Facial Expression Capturing with ARKit and Facial Animation Recreation Using Blender and Unity

A Final Project Report for CPSC 8110: Technical Character Animation

Hanjie "Hollis" Liu
School of Computing
Clemson University
Clemson, SC, USA
hanjiel@clemson.edu

ABSTRACT

In this report, I explore the possibility of facial expression capture on hand-held devices such as the iPhone. The main difference between the work this paper lays out and current state of the art facial animation solutions such as Apple's Animoji is that this paper is merely using the iPhone as a facial data capturing device rather than the host for facial animation APIs. As a result, the captured data can be flexibly used on any platform for any purpose while maintaining the convenience of doing a capturing session on hand-held devices.

The main goal of this paper is to explain the pipeline the author built that captures human facial expression using the iPhone X and recreates those facial expression on a character model in Unity using the captured data. In addition, the author will touch on some of the design choices he made to facilitate the pipeline.

KEYWORDS

Facial tracking, iPhone, iOS, ARKit, Blender, Unity

1 INTRODUCTION

Human facial expressions are incredibly complex and versatile, thus capturing those minute details can be very challenging. There are many published research papers on the topics of facial capturing that ranges from real time tracking to high fidelity offline tracking/rendering.

The tracking solution this paper offers is unique in that it leverages the sensors and computing power on a daily device that is used by millions of people. While the detail of facial tracking lacks in quality compares to some other tracking methods, it nonetheless provides an acceptable tracking result that can be used in many circumstances.

The iPhone X is the capturing device used in this paper. Apple announced this device with the TrueDepth camera [Apple] that is capable of tracking facial data in real time in 2017. Later in 2018, Apple announced the software APIs (namely, ARKit) that developers can take advantage of to extract those facial tracking data.

Multiple facial tracking sessions were done on the author of this paper. There is no addition hardware requirement so the setup is minimal and convenient. Custom software is written to support the capturing needs.

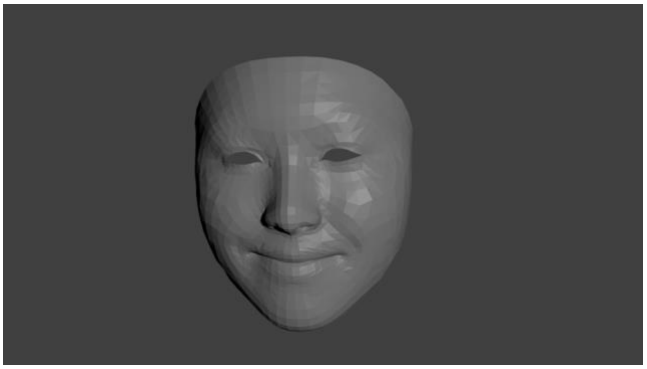

Figure 1: Blender render of a raw mesh frame captured using the solution proposed by this paper

There were three main challenges that this paper tackled to complete the proposed facial tracking solution: 1) capture and serialize huge amount of real time facial data. 2) migrate and parse the data from the device to a computer. 3) map the captured data onto an Unity model to recreate facial animation.

2 RELATED WORK

In addition to the custom software that was written to support the capturing session itself, there are many software dependencies that are needed for the solution. They mainly serve the purpose of data parsing, model rigging and animating. Those software requirements are listed in this section.

Xcode, along with the Swift programing language and ARKit is the software development environment for the facial capturing

app. The custom application written is built on top of the sample app that was provided by Apple. [Apple, 2018]

Python language is used for data parsing on recorded meshes that are serialized in XML format.

Blender is used for rendering all raw facial mesh and rigging the human face model named "Anakin head 3D Model" that was acquired for free. [Korneev]

Unity is used for importing the rigged face model and producing the final facial animation based on the recorded facial data file.

3 CAPTURING PIPELINE

3.1 The Capturing App

The custom app adds a "start session/stop" button to the provided sample app that signals the start and finish of a capture session. The app records all the vertices and indices of a facial mesh at a reduced 24 frames per second frame rate in order to lower file sizes while preserving essential animation details.

Because the indices return by ARKit is zero indexed while common OBJ files for 3D model description is one indexed, every index data point needs to be offset by one to reconstruct a well-defined mesh.

Vertex and index data for every frame are store in two arrays that are later serialize into two XML files that preserves the data structure. The files are stored in the "documents" directory of the app so it can be easily exposed to the "Files" app on iOS. The AirDrop sharing protocol is used to transfer those facial data files to a computer.

3.2 Data Parsing

Because vertex and index data are separated in different files that contain all frames, data parsing is necessary to extract data for each frame and generate an OBJ file for each frame. The Python script shown in figure 2 is written to parse those data files.

```
from xml.etree import ElementTree
root = ElementTree.parse("face_vertex.txt").getroot()[0]
index_root = ElementTree.parse("face_index.txt").getroot()[0].findall("array")

for frame, f in enumerate(root.findall("array")):
    out_file = open("result/combo_" + str(frame).zfill(3) + ".obj", "w")

    for v in f.findall("array"):
        r = v.findall("real")
        out_file.write('v {} {} {}\n'.format(r[0].text, r[1].text, r[2].text))

    for face in index_root[frame]:
        r = face.findall("integer")
        out_file.write('f {} {} {}\n'.format(r[0].text, r[1].text, r[2].text))

    out file.close()
```

Figure 2: Python code for parsing the facial mesh XML files

Once the OBJ files for each frame are acquired. They can be used for batch render in Blender for some preliminary results. Figure 1 shows one frame that was captured when the author was smiling. A video is generated by stiching up all frames in order using FFmpeg. The video suggests that there is some lag in the captured animation compared to the original video but the overall facial expression is greatly preserved.

4 ANIMATION RECREATION

4.1 Model Rigging

Due to the fact that there were no freely available and easy to use facial rigged model that the author could find online, a rigging process is required for recreating the facial animation.

The Rigging process is done in Blender on the Anakin model. Shape keys and the sculpting tool were used to complete the rigging process. The rigged blender model can be seamlessly imported to Unity as shown in figure 3.

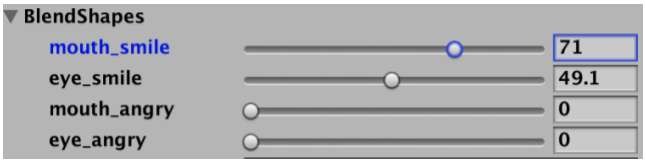

Figure 3: Imported rig controls in Unity as blend shapes

Four blend shapes are used to recreate happy and angry expressions.

4.2 Feature Mark Selection

It is unnecessary and expensive to use all vertices in the facial mesh to recreate animation in Unity since blend shape is used. Thus it becomes important to select certain feature mark vertices to represent the overall facial motion.

Since the returned meshes always have the same amount of vertices and indices, it is safe to assume that a vertex at a given index position will always represent the same point in a facial structure.

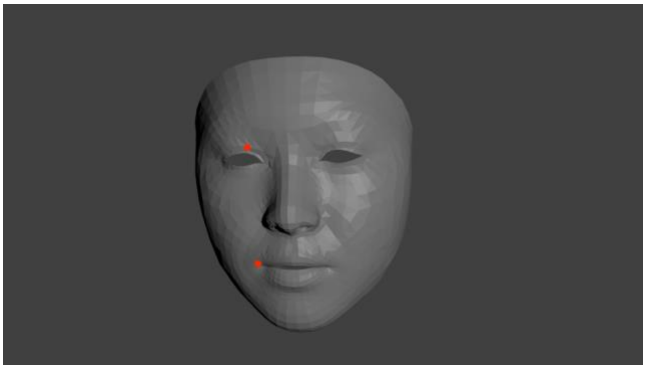

Figure 4: Two feature vertices marked with red dots

As figure 4 shows, a vertex at the tip of the lips and another vertex at the top of the eye lid are selected as feature marks. The reasoning is that those two vertices offer more representative motion data than others.

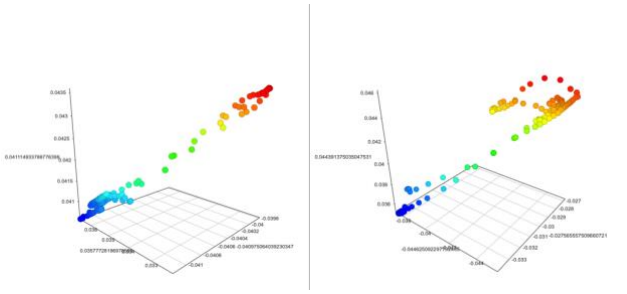

Figure 5: Plot of selected vertex positions across all captured meshes (left: eye vertex, right: mouth vertex)

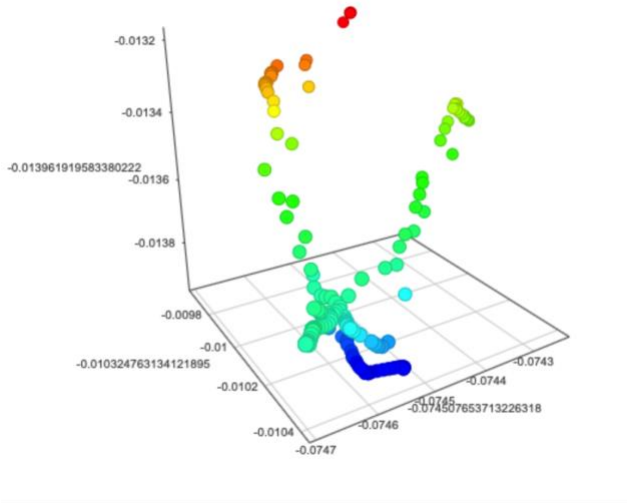

Figure 6: Plot of vertex positions across all captured meshes for a non-selected vertex that is at the edge of the facial mesh

By comparing figure 5 and figure 6, it is not difficult to see that the selected feature mark vertices offer better separated and more linear data while the non-selected edge vertex fails to offer a clear data pattern.

4.3 Unity Animation

Unity animation is the last stage of the animating process. The Anakin model is textured and smooth shaded to look more realistic as shown in figure 7.

A C# script is written to read in the feature mark vertex data files and animate the blend shapes based on the Euclidian distance of each vertex to the first vertex in each file (base vertex, representing resting stage). A sign is calculated for each vertex by comparing the Y-axis value for each vertex with the base vertex to determine if that region of the face should move upwards or downwards. Linear scaling for those distance value is also necessary so that the data range is roughly 100 (percent).

Figure 7: Rigged model rendered in Unity

5 RESULTS

The result video shows the final render mostly matches the original facial expression video recording as well as the raw facial mesh renders.

Ideally, since all capturing session and rendering are kept at 24 frames per second, the total run time of animations should match up exactly. However, some lag in both animation renders were observed.

6 DISCUSSION

It is certainly great to see the final result works as proposed. The animation greatly matches the facial expression captured on the original device. Considering the whole capturing session is done on a battery powered phone, the overall result is satisfactory. However there are some significant limitations within this solution as well

The capturing precision is somewhat limited by the sensor resolution and computing power of the phone. As a result there are tiny muscle movements that can never be picked up by the phone.

The animation recreation process has limited blend shapes so it doesn't support expressions such as opening the mouth. Besides, The final animation does not look very natural because the rigging process was crude and only supports symmetrical rigging.

Some potential future works include adding more blend shapes, using more advanced hardware and acquiring better rigging models either by purchasing or working with artists.

Overall, the future in mobile real-time facial capturing is bright because it enables users to play, interact and communicate in such rich and powerful ways on the palm of their hands.

REFERENCES

[1] Apple (2019). Advanced camera system Say hello to a new era of photography. https://www.apple.com/iphone-xr/cameras/

[2] Apple (2018). Creating Face-Based AR Experiences. https://developer.apple.com/documentation/arkit/creating_face-based_ar_experiences.

[3] Korneev, Valery. Anakin head 3D Model. https://archive3d.net/?a=download&id=440a1067