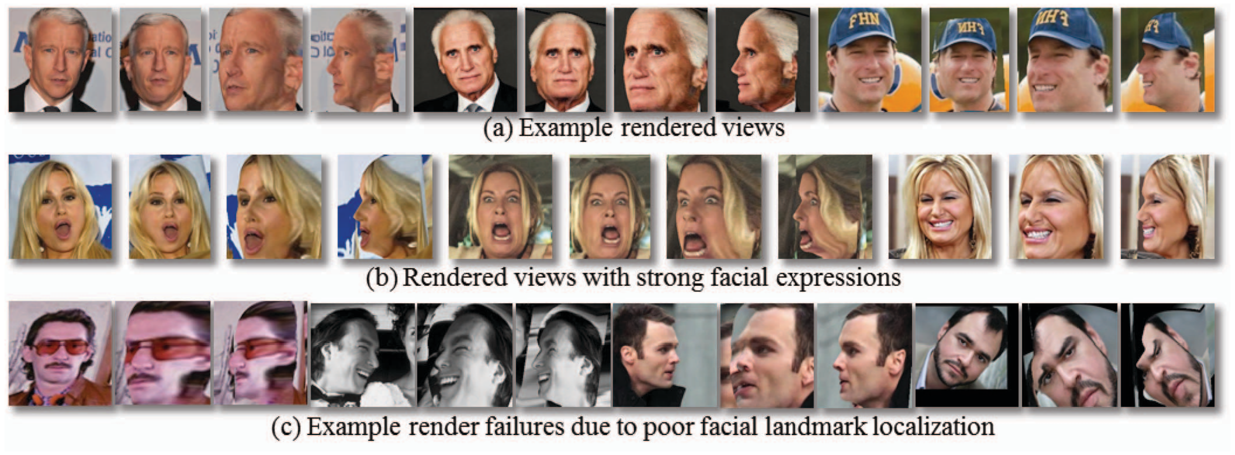
(a) Example rendered views
(b) Rendered views with strong facial expressions
(c) Example render failures due to poor facial landmark localization

Fig. 6: Qualitative rendered faces.

[7] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, “Deepface: Closing the gap to human-level performance in face verification,” in *Proc. Conf. Comput. Vision Pattern Recognition*. IEEE, 2014.

[8] C. Ferrari, G. Lisanti, S. Berretti, and A. Del Bimbo, “Dictionary learning based 3D morphable model construction for face recognition with varying expression and pose,” in *3DV*, 2015.

[9] ——, “Effective 3d based frontalization for unconstrained face recognition,” in *Int. Conf. on Pattern Recognition*, 2016.

[10] V. Blanz, K. Scherbaum, T. Vetter, and H. Seidel, “Exchanging faces in images,” *Comput. Graphics Forum*, vol. 23, no. 3, 2004.

[11] V. Blanz and T. Vetter, “Morphable model for the synthesis of 3D faces,” in *Proc. ACM SIGGRAPH Conf. Comput. Graphics*, 1999.

[12] F. Yang, J. Wang, E. Shechtman, L. Bourdev, and D. Metaxas, “Expression flow for 3D-aware face component transfer,” *ACM Trans. on Graphics*, vol. 30, no. 4, p. 60, 2011.

[13] I. Kemelmacher-Shlizerman and R. Basri, “3D face reconstruction from a single image using a single reference face shape,” *Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 2, pp. 394–405, 2011.

[14] T. Hassner, “Viewing real-world faces in 3D,” in *Proc. Int. Conf. Comput. Vision*. IEEE, 2013, pp. 3607–3614, available: www.openu.ac.il/home/hassner/projects/poses.

[15] J. F. Hughes, A. Van Dam, J. D. Foley, and S. K. Feiner, *Computer graphics: principles and practice*. Pearson Education, 2014.

[16] X. Xiong and F. De la Torre, “Supervised descent method and its applications to face alignment,” in *Proc. Conf. Comput. Vision Pattern Recognition*. IEEE, 2013.

[17] T. Baltrusaitis, P. Robinson, and L.-P. Morency, “Constrained local neural fields for robust facial landmark detection in the wild,” in *Proc. Int. Conf. Comput. Vision Workshops*, 2013.

[18] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[19] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[20] P. Paysan, R. Knothe, B. Amberg, S. Romdhani, and T. Vetter, “A 3d face model for pose and illumination invariant face recognition,” in *Advanced Video and Signal Based Surveillance, 2009. AVSS ’09. Sixth IEEE International Conference on*, Sept 2009, pp. 296–301.

[21] X. Zhu, Z. Lei, X. Liu, H. Shi, and S. Li, “Face alignment across large poses: A 3d solution,” in *Proc. IEEE Computer Vision and Pattern Recognition*, Las Vegas, NV, June 2016.

[22] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *Int. Conf. on Learning Representations*, 2015.

[23] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, “Imagenet large scale visual recognition challenge,” *Int. J. Comput. Vision*, pp. 1–42, 2014.

[24] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, “Multi-view convolutional neural networks for 3d shape recognition,” in *Proc. Int. Conf. Comput. Vision*, 2015, pp. 945–953.

[25] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” *arXiv preprint arXiv:1408.5093*, 2014.

[26] B. F. Klare, B. Klein, E. Taborsky, A. Blanton, J. Cheney, K. Allen, P. Grother, A. Mah, M. Burge, and A. K. Jain, “Pushing the frontiers of unconstrained face detection and recognition: IARPA Janus Benchmark A,” in *Proc. Conf. Comput. Vision Pattern Recognition*, 2015.

[27] J. Klontz, B. Klare, S. Klum, E. Taborsky, M. Burge, and A. K. Jain, “Open source biometric recognition,” in *Int. Conf. on Biometrics: Theory, Applications and Systems*, 2013.

[28] J.-C. Chen, S. Sankaranarayanan, V. M. Patel, and R. Chellappa, “Unconstrained face verification using fisher vectors computed from frontalized faces,” in *Int. Conf. on Biometrics: Theory, Applications and Systems*, 2015.

[29] D. Wang, C. Otto, and A. K. Jain, “Face search at scale: 80 million gallery,” *arXiv preprint*, vol. arXiv:1507.07242, 2015.

[30] J.-C. Chen, V. M. Patel, and R. Chellappa, “Unconstrained face verification using deep cnn features,” in *Winter Conf. on App. of Comput. Vision*, 2016.

[31] T. Hassner, I. Masi, J. Kim, J. Choi, S. Harel, P. Natarajan, and G. Medioni, “Pooling faces: Template based face recognition with pooled face images,” in *Proc. Conf. Comput. Vision Pattern Recognition Workshops*, June 2016.

[32] S. Sankaranarayanan, A. Alavi, C. Castillo, and R. Chellappa, “Triplet probabilistic embedding for face verification and clustering,” in *Int. Conf. on Biometrics: Theory, Applications and Systems*, 2016.

[33] S. Sankaranarayanan, A. Alavi, and R. Chellappa, “Triplet similarity embedding for face verification,” *arxiv preprint*, vol. arXiv:1602.03418, 2016.