BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2022
http://excel.fit.vutbr.cz

Animation of Avatar Face Based on Human Face Video

Martin Takács*

Abstract
This paper presents an application for animating a 3D avatar based on a single camera or video input of the human face in real time. The resulting application consists of three modules — face tracking, avatar animator, and a server for transferring face data. The input frame from the camera or video is processed and data are sent to the server from where the data can be sent over to multiple avatar animators so it is possible to animate multiple avatars based on a single input.
Face tracking works in real time (30+ FPS based on camera and device on which it runs) and the avatar animator runs in a web browser so no additional installation is required.
Animating 3D avatars based on human face is becoming more and more popular thanks to content creators who do not share their real face, but still want to interact with their audience or act as a virtual character on streaming platforms such as YouTube or Twitch.

Keywords: Face Tracking — 3D Character — Face Animation

Supplementary Material: Demonstration Video — Downloadable Code

*xtakac07@fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Online content creators on streaming platforms (Youtube, Twitch) present themselves behind virtual avatars to approximate the in-game-like feeling for their audience. The audience does not see the real person, but rather an real-time animated avatar based on the human face input – to protect the author's privacy.

This project aims to simplify the work of aforementioned content creators or users. The users just need to run the system and they obtain three modules, which run independently, and can choose among multiple avatars. They can run the avatar animator module in multiple windows to animate multiple avatars at the same time. As the avatar animator module runs in a web browser, no additional installation is required. Therefore, the only requirement for the proper run of the project is a camera connected to the computer.

Several different technological solutions exist for face tracking as well as loading and animating avatars. Tensorflow.js and Mediapipe are runtimes used for face tracking on desktops and on mobile devices. Both runtimes were compared in [1] using pose detection. Results have revealed, that Tensorflow.js runs significantly slower on desktop and Android devices. It has surpassed Mediapipe only on iPhones.
Traditional solutions for estimating head position require additional hardware, such as a head-wearable marker structure. This approach is explained in [2].
Not all methods for animating avatars use standard 3D models. One of the exceptions is a Live2D model, which is slightly more popular than 3D, because it gives the impression of a moving picture. As explained

in [3], Live2D is a 2D image that is animated without requiring any changes. However, 3D formats, such as VRM[1], FBX or GLTF are more practical, since they offer more variability and scalability.

My solution allows users to use any rigged 3D avatar in the GLTF file format since this file format is hierarchical and very suitable for deforming bones of armature. As mentioned before the project consists of three modules. Two of which are python scripts (face tracking and websocket server) and the last one is a simple HTML page and can run in most of the web browsers (avatar animator). The face tracking and websocket server can have only one instance running at a time unless the ports are manually changed, while the avatar animator can have as many instances running as the hardware allows. The user first needs to run the websocket server. Then the face tracking and the avatar animator modules will automatically connect. The websocket server listens to the face tracking module which sends data about human face from camera input and then the websocket server distributes those data to every avatar animator connected.

2. Face Tracking

The core part of the project is to correctly recognize and capture human face in the video input, whether it is from the camera or video file.

To do this, I use python OpenCV package to read video input frame after frame. It is possible to choose among multiple connected cameras (0 is default index) or specify the path to the video file, in which I want to capture a human face.

Frames are processed consecutively. Firstly the image is flipped horizontally and color space is converted from BGR to RGB. After this, the image is proecessed to capture the human face and shown in a separate window as a frame from camera or video and human face annotations as shown in Figure 1.

To process the human face in the image, I use Google's open-source framework for media processing – MediaPipe. The framework is distinctly explained in [4].

As explained in [5], FaceMesh is a solution that estimates facial surface geometry – 468 (refined 478) landmarks in real time. Landmarks are vectors in 3D space that make it easier to calculate transforms for the avatar. As shown in Figure 2, in my project, not all 478 landmarks are necessary, because avatars has a very limited number of bones that could be deformed. It is enough to take only the most important ones, such

[1] VRM specification could be found here.

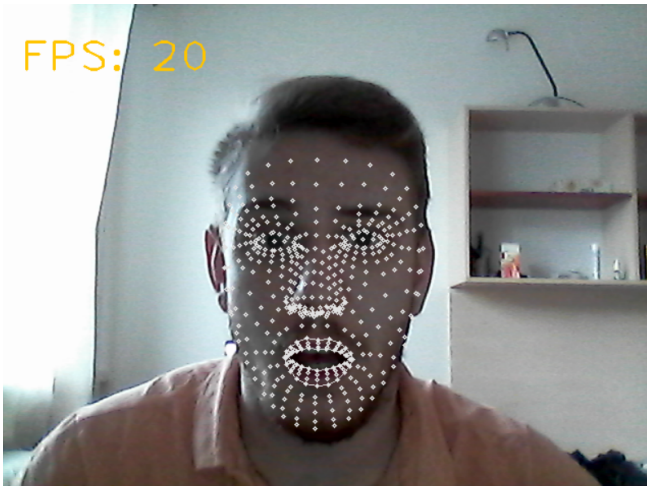

Figure 1. Processed face and drawn face landmarks.

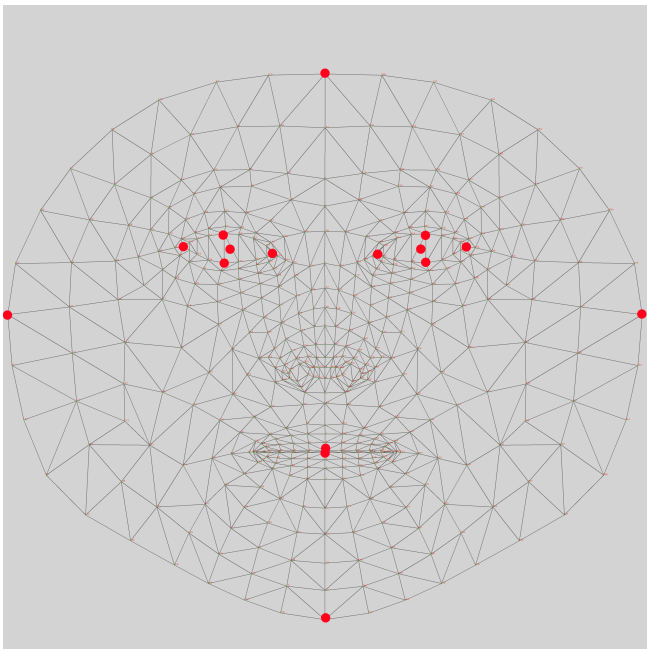

Figure 2. Highlighted landmarks - only few landmarks out of total 478 are actually important for my solution. The original image of face geometry is from google's mediapipe repository.

as the very top, very bottom and sides of the face, lips, eyes, irises, eyelids and nose.

From obtained landmarks (vectors) I can easily calculate how much is the mouth opened, if the right or left eye blinked, head rotations (rotation, nod, turn) and, as shown in Figure 3, the direction in which are eyes looking. Most calculations are done through calculating an euclidean distance between two points except for the nodding and the turn of the head. The nodding and turning of the head are calculated through the rotation matrix which gives much better and more precise result than the simple euclidean distance between two points. As explained in [6], three euler angles could be obtained from the rotation matrix.

Finally, calculated parameters are sent to the web-

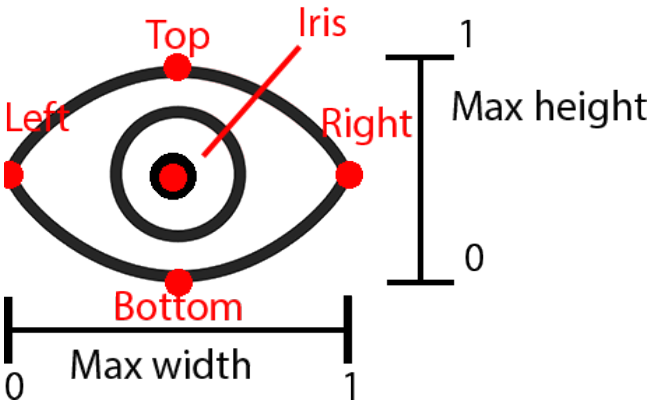

Figure 3. Calculating position of the iris inside the eye. Co-ordinates are [0, 0] - bottom left, [1, 1] - top right.

socket server as a message in the JSON format. The websocket server then broadcasts that message to all avatar animators connected to the server as will be explained later in Section 4.

3. Avatar animator

When data from the real world has been obtained it is possible to animate the virtual avatar based on those data. This is done right in the user's browser so no additional application is required and multiple tabs with different avatars can be opened at the same time.

Before animating the avatar, the Animator needs to connect to the websocket server from where it listens to incoming messages that contain important data for animating the avatar. Avatar animator never sends messages back to the server, only listens to it.

Each avatar needs to be rigged before use, but no animations are required as the avatar will be animated in real time based on the real world data. Avatar also needs to have named bones and has to be exported in GLTF file format, which allows for easy traversing of the whole avatar.

To load the GLTF model, create a scene, render it and to animate I use module ThreeJS.As mentioned in [7], ThreeJS is very suitable for skeletal animations. After the loading of a GLTF file - an avatar, it is added to the center of the scene.
ThreeJS allows for the setting of the lighting, background. These settings can be dynamically changed via GUI overlay.
ThreeJS also allows for the camera control, so the user is able to rotate around the avatar, zoom in and zoom out.

Each model can have different properties such as the size or direction of bones. For users to be able to continuously switch among different avatars, each model needs a configuration file with bones names,

```
{
    "bones" : {
        "bone_jaw": "Quijada",
        "bone_head": "Cabeza",
        "bone_eye_L": "Ojo_Control",
        "bone_eye_R": "Ojo_Control_2",

        "multipliers": {
            "jaw":-4,
            "head_rot":1,
            "head_nod":1,
            "head_turn":1
        },

        "offsets": {
            "head_rot": 0,
            "head_nod": 0,
            "head_turn": 0
        }
    },
    "scale_factor" : 3
}
```

Figure 4. Example of a configuration file for the avatar.

```
// Jaw
if (bone_jaw != null){
    scene.getObjectByName(bone_jaw).
        rotation.x = base_jaw_rot + msg.gap
         * jaw_mul
}
```

Figure 5. Example of updating bone transforms of the jaw.

multipliers and offsets for bones, as shown in Figure 4. Scale multiplier defines how much the model should be scaled for proper . This configuration file is loaded before the actual model.

After loading the model, scene is configured with lights, background, and a controllable camera. To switch among avatars, I use a simple HTML form with the dropdown menu where option values are paths to different GLTF files.

With properly loaded object and obtained real world data, the scene is set and the avatar ready to be animated. To do this, I find a bone, which name corresponds with the name given in the configuration file and change its rotation around the axis where necessary.

Figure 5 shows the example of code to update a particular bone transform with newly obtained data, where the *bone_jaw* is the name of the bone deforming the jaw, *base_jaw_rot* is the default rotation of jaw right after the model is loaded, *jaw_mul* is multiplier specified in the configuration file of the model and *msg.gap* are data obtained from the websocket server.

The similar approach is used for bones of head,

Figure 6. Avatar after applying new transforms for bones.

eyes and eyelids leaving the impression that the avatar is really moving alongside the user in the real world or in the video. Figure 6 shows the moving avatar. Its head is slightly rotated, mouth is opened and eyes are looking to the right side.

4. Websocket server

With fully functional Face tracking and Avatar animator, their only problem is that they are completely independent modules. However, this might be an advantage depending on the point of view. Running Avatar animator independently from Face tracking allows users to run multiple instances of Avatar animator at the same time and animate them based on the same input data from Face tracking.

However, the intermediator, the middle man is required to transfer data correctly from the Face tracking module to the Avatar animator. That means bidirectional communication is required to both receive data and send data. As explained in [8], the websocket server allows this.

The websocket server registers all clients connected to it and then broadcasts messages to all registered clients. Even Face tracking module is a registered client, but it does not handle incoming messages and it only works one way (from Face tracking module to the websocket server) as shown in Figure 7.

The Avatar animator works the other way around. It only listens to messages from the websocket server and never sends data back. That would create unnecessary loops and those kinds of messages would need to be filtered.

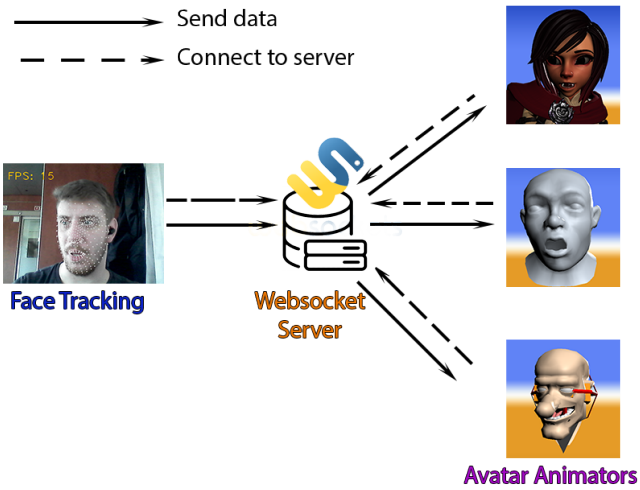

Figure 7. Websocket server recieves data from Face tracking and distributes them to Avatar animators.

5. Evaluation

Three modules were tested on Windows 10 device, using AMD Radeon RX 480 GPU, Intel Core i7-9700F CPU and two different cameras: Genius WideCam F100 and Xiaomi Redmi Note 9 Pro's front camera connected via Iriun webcam software[2].

Both cameras performed similarly, their FPS matched at 30 according to the FPS counter. However, since Genius camera has wide angle (120°), the face appears to be smaller in the input image, distance between landmarks is also smaller and thus making results of calculations less accurate.

The system is intended to work with a single camera, however, it is possible to have multiple cameras connected to the websocket server. This creates undesired behaviour of avatars, because data from both face tracking modules are broadcasted to all connected avatar animators.

As mentioned above, the biggest advantage of this solution in comparison to other existing solutions is the ability to animate multiple avatars simultaneously. This ability proved in tests[3] to be working as expected (60+ FPS render speed).

6. Conclusions

I created the system that takes human face video input from a camera or video file and calculates important transforms from face landmarks. Based on those transforms, the system animates the avatar in real time. It is possible to animate multiple avatars at the same time.

The system captures face and animates the avatar in 30+ FPS depending on the hardware used so the

[2] https://iriun.com/

[3] Test showing both camera FPS and rendering FPS, while rendering 7 avatars - https://imgur.com/UwGXeRq.

movement is smooth for the human eye.

Captured face returns landmarks that can jiggle if the camera is not decent enough which creates a shaking impression of the avatar. Future work is needed to smooth those kinds of movements. Also, the Avatar animator is limited to GLTF files and it would be advantageous to extend the loading system to accept other file formats such as FBX or VRM.

Acknowledgements

I would like to thank my supervisor prof. Ing. Herout Adam, Ph.D. for his help, advice, comments, and guidance in the course of making this project.

References

[1] Ivan Grishchenko et al. 3d pose detection with mediapipe blazepose ghum and tensorflow.js. [online], 2021.

[2] Jeroen Lichtenauer and Maja Pantic. Monocular omnidirectional head motion capture in the visible light spectrum. pages 430–436, 11 2011.

[3] Koichi Nakagawa et al. The use of live2d as an animation education tool. *Bulletin of Kurashiki University of Science and the Arts*, (22):15–21, 2017.

[4] Camillo Lugaresi et al. Mediapipe: A framework for building perception pipelines. *CoRR*, abs/1906.08172, 2019.

[5] Yury Kartynnik, Artsiom Ablavatski, Ivan Grishchenko, and Matthias Grundmann. Real-time facial surface geometry from monocular video on mobile gpus. *CoRR*, abs/1907.06724, 2019.

[6] Gregory G Slabaugh. Computing euler angles from a rotation matrix. *Retrieved on August*, 6(2000):39–63, 1999.

[7] Amit L. Ahire, Alun Evans, and Josep Blat. Animation on the web: A survey. In *Proceedings of the 20th International Conference on 3D Web Technology*, Web3D '15, page 249–257, New York, NY, USA, 2015. Association for Computing Machinery.

[8] Alexey Melnikov and Ian Fette. The WebSocket Protocol. RFC 6455, December 2011.