

LUDWIG-MAXIMILIANS-UNIVERSITÄT AT MÜNCHEN
Department “Institute of Informatics”
Teaching and Research Unit Media Informatics
Prof. Dr. Andreas Butz



Bachelor's Thesis

Automated Facial Rig Registration for Motion Capture

Julius Girbig
julius@girbix.de

Period of completion: August 06, 2021 to January 31, 2022
Supervisors: Dr. Sylvia Rothe and Changkun Ou
Supervising professor: Prof. Dr. Andreas Butz

Abstract

To create believable 3D animated faces, actors' facial expressions are retargeted from monocular RGB videos onto 3D character rigs. Though, the workflow requires large amounts of tedious and time-consuming manual labor, due to the psychological sensitivity and complex structure of the facial rigs. Consequently, the goal of this work was to automate the facial motion capture pipeline as much as possible. This thesis proposes a fully automated solution to register virtually any rig and proposes an unsupervised learning based algorithm capable of posing the registered rig based on a single frame of a video performance, synthesized by a deepfake algorithm. This makes the pipeline independent of the actor's external characteristics, lighting and background. A user study in the form of an expert interview was conducted to evaluate the usability and quality of the prototype design, which showed that the algorithm is able to save averagely 56% of the expert's time. The prototype's output poses are also preferred in 60% of the 30 cases, when compared to a traditional point tracking solution. The study also suggests that the algorithm is able to supersede other monocular point tracking methods if minimal manual labor is required and is more easy to use than current solutions. To the best of our knowledge, the algorithm proposed in this work is the first unsupervised image-based learning algorithm for retargeting facial animations based on a single frame of a processed RGB video to a generic 3D character, not relying on any actor-specific initialization or training.

Assignment

Automated Facial Rig Registration for Motion Capture

Problem Statement Facial motion tracking often requires retargeting and a lot of manual labor from animators. The data is often not usable in 3D applications, if the depth information is only roughly approximated or non-existent.

Scope of the Thesis This work focuses on 3D facial motion tracking based on a monocular RGB video and any facial character rig. The scope is limited to recruiting 10 experts of the 3D animation field, where each of the volunteers will participate in an interview and answer a questionnaire. Techniques using depth sensors, 3D scans, landmark tracking and highly labor-intensive approaches are not covered by this work.

Tasks

- Retrieve a deepfake algorithm that also works with humanoid, but non-human characters
- Create software that combines the preprocessing and the fitting process into one application
- Evaluate the outcome

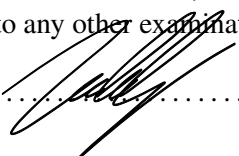
Requirements

- Robust to a wide variety of video inputs and 3D characters
- System is unaffected by different external characteristics of the actors
- The algorithm is rig-independent and can work with multiple kinds of bone setups

Keywords Facial Motion Capture; Rig Independence; Facial Animation; Retargeting; Image-based Techniques

I hereby declare that I have prepared this thesis entirely on my own and have not used any auxiliary materials other than those indicated. All passages of the thesis, which are taken in the wording or the sense of publications or lectures of other authors, I have marked as such. The work has not yet been submitted in whole or in part to any other examination authority, nor has it been published.

München, January 31, 2022



Acknowledgment

I would like to thank my supervisors, Dr. Sylvia Rothe and Changkun Ou, without whose guidance and constructive criticism I would not have been able to develop and formulate a thesis in such an enjoyable way. A special thanks goes to Mr. Ou, who supported me with his expertise during the design phase of the prototype.

I would also like to thank Karo Castello for the insightful discussion, which provided an application-oriented perspective and ultimately led to the initial rough concept.

Contents

1	Introduction	1
1.1	Thesis Structure	3
2	Related Work	5
2.1	Facial Rigging	5
2.2	Traditional Motion Capture	9
2.3	Learning-based Approaches	10
3	Methodology	15
3.1	Conceptual Approach	15
3.1.1	Initial Concepts	15
3.1.2	Discarded Concept	16
3.1.3	Final Concept	16
3.2	Image Processing	17
3.3	Blender Setup	17
3.4	Classical Algorithm	18
3.5	Neural Network Design	18
3.5.1	Generative Adversarial Network	19
3.5.2	Convolutional Neural Network	19
3.5.3	Autoencoder	20
3.5.4	Loss Functions	22
3.5.5	Hyperparameter Tuning	23
4	User Study and Evaluation	25
4.1	Study Design	25
4.2	Results	26
4.3	Evaluation	26
5	Discussion	29
6	Conclusion	31
7	Appendix	33
	Bibliography	41

1 Introduction

Realistically animated faces in media have posed multiple challenges since their introduction in the commercial world of video games and movies. When dealing with 3D facial animation, there is no getting around the anatomical structure of a face when trying to digitally recreate a realistic face. Depending on the way they are counted, a human face has more than 30 muscles and a highly deforming layer of skin on top. For the area in between the nose and the upper lip alone, upwards of six muscles are moving the surface of the face [Westbrook et al., 2019]. This complex system of nerves, muscles and skin is responsible for important functions we rely on in our daily life, such as eating, speaking and the expression of emotion. Humans are in general well versed with their ability to convey emotion through their own face and the recognition of facial movement and expressions, as psychological research has found that the accurate processing of facial expression is a social necessity and a prerequisite for successful social living [Niedenthal and Brauer, 2012], implying a need for self-improvement among participants of social interactions. The geometry of the face is consequently a psychologically sensitive and mechanically complex structure.

A field in which the previously named significance of accurately deformed faces plays a large role, is Human-Computer Interaction (HCI). Every time a person interacts with some form of computer-generated or -driven humanoid face representation (e.g. a robot), the person's response can change based on the type of depiction presented in front of them. This phenomenon refers to the so-called "Uncanny Valley" as first described in Masahiro Mori's article "Bukimi No Tani" (eng.: The Uncanny Valley) in 1970. In the article, Mori presented a correlation between the human likeness of robots and the emotional response of humans reacting to them and emphasized the difference between dynamic and static objects, the moving ones having a larger effect on the response from the observer [Mori, 1970] than their motionless equivalents (as shown in Figure 1.1). While the Uncanny Valley is a widely used term, later research has shown that there is no such thing as a "valley", rather that people become progressively sensitive to human-like robots and media with increasing realism [Hanson et al., 2005]. Furthermore, the study found that if the visual representation of a human is designed well, any level of realism can be perceived positively. While the requirements of "well designed" were not specified, they revised the idea of a "valley" into a theory they named "Path of Engagement" (POE). This POE theory states that realistic faces possess a denser information flow and are therefore prone to causing unfavorable emotions, such as disturbance, oddity or "surreal" feelings, more easily than their less realistic counterparts.

Although the authors express the need for further exploration around the proposition, both the Uncanny Valley theory and the POE theory agree that the level of authenticity of artificial faces correlates with the chance of triggering negative reactions. Moreover, both theories coincide that moving faces are introducing additional challenges, thus complicating the task of creating positively perceived faces across the entire spectrum of human likeness.

Creating dynamic faces for robotics is not the only application for realistic human representations, though. Movies, video games and other types of dynamic media have been increasingly incorporating realistic computer-generated faces, undertaking the previously named challenges to create likable representations of human or human-like characters. To tackle the problem of com-

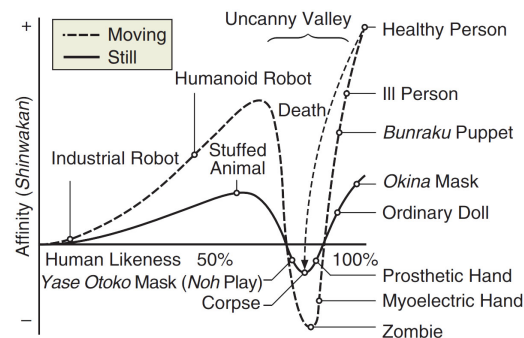


Figure 1.1: The uncanny valley as depicted by Mori [Mori, 1970], translation by MacDorman and Kageki [Mori et al., 2012].

plex facial layouts and high requirements for simultaneously realistic and appealing faces, 3D animators and software engineers have developed multiple approaches, such as using a number of controllers to create a network structure capable of moving a 3D character's face. The controllers, part of such a virtual skeleton (also known as "rig"), are referred to as "bones" and their importance for deforming 3D faces can be quantified by measuring the difference in concentration of bones in the face and the rest of the body. "A Facial Rigging Survey" from 2012 [Orvalho et al., 2012] listed a number of significant facial rig approaches ranging from 1996 up to 2011 and by looking at one of the most recent rigs, it is noticeable that even the simplistic body of the cartoon-like 3D character "Otto" [Vasconcelos, 2011], is mapped to 33 bones for the face and 32 for the rest of the body, while more realistic rigs usually contain several times of that amount of bones for the face. This statement reinforces the previously formulated observation, in that the face is a mechanically complex structure, from more than just a medical perspective. Often needing more controllers to accurately deform a 3D representation of itself, than any other part of the human body.

Multiple research institutions and corporations are determined to find viable solutions concerning the difficulties described earlier, including The Walt Disney Company, one of the top 5 international media corporations in the world (as of 2020) [Wäscher and Hachmeister, 2021]. A state-of-the-art report published in 2018 by a network of research laboratories of The Walt Disney Company, called "DisneyResearch", summarizes the current approaches, applications and proposals regarding the reconstruction and tracking of 3D faces from a single viewpoint [Zollhöfer et al., 2018]. The authors state that monocular facial tracking is not able to supersede the well integrated controlled multi-view setups, that are part of today's content creation pipelines, yet, due to their lower tracking quality. Although they emphasize that further research is needed to narrow the gap between lower performing single-view and high-quality multi-view solutions, in order to pave the way for more accessible facial tracking technology and new types of applications. The report also concludes, that the existing pipeline for creating photo-realistic virtual humans still requires large amounts of tedious and time-consuming manual labor and is far from being a fully automated workflow. Accelerated production times and a large impact on the 3D animation pipeline would be the consequence, if such a process could be automated, according to the authors.

Following up to this call to action, the objective of this thesis is to automate the facial motion tracking pipeline in its entirety. To a point where a 3D animator needs to provide an algorithm with just an RGB video of an actor performing the desired facial expressions and any preexisting facial rig, while maintaining a certain degree of animation quality. While it would make an interesting endeavor, the goal of this research is not to meet the standard in quality of current state-of-the-art facial motion capture technology. Rather automating the pipeline as much as possible, to provide accessible software without the usual learning curve. Therefore, independent game developing studios, 3D animation freelancer or even educational institutions are able to create basic facial animation in a feasible and cost-effective manner. To ensure that 3D animators of all skill levels are able to use such an algorithm without having to learn any prior concepts, a robust system must be developed that can work with a wide variety of video inputs and 3D characters. This demands a pipeline capable of working with multiple lighting setups, different backgrounds, that is unaffected by the actor's external characteristics and is rig-independent. The latter is particularly interesting, because our study showed that setting up a character rig with motion capture software is on average a tedious, unpleasant and manual labor intensive process. Consequently, being able to use virtually any facial rig with a motion capture software would be a powerful tool and elevate the baseline of what is possible with limited skills and knowledge.

We present a new technique for registering and analyzing facial rigs for an image-based motion capture algorithm. Additionally, we propose an end-to-end pipeline incorporating a deep learning approach for unsupervised animation generation, solely based on a single RGB video.

1.1 Thesis Structure

This work is structured into multiple chapters, starting with the current and [first chapter](#), which is aimed to give a brief introduction into the field and cover the goal of this work. Continuing with [chapter 2](#), a related work section is provided, where multiple facial deformation methods are described as well as relevant prior knowledge. Furthermore, our approach and methodology are described in [chapter 3](#), with details about failed and successful implementations in regard to the prototype. Subsequently, [chapter 4](#) describes the conducted user study and its results as well as objective results, such as measurements and outputs taken from the prototype. [Chapter 5](#) discusses the key findings and limitations, interprets and relates them to a recent publication of the same research area. In the [penultimate chapter](#), the main arguments are summarized and formed to a conclusion, next to the listing of future research opportunities. The appendix forms the [last chapter](#), providing additional information about the literature study.

2 Related Work

To gather more information about current state-of-the-art approaches and pipelines, a literature study was conducted and will be covered in the following. Based on the literature, a theoretical background can be established to create a catalog for the techniques currently used in the facial motion capture pipeline, as well as ongoing challenges and potential opportunities in the future. For further details about the literature study, see Section 7.

2.1 Facial Rigging

To realize dynamic 3D characters, animators create a rigging system to control the movement of the character using different approaches. Such as blendshapes and free-form deformations, as more direct ways to animate a 3D model, compared to bone- and physically-based approaches. In the following, all four approaches will be illustrated briefly and compared by relevancy, to give insight to their respective complexity and put the comparison on a more quantifiable basis.

Blendshape-based rigging for instance, is the primary method used in the reviewed literature with 6 of the 20 publications using blendshapes and 10 additional papers using systems that fit into the blendshape category or a hybrid system. The idea behind such approaches is based on the assumption that the topology of the vertices of a 3D character model (also: mesh) does not change during their animations. Therefore, a relation between the position of each vertex in the animated position and the neutral pose can be formulated as delta positions of the respective vertex. Using a factor to scale the delta positions, the animated pose can be blended linearly with the neutral pose. This is also referred to as linear blendshape weights [Chandran et al., 2020, Zollhöfer et al., 2018, Gibet et al., 2011, Ribera et al., 2017]. The delta blendshape system is used in software, such as Blender and Maya, which are the currently most used software for animation by a group of experts (6 and 5, out of 14).

In order to have a better insight into the fundamentals of blendshapes, an algebraic notation is provided, based on [Lewis et al., 2014].

A 3D mesh \mathbf{m} comprises, but is not limited to, vertex positions \mathbf{v} containing triplets of coordinates (x, y, z) respectively. Since blendshapes affect the vertex positions only, we define a 3D mesh as a vector of vertex positions for illustration purposes:

$$\mathbf{m} \equiv (\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{k-1}, \mathbf{v}_k), \quad (1)$$

where k is the amount of vertex position vectors in the mesh \mathbf{m} . Using the delta blendshape principle, a blendshape \mathbf{b} of the pose p (notated as \mathbf{b}_p), can be described as the difference of the neutral mesh \mathbf{m}_0 and a posed mesh \mathbf{m}_p or simply as $\Delta\mathbf{m}_p$, scaled by the respective weight w_p :

$$\mathbf{b}_p = w_p(\mathbf{m}_0 - \mathbf{m}_p) = w_p\Delta\mathbf{m}_p. \quad (2)$$

Consequently, by using n blendshapes to pose a neutral mesh \mathbf{m}_0 , the final mesh \mathbf{m}_{final} can be expressed as

$$\mathbf{m}_{final} = \mathbf{m}_0 + \sum_{i=1}^n w_i\Delta\mathbf{m}_i = \mathbf{m}_0 + \sum_{i=1}^n \mathbf{b}_i. \quad (3)$$

Another publication [Savoye, 2018] proposed a similar approach using free-form deformations (FFD) as an additional way to directly alter the position of vertices in a mesh. Using so-called "cages" surrounding a 3D mesh, one can move numerous vertices of a mesh by changing the position of just a few or a single vertex.

For ease of discussion, the algebraic representation of such a system is broken down into a two-dimensional setting. An example of one of these 2D cages is provided in Figure 2.1.

The aim is to alter the position of the vertex \mathbf{v}_0 using the surrounding cage vertices \mathbf{v}_i in a similar way to the previously discussed blendshapes using a weight w , so that

$$\mathbf{v}_0 = \sum_{i=1}^k w_i \mathbf{v}_i. \quad (4)$$

The weight w_i , according to [Floater, 2003], can be described as

$$w_i = \frac{\omega_i}{\sum_{j=1}^k \omega_j}, \quad (5)$$

$$\omega_i = \frac{\tan(\alpha_{i-1}/2) + \tan(\alpha_i/2)}{\|\mathbf{v}_i - \mathbf{v}_0\|}, \quad (6)$$

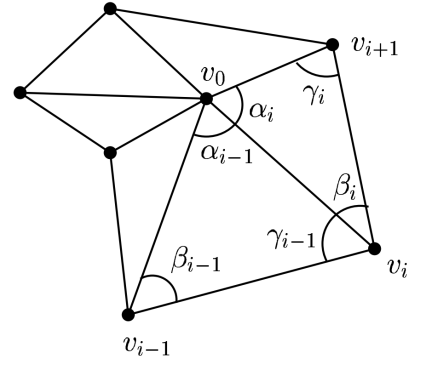


Figure 2.1: A 2D cage of vertices, affecting a center point [Floater, 2003].

where the weights w_i and ω_i are barycentric coordinates for \mathbf{v}_0 with respect to $\mathbf{v}_1, \dots, \mathbf{v}_k$. The angle α_i , as illustrated in Figure 2.1, describes the angle $\angle \mathbf{v}_i \mathbf{v}_0 \mathbf{v}_{i+1}$, for all k vertices in \mathbf{v}_i . Only a single publication of the reviewed papers used a form of FFD system though, implying that it is not a widely used method for facial animation.

Moving away from proposals where an animator would influence the vertices' position directly, a physically-based approach is described in [Kähler et al., 2003]. The authors use a variant of a deformable, anatomy-based head model initially described by [Kähler et al., 2001], where the skin and muscle movements are simulated using a mass-spring system.

Using spring-connected nodes to connect the skin to either facial muscles or the skull, they mimic the non-linear elastic properties of human skin, as shown in Figure 2.2. To prevent volume-loss at the point of contraction, additional springs and nodes are introduced to counter the force of the springs from the contracted muscle. These mirrored nodes also move in the same way as the original nodes, to ensure that the nodes of the skin mesh moves in a direction tangential to the skull and muscle surface. This setup provides a more realistic contraction across the skin mesh of the head model. A single node of the skin mesh can influence the position of one or more vertices, resembling the cage-based deformations discussed previously.

To give a brief insight of how a muscle contraction of such a system can look like, a possible algebraic notation of a sphincter muscle is provided.

According to [Kähler et al., 2001], a sphincter muscle's contraction can be described through the absolute positions of the nodes (also: control points) of the muscle:

$$\mathbf{q}_i = \mathbf{p}^* + (1 - c)(\mathbf{p}_i - \mathbf{p}^*), \quad (7)$$

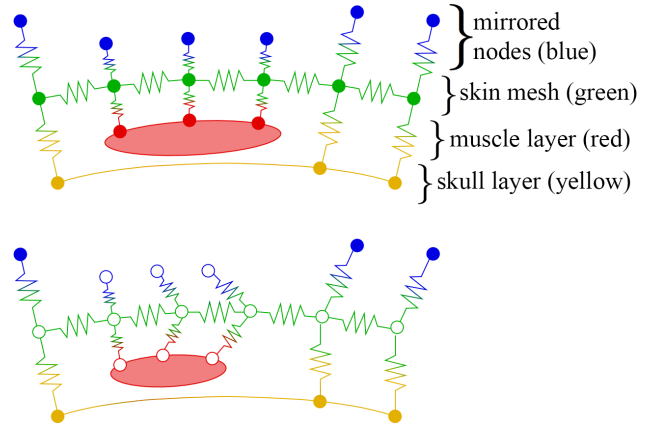


Figure 2.2: The mass-spring system of a physically-based head model by [Kähler et al., 2003]. Top: Relaxed muscle. Bottom: Contracted muscle with the affected nodes marked with \circ . Adapted graphic according to [Kähler et al., 2001].

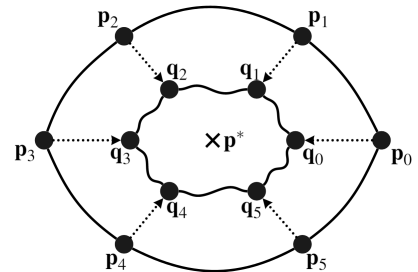


Figure 2.3: The linear contraction ($c = 0.5$) of a sphincter muscle fiber to its center. Adapted graphic according to [Kähler et al., 2001].

where $\mathbf{p}^* \in \mathbb{R}^3$ is the center of the muscle, \mathbf{p}_i are the relaxed control point positions, c is the weight of the contraction and \mathbf{q}_i are the control points in a contracted position. The contracting motion is illustrated in Figure 2.3. Similarly to the FFD above, only a single physics-based approach was found in the reviewed literature. The rarity of a method like this in consumer media, can be explained through the increased computing power, necessary to simulate such a spring-based system. Therefore, the application for physics-based methods lies in areas that need a different kind of accuracy, such as the medical field.

As the last of the four mentioned approaches to facial rigging by [Orvalho et al., 2012], bone-based rigging is the second most used practice from the acquired literature. We refer to [Mukundan, 2012], because the literature that mentioned bones for facial animation did not cover the topic of bone animation theoretically.

Bone-based rigging mimics some of the mechanical characteristics of a human skeleton, but can also be used to move non-rigid areas of the body. Bones are controllers, placed in 3D space to alter the position of vertices in relation to the bone. Even though bones are often viewed as the connection between two joints of a skeleton, in the animation field, bones and joints are more closely related; to a point where they can refer to the same entity. This is due to the fact that a head bone, for instance, is translated, rotated and scaled around the same origin as a head joint. Since there is no other use case for these controllers in this paper, we will refer to them as "bones" from now on.

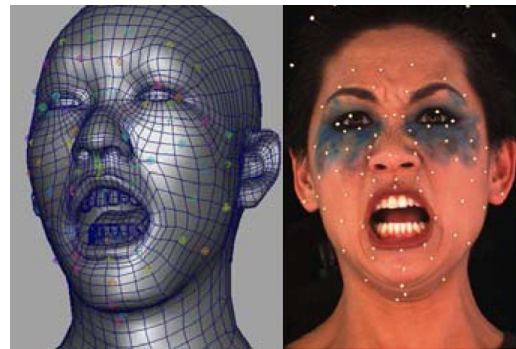


Figure 2.4: A facial bone rig created for motion capture with landmarks [Borshukov et al., 2006].

In the first stage of rigging, the animator (or: rigger) places the bones in a pattern that covers the desired points of interest, also referred to as "facial landmarks". In the case of facial motion capture with landmarks, a rigger will place bones on all of the landmarks of the actor's face, like in Figure 2.4. A skeleton is usually created in a hierarchical tree-like structure, with the root bone often associated with the position of the character in 3D space. The first bone in the hierarchy, that belongs to the character, is usually the pelvis bone [Mukundan, 2012], because of its proximity to the human body's center of mass. To ensure the coherence of the hierarchy, the pelvis bone is part of a parent-child relationship with the root bone, where the pelvis is the child of the root bone. This causes the pelvis bone to translate, rotate and scale with the root bone, while maintaining its predefined offset. The coordinate space associated with the root bone is called "skeleton space", which, in the case of the root bone being in the same spot as the character mesh, is the same as the coordinate space of the mesh. The pelvis bone's local coordinate space, along with all other bone's local coordinate spaces, is referred to as their respective "bone space" [Mukundan, 2012]. Continuing down the hierarchical bone chain, via the spine and the neck, the head bone is the first relevant bone for this paper. To facilitate the subject matter, we will first focus only on the head and the jawbone.

In Figure 2.5, the relationship between the jaw and head bone are illustrated. The upper image shows the default pose of the skeleton (also: rest pose). The lower one shows the same 3D character with the head tilted downwards and the jaw opened. This is achieved through rotating the head and jaw bone, by the angles α_1 and α_2 respectively. The dotted lines in the lower picture show the resting transformation of the two bones in bone space. The dotted line of the jaw bone is tilted, because the head bone was moved and in order to maintain its offset as a child of the head, the jaw bone moved simultaneously.

Linear Blend Skinning

To bind vertices to bones, all bones are mapped upon creation to indexes and the bone index influencing a specific vertex is saved as a new data field in the vertex object. The assignment of vertices to bones is also referred to as "skinning" [Mukundan, 2012]. To illustrate how this mechanism works, we assign a matrix \mathbf{B}_i to all of the bones, where i is the index of the respective bone, that defines the transformation from the bone space to the skeleton space [Mukundan, 2012].

In order to compute the new position, rotation and scale of a vertex \mathbf{v} after the jaw has moved (see Figure 2.5), the resting pose of \mathbf{v} , given as the transformation matrix \mathbf{V}_0 in the coordinate space of the mesh (\approx skeleton space), needs to be transformed by the inverse matrix of the assigned bone. Since the coordinates of the vertex are now in the local bone space of the jaw, a joint angle transformation matrix \mathbf{B}'_i , based on the bone with index i and the angle α_2 , can be applied. This also returns the coordinates to the skeleton space, which can be notated as:

$$\mathbf{V}_1 = (\mathbf{B}'_i \mathbf{B}_i^{-1}) \mathbf{V}_0, \quad (8)$$

where \mathbf{V}_1 is the transformation matrix of vertex \mathbf{v} in a posed position [Mukundan, 2012]. However, this applies only if \mathbf{v} is assigned to a single bone only. In the case of multiple bones affecting the vertex \mathbf{v} , a weight value w_i is assigned to each bone with index i , which scales the influence the respective bone has on \mathbf{v} . The final transformed point \mathbf{V}_1 is obtained as

$$\mathbf{V}_1 = \left[\sum_{i=1}^n w_i (\mathbf{B}'_i \mathbf{B}_i^{-1}) \right] \mathbf{V}_0, \quad (9)$$

where n is the number of bones affecting the vertex \mathbf{v} and

$$\sum_{i=1}^n w_i = 1 \quad (10)$$

must apply, according to [Mukundan, 2012]. The previously covered practice of binding vertices to one or more bones, that scale their respective influence linearly, is referred to as "Linear Blend Skinning" and is prevalent enough, that all of the animation programs used by 10 experts either do not offer skinning capabilities or use linear blend skinning. Current animation software, such as Blender and Maya, possess four bone index entries per vertex, therefore up to four bones can influence a single vertex at a time. To accurately assign a weight value for each bone to every vertex, an artist digitally paints the vertex weights, often directly onto the 3D mesh. The resulting painted "weight maps" are bone-specific and an example of a weight map of a jaw bone can be observed in Figure 2.6. These maps are sometimes displayed in different color spaces, Blender uses a color

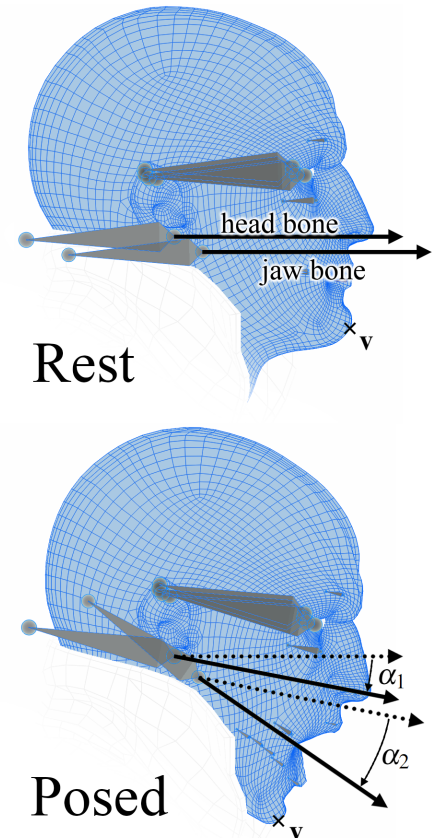


Figure 2.5: A 3D character with a head and a jaw bone. Top: The rest pose of the skeleton. Bottom: The posed head and jaw bones.

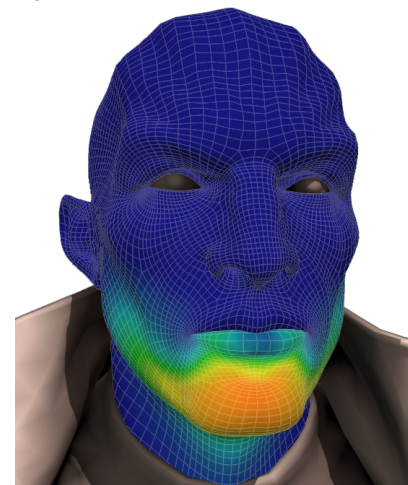


Figure 2.6: The weight map of the jaw bone applied to a 3D character.

space similar to the jet color map, where red represents the maximum weight value of 1 and blue indicates a factor of 0, for instance. Maya on the other hand, uses a linear color map, that maps a value of 0 to black and a 1 to white. One could also observe, that even though the skinning is saved in the respective vertex, a weight map also represents the area of influence from the bone on the 3D mesh.

Although only 3 out of 20 publications used bone-based approaches for facial animation, "A Facial Rigging Survey" from 2012 [Orvalho et al., 2012] mentions that blendshape rigs are performing worse regarding smooth interpolations between poses, than bone-based rigs. This can be explained through the usage of landmarks in facial animation, since blendshapes often contain poses of either the whole face or large portions of it, while skeletal animation relies on point transforms. Thus an animator can ensure that the corner of a 3D character's mouth, if bound to a bone controller, stays in the same spot during expression transitions, or interpolates smoothly from point **a** to **b**. A blendshape rig does not offer this kind of fine control. Additionally, blendshapes can still be incorporated into a bone rig, creating a hybrid rig where the main motion is driven by a bone rig and skin deformations, such as wrinkles, can be animated through blendshapes.

2.2 Traditional Motion Capture

From the 20 reviewed papers, only 16 contained specifications on the facial motion capture techniques used. Two papers, of the mentioned 16, still used some form of the more traditional landmark-based facial tracking technique, without learning-based algorithms. The authors of the first paper [Borshukov et al., 2006] used a synchronized multi-camera setup, incorporating 3 high-definition RGB cameras and 8 infrared cameras. Through the tracking of the approximately 70 markers placed on the actor's face, a one-to-one 3D reconstruction of the marker's positions was accomplished (as pictured in Figure 2.4). The second publication uses 2 or 4 head mounted cameras (HMC), depending upon the project [Cantwell et al., 2016]. For the reconstruction technique, they refer to [Bhat et al., 2013], which shows a similar approach to [Borshukov et al., 2006], but using only a HMC system and incorporating spline-based solving algorithms for the lips and eyes.

Four publications used scan interpolation in a way to realistically animate faces. The first paper covering the "Digital Emily Project" [Alexander et al., 2009], captured the actor's face using a setup of 156 lights and multiple cameras called the "Light Stage 5" (see Figure 2.7). After multiple expression scans were captured, the authors constructed blendshapes using landmarks, drawn initially on the actor's face. To track the facial motion, a proprietary video analysis and animation system from the company Image Metrics was utilized. Even though, the authors do not cover the specifications of the software used within the project and just call it "Image Metric's proprietary video analysis and animation system" [Alexander et al., 2009]. After some investigation, we found out that the company Image Metrics sold their facial animation branch to a company called "Faceware Technologies, Inc.". The earliest version of the facial analyzer software (to the best of our knowledge) with the version number 1.0.0.13307, was found in a promotional video from 2012. Based on the video and the information gathered from a version we got to try (version number 3.2.0.420), we make multiple assumptions, one of them being that a similar system was used on the "Digital Emily Project". The facial analyzing software tracks the facial motion without markers, but using a face's nat-

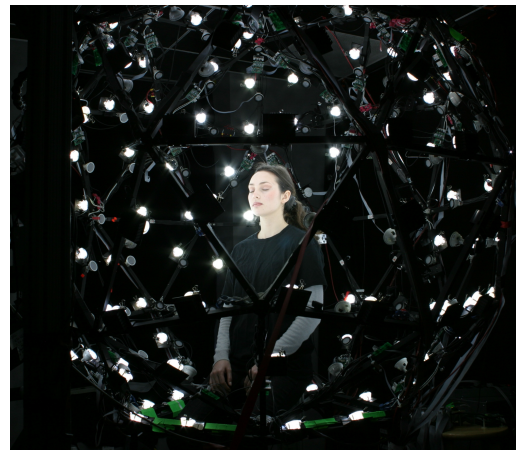


Figure 2.7: An actor being scanned in the Light Stage 5 [Alexander et al., 2009].

ural features to create curves, that can later on be brought into proportion with facial poses. Therefore, an animator needs to create only a small number of key poses, that are interpolated based on the curves of the tracked points. The markerless tracking of the facial landmarks is illustrated in Figure 2.8, which is a screenshot of the Faceware Analyzer 3 software’s output.

The rest of the non-learning-based approaches using scans, not mentioned yet, all used a system based on some form of optical flow. Optical flow is a 2D representation of motion information, where a single or a block of pixels of an image are tracked and saved as a 2D vector carrying velocity information (shown in green in Figure 2.9). All of the three remaining papers used a similar approach [von der Pahlen et al., 2014, Fyffe et al., 2015, Andrus et al., 2017], in a sense that all systems used optical flow to deform the face of a 3D character, using projection algorithms to transfer the 2D data onto the 3D model. Although not specified in [Andrus et al., 2017], the two other publications relied on a multi-camera setup, similar to the Light Stage 5 in Figure 2.7, to support the optical flow algorithm and achieve more accurate results. Though the authors of [Fyffe et al., 2015] state that motion transfer from one character to another, using their system, remains subject of future research.



Figure 2.8: A screenshot of the Faceware Analyzer 3 software tracking an actor’s face.



Figure 2.9: The optical flow of a steady camera with moving people [Patait, 2019].

2.3 Learning-based Approaches

Approximately two-thirds (10 out of 16) of the reviewed papers incorporated some form of learning- or regression-based algorithm. For ease of discussion, we use regression-based algorithms interchangeably with learning-based algorithms and refer to both as the same entity. To better distinguish between the propositions, we categorize them in three groups: landmark mapping, geometry regression and image-based mapping. Beginning with the first and biggest category, two papers [Moser et al., 2018, Hendler et al., 2018] point to a third paper [Moser et al., 2017] of the selection, using the same system, but respectively improving it in their proposal. The base system, described in [Moser et al., 2017], is based off of the works from [Bermano et al., 2014] and [Bickel et al., 2008]. We assume from the information of the last three mentioned references, that the system driving the faces works off of a regression-based mapping between a low-resolution input (landmarks or rig) and a high-resolution geometry output. The mapping is learned based on a dataset of facial scans, allowing the model to generate realistic faces, based on the input from head mounted cameras, that were only possible in seated capture techniques before [Moser et al., 2017]. Another regression-based algorithm is covered in [Gibet et al., 2011], where the authors map the positions of facial landmarks on an actor’s face to blendshape weights, using Gaussian Process Regression. Another proposal is an algorithm that maps 3D tracked facial landmarks between either an actor and a rig or between different rigs, with special attention to the range of motion of the respective face [Ribera et al., 2017]. The authors achieve the mentioned goal, by learning the actor’s expression range from an initially captured training sequence, for a blendshape rig. The last of the learning-based landmark mapping approaches only focuses on the jaw position, but does so through learning the position of multiple points on an anatomical jaw bone, using facial landmarks [Zoss et al., 2019]. The authors trained a non-linear mapping using a dataset containing images of varying jaw poses and jaw bone

positions.

Another category of learning-based algorithms is the geometry regression. In the first instance, researchers used a regression framework, initially described in [Cao et al., 2012], that regresses a contour onto the eyelids for improved tracking [Bermano et al., 2015]. The authors also use optical flow, as described earlier, among other techniques. One other geometry-based learning algorithm is presented in [Chandran et al., 2020]. The authors use deep neural networks to generate expressive 3D faces, with the identity and the expression of the face decoupled from one another. This distinction allows facial motion tracking capabilities, through the learning of a mapping between the 2D landmarks and the pre-trained network, generating believable human faces [Chandran et al., 2020].

Finally, image-based regression is used in two other works. In the first one, the authors trained a regressor using synthesized poses of a blendshape character rig from the performing actor, from the perspective of a head mounted camera. Since the rig will be driven by a video of the actor performing the desired expressions, the regressor is able to map them to the learned poses and outputs weights for the blendshapes, without the usage of landmarks. This purely image-driven approach also comes with the advantage of the whole face being used as a high-resolution input, compared to low-resolution inputs like trackers or landmarks. The last publication discussed in this section deserves special attention, compared to the rest of the discussed literature in this section, because of its recency and the fact that it is closely related to the work in this paper. This is due to the fact that it was published in the same month as the processing period of this thesis ended. "Semi-supervised video-driven facial animation transfer for production" proposes an algorithm for automatic facial motion capture, using monocular RGB video data and a learning-based approach [Moser et al., 2021]. Their method already meets most of the requirements and tasks of this thesis, through incorporating an algorithm that uses an actor's expressions from an input video to generate a video of the soon-to-be-animated 3D character, performing the same facial movements. But it fails in the case of lighting, background and actor changes, as well as field of view, clothing and occlusion, as seen in Figure 2.10 [Moser et al., 2021]. The publication itself illustrates however, the relevancy and activity of this research field and that there is still plenty to improve.



Figure 2.10: Failure cases of the algorithm proposed in [Moser et al., 2021].

Deep Neural Networks

For further elaboration, a brief overview of deep neural networks (including deepfake algorithms) is provided in the following, based on an up-to-date textbook released in October 2021 [Mehlig, 2021]. The general goal of neural network architectures is to provide a computational model, inspired by the basic unit of the nervous system in a human brain, the neuron. The first iteration of a simulated neuron was the McCulloch-Pitts neuron, according to the book. The concept is to make the neuron switch between two states (active and inactive), depending on some arbitrary input in the form of one or multiple weighted scalars and a threshold. Even

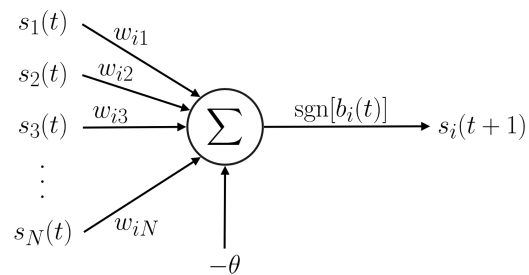


Figure 2.11: Schematic diagram of a neuron, adapted from [Mehlig, 2021].

though this configuration is often referred to as a "perceptron", the authors also refer to it as the McCulloch-Pitts neuron [Mehlig, 2021]. Since the model performs repeated computations in discrete time steps t , the resulting state s (or: output) of the neuron with index i after one time step is notated as $s_i(t+1)$. As pictured in Figure 2.11, the output of the neuron can be formally expressed as:

$$s_i(t+1) = \text{sgn} \left[\left(\sum_{j=1}^N w_{ij} s_j(t) \right) - \theta_i \right], \quad (11)$$

where θ_i is the threshold respective to neuron i and N is the number of input neurons [Mehlig, 2021]. The function name $\text{sgn}(b)$ refers to the signum function:

$$\text{sgn}(b) = \begin{cases} -1, & b < 0, \\ +1, & b \geq 0. \end{cases} \quad (12)$$

The argument of the signum function, shown in Equation 11, later labeled as b , is also referred to as the "local field" of the neuron. It consists of the sum of the states of the input neurons, scaled by their respective weights and adding the threshold to the result [Mehlig, 2021]. To combat fluctuations or other undesired effects in a network of neurons, other so-called "activation functions" are used, such as $\text{sigmoid}(b)$ or $\text{tanh}(b)$, replacing the signum function in Equation 11 [Mehlig, 2021]. The negative threshold $-\theta$ is also referred to as the bias of the neuron, by machine learning libraries such as PyTorch. To make the network learn, an error of the current network needs to be computed. Using the example given in the book, a distance d_v , where v indicates an already known result of the network usually provided by a dataset, can be calculated. The predicted output x and the desired output $x^{(v)}$ are subtracted from each other and subsequently squared, resulting in a positive number, indicating the magnitude of difference between $x^{(v)}$ and x . This is also known as computing the error (or: loss) of a network, using the mean squared error loss function [Mehlig, 2021]:

$$d_v = (x - x^{(v)})^2. \quad (13)$$

Since x represents the last neuron's output, calculated through at least one iteration of Equation 11, the computed error d_v can be used to train the variable parameters of the neural network, consisting of the weights w_{ij} and the bias $-\theta_i$ of the respective neurons. This process is also called "backpropagation" and computes gradients for all parameters to minimize the loss d_v , making use of algorithms such as stochastic gradient descent [Mehlig, 2021].

The specification in the name "deep learning" and "deep neural network", compared to a generic neural network, derives from the higher amount of so-called "hidden layers", compared to artificial neural networks (ANN). A hidden layer is defined as a computational layer between the input and output layer of a neural network [Mehlig, 2021].

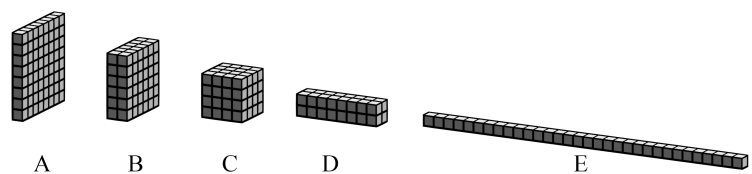


Figure 2.12: An example of a 2D convolution using a 3x3 kernel and a stride of 1. The initial input image (A) of size 8x8 pixels is convolved into two feature maps sized 6x6 (B). Followed by two more convolutions yielding four 4x4 (C) and eight 2x2 feature maps (D) respectively. The last operation flattens the feature maps into a one-dimensional vector of size 32 (E).

Deep learning based on higher dimensional data, such as grayscale or RGB images, introduced a set of new challenges. Networks in which the neurons of one layer are connected to all neurons of the following layer, known as "fully connected" neural networks, have higher computational and memory requirements compared to so-called "convolutional neural networks" (CNN) [Mehlig, 2021]. A CNN takes advantage of a principle observed in the human brain. The principle being, that neurons dedicated

for image recognition are designed to recognize local features such as edges or corners in images. These so-called "feature maps" are extracted using filters that are referred to as "kernels", in an operation named "convolution" (shown in Figure 2.12). Since an image often contains the same feature multiple times, the same kernel can be reused, reducing the total amount of neurons of the network. This results in another advantage; since there are fewer neurons in a CNN, also due to the usage of pooling layers reducing the size of feature maps, compared to a fully connected neural network, the network is less prone to overfitting [Mehlig, 2021] and requires fewer computations. Through the usage of convolutional layers (among others), that apply kernels to an input image, followed by one or more fully connected layers, certain predictions can be made about the input image by reducing the size of the input to a single output value.

Taking advantage of the mentioned and other concepts, efficient encodings of images can be learned through so-called "autoencoders". As shown in Figure 2.12, a CNN is capable of reducing the amount and dimension of input variables to achieve a specific output array of numbers, in the case of autoencoders this is referred to as the "encoding" of the image. Therefore the CNN is often referred to as the "encoder". The other part of an autoencoder consists of a "decoder", which maps the encoded image back to the original input [Mehlig, 2021]. By comparing the output of the decoder with the input of the encoder, a loss can be computed (see Figure 2.13).

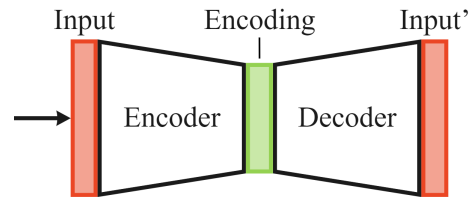


Figure 2.13: An illustration of an optimally trained autoencoder. The input and output (Input') are both marked in red, since they are equal.

Another neural network architecture relevant to this work is the "generative adversarial network" (GAN). Consisting of two multilayer perceptrons, the "generator" and the "discriminator". Using, besides others, a noise input, the generator generates samples of the desired data type. The discriminator receives either a real sample of the ground truth or a generated sample (fake sample) from the generator as an input and needs to differentiate whether it is a real or fake sample [Goodfellow et al., 2014]. This can also be expressed as a two-player minimax game, where the participants try to outdo each other.

Making use of advancements in the research field of deep learning, neural face swapping approaches, as mentioned in [Moser et al., 2021], have been developed. The so-called "deepfake" algorithms, a mixture of the words "deep" from deep learning and "fake", are able to generate synthesized image or video data, where the likeness of a person is replaced with another using a deep neural network. To keep this section concise, we will not elaborate further on the inner workings of a deepfake algorithm, but show that they are capable of transferring facial motion from one identity to another, even with varying video input. As seen in Figure 2.14, the algorithm takes a neutral frame of the target face and a video of an actor, meant to be puppeteering the target face (puppeteering deepfake algorithm). The result is a synthesized video, where the facial movements of the actor are transferred correctly onto the target face, even with varying backgrounds, actors and cropping. This 2D mapping preserves a high amount of detail compared to landmarks and might be interesting to explore.

To summarize the potential for improvement, as described in the literature: [Fyffe et al., 2015] mentions that frame-to-frame motion analysis systems, such as optical flow algorithms, are subject to the accumulation of error over time, or what they refer to as "drift". Since an optical flow system is used in other works [Andrus et al., 2017, von der Pahlen et al., 2014, Alexander et al., 2009], we assume that they encountered the problem as well. Though the works of [Fyffe et al., 2015] aimed at minimizing the drift, which they achieved according to the paper. Even though the authors treat the problem as solved, the system still relies on facial scans and does not solve the issue of a high amount of required pre-generated data. Remaining on the topic at hand, [McDonagh et al., 2016] describes that machine learning algorithms that are trained on an

actor’s face, either are not able to extrapolate from capture data, acquired in different camera or lighting setups, or require an unbearable amount of labor to capture and label a sufficient dataset. A learning-based algorithm, not reliant on a dataset, might be an opportunity worth exploring. Similarly, [Moser et al., 2021] mention that their deep learning algorithm degenerates in unseen conditions and name similar challenges as [McDonagh et al., 2016], such as different clothing of the actor, background changes, changes in lighting and field of view, as well as partial occlusion, the latter also being mentioned in [Zollhöfer et al., 2018]. Therefore, either a sufficiently trained algorithm or a system, that works independently from the actor’s characteristics and is strictly expression-based, would be required. Three other publications mention, among others, problems such as an insufficient amount of algorithm training [Zoss et al., 2019] and noisy or even incomplete data [Gibet et al., 2011, Ribera et al., 2017]. Finally, facial motion capture data can lack expressiveness, compared to the actor’s performance [Gibet et al., 2011] and requires extensive involvement of the animator. This can make the entire process time consuming and expensive, and is therefore usually employed only for main characters [Moser et al., 2018]. A method reducing the amount of manual labor required from the animator, seems to be the next step. On the note of manual tweaking, [Gibet et al., 2011] describe the issue of the blendshape data being directly linked to the facial performance capture. Consequently, an approach that allows mapping to a bone-based rig might be desirable, allowing the animator to tweak specific bone positions after the capture.

The contributions of this thesis in respect to the above are:

- For the first time to the best of our knowledge, an unsupervised image-based learning algorithm for retargeting facial animations from a frame of a processed RGB video to a 3D character, that does not rely on an actor-specific initialization or training and only uses an automatically generated weight map data structure with no additional datasets required.
- A fully automated end-to-end pipeline for registering generic facial rigs for processing with a neural network architecture and producing animations in the form of bone transforms.
- A high-resolution method utilizing a generic render engine for comparing facial poses and deformations, independent from low-resolution facial representations, such as landmarks.

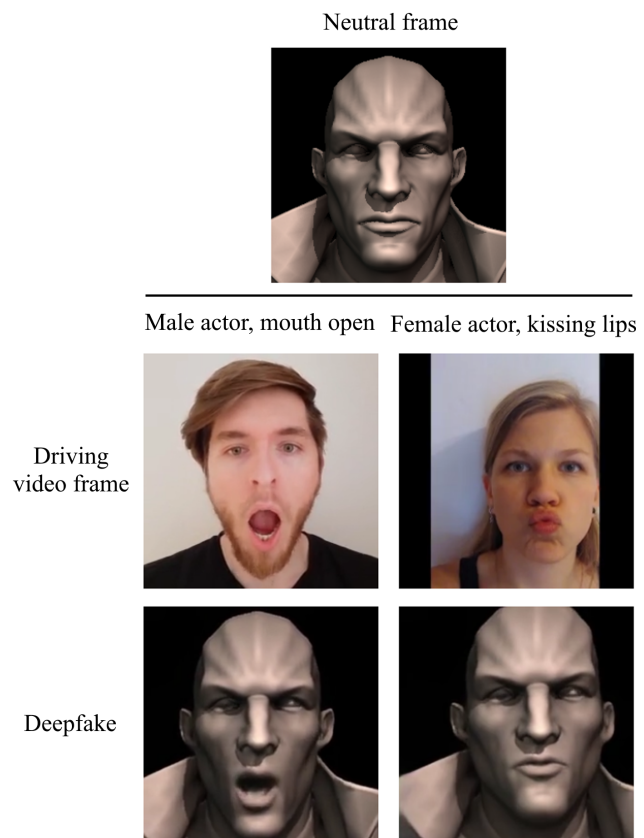


Figure 2.14: An example of a puppeteering deepfake algorithm, based on [Siarohin et al., 2019]. Neutral frame: The neutral face of the target character. Top row: Frames taken from a video of an actor’s performance. Bottom row: Synthesized video data, based on the neutral frame and the respective driving video.

3 METHODOLOGY

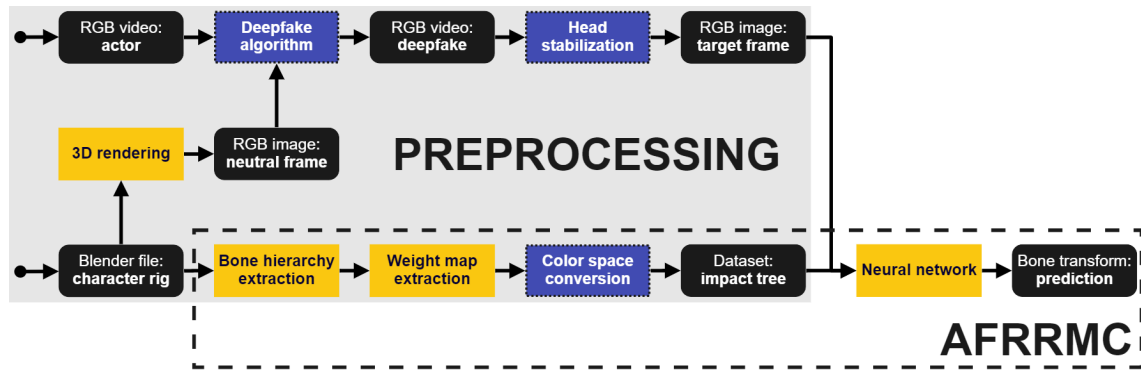


Figure 3.1: An overview of the proposed pipeline, the dashed box representing the proposed AFRRMC algorithm.

3 Methodology

We describe below our prototype (hereafter also referred to as the AFRRMC algorithm) and its development during the research process with reference to the theoretical background discussed earlier. The animation software used to accomplish the given tasks is Blender release version 2.93.7. The different stages of the conceptualization process of our approach, utilizing monocular RGB videos to drive facial animations independently from 3D character and actor, will be covered below. Subsequently, descriptions of the different stages of our program, divided into thematic categories, will be provided.

3.1 Conceptual Approach

The original concept emerged after a discussion with an expert in the animation field (4 years of professional experience) on the topic of workflow optimizations and differs from the final concept in a number of aspects, which are described in Section 3.1.3.

3.1.1 Initial Concepts

At first, the idea was to use a step-based algorithm that was able to move the bones of a facial rig gradually, based on an image comparison algorithm which computes an error map using two input images and a threshold. The output of the algorithm is the amount of differing pixels, based on the threshold, and an image, where the pixels outside of the range of the threshold are colored in red and the rest in white. The input images of such an algorithm would need to be one image representing the current state of the face in the animation software and another image showing the target expression on the same character. The latter was achieved through the usage of a deepfake algorithm, which successfully mapped the motion of an arbitrary actor's face onto a neutral face image, taken with a virtual camera inside the animation software (in our case: Blender). Using this "area of discrepancy" concept and having the mechanisms of bone weight maps in mind, a design where a bone is chosen based off of the overlap between the area of discrepancy and the area of influence of the bone, was worked out. The design converted the color space (jet) of the weight maps rendered from a front facing camera perspective, in a white (0) and red (1) color space. Thereupon the images, we refer to as "impact maps", were combined in a hierarchical way that matched the skeleton hierarchy. For instance, the impact map of the head bone, also contained all of the other red pixels of the facial bones, because they are the children of the head bone in our example rig. After that, the impact maps would be saved in a table, mapped to the respective bone name, for later access. The last step of the preprocessing stage of the concept was a face stabilization and background removal process. The background removal was initially

pictured as a face segmentation algorithm, resulting in a target image, containing only the face of the synthesized character expression.

Beginning the execution loop, the concept ought to run the image comparison algorithm, retrieving the amount of wrong pixels in between the image of the animation software and the target image. If the number would be within a certain threshold, the execution would be stopped immediately after. Otherwise, the program would continue by searching for the best matching impact map from the impact table and an algorithm would determine the next best bone transform using small steps to iteratively converge to the unknown target transform. During each step the algorithm would be guiding itself in the right direction using the image comparison algorithm's outputs.

3.1.2 Discarded Concept

While this work does not include approaches using depth sensors, we explored the possibility of training a neural network using a dataset of captured depth maps (as shown in Figure 3.2. Though, the algorithm was not able to extrapolate from real-world datasets to synthesized video footage of a 3D character, not being able to open the mouth of the synthesized face and other anomalies. This led to the approach being less appealing than the initial concept.



Figure 3.2: One sample pair of the captured depth dataset using a Microsoft Kinect v2 sensor.

3.1.3 Final Concept

Although the initial concept already contained a basic framework of the algorithm, many sections required revisions during the development process. For instance, the most substantial redesign was that the bone transform algorithm would use a learning-based algorithm and the concept of a stepping algorithm was scrapped and replaced by a neural network training itself gradually using its own predictions (justifications are presented in Section 3.4). This also changed the preprocess design, since the color space conversion would not need to match a color space anymore and was therefore revised into a single-channel conversion (shown in Figure 3.4). Furthermore, the summation of the impact maps proved to not affect the network significantly (as shown in Figure 3.3), hence the addition was removed. The background removal step was also discarded, because in the way that the algorithm would train itself, every iteration of the training would require a facial segmentation and background removal component, which would disproportionately reduce performance relative to the difference in quality. Finally, the exit of the execution loop, based on the error pixel count, was removed and replaced by a method that would allow the user to specify the amount of iterations and stop the algorithm at any point, while being able to review the current state of the training through data output, such as images and curves.

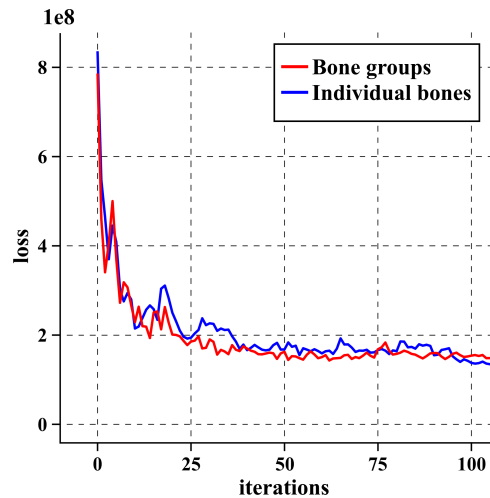


Figure 3.3: A diagram showing the similarity of the loss, in the case of an impact map containing all of its children (red) or not (blue).

3.2 Image Processing

While there are multiple types of image processing methods involved in the AFRRMC prototype, the deepfake algorithm takes up the biggest part of the computations. For the purpose of this work we assume that deepfake algorithms are readily available. The algorithm used in this project is based on the work from [Siarohin et al., 2019] and was chosen due to its availability at the time. It is not incorporated into the prototype and can therefore be exchanged without modifications to the AFRRMC algorithm. The AFRRMC algorithm's requirements of the input is merely a frame of a deepfake, provided by the user, opening up future opportunities.

For instance, an improved deepfake algorithm or potentially a different kind of synthesized image generators, such as body motion transfer algorithms, could be utilized in the same manner. However, we focus on facial animation and after providing the chosen deepfake algorithm with a rendered neutral image and a target video, we extract a frame from the resulting synthesized video (as seen in Figure 2.14) and feed it as an input to the AFRRMC prototype. We will refer to this input image as the "target image" from now on.

Our system begins with the rig registration process (as shown in Figure 3.1), described in more detail in the next section, and saves the extracted weight maps in a tree-like data structure, compared to the initial concept, aimed to use a table. This change enables us to treat the data in a hierarchical way, similar to the layout of the registered skeleton. Subsequently, the algorithm converts the jet color space of the rendered weight maps into a grayscale picture (seen in Figure 3.4) and aligns it to the neutral frame rendered in the beginning. This is achieved by finding the lowest overlapping error, computed by the sum of the absolute pixel difference of the two images, while iterating through the image coordinates. This could be greatly improved in terms of quality and speed, but since this stage is only executed once during the initialization, the speed was sufficient for our needs. Following down the execution chain, the next task requiring image processing is to blur both the target image and the current state of the character's face, to add a certain amount of robustness to the network. This is achieved through a Gaussian smoothing function with a kernel size of 7 and a sigma value of 5. Finally, the image comparison algorithm, originally intended to evaluate the quality of the algorithm's output, was removed, since our prototype uses a neural network to predict bone transforms, based on a loss function that incorporates an algebraic equivalent of an image comparison. This will be covered in Section 3.5.4.

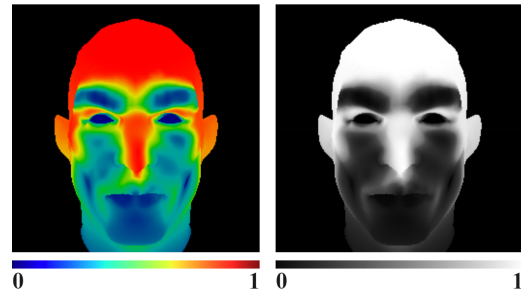


Figure 3.4: Left: The rendered weight map of the head bone, displayed in a jet colormap by Blender. Right: Impact map, converted from weight map.

3.3 Blender Setup

To be able to compare a deepfake of an actor and a rendered image of a 3D character, the rendering quality of our animation software needs to meet a certain standard, to match the deepfake algorithm's aesthetics. For example, we observed that the computation of shadows is necessary, because the deepfake generated the inside of our 3D character's mouth as a relatively dark color. To match the look of the synthesized video and enable a color-matching method, we used Blender's OpenGL renderer called "Workbench" to maximize performance, while maintaining some artist control over the character's external characteristics and the lighting of the 3D scene. The virtual camera used to perform the rendering is placed in front of the face and set to a resolution of 256x256 pixels, matching the output size of the deepfake algorithm.

As mentioned previously, the algorithm starts with the rig registration by automatically joining all 3D meshes loaded into the scene, that possess an armature modifier; indicating that they are part

of the character rig. Subsequently, the prototype switches to the "weight paint" mode and saves all bone names in a list using a user-provided input of the head bone name, which will later be used to build the previously mentioned tree-like data structure, we refer to as the "impact tree". Using the list of facial bones, the prototype iterates through all of them, rendering their respective weight map from the perspective of the camera and converting them into impact maps, as described above (and seen in Figure 3.4). The final steps of the Blender initialization includes merging the bone list and weight maps into the previously mentioned impact tree and switching the viewport back to "object mode" in which facial expressions can be rendered for the training.

3.4 Classical Algorithm

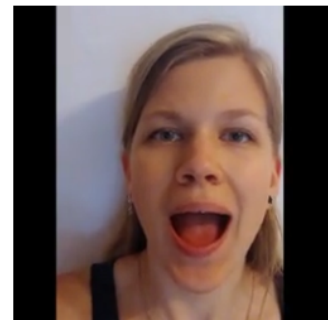
In contrast to the final approach, the initial concept incorporated a step-based algorithm that was aimed to converge to an optimal bone placement. We initially developed the later discarded algorithm, which we refer to as the "approximator" and cover it in the following.

Iterating through each of the six axes, three for location and rotation respectively, the algorithm divides a provided maximum search space into two intervals of the same size, yielding the minimum, maximum and center point of the respective axis. For each of the retrieved points, the algorithm renders an image of the character's face, posed with each of the data point as input. Then the algorithm compares the rendered image to the target image via a pixel difference and saves it as an error for each point. Once the lowest error of the three subtractions is computed, the search space is halved and the middle of the search space is set to the point with the lowest error. This way of computing an optimum delivered decent results sooner than the neural network training. In comparison, the AFRRMC prototype delivered a posed face, roughly matching the actor's image (as seen at the bottom of Figure 3.5) after 150 iterations, lasting 754 seconds on an NVIDIA GeForce GTX 1080 Ti GPU and an Intel® Core™ i7-6800K CPU in total. Using the same hardware and the approximator algorithm, the image in the middle was computed in 54 iterations, taking 83.6 seconds to compute. Although the neural network algorithm was not fully implemented in the design phase of the approximator, we expected better results from a neural network approach. One justification for this statement is the difference in computation and extrapolation. The approximator is unable to extrapolate from previously computed scenarios and also not capable of reaching specific poses that the neural network can, due to exploration limitations in the axes value computation. Additionally, a trained neural network's advantages are not limited to extrapolation based on unseen data, but can also execute faster than the approximator since no rendering is required to generate the desired bone transforms, decreasing the computation time (in our case to: 1.218 seconds). Consequently, the concept of a classical algorithm was discarded and a neural network approach was chosen.

3.5 Neural Network Design

The neural network design went through multiple design stages and concept changes, which was due to our relatively low experience

Driving video frame



Classical algorithm



Neural network



Figure 3.5: Top: The frame used to drive the deepfake and thus the rest of the algorithms. Middle: The classical algorithm after reaching a step size of $1.198 \cdot 10^{-5}$. Bottom: The neural algorithm after 150 iterations.

in the beginning of the development process. The largest challenge posed the dependency of the loss calculation on the Blender renderer. To the best of our knowledge, there are no differentiable renderers that allow the usage of bone controllers from generic 3D characters and therefore we needed to find a workaround solution. For the implementation of the neural networks, the open source machine learning library PyTorch [Paszke et al., 2019] was utilized. In the following, two failed attempts (Sections 3.5.1 and 3.5.2) and the final neural network design (Section 3.5.3) are described.

3.5.1 Generative Adversarial Network

The first failed approach regarding a neural network-based prototype, was to use a convolutional neural network (CNN) as the generator inside of a generative adversarial network (GAN). In our case, the CNN reduced four inputs, the target image, viewport image, impact image and a noise vector, to a transform vector of size 6, relating to the amount of axes. The added noise vector was intended to ensure the learning process of the GAN. The idea behind using a GAN was that we made the assumption that the generator and discriminator are independent from each other, therefore allowing learning directly based on the rendered viewport image. Though, a generator learns through chaining the computations of the generator to the discriminator [Goodfellow et al., 2014] and must therefore be differentiable at any point of the calculations, requiring a change in concept.

3.5.2 Convolutional Neural Network

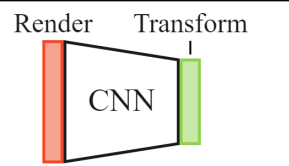
Moving away from an image-to-image learning-based algorithm, we designed a CNN (based on wrong assumptions) aimed to learn from its own predictions, inspired by the step-based approach described in Section 3.4. The algorithm (as shown in Figure 3.6) used a single CNN, which learned a provided transform vector of size 6 from an input image, being a rendering of the 3D character’s face. The image would be swapped out every x iterations by the target deepfake image, to make the network predict a new transform vector, which would then be used with the resulting rendered image to train the algorithm again. This algorithm was aimed to make the network gradually discover the most optimal face pose. The approach also intended to skip the need for the Blender renderer being incorporated into the algorithm’s backpropagation, through the direct learning of a vector from an image. To initialize the network, we assumed (at the time) it would be sufficient to let the algorithm render an image based on a transform vector filled with zeroes and train it for 100 iterations. The loss function for the loss L_{CNN} used in the training can be expressed as:

$$L_{CNN} = \sum_{i=1}^6 \mathbf{d}_i, \quad (14)$$

$$\mathbf{d} = |\mathbf{t}_{out} - \mathbf{t}_{target}|, \quad (15)$$

where \mathbf{t}_{target} is the provided target transform vector, \mathbf{t}_{out} is the predicted transform vector from the network and \mathbf{d} is the absolute difference between the vectors. The final loss L_{CNN} is calculated by

1. Training



2. Prediction

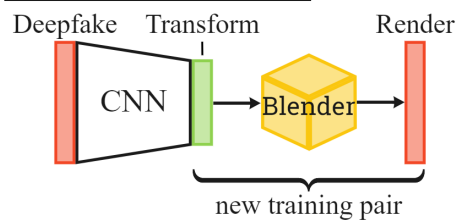


Figure 3.6: Top: The training algorithm, utilizing a pre-generated image and transform vector pair. Bottom: The prediction step, where a new training pair is generated using the target image.

the summation of the elements of \mathbf{d} , where \mathbf{d}_i is an element of \mathbf{d} with index i (specifically: the axis with number i).

Once the initialization stage was completed, the algorithm proceeded with the training loop. The concept of the training revolved around the idea of gradually training the network for a specified amount of times, using randomly generated target vectors. At the start of a single training iteration, the algorithm looped for 200 iterations through a "sub-training loop". First, the last prediction of the network and a random offset (in our case between -0.2 and 0.2) were added to each of the predicted vector's axes and an image, using the generated transform vector as the input for Blender, was rendered. The rendered image would now be treated as the target image and the network was trained by a single forward and backward pass, using the image as an input and the previously generated transform vector as a target, ending one sub-training loop. Once the inner loop finished, the actual target image (deepfake) was used to predict a new transform vector, by a forward pass through the CNN. Similar to the sub-training loop, a new image was rendered based on the transform vector and the network was subsequently trained for a single iteration using the latest computed image and vector. This step concluded the training loop of the CNN-based algorithm.

However, the network failed to deliver sufficient results, which we assume is due to a number of problems. One of them being that the network did not learn any meaningful information during the initialization stage, since it only learned to produce a zero vector based on a single image. Additionally, the training loop did not incorporate a batch learning implementation that combined multiple error calculations and instead learned from a single image and vector pair during the respective iteration. This was desirable to enable a one-shot algorithm, not requiring a dataset. The neural network design of a single CNN was scrapped during the concept development, since another network architecture (described in the following section) seemed to better suit the task.

3.5.3 Autoencoder

Having encountered multiple challenges with incorporating the Blender renderer into the neural network passes, the concept of using an autoencoder established itself during the prototyping process. If we formulate our task differently: we try to convert from image data (the target image) to another data type (the bone transform) and back into an image (by rendering an image using Blender), to then compare it to the target image. As previously described, an autoencoder learns a specific encoding of an input, then maps the encoding back to the input [Mehlig, 2021] and compares it using a loss function. This relation between our task and the architecture of an autoencoder was not entirely clear to us from the beginning of the implementation. We first assumed that it would need to be possible to backpropagate the Blender renderer step of the network, to make the forward pass differentiable. Though, we eventually moved away from using only the Blender renderer as a way to backpropagate the network and started prototyping with the idea of creating a second network, mimicking the computations of the rendering in Blender. As shown in Figure 3.7, the target image (Input) can be encoded using a CNN (Encoder) into a transform vector (Encoding) and then serve as an input for Blender to render an image (Render). The same encoding (marked in green) can be used as the input for a second network (Decoder), aimed to produce an image similar to the render from Blender (Render'). Due to their similarity (but not equality) be-

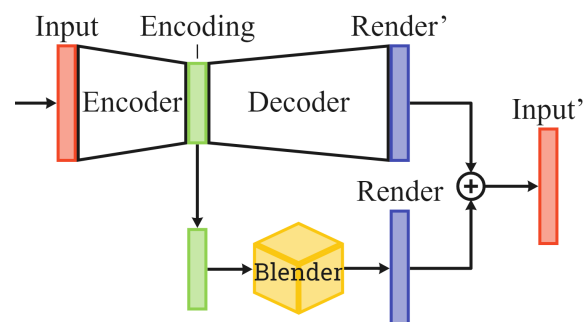


Figure 3.7: An example of how an autoencoder can be used to treat non-differentiable branches in a neural network.

tween the Render and Render', both are highlighted in blue. After an element-wise addition of the two images and a subsequent division by two, the resulting output (Input') should, in an optimal scenario, be close to the target image (Input). This relation is highlighted in red.

Initially the autoencoder design incorporated a CNN as the encoder, consisting of seven convolutional layers. The first six convolutions used a filter (or kernel) of size 4×4 and a stride of 2 and a padding parameter of 1. Only the last convolution used different values for the stride and padding, because the last convolutional layer was used as a form of classification. The seventh layer reduced the feature count to six features, relating to the required six values for the transform vector, by using a stride of 1 and zero-padding. Though the results were not satisfying our needs, even with different classification layer implementations and we looked into other architectures to improve the learning-based algorithm. The works of [Huang et al., 2016] showed that neural networks based on their so-called Dense Convolutional Network (DenseNet) architecture tend to perform better with both, increased and decreased parameter count and computations. Therefore we implemented a DenseNet as the encoder, which will be covered in the following.

DenseNet The concept of a DenseNet is based on the idea of adding additional "skip connections" between multiple layers, by incorporating the output of one layer into the input of another layer of the network. Skip connections are not a novel concept in DenseNets though, as the predecessor of DenseNets, "Residual Networks" (ResNets) already makes extensive use of them [He et al., 2015]. In the case of a ResNet, skip connections are implemented to skip one or more network layers by adding the output of the previous layer to the output of the last skipped layer. This network architecture has proven to be able to train deeper neural networks with higher accuracy and less complexity compared to so-called "VGG" networks [He et al., 2015]. DenseNets build on that concept by connecting the output of every layer with the output of every consecutive layer, using skip connections. Though, the skip connections do not add the outputs together, they concatenate them. Furthermore, the size of the feature maps (e.g. the output of a convolutional layer) increases drastically, decreasing the performance of the network. Therefore the authors of [Huang et al., 2016] split the network layers into what they refer to as "Dense Blocks" and add transitional layers in between them, that reduce

Operations	Parameters
Conv2d	(kernel_size, stride, padding)
MaxPool2d	(kernel_size, stride, padding)
AvgPool2d	(kernel_size, stride, padding)
AdaptiveAvgPool2d	(output_size)
Linear	(in_features, out_features)

Table 3.1: PyTorch commands, such as Conv2d for 2D convolutional layers, and their most relevant parameters.

Layers	DenseNet-Encoder
Convolution	Conv2d(7×7 , 2, 3)
Pooling	MaxPool2d(3×3 , 2, 1)
Dense Block (1)	$\begin{bmatrix} \text{Conv2d}(1 \times 1, 1, 0) \\ \text{Conv2d}(3 \times 3, 1, 1) \end{bmatrix} \times 4$
Transition Layer (1)	Conv2d(1×1 , 1, 0) AvgPool2d(2×2 , 2, 0)
Dense Block (2)	$\begin{bmatrix} \text{Conv2d}(1 \times 1, 1, 0) \\ \text{Conv2d}(3 \times 3, 1, 1) \end{bmatrix} \times 8$
Transition Layer (2)	Conv2d(1×1 , 1, 0) AvgPool2d(2×2 , 2, 0)
Dense Block (3)	$\begin{bmatrix} \text{Conv2d}(1 \times 1, 1, 0) \\ \text{Conv2d}(3 \times 3, 1, 1) \end{bmatrix} \times 16$
Transition Layer (3)	Conv2d(1×1 , 1, 0) AvgPool2d(2×2 , 2, 0)
Dense Block (4)	$\begin{bmatrix} \text{Conv2d}(1 \times 1, 1, 0) \\ \text{Conv2d}(3 \times 3, 1, 1) \end{bmatrix} \times 12$
Classification Layer	AdaptiveAvgPool2d(1×1) Linear(98, 6)

Table 3.2: The architecture of AFRRMC's encoder split into the different layers.

the feature map size through a convolutional layer and a pooling layer.

The DenseNet implementation of the AFRRMC encoder is based on an example file from a PyTorch repository [PyTorch, 2022] with alterations from [Arora, 2020]. The architecture of the encoder is described in Table 3.2, where the layers and operations are listed using their respective PyTorch method names and parameters. Table 3.1 lists the layer names used in the architectural table and their most relevant parameter names.

The decoder of the AFRRMC algorithm consists of seven fractionally-strided convolutional layers. While these layers do not compute a true inverse of a convolutional layer, they are also often referred to as deconvolution operations. In contrast to convolutional layers, they are able to increase the dimensions of an image, while decreasing the feature map size. Therefore they can be used to reverse convolutional operations and are subsequently used in the decoder design of this work. All except for the first of the layers utilize a kernel of size 4×4 , stride of 2 and a padding of 1. Only the first "deconvolutional" layer uses a stride of 1 and a padding of 0.

3.5.4 Loss Functions

In order to make the network learn, multiple loss functions were designed in order to improve the performance and precision of the network. Three different variants in chronological order will be described in the following.

The initial loss design was based on two forward passes of the encoder, resulting into two transform vectors \mathbf{t}_1 and \mathbf{t}_2 . The function computing the loss $loss_1$ can be abstracted and expressed through the following algebraic notation:

$$loss_1 = \sum_{x=1}^m \mathbf{I}_x + a \sum_{y=1}^6 \mathbf{v}_y + \Upsilon(\mathbf{t}_1) + \Upsilon(\mathbf{t}_2) + \sum_{z=1}^n \mathbf{L}_z, \quad (16)$$

$$\mathbf{I} = |2\mathbf{T} - (\mathbf{F} + \mathbf{R})|, \quad (17)$$

$$\mathbf{v} = |\mathbf{t}_1 - \mathbf{t}_2|, \quad (18)$$

$$\mathbf{L} = |\mathbf{F} - \mathbf{R}|, \quad (19)$$

where the matrix \mathbf{I} is computed by the absolute of the subtraction of the target image \mathbf{T} times two and the addition of the rendered image \mathbf{R} and the decoder's image \mathbf{F} as notated in Equation 17. The vector \mathbf{v} is the absolute of the subtraction of the two transform vectors \mathbf{t}_1 and \mathbf{t}_2 . The factor a consists of the maximum error value for the sum of the comparison image matrix, being 131072, divided by the sum of all elements of the maximum search space vector, amounting to 39.42477. This was aimed to increase the image-based error while the predicted transform was outside of the search space. The added function $\Upsilon(\mathbf{b})$ represents another function that increases the error of the network proportionally to the distance of \mathbf{b} from the maximum search space, while the sum of the matrix \mathbf{L} was added to increase the decoder's performance.

The second loss design moved away from the idea of having a single loss function for both the decoder and encoder, which improved the decoder's performance drastically. The abstracted loss functions $loss_{e2}$ for the encoder and $loss_{d2}$ for the decoder, can be notated through the following:

$$loss_{e2} = \sum_{x=1}^m \mathbf{I}_x, \text{ where} \quad (20)$$

$$\mathbf{I} = \left| \left(\mathbf{T} - \frac{\mathbf{F} + \mathbf{R}}{2} \right)^2 + (\mathbf{R} - \mathbf{F})^2 \right| \text{ and} \quad (21)$$

$$loss_{d2} = \sum_{z=1}^n \mathbf{L}_z, \text{ where} \quad (22)$$

$$\mathbf{L} = |(2(\mathbf{F} - \mathbf{R}))^2|. \quad (23)$$

Though, we noticed that the encoder was able to improve its precision through the concept of reusing predicted values as targets, as in the initial loss function $loss_1$. Leading to the encoder's following and final loss function $loss_{e3}$:

$$loss_{e3} = \left(\sum_{x=1}^m \mathbf{I}_x \right)^2 + \sum_{y=1}^6 \mathbf{v}_y + \Upsilon(\mathbf{t}_2), \text{ where} \quad (24)$$

$$\mathbf{I} = \left| \mathbf{T} - \frac{\mathbf{F} + \mathbf{R}}{2} \right|. \quad (25)$$

3.5.5 Hyperparameter Tuning

Due to the limited time frame of this work, only a limited number of hyperparameter tunings could be performed. The tuning is described in the following.

Optimizers Due to the codebases of other projects often using the Adam optimizer, we initially implemented it as well into our encoder. Though it was noticeable that the Adam optimizer's error frequently fluctuated and after comparing it to the stochastic gradient descent (SGD) optimizer provided by PyTorch, the SGD optimizer outperformed the Adam optimizer as seen in Figure 3.8. Though, the work of [Chauhan et al., 2021] has shown that Adamax generally outperforms the Adam, AdamW and SGD optimizer, as the six best tuning results featured the Adamax optimizer. Consequently the Adamax optimizer was implemented into the AFRRMC's encoder to improve the performance.

Trend Slope Calculation Even though it is not used in the final implementation, a trend slope calculation has proven to be very useful for both debugging and our learning process. Initially, the trend slope was aimed to dynamically adjust the learning rate of the optimizer, based on the current slope of the loss graph. This allowed the algorithm to dynamically respond to plateaus and local minimums using the following calculation.

First we create two arrays, the array \mathbf{x} with matching indexes starting from 1 and array \mathbf{y} with the values of data points. Then we compute four variables: a , b , c and d . a multiplies both of the arrays \mathbf{x} and \mathbf{y} element-wise and then computes the arithmetic mean of the elements. b calculates the average of \mathbf{x} and \mathbf{y} respectively and multiplies the results with each other. c squares the array \mathbf{x} and then computes a mean, while d first computes the mean of \mathbf{x} and then squares it. The final slope is calculated by the formula:

$$\frac{a - b}{c - d}. \quad (26)$$

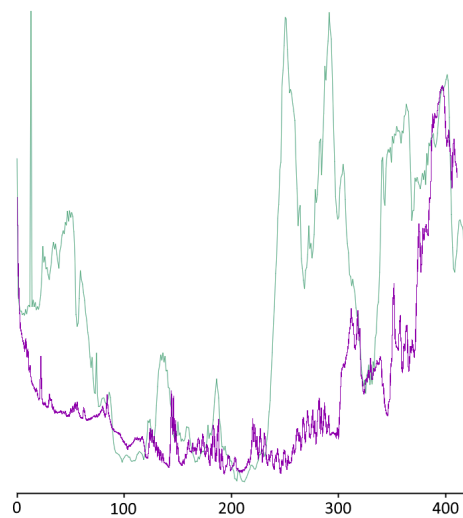


Figure 3.8: The difference in robustness during a training between the Adam optimizer (green) and the SGD optimizer (purple).

Static Parameter Tuning To tune hyperparameters of the network, such as the learning rate of the encoder and decoder or the size of the feature maps in both, several efforts were taken. Initially the Python library Tune from the Ray API was considered due to its accessibility and ease of implementation. Though, the algorithm needs access to Blender in the form of a built Python module, which is not distributed along with regular releases. Therefore we switched to a Bayesian Optimization algorithm from [Nogueira, 2020]. Though, due to the rapid changes in network design, only the learning rate of the decoder and the maximum gradient norm of the encoder were sufficiently tuned.

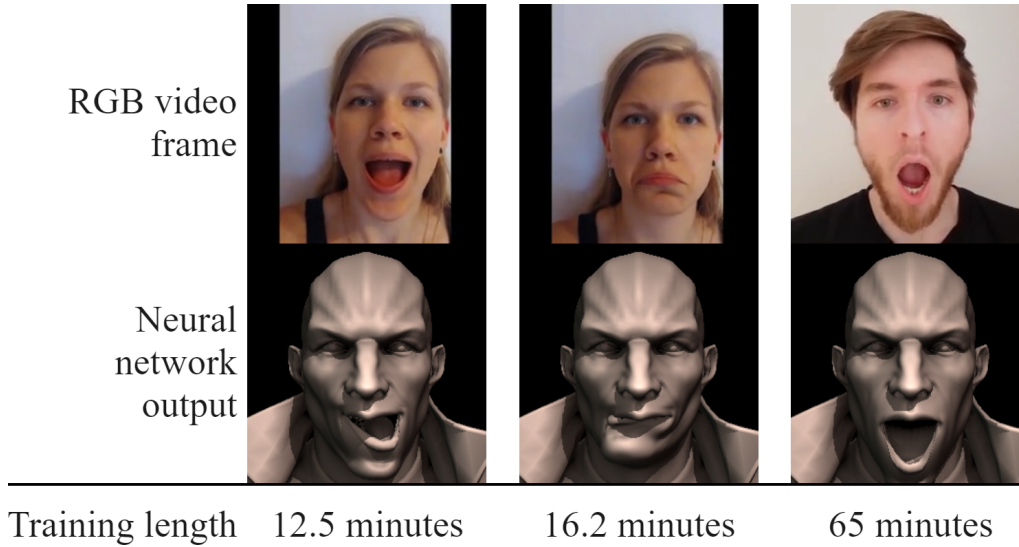


Figure 4.1: Selected results of the AFRRMC algorithm. Top row: The input frames of the AFRRMC pipeline. Bottom row: The corresponding outputs of the trainings.

4 User Study and Evaluation

Due to the subjectivity of 3D animations and the workflows of the field, we conducted a user study to evaluate the algorithm and pipeline previously described. The objective and subjective results of the prototype and the study as well as the structure of the latter are described in the following.

4.1 Study Design

Since the evaluation of the research subject relies on experience with 3D animation and its workflows, we needed to gather animation experts meeting specific requirements. The requirements were defined for our case as follows:

Definition An expert is a person who either worked professionally in the 3D animation field before or is enrolled in some form of 3D animation studies.

Due to the requirements narrowing down the sample size of the study, we aimed for a qualitative user study, based on the concept of expert interviews. We conducted 10 semi-structured interviews which allowed for additional questions and elaborations from both, the interviewer and interviewee. The questions amounted to 69 in total, where the format of the questions varied between single choice, multiple choice, open questions and likert scales. The interview also contained two narrative parts, where the interviewer explained basic concepts necessary to understand the research subject. This included theoretical background of facial motion capture and its weaknesses, deepfake algorithms, a basic understanding of neural network algorithms and trainings, as well as the AFRRMC pipeline and an abstraction of the inner workings of the prototype.

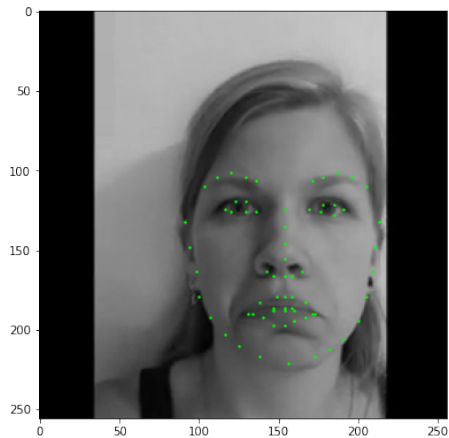


Figure 4.2: The facial landmark tracking algorithm by [Bulat and Tzimiropoulos, 2017].

To be able to compare our method to pre-existing solutions, we chose a point tracking solution by [Bulat and Tzimiropoulos, 2017] from recent available landmark tracking algorithms. Though, the output (as shown in Figure 4.2) cannot be used as a direct comparison. But needs to be resized, which was achieved through bicubic interpolation, and imported into Blender and tracked by an animator. To ensure an optimal comparison, the mapping and tracking was done by hand, while the result was not tweaked in any way. This would follow our goal to automate as much as possible and introduce minimal manual labor during the process.

4.2 Results

The algorithm was trained for 12.5, 16.2 and 65 minutes respectively, resulting into the output poses shown in Figure 4.1. The time to compute one of the poses based on a previously stabilized image, averaged to 1.218 seconds with a standard deviation (SD) of 0.0046 seconds. These timings could be achieved, since only a part of the network and no rendering is necessary to compute the output. The times were measured on a personal computer featuring an NVIDIA GeForce GTX 1080 Ti GPU and an Intel® Core™ i7-6800K CPU.

The participants of the study had averagely 5.725 years of experience with 3D animation and 2.829 years of professional experience in the field. 4 of the 10 interviewees also worked on a AAA game or movie title as an animator or part of an animation related department before. The term AAA lacks a general definition, therefore we defined it as the following: "AAA" refers to a game, movie or other kind of media with photorealistic, high fidelity graphics and a budget of at least USD 10 million.

4.3 Evaluation

To ensure that our problem statement is valid, 11 questions were aimed to answer how much of an issue problems such as motion data mapping, expression intensity matching, rig incompatibility and low resolution data is for 3D animators. All participants confirmed the problem statement and 90% of them encountered it before. The single exception was an animation student who wasn't involved yet with facial animation at the point of the study. Though, none of the interviewees had found a solution for this problem. They were asked to describe how much of the setup process of characters is manual labor, based on a likert scale, where a 1 corresponds to "nothing" and a 10 to "everything". They rated it a 6.8 on average. Another scale was presented to the participants where a 1 corresponds to "enjoyable/quick" and a 10 to "unpleasant/tedious", rating the setup process a 7.7. In response to whether the process was "very easy" (1) or "very hard" (10), the answers averaged to a 6.6. They also rated their experience with facial motion capture from "it's a breeze" (1) to "really tedious" (10) with a mean of 7.2. Regarding the time needed to setup an already existing facial rig for motion capture, the responses of 9 participants averaged to 21 hours with a relatively high SD of 55.17 hours. The last participant was not able to form a numeric answer to the question because "it needs a whole team for that".

Before showing the participants the algorithm's output, we asked them about specific requirements that the prototype would need to meet, for them to implement AFRRMC into their own workflow. Regarding the amount of time needed for the algorithm to train, the interviewees responded with 8.108 days on average, with an SD of 11.868 days. The requirement for the execution time averaged 20.186 hours with an SD of 52.48 hours. Between "the algorithm should be barely usable with a documentation" (1) and "plug and play" (10), the experts averaged to 6.4 points with an SD of 1.713 points. To quantify the quality requirement, a likert scale was presented twice (once before and once after showing the resulting poses) to the participants where they needed to rate between the "resemblance of a human face" (1) and "photorealistic" (10), averaging to a 5.7 with an SD of 1.509 points.

After showing the experts the results of the poses, they rated the quality of the prototype with

a 6, where a 1 corresponds to "resembles a human expression" and a 10 to "photorealistic". The SD of the likert scale amounted to 1.414 points. The usability was rated a 7.5 with an SD of 2.014 points, where 1 corresponds to "very hard to use" and 10 to "very easy to use". When the results of the AFRRMC algorithm were compared with another method, in 60% of the 30 cases the AFRRMC algorithm was preferred. When compared to current facial motion capture solutions, the experts rated the prototype a 5.3 between "much worse" (1) and "much better" (10) on a likert scale, with an SD of 1.767 points.

Out of the 10 experts, 80% would use the AFRRMC prototype in its current state and 50% think that AFRRMC is viable in a large-scale or AAA production, while 6 of the 10 participants explicitly stated that it would be beneficial for smaller sized independent studios. The participants also stated that averagely 56% of their time would be saved if they had access to the prototype, with an SD of 25.033%. When asked what they would like to see primarily improved, 80% of the participants answered either with "the quality" or "the precision". A single expert prioritized the speed of the algorithm to match real-time applications, and another participant prioritized a slider for scaling the expressiveness of the poses.

We also asked the participants about the potential and direction of this specific research and method, where all of them agreed that it has potential, while 90% used one of the preceding words: "a lot", "very" and "big". There were also multiple use cases mentioned for this method, apart from using the output directly as a facial animation. For instance, it could be used for educational purposes. One of the experts mentioned that they would like to show it to students for them to try it out. Another one mentioned that it could be used for computer-animated avatars, where a high amount of facial data needs to be processed in a minimal amount of time. The prototype could also be used as reference, as mentioned by two experts, since usually animators only work based off the sound of the performance, according to one participant. Another mentioned use case is to use the AFRRMC algorithm's output as a baseline and add additional correction layers of animated keyframes on top. Lastly, new research opportunities were mentioned, since the prototype "would allow other kinds of projects that we currently don't even dream about", according to an expert.

5 Discussion

The results of the study suggest that the current animation workflow is flawed and that multiple parts, such as setting up a 3D character rig for facial motion capture, are labor intensive. Not only does the setup require averagely over 20 work hours of the interviewed expert's time, but it is also rather unpleasant and tedious. These findings support the statement from [Zollhöfer et al., 2018], that the animation workflow as a whole is highly time-consuming and labor intensive, and lacks automation considerably. Though, the results have also shown that the proposed method is able to save averagely more than half of the expert's time, while still keeping a certain degree of quality and involving a minimal amount of manual labor. This implies that the automation of the facial motion tracking pipeline was successful. Consequently, the objective of this work has been accomplished.

The findings also suggest that the AFRRMC method can supersede other monocular point tracking methods if minimal manual labor is required. The study also implies that the prototype is more easy to use than current solutions and can even be utilized in learning environments for animation students. The majority consensus between the experts hints to the statement that AFRRMC is useful to the majority of 3D animators. Though, due to the small sample size of our study, more research is needed to confirm the last-mentioned statement.

AFRRMC is also robust to a variety of video inputs in cases including, but not limited to, the ones shown in Figure 2.10, as different lighting setups and actors can be used as shown in Figure 4.1. This suggests that AFRRMC is more robust than the to date most recent work in this research area by [Moser et al., 2021], due to the usage of a deepfake algorithm, eliminating variables such as the actor's identity and environment.

Based on the amount of professional and general experience about 3D animation, we argue that the participants of our study are qualified to participate in this research as "experts" that can effectively evaluate the prototype and method and can be consulted for other questions. Contrary to the previously mentioned results, the relatively high standard deviation in the execution duration requirement question suggests either a use case conflict or a lack of comprehension around the topic of neural networks. Since the prototype generates only a single frame during the execution, the time for one second of facial animation at 30 frames per second, would be 30 seconds if the algorithm takes a single second to compute. Though, 8 of the 10 participants answered with values equal or above of one minute, ranging up to seven days. This seemed contradictory to us, but even after further clarification in the most prominent cases from the interviewer, none of the participants wanted to change their answer. Though, we can still compare the requirements against the actual measurements. The time needed for the algorithm to train takes up 0.27% of the required time, while the execution is averaging 0.0017%. This shows how distant the requirements and the real measurements are, but still confirms that the prototype is able to meet the requirements set by the experts.

While half of the participants responded positively to whether the prototype could be used in a large scale or AAA production, we agree with the majority of the interviewees, that this method is more useful to independent studios or individuals. Larger studios often have their own in-house software and are often able to hire more animators to combat the labor intensive nature of the process, according to the experts.

Next to the previously mentioned, relatively small sample size of the study, time was the most limiting factor. The tight schedule of this work and the extensive prototyping phase with multiple approaches, limited the amount of tuning and optimizations considerably. Even on the same day as the study commenced, final optimizations were implemented into the prototype, leaving smaller time frames for the final trainings of the algorithm. Therefore we argue that the performance and especially the quality and precision can be greatly improved if the prototype is revisited in future works. This statement is also supported by the expert's responses about the potential of the method.

An additional limitation of this work is that due to time constraints, the algorithm could only be tested on a single rig and using only one deepfake. Although we argue that due to the rig registering algorithm, any bone-based facial rig should be processable and should yield results similar to the rig used in Figure 4.1. This last scenario remains untested though and requires further research. Additionally, due to the same constraint, we were not able to test the AFRRMC algorithm using different deepfake algorithms. It might be beneficial to make use of other types of algorithms too, such as full-body deepfake algorithms.

6 Conclusion

A 3D animated face is a psychologically sensitive and structural complex geometry, requiring large amounts of tedious and time-consuming manual labor during the setup and tracking process. The monocular facial motion tracking pipeline is far from a fully automated workflow, but we showed in this work that it can be highly automated, while keeping a certain degree of quality.

We proposed for the first time to the best of our knowledge, an unsupervised image-based learning algorithm that is able to pose a 3D character's bone controllers, matching a single input image. The algorithm does not rely on any pre-generated datasets, apart from the automatically registered rig. The utilization of a deepfake algorithm allows the algorithm to skip certain challenges, such as varying lighting setups, backgrounds and actors.

We also proposed a fully automated end-to-end pipeline that is able to register a 3D character rig in a way that allows our proposed unsupervised neural network to process the rig. This method of learning positional and rotational parameters for virtually any rig allows for a wide range of applications and opens new research opportunities.

Lastly, we proposed a method for comparing facial deformations and poses, utilizing a generic render engine, which was implemented into the neural network's learning algorithm. We created a prototype based on the algorithm and methods previously mentioned and conducted a user study based on the concept of an expert interview for evaluation. According to the experts, AFRRMC is preferred in 60% of the cases when compared to a traditional point tracking method and the study suggests that the method is more easy to use than current facial motion capture solutions. While further research is needed (due to the small sample size of the study) to determine how useful the prototype is, our study already suggests that more than half of the amount of hours an animator spends, setting up a facial rig and tracking the motions, can be saved through our method.

As previously mentioned, the sample size of the study amounted to a relatively small size of 10 participants. To confirm the observations and evaluations of this work, future research with a higher count of participants is needed. The AFRRMC algorithm was also not sufficiently tuned and optimized, leaving room for future endeavors to increase the network's precision and speed. And while we argue that the proposed pipeline, consisting of a deepfake algorithm, a rig registration algorithm and a neural network, is already sufficient for most tasks, more research about their respective architecture and especially about interfaces between the algorithms and the user is needed.

The AFRRMC prototype has shown that it is possible to automate the facial animation workflow as a whole. But this work barely lays out the foundation for future fully automated facial motion capture solutions, making high fidelity facial animations available to every production studio at every level and potentially for the general public.

7 Appendix

Literature Study

To accurately build on existing work, we collected research papers from the following sources: the ACM Digital Library (ACM DL) and publications from DisneyResearch|Studios (abbreviated hereafter as: DRS), who are partners of technology units such as Pixar Animation Studios, Lucasfilm/ILM, Marvel Studios and others. The DRS database was considered, because it contains some publications that are not available in the ACM Digital Library, but are relevant to this topic. Looking at the size of the database, they differ by multiple thousands in scale, ACM offering 2 983 116 publications and DRS just 486 (as of December 13th, 2021). The search terms used were adjusted in favor of DRS, allowing for more publications to be reviewed. The publications of DRS were filtered by the term:

```
face tracking
```

As a result, 11 publications from the DRS database were added to the selection. To incorporate the initially set objective, to automate the facial motion tracking pipeline, the search term was formulated to filter the existing work of the ACM DL appropriately. No restrictions were applied on the time period for either database, as less recent approaches may become more feasible with modern technology and increased computational resources. The search term for the ACM DL is as follows:

```
("automation" OR "automate")  
AND ("facial tracking"  
OR "facial motion capture")  
AND ("workflow" OR "pipeline")
```

Consequently, the ACM Digital Library was able to contribute 38 research papers, yielding 49 publications in total.

To extract the most relevant publications, all of them were reviewed and either selected or discarded following a specific procedure. First, the title and abstract of each paper were read and conference proceedings were additionally manually searched for other relevant publications in the area. By categorizing the findings in terms of how well they fit into the context of our objective, 6 publications from DRS, 14 papers and 11 conference proceedings from the ACM DL were screened out to the best of our knowledge. These values might seem relatively high, but since the specifications of this thesis involve RGB video data, all of the approaches using infrared sensors or other types of depth cameras were discarded. The remaining 6 conference

Search results	Publications
ACM DL	38
DRS	11
Intermediate result	49

Context filter	Publications
ACM DL	-25
DRS	-6
Intermediate result	18

Proceedings	Publications
Replaced	-6
Additional papers	9
Intermediate result	21

Discrepancy filter	Publications
Duplicates	-1
Total	20

Table 7.1: The literature study selection process, amounting to 20 papers being reviewed from the initial 49 search results.

proceedings were replaced by their most relevant papers respectively, resulting in 9 new additions to the selection, increasing the count by 3 to the intermediate result of 21. During a final filtering, one publication was found to be a duplicate of an already selected publication extracted from a proceeding and consequently removed. Therefore, the total number of remaining papers to be analyzed thoroughly amounts to 20 publications, as shown in Table 7.1.

List of Tables

3.1	PyTorch commands, such as Conv2d for 2D convolutional layers, and their most relevant parameters.	21
3.2	The architecture of AFRRMC's encoder split into the different layers.	21
7.1	The literature study selection process, amounting to 20 papers being reviewed from the initial 49 search results.	33

List of Figures

1.1	The uncanny valley as depicted by Mori [Mori, 1970], translation by MacDorman and Kageki [Mori et al., 2012].	1
2.1	A 2D cage of vertices, affecting a center point [Floater, 2003].	6
2.2	The mass-spring system of a physically-based head model by [Kähler et al., 2003]. Top: Relaxed muscle. Bottom: Contracted muscle with the affected nodes marked with \circ . Adapted graphic according to [Kähler et al., 2001].	6
2.3	The linear contraction ($c = 0.5$) of a sphincter muscle fiber to its center. Adapted graphic according to [Kähler et al., 2001].	6
2.4	A facial bone rig created for motion capture with landmarks [Borshukov et al., 2006].	7
2.5	A 3D character with a head and a jaw bone. Top: The rest pose of the skeleton. Bottom: The posed head and jaw bones.	8
2.6	The weight map of the jaw bone applied to a 3D character.	8
2.7	An actor being scanned in the Light Stage 5 [Alexander et al., 2009].	9
2.8	A screenshot of the Faceware Analyzer 3 software tracking an actor’s face.	10
2.9	The optical flow of a steady camera with moving people [Patait, 2019].	10
2.10	Failure cases of the algorithm proposed in [Moser et al., 2021].	11
2.11	Schematic diagram of a neuron, adapted from [Mehlig, 2021].	11
2.12	An example of a 2D convolution using a 3x3 kernel and a stride of 1. The initial input image (A) of size 8x8 pixels is convolved into two feature maps sized 6x6 (B). Followed by two more convolutions yielding four 4x4 (C) and eight 2x2 feature maps (D) respectively. The last operation flattens the feature maps into a one-dimensional vector of size 32 (E).	12
2.13	An illustration of an optimally trained autoencoder. The input and output (Input’) are both marked in red, since they are equal.	13
2.14	An example of a puppeteering deepfake algorithm, based on [Siarohin et al., 2019]. Neutral frame: The neutral face of the target character. Top row: Frames taken from a video of an actor’s performance. Bottom row: Synthesized video data, based on the neutral frame and the respective driving video.	14
3.1	An overview of the proposed pipeline, the dashed box representing the proposed AFRRMC algorithm.	15
3.2	One sample pair of the captured depth dataset using a Microsoft Kinect v2 sensor.	16
3.3	A diagram showing the similarity of the loss, in the case of an impact map containing all of its children (red) or not (blue).	16
3.4	Left: The rendered weight map of the head bone, displayed in a jet colormap by Blender. Right: Impact map, converted from weight map.	17
3.5	Top: The frame used to drive the deepfake and thus the rest of the algorithms. Middle: The classical algorithm after reaching a step size of $1.198 \cdot 10^{-5}$. Bottom: The neural algorithm after 150 iterations.	18
3.6	Top: The training algorithm, utilizing a pre-generated image and transform vector pair. Bottom: The prediction step, where a new training pair is generated using the target image.	19
3.7	An example of how an autoencoder can be used to treat non-differentiable branches in a neural network.	20
3.8	The difference in robustness during a training between the Adam optimizer (green) and the SGD optimizer (purple).	23
4.1	Selected results of the AFRRMC algorithm. Top row: The input frames of the AFRRMC pipeline. Bottom row: The corresponding outputs of the trainings.	25
4.2	The facial landmark tracking algorithm by [Bulat and Tzimiropoulos, 2017].	25

Contents of enclosed CD

1. /BoneTree/ - An example of a generated impact tree from the rig registration algorithm
2. /Deepfakes/ - The synthesized videos using different actors and lighting setups
3. /DepthApproach/ - Captured datasets from a discarded depth approach
4. /LandmarkTrack/ - The results of the landmark tracking algorithm used in the comparison
5. /Notes/ - Contains meeting, prototype design and rough implementation notes, as well as timetables and the disputation slides
6. /Outputs/ - The outputs of all algorithms developed in this thesis
 - 6.1. AFRRMC/ - The outputs of the AFRRMC algorithm
 - 6.2. TradTracking/ - The outputs of the landmark tracking algorithm
7. /Source/ - The source code of the AFRRMC algorithm, different implementation stages and sample files
8. /Study/ - The study design, questionnaire, narration and its results
 - 8.1. Participants/ - The recordings and consent forms of each participant
9. /TargetImages/ - Frames taken from deepfake videos and deprecated test targets
10. /Thesis/ - The \LaTeX source code of the thesis, a Blender file which was used for illustration purposes and a compiled PDF version of the thesis
11. /README.md - A table of contents of the enclosed CD

Bibliography

References

- [Alexander et al., 2009] Alexander, O., Rogers, M., Lambeth, W., Chiang, M., and Debevec, P. (2009). Creating a photoreal digital actor: The digital emily project. In *2009 Conference for Visual Media Production*, pages 176–187. <https://doi.org/10.1109/CVMP.2009.29>.
- [Andrus et al., 2017] Andrus, C., Balint, E., Deng, C., and Coupe, S. (2017). Optical flow-based face tracking in the mummy. In *ACM SIGGRAPH 2017 Talks, SIGGRAPH '17*, New York, NY, USA. Association for Computing Machinery. <https://doi.org/10.1145/3084363.3085048>.
- [Arora, 2020] Arora, A. (2020). Densenet architecture explained with pytorch implementation from torchvision. <https://amaarora.github.io/2020/08/02/densenets.html>.
- [Bermano et al., 2015] Bermano, A., Beeler, T., Kozlov, Y., Bradley, D., Bickel, B., and Gross, M. (2015). Detailed spatio-temporal reconstruction of eyelids. *ACM Trans. Graph.*, 34(4). <https://doi.org/10.1145/2766924>.
- [Bermano et al., 2014] Bermano, A. H., Bradley, D., Beeler, T., Zund, F., Nowrouzezahrai, D., Baran, I., Sorkine-Hornung, O., Pfister, H., Sumner, R. W., Bickel, B., and Gross, M. (2014). Facial performance enhancement using dynamic shape space analysis. *ACM Trans. Graph.*, 33(2). <https://doi.org/10.1145/2546276>.
- [Bhat et al., 2013] Bhat, K. S., Goldenthal, R., Ye, Y., Mallet, R., and Koperwas, M. (2013). High fidelity facial animation capture and retargeting with contours. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '13*, pages 7–14, New York, NY, USA. Association for Computing Machinery. <https://doi.org/10.1145/2485895.2485915>.
- [Bickel et al., 2008] Bickel, B., Lang, M., Botsch, M., Otaduy, M. A., and Gross, M. (2008). Pose-space animation and transfer of facial details. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '08*, pages 57–66, Goslar, DEU. Eurographics Association. <https://dl.acm.org/doi/10.5555/1632592.1632602>.
- [Borshukov et al., 2006] Borshukov, G., Montgomery, J., and Werner, W. (2006). Playable universal capture: Compression and real-time sequencing of image-based facial animation. In *ACM SIGGRAPH 2006 Courses, SIGGRAPH '06*, pages 8–es, New York, NY, USA. Association for Computing Machinery. <https://doi.org/10.1145/1185657.1185848>.
- [Bulat and Tzimiropoulos, 2017] Bulat, A. and Tzimiropoulos, G. (2017). How far are we from solving the 2D & 3D face alignment problem? (and a dataset of 230,000 3D facial landmarks). In *International Conference on Computer Vision*. <https://arxiv.org/abs/1703.07332>.
- [Cantwell et al., 2016] Cantwell, B., Warner, P., Koperwas, M., and Bhat, K. (2016). IIm facial performance capture. In *ACM SIGGRAPH 2016 Talks, SIGGRAPH '16*, New York, NY, USA. Association for Computing Machinery. <https://doi.org/10.1145/2897839.2927438>.
- [Cao et al., 2012] Cao, X., Wei, Y., Wen, F., and Sun, J. (2012). Face alignment by explicit shape regression. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2887–2894. <https://doi.org/10.1109/CVPR.2012.6248015>.

- [Chandran et al., 2020] Chandran, P., Bradley, D., Gross, M., and Beeler, T. (2020). Semantic deep face models. In *2020 International Conference on 3D Vision (3DV)*, pages 345–354. <https://doi.org/10.1109/3DV50981.2020.00044>.
- [Chauhan et al., 2021] Chauhan, T., Palivela, H., and Tiwari, S. (2021). Optimization and fine-tuning of densenet model for classification of covid-19 cases in medical imaging. *International Journal of Information Management Data Insights*, 1(2):100020. <https://doi.org/10.1016/j.jjime.2021.100020>.
- [Floater, 2003] Floater, M. S. (2003). Mean value coordinates. *Computer Aided Geometric Design*, 20(1):19–27. [https://doi.org/10.1016/S0167-8396\(03\)00002-5](https://doi.org/10.1016/S0167-8396(03)00002-5).
- [Fyffe et al., 2015] Fyffe, G., Jones, A., Alexander, O., Ichikari, R., and Debevec, P. (2015). Driving high-resolution facial scans with video performance capture. *ACM Trans. Graph.*, 34(1). <https://doi.org/10.1145/2638549>.
- [Gibet et al., 2011] Gibet, S., Courty, N., Duarte, K., and Naour, T. L. (2011). The signcom system for data-driven animation of interactive virtual signers: Methodology and evaluation. *ACM Trans. Interact. Intell. Syst.*, 1(1). <https://doi.org/10.1145/2030365.2030371>.
- [Goodfellow et al., 2014] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial networks. <https://arxiv.org/abs/1406.2661>.
- [Hanson et al., 2005] Hanson, D., Olney, A., Prilliman, S., Mathews, E., Zielke, M., Hammons, D., Fernandez, R., and Stephanou, H. (2005). Upending the uncanny valley. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 4, AAAI'05*, pages 1728–1729. AAAI Press. <https://dl.acm.org/doi/10.5555/1619566.1619636>.
- [He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385. <http://arxiv.org/abs/1512.03385>.
- [Hendler et al., 2018] Hendler, D., Moser, L., Battulwar, R., Corral, D., Cramer, P., Miller, R., Cloudsdale, R., and Roble, D. (2018). Avengers: Capturing thanos’s complex face. In *ACM SIGGRAPH 2018 Talks, SIGGRAPH '18*, New York, NY, USA. Association for Computing Machinery. <https://doi.org/10.1145/3214745.3214766>.
- [Huang et al., 2016] Huang, G., Liu, Z., and Weinberger, K. Q. (2016). Densely connected convolutional networks. *CoRR*, abs/1608.06993. <https://arxiv.org/abs/1608.06993>.
- [Kähler et al., 2001] Kähler, K., Haber, J., and Seidel, H.-P. (2001). Geometry-based muscle modeling for facial animation. *Watson, Benjamin; Buchanan, John W.: Proceedings Graphics Interface 2001 (GI-2001), Morgan Kaufmann, 37-46 (2001)*. <https://dl.acm.org/doi/10.5555/780986.780992>.
- [Kähler et al., 2003] Kähler, K., Haber, J., and Seidel, H.-P. (2003). Reanimating the dead: Reconstruction of expressive faces from skull data. *ACM Trans. Graph.*, 22(3):554–561. <https://doi.org/10.1145/882262.882307>.
- [Lewis et al., 2014] Lewis, J. P., Anjyo, K., Rhee, T., Zhang, M., Pighin, F., and Deng, Z. (2014). Practice and Theory of Blendshape Facial Models. In Lefebvre, S. and Spagnuolo, M., editors, *Eurographics 2014 - State of the Art Reports*. The Eurographics Association. <https://doi.org/10.2312/egst.20141042>.

- [McDonagh et al., 2016] McDonagh, S., Klaudiny, M., Bradley, D., Beeler, T., Matthews, I., and Mitchell, K. (2016). Synthetic prior design for real-time face tracking. In *2016 Fourth International Conference on 3D Vision (3DV)*, pages 639–648. <https://doi.org/10.1109/3DV.2016.72>.
- [Mehlig, 2021] Mehlig, B. (2021). *Machine Learning with Neural Networks: An Introduction for Scientists and Engineers*. Cambridge University Press. <https://doi.org/10.1017/9781108860604>.
- [Mori, 1970] Mori, M. (1970). Bukimi no tani [the uncanny valley]. *Energy*, 7(4):33–35. <https://ci.nii.ac.jp/naid/10027463083/en/>.
- [Mori et al., 2012] Mori, M., MacDorman, K., and Kageki, N. (2012). The uncanny valley [from the field]. *IEEE Robotics & Automation Magazine*, 19:98–100. <https://doi.org/10.1109/MRA.2012.2192811>.
- [Moser et al., 2021] Moser, L., Chien, C., Williams, M., Serra, J., Hendler, D., and Roble, D. (2021). Semi-supervised video-driven facial animation transfer for production. *ACM Trans. Graph.*, 40(6). <https://doi.org/10.1145/3478513.3480515>.
- [Moser et al., 2017] Moser, L., Hendler, D., and Roble, D. (2017). Masquerade: Fine-scale details for head-mounted camera motion capture data. In *ACM SIGGRAPH 2017 Talks, SIGGRAPH '17*, New York, NY, USA. Association for Computing Machinery. <https://doi.org/10.1145/3084363.3085086>.
- [Moser et al., 2018] Moser, L., Williams, M., Hendler, D., and Roble, D. (2018). High-quality, cost-effective facial motion capture pipeline with 3d regression. In *ACM SIGGRAPH 2018 Talks, SIGGRAPH '18*, New York, NY, USA. Association for Computing Machinery. <https://doi.org/10.1145/3214745.3214755>.
- [Mukundan, 2012] Mukundan, R. (2012). *Skeletal Animation*, chapter 4, pages 53–76. Springer London, London. https://doi.org/10.1007/978-1-4471-2340-8_4.
- [Niedenthal and Brauer, 2012] Niedenthal, P. M. and Brauer, M. (2012). Social functionality of human emotion. *Annual Review of Psychology*, 63(1):259–285. PMID: 22017377. <https://doi.org/10.1146/annurev.psych.121208.131605>.
- [Nogueira, 2020] Nogueira, F. (2020). Bayesianoptimization. <https://github.com/fmfn/BayesianOptimization>.
- [Orvalho et al., 2012] Orvalho, V., Bastos, P., Parke, F., Oliveira, B., and Alvarez, X. (2012). A facial rigging survey. In Cani, M.-P. and Ganovelli, F., editors, *Eurographics 2012 - State of the Art Reports*. The Eurographics Association. <https://doi.org/10.2312/conf/EG2012/stars/183-204>.
- [Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc. <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.

- [Patait, 2019] Patait, A. (2019). An introduction to the nvidia optical flow sdk. <https://developer.nvidia.com/blog/an-introduction-to-the-nvidia-optical-flow-sdk/>.
- [PyTorch, 2022] PyTorch (2022). torchvision. <https://github.com/pytorch/vision/tree/adf8466ea53dc3f7a18c84c3b86ccde0c7cadb7a>.
- [Ribera et al., 2017] Ribera, R. B. i., Zell, E., Lewis, J. P., Noh, J., and Botsch, M. (2017). Facial retargeting with automatic range of motion alignment. *ACM Trans. Graph.*, 36(4). <https://doi.org/10.1145/3072959.3073674>.
- [Savoye, 2018] Savoye, Y. (2018). Cage-based performance capture. In *ACM SIGGRAPH 2018 Courses*, SIGGRAPH '18, New York, NY, USA. Association for Computing Machinery. <https://doi.org/10.1145/3214834.3214836>.
- [Siarohin et al., 2019] Siarohin, A., Lathuilière, S., Tulyakov, S., Ricci, E., and Sebe, N. (2019). First order motion model for image animation. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc. <https://proceedings.neurips.cc/paper/2019/file/31c0b36aef265d9221af80872ceb62f9-Paper.pdf>.
- [Vasconcelos, 2011] Vasconcelos, V. (2011). *Blender 2.5 Character Animation Cookbook*. Packt Publishing. <http://www.virgiliovasconcelos.com/blender-animation-cookbook/>.
- [von der Pahlen et al., 2014] von der Pahlen, J., Jimenez, J., Danvoye, E., Debevec, P., Fyffe, G., and Alexander, O. (2014). Digital ira and beyond: Creating real-time photoreal digital actors. In *ACM SIGGRAPH 2014 Courses*, SIGGRAPH '14, New York, NY, USA. Association for Computing Machinery. <https://doi.org/10.1145/2614028.2615407>.
- [Wäscher and Hachmeister, 2021] Wäscher, T. and Hachmeister, L. (2021). Top 50 - international media corporations. Media data base, Institute of Media and Communications Policy (IfM). <https://www.mediadb.eu/en.html>.
- [Westbrook et al., 2019] Westbrook, K. E., Nessel, T. A., Hohman, M. H., and Varacallo, M. (2019). Anatomy, head and neck, facial muscles. *StatPearls*. <https://www.ncbi.nlm.nih.gov/books/NBK493209/>.
- [Zollhöfer et al., 2018] Zollhöfer, M., Thies, J., Garrido, P., Bradley, D., Beeler, T., Pérez, P., Stamminger, M., Nießner, M., and Theobalt, C. (2018). State of the art on monocular 3D face reconstruction, tracking, and applications. *Computer Graphics Forum*. <https://doi.org/10.1111/cgf.13382>.
- [Zoss et al., 2019] Zoss, G., Beeler, T., Gross, M., and Bradley, D. (2019). Accurate markerless jaw tracking for facial performance capture. *ACM Trans. Graph.*, 38(4). <https://doi.org/10.1145/3306346.3323044>.