BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

ANIMATION OF AVATAR FACE BASED ON HUMAN FACE VIDEO

ANIMÁCIA TVÁRE AVATARA NA ZÁKLADE ZÁBEROV ĽUDSKEJ TVÁRE

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **MARTIN TAKÁCS**
AUTOR PRÁCE

SUPERVISOR **prof. Ing. ADAM HEROUT, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2022

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2021/2022

Zadání bakalářské práce

25040

Student:	**Takács Martin**
Program:	Informační technologie
Název:	**Animace tváře avatara na základě záběrů lidské tváře**
	Animation of Avatar Face Based on Human Face Video
Kategorie:	Počítačová grafika

Zadání:

1. Seznamte se s problematikou detekce lidské tváře a jejích detailních rysů v reálném čase.
2. Seznamte se s problematikou animace modelu lidské tváře (avatara).
3. Experimentujte s dostupnými technologiemi pro rozpoznání rysů tváře ve videu a pro animaci modelu lidské tváře.
4. Navrhněte systém pro animaci modelu lidské tváře (avatara) na základě pohybů reálného člověka ve videu.
5. Implementujte navržený systém a optimalizujte jej.
6. Experimentujte s možnostmi provozování vytvořeného systému na mobilním zařízení.
7. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Gary Bradski, Adrian Kaehler: Learning OpenCV; Computer Vision with the OpenCV Library, O'Reilly Media, 2008
- Richard Szeliski: Computer Vision: Algorithms and Applications, Springer, 2011
- Mark Segal, Kurt Akeley: The OpenGL Graphics System: A Specification (Version 4.6 (Core Profile) - October 22, 2019)
- série knih: Game Programming Gems
- série knih: GPU Gems
- Johnson J.A.: A Survey of Computer Graphics Facial Animation Methods: Comparing Traditional Approaches to Machine Learning Methods, MSc. thesis, California Polytechnic State University, San Luis Obispo, 2021

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3, značné rozpracování bodů 4 a 5.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu:	Černocký Jan, doc. Dr. Ing.
Datum zadání:	1. listopadu 2021
Datum odevzdání:	11. května 2022
Datum schválení:	1. listopadu 2021

Abstract

This thesis presents an application for animating 3D avatar based on a single camera or video input of human face in real time. The resulting application consists of three modules – face tracking, avatar animator, and a server for transferring face data. The face tracking module computes new transforms for the animation from human face and benefits from Mediapipe's Facemesh to estimatate the current face geometry. Avatar animator module is a web-based application for rendering and animating 3D avatars through skeletal animations, based on the Three.js library. Both modules make use of the continuous bidirectional communication of websockets through the single server. Performance of the face tracking module depends on the camera and device on which it is running, but regular web camera is usually enough for speed of 30+ FPS and animation runs at the speed of 60+ FPS with multiple avatars.

Main contributions of this project are (a) Calculating transforms from human face is suitable for the skeletal animations, which are usually easier to create and more accessible than blend shapes. (b) Using the face tracking and the avatar animator as independent modules causes that from the single camera/video input it is possible to animate multiple avatars on different devices.

Abstrakt

Táto bakálarska práca predstavuje aplikáciu pre animovanie 3D avatarov v reálnom čase, na základe záberov ľudskej tváre z jedinej kamery alebo videa. Výsledná aplikácia pozostáva z troch modulov – snímanie tváre, animátor avatara a sever, ktorý prenáša dáta z modulu na snímane tváre. Modul snímania tváre vypočítava z ľudskej tváre nové dáta pre animáciu a využíva pritom Facmesh of Mediapipe pre určovanie obrysov a črtov tváre. Animátor avatara je webová aplikácia pre vykreslovanie a animovanie 3D avatarov, pomocou kostrových animácií, pričom využíva knižnicu Three.js. Obidva moduly využívajú výhody súvislej obojsmernej komunikácie protokolu websockets, pripojením na jediný server. Výkon modulu na snímanie tváre záleží od kamery a zariadenia, na ktorom beží, ale bežne dostupná web kamera obvykle postačuje pre rýchlosť snímania 30+ FPS a animovanie beží na 60+ FPS s viacerými avatarmi.

Hlavným prínosom tejto práce sú (a) Počítanie dát z ľudskej tváre je vhodné pre kostrové animácie, ktoré sa väčšinou jednoduchšie na vytvorie a sú viac dostupné ako metóda splývania tvarov (angl. blend shapes). (b) Používanie snímania tváre a animátora avatara, ako nezávislých modulov spôsobuje to, že na základe jediného vstupu z kamery/videa je možné animovať viacerých avatarov na rôznych zariadeniach.

Keywords

face tracking, 3D character, face animation, skeletal animation, websockets

Kľúčové slová

snímanie tváre, 3D avatar, animácia tváre, kostrové animácie, websockets

Reference

TAKÁCS, Martin. *Animation of Avatar Face Based on Human Face Video.* Brno, 2022. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor prof. Ing. Adam Herout, Ph.D.

Animation of Avatar Face Based on Human Face Video

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of prof. Ing. Adam Herout, Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

. . . .
Martin Takács
May 10, 2022

Acknowledgements

I would like to express gratitude to my supervisor prof. Ing. Adam Herout, Ph.D. for his help, motivation, valuable and honest feedback, and for his patience with my questions that were sometimes even trivial.

Contents

List of Figures

Chapter 1

Introduction

This project aims to simplify work of online content creators (e.g. streamers) that want to hide their true identity, while maintaining the interactivity with their audience through virtual avatar. As the title suggests, its main purpose is to animate virtual avatar based on the video input from the camera or the video file, containing the human face in the image. To achieve this, the system makes use of the Mediapipe's perception pipeline and FaceMesh solution for estimating the face geometry from the ordinary web camera. Avatar animator takes advantage of skeletal animations to properly move only those parts of the avatar that need to be moved, right in the user's web browser. Both face tracking module and avatar animator module are interconnected through the single websocket server, which provides continuous bidirectional communication and can broadcast information from the single face tracking module to multiple avatar animators at the same time.

Hiding behind the virtual avatar was firstly popular in Japan, where in 2017 Project A.I. with the main character "Kizuna A.I." started the "boom" of virtual youtubers (hereinafter referred to as vtubers) and, as Nordvall described them, digital multi-space communities [18].

Vtubers firstly used full-body pose tracking and motion capture for their avatars. Nevertheless, this system required the use of extra hardware, such as multiple RGBD video capture devices (Kinect) to properly capture motions of the actor behind the avatar. Ever-growing popularity of this type of content lead to the simplification of this process, with focusing only on the head and face of the actor.

In this thesis I researched multiple solutions for face tracking and animating the avatar, as well as for multi-platform communication to take full advantage of independence of aforementioned modules.

Main contributions of this project are (a) Calculating transforms from human face is suitable for the skeletal animations, which are usually easier to create and more accessible than blend shapes. (b) Using the face tracking and the avatar animator as independent modules causes that from the single camera/video input it is possible to animate multiple avatars on different devices.

Chapter 2

Detection of Human Face, Animation of Avatar, and Interconnection Between Modules

The exponentially growing popularity of online content of so called vtubers and forms of entertainment which require computer vision technologies such as the face and/or pose tracking or the motion capture leads to the need of simplifying the creation of such forms of entertainment. Today, all the content creator needs is a software in the computer and/or an app on their mobile device to replace the image of his real self with the virtual avatar and to make use of the newest and the yet fastest media processing algorithms.

In this section I will discuss existing solutions and used technologies for the purpose of tracking the human face and/or animating the virtual avatar, and how two independent modules could communicate.

2.1 Related Work and Existing Solutions

Most companies with this kind of the online entertainment (such as Hololive[1], Nijisanji[2] or VShoujo[3]) use their own software to capture the movement of their actor's head and face expressions, and to project it on the virtual avatar, but they do not publicly share information about it. However, the overall functionality of such software is clear, since some content creators already partially mentioned it online on their respective accounts/channels[4].

The software those companies use, usually consists of two main parts/modules:

- An application used for **capturing the face** which usually runs on the mobile device (typically iPhone) since it has a better camera than most low price web cameras for computer,
- and a computer program which handles **the animation and the rendering** of the virtual avatar.

[1] https://en.hololive.tv/
[2] https://www.nijisanji.jp/en
[3] https://www.vshojo.com/
[4] A simple setup requires only a camera and a software – https://www.wikihow.com/Become-a-Vtuber

In my solution, I expanded this model by adding the intermediator – the websocket server. The reason for this is explained later in Section 2.5. In this section I discuss some of the existing solutions that animate the avatar based on the human face video.

Cao et al. presented in their work [5] a real-time facial animation system based on 3D shape regression of the face. Their system uses a single web camera with resolution 640×480 pixels, and records video at the speed of 30 frames per second. At the beginning the setup step is required. The user performs a set of standard expressions and specific head poses, and the system captures their face. A set of facial landmarks are automatically detected in those images. Landmarks can be manually corrected by the user if needed. By using blend shape model for each image the 3D facial shape is calculated. The user-specific 3D shape regressor is then trained by using those facial shapes and input images.

Every avatar has predefined blend shapes. By performing an expression in front of the camera, the regressor creates the user-specific 3D mesh. The texture is then applied on the mesh. Finally, the avatar is animated using blend shape interpolation technique. This method (morphing) is discussed in Section 2.4.1.

Cao et al. in [4] further improved regression-based algorithm from [5], and proposed fully automatic approach to facial tracking and animation with a single web camera. Their approach no longer require the time-consuming calibration step. Contrary to that, it learns a generic regressor from public image datasets which can be applied to any user. Nevertheless, the user identity can not change across frames. Their Displaced Dynamic Expression (DDE) model is a representation of facial shape that combines a 3D parametric model and 2D landmark displacements.

Li et al. in [13] presented a real-time and calibration-free facial expression capture framework. The framework uses 3D sensor with the ability to capture dense depth data, such as Kinect. At the beginning of the face tracking process, the initial scan of the user with neutral facial expression is captured. This helps to automatically generate generic linear blend shape expressions. An adaptive PCA model, based on correctives is used to rapidly adapts to the expressions of the current user during the tracking. The corrective shapes are used to learn the distinct look and expressions of the user. This helps to mimic the user's facial expressions more accurately, since those can differ for each user.

The framework uses a 3D depth map and 40 2D facial landmarks (lip contours, eye contours and eyebrows). Facial landmarks are obtained from the Live Driver (ImageMetrics[5]). Live Driver implements a data-driven tracking algorithm that works on any individual, under any lighting condition.

In [8] Garolera et al. presented a web-based real-time markerless facial retargeting system. By utilizing the library Beyond Reality Face (BRF)[6], they were able to track facial landmarks using only a web camera without depth sensor, right in the web browser. Their system offers two methods for controlling the avatar's face: facial retargeting and facial rig with 2D interpolation.

Facial retargeting reproduces user's facial expressions on the avatar, using a single web camera. Their system uses morphing animation method. Nevertheless, their method is compatible with skeletal animations. BRF provides 2D facial landmarks every frame. Their system then calculate difference between them and landmarks from the pre-initialized neutral face. From the calculated differences the system animates the avatar using blend shapes.

[5]http://image-metrics.com/
[6]https://www.beyond-reality-face.com/

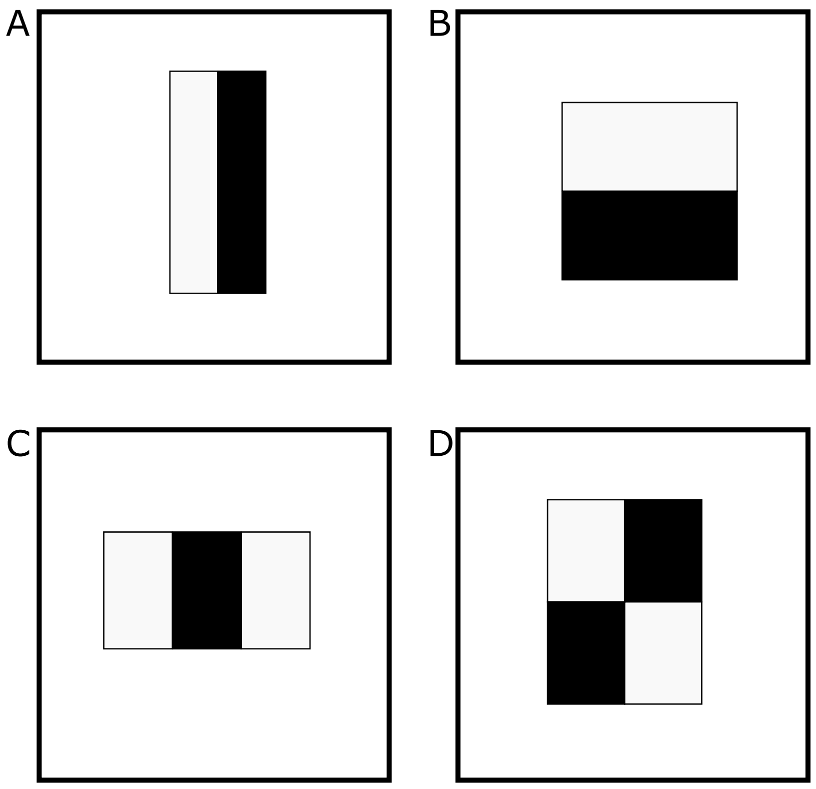


Figure 2.1: Viola-Jones method uses three kinds of rectangle features to detect faces [24]. Two-rectangle features are shown in (A) and (B), (C) shows a three-rectangle feature, and (D) a four-rectangle feature.

Facial rig uses preconfigured expressions of the character which are controlled through an interface, e.g. a gamepad or a web interface. They defined several facial expressions in the valence-arousal model [23]. By using the interface, the avatar blends between predefined expressions.

Weise et al. in [25] primary address ensuring of the robust processing of the low resolution (640×400) and high noise levels of the input data from Microsoft Kinect. The setup step consists of the user performing a predefined sequence of example facial expressions. By using Viola-Jones method [24] their system detect the face in the first frame of the setup step. Figure 2.1 demonstrates the use of simple features to detect face in the Viola-Jones method. The value of two-rectangle feature is the difference between the sum of pixels within two rectangular regions. The value of three-rectangle feature is the sum of pixels of two outside rectangles subtracted from the center rectangle. The value of four-rectangle feature is the difference between the sum of pixels within white regions and black regions. Using the Poisson reconstruction the skin texture is generated from the color images.

Their linear PCA model that represents the variations of different human faces in neutral expression is firstly registered towards the recorded neutral pose. From this, the template mesh is obtained. The template mesh roughly matches the user's face geometry. The template is then wrapped to each of the recorded expressions. Additional texture constraints are added in the mouth and eye regions to improve registration accuracy.

The user-specific blend shapes are reconstructed from three things: a generic blend shape model, reconstructed example expressions, and approximate blend shape weights.

Blend shape weights specify the linear combination of blend shapes for each facial expression. These weights are manually determined only once and kept constant for all users.

2.2 Face Tracking Solutions

To animate the avatar based on the human face from either camera or video, it is crucial to correctly detect and track presence of the human face in the input frames. This process is called the face tracking.

The approach to the human face tracking came a long way in the last decade. It changed all the way from the omni-directional monocular marker-based head-pose estimation [14] to the perception pipelines using media processing algorithms and pretrained neural networks for recognising human face in the image [3, 4, 5, 7, 8, 12, 13, 19, 25].

Several different technological solutions exist for face tracking. I'll be discussing them in this section.

2.2.1 Solutions That Require Special or Additional Hardware

Former solutions for estimating head position require additional hardware, such as a head-wearable marker structure. This approach is explained by Lichtenauer et al. in [14]. Using metal frame and passive white markers it can detect markers even under difficult lighting conditions.

Other solutions that require additional hardware use RGB-D cameras, such as Microsoft Kinect 2 for additional depth information of the image. As Derkach et al. explained in [7], methods that operate on 3D data are less sensitive to changes in illumination and viewpoint than 2D image-based approaches. Their approach use only depth information from user's camera to estimate head position, as shown in Figure 2.2.

Nevertheless, estimating head position would not be enough to animate the whole face of the avatar, since data about eyes and mouth would be missing. This project aims to obtain those data as well as estimating head position, without the use of the additional equipment, such as Kinect or head-wearable marker structure.

2.2.2 Solutions Based on Pretrained Models for Face Tracking

In my solutions, as explained in Section 3.1, I chose to implement face tracking as a Python script, running on the desktop. However, to better understand my choice of technology, in this section I discuss several different technologies that engage in face tracking.

Abadi et al. presented Tensorflow in [1] as a system for large scale machine learning. It uses a unified dataflow graph to represents the state of the algorithm and its computation. It can run trained models for inference on various platforms ranging from large datacenters to mobile devices. As Yuan et al. proposed in [26], a convolutional neural network based on TensorFlow is suitable for real-time face detection.

Nevertheless, as Smilkov et al. stated in [21], production-quality ML libraries are typically written for Python and C++ developers. The importance of JavaScript community and web-based applications lead to the development of the Tensorflow.js library. Figure 2.3 shows the example of face detection, using Tensorflow.js' Blazeface detector[7].

Both Tensorflow.js and Mediapipe are runtimes used for face tracking on desktops and on mobile devices. Both runtimes were compared in [10] by Grishchenko et al., using pose

[7]https://github.com/tensorflow/tfjs-models/tree/master/blazeface

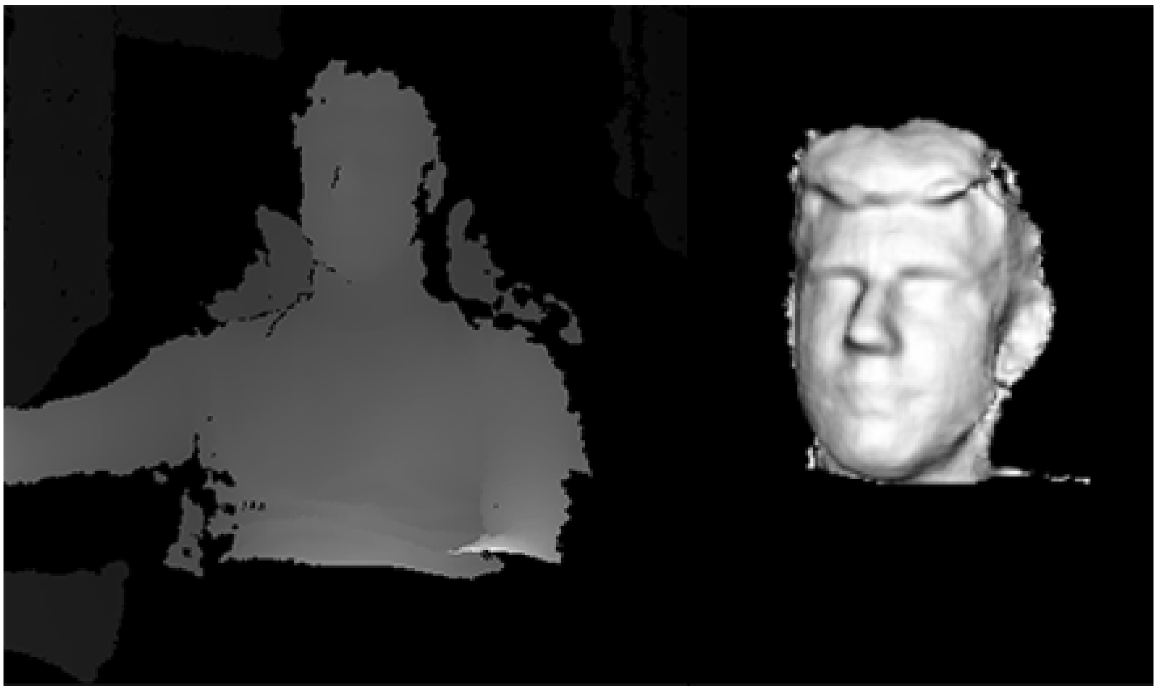


Figure 2.2: Depth map produced by Kinect's depth image stream (on left) and produced 3D mesh of the head (on right).

Device	Mediapipe (FPS)	Tensorflow.js (FPS)
MacBook Pro 15" 2019	75 | 67 | 34	52 | 40 | 24
iPhone 11	9 | 6 | N/A	43 | 32 | 22
Pixel 5	25 | 21 | 8	14 | 10 | 4
Desktop	150 | 130 | 97	42 | 35 | 29

Table 2.1: Results of comparing Mediapipe and Tensorflow.js runtimes showing inference speed. The detailed table of results could be found in [10].

detection. Results, as shown in Table 2.1, have revealed that Tensorflow.js runs significantly slower on desktop and Android devices. It has surpassed Mediapipe only on iPhones.

2.3 Mediapipe – Building a Perception Pipeline

Based on the results from Table 2.1, it is clear that Mediapipe is a better solution for this project, unless we want to use iPhone as a target device. In this section I discuss Mediapipe in further details.

As Lugaresi et al. explained in [15], Mediapipe is Google's multi-platform open-source framework for perceptual input processing. Mediapipe connects individual perception models into maintainable pipelines, which eliminates problems with implementing additional processing steps or inference models into the perception application that processes sensory data.

Mediapipe can build a perception pipeline as a graph of modular components, such as inference models or media processing functions. Graph takes sensory data (video/audio) as

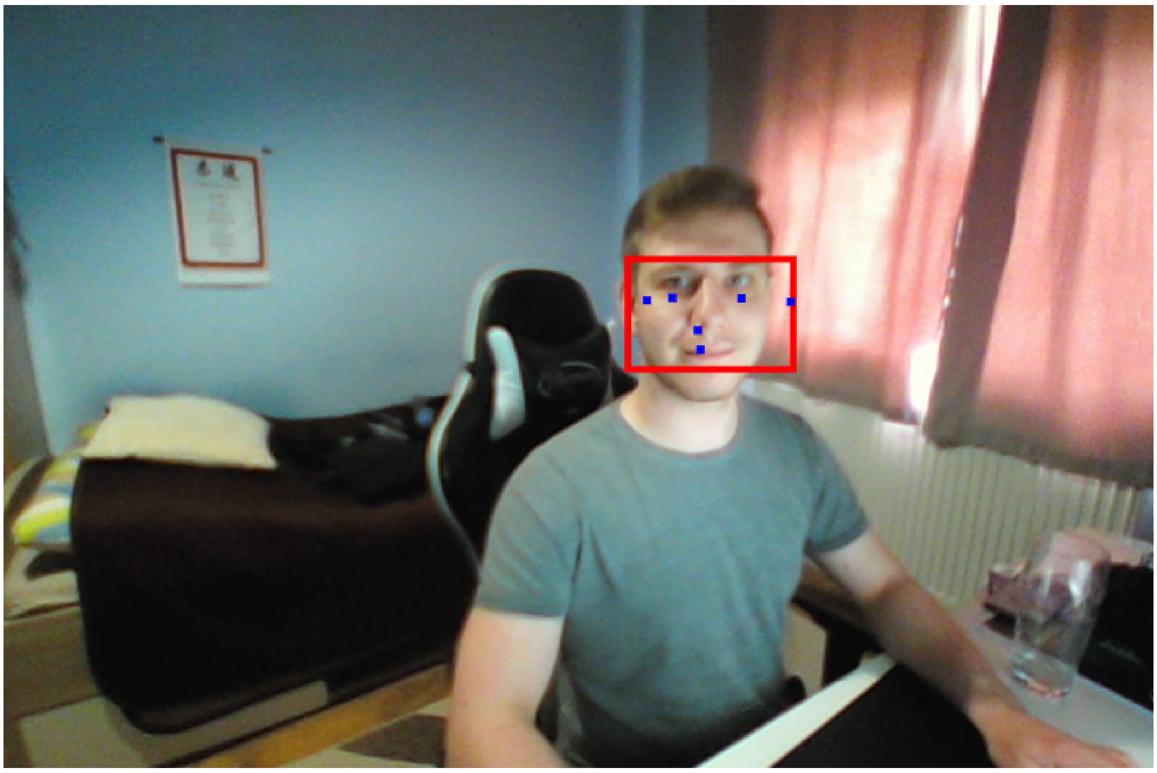


Figure 2.3: Example of face detection using Blazeface lightweight model. This model detects faces and returns 6 landmarks. It is also available as a part of Mediapipe.

an input, and after processing it, outputs the percieved description, such as face landmark annotations.

Mediapipe's graphs do not define internals of neural networks, as TensorFlow does. Instead, it specifies pipelines with one or more models embedded. Its pipeline is defined as a directed graph of components (graph nodes). Each component is a calculator, and it is possible to define custom calculators.

Calculators are connected via streams, through which data (packet sequences) flows. Streams maintain their own queues, so the receiving node can process data at its own tempo. Figure 2.4 demonstrates the graph for face detection[8].

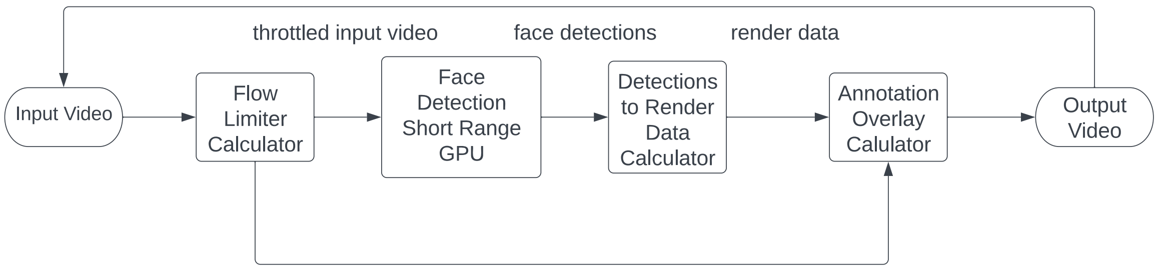
With Mediapipe it is fairly easy to build and maintain perception pipeline. Performing a complicated task in real-time is easier by defining custom calculators, using resources efficiently and processing media streams in parallel and at different rates.

2.3.1 Facemesh – Estimating the Face Geometry

This section explains the Mediapipe's solution to estimating the 3D geometry of human face – Facemesh.

As explained in [12] by Kartynnik et al., Facemesh is a solution that estimates facial surface geometry – 468 landmarks in real time. Landmarks are vectors in 3D space that

[8]https://github.com/google/mediapipe/blob/master/mediapipe/graphs/face_detection/face_detection_mobile_gpu.pbtxt



throttled input video
face detections
render data
Input Video
Flow Limiter Calculator
Face Detection Short Range GPU
Detections to Render Data Calculator
Annotation Overlay Calulator
Output Video

Figure 2.4: Mediapipe's graph for Facemesh. Calculators (nodes) use streams as the input/output to pass data among themselves.

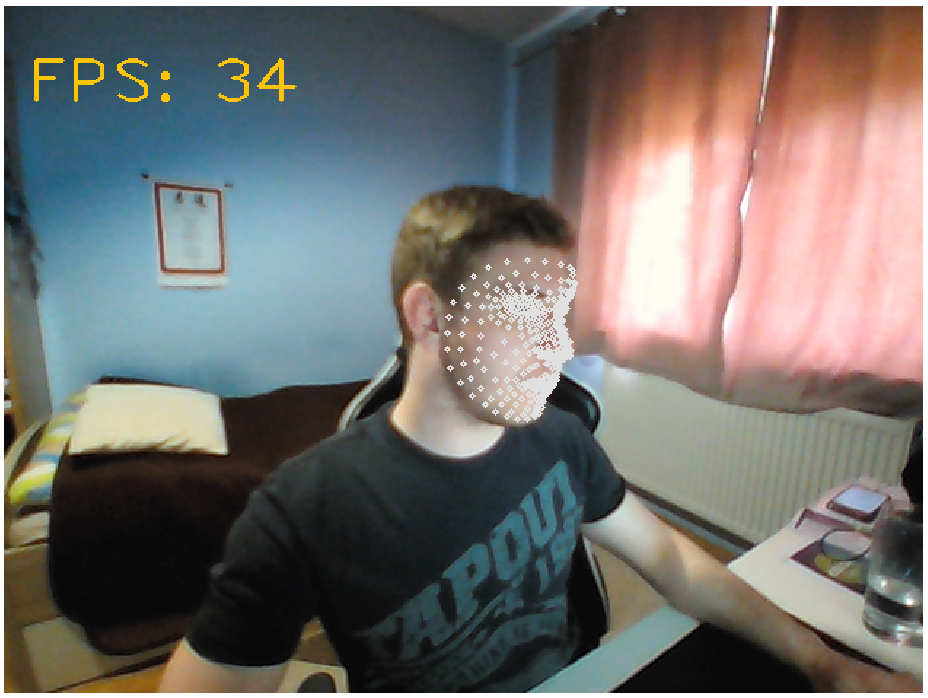


Figure 2.5: Facemesh produces 478 landmarks, which define the face geometry, even under difficult angle and light conditions

make it easier to copmute transforms for the avatar. By applying attention mesh model[9] and refining landmarks, it is possible to obtain ten additional landmarks around irises.

Facemesh estimates positions of 3D vertices with a neural network. There are 468 vertices in total (478 with irises). Vertices in the mesh are independent from each other and are arranged in fixed quads. The face areas that tend to differ among humans, such as nose area, have more point density. An example of the face detection using Facemesh is shown in Figure 2.5.

Facemesh takes a single RGB frame with no depth sensor information as an input. After processing the image, it outputs the complete 3D mesh as a face representation.

The **image processing pipeline** works as follows:
Firstly, the input frame is processed by a lightweight **face detector** that, as Bazarevsky et al. explained in [3], produces 6 facial keypoints (landmarks) and and axis-aligned rectangle.

[9] https://google.github.io/mediapipe/solutions/face_mesh.html#attention-mesh-model

Produced keypoints are: eye centers, ear tagions, mouth center and a nose tip. This allows the face detector to estimate the face rotation (roll angle).
Secondly, the obtained rectangle is cropped from the original image and resized to fit as the input to the **mesh prediction neural network** (ranging from 128×128 to 256×256 pixels). The model produces 3D landmark coordinates, which are subsequently mapped back into the original image coordination system.

Pixel coordinates of the given points in the 2D input image and x and y coordinates of the vertices are the same. The z coordinates represent the depth relative to the reference plane. They are re-scaled to maintain a fixed aspect ratio between the span of x and z coordinates.

In the camera/video input (sequence of frames) the facial crop from the previous frame is available to be used in the next fame as well. This means that the face detector is needed only in the first frame or during re-acquisition [12].

2.4 Rendering and Animation of 3D Models

In the present there are two main styles which content creators use for their avatars – Live2D[10] and 3D. Nakagawa described Live2D in [17] as a 2D image that is animated without requiring any changes. While Live2D provides more unique and unusual art style for a real-time moving character, 3D avatars provide much more variability and scalability. Not only for the model of the avatar, but also for the whole scene. 3D avatar looks better in the 3D scene, while moving picture (Live2D) would fits better in 2D scene.

In this section I discuss popular animating techniques and options for animating 3D models, such as morphing, skeletal animations and free-form deformation, and how to render results in the user's web browser.

2.4.1 Animation Techniques

To animate a virtual avatar – to deform its shape over time, Garstenauer and Kurka in [9] divided animation styles into three methods:

- **Morphing** – also known as morph target animation or shape blending,
- **Skeletal animation** – using mesh skinning,
- **Free-form deformation**.

Free-form deformation (FFD) is usually used on specific parts of character's body, such as muscles, hair or cloth, to stretch and squash them. To animate a mesh, three-dimensional grid of control points is placed over the surface, then deformed and the deformation is applied to the underlying surface.

Morphing. Each character have several key shapes defined. Those key shapes are called *morph targets* or *blend shapes*. By applying *blend shape interpolation*, the character deform its shape between respective blend shapes. This method was used by Cao et al. in [5]. Their facial animation system is based on 3D shape regression. Nevertheless, they use user-specific blend shape models, thus a setup step is required for each new user. A setup step consists of the user acting a set of standard facial expressions, from which the user-specific regressor is trained. This could be time-consuming for a new and unexperienced user.

[10] https://www.live2d.com/en/

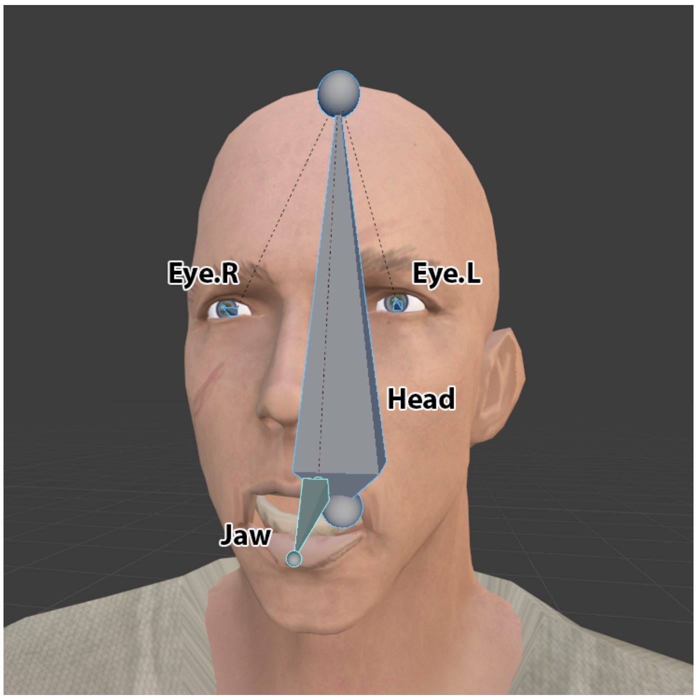


Figure 2.6: An example of the skeleton, deforming a mesh, and its bones with their respective names. A head is the parent for all other bones.

Skeletal animation is nowadays the most popular character animation method [9]. This method makes use of the skeleton, also called a *rig* that consists of *bones* or *joints*, as shown in Figure 2.6. Bones are then used to control vertex groups to deform specific parts of the mesh. This method is very popular especially in the game development, because of its variability and it is fairly easy to implement and modify. Skeletal animations could also be exported and reused by other similar characters. Dai et al. demonstrates the reusability of skeletal animations in [6], by using BVH (Biovision Hierarchy)[11] files, which are commonly used in motion capture.

2.4.2 Animation in the Web Browser

Ahire et al. stated in [2] that animation data can require a lot of space, which is not suitable for the browser, unless efficient compression algorithms are applied on the animation file, so it is possible to rebuild them with minimal information. To process data and render the model, JavaScript is generally used. Some techniques to compress and transmit both static and dynamic 3D objects can be extended for animation files. Nevertheless, those techniques lack uniformity.

An imperative approach to visualize 3D web graphics in real-time is *WebGL*. It uses HTML5 canvas element in combination with JavaScript to expose graphics API based on OpenGL ES 2.0. Moreover, it can use GPU based methods, so the deformation of vertices and fragment values will be computed on the GPU in the shaders.

To actually work with the 3D object in the web browser, several JavaScript libraries could be used:

[11]https://research.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/BVH.html

- **SpiderGL**,
- **Babylon.js**,
- **Three.js**.

SpiderGL aims to provide an abstraction for developers to use the underlying graphical capability of WebGL. It supports asynchronous content loading. Nevertheless, it lacks the support for animations.

Babylon.js is an open source framework supported by Microsoft. It has features, such as collision detection and physics engine and thus resembles a game engine, but it also supports key frame animations and skeletal animations.

Three.js is very popular nowadays. The good performance and good abstraction makes it easier to use than dealing with low level initialization. It can handle morphing animations, skeletal animations and key frames as well as blending and interpolating among the key frames. By changing parameters of the bones in the skeleton such as position, rotation or scale, it is possible to animate characters in real-time. Three.js attempts to bring wide range of animation features.

2.4.3 File Format of 3D Model

Several file formats could be used to load 3D objects in the web browser. Frequently used file formats are:

- **OBJ**,
- **X3D**,
- **GLTF**.

Wavefront's **OBJ** defines only a 3D geometry in a human readable form, but it does not support any kind of animation. The only animation that could be achieved with OBJ is to export each key frame as a separate file and then render them in the order of the animation. This consumes more memory and it is ineffective with real-time skeletal animations.

Extensible 3D (**X3D**) is the royalty-free open standard. X3D supports Humanoid Animation (HAnim[12]). HAnim uses a "standard" human models that have a certain joint hierarchy similar to human skeleton, and it needs an initial or default skeleton pose. To animate facial expressions, a method very similar to morphing is being used. Kapetanakis et al. used X3D files in [11] to manage virtual 3D scene in web browsers of connected clients through the websocket server.

GL Transmission Format (**GLTF**[13]) minimazes both the size of 3D assets, and the runtime processing needed to unpack and use those assets. This file format has been strongly adopted by JavaScript frameworks such as Three.js and Babylon.js, as well as web-focused companies such as Google and Facebook. GLTF coulb be saved in both binary and JSON formats for human readability. It supports morphing as well as skeletal animations. JSON format for the scene hierarchy makes it very easy to access any object (called a node) in the scene. Bones are also nodes. This makes it very effective to transform bones in real-time. GLTF supports Open3DGC encoding, which is designed for fast decoding in JavaScrip [2].

[12] https://www.web3d.org/working-groups/humanoid-animation-hanim

[13] https://github.com/KhronosGroup/glTF

2.5 Communication of Multi-Platform Modules

With fully functional face tracking and avatar animator modules, their only problem is that they are completely independent modules. They do not even have to run on the same device. This creates the necessity to transfer data from face tracking module to avatar animator module.

To transfer data correctly, the intermediator, the middle man is required to connect two modules and ensure the communication from the face tracking module to the avatar animator. The intermediator needs to be able to receive data from the face tracking module, as well to propagate them to all running avatar animators. That means bidirectional communication is required to both receive data and send data.

2.5.1 Existing Solutions to Communication of Web Browser Applications

The architecture and implementation of my solution will be discussed in the next chapter. Nevertheless, it is important to be acquainted with what kind of technologies are modules using, before determining their connection between them.
Face tracking module used in my solution is a python script, while the avatar animator module is a web browser application, running a JavaScript code. This means that solution to establishing a connection between them has to be multi-platform.

For a long time, standard HTTP connection was used for the purpose of communication between the server and client (polling, long-polling and streaming). However, this brings its drawbacks in bandwidth consumption, latency and usability. They are mainly caused by large headers used and the request-response model of the HTTP protocol, which wasn't developed for bidirectional real-time communication. These drawbacks were further discussed by Srinivasan et al. in [22].

As mentioned in [22], to provide real-time communication between client and server, browser plug-ins and standalone applications have been used. They are not restricted to HTTP and could use any network protocol for communication.
The commonly used protocol for standalone applications is TCP. However, it does not provide any standardized security mechanism, so additional implementation and maintenance of security solution is required.

As Melnikov et al. explained in [16], the websocket protocol allows for two-way communication that does not rely on opening multiple HTTP connections (e.g. XMLHttpRequest). It runs over a single TCP connection and wokrs over standard HTTP ports (80 and 443). Nevertheless, the design of websockets is not restricted to work over HTTP ports only. Future implementations could use a simpler handshake over a dedicated port, which implies that websocket traffic does not match standard HTTP traffic.

Chapter 3

Solution Design

After reviewing possible solutions for tracking facial expressions, animating 3D avatars and communication between modules I decided to use following technologies:

- Mediapipe's **FaceMesh** – to track face on the camera or video,
- **Three.js** – to render 3D scenes in the web browser and to control skeleton of the avatar,
- **Websockets** – to provide persistent communication between two modules on different platforms.

The basic idea is to capture video, recognise human face and estimate its geometry, calculate transforms for the animation and send those data to the websocket server. From there those data will be propagated to all connected avatar animators. Those firstly correctly load the 3D model of the avatar and then animate it based on the obtained data from the websocket server. The pipeline of my solution is shown in Figure 3.1.

Following sections explain my design of workflow for individual modules in detail.

3.1 Face Tracking

In this section I discuss the design of the face tracking module. How to capture face, calculate new transforms from landmarks and propagate those changes to the websocket server.

As explained in [12], FaceMesh returns 468 3D vectors (landmarks)[1] from the input frame. By applying attention mesh model, it returns 478 landmarks, including ten landmarks around irises. As shown in Figure 3.2, not all 478 landmarks will be necessary to calculate new transforms of the avatar, because avatars has a very limited number of bones that could be deformed. It is enough to consider only the most important landmarks, such as the very top, very bottom and sides of the face, lips, eyes, irises, eyelids, and nose.

After calculating all necessary parameters, they are sent to the websocket 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 3.3.

From obtained landmarks (vectors) it is possible to calculate how much is the mouth opened, if the right or left eye blinked, head rotations (roll, pitch, yaw), and the line of sight. Following sections explain on how to make those calculations.

[1] Original image of indexed landmarks on canonical face could be found at `https://github.com/google/mediapipe/blob/master/mediapipe/modules/face_geometry/data`

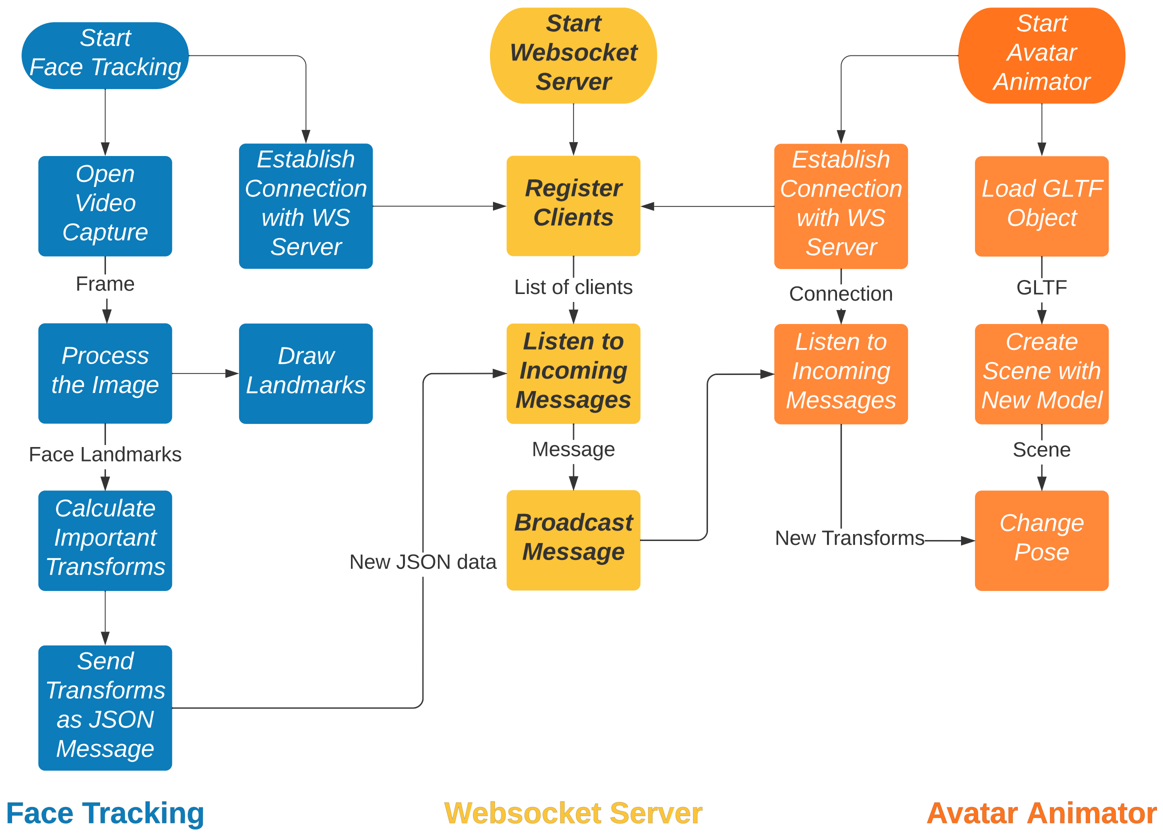


Figure 3.1: The pipeline of my solution for Animation of the avatar face based on the human face video.

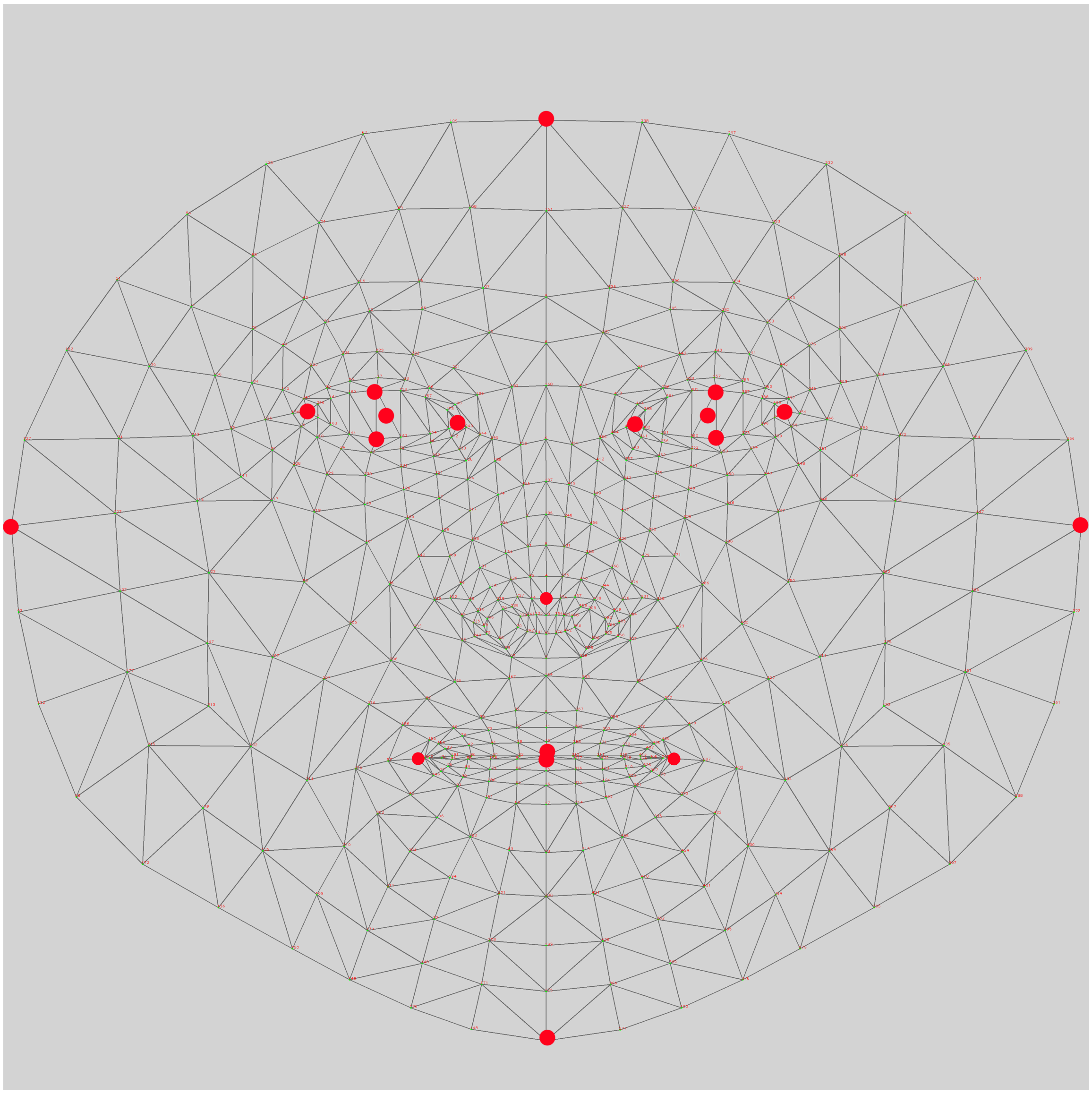


Figure 3.2: Highlighted landmarks – only few landmarks out of total 478 are actually important for my solution.

3.1.1 Estimating the Head Rotation

The head can rotate around three axes. Rotation around axis x, y and z are called the pitch, roll and yaw or the nod, rotate and turn, respectively, as shown in Figure 3.3. While all three of them could be calculated at the same time through rotation matrix, calculating the roll separately gives better results.

The roll of the head (rotation around y axis) could be calculated very easily. Two landmarks are all that is needed – the topmost and the lowest landmark of the head. By subtracting their x coordinates, we get either positive or negative value, which means the head is tilted to the left or to the right. Zero would mean the head is not rotated around y axis.

The pitch and yaw 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 by Slabaugh in [20], three Euler angles could be obtained from the rotation matrix. Nevertheless, only two of them are actually needed, since the roll of the head is calculated differently. General rotation matrix have form as

$$\mathbf{R} = \begin{Vmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{Vmatrix} = R_x(\phi)R_y(\theta)R_z(\psi)$$

where $R_x(\phi), R_y(\theta)$ and $R_z(\psi)$ are rotations of ϕ, θ and ψ radians around axes x, y and z, respectively. However, $R_y(\theta)$ is not needed, since the roll is calculated separately.

3.1.2 Calculating How Much is Mouth Opened

To calculate how much is mouth opened, all is needed is to take the lowest landmark of the upper lip and the topmost landmark of the bottom lip and calculate the euclidean distance between them. The euclidean distance is described by Equation 3.1.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{3.1}$$

where (x_1, y_1) and (x_2, y_2) are x and y coordinates of two aforementioned landmarks.

3.1.3 Calculating the Line of Sight

Figure 3.4 shows how to calculate the direction in which are eyes looking. This is done through calculating an euclidean distance between the most left corner of the eye and the most right corner, and between the lowest landmark of the eye and the topmost landmark of the eye.

Then the euclidean distance between the iris and the most left corner of the eye is calculated. After dividing obtained distances between the most left corner of the eye and iris and between the most left corner of the eye and the most right corner, we get the value for the horizontal line of sight on scale of 0 to 1, where 0 is the most left and 1 is the most right.

The vertical line of sight is calculated similarly – by dividing the euclidean distance between the lowest landmark of the eye and the iris with euclidean distance between the lowest landmark of the eye and the the topmost landmark of the eye. The obtained vertical line of sight is on scale of 0 to 1, where 0 is the lowest and 1 is the topmost.

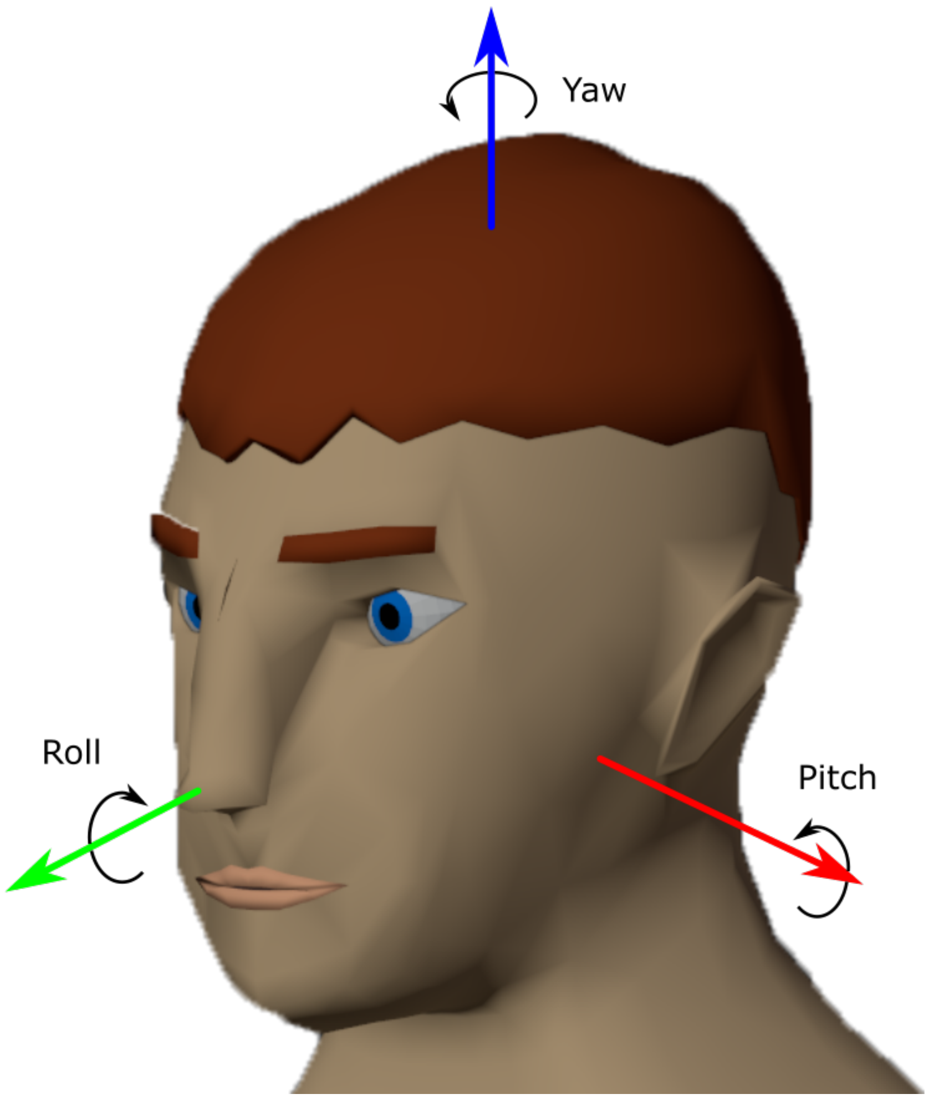


Figure 3.3: Rotation of the head around three axes in terms of pitch, roll and yaw angles.

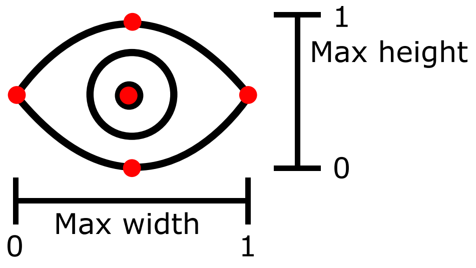


Figure 3.4: Calculating position of the iris inside the eye. Co-ordinates are [0, 0] - bottom left, [1, 1] - top right.

3.1.4 Determining If the User has Blinked

To find out if the user has blinked, four landmarks are needed for each eye – the most left corner, the most right corner, the topmost and the lowest landmark of the eye. After calculating the euclidean distance between horizontal points h and between vertical points v for each eye, we make v to h ratio, as shown in Equation 3.2.

$$b_r = \frac{v_r}{h_r}, b_l = \frac{v_l}{h_l} \tag{3.2}$$

where b_r and b_l stand for left blink and right blink value, respectively. Those values can later be compared with a threshold, to determine if the user has blinked.

3.2 Avatar Animator

When data from the real world has been obtained it is possible to animate the virtual avatar based on those data. The main goal of this project is to do this right in the user's web browser so no additional application is required and multiple tabs with different avatars could be opened at the same time.

In this section I discuss the design of the Avatar animator. I start with loading the avatar, controlling the scene, and finally animating the avatar.

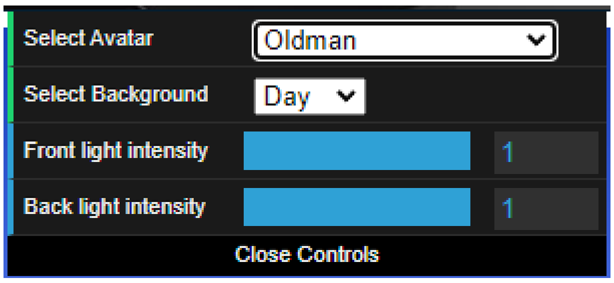
3.2.1 Loading the Avatar

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. Its hierarchical structure allows for easy traversing of the whole avatar. The only required bone, however, is the head bone, which is usually one of the most common bones for human avatars, and acts as a parent bone for every other face-deforming bone (eyes, jaw, etc.).
Other bones – eyeballs, jaw and eyelids are optional and they do not have to be animated, if the avatar does not have them, since each avatar could be rigged differently.

The rigging of the avatar is usually not the only difference among avatars. The scaling of the model, bone orientations, and bone names could differ as well. Hence, it is important to map those things to the avatar. To do that, an external configuration file is needed for each avatar. This configuration file is a simple JSON file, containing respective values for each bone name, orientation of some bones, multipliers and offsets for animations, and the scale factor. Avatar animator would then scale the avatar and deform only those bones that are mentioned in the configuration file with their respective names.

3.2.2 Control the Scene

After loading the model, scene is configured with lights and a background. Those can be changed in real-time as mentioned above. Figure 3.5 demonstrate possible changes a user can make – change light intensity and switch among different backgrounds as well as among different avatars in real-time, via simple GUI. Those settings are explained later in Section 4.1.2. In Figure 3.6 is a showcase of different backgrounds and in Figure 3.7 is a showcase of selectable avatars. User has also full control over the camera and can move freely in the scene around the avatar as well as zoom in and zoom out.


Select Avatar: Oldman

Select Background: Day

Front light intensity: 1

Back light intensity: 1

Figure 3.5: It is possible for a user to use GUI to select various avatars and backgrounds as well as to change the light intensity in real-time.

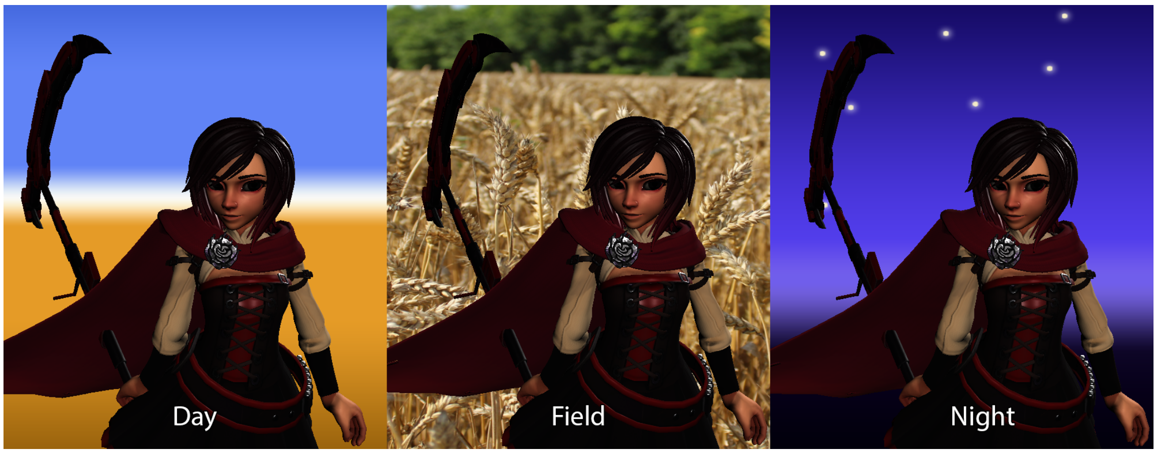


Figure 3.6: Showcase of various backgrounds.

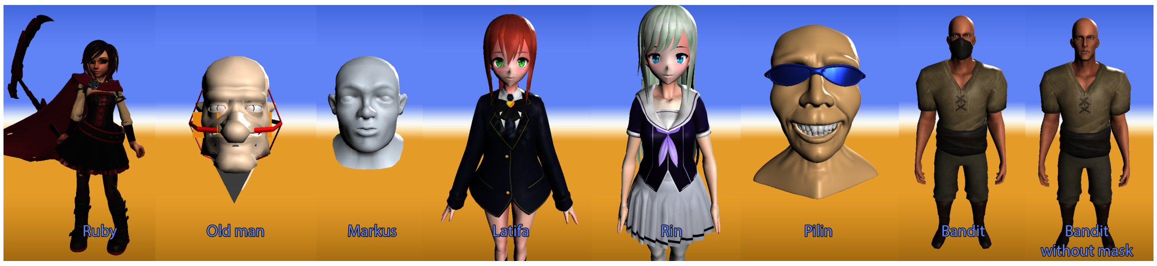


Figure 3.7: Showcase of 8 different avatars.

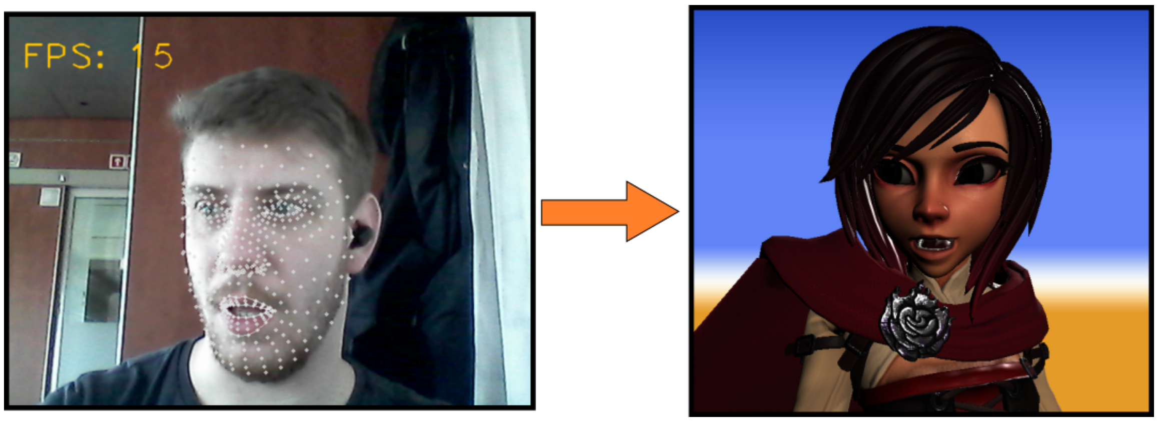


Figure 3.8: Newly calculated data are sent over the websocket server every frame, which creates smooth movements of the avatar.

3.2.3 Animating the Avatar

To obtain data which are important for animations, the avatar animator module needs to be configured as a websocket client. It should connect to the websocket server automatically and continuously listen to incoming messages. If the connection is lost, the user is notified and the avatar is not animated anymore, but rests in the last known pose.

After obtaining new data from the websocket server, it is important to update transforms for each bone that the particular avatar has defined. Thanks to GLTF hierarchical structure, it is fairly easy to traverse the whole scene to look for the particular bone (bone is also called an object) by its name. After multiplying new transforms and adding the offset from the configuration file, the avatar is animated. This is done every frame in the face tracking module, so the animation is seamless as could be seen in Figure 3.8.

3.3 Websocket Server

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

To correctly transfer data from the face tracking module to avatar animator modules, the intermediator is required. In my solution, I chose the websocket server to be the intermediator. The websocket protocol is a multi-platform protocol that has very simple Application Programming Interface (API). Its API is explained in more detail in [16, 22]. This makes it fairly easy to implement to both the face tracking and the avatar animator modules.

The websocket server registers all clients that established connection with it and then broadcasts incoming 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 3.9.

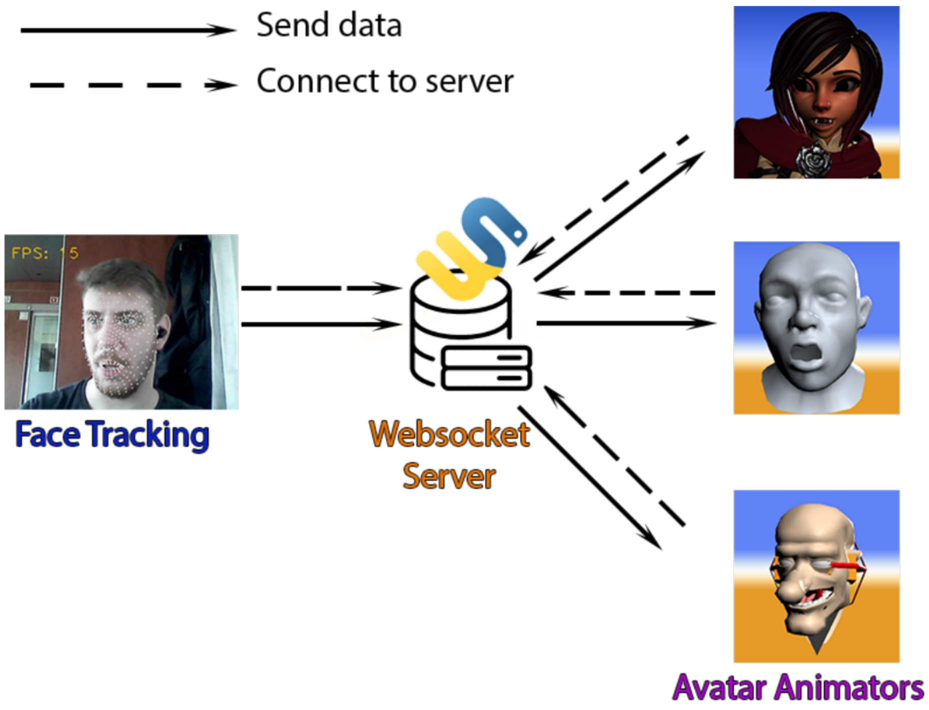


Figure 3.9: Websocket server recieves data from the face tracking module and distributes them to avatar animators.

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.

Chapter 4

Implementation and Evaluation

4.1 Implementation Details

In this section I describe how I implemented three modules – face tracking (python script), avatar animator (JavaScript application) and the websocket server (python script).

4.1.1 Script for Capturing the Face

Face tracking module is in fact a python script implemented using python version 3.9.7. It firstly loads the input mode from arguments. Integer number (0 or greater) indicates the web camera index, from which the input will be read, since computer can have more than one camera connected. Anything else will be read as a path to the video file.

Then it opens a camera, using package *OpenCV*[1]. The same package is then used to read every frame (hereinafter referred to as an image) of the video input, and convert color space of the image from BGR to RGB. Marking the image as not writable will improve performance.

The image is then processed using the Mediapipe's solution – Facemesh. This solution is explained in Section 2.3.1. Settings for the facemesh model used in this project are as follows:

- **Maximum number of faces** – 1
- **Minimum detection confidence** – 0.5
- **Minimum tracking confidence** – 0.5
- **Refine landmarks** – *True*

Number of faces is set to 1, since there is only one avatar in each avatar animator, so it is unnecessary to look for other faces in the image. Confidence values are set to default. The option to refine landmarks is not set by default. This would mean that processing the image by the facemesh would return only 468 landmarks, ignoring landmarks defining irises (5 for each eye). Tracking irises is important to determine the line of sight, as mentioned in Section 3.1.3.

After processing the image, it is set back to writable and converted back to BGR color space, for future displaying of the video.

[1]https://opencv.org/

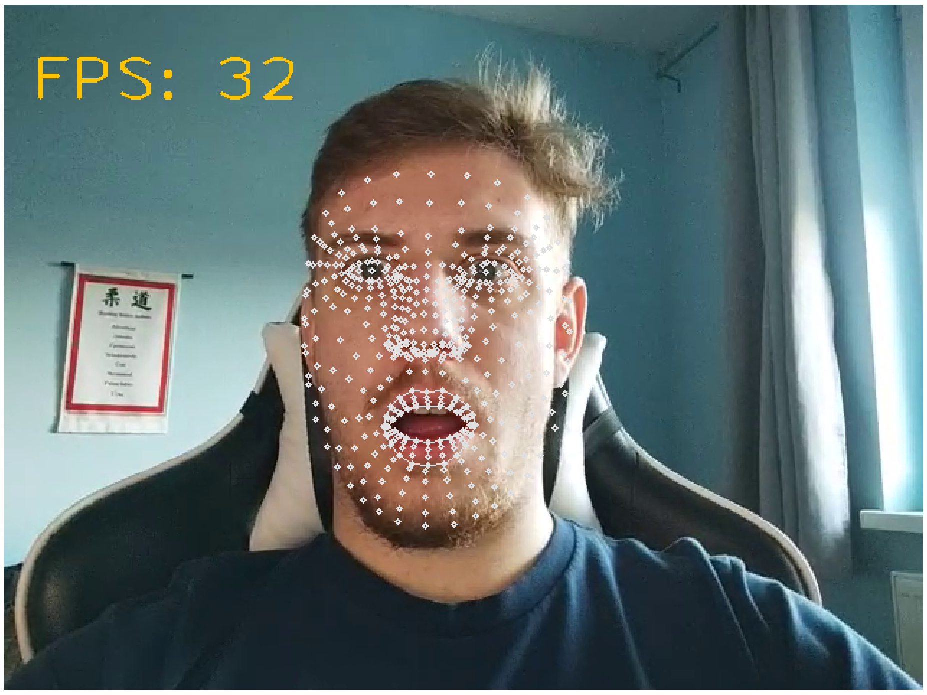


Figure 4.1: The resulting window of Face tracking module consists of the input frame, calculated landmarks, and inference speed.

The processed image returned 478 landmarks defining the face geometry. From those landmarks are calculated new transforms – data to be sent over to the websocket server. Those calculations were discussed in Section 3.1. They are calculated with the help of *math* and *numpy* packages. The newly calculated data are asynchronously sent to the websocket server using *asyncio*[2] package and *websockets*[3] package to handle the websocket connection with the server.

After calculating and sending over all necessary data, landmarks are drawn over the user's face in the image and the inference speed is calculated. This is done strictly for user's informative purposes. The resulting image is then displayed in a separate window using *OpenCV*, as show in Figure 4.1.

4.1.2 Web Browser Application for Animating the Avatar

Avatar animator is a JavaScript module that primarily utilize JS library/API – *Three.js* in cojuction with HTML canvas element and WebGL. Three.js was explained in more details in Section 2.4.2. The script starts with creating an empty scene.

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 them. When the connection dies, the user is alerted.

As explained in Section 3.2.1, each avatar 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 bone names, multipliers and offsets for bones,

[2]`https://docs.python.org/3/library/asyncio.html`
[3]`https://websockets.readthedocs.io/en/stable/`

their orientation, scale factor, and camera transforms, as shown in Figure 4.2. Scale factor defines how much the model should be scaled for proper displaying on the camera. This configuration file is loaded before the actual model and has JSON file format. An example of the configuration file is shown in Figure 4.2. It is important for the configuration file to have the same name as its respective avatar GLTF file.

To load the GLTF model, the script uses ThreeJS' *GLTFLoader*[4]. After the loading of the GLTF file – an avatar, it is added to the center of the scene. Base jaw's and eyelids' rotation values are saved for future animations and the avatar's head is set as a target for the camera.

Every time a user selects different avatar, the old one is removed from the scene and the process of loading the configuration file and GLTF file is repeated with the newly selected avatar.

The scene is equipped with the simple background texture and two lights – in front of the avatar and in the back. As explained in Section 3.2.2, those settings , as well as the avatar selection, are managed by the GUI script that utilize *dat.GUI*[5] – a lightweight graphical user interface for changing variables in JavaScript.

Avatar animator also utilizes most common camera for 3D scenes – perspective camera[6]. The camera takes advantage of orbit controls[7], for the user to be able to move freely around the avatar.

With properly loaded object and obtained real world data, the scene is set and the avatar ready to be animated. This is done every time the data are received from the websocket server. To animate the avatar, the scene is being traversed to find the bone, which name corresponds with the name given in the configuration file, and change its rotation around the axis where necessary. Changing transforms of the bone other than the rotation is not necessary. Figure 4.3 shows an example of rotating the bone around its x axis.

Head and irises use the offset from the configuration file instead of the base rotation. Nevertheless, eyelids for blinking animations use base rotation as well to return to their default state when eyes are opened. Blinking is working with only one avatar (Ruby) in the current state, because other avatars do not have bones for controlling eyelids. To blink, the blink value from the data from the websocket message needs to be less than certain threshold. This threshold was manually set to 0.27. How to calculate blink value was explained in Section 3.1.4.

Updating bone transforms of the avatar every frame leaves the impression that the avatar is really moving alongside the user in the real world or in the video.

4.1.3 Websocket Server – Script

The websocket server script was implemented using python version 3.9.7. It uses *asyncio* package to handle asynchronous operations, such as handling messages. *Websockets* package is used to serve on the specified port and to broadcast messages. By default this port is set to be 8765.

Firstly, the server is created using parameters – message handler (function) and port (integer) on which it will operate. Whenever a new message is received by the server, the message handler process this message.

[4]https://threejs.org/docs/#examples/en/loaders/GLTFLoader
[5]https://github.com/dataarts/dat.gui
[6]https://threejs.org/docs/#api/en/cameras/PerspectiveCamera
[7]https://threejs.org/docs/#examples/en/controls/OrbitControls

```
{
    "bones": {
        "bone_jaw": "Jaw",
        "bone_head": "Head",
        "bone_eye_L": "EyeL",
        "bone_eye_R": "EyeR",

        "multipliers": {
            "jaw": -4,
            "head_rot": 5,
            "head_nod": 5,
            "head_turn": 5,
            "eye_L_V": 1,
            "eye_L_H": 1,
            "eye_R_V": 1,
            "eye_R_H": 1
        },

        "offsets": {
            "head_rot": 0,
            "head_nod": 0,
            "head_turn": 0,
            "eye_L_V": -2,
            "eye_L_H": -0.5,
            "eye_R_V": -2,
            "eye_R_H": -0.5
        },

        "axis": {
            "eye_H": "z",
            "eye_V": "x"
        }
    },
    "scale_factor": 7,

    "camera_position": {
        "x": 0,
        "y": 12,
        "z": 10
    },
    "camera_offsetY": 8
}
```

Figure 4.2: Example of the configuration file for the bandit avatar. It contains, bone names, bone multipliers and offsets, horizontal and vertical axis for eyes, scale factor, and camera transforms.

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

Figure 4.3: 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 base 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 the data obtained from the websocket server (how much is mouth opened).

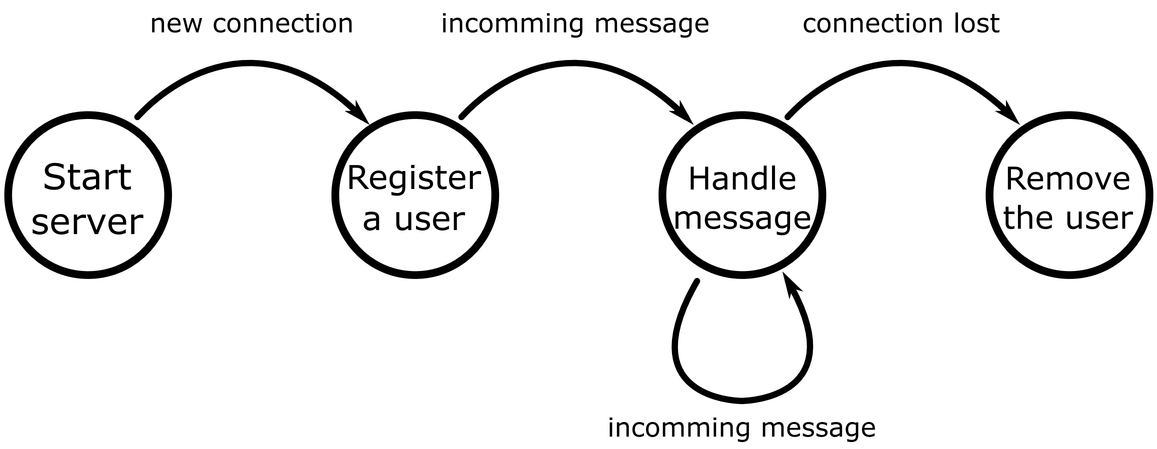


Figure 4.4: After registering a user, the continuity of the websocket connection enables the server to asynchronously handle messages for each registered user.

Message handler uses a global set of connected (registered) users (websocket clients). When there is no entry of the new websocket connection, it is added to the set. Then every message received from that connection is broadcasted to all registered users. When the connection with the client is lost, the user entry is removed from the set. Figure 4.4 demonstrates registering users and handling messages.

4.2 Flaws and Drawbacks

The solution I designed and implemented has its own drawbacks, but further work could increase precision of face tracking and make animations smoother. In this section I discuss flaws of this solution and potentially undesired behaviours.

4.2.1 Lack of Filtering in the Websocket Server

In current stage of development, the websocket server registers every client that establishes a connection with it. There is no filtering currently implemented, which means the websocket server does not verify the identity of the connected client. This could lead to connecting multiple face tracking modules to the same websocket server, which would create undesired behaviour as explained in Section 4.3.

In the worse case scenario, a completely unrelated client that sends and receive different kind of messages, could connect to the server. What could follow would depend on the connected client. It could independently send messages, containing unrelated face data, which would lead to the same behaviour of the avatar animator as in the case, where two cameras are connected.
If the client sends different message, not containing face data, the avatar animator would simply ignore them.
Nevertheless, the unrelated client could only listen to the broadcasted messages and use obtained face data for unknown purposes.

4.2.2 Jittering of Landmarks

When using Mediapipe's facemesh model for face tracking, landmarks tend to jitter (shake) from frame to frame. This is a known issue[8] of facemesh. Possible solution for this would be to implement a filter (e.g. Kalman filter) or landmarks smoothing calculator[9] that is defined in Mediapipe's Face Effect demo app.

4.2.3 Lack of Bones in Skeletons

While using skeletal animations has its advantages, as mentioned in Section 2.4.1, there is a reason to why most of the existing works use blend shapes instead. Using only few bones for the whole face can not reproduce a real facial expression. Human face has about 20 skeletal muscles[10] that control facial expressions. Few bones (head, jaw, irises and eyelids) can not control them.

Blend shapes, on the other hand, could reproduce much more expressions using blend shape interpolation method. It would be enough to create four or eight default blend shapes, and interpolate among them based on the input from face tracking. However, avatar models used in this solution are all rigged and optimized for skeletal animations.

4.3 Performance Tests

The primary testing factor was an inference speed measured in frames per second (FPS).

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[11].

Both cameras performed similarly, their inference matched at 30 FPS according to the FPS counter. However, since the 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, and the avatar is moving according to different face tracking module on each frame.

[8] https://github.com/google/mediapipe/issues/825
[9] https://github.com/google/mediapipe/tree/master/mediapipe/calculators/util
[10] https://www.kenhub.com/en/library/anatomy/the-facial-muscles
[11] https://iriun.com/

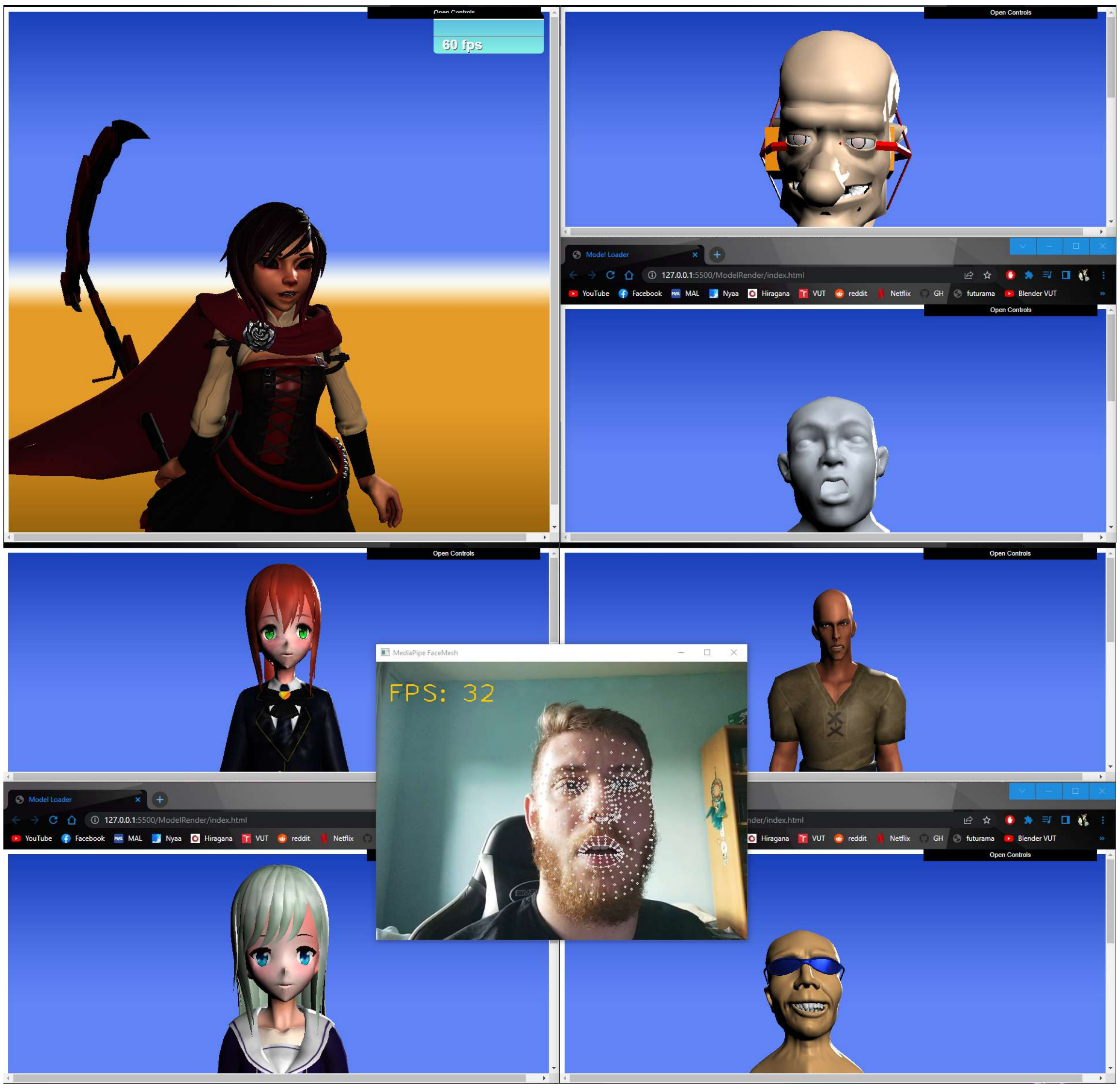


Figure 4.5: Test showing both camera FPS (32) and rendering FPS (60+), while rendering 7 different avatars.

As mentioned above, the biggest advantage of this solution in comparison to other existing solutions is the ability to animate multiple avatars simultaneously. As shown in Figure 4.5, this ability proved in tests to be working as expected (60+ FPS render speed)[12].

[12]Full resolution screenshot can be found at `https://imgur.com/UwGXeRq`

Chapter 5

Installation Guide and User Manual

To install the current version of the project, navigate to https://github.com/Junacik99/bac and clone or download the repository. Steps for installing individual modules are explained in the following sections.

5.1 Installing the Websocket Server

To install the websocket server on the device, Python version 3.9.7 or later is required. Make sure you have the correct version of Python installed using command:

```
python --version
```

If the correct version of Python is already installed, open command line/terminal and navigate to the downloaded repository (*bac-main*). Navigate to the *WebSocketServer* directory. From there issue command:

```
pip install -r requirements.txt
```

to install all required dependencies. If the face tracking module was installed before the websocket server, this step can be ignored, because all the dependencies are already installed.

To run the websocket server from the *WebSocketServer* directory, issue command:

```
python websoc.py
```

After starting the server, you should see the message: *Starting websocket server.* This means that the server is running on port 8765 and ready to serve clients. To change the port for the server, open *websoc.py* and change value of *port* to desired port number.

5.2 Installing the Face Tracking Module

To install the face tracking module on the device, Python version 3.9.7 or later is required. Make sure you have the correct version of Python installed using command:

```
python --version
```

If the correct version of Python is already installed, open command line/terminal and navigate to the downloaded project repository (*bac-main*). Navigate to the *FaceTracking* directory. From there issue command:

```
pip install -r requirements.txt
```

to install all required dependencies.

To run the face tracking from the *FaceTracking* directory, issue command:

```
python FaceTracking.py [mode]
```

where *mode* is the optional argument. If *mode* is an integer number (0 or greater), the face tracking module will look for the connected camera with index *mode*. If *mode* is anything else, the face tracking module will take it as a path to the input video file. If loading the video does not work, check the path for any typos and grammar errors, and make sure a proper version of ffmpeg or gstreamer is installed[1].

If there are more than one argument given, the face tracking module will ignore all but the first, which will be consider valid mode.

If no arguments are given, then 0 is a default mode and the face tracking module will try to read input from the default camera.

Attention! By default the address and the port of the websocket server are set to *ws://localhost:8765*. To change it, navigate to *ws_client.py* and change the value of *ws_address*.

5.3 Installing the Avatar Animator Module

Avatar animator is already deployed online and publicly available at `http://www.stud.fit.vutbr.cz/~xtakac07/`. No installation is needed, but the websocket server should be running before accessing the avatar animator. By default, it looks for the websocket server on *localhost*, port 8765. To change it, follow instructions bellow.

To install the avatar animator module on your own device ***Three.js*** library is required. Visit `https://threejs.org/` and hit the download button. This will automatically start the download of the library.

Extract the downloaded zip file and move it to the *ModelRender* directory of the *bac-main* repository. The *three.js-master* directory should be root directory for *build* and *examples* directories.

To control GUI elements the ***dat.gui*** library is required. To download *dat.gui* clone the repository from `https://github.com/dataarts/dat.gui`. If you downloaded zip file, extract it. Then the cloned or extracted repository has to be moved to the *ModelRender* directory, so the *dat.gui-master* directory contains *build* directory.

After cloning the repository and inserting *Three.js* and *dat.gui* modules inside *ModelRender* directory, the avatar animator module is ready to be deployed. To this I recommend using Visual Studio Code with Live Server extension[2]. It helps you launch a local server for development.
Alternatively, it is possible to deploy whole *ModelRender* directory on a server, such as `https://www.stud.fit.vutbr.cz` for VUT FIT students, to access avatar animator functions.

By default, the avatar animator looks for the websocket server on *localhost*, port 8765. To change it, open *index.js* script and change value of *ws_address*.

[1]`https://docs.opencv.org/4.x/dd/d43/tutorial_py_video_display.html`
[2]`https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer`

Disclaimer I do not own any of the provided assets files and they have been provided to me for free from online markets. Provided assets:

- Ruby Rose[3]
- Old man[4]
- Markus[5]
- Latifa[6]
- Rin[7]
- Pilin[8]
- Bandit[9]

[3]https://skfb.ly/6QSUK

[4]https://www.turbosquid.com/3d-models/free-blend-mode-old-man-rigged/625963

[5]https://www.turbosquid.com/3d-models/free-blend-mode-markus-sculpt/536148

[6]https://www.cgtrader.com/free-3d-models/character/woman/latifa-v2-original-vrchat-and-game-ready

[7]https://www.cgtrader.com/free-3d-models/character/woman/rin-vrc-avatar

[8]

[9]https://www.turbosquid.com/3d-models/basic-bandit-3d-1250561

Chapter 6

Conclusions

The goal of this thesis was to create a system that would animate the face of the 3D avatar in real-time based on the input from the camera/video containing human face in the image.

Based on the Mediapipe's perception pipeline and Facemesh solution for estimation of the human face geometry I was able to calculate transforms of specific bones for the avatar (head, jaw, eyeballs, and eyelids). JavaScript library Three.js allows for the avatar animator module to run right in the user's web browser, so no installation is needed. It also allows for the access to transforms of individual bones of the model, which makes it fairly easy to create skeletal animations in real-time. Due to the independence of individual modules – face tracking and avatar animator, and their interconnection via websocket server, it is possible to animate multiple avatars at the time, based on the single input.

The Mediapipe is multi-platform framework and is compatible with android devices, but further development is needed to create a standalone mobile application for the face tracking. In the present it can run devices that have installed Python version 3.9.7 or later. Nevertheless, since the avatar animator is a web-based application, it can run on mobile devices. The initial loading of the avatar lasts longer than on device with a powerful GPU, such as PC, but the animation is just as smooth as it would be on the computer.

To achieve better and more realistic visual representation of human expressions on the avatar, the avatar should have more bones for facial control, and additional computations at the face tracking module would be required as well.

Nevertheless, using skeletal animations has its advantages. There is no need to define many blend shapes for each model separately. The bone hierarchy remains the same for most humanoid avatars, and thus could be reused for multiple avatars.

Bibliography

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A. et al. TensorFlow: A System for Large-Scale Machine Learning. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, November 2016, p. 265–283. ISBN 978-1-931971-33-1. Available at: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.

[2] Ahire, A. L., Evans, A. and Blat, J. Animation on the Web: A Survey. In: *Proceedings of the 20th International Conference on 3D Web Technology*. New York, NY, USA: Association for Computing Machinery, 2015, p. 249–257. Web3D '15. DOI: 10.1145/2775292.2775298. ISBN 9781450336475. Available at: https://doi.org/10.1145/2775292.2775298.

[3] Bazarevsky, V., Kartynnik, Y., Vakunov, A., Raveendran, K. and Grundmann, M. BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs. *CoRR*. 2019, abs/1907.05047. Available at: http://arxiv.org/abs/1907.05047.

[4] Cao, C., Hou, Q. and Zhou, K. Displaced Dynamic Expression Regression for Real-Time Facial Tracking and Animation. *ACM Trans. Graph.* New York, NY, USA: Association for Computing Machinery. jul 2014, vol. 33, no. 4. DOI: 10.1145/2601097.2601204. ISSN 0730-0301. Available at: https://doi.org/10.1145/2601097.2601204.

[5] Cao, C., Weng, Y., Lin, S. and Zhou, K. 3D Shape Regression for Real-Time Facial Animation. *ACM Trans. Graph.* New York, NY, USA: Association for Computing Machinery. jul 2013, vol. 32, no. 4. DOI: 10.1145/2461912.2462012. ISSN 0730-0301. Available at: https://doi.org/10.1145/2461912.2462012.

[6] Dai, H., Cai, B., Song, J. and Zhang, D. Skeletal Animation Based on BVH Motion Data. In: *2010 2nd International Conference on Information Engineering and Computer Science*. 2010, p. 1–4. DOI: 10.1109/ICIECS.2010.5678292.

[7] Derkach, D., Ruiz, A. and Sukno, F. M. Head Pose Estimation Based on 3-D Facial Landmarks Localization and Regression. In: *2017 12th IEEE International Conference on Automatic Face Gesture Recognition (FG 2017)*. 2017, p. 820–827. DOI: 10.1109/FG.2017.104. Available at: https://doi.org/10.1109/FG.2017.104.

[8] Garolera, E. V., Llorach, G., Agenjo, J. and Blat, J. Real-time face retargeting and a face rig on the web.

[9] Garstenauer, M. and Kurka, D.-I. D. G. *Character animation in real-time*. Citeseer, 2006.

[10] Grishchenko, I., Bazarevsky, V., Bazavan, E. G., Li, N. and Mayes, J. *3D Pose Detection with MediaPipe BlazePose GHUM and TensorFlow.js* [[online]]. 2021. Available at: https://blog.tensorflow.org/2021/08/3d-pose-detection-with-mediapipe-blazepose-ghum-tfjs.html.

[11] Kapetanakis, K., Panagiotakis, S. and Malamos, A. G. HTML5 and WebSockets; Challenges in Network 3D Collaboration. In: *Proceedings of the 17th Panhellenic Conference on Informatics*. New York, NY, USA: Association for Computing Machinery, 2013, p. 33–38. PCI '13. DOI: 10.1145/2491845.2491888. ISBN 9781450319690. Available at: https://doi.org/10.1145/2491845.2491888.

[12] Kartynnik, Y., Ablavatski, A., Grishchenko, I. and Grundmann, M. Real-time Facial Surface Geometry from Monocular Video on Mobile GPUs. *CoRR*. 2019, abs/1907.06724. Available at: http://arxiv.org/abs/1907.06724.

[13] Li, H., Yu, J., Ye, Y. and Bregler, C. Realtime facial animation with on-the-fly correctives. *ACM Trans. Graph.* Citeseer. 2013, vol. 32, no. 4, p. 42–1.

[14] Lichtenauer, J. and Pantic, M. Monocular omnidirectional head motion capture in the visible light spectrum. In:. November 2011, p. 430–436. DOI: 10.1109/ICCVW.2011.6130273.

[15] Lugaresi, C., Tang, J., Nash, H., McClanahan, C., Uboweja, E. et al. MediaPipe: A Framework for Building Perception Pipelines. *CoRR*. 2019, abs/1906.08172. Available at: http://arxiv.org/abs/1906.08172.

[16] Melnikov, A. and Fette, I. *The WebSocket Protocol* [RFC 6455]. RFC Editor, december 2011. DOI: 10.17487/RFC6455. Available at: https://www.rfc-editor.org/info/rfc6455.

[17] Nakagawa, K. The Use of Live2D as an Animation Education Tool. *Bulletin of Kurashiki University of Science and the Arts*. 2017, no. 22, p. 15–21.

[18] Nordvall, B. *Down the Rabbit Hole: Hololive Myth, community, and digital geographies*. 2021.

[19] Seemann, E., Nickel, K. and Stiefelhagen, R. Head pose estimation using stereo vision for human-robot interaction. In: *Sixth IEEE International Conference on Automatic Face and Gesture Recognition, 2004. Proceedings.* 2004, p. 626–631. DOI: 10.1109/AFGR.2004.1301603.

[20] Slabaugh, G. G. Computing Euler angles from a rotation matrix. *Retrieved on August*. 1999, vol. 6, no. 2000, p. 39–63.

[21] Smilkov, D., Thorat, N., Assogba, Y., Nicholson, C., Kreeger, N. et al. TensorFlow.js: Machine Learning For The Web and Beyond. In: Talwalkar, A., Smith, V. and Zaharia, M., ed. *Proceedings of Machine Learning and Systems*. 2019, vol. 1, p. 309–321. Available at: https://proceedings.mlsys.org/paper/2019/file/1d7f7abc18fcb43975065399b0d1e48e-Paper.pdf.

[22] Srinivasan, L., Scharnagl, J. and Schilling, K. Analysis of WebSockets as the New Age Protocol for Remote Robot Tele-operation. *IFAC Proceedings Volumes*.

Firstth ed. 2013, vol. 46, no. 29, p. 83–88. DOI: https://doi.org/10.3182/20131111-3-KR-2043.00032. ISSN 1474-6670. 3rd IFAC Symposium on Telematics Applications. Available at: https://www.sciencedirect.com/science/article/pii/S1474667015343688.

[23] Toisoul, A., Kossaifi, J., Bulat, A., Tzimiropoulos, G. and Pantic, M. Estimation of continuous valence and arousal levels from faces in naturalistic conditions. *Nature Machine Intelligence*. Nature Publishing Group. 2021, vol. 3, no. 1, p. 42–50.

[24] Viola, P. and Jones, M. Rapid object detection using a boosted cascade of simple features. In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. 2001, vol. 1, p. I–I. DOI: 10.1109/CVPR.2001.990517.

[25] Weise, T., Bouaziz, S., Li, H. and Pauly, M. Realtime Performance-Based Facial Animation. *ACM Trans. Graph.* New York, NY, USA: Association for Computing Machinery. jul 2011, vol. 30, no. 4. DOI: 10.1145/2010324.1964972. ISSN 0730-0301. Available at: https://doi.org/10.1145/2010324.1964972.

[26] Yuan, L., Qu, Z., Zhao, Y., Zhang, H. and Nian, Q. A convolutional neural network based on TensorFlow for face recognition. In: *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. 2017, p. 525–529. DOI: 10.1109/IAEAC.2017.8054070.