Face AR: Real-Time Blendshape Manipulation with Facial Landmark Detection from 2D Video

Supratik Banerjee
18309796
sbanerje@tcd.ie
YouTube: https://youtu.be/vk3gXGg-_7c
GitHub: https://github.com/supratikbanerjee/FaceAR/

Abstract

This research focuses on implementing features similar to ARKit's Face Tracking System which is used for blendshape manipulation in Unity3D and Unreal Engine. Instead of directly implementing it for a Game Engine, this work demonstrates the core technology. ARKit's FaceAR uses iPhone X's TrueDepth Camera to map the user's facial expressions to the blendshapes. In this work, an alternate solution is proposed, where we take 2D video feed from a regular camera, detect facial landmarks and use those points to calculate blend weights to manipulate the blendshapes. Apart from mirroring the facial expression, the system also uses these landmark points to calculate the facial rotation transform about all three axis.

Introduction

With high availability of RGBD cameras such as Microsoft's Xbox Kinect and Apple's TrueDepth camera in iPhone X and above, performance-driven facial animation, an integral part of film and game production, is now readily available at consumer level. However video cameras are more widely available and video-based facial animation are still a big challenge. Even though the technology is evolving towards increasing the availability of consumer grade RGBD cameras, these cameras are relatively more expensive than regular video cameras. This leaves room for further work and development of facial animation solution using 2D video inputs.

The potential for performance-driven facial animation with inputs from video camera has been well exploited in the industry. Several companies such as Faceware Technologies offer softwares to accomplish the task, but no free software is available for small studios to use. The only free solution currently available at consumer level is ARKit's FaceAR for which one needs to buy an expensive iPhone. This was the research's primary motivation, to introduce an open source proof of concept solution to this technology.

The idea of this research is inspired by the way marker based facial animation is achieved and to do the same without having to use any physical markers. The landmarks detected by Regressing Local Binary Features (LBF) [1] are used as virtual facial feature markers augmented on the input feed. Based on these markers a simple solution is proposed where the distance of each marker is compared with its neighbouring markers to derive the facial movement relations.

Background

Performance-driven facial capture have a long history in computer vision and graphics. Expression Retargeting is referred to the technique of transferring the facial expressions of an actor onto a 3D model with different facial proportions compared to the actors. This is achieved by the use of blend-shapes which have all the target positions of the 3D model. The performance actor's facial movements are then mapped with the blend-shapes by calculating the difference from the neutral face and target face. There are various facial capture techniques which can be classified as either marker-based or marker-less techniques. In the film and game industry marker based techniques are heavily used due to their reliability in tracking and low computational cost. With advent of newer RGBD camera technology markerless techniques are gaining popularity, as they free the actors from a cumbersome setup process.

The only currently known free solution for real-time facial mocap software's like Faceshift and Faceware's is ARKit's FaceAR implemented in Unity3D and Unreal Engine. The facial capture system with ARkit uses Apple's TrueDepth camera to track the user's facial motion. It compares the the current input pose of the user's face against 51 individual face poses native to ARkit's SDK. Each pose in the SDK targets a specific portion of the face. As a given part of the user's face approaches the shape of a final pose, the value of that pose blends between 0.0 and 1.0 For example, if the user closes their left eye, the LeftEyeBlink pose would blend from 0.0 to 1.0. In Unreal Engine [2], the 51 blend faces of the SDK are taken as input and these values drive the 3D model's motion. To accomplish this, the blendshapes have to be named as per the naming conventions of ARKit. If the naming convention isn't the same then a remapping system has to be used to match the ARkit meshes.

Implementation

The system is implemented in C++ using OpenGL to render the blendshape interpolations and OpenCV to track face and detect facial feature landmarks. The system comprises of three primary components: Facial Blendshape Animation, Landmark to Model and Calibration System.

Facial Blendshape Animation: This module comprises of rendering system which receives blend weight values from Landmark to Model module and interpolates between the delta blendshapes. For this implementation the blendshape models have been used from the mery project [3]. It takes the neutral face and seven other blendshapes: "left_brow_raise", "right_brow_raise", "jaw_open", "left_smile", "right_smile", "left_eye_closed", "right_eye_closed". With these target faces and the neutral face we compute the delta blendshapes **Fig.1** as follows:

$$\mathbf{f} = \mathbf{b}_0 + \sum_{k=1}^{n} w_k(\mathbf{b}_k - \mathbf{b}_0)$$

is denoted in matrix notation as:

$$\mathbf{f} = \mathbf{b}_0 + \mathbf{B}\mathbf{w}$$

Where $\mathbf{f}$ is the resulting face, $\mathbf{B}$ is the blendshape matrix whose column vectors, $\mathbf{b}_k$ are the blendshapes for each target position, $\mathbf{b}_0$ is the blendshape target of the neutral face, and $\mathbf{w}_k$ are their corresponding blend weights.

$$\begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} v_1^0 \\ \vdots \\ v_n^0 \end{bmatrix} + \begin{bmatrix} v_1^1 - v_1^0 & \dots & v_1^k - v_1^0 \\ \vdots & \ddots & \vdots \\ v_n^1 - v_n^0 & \dots & v_n^k - v_n^0 \end{bmatrix} \begin{bmatrix} W_1 \\ \vdots \\ W_k \end{bmatrix} \quad v = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

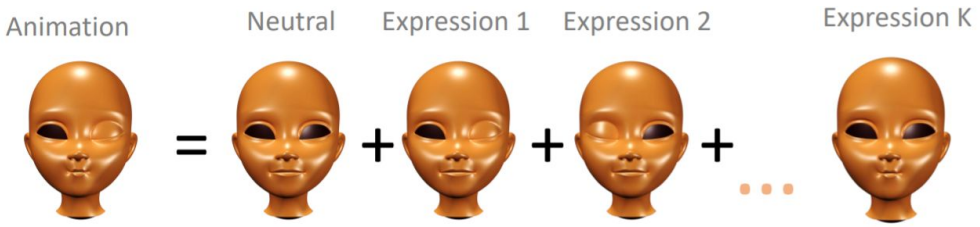

Fig.1: Final animation from two delta blendshapes Expression 1, K and Neutral Face

Landmark to Model: The Landmark to Model module first detects a face using LBP [4] Cascade. Followed by the detected face, LBF Landmark detection is used which gives us 2D coordinates of the landmarks on the face. Since these landmarks have significant amount of stability issue, a Simple Moving Average is used over five samples. This smoothes out the the marker positions at the cost of a small latency. The system has some predefined marker positions using which it computes the blend weights.

For example, to open or close the mouth, it takes the minimum and maximum points of the mouth markers (***marker1, marker2***) in the calibration step. Each marker is defined as a 2D point on the image. After taking these point values, the difference give the maximum range of mouth opening.

marker1 = upper lip
marker2 = lower lip
range = max(marker2 - marker1) - min(marker2 - marker 1)

Now we compute the current face's difference between the marker (***new_marker1, new_marker2***) positions and take the difference between this value and the minimum value recorded. Dividing this with the range gives us the weight for the mouth opening blendshape:

new_marker1 = upper lip
new_marker2 = lower lip
weight = [(new_marker2 - new_marker1) - min(marker2 - marker 1)] / range

The markers from OpenCV's LBF Landmark detector has fixed indexes for each feature **Fig.2**. For the mouth opening and closing we use the marker 63 and 67 (62, 66 index wise). Similarly, there are implementations for Left Eyebrow Raise, Right Eyebrow Raise, Left Smile, Right Smile, Left Eye Close and Right Eye Close which are defined by an integer 2D array of marker pairs:

int face_pairs[7][2] = {{33, 19}, {33, 24}, {66, 62}, {8, 48}, {8, 54}, {37, 41}, {44, 46}};

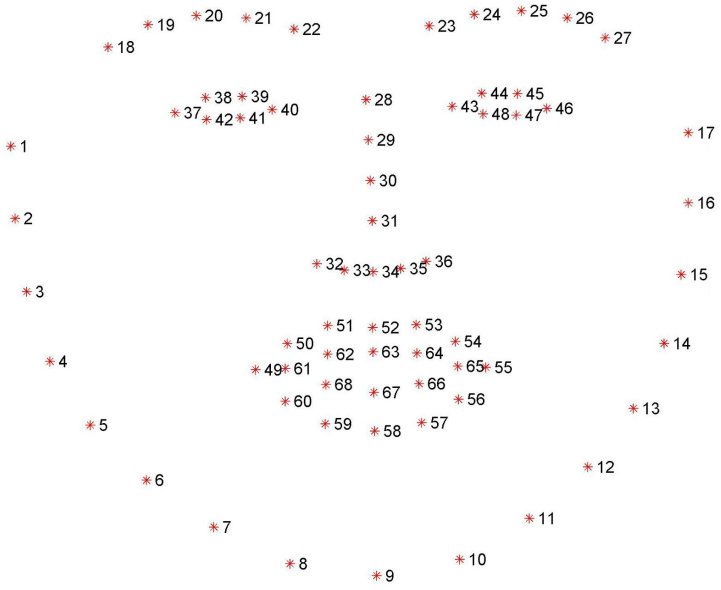

Fig.2: Indexes of facial feature landmarks (starting 1)

The landmark detection provided by OpenCV scales according to the distance of the face from the camera. This leads to a problem where the positions of the markers change thus rendering the max min record during calibration useless. To avoid this, the equation proposed above is slightly altered, where there is a division with the width of the head (Point 1-17) to make the measurements scale invariant.

weight = [((new_marker2 - new_marker1) / head_width) - min(marker2 - marker 1)]/range

It is also important for the system to capture the rotational transform of the head. For this, the system uses two methods. It captures the rotation along X and Y-axis based on the distances of marker positions. For rotation about X-axis, we check the distance between the bottom of the nose (34) and the bottom of the nose bridge (31). As the head moves up the distance increases and the other way as it moves down. This difference is mapped into a rotation angle in radian to tilt the head up or down. Similarly we rotate the head about Y-axis by check the distance between the left side of the jaw (3) and the bottom of the nose bridge (31), which enables rotation from left to right. To compute rotation about Z-axis, we take an arbitrary vector parallel to the X-axis and Vector from point (17) to (16) which is parallel to Y-Axis and calculate the angle between the two. This angle is then mapped to radians as the Left-Right head tilt motion.

Calibration: This module has a calibration sample limit, which dictates the length of the calibration process. During this process, the actor is required to perform random facial movements stretching all facial muscles to their maximum limits so that the calibration can identify the neutral face and the max target range. Once the limit of sample collection has been reached, it takes in all the landmark pair distances smoothened using simple moving averages over 5 samples as a stream and computes the minimum and maximum distances for each blendshape.

The landmark pair distances is computed using the array **face_pairs** presented above as follows:

$$sma\_stream = [sma(face\_pairs_{k,0}) - sma(face\_pairs_{k,1})] / head\_width$$

Here, for **k** blendshapes, we compute the simple moving average for every marker pair over 5 samples and take their difference. This process takes place **N** times collecting **Nk** samples of SMA difference where **N** is the calibration sample limit. Using this **sma_stream**, which is basically a list of **Nk** SMA pair difference we compute the minimum and maximum distance of each blendshape pair and store in a 2D array (**min_max**). This computed **min_max** array is then returned to the Landmark to Model module to compute the blend weights.

Experiments

The biggest challenge and most time consuming part of experimentation was figuring out which facial landmark points would be most appropriate to get the blendshape accuracy as high as possible. This was primarily because, it needed time to experiment with all manually decided points to observe them over different facial movements and how much does any particular movement alter or affect other parts movements.

The first iteration of the implementation did not use simple moving average filter, which lead to very jerky animations as the landmarks aren't stable and keep flickering causing the blendshapes to move randomly. This was first solved using a three sample simple moving average, which was better bot still jerky. As the sample size was increased the performance significantly started dropping. Experimenting with different sample sizes lead to seeing the best results with a five sample moving average. The next hurdle came along with the drop in framerate due to Haar cascade classifier. It is relatively expensive for this application even on an i7-6700HQ. This lead to the switch from Haar to LBP cascade classifier. Even though there is a slight reduction in accuracy, it brought up the performance significantly from 13 FPS up to 30 FPS.

Results

The results obtained by the implementation are tested as per visual aesthetics. The system was tested on multiple faces. For each face the setup time is less than a half a minute for calibration. Even though the blendshapes are rendered at about 30 FPS there is a slight delay in the input to out due to the 5-SMA making the quality look slightly jittery. The expressions calculated using 5-SMA are visually pleasing, where as the rotations don't work very well with the same. The head looks slightly wobbly even when the actor stays still. Due to the way rotations are mapped, sometime dramatic changes in some facial movements trigger rotations about the X-Axis.

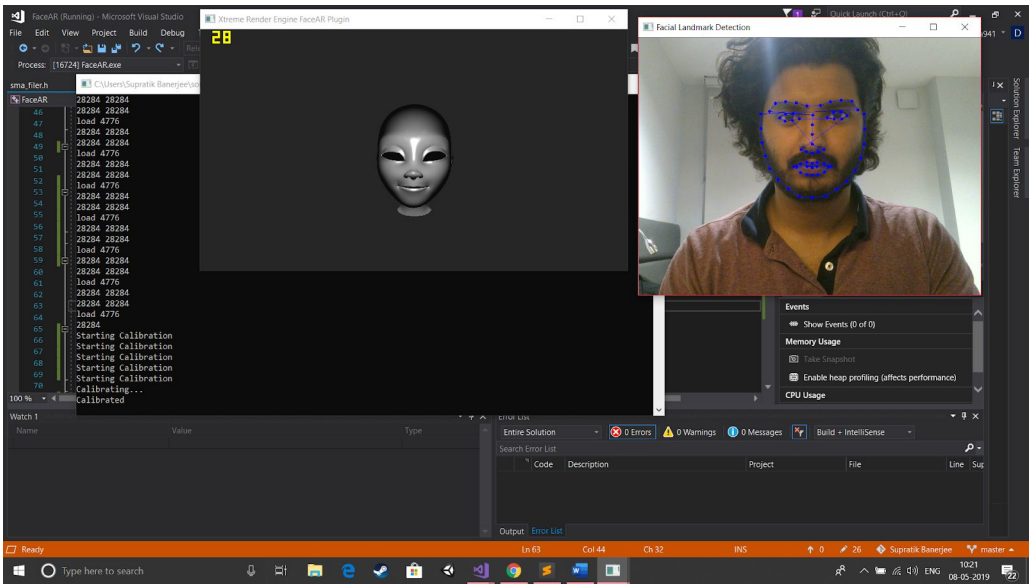

Fig.3: Neutral Face 1

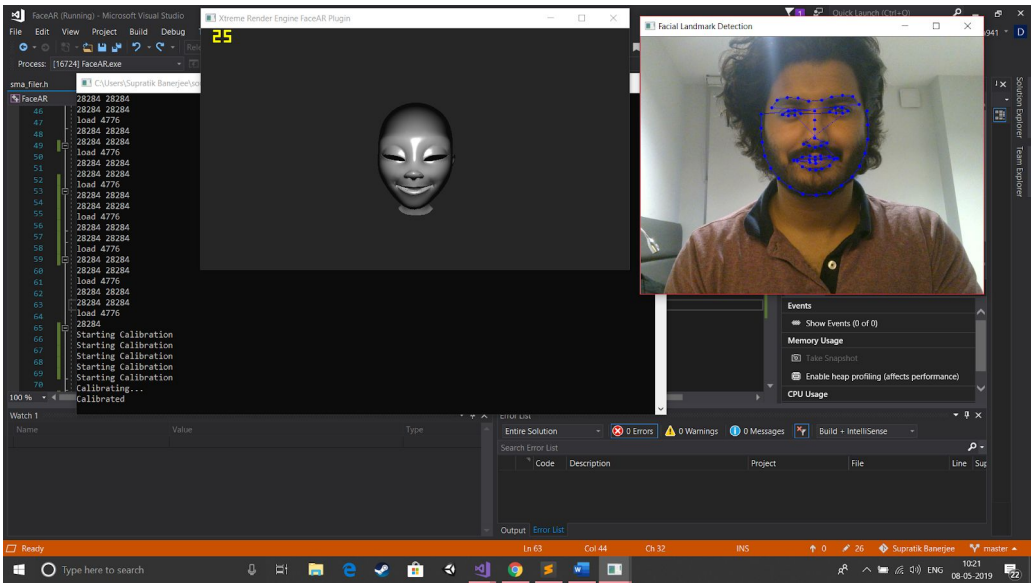

Fig.4: Smile Face 1

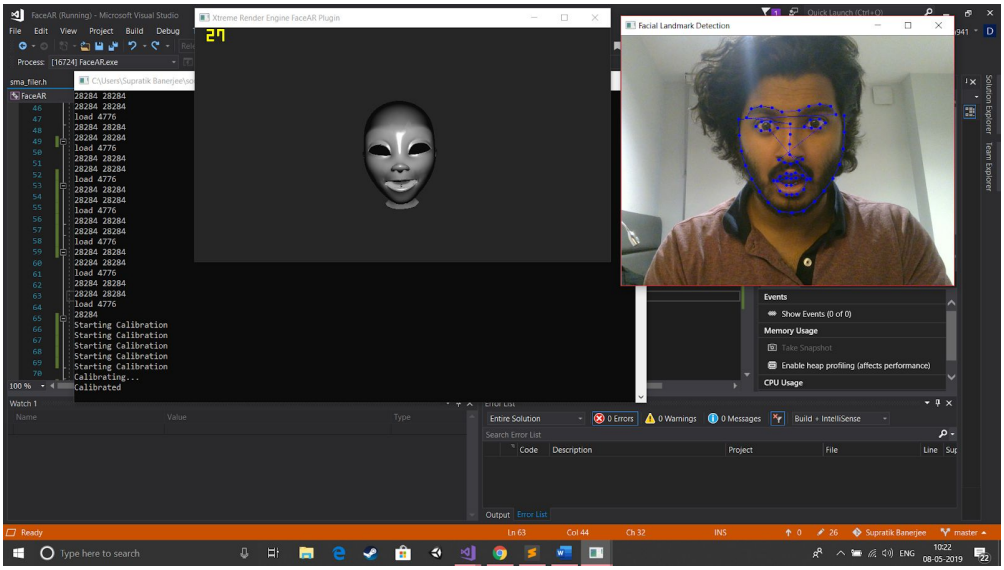

Fig.5: Surprised Face 1

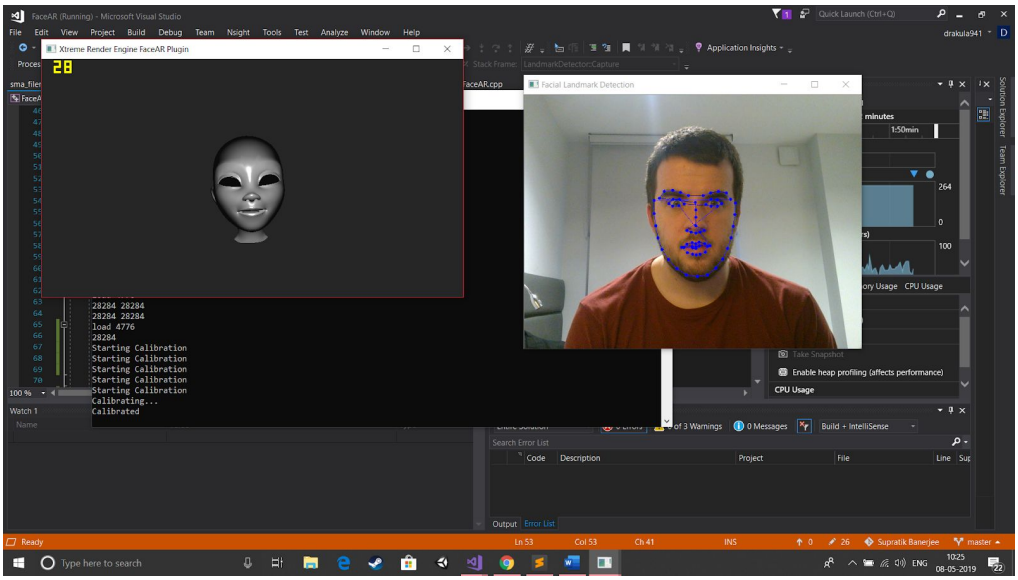

Fig.6: Neutral Face 2

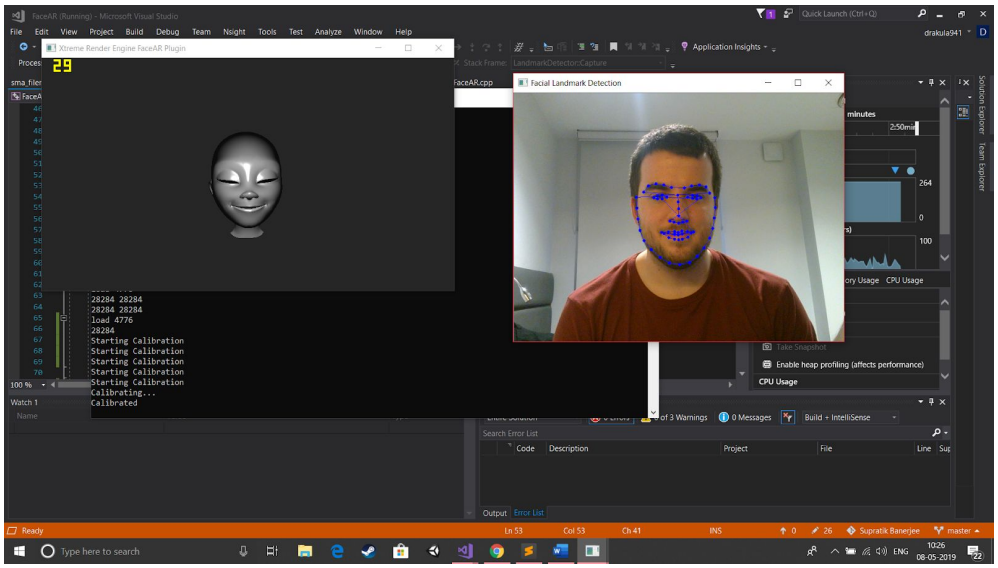

Fig.7: Smile Face 2

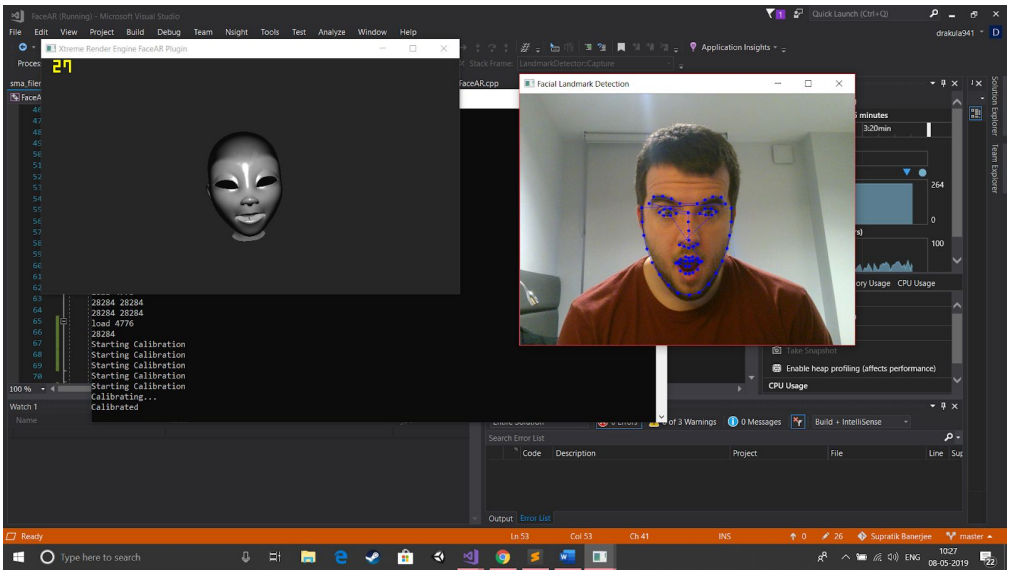

Fig.8: Surprised Face 2

Discussion

The proposed system implementation originated more out of an intuition rather than literature study. I still wanted to explore existing publication [5] to accomplish the task. But due to the difficulty of acquiring the training data and the complexity of implementation as the paper was derived from multiple other publications, seemed over ambitious in the given time. Hence I resorted to implementing my own idea. While looking for literature on my own idea, I connected with another developer on YouTube [6] who is working with a very similar idea and motivation using OpenCV and Unity3D. I am extending this project as a plugin to my own rendering engine that I have been working on, which evolved out of my assignments in the module Real-Time Rendering.

Conclusion

A complete system for markerless facial capture and animation was implemented which works closely similar to ARKit's FaceAR, but with 2D video input. This technology if advanced properly can help many independent studios who focus on making small budget to no budgets films and games. With the implementation presented in this paper and contributions by other talented developers like Fabrizio Zagaglia, who is taking this area of research and implementing in his own studio, It doesn't seem very far in the future when we will see a lot of robust and high quality open source Facial Mocap systems available online accessible to everyone.

Future Work

The stability of the system can still be significantly improved with other methods to handle the model rotations. A lot more blendshapes can be added for greater detailed animations. Instead of using traditional computer vision techniques for face detection and landmark detection we can try Neural Networks to achieve the same. The depth of the face can also probably be simulated, by taking some average value measured manually across multiple faces to find the relative depth of all the marker positions. This could lead to faking 3D with 2D video.

References

[1] Ren, S., Cao, X., Wei, Y. and Sun, J. (2016). Face Alignment via Regressing Local Binary Features. IEEE Transactions on Image Processing, 25(3), pp.1233-1245.

[2] Unreal Engine, "Face AR", https://docs.unrealengine.com/en-us/Platforms/AR/HandheldAR/FaceARSample

[3] Mery Project, "Free Characters for Animators", https://www.meryproject.com/

[4] Ojala, T., Pietikainen, M. and Maenpaa, T. (2002). Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(7), pp.971-987.

[5] Cao, C., Hou, Q. and Zhou, K. (2014). Displaced dynamic expression regression for real-time facial tracking and animation. ACM Transactions on Graphics, 33(4), pp.1-10.

[6] Fabrizio Zagaglia, Face Capture System https://www.z4g0.info/index.php?path=tech/libs/facecapture