Tampere University

Konsta Jurvanen

ANIMATION OF A SPEAKING CHATBOT

The Deep Speaking Avatar project

Bachelor's thesis
Faculty of Information Technology and Communication Sciences
Examiner: Prof. Joni Kämäräinen
November 2022

ABSTRACT

Konsta Jurvanen: Animation of a speaking chatbot
Bachelor's thesis
Tampere University
Information technology
November 2022

This thesis is a mixture of scientific research related to pre-developed technologies for animating virtual avatars and the actual development of an own application for the purpose. The topic was restricted by compatibility with other components of a chatbot pipeline, available techniques for computer graphics, and the required user experience for a Deep Speaking Avatar application. Other components in the project's pipeline were implemented by four other students as their own bachelor's theses. Two other components of the pipeline are relevant to our application, one of which is face recognition guiding the rotation of the avatar's head. The other component dependency was the output of the text-to-speech module that would time the lip movements of the avatar.

Available techniques were first discovered for facial expression modeling and face-audio synchronization. Two of them were tested, but both proved to be too restricted to be animated and rotated in real-time. Both techniques were competent in their intended aspects but fulfilling all the requirements for the Deep Speaking Avatar required lowering the standards of the avatar's realism and making an own implementation using general-purpose tools. A suitable amount of implementational freedom and technical abilities was offered by the popular 3D-rendering and development tool Unity. Another popular tool, Blender, was used to craft the 3D-mesh model that is animated and rendered in Unity.

The main desired functionalities of the avatar were achieved successfully in the specific technical conditions used to test the implementation. The built avatar can turn its head approximately in the direction of the user's face, and the lips of the avatar move at a monotonic pace when the avatar produces speech. Relative locations of the camera, user, and screen should not differ much from the used test setup. However, ideas to improve the application's adaptivity to different physical setups were discussed. A Linux environment with the other Avatar-pipeline components is required to run the Deep Speaking Avatar, and an integration script, written by another student can be used for setting up the pipeline. A good amount of computational power is required to run the whole system because many of the chatbot modules utilize heavy neural networks.

Keywords: 3D animation, lip-synchronization, chatbot, virtual face.

TIIVISTELMÄ

Konsta Jurvanen: Puhuvan keskustelurobotin animointi
Kandidaatin työ
Tampereen yliopisto
Tietotekniikka
Marraskuu 2022

Tämä työ on toteutettu osana suurempaa viiden opiskelijan kandidaatintöiden kokonaisuutta. Ihmisen kanssa kommunikointiin pystyvälle keskustelubotille täytyi toteuttaa visuaalinen näkymä tietokonegrafiikkaa hyödyntäen. Merkittävimpiä haasteita aiheutti virtuaalisten kasvojen visuaalisen ilmeen elävöittäminen muilta avatarin teknisiltä osilta saadun syötteen mukaisesti. Avatar-järjestelmän käyttäjän kasvojen tunnistukseen käytetty osa tuottaa kameran avulla kasvojen sijainnista lähes reaaliaikaista koordinaattidataa, jolla kontrolloidaan pään kääntelyä keskustelijaa kohti. Järjestelmään kuuluvan keskustelurobotin tuottaessa puhetta, virtuaalikasvojen huulia liikutetaan synkronoidusti vastaanotetun audiosyötteen kanssa.

Ongelmaa lähdettiin ratkaisemaan etsimällä valmiita teknologioita kasvojen animointiin ja huulien audio-synkronointiin. Kaikki löydetyt tekniikat osoittautivat liian rajoitetuiksi soveltuakseen reaaliaikaiseen interaktiiviseen animointiin halutussa käyttötarkoituksessa, joten riittävän mukautuvaksi pohjaksi valikoitui tunnettu pelimoottori ja 3D-kehitysympäristö Unity. Blender -työkalulla rakennettiin animointirungolla kontrolloitava kolmiulotteinen kasvomalli, joka vietiin Unityn ympäristöön animoitavaksi. Unityn ohjelmointimahdollisuuksia hyödyntäen kasvomallin animaatio saatiin yhdistettyä muihin keskustelubotin toimintoihin välittämällä prosessien välisiä ohjaustietoja yhteisiin tekstitiedostoihin tehdyillä luku- ja kirjoitusoperaatioilla.

Työn tuloksena onnistuttiin muodostamaan puheen aikana tasaiseen tahtiin huuliaan liikuttavat kasvot, jotka seuraavat sopivalla etäisyydellä liikkuvaa ihmishahmoa testatuissa olosuhteissa. Sovellus olettaa kameran olevan kohdistettu saman suuntaisesti, kuin näyttö, jolla avatar esitetään. Työn tekniset tavoitteet saatiin toteutettua, mutta realistisen kasvoanimaation saavuttaminen olisi vaatinut edistyneempiä teknologioita. Projektin tuotos kokonaisuudessaankaan ei aivan täysin vastaa ihmisen kanssa keskustelua, sillä avatar ei automaattisesti tunnista käyttäjän puheen alkua. Rajoitetusta käyttöympäristöstä huolimatta perustoiminnallisuudet toimivat mahdollisena pohjana jatkokehitykselle, ja nykyiseenkin toteutukseen sopivia jatkokehitysideoita pohdittiin.

Avainsanat: 3D-animointi, huulisynkronia, keskustelurobotti, virtuaalikasvot.

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

3D	Three dimensional
AI	Artificial intelligence
AU	Action unit
FACS	Facial action coding system
FAP	Face animation parameter
FBX	Filmbox file format
TTSI	Text-to-speech interface
VOCA	Voice Operated Character Animation
VR	Virtual reality

1. INTRODUCTION

In social interaction between two humans, words are only one of the several ways of communication. Nonverbal behavioral cues like facial expressions and postures are used to convey social signals and differentiate human-to-human communication significantly from human-to-computer interaction [21]. Facial animation has been experimentally proven to help people hear an audio signal correctly, especially in noisy conditions. The same research also indicated a slight improvement in the user experience of a simple application when a speaking avatar was included to instruct and welcome the user instead of using spare text or audio for that purpose [14]. Therefore, the field of research seems promising regarding the task of creating more immersive digital experiences. Example of a virtual avatar is shown in Figure 1.

Figure 1: A Screen capture from the Avatar application

This project is an experimental effort toward finding and combining techniques that can be utilized to build an interactive animated facial model. The model will function as a visual presentation for a speaking interactive chatbot. The goal of the work is to plant the seed for further research on animating a realistic conversational avatar. However, while

an animated person gets increasingly realistic, a challenging phenomenon called The Uncanny Valley will become a problem. The phenomenon demonstrates the difficulty of pleasing the human eye with a computer-generated character. The name of this phenomenon originates from the shape of a graph that describes the pleasantness of an artificial character related to the level of human-likeness of the character. When the character starts to approach realistic levels of human-likeness, the perceived pleasantness decreases drastically. This can be visualized as a deep “valley” between the perceived pleasantness of a human and a less realistic artificial character, as seen in Figure 2. [11]

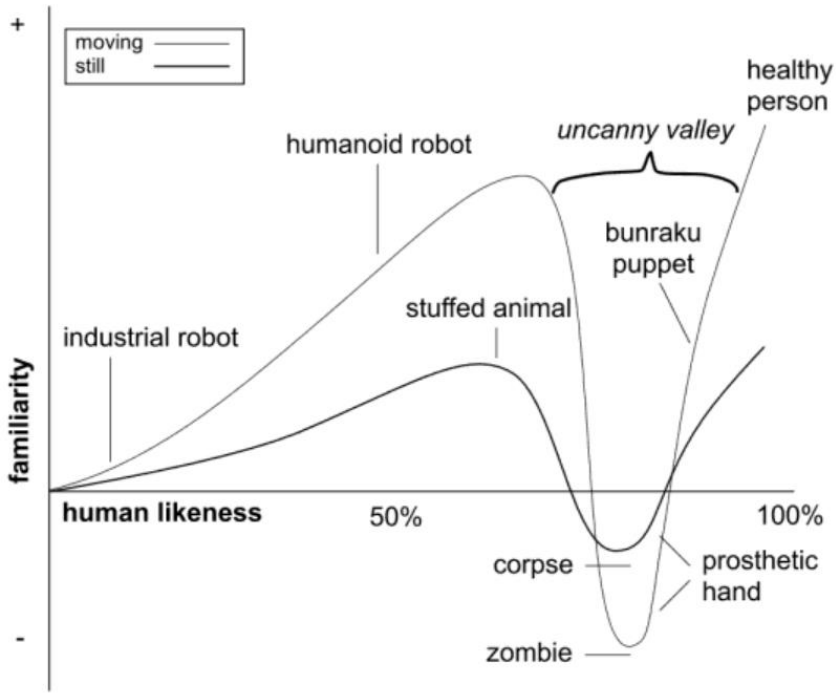

Figure 2: A graph that visualizes the unpleasantness of almost humanlike characters [15].

Even with the challenge of The Uncanny Valley ahead, research was made for techniques for realistic facial animation. The three main aspects of non-verbal communication, that are taken into consideration in this project are lip synchronization, eye contact and facial expressions. A technique to combine all these aspects in a perfectly realistic manner was not discovered for this project. However, the separate aspects of facial animation had previously been pursued individually and notable applications were found for those purposes.

The two main categories of existing work regarding the topic were discovered to be complex facial 3D models and machine learning-based lip-synchronization. Only after the development work for this thesis had been completed, it was found that the graphics

technology company NVIDIA is developing a promising tool called Omniverse Audio2Face, with realistic examples of animated speaking characters. They have already managed to create an animation system that can animate a speaking head for live audio [1]. Omniverse Audio2Face seems to meet all the requirements to be used in the Deep Speaking Avatar pipeline, but it is only briefly mentioned in this thesis because it was released and discovered after the research phase of this project.

The previous approaches that concentrated on realistic facial models have fought against the might of the Uncanny Valley, by mimicking even the internal anatomical structures of a human face. OpenFACS is a good example, as it models facial expressions with different combinations of values for the activations of so-called Action Units (AU) that control the virtual face similarly to facial muscles in real life [6]. The other approaches utilizing machine learning, seem to fall deep into the valley, with minimal facial expressions, but convincing speech-controlled lip movements [7].

Not finding a fitting technique to fulfill the requirements of the avatar led to an approach in the project that concentrates more on achieving conceptual goals with an own implementation, rather than creating an immersive user experience with advanced technologies. A walkthrough of the process of creating the avatar model with Blender and programmatically animating it in Unity is provided in Section 3. It was written to document the work for possible future development of the application. Additional objective was to inspire solutions for replacing methods, if that will be the approach taken in the future.

The complete project was conducted by a group of 5 students, each of whom implemented their own part of the Deep Speaking Avatar pipeline. The full pipeline is divided into face detection, speech-to-text conversion, text-to-text chatbot, text-to-speech conversion and the animated visual representation of the Avatar. The 3D face, which was the result of this project, communicates with the face detection module to follow the face of the person that is closest to the camera. The text-to-speech module is utilized by the animation application to get signs of when to start or stop moving the lips of the avatar, according to the timing of the speech output. The rest of the pipeline does not include any interfaces to the visual representation. Future improvements could include integration of emotional input, extracted from the text-producing module, to animate facial expressions for the avatar [18].

2. RELATED WORK

The first of the discovered styles of character animation was explicit modelling of facial structures and movements in different facial expressions. For expressing emotions on three-dimensional face models, two prominent techniques were discovered: Facial action coding systems (FACS) and the MPEG-4 framework. FACSs rely on mimicking the effect of facial muscles on facial expressions. The anatomy of a human face is modelled by dividing the face into several action units (AU), that represent the smallest muscle groups of the face that can move independently [6]. Example of such division is shown in Figure 3. Different combinations of AU values can create expressions that correspond to human emotions [6]. Examples from the OpenFACS application are shown in Figure 4.

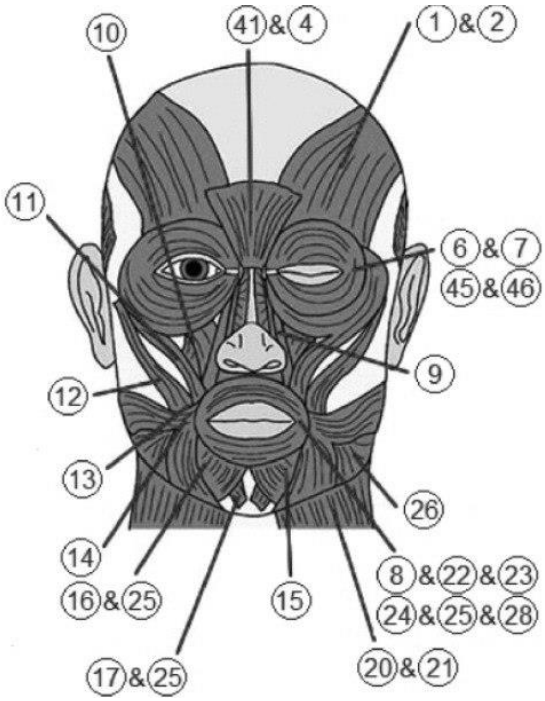

Figure 3: Action Units in HapFACS character animation system [2].

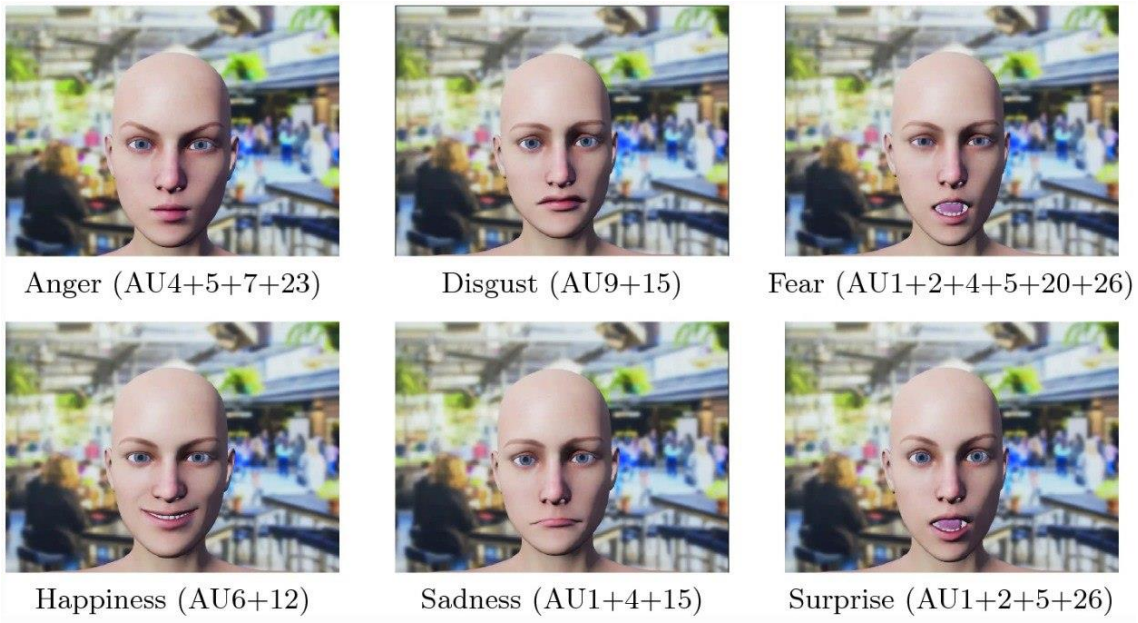

Figure 4: Different Action Unit combinations in OpenFACS [6].

The MPEG-4 framework defines a similar facial animation interface with Face animation parameters (FAP) that control the distances between feature points scattered around the face model in a specific pattern. MPEG-4 could also be integrated with the text-to-speech interface (TTSI) for lip-synchronization. [19]

These techniques would have specialized in bringing emotional expressions to the Avatar but would have been challenging to combine with real-time lip movements and head-turning. A system called HapFACS could have been integrated with the Haptek avatar system to combine lip-synchronization and expressions from Facial Action Coding System [2]. However, accessing the website for Haptek and Haptek Player failed, so we were not able to test HapFACS and Haptek for the Deep Speaking Avatar.

The less explicit methods have left the complicated anatomical structure of a human face to be theoretically hidden in the factors inside neural networks, without complicated head modeling. Machine learning algorithms could then be used to adjust the factors to find relations between spoken text and facial movements of training samples. VOCA (Voice Operated Character Animation) is an example of this kind of approach for lip-synchronization and it was one of the most potent techniques assessed for the Deep Speaking Avatar but fell short in processing time and output format for the purpose.

The implementation of VOCA utilizes a version of the DeepSpeech -deep learning model for first converting speech audio data to probabilities of characters. Those probabilities are used as features for an encoder network, consisting of four convolutional layers and two fully connected layers. The convolutions are performed over the time dimension for finding temporal connections between the character features. The result from the latter

fully connected layer is fed to a final linear layer that produces a displacement array for the vertices of a facial mesh model. When the displacements are added to a neutral mesh template, the changes in the face position can be presented by rendering the resulting mesh. The processing time of this model was too slow for real-time rendering, which led to abandoning it as an option for the Deep Speaking Avatar. However, VOCA is available as an open-source project, so it could be optimized by at least omitting the DeepSpeech part and using the output of the text-producing module directly as features for the VOCA network. The VOCA pipeline is visualized in Figure 5. [7]

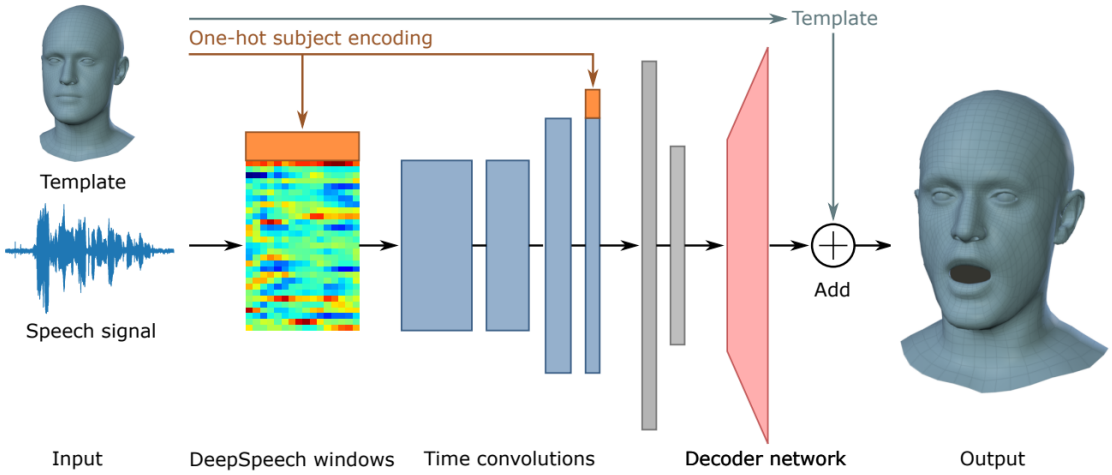

Figure 5: Visualization of the VOCA network architecture [7].

Only after explicitly constructing an avatar for the project, we discovered that NVIDIA had recently developed an industrial-level tool called Audio2Face for audio-to-face synchronization as part of their 3D creation platform Omniverse. The tool offers the possibility to animate characters in real-time and allows including facial expressions corresponding to customizable levels of emotions. The tool is also capable of automatically inferring emotions of speech audio and creating animations accordingly. Omniverse Audio2Face is based on deep learning, and as the demos show impressive results, it seems to be the future of development in the field of research. [13]

3. METHOD

Requirements to build an interactive speech-controlled avatar are the following. Firstly, the avatar requires a 3-dimensional computer graphics model with rotating eyes and transformable skin mesh. Secondly, the system must adapt its behavior according to the information from the other modules. Such information includes the position of the user and the timing of the avatar's speech.

The first of the previously mentioned requirements was fulfilled by creating the avatar model with Blender, a free, open-source computer graphics software for 3D creation. The latter requirement was met by exporting the 3D model from Blender and animating it using the real-time 3D development platform Unity, with the help of its scripting system and a feature called Animation Rigging, that allows controlling the model during the execution of the program.

The Unity project files, and the Blender model are available in Github: https://github.com/konstajurvanen/AvatarFace. The repository includes all the required files to build an executable file for the Avatar face application in Unity.

3.1 3D model of the avatar face

The final model created has a head bone that controls the rotation of the head, and all the other bones will follow the movements of the bone. A lower hierarchy level includes eye bones that can rotate independently around their inside joints. The inside joints of the eye bones follow the movements of the head bone. The meshes of the eyes are separate from the head's mesh, so the eye bones can control their rotation without impacting the head's mesh. Another bone in the lower hierarchy level is a jawbone to animate the lower lip of the avatar when speaking. The joint in the back of the neck follows the head bone, and the other tip of the bone affects stretchy bones in the lower lip to animate it. Lip bones on the edges are stationary to the head bone, but the rest of the lower lip will move with the jawbone. The complete model is visualized in Figure 6.

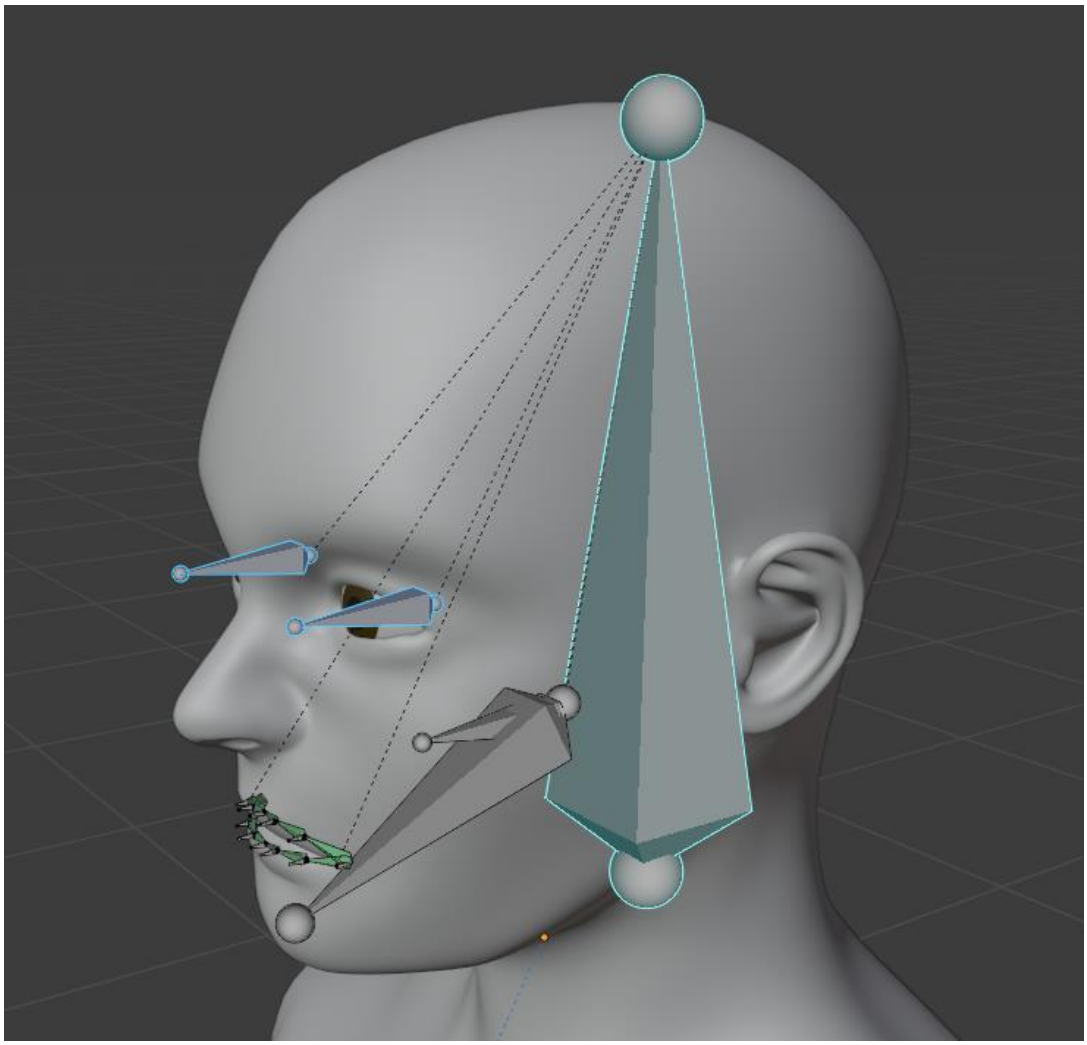

Figure 6: *Armature and mesh of the avatar.*

3.1.1 Bone structure and skin modeling in Blender

In Blender, the geometrical structure of a 3D model is defined by meshes that consist of vertices, edges, and polygonal faces. Vertices are simply points in the 3D coordinate system stored as an array. Edges connect the vertices and the area between a closed configuration of these edges defines a face. A face is often the only visible part of the mesh when it is rendered, and its visual appearance is defined by the material that is assigned to it [4]. The components forming a mesh are visualized in Figure 7.

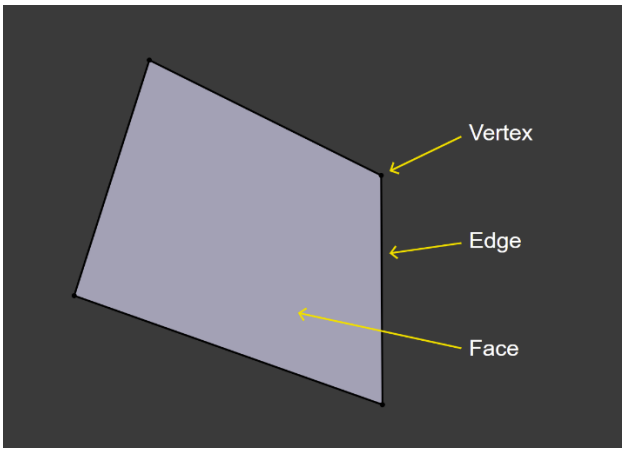

Figure 7: *Structure of a mesh [4].*

When animating a Blender model, the mesh of the model needs to have rules to adjust the geometry into the desired shapes, positions, and rotations. These rules can be defined by a method called rigging. Rigging the model requires constructing an armature of bones that can control the shape and position of the mesh. A bone is defined by the position of two joints: the tip and the root. The joints are connected by the body of the bone. A bone can be attached to another bone by their joints and the bones can be rotated around joints. The structure of a bone is illustrated in Figure 8.

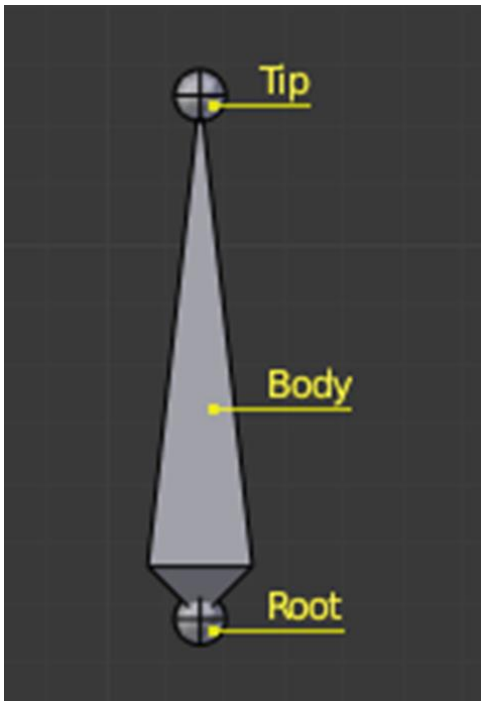

Figure 8: *Elements of a bone in Blender [4].*

Bones in Blender behave similarly as they do in real-world skeletons. The vertices of a mesh can be set to follow the transformations of a bone that is attached to another bone. The bones form a hierarchy that is defined by parent-child relationships between the

different bones of an armature. For example, in the Blender model of the avatar's face, the head bone is the highest in the hierarchy, because all the smaller bones of the model need to follow the rotations of the face. Unlike in real-world skeletons, bones in Blender do not have to be directly connected by their joints to affect other bones' positions. A parent-child relationship can be defined to be relative, so the offset between the bones' positions will be preserved when the pose is modified. [4]

Bones can be categorized into two groups by their behavior: control bones and deforming bones. Control bones are not strictly connected to meshes, but their job is to construct the hierarchical structure that will allow the control of the deforming bones in the desired manner. Deforming bones are responsible for controlling the mesh and they can also stretch between control bones when the pose of the armature is modified. The strength of the deforming bones' influence on vertices is defined by the weight that is assigned to each vertex or vertex group regarding a certain bone [4]**.** In Figure 9, the deforming bones of the avatar's lips are seen as green, and they are constrained to follow the grey control bones.

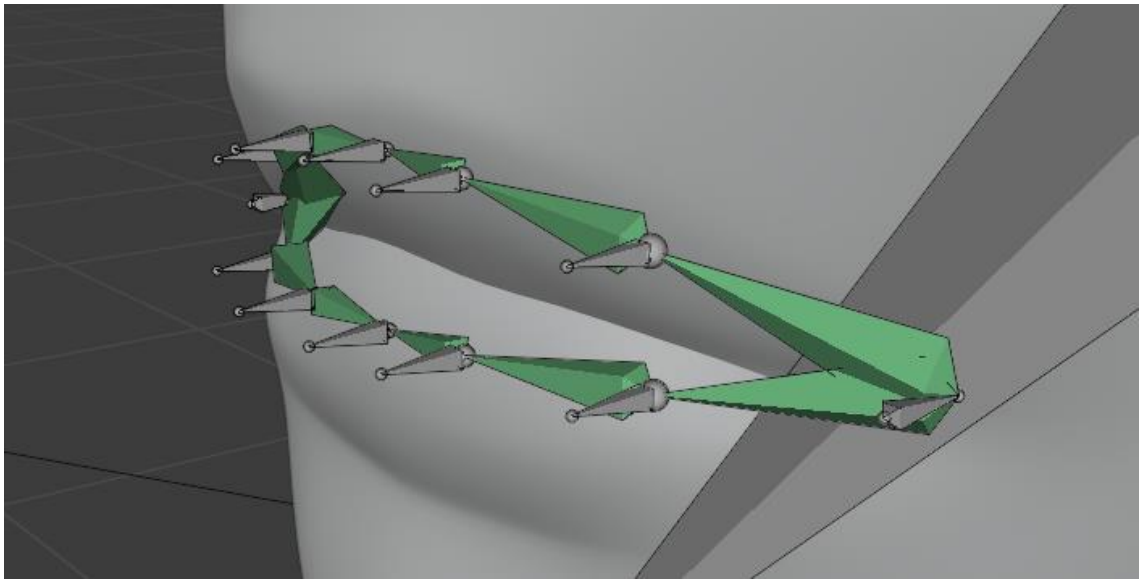

Figure 9: *Mouth armature of the avatar's Blender model.*

3.1.2 Blender workflow

The building process of the avatar's 3D model was started by downloading a premade Blender model of a head from https://blendswap.com/blend/11745. To make modifications to the model, the layout window can be opened by pressing the + -icon next to the tabs for different workspaces and then selecting *General->Layout.* The important parts

of the model are the meshes for the head, the eyes, and the background plane in *Collection 1*. The file includes also lights and a camera, but those did not work when imported to Unity, so they can be removed from the Blender model.

The right eye of the model is generated by mirroring the left eye mesh along the x-axis. To allow the eyes to be rotated separately, the mirrored right eye was removed. The origin of the eye is originally located in the middle of the head, and it was adjusted to the middle of the eyeball so the eyeball would not move around the head when rotated. The left eye was then duplicated and moved to the right eye socket. After these modifications, the process of building the armature was started. The complete armature of the Blender model is shown in Figure 6.

A head bone was added to become the highest bone in the bone hierarchy, located in the center of the avatar's head mesh. An eye bone for each side was added as children of the head bone to control the eye rotations. The parent-child relationships are visualized in Blender by the dotted lines connecting the bones, as seen in Figure 6. The head bone and the eye bones were implemented with the help of a tutorial video on YouTube [16].

The armature structure to control the lips of the avatar's mesh consists of 12 small control bones in the lip area, that are connected by 12 deforming bones as shown in Figure 9. The five control bones in the upper lip of the avatar are children of the mouth bone that has the head bone as its parent. However, the upper lip rig is not necessary as animating only the lower lip resulted in a more realistic-looking animation. The lower lip is controlled by the jawbone which has the five control bones of the lower lip as its children. The control bones in the lip corners on each side are inherited straight from the head bone as they are not supposed to move on the mouth animation, but they function as a hinge for the outermost deforming bones. The control bones on the right side of the avatar's lips, from the perspective of the avatar, are mirrored on the x-axis from the left side bones. The deforming bones connecting the control bones are constrained by Stretch To -constraints chained from the outermost bones towards the center of the lips. A YouTube tutorial was used as an example for the mouth rig [3].

The meshes of the eyes were connected to the eye bones by simply inheriting each eye mesh from the corresponding eye bone. This was possible because there is only one bone controlling each mesh. The deforming bones of the lips, however, all contribute into controlling their nearby areas in the mesh. Therefore, the effect of each deforming bone on each vertex, must be explicitly defined. Blender offers an option for calculating automatic weights for the *Armature Deform* operation. The automatic weighting operation

divides the mesh into vertex groups for each deforming bone and calculates a weight, that a particular bone has on a vertex group, based on the distance between the bone and the vertices [4].

The armature deform operation with automatic weights was utilized for the avatar, but further modifications were required as the mouth armature ended up controlling vertices from too large area and the eye bones were assigned to modify the head mesh. The Weight Paint mode was used to reduce the weight of the lip bones on unwanted parts of the mesh, and to clear all the weights of the eye bones on the head mesh. After all the previously described operations, the model is ready to be exported from Blender.

3.2 Face model rendering and scripting in Unity

The avatar is controlled by the actions of the person, who is talking with it so the animation cannot be completely predefined. The application needs scripts to define how the model should respond to the actions of the user [20]. This also means that the avatar must be rendered on the fly when the application is running, and the desired animation is being defined. Rendering means computationally producing a visual on-screen representation from a computer graphics model [10].

A simple solution for the requirements described in the previous paragraph was offered by the real-time development platform, Unity. Games require similar real-time rendering as the avatar, so the Unity game engine allows building an application for the avatar as a simple game that is figuratively played by the chatbot. The Unity scripting system allows the required user interactions to be implemented with programs written in C# (programming language), which Unity supports natively [20].

The 3D model created in Blender was imported to Unity, after which it will be stored in the Assets folder of the Unity project and could be seen in the Project window of the Unity user interface. The asset was added to the scene by dragging it from the Project window, either to the Hierarchy window or the Scene view [20].

The model was exported from Blender as an FBX file with the default export settings. A new Unity project for the avatar was created from Unity Hub with the 3D template for Unity projects. The template contains a camera and a directional light that were retained in the scene. The Blender FBX file was imported and added to the scene.

3.2.1 Animation Rigging

To control the animation of the 3D model imported from Blender, constraints to the avatar's armature were added with a method called Animation Rigging. This requires the Animation Rigging package to be installed from the Unity Package Manager. The package includes several constraint components that can be attached to a game object for different animation techniques. GameObject is a class representing objects that can exist in a scene. The component used for defining the avatar animations was the Multi-Aim Constraint, which rotates the attached object towards an object that is set as the source [20]. The source objects' positions are controlled according to user interactions with a separate C# script attached to each source object.

The Animation rigging features are implemented by adding a Rig Builder component to the root of the avatar's armature from the Inspector window. The game objects required for controlling the animations are added to the armature as a separate animation rig called *TargetRig*. The created animation rig is added as rig layer to the Rig Builder component to allow the animation rig to control the armature. The root of the animation rig should be on the same level as the root of the avatar's bone structure in the armature hierarchy [20].

The target rig includes two target objects as its children, that are used as source objects for the avatar animation. A ball-shaped *Effector* is added to visually represent the target that the eyes of the avatar should follow. The other source object for jaw movements is visualized by a box-shaped *Effector*, and the object controls the jawbone of the avatar for lip movements. The effectors will be invisible when the scene is rendered [20]. The target objects are controlled by C# -scripts attached to them, as explained in Section 3.2.2.

The *Target* -object is followed by multi-aim constraints of three separate bones in the armature. A head-aim object in the *Target* is responsible for applying the multi-aim constraint to the head bone, which allows controlling the rotation of the avatar's face. The eyes are controlled with a multi-aim constraint on each eye-aim, having the *Target* as the source object. Small offset was added on the Z-axis for the eye-aim components in opposite directions. This was done to make the focus of the eyes appear more natural.

The weight of the constraint for the head-aim was reduced to 0.65 to make the rotation of the face more subtle than the rotation of the avatar's eyes. This way the eye bones can follow the same target as the head bone, without making it feel like the avatar is turning its head away from the user. The configuration of the head bone is presented in

Figure 10. The eye bones were constrained to follow the *Target* object with a full weight of 1.0, so the eyes will rotate directly towards the target. The head bone is pointing upwards along the Y-axis, so that should be set as the up axis for the multi-aim constraints. The front side of the head bone is facing towards the Z-axis, so that should be set as the aim axis for the multi-aim constraint. The same logic was used to define the aim axis (Y) and the up axis (X) for each eye aim constraint.

A separate jaw-target object was added to the animation rig to control the lip movements when the avatar is speaking. The multi-aim constraint for the jawbone is added to a jaw-aim object with corresponding source object in the target rig. The script attached to the jaw-target defines an oscillating movement pattern that is defined by a sine function on the Y-axis position. Volume of the oscillation was toned by experimenting with different weights for the multi-aim constraint of the jaw-aim object. The jawbone only controls the lower lip of the avatar, and the movement of the lip shall happen only around the X-axis. This was achieved by ticking off Y and Z from the constrained axes in the settings of the multi-aim constraint attached to the jaw-aim.

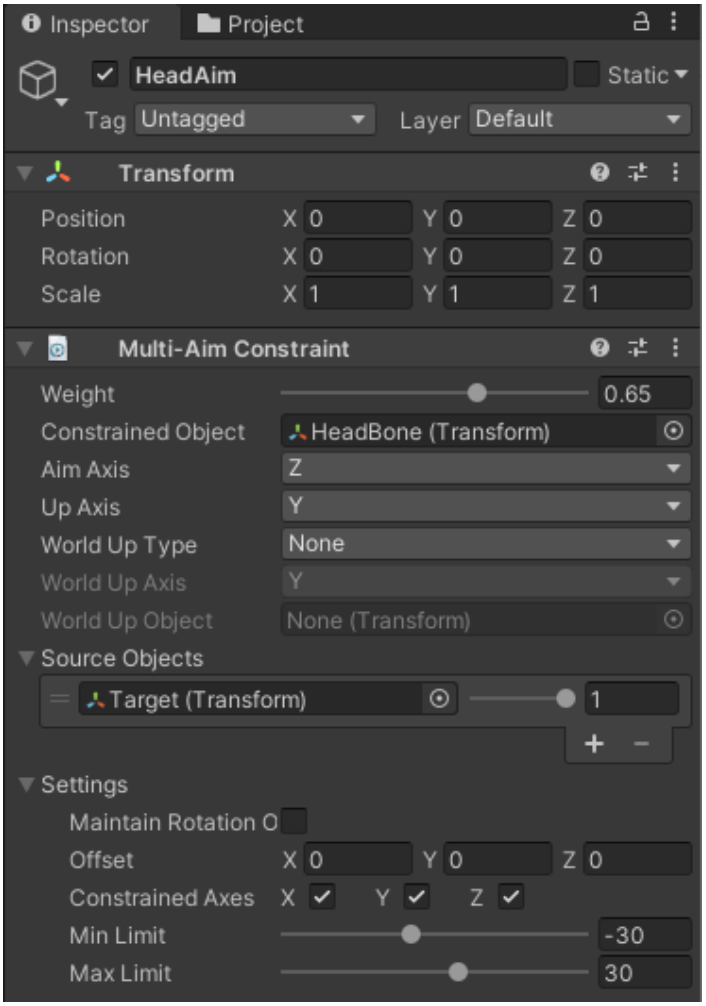

Figure 10: *Properties of the head's aim object in Unity.*

3.2.2 Scripting in Unity

Unity supports the C# -programming language natively which allows the developer to create unique gameplay features for their games [20]. This project takes advantage of the scripting system by integrating the face detection system and the speech production module of the avatar project to control the avatar's face in correlation with their output. The integration is based on using separate input-output files where the other modules constantly write the required information, and the scripts of the Unity application collect the information by reading the files. The output from the speech production module to would be the character '1' if the character is speaking and '0' if not. The face detection module updates a text file containing the image coordinates of the biggest detected face in the camera input.

In Unity, a new script file can be created for a game object by adding a new script as a component from the inspector window of the game object. Unity will open a template script for a new class in a text editor. The template script structure is shown in Figure 11. The new class inherits from MonoBehaviour, which is a built-in class that functions as the interface to the internal Unity system. The class template defines two empty functions: *Start* and *Update*. The Unity engine calls the *Start* function before any frame updates, which allows performing initializations like creating a reference to another game object in the game scene. The *Update* function is called once per frame, and it is responsible for changing the position, state, or behavior of objects in the game before each frame is rendered. [20]

```
using UnityEngine;
using System.Collections;

public class MainPlayer : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```

Figure 11: *The template class for Unity scripts [20].*

If the position of an object is changed in the *Update* function like in the following line of code:

```
Transform.position = new Vector3(0f, 0f, 0f);,
```

the object will immediately move to the center of the scene coordinate system, without any transition animation. To make the movements of the avatar look smooth, coroutines were utilized to write asynchronous code that is responsible for moving the target objects, that were created in the animation rigging procedure. Coroutines are functions that can be executed over a sequence of frames, by pausing the execution between frames by a yield return statement. In C#, coroutines are defined by functions with the return type of *IEnumerator*. The execution of a coroutine must be started by calling the *StartCoroutine* function, and either the name of the *IEnumerator* function or a reference to an instance of the *IEnumerator* function should be given as a parameter to the function. A coroutine can also be interrupted with the *StopCoroutine* function, so conflicting coroutines can be avoided [20].

The avatar has two main tasks implemented with the help of the Unity scripting system: controlling the face and eye -rotations to follow the user's face and moving the lips when the avatar is speaking. Both tasks require a text file to be read to acquire the information needed for the real-time changes in the avatar's behavior. For that reason, a new class called *Util* was created in a new script file attached to an empty game object. The class is defined as static and public, so the class member method called *Readfile* can be utilized in other scripts by importing the class with the "using static" -directive [8].

The task of following the face of the user is solved by first reading a text file that contains the coordinates of a box surrounding the face of the user. The coordinates are in the following format: $x_{min}; y_{min}; x_{max}; y_{max}$, where each value is a float between the values 0.0 and 1.0. The values of x_{min} and x_{max} indicate the left and the right boundaries of the box in the horizontal direction relative to the image width in pixels. Respectively, y_{min} and y_{max} define the upper and lower boundaries of the box in the vertical direction. The center point of the box is calculated after the coordinate string has been parsed into an array of floats.

After having the two floats for the center point of the face detection result, the point in the relative image coordinates must be converted into a position in the three-dimensional scene. The converted point should make the avatar appear to be turning towards the user in the real world. The conversion was implemented by defining the width and the height of an imaginary scene coordinate system so that it would form a plane in the virtual

world. The size of the virtual plane was tuned to approximately match the size of the camera view.

The scale of the conversion had to be calibrated experimentally since there are several ambiguous parameters affecting the relations between the Unity scene and the real-world setup, which would make mathematical calibration extremely complicated. Examples of these parameters are the image size of the camera, the positions of the camera, screen, and user, and the relation between units of measurement in Unity and the real world.

The image coordinate system of the face detection was converted to have the origin in the position: $x = 0.5, y = 0.5$, which is the center of the image. This definition was made based on the assumption that the camera is facing the user from approximately the same direction as the screen, on which the avatar is displayed. The conversion allows the position of the virtual camera in the Unity scene to be used as the origin for the scene coordinate system since the virtual camera is placed directly in front of the avatar in the Unity scene.

To find the position of a target in the converted image coordinate system, the displacement between the origin and a detected face in image coordinates was calculated in both dimensions. Now that the displacement is a value between -0.5 and 0.5, it can be scaled to the scene coordinate system by multiplying it by the scene width or height. The desired position of the target in the scene coordinate system can now be found by adding the displacement in both dimensions to the scene position of the camera. As an example, the scene x-coordinate is calculated as follows, with C#:

```
// Transform coordinate system
float xImgDisplacement = x - COORD_CENTER_X;
// Scale the displacement to scene coordinates
float xDisplacement = xImgDisplacement * SCENE_WIDTH;
// Use the camera's position to find the correct scene position
float sceneX = cameraCenter.x + xDisplacement;,
```

where *x* is the horizontal image coordinate of a detected face with a value between 0.0 and 1.0. The float *sceneX* is the value to be set for the target object's transform's x-coordinate in the Unity scene.

The z-coordinate was explicitly defined to be a constant that makes the target following procedure work properly when the user is from one to two meters away from the screen. The calculation was not done computationally from input, because the distance between the chatter and the camera is unknown, although the size of the surrounding box of the face detection can provide rough estimations.

Once the position for the target is defined, a coroutine, that moves the target towards the desired position frame-by-frame, is started. The coroutine utilizes a built-in method *MoveTowards* of *Vector3*, which is a struct for representing three-dimensional vectors in Unity. Another built-in method *Time.deltaTime* was used together with an explicitly defined moving speed, to calculate a suitable distance for the target to move within each frame update. The method, *Time.deltaTime,* returns the time in seconds from the last frame update, and the moving speed describes how many distance units the target should move in the scene in one second. This way, a steady movement for the target is achieved regardless of changes in frame rate, when *Time.deltaTime* is multiplied by the moving speed. [20]

4. WORKING PROCESS

While searching for available facial animation techniques like OpenFACS and VOCA, the implementations were tested using tutorials for simple applications. OpenFACS was installed on a Linux environment and the dependencies were installed according to the official tutorial in GitHub [17]. Ways to rotate an avatar face with the application were not found during the experiment or by searching the internet. A static face, the expression of which could be explicitly modified by the application interface, was the furthest point that was achieved during the experiment. Head and eye rotation were therefore still unreachable. The OpenFACS user interface can be seen in Figure 12.

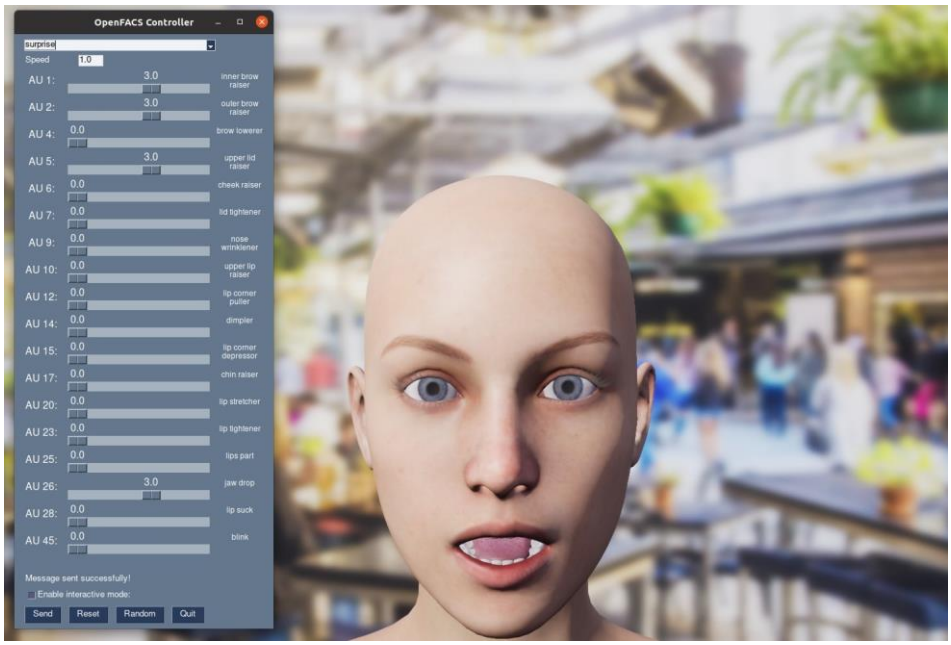

Figure 12: Image of the graphical user interface of OpenFACS [17].

To add animation to a virtual face the voice-operated character animation tool VOCA was tested similarly. The official tutorial in GitHub was used to set up the Linux environment [5]. The steps included installing the separate mesh processing libraries with instructions from another GitHub page [12]. Now VOCA was able to render a face and the voice operating was then tested.

An audio capturing tool for Linux was installed and used to capture some test audio with a laptop. Short audio snippets were recorded in English and Finnish and moved to the folder where VOCA was installed. The tool was then used to produce videos from the audio snippets and the default facial mesh from a library called FLAME that was included in the installed mesh processing libraries [12]. The recorded audio clips had poor quality

due to using the bad internal microphone of the laptop, so lip animation in the videos was not very realistic either. The results could have had higher quality with better audio capturing device. However, during the experiments, it occurred that producing a constant video stream with the tool would not be possible since the output is static video and producing it for audio of few seconds would take up to several minutes. So, using better microphones with VOCA was not considered due to the impracticalities of VOCA regarding real-time-animation. Demonstration of VOCA output can be seen in Figure 13.

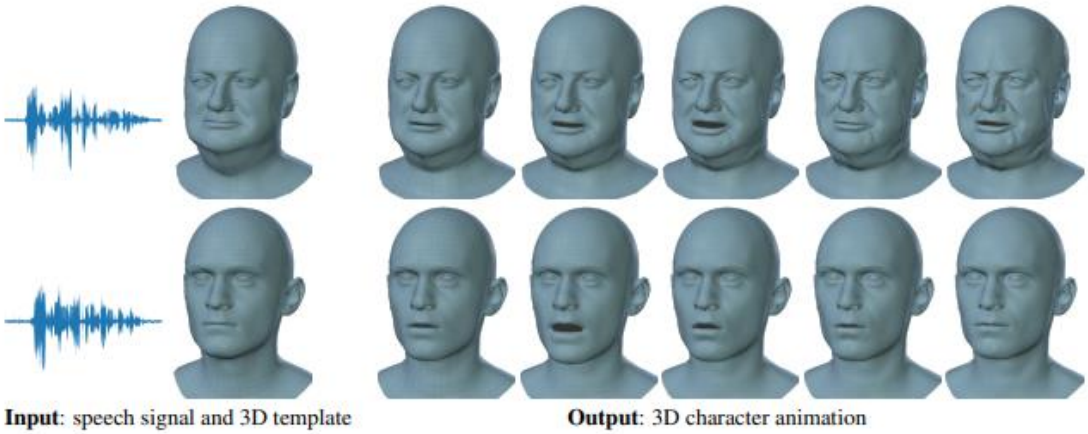

Figure 13: Input and results of VOCA visualized [7].

The research with the final solution to the problem was started by researching tools for creating 3D models. Blender was considered a good option due to its popularity and ability to export models. After researching popular solutions for mesh animations like Unreal Engine, Unity, and lower-level techniques like OpenGL or WebGL, Unity was noted to be the best fitting to the situation. Unity's compatibility with Blender, and the simple scripting mechanism of Unity were the deciding factors that lead to the decision [20]. I as the author had not studied any computer graphics courses, so using OpenGL or WebGL would have required acquiring competence in a completely unfamiliar field of computing.

The 3D model was built with Blender and then imported to Unity, but the combination of the tools ended up not being so practical during development. The reason for the impracticalities was the inability of modifying the imported 3D models in Unity. As one would expect, the first version of the model was not perfect, and many modifications had to be made in Blender later. This was time consuming since every new modified model had to be imported and re-connected with the animation rig in Unity. Other properties of Unity, like the scripting system proved to be easy to learn and compatible with the requirements of the avatar project [20].

When the first version of the Unity application was ready, the project was compiled into an executable file for Linux platforms. The executable was moved to a portable USB drive and transferred to the computer that was to be used with the complete Avatar project. Test text files for face coordinates and speaking-quiet-boolean value were produced manually, and the application was launched in the correct directory.

5. FUTURE WORK

The created avatar can only follow the user and move its lips when speaking. Many features are still missing from a realistic virtual chatting experience, like expressing emotions of the avatar and accurate synchronization of lip movements with speech. This leaves room for improvement in the future, but significant changes could also require work on the other modules of the Deep Speaking Avatar pipeline. More subtle fixes could include, for example, adding variability to the lip movement pattern of the avatar by including another layer of sine-like oscillation to the speed of the lip movements.

The impracticalities related to the problem of integrating an avatar presented on a two-dimensional screen into a three-dimensional conversational setup could be overcome by developing a version of the avatar application for virtual reality (VR) platforms. Unity projects can be configured for VR platforms, so parts of the current Unity application could possibly be utilized in a VR implementation [20]. This would remove the need for laborious scale calibrations between different coordinate systems by technically bringing the user of the avatar into the same scene with the avatar.

If the system shall not use the VR technology for future work, some improvements could be made for the real-world – game scene conversion, by integrating a distance sensor into the system to allow three-dimensional spatial data to be transferred. Distance from the user could be roughly estimated from the size of the box that is received as output from the face detection module.

Another option for the implementation would be using OpenGL instead of Unity and Blender, which would offer more freedom with the solutions. However, the lower-level tools would also require more work and knowledge about computer graphics. To improve the portability of the avatar face application, web-based computer graphics solutions could be used. WebGL 2 is a client-based real-time rendering library that is written in JavaScript, so it is compatible with the web browsers of most desktop and mobile devices [10]. If developing the complete graphics pipeline with WebGL would not be desired, The Unity project could be built for the WebGL platform and presented with a loader written in JavaScript.

A web implementation would open possibilities to separate the rest of the Deep Speaking avatar pipeline into a backend application that could be run on its own server. However, considering the integration with the rest of the Deep Speaking Avatar pipeline, a couple of technical difficulties would have to be solved. One challenge would be integrating the

user’s web camera and microphone as input devices for the frontend application. Another problem would be effectively exchanging real-time data through internet protocols between the applications [9].

6. CONCLUSIONS

During this work, the challenge of creating a realistic and interactive virtual face has proven to be very technically demanding. Since the project was completed as a bachelor's thesis, the amount of effort and time that could be put into the work was limited. Considering these two facts, the results achieved in this research were reasonable. The minimal desired features were achieved and opportunities to improve on the subject were discovered. The work on the subject is planned to be continued by upcoming thesis writers, and this research should provide helpful background for that.

The project was quite a dive into the unknown since the work was started without any prior information on the technical and practical challenges of the subject. I, as the author, did not have any experience in computer graphics and animation, or even the C# programming language that ended up being used in the system. I also had to make myself familiar with the Linux operating system when installing and setting up environments for the experimented techniques (OpenFACS and VOCA). Despite the lack of knowledge beforehand, a workable Unity application was developed to be used to demonstrate the Deep Speaking Avatar project. A still image of the final product is shown in Figure 1.

This research has helped in making the subject easier to approach by discovering the main problems to be solved and possible solutions for those problems: The main future challenge remains in combining features of the explored solutions in a way that the best abilities of each solution could be combined. Looking at the topic later during the writing of this thesis, the smartest approach in the development of the avatar would most likely be waiting for the big companies, like NVIDIA with their Omniverse Audio2Face, to fully release their implementations, so industrial-level facial animation could be utilized in the Deep Speaking Avatar pipeline [13].

REFERENCES

[1] Albawaba (London) Ltd. (2021) United States: NVIDIA Announces Platform for Creating AI Avatars. 9 November. Available at: https://www.proquest.com/docview/2595323657/fulltext/420BDF8B207A4CF7PQ/1 (Accessed: 9.9.2022).

[2] Amini R., Lisetti C. and Ruiz G., "HapFACS 3.0: FACS-Based Facial Expression Generator for 3D Speaking Virtual Characters," in IEEE Transactions on Affective Computing, vol. 6, no. 4, pp. 348-360, 1 Oct.-Dec. 2015, https://doi.org/10.1109/TAFFC.2015.2432794 (Accessed: 15.5.2021).

[3] Bernier, J. (2016) Blender Mouth Rigging Tutorial (EN - ES subtitles). 12 December. Available at: https://www.youtube.com/watch?v=Map_ro-xUwg (Accessed: 21.4.2021).

[4] Blender 2.92 Reference Manual (no date) Available at: https://docs.blender.org/manual/en/latest/index.html#blender-blender-version-reference-manual (Accessed: 11.5.2021).

[5] Bolkart, T. (no date) VOCA. Available at: https://github.com/TimoBolkart/voca (Accessed: 4.9.2022).

[6] Cuculo V. and D'Amelio A. (2019) OpenFACS: An Open Source FACS-Based 3D Face Animation System. In: Zhao Y., Barnes N., Chen B., Westermann R., Kong X., Lin C. (eds) Image and Graphics. ICIG 2019. Lecture Notes in Computer Science, vol 11902. Springer, Cham. https://doi.org/10.1007/978-3-030-34110-7_20 (Accessed: 11.9.2022).

[7] Cudeiro D. et al. "Capture, Learning, and Synthesis of 3D Speaking Styles" in IEEE International Conference on Computer Vision and Pattern Recognition (CVPR) 2019. https://ps.is.tuebingen.mpg.de/publications/voca2019 (Accessed: 9.9.2022).

[8] C# reference. (no date) Available at: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/ (Accessed: 11.5.2021).

[9] Fiedler G. (2017) Why can't I send UDP packets from a browser? Available at: https://gafferongames.com/post/why_cant_i_send_udp_packets_from_a_browser/ (Accessed 11.9.2022).

[10] Ghayour F. and Diego C. Real-Time 3D Graphics with WebGL 2: Build Interactive 3D Applications with JavaScript and WebGL 2 (OpenGL ES 3. 0), 2nd Edition. Birmingham: Packt Publishing, Limited, 2018. https://learning.oreilly.com/library/view/real-time-3d-graphics/9781788629690/ (Accessed: 10.5.2021).

[11] Masahiro, M. (2012) "The Uncanny Valley: The Original Essay by Masahiro Mori". IEEE Spectrum, 12 June. Available at: https://spectrum.ieee.org/the-uncanny-valley (Accessed: 8 September 2022).

[12] MPI-IS. (no date) Mesh. Available at: https://github.com/MPI-IS/mesh (Accessed: 4.9.2021).

[13] NVIDIA (2022) Omniverse Audio2Face. Available at: https://www.nvidia.com/en-us/omniverse/apps/audio2face/ (Accessed: 9 September 2022).

[14] Pandzic, I., Ostermann, J. & Millen, D. User evaluation: Synthetic talking faces for interactive services. The Visual Computer 15, 330–340 (1999). https://doi.org/10.1007/s003710050182 (Accessed: 16.5.2021).

[15] Parke, Frederic I. & Waters, K. Computer facial animation. CRC press, 2008. https://books.google.fi/books?id=yEzrBgAAQBAJ&lpg=PP1&ots=nud6m7meDy&dq=Parke%2C%20F.I.%2C%20Waters%2C%20K.%3A%20Computer%20Facial%20Animation.%20AK%20Peters%2FCRC%20Press%20(2008)&lr&pg=PA6#v=onepage&q&f=false (Accessed 5.4.2021).

[16] Pearson, D. (2014) Eye Rigging Tutorial in Blender. 26 February. Available at: https://www.youtube.com/watch?v=pzfKz8NX1rQ (Accessed: 27.4.2021).

[17] Phuselab. (no date) OpenFACS. Available at: https://github.com/phuselab/openFACS (Accessed: 4.9.2022).

[18] Shelke, N., Deshpande, S., Thakare, V. (2017). Approach for Emotion Extraction from Text. In: Satapathy, S., Bhateja, V., Udgata, S., Pattnaik, P. (eds) Proceedings of the 5th International Conference on Frontiers in Intelligent Computing: Theory and Applications. Advances in Intelligent Systems and Computing, vol 516. Springer, Singapore. https://doi-org.libproxy.tuni.fi/10.1007/978-981-10-3156-4_70 (Accessed: 8.9.2022).

[19] Tekalp A. M., Ostermann, J., Face and 2-D mesh animation in MPEG-4, Signal Processing: Image Communication, Volume 15, Issues 4–5, Pages 387-421, 2000, https://doi.org/10.1016/S0923-5965(99)00055-7 (Accessed: 16.5.2021).

[20] Unity User Manual 2020.3 (LTS) (2021) Available at: https://docs.unity3d.com/Manual/UnityManual.html, (Accessed: 11.5.2021).

[21] Vinciarelli A. et al., "Bridging the Gap between Social Animal and Unsocial Machine: A Survey of Social Signal Processing," in IEEE Transactions on Affective Computing, vol. 3, no. 1, pp. 69-87, Jan.-March 2012, https://ieeexplore.ieee.org/document/5989788 (Accessed: 8.9.2022).