Modeling Facial Expressions in 3D Avatars from 2D Images

Emma Sax
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
saxxx027@morris.umn.edu

ABSTRACT

Realistic computer graphics have become increasingly important with the rise of video games, virtual worlds, and animated television. Humans and their facial expressions have traditionally been tricky objects to accurately model. We discuss a basic algorithm and an improved variant which both address this challenge in real time. The benefit of the algorithms' real-time performance is that users can be provided with accurate 3D visual renderings of human faces based on 2D video frames, as the video is being recorded. We show the results of the algorithms discussed, and describe the benefits and drawbacks of the various algorithms and their animations.

Keywords

facial expression tracking, facial animation, blendshape models, regression models

1. INTRODUCTION

Facial expressions convey the emotional state of an individual, such as whether a person is upset, happy, impatient, or in pain. Because humans use facial expressions so frequently and most humans are naturally adept at comprehending these subtle signals, it is beneficial for users of graphics programs if accurate facial expressions are modeled and rendered. The widespread use of cameras on smartphones, laptops, and tablets has increased the demand for performant facial modeling for consumer-level applications. An example would be a video gamer who is using a character, or avatar, that tracks the gamer's facial expressions in real time. It is useful for the gamer if their avatar is reflecting accurate, real-time facial expressions. Therefore, it is necessary to have algorithms available that are capable of creating these renderings quickly and accurately.

Modeling a facial expression refers to designing the shape of the face, head, and facial expression. *Animating*, on the other hand, in when the facial models are overlayed with animations and multiple models are strung together to create a moving picture. Both modeling and animating are necessary steps when developing a fully rendered avatar of a facial expression. For the purposes of this discussion, only the modeling step will be described. When algorithms model facial expressions, a 2D video frame of a user must be captured and the algorithms must track specific landmarks on the user's face (e.g. tip of the nose, corners of the eyes, etc.). Then, the algorithm can render any number of 3D avatars with the same facial expression, as shown in Figure 1. This whole process should occur in real time in order to provide users of the system with immediate results.

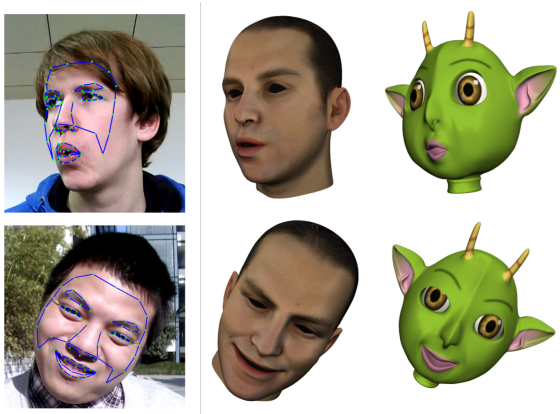

Figure 1: An example of using 2D photographs of users to generate both graphic representations and avatars with similar facial expressions. The lines overlayed on the 2D photographs represent important physical features along the user's face that are used to generate facial models. [4]

The real-time algorithm for accurately modeling the expressions seen in Figure 1 is the 3D Shape Regression Tracking solution outlined in Cao et al. [4]. After describing this algorithm in Section 3, we show an improved adaptation: Displaced Dynamic Expression (DDE) Regression Tracking in Section 4, as described in Cao et al. [3]. In Section 5, we end with some comparisons and conclusions.

2. BACKGROUND

Ekman's *Facial Action Coding System* (FACS) is a coding system designed to track human facial muscles and expressions and place the results in a database. It categorizes human facial movements by their appearance on the face. Us-

UMM CSci Senior Seminar Conference, December 2016 Morris, MN.

Figure 2: Examples of basic linear blendshape models. These blendshapes have been morphed to represent four common facial expressions (from left to right): a yawn, a smirk, a kiss, and a basic wink. [2]

ing FACS, any anatomically possible facial expression can be coded. *Action Units* (AUs) are contractions or relaxations of one or more facial muscles. Facial expressions are categorized by which AUs are used to create each individual facial expression. Therefore, facial expressions are represented by which facial muscles are contracted and relaxed [10].

The two algorithms described here use linear *blendshape models* to help generate 3D graphics. A *blendshape model* generates a facial pose as a linear combination of a number of *blendshapes*. A *blendshape* is is a visual approximation of a facial expression in which a single collection of points, called a *mesh*, are deformed to a series of fixed vertex positions. These vertex positions can represent a detailed facial expression, as shown in Figure 2. The base shape of the blendshape model is an expressionless face where all AUs are relaxed [7].

FaceWarehouse is a public, online database of 3D facial expression models and blendshapes. It is composed of facial shapes that were generated from 150 individuals from a range of ages and ethnicities. FaceWarehouse originally used a Kinect system to capture the geometry and texture information from each subject. It then tracked features on the subjects' faces using FACS. Lastly, using these tracked features, generic blendshape models were calculated for each generic facial expression [5].

Both of the algorithms we will describe use *regressors*. A *regressor* is an algorithm that is specifically trained to analyze the relationships between input data and output data that were computed in previous runs of the algorithm. These relationships allow the regressor to predict what the next output should be. Because regressors rely on previous data, regressors must be trained in advance. For the regressors we will describe here, the regressor is trained to take single video frames as input, look at previous runs of the same regressor in which the inputs are similar, and use those previously generated 3D facial shape outputs to predict what the most likely facial shape output is for each video frame.

3. 3D SHAPE REGRESSION TRACKING

The first basic solution for rendering complete facial expressions in graphics is 3D Shape Regression Tracking, as described in Cao et al. [4]. This algorithm utilizes a setup stage with multiple parts. First, using a standard camera, a one-time training step captures images of a specific user, and a set of facial landmarks are placed on the 2D images, as described Section 3.1. Second, a single blendshape is generated for each captured image. This creates a user-specific blendshape model. Third, the regressor is trained with the captured images as input data and the corresponding blendshapes as output data. After the setup is completed, the regressor is able to take 2D images from video frames as input and accurately output a 3D facial shape. The regressor is trained in advance in so that the regressor is able to use both the previous outputs of the regressor and the trained data as a prediction model.

Lastly, animations are overlayed on the regressor's results to add coherence in the overall 3D outcome. The processes for real-time tracking and adding animations are described in Weise et al. [9]. The final results of the entire algorithm can be seen in the right and middle columns of Figure 1.

3.1 Gathering Data

The 3D Shape Regression Tracking solution consists of a one-time data gathering step that must be completed for each individual user. In this step, 60 images are captured. In each image, the user shows pre-defined face positions and expressions. All of the images are categorized into two groups. The first group contains different head poses, but all with a neutral facial expression. The poses are expressed in terms of head angles, such as turning the head side to side, forward and backward, up and down, and tilting right and left. The second category consists of different facial expressions, which include mouth stretch, smile, brow raise, and wink left eye.

After capturing images, a set of *facial landmarks* are automatically located using a 2D facial feature alignment technique used in Cao et al.'s earlier work [6]. A *facial landmark* is a point that is placed on a facial feature of a user and is used to track that feature as the user moves. This technique automatically finds 75 landmark positions in each image captured. These 75 positions are broken up into 60 *internal landmarks* and 15 *contour landmarks*. *Internal landmarks* are places on the user's face that describe features such as the corners of the user's lips, the placement of the eyebrows, etc. *Contour landmarks* are positions that describe the shape of the user's face and head. Although the facial feature alignment technique automatically places the landmarks, all of them can be overwritten manually by the user if they are incorrect. The positions located on the image will each eventually correspond to a landmark on the 3D facial shape.

3.2 User-Specific Blendshape Generation

Now that all of the initial data has been gathered, the next step is to generate a blendshape for each input image. A single FaceWarehouse blendshape example will serve as a generic blendshape to work with, and 46 FACS AUs are used on the blendshape to help develop specific facial expressions.

The generic model the solution uses has two attributes: identity of the individual and expression. Identity is shown within blendshapes by making slight adjustments to specific points to account for different head and facial feature shapes. This is necessary because each user will have a uniquely different face shape. Therefore, the algorithm must compute the user's identity for each new user.

Expressions are made by contracting and stretching various points on the mesh that correspond to facial muscles, as shown in the leftmost blendshape in Figure 2, where the points in the mesh that correspond to the chin are stretched downward, making it look like the face is yawning.

In order to compute accurate blendshapes for each input image, a transformation is performed to allow the regressor

to take head turns, rotations, and translations into account. This is done by finding a transformation that minimizes the amount of error between the projections of the 3D landmark vertices on the mesh and the 2D labeled landmark positions. After the identity, expression, and transformation has been found, then a set of 60 expression blendshapes (one for each input image) for that specific user can be computed.

3.3 Selecting Specific Training Data

Now that all input images have expression blendshapes, a set of data can be selected for training the regressor. Cao et al. [4] describes a few steps required before selecting training data. First, for each image, the corresponding 3D facial shape must be found and assembled to be used as potential training data for the regressor.

The second step is an augmentation step in which the set of captured images and their 3D facial shapes is increased to achieve better generalization of the regressor's facial shapes. For each input image, the corresponding facial shape is translated and transformed via a transformation matrix to cover new head angles and rotations. The transformation matrix used is recorded, and this provides enough data to retrieve the appearance data of the original facial shape. This process increases the amount of original data and the range of 3D facial shapes.

In the third step, the algorithm selects the training set for the regressor. Because the regressor will eventually be used on video frames (which occur in sequential order), it is likely that the output for a frame is similar to the output of its previous frame. However, it is also useful to take randomness into account. For each given image, a set of most similar shapes are collected from the original shapes. Then, a set of randomly chosen shapes from the augmentation step above are chosen. The union of these two sets make up the initial training data.

3.4 Training the Regressor

Now that the initial training set of data has been selected, the regressor can be trained to genereate 3D facial landmarks from 2D images. During runtime, the regressor utilizes two parts. Therefore, each part must be trained individually. This process can be seen in Algorithm 1.

Algorithm 1 Training the Regressor

Input: N Training data (I_i, M_i^a, S_i, S_i^c)
Output: Two-part regressor

```
/* part one*/
for t = 1 to T do
    {d_p^t} ← randomly generate P offsets
    for i = 1 to N do
        V_i ← App(I_i, M_i^a, S_i^c, {d_p})
        Compute the P^2 feature values for V_i

    /* part two*/
    for k = 1 to K do
        for f = 1 to F do
            Y_f ← randomly generate a direction
            for i = 1 to N do
                δS_i ← S_i − S_i^c
                c_i ← δS_i × Y_f
            Find the index-pair with the highest
                correlation with {c_i} and randomly
                choose a threshold value
        for i = 1 to N do
            Calculate new features in V_i using the F
                index-pairs
            Compare the features to the thresholds
                to determine which bin the data
                belongs to
        for each of the 2^F bins do
            Compute δS_{b_i}
            for each training data l in the bin do
                S_l^c ← S_l^c + δS_{b_i}
```

For the first part of the regressor, a set of *index-pair features* must be generated. In general, an *index-pair feature* relates the position of a 2D landmark with the position of the 3D landmark on the blendshape mesh. These features are computed from the *appearance vector* of a 3D facial shape (S_i^c). *Appearance vectors* are composed of the intensities of P randomly selected 3D facial points p. Each p is represented as the sum of a landmark position S_i^c and an offset d_p.The intensity value of p is gathered from p's corresponding 2D positions in the original image i. When all of the intensity values of the P points are assembled, they form the appearance vector V_i. This procedure of calculating the appearance vector for S_i^c is denoted as $App(I_i, M_i^a, S_i^c, \{d_p\})$ where M_i^a refers to the transformation matrix used earlier and I_i is the original 2D image. For each appearance vector, P^2 index-pair features are computed by calculating the differences between each pair of elements in V_i.

For the second part of the regressor, effective index-pair features must be selected. To choose these features, the algorithm calculates the difference between the ground truth shape S_i and the current shape S_i^c. Then, the difference δS_i is multiplied by a randomly selected direction to produce a scalar c_i. Finally, the index-pair with the highest correlation with the set of scalars is chosen. The technique of randomly multiplying by a direction has been proven to be a powerful tool for dimensionality reduction, as shown in Bingham and Mannila [1]. This process is repeated F times to yield F index-pair features. All F features together make a primitive regressor structure called a *fern*.

In a fern, each of the F features is assigned a random threshold value. The thresholds divide the space of all index-pair features into 2^F *bins*, or categories. Next, for each index-pair, the features in V_i are calculated and the features are compared to the thresholds to decide which bin to place the index-pair in (or how to categorize each index-pair). After classifying all of the training data into a bin group, compute the regression output δS_b for each bin. Lastly, for all training data l in each bin, update the current shape with the regression output: $S_l^c = S_i^c + \delta S_b$.

The regressor training process is iterated T times, with K ferns to progressively refine the regression output. From experiments, it has been found that results are optimal when $T = 10$, $K = 300$, $P = 400$, and $F = 5$. Increasing the number of iterations brings more accuracy, but takes longer and adds computational costs.

3.5 Runtime Regression

With the 3D shape regressor fully trained, the 3D facial shape S for the image I in the current video frame can be obtained. The algorithm takes the previous frame's output facial shape S' and the current input video frame I as input.

Algorithm 2 Runtime Regression

Input: Previous facial shape S', current image I
Output: Current frame's facial shape S

```
Get the shape S_r in {S_i^o} most similar to S'
Find M^a that best aligns S' with S_r
S'* ← M^a S'
{S_l} ← Choose L shapes most similar to S'*
for l = 1 to L do
    for t = 1 to T do
        V ← App(I, (M^a)^{-1}, S_l, {d_p^t})
        for k = 1 to K do
            Get the F index-pairs recorded
                during training
            Calculate the F feature values
            Use the feature values to locate its bin b
                in the fern
            Get δS_b in b
            S_l ← S_l + δS_b
S* ← Compute the median shape of {S_l, 1 ≤ l ≤ L}
S ← (M^a)^{-1} S*
```

The output is a prediction of the current frame's 3D facial shape S. This process is outlined in Algorithm 2.

First, we find a shape S_r in the original shape set $\{S_i^o\}$ that is most similar to S'. Then, we find a transformation matrix M^a that aligns S' and S_r and use that matrix to make S'^*. Then, the L most similar shapes to the transformed shape S'^* are found from all 3D shapes in the training data. Each chosen shape S_l is used as an initial shape estimate and is passed through the regressor.

In the first part of regression, the appearance vector V is calculated using the image, current shape, inverse transformation matrix, and the offsets. In the second part, the F index-pairs recorded during training are gathered and the appearance feature values for V are calculated. The feature values are compared with the recorded thresholds to locate the correct bin b, which contains δS_b. Finally, we can update the current shape: $S_l = S_l + \delta S_b$. Then, after obtaining all regression results for all $\{S_l\}$, the median shape has its transformation matrix inverted and becomes S for the current frame.

There are two elements of this regression model that make it more successful than other previous regressors. First, because the regression uses a set of similar shapes $\{S_l\}$ instead of a single shape to generate S, this allows the solution to better deal with uncertainty and to avoid error accumulation. Second, because the regressor chooses similar shapes to the transformed shape S'^* instead of choosing shapes similar to the single S', the algorithm can account for different head orientations and directions. The benefits of these added elements can be seen clearly in Figure 3 in the middle column, where the bottom two rows' estimates of the user's right cheek are off, and in the right column, where the bottom two rows' estimates of the user's left cheek are off.

3.6 Results

In the first part of the setup stage (data gathering), the user performs a sequence of 60 facial poses/expressions. This takes less than 10 minutes, and is completed once for each individual user. It takes approximately 25 minutes for a first-time user to adjust 2D landmark positions for facial features. The remaining data tasks, including the blendshape generation, camera calibration, training data preparation, and training the 3D shape regressor, are all completed in less than 10 minutes. In total, the setup and preprocessing takes less than 45 minutes per user [4]. However, for this type of application, a 45 minute setup step is longer than desirable.

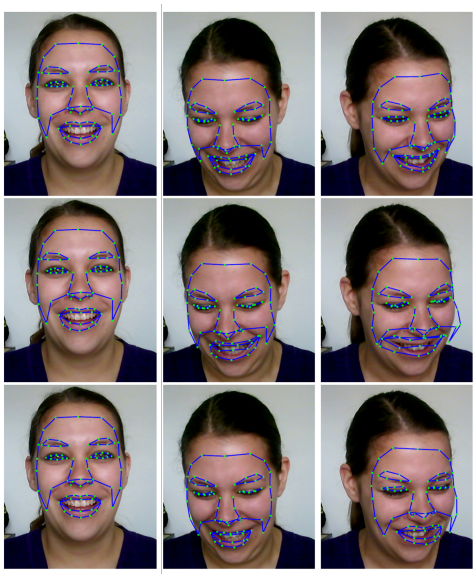

Figure 3: Comparisons of the Cao et al. [4] results (top row) compared to using the previous frame's shape as the initial shape (middle row) and omitting the transformation step (bottom row). [4]

This algorithm was implemented on a PC with an Intel Core i7 (3.5GHz) CPU with an ordinary web camera, recording 640×480 images at 30 fps. At runtime, the computational time for rendering a 3D avatar takes less than 5 milliseconds per frame. Face tracking takes about 8 milliseconds per frame, making the overall runtime performance less than 15 milliseconds. This makes this solution promising for consumer-level applications since it renders in real time.

However, there are also limitations to this approach. It relies on facial appearance information in the video, and therefore can only deal with partial occlusions. When there are larger occlusions present, the algorithm may output incorrect shapes. The approach also struggles when trying to handle dramatic changes in lighting. Although the approach works well when the background is changing slightly, it fails to output accurate shapes when the lighting environment differs substantially from the environment during setup, as can be seen in Figure 5.

4. DDE REGRESSION TRACKING

3D Shape Regression Tracking was able to produce 3D renderings from 2D video frames in real time, but with some inaccuracies. Therefore, some improvements were made upon the 3D Shape Regression Tracking solution. This refined variant is known as the Displaced Dynamic Expression (DDE) Regression Tracking. It is fully described in Cao et al. [3]. Unlike the previous algorithm, this solution does not require calibration for each individual user, and instead uses a DDE model to represent the facial shapes for any given user. In

this algorithm, public FaceWarehouse images are used as training data for a generic regressor. Through the training and use of the generic regressor, the regressor's results can be applied to any user to immediately infer accurate 2D facial landmarks and 3D facial shapes from 2D video frames. The overall runtime is almost identical to the runtime seen in Section 3.5. After the use of the regressor for each video frame, the results are put through a post-processing and Dynamic Expression Model (DEM) adaptation step, which further refines the regressor's output. More detailed explanations of the post-processing and DEM adaptation steps can be found in Cao et al. [3].

4.1 DDE Model

The Displaced Dynamic Expression (DDE) model is designed to represent both the 3D shape of the user's facial expressions and the 2D facial landmarks. Regressor training data, the data stored, and the runtime data will all be in the form of a DDE. Similarly to Weise et al. [9] and Cao et al. [4], the 3D facial shape is represented by a linear combination of expression blendshapes, plus a random rotation and translation. Like the previous algorithm, the blendshape model is based on 46 FACS AUs. The expression blendshapes of users are calculated in a similar way to the first algorithm.

A 2D facial landmark is generated by adding a 2D displacement and the projection of the landmark's corresponding vertex on the 3D facial mesh. Because the user identity is not pre-calibrated, the 2D displacement of facial landmarks is what allows for changes in user identity. Therefore, the 2D facial shapes are represented by the set of all 2D landmarks.

4.2 Preparing and Constructing Training Data

The training algorithm takes a set of facial images from FaceWarehouse as input. Because of this, the input data represents a variety of users. For each input image I, 73 2D landmarks are manually labeled to produce the 2D facial shape. All images of the same user are identified, since there could be a large number of unique users. By identifying each individual user's images, all shape vectors are optimized together with the same identity coefficients.

Similarly to Cao et al. [4], this algorithm must gather the 3D facial shape from all of the 2D input images. However, this algorithm must also assemble the 2D landmark displacements. Both of these pieces together make training data in the form of a DDE. Both the user's identity and shape vector can be solved by alternately optimizing each parameter while fixing the others in iteration.

This algorithm uses a generic *Cascaded Pose Regression* (CPR) training regressor described in Cao et al. [6]. This method requires creating *guess-truth pairs* for each image in order to relate the differences in parameters to image features. A *guess-truth pair* contains both a ground truth shape vector and a guessed shape vector. Both vectors are optimized at the same time in order to avoid introducing a change in non-regressed input parameters (i.e. camera calibration and user identity).

By using this method of construction, these training pairs simulate cases where input parameters may be inaccurate. When this occurs, the regressor is expected to produce large displacements to get correct landmarks. This is demonstrated in Figure 4 where the user identity from the middle image is applied to the lefthand image, resulting in inaccurate 2D landmarks. Therefore, because the ground truth shape vector and the guessed shape vector are optimized at the same time, the training pairs can account for the necessary displacements needed to render accurate 2D landmarks, shown in the rightmost image.

Figure 4: Random identity example where the user identity from the center image is applied to the image on the far left while keeping original displacements, resulting in inaccurate 2D landmark positions. Therefore, displacements must be recomputed to get correct landmark positions (far right). [3]

4.3 Training the Regressor

After constructing all of the training pairs, then a regression function from the guessed shape vector to the ground truth shape vector can be learned. The regression function is based on information from the input image. This algorithm follows the same two-part regression approach from Cao et al. [4], which is also seen in Algorithm 1. There are two differences between the regression function in Algorithm 1 and this function. The first is that this solution uses $T = 15$ to increase accuracy. The second is that this algorithm's shape representation is a combination of 3D shapes and 2D displacements. Therefore, this solution must account for this difference.

Instead of representing the 3D facial points p as a sum of a landmark position and an image offset, as in the previous algorithm in Section 3.4, this solution will represent p as a linear combination of two 2D landmarks. The position of a p is defined as the center coordinates in a triangle formed by 2D landmarks. For each of the 400 p, a triangle in reference to the 2D facial shape is found where the center is closest to the point. Then, the point's center coordinates are computed. During appearance vector extraction, p is located according to the triangle positions and the center coordinates. The reference 2D facial shape is computed by averaging the 2D shapes of all training images.

4.4 Results

During setup, the regressor training takes about 6 hours. However, this setup step must only occur once, since the algorithm is designed to generate 3D facial landmarks without the user identity previously being set. This algorithm was implemented on a PC with an Intel Core i5 (3.0GHz) CPU, with an ordinary web camera producing 640×480 video frames at a maximum of 30 fps. During runtime, for each video frame, this approach takes approximately 12 milliseconds to regress the shape vector and about 8 milliseconds to execute the post-processing and DEM adaptation steps mentioned above. Therefore, it takes approximately 20 milliseconds to render a 3D avatar for each video frame [3].

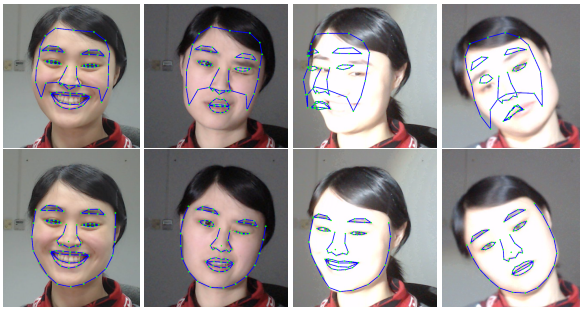

Figure 5: The DDE Regression Tracking approach (bottom row) in comparison to the 3D Shape Regression Tracking approach (top row) under significant lighting changes. [3]

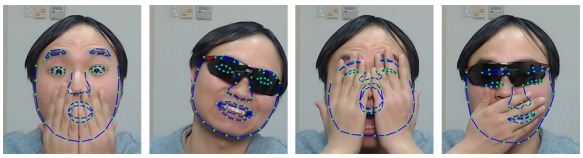

Figure 6: The DDE Regression Tracking approach properly handling some partial occlusions (left two images), but failing with larger occlusions (right two images). [3]

Although the runtime for this algorithm takes longer than the runtime for the previous algorithm, 20 milliseconds and 15 milliseconds respectively, the main benefit of this solution over the 3D Shape Regression Tracking solution is that any user can be instantly tracked; no data gathering step is necessary. The users can also be switched during the video, with little noticeable lag in the output results.

Experiments have also shown that this solution is more robust than the previously described solution. This newer solution can handle more dramatic lighting changes, as shown in Figure 5 where the top row is results from the 3D Shape Regression Tracking and the bottom row is from the DDE Regression Tracking. It can be seen that the two rightmost images on the top row are incorrect in the estimates of 3D facial landmarks, while the bottom row is much more accurate. This solution also does considerably better than the previous algorithm when handling head rotations and partial occlusions. However, the solution still can show inaccurate results when large portions of the user's face are covered, as shown in Figure 6, where the left two images are showing a partially occluded user and the right two images are showing users with large occlusions.

5. CONCLUSION

Here we have examined a primary solution for rendering 3D virtual images from 2D video frames in real time. In the initial solution, input images of a specific user are gathered, blendshapes are derived for that user, and the regressor is trained with the user's data and the blendshape model. Finally, the regressor computes the 3D facial landmark positions in real time and overlays animations on top of the regressor's results. In the improved solution, a DDE model is used to represent the input images and the images' corresponding data, and the algorithm derives guess-truth pairs to select training data. Then, the regressor is trained with the selected data and features represented as coordinates in a triangle formed by 2D landmarks. During runtime, the regressor computes 3D facial landmarks from 2D video frames, before the post-processing and DEM adaptation steps occur. Lastly, animations are overlayed on top of the results.

Both of these solutions work in real time, as stated in Section 3.6 and Section 4.4, and both solutions are successful at rendering accurate 3D facial animations from 2D video frames within certain conditions. It is also shown in Figure 5 that the DDE Regression Tracking works considerably better than the 3D Shape Regression Tracking when there are dramatic lighting changes.

Further research in this area includes a solution driven by both visual and audio data, Liu et al. [8], and a solution that works completely online without the use of facial markers or training stages, Bouaziz et al. [2].

Acknowledgments

I would like to extend my gratitude to Nic McPhee, KK Lamberty, Jacob Opdahl, Alice Barnett, Paul Friederichsen, and my family and friends for all of their support, constructive criticism, and feedback on this paper.

6. REFERENCES

[1] E. Bingham and H. Mannila. Random projection in dimensionality reduction: Applications to image and text data. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 245–250, New York, NY, USA, 2001. ACM.

[2] S. Bouaziz, Y. Wang, and M. Pauly. Online modeling for realtime facial animation. *ACM Trans. Graph.*, 32(4):40:1–40:10, July 2013.

[3] C. Cao, Q. Hou, and K. Zhou. Displaced dynamic expression regression for real-time facial tracking and animation. *ACM Trans. Graph.*, 33(4):43:1–43:10, July 2014.

[4] C. Cao, Y. Weng, S. Lin, and K. Zhou. 3d shape regression for real-time facial animation. *ACM Trans. Graph.*, 32(4):41:1–41:10, July 2013.

[5] C. Cao, Y. Weng, S. Zhou, Y. Tong, and K. Zhou. Facewarehouse: A 3d facial expression database for visual computing. *IEEE Transactions on Visualization and Computer Graphics*, 20(3):413–425, Mar. 2014.

[6] X. Cao, Y. Wei, F. Wen, and J. Sun. Face alignment by explicit shape regression. *Int. J. Comput. Vision*, 107(2):177–190, Apr. 2014.

[7] J. P. Lewis, K. Anjyo, T. Rhee, M. Zhang, F. Pighin, and Z. Deng. Practice and Theory of Blendshape Facial Models. In S. Lefebvre and M. Spagnuolo, editors, *Eurographics 2014 - State of the Art Reports*. The Eurographics Association, 2014.

[8] Y. Liu, F. Xu, J. Chai, X. Tong, L. Wang, and Q. Huo. Video-audio driven real-time facial animation. *ACM Trans. Graph.*, 34(6):182:1–182:10, Oct. 2015.

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

[10] Wikipedia. Ekman's facial action coding system — Wikipedia, The Free Encyclopedia, 2016.