Applications of Machine Learning for Character Animation

Stephen Bailey

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-30
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-30.html

May 1, 2021

Applications of Machine Learning for Character Animation

by

Stephen Wells Bailey

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor John Wawrzynek, Chair
Professor Sergey Levine
Professor Greg Niemeyer

Fall 2020

Applications of Machine Learning for Character Animation

Abstract

Applications of Machine Learning for Character Animation

by

Stephen Wells Bailey

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor John Wawrzynek, Chair

An important goal of character animation is to create believable, life-like movements and expressions. For film, artists spend significant amounts of effort to add sufficient complexity to a character rig to enable believable and emotionally evocative performances. However, once a complex character rig has been authored, an artist then needs to spend a significant amount of effort to animate a character and bring it to life. For video games, mesh deformations and geometry processing must be real-time, which affects the types of deformations included in a character rig for an interactive application. As a result, video game characters tend to lack some of the sophisticated deformations and motions seen in film-quality characters.

This dissertation explores applications of machine learning for improving the quality of deformations in real-time character rigs as well as applications to assist artists in producing high-quality animations. We detail a deep learning-based approach to enable complex film-quality mesh deformations to run in real-time for both a character's body and face. Our method learns mesh deformations from an existing character rig and produces an accurate approximation using significantly less computational time. In addition to mesh deformations, we present a statistical approach to synthesize novel animations from a collection of artist-created animations. Thus, single-use animations for film can be leveraged for additional applications. We also present a method for generating facial animation from a recorded performance, which provides artists with an initial animation that can be fine-tuned to meet stylistic and expressive needs.

To my grandparents John and Alice Bailey, who supported me throughout my studies but were unfortunately unable to see me finish.

Contents

List of Figures

List of Tables

Acknowledgments

I would first like to thank my wife Kate Pfeiffer for sticking with me throughout my entire PhD studies. I appreciate her willingness to follow me wherever I go and to listen to my technical talks even though she felt completely confused at times.

My PhD work has been split among Berkeley, DreamWorks Animation, and Unity Technologies, and there are many people to thank from each institution. At DreamWorks, I would first like to thank Carmen Badea. She brought me on as a summer intern, which started my journey in collaborating with the film studio. Next, I would like to thank Martin Watt for mentoring me during my first research project. Although I was not enthusiastic about machine learning at the time, his interest in applying ML to animation inspired me to learn as much as I could about the topic. I would also like to thank Paul DiLorenzo who mentored me for my deformation approximation projects at DreamWorks. I appreciate his effort to ensure that I always had the resources needed to conduct my research and always knew who to put me in contact with when I had questions. I'd like to give a big thanks to Dave Otte for sharing his expertise in character rigging with me as well as his willingness to use his calm, soothing voice to narrate the supplemental videos for my publications. Andrew Pearce deserves my thanks for taking time to proofread my papers as well as helping to ensure that the visuals in my work were approved by the company for publication. Finally, I'd like to thank Dio Gonzalez, Bret Statsny, Iris Cheung, and Greg Junker for their help whenever I had questions regarding software engineering as well as the frequent breaks for tea time.

At Unity Technologies, I'd first like to thank Morten Mikkelsen who gave me the opportunity to research facial animation and who also did anything within his abilities to ensure that I had every possible resource needed to succeed. I'd next like to thank Dan Roarty and Ian Spriggs for creating impressively detailed facial rigs for my research. Equally important was the work of Atri Dave who was willing to pose hundreds of expressions on the facial rig and produce any type of animation that I needed for my work. Finally, I'd like to thank Sean Patrick Sherwin for providing me with countless recordings and facial performances. Most importantly, however, I'd like to thank him for having a good-looking face that I did not mind staring at for hours each day as I conducted my research.

At Berkeley, I will start by thanking my academic advisor James O'Brien for taking me on as a student. I appreciate his willingness to let me freely explore my own research interests along with his guidance that helped me develop into a better researcher. I'd like to acknowledge my dissertation committee members John Wawrzynek, Sergey Levine, and Greg Niemeyer as well as Ren Ng on my qual committee. I'd also like to give thanks to Jonathan Shewchuk who provided valuable feedback on my dissertation. I'd next like to thank Rahul Narain and Tobi Pfaff for their positive influence during my first year of grad school as well as Tobi's help to make my publications sound more "scientific". Finally, I'd like to thank Armin Samii for being supportive throughout my entire PhD experience. I greatly appreciate all of our wine nights as welcome breaks from work.

Chapter 1

Introduction

As computational power has steadily risen over the years, so, too, has the realism and complexity of computer-generated scenes and animations. The complexity and computational needs of techniques used by visual effects artists has kept pace with improvements in computation speed as summarized by Blinn's Law, which states "as technology advances, rendering time remains constant" [118]. For example, a single frame from a recent animated film could range from several hours to several days to render on modern hardware. In 1995, Pixar's original *Toy Story* required similar render times on hardware from that time. However, if rendered on current-day machines, the film would take a fraction of the time to render.

Although image rendering takes a significant portion of the computation time spent generating a movie, other aspects of the film have grown in complexity as well. Character mesh deformations, for example, have also become more computationally demanding over the years. These mesh deformations are driven by character rigs, which controls how a mesh is deformed according to a set of input parameters. As the detail and quality of mesh deformations grow, so, too, does the complexity of the character rig.

At DreamWorks Animation, for example, character rigs were so complex that they were unable to evaluate at interactive rates before the development of LibEE and Premo, their current in-house animation software [153]. Previously, animators would enter numbers in a spreadsheet and would wait for their workstation to compute the deformed mesh and update a character on their screen at non-interactive rates. To keep up with the growing complexity of character rigs, they developed their current software to utilize multi-threaded hardware on high-end computing machines. As a result, animators are now able to adjust rig parameters and see the changes in the deformed character mesh in real-time, which can increase their productivity. Despite these improvements, artist still require a significant amount of time to produce a high-quality character animation. For example, one artist might spend a week of effort to author 5 to 10 seconds of animation for a feature film.

In contrast to film, character rigs for video games and real-time applications are not nearly as sophisticated. These real-time rigs are designed for fast evaluation times and often sacrifice quality and realism for speed. One popular rigging technique for real-time rigs is called linear blend skinning. In this approach, a rig is defined by an underlying skeletal structure, and the mesh deformations are determined by the configuration of bones in the skeleton. Each vertex in the mesh is then deformed through a pre-determined weighted sum of the bone transformations. Despite the limitations of the types of deformations linear blend skinning can achieve, it is still one of the more popular methods for real-time applications due to its simplicity and speed.

1.1 Contributions

This dissertation explores two aspects of character animation: the computational cost of evaluating film-quality character rigs and the labor cost of producing high-quality facial animation. I propose methods based on recent trends in the field of machine learning that significantly reduce evaluation times of mesh deformations as well as reduce the authorship time of novel character animations. I make the following contributions:

- **Fast mesh deformation approximations** This work develops methods to reduce the time required to compute mesh deformations for film-quality rigs. These methods allows for better interactivity with character rigs during animation authoring and use in real-time games and applications. I propose two separate approaches for approximating deformations for the body of the character and the face of the character. Both approaches show significant improvements in rig evaluation time while preserving a high level of accuracy in the deformation approximations.

- **Repurposing existing animation for novel synthesis** I propose a method for automatically animating characters in real-time based on an existing corpus of artist-created animation. This method allows film-quality animation to be repurposed and reused for interactive animation while preserving the stylistic details that define a character's motion. I demonstrate this method through synthesis of expressive facial animation of an animal character from film.

- **Low-cost facial animation through performance capture** This work develops a method to animate a facial rig from a single camera recording of an actor's facial performance. This method works with artist-created facial rigs and can animate the character with recordings captured with inexpensive webcams. This approach does not require high-quality facial scans of the actor nor does it require the geometry of the facial rig to match the shape of a recorded actor's head.

1.2 Dissertation Outline

The goal of this dissertation is to apply machine learning techniques to improve the computational time of character rigs as well as reduce the manual labor required to author new character animations.

First, in Chapter 2, I cover mesh deformations for the body of character rigs. Character bodies are typically rigged through an underlying skeleton consisting of joints and rigid bones. The deformed mesh is then defined as a function of the skeleton configuration. For characters that feature sophisticated mesh deformations, significant computational power might be spent preserving mesh volume when limbs move or generating skin wrinkles when joints are bent. To compress this computation, I present a deep learning-based approach that trains a neural network to approximate the mesh deformations from the original character rig. The network learns a function that maps bone configurations to deformed vertex positions in the character mesh. I evaluate the method on a set of complex character rigs used in feature film production.

Next, in Chapter 3, I address the challenges of approximating facial mesh deformations. The method for approximating body deformations relies on an underlying skeletal structure. However, facial deformations are typically produced through skin sliding and muscle contractions rather than bone movement. As a result, the body approximation in not suited for producing facial deformations. Instead, I propose a method that maps vertex positions to a 2D deformation map. I then construct a deep learning-based model consisting of both dense layers and convolutional layers. This network approximates vertex positions through deformation maps. In addition, I propose a coarse-to-fine approximation approach to focus computational effort on complex regions of facial meshes such as around the eyes and mouth. I demonstrate the effectiveness of this approach on film-quality characters and show that the model can reconstruct fine-scale facial details such as skin wrinkles while running significantly faster than the original character rigs. There is also a beneficial side-effect of approximating both body and facial deformations as neural networks. The trained network parameters represent a compressed version of the character rig and can be shared and evaluated independent of the original rigging software used to author the character.

A fast deformation approximation creates possibilities for new and interactive forms of character control. In Chapter 4, I discuss methods for manipulating and posing characters through both the fast body approximation and the facial approximation. For the body approximation, I present an inverse kinematics (IK) method that works specifically for two-bone limbs such as arms and legs. We implement this method on a mobile device and show that our body approximation combined with this IK method is fast enough to run in real-time on low-powered devices. Applying IK to a facial rig poses a challenge because facial rigs typically do not contain bones.

In this case, IK approaches would compute the gradient of a control point on the mesh with respect to the rig's parameters. For film-quality characters, facial rigs execute custom-written code, and gradient evaluation may either not be possible or prohibitively expensive to compute in real-time. However, because the facial approximation is implemented as a neural network, gradients can be readily computed through the model. Nevertheless, traditional IK methods solve for rig parameters through iterative methods, but the gradient computation through the approximation model is still too complex to evaluate multiple times per frame in real-time. Instead, I propose a learning-based approach that approximates the IK problem as a fixed-length feed-forward neural network. This solution allows for IK to run at interactive rates with the facial approximation model.

In addition to IK methods, I also present other methods to animate characters. Chapter 5 describes a method for synthesizing facial animations. In this approach, a system specifies an emotional expression or a certain pose for a character, and this method synthesizes facial motion to match the specifications. This method employs a nonparametric statistical model that compresses existing artist-created animations into continuous low-dimensional latent spaces. Animations can be created by defining a path through the latent space and then mapping points along the path to facial poses on the character rig. By labeling different regions of the latent spaces with specific emotional expressions, the system can produce animation with specific emotions by generating a path through the corresponding labeled region of latent space. I demonstrate this method through an interactive application in which a character responds in real-time with various expressions based on a user's input.

As one other method of character control, I present a facial performance capture method in Chapter 6. Facial performance capture refers to methods that produce a facial animation that matches a recorded performance of an actor. Typically, these types of methods aim to morph a character rig to match the shape of the recorded actor. To match the geometry of an actor's head, expensive scanning equipment is often required. However, I propose an approach that instead warps an actor's recorded performance to match the appearance of the character model, which avoids using any facial scanning equipment. I propose a deep learning-based style transfer method to warp video frames in pixel space. To produce the animation, I then utilize iterative inverse rendering techniques to deform the facial mesh to match the appearance of the model in the style transferred image.

1.3 Statement of Multiple Authorship and Prior Publication

Some of the research presented in this dissertation has previously been published as papers in which I am the first author. Here, I acknowledge the contributions of my collaborators who helped bring by research into its current state.

I completed the body approximation method [5], published in SIGGRAPH, while working at DreamWorks Animation under the mentorship of Paul DiLorenzo. Both Paul and Dave Otte helped me understand the complexity of production-quality character rigs, and they helped identify the need for fast rig evaluation in film production.

The facial approximation method and the facial IK method [6], also published in SIGGRAPH, were completed with guidance from Paul DiLorenzo as well. In addition, Dalton Omens collaborated in the core design of the method and assisted with the method's implementation.

My work on facial animation synthesis through pre-existing artist-created animation [4] is published in SCA. This work was also completed while I worked with DreamWorks Animation under the mentorship of Martin Watt.

My work on performance facial capture is unpublished. This research has been conducted while working with Unity Technologies under the mentorship of Morten Mikkelsen. The artists Ian Spriggs and Atri Dave have provided me with the character rig and facial expressions used in the work. Additionally, Sean Patrick Sherwin collected all of the recordings used for my research.

Finally, my academic advisor James F. O'Brien provided valuable feedback on all of my research projects detailed in this document.

Chapter 2

Fast and Deep Body Deformations

2.1 Introduction

The level of detail that can be included in character rigs for interactive applications such as video games and virtual reality is limited by computational costs. These types of rigs need to run at interactive rates, and therefore need to be evaluated as quickly as possible. Because of this limitation, real-time character rigs often lack a high level of detail. In contrast, film-quality character rigs are not limited by hard computational constraints and their mesh deformations appear more detailed and complex. Unfortunately, film-quality character rigs are not suitable for real-time applications. A single film-quality rig might be able to run at interactive rates on a high-end machine, but typically only after tremendous effort has been spent to optimize the rig, parallelize its evaluation, and shift parts of the computation

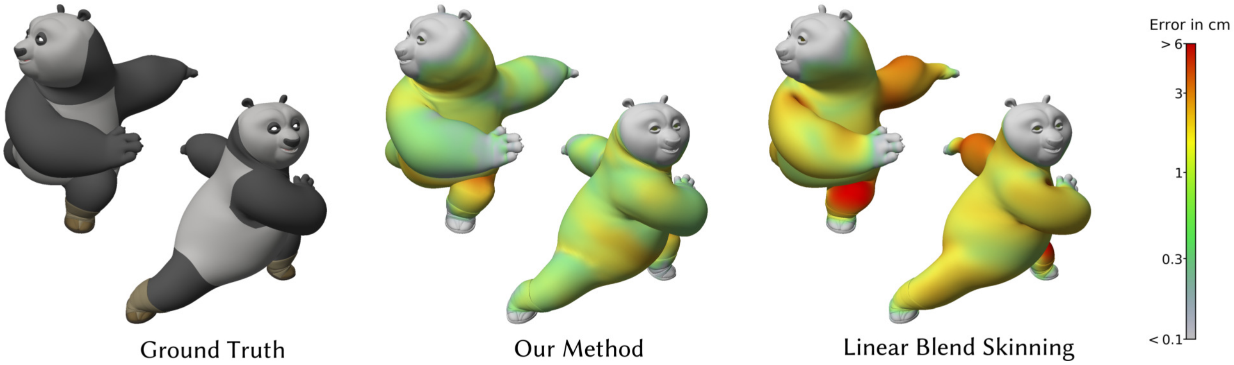


Figure 2.1: Comparison of a deformed mesh using a fully evaluated rig, our fast deformation approximation, and linear blend skinning. The meshes are colored to indicate the distance error for each vertex compared with the ground truth mesh.[1]

[1] These images are Property of DreamWorks Animation L.L.C., used with permission.

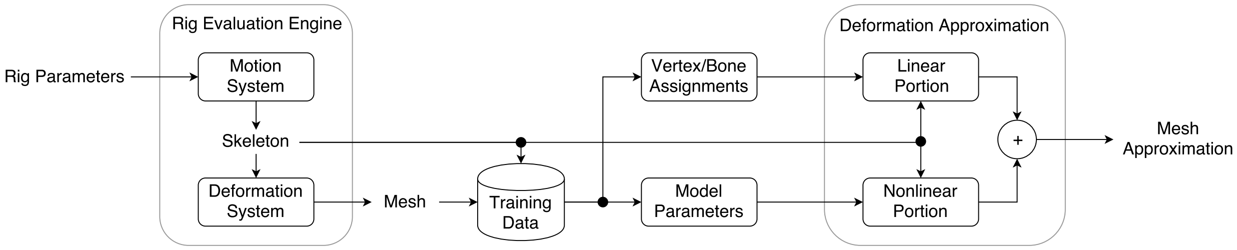


Figure 2.2: Our approximation method learns the deformation system of a character rig by splitting the mesh deformation into a linear portion and a nonlinear portion. The linear approximation uses rigid skinning, and the nonlinear approximation is learned from a set of training examples generated from the original rig evaluation function.

to the high-end machine's GPU. We would like to use these high quality rigs to increase the detail of characters in interactive applications, particularly those running on modest computing platforms such as mobile devices or game consoles. However, directly plugging these computationally intensive rigs into an interactive application is generally infeasible. The performance increases that come as hardware improves over time is unlikely to bring film-quality rigs to real-time, because as performance improves, the level of complexity and fidelity one expects in a film rig also tends to increase. Furthermore, many of these real-time applications need to run on modest computing hardware such as phones or game consoles. Assuming the application is run on a fully loaded high-end machine is reasonable for users who are professional animators, but not for most other users.

To address this limitation, we present a data-driven approach to learn a computationally less expensive approximation for character rigs. Our approximation method reduces the computation enough to evaluate film-quality rigs in real-time on mobile devices. An overview of the method is outlined in Figure 2.2. Most character rigs are designed with two main components: a skeletal motion system and a deformation system [107]. The skeletal motion system is responsible for mapping the input rig parameters that specify a pose to a configuration of the character's skeleton which is composed of bones and joints. The deformation system then maps the skeleton configuration to the final mesh geometry. The deformation system determines how the character's skin moves and includes effects such as muscle bulging and and skin wrinkles. In most character rigs, the deformation computation typically requires the most time and thus is a bottleneck.

We propose a method to approximate the deformation system faster than the original. Given an input skeleton, our system can significantly speed up the overall rig evaluation by dramatically improving the speed of the deformation system. Furthermore, our method achieves a high level of accuracy such that errors are not visually apparent.

2.2 Related Work

A common method for rigging a character involves first defining an underlying skeleton and then deforming the character's mesh based on the positions and orientations of the skeleton's bones. One of the fastest methods to compute the deformation from the skeleton is linear blend skinning or skeleton subspace deformation as described by Magnenat-Thalmann, Laperrière, and Thalmann [102]. This method computes the deformation of a mesh from a rest pose as a weighted sum of the skeleton's bone transformations applied to each vertex. Although linear blend skinning can compute deformations quickly, these deformations can suffer from volume loss and the "candy wrapper" problem. Prior research has explored methods to solve the shortcomings of linear blend skinning. Multi-weight enveloping [151] addresses these problems by using using blending weights for each entry in the bone transformation matrices, and the weights are automatically learned from example poses of the rig. Quaternion-based methods [50], such as spherical blend skinning [67], are other approaches that address the limitations of linear blend skinning without significantly increasing the computational cost of the deformation.

Although linear blend skinning provides a fast method to compute mesh deformations, there are some types of deformation that are challenging to express with this approach. For example, skin slide, muscle bulges, and cloth wrinkles are difficult to achieve using only linear blend skinning. These effects, however, can be achieved using additional skinning methods at the cost of additional computation. Some of these methods include pose space deformations [94, 135] and cage-based deformations [63, 64]. Realistic character deformations can also be computed through physics-based approaches [25], and highly realistic results can be achieved by accurately modeling the underlying anatomy of a character [89].

Skinning decomposition is the process of identifying bone transformations and bone-vertex weights to best approximate a given animation with linear blend skinning. Proposed solutions to the skinning decomposition problem provide a compressed representation of the animation as well as an efficient method to play back the animation in real-time. Prior research has explored methods to approximate arbitrary deformations [60, 68, 65, 85]. These methods seek to fit a bone structure to a series of mesh animations and optimize the bone influences for each vertex to best reconstruct the original animation. Alternatively, a volumetric approximation can be fitted to an animation using sphere-meshes [139], and linear blend weights can be quickly computed with respect to the underlying spheres. With these methods, large animations can be efficiently played back using hardware acceleration, and the deformed meshes can be stored in a compressed format given the fitted bone structure or sphere-meshes. One drawback of these approaches is that new animations cannot be quickly fitted to the rigs because the bones are optimized for a specific set of deformations. Furthermore, an animator would need to learn to use the fitted bone structure in order to author new animations. Example-based skinning methods have been developed [109, 150, 33, 112] that uses the original skeleton from a character rig. These methods use training examples to learn a deformation model that can approximate mesh deformations given new skeleton poses not seen during model training.

A linearization method [66] computes deformations by adding virtual bones to approximate nonlinear deformations in a character. This method utilizes the underlying skeleton of a rig, which allows new skeletal motion to be easily applied to the rig. However, that algorithm works only for deformations computed through a differentiable skinning technique such as dual quaternion blend skinning. Our method, on the other hand, allows for more general deformations and only assumes that the deformations can be computed as a function of the character's skeleton.

Because these skinning decomposition methods compute a compact representation of an animation with bones using linear blend skinning, the deformation evaluation is fast and efficient. However, editing the compressed animations can be difficult because the computed bones are not organized in any meaningful hierarchy. Some work [126, 31, 49, 84] has addressed this limitation by extracting a skeleton structure with joints while also computing bone transformations and vertex-bone weights for an example animation. By providing a hierarchical skeletal system, an animator can more easily edit an existing motion, but extra computation would be required to fit the skeleton to a new animation of the same mesh. These methods approximate mesh deformations when existing animations are provided with or without an underlying skeleton. Our method, in contrast, approximates deformations for a mesh without any example animations; however, it does require that the mesh have an underlying skeleton. We use the underlying skeleton of a character rig without modification, which lets an animator author new poses using the familiar original rig while benefiting from our fast approximate evaluation.

All of these previous skinning decomposition methods seek to find compact representation of an animation using bones with linear blend skinning. Because linear blending is fast, the compressed animations can be computed quickly, but the limitations of linear blend skinning can cause inaccuracies and undesirable artifacts. To improve the speed of linear blend skinning, a sparseness constraint must be imposed on the vertex-bone weights. In some cases where a vertex is influenced by a large number of bones, this sparseness constraint can lead to large inaccuracies in the approximation. One proposed solution [86] to this problem is to compress an animation with a two-layer linear blend skinning model, which accurately reduces the computational cost of evaluating dense skinning weights.

Our approach is also based on decomposing a deformation with linear blend skinning. To reduce the computational cost, we assign each vertex to a single bone, but to overcome the limitations of this skinning method, we also propose an extra nonlinear step, which is modeled as a function of all the bones that influence a vertex. Although our approach does require more computation than a rig using linear blend skinning, we show that the deformations can still be computed efficiently for real-time applications and that our approximation approach can reproduce deformations to a high level of accuracy on film-quality rigs as seen in our results.

2.3 Method

The rig function $\mathbf{r}(\mathbf{p})$ maps a set of artist-level rig parameters, denoted with $\mathbf{p}$, to a deformed polygonal mesh. We follow a similar notation for the rig function as described in Hahn et al. [47], and we assume that the rig function is a black-box. We further assume that the topology of the mesh is constant for all possible parameters $\mathbf{p}$, which allows us to express the rig function as $\mathbf{V} = \mathbf{r}(\mathbf{p})$ where $\mathbf{V}$ is a list of the vertex positions in the mesh. An intermediate step of the rig function computes the skeleton $\mathbf{S}$ of a character. The skeleton's configuration is specified by a set of linear transformations and translations for each bone in the skeleton. Specifically for a skeleton with m bones, $\mathbf{S} = [\mathbf{X}_1, \mathbf{t}_1, \mathbf{X}_2, \mathbf{t}_2, \ldots, \mathbf{X}_m, \mathbf{t}_m]$ where $\mathbf{X}_j$ is the 3×3 linear transformation matrix of bone j and $\mathbf{t}_j$ is the translation of bone j. The transformations and translations are expressed in a global coordinate frame. We further assume that the rig function can be expressed as the composition of two functions: a skeletal motion system mapping rig parameters to a skeleton and a deformation system mapping a skeleton to vertex positions. The skeletal motion system is denoted by $\mathbf{S} = \mathbf{m}(\mathbf{p})$, and the deformation system is denoted by $\mathbf{V} = \mathbf{d}(\mathbf{S})$. Composing these two systems, the rig function can be expressed as $\mathbf{r}(\mathbf{p}) = (\mathbf{d} \circ \mathbf{m})(\mathbf{p})$.

Our method provides an approach to approximate the deformation function $\mathbf{d}(\mathbf{S})$ by decomposing the function into two parts: a linear computation and a nonlinear computation. The linear portion uses rigid rotations and translations to deform the vertices in the mesh according to the bone transformations in the skeleton. This computation is fast, but the resulting mesh is visibly different from the target mesh $\mathbf{V} = \mathbf{d}(\mathbf{S})$. To correct this difference the nonlinear component utilizes a universal function approximator to estimate the remaining residual error between the mesh obtained from rigid rotations and the target mesh. The nonlinear function approximator learns from a set of randomly generated skeletons and corresponding deformed meshes that are computed offline using the rig function $\mathbf{r}(\mathbf{p})$.

Deformation Approximation

We view the rig function as a deformation applied to a mesh in some rest pose. This deformation has linear and nonlinear components, and when combined the two components fully describe the deformation applied to the mesh.

Linear Skinning

The linear deformation can be applied directly from the input skeleton by multiplying the vertices in the mesh with the bone transformation matrices. In our skinning method, we assign each vertex to a single bone where the vertex k is assigned to bone b_k. Starting with a mesh in a rest pose $\mathbf{V}^0$ and the corresponding skeleton $\mathbf{S}^0$, the linear deformation by a new skeleton $\mathbf{S}$ for vertex k can be computed as

$$\hat{\mathbf{d}}_k(\mathbf{S}) = \mathbf{X}_{b_k} \left(\mathbf{X}^0_{b_k}\right)^{-1} \left(\mathbf{v}^0_k - \mathbf{t}^0_{b_k}\right) + \mathbf{t}_{b_k} \tag{2.1}$$

where $\mathbf{X}^0_{b_k}$ and $\mathbf{t}^0_{b_k}$ are the transformation matrix and translation vector for bone b_k in the skeleton $\mathbf{S}^0$ of the rest pose, and $\mathbf{v}^0_k$ is the position of vertex k in the mesh of the rest pose.

We assume that we only have black-box access to the deformation function, and we therefore cannot rely on information about the character rig to identify vertex-bone assignments. Instead, we assign each vertex to a single bone that best explains the vertex's deformation across a set of example poses. The bone assignment b_k is determined by selecting the bone which minimizes the least squares error of the rigid transformation of the vertex by the bone.

The linear deformation is visibly different from the target deformation where $||\mathbf{d}(\mathbf{S}) - \hat{\mathbf{d}}(\mathbf{S})||^2 \gg 0$. We view this residual error as a nonlinear function, which allows for sophisticated stretching, compression, and volume preservation. These features cannot be handled by a linear transformation alone.

Nonlinear Deformation

The residual $\mathbf{d}(\mathbf{S}) - \hat{\mathbf{d}}(\mathbf{S})$ expresses the error in terms of the global coordinate system and thus depends on global transformations of the skeleton. Ideally, we would like to express the residual in such a way that the error for some vertex k is only affected by a local neighborhood of bones in the skeleton. By representing the residual locally for each vertex, the error function becomes easier to learn because the residual for each vertex no longer depends on the transformations of every ancestor bone in the skeleton hierarchy.

To specify the residual locally, we define the nonlinear deformation function for some vertex k as follows

$$\mathbf{f}_k(\mathbf{S}) = (\mathbf{X}_{b_k})^{-1}(\mathbf{d}_k(\mathbf{S}) - \mathbf{t}_{b_k}) - \left(\mathbf{X}^0_{b_k}\right)^{-1}\left(\mathbf{v}^0_k - \mathbf{t}^0_{b_k}\right) \tag{2.2}$$

where $\mathbf{d}_k(\mathbf{S})$ is the position of vertex k as computed from the original rig deformation function. This function removes the transformation of the rest pose from the vertex $\mathbf{v}^0_k$ and the transformation of the deformed pose from the deformed vertex $\mathbf{d}_k(\mathbf{S})$. The difference of these two positions gives us the nonlinear deformation of the vertex in the coordinate space of the bone b_k.

The deformation function can now be expressed as

$$\mathbf{d}_k(\mathbf{S}) = \mathbf{X}_{b_k}\Big(\left(\mathbf{X}^0_{b_k}\right)^{-1}\left(\mathbf{v}^0_k - \mathbf{t}^0_{b_k}\right) + \mathbf{f}_k(\mathbf{S})\Big) + \mathbf{t}_{b_k} \tag{2.3}$$

We denote our approximation with model parameters θ as $\mathbf{n}_k(\mathbf{S};\theta) \approx \mathbf{f}_k(\mathbf{S})$, and the deformation approximation $\tilde{\mathbf{d}}_k(\mathbf{S};\theta)$ can be expressed as the sum of the linear and nonlinear functions

$$\tilde{\mathbf{d}}_k(\mathbf{S};\theta) = \hat{\mathbf{d}}_k(\mathbf{S}) + \mathbf{X}_{b_k}\mathbf{n}_k(\mathbf{S};\theta) \tag{2.4}$$

The optimal model parameters $\hat{\theta}$ are estimated by minimizing the squared error loss over a set of n training examples

$$\hat{\theta} = \arg\min_{\theta} \sum_{i=1}^{n} \left\| \mathbf{d}_k(\mathbf{S}^i) - \tilde{\mathbf{d}}_k(\mathbf{S}^i;\theta) \right\|^2 \tag{2.5}$$

Instead of using one function approximator per vertex, we group the vertices into subsets and train a function approximator that outputs each vertex in the subset. In order to take advantage of the local deformation defined in $\mathbf{f}(\mathbf{S})$, we separate the vertices of the mesh into subsets P_i based on the bones that they are assigned to such that $P_i = \{k \mid b_k = i\}$. By dividing the vertices into sets this way, the nonlinear deformations for vertices in set P_i are defined in the same coordinate system, which makes the deformation function easier to learn.

Implementation

Because the function $\mathbf{f}_{P_i}(\mathbf{S})$ can be highly nonlinear, we need a model that is capable of learning arbitrary continuous functions. Feed-forward neural networks are universal function approximators [56] that can learn such functions. Given any continuous function, a neural network of sufficient size can approximate the function arbitrarily closely. This property thus makes neural networks good candidates for approximating the nonlinear deformation component of the rig function.

In our experiments, we trained each neural network with two fully connected hidden layers and a dense output layer. In the following section, we describe how we determine the number of layers and the number of hidden units per layer. The hidden layers used the tanh nonlinearity, and the output layer was a dense linear layer. Other activation functions such as the rectified linear unit [41] could have been used, but we only evaluated tanh in our work. We trained each network on inputs of the bone transformation matrices and the translation vectors given in the frame of reference of the parent bone. The transformation matrix for bone j with parent p is given as $\mathbf{X}_p^{-1}\mathbf{X}_j$, and the translation vector is given as $\mathbf{X}_p^{-1}(\mathbf{t}_j - \mathbf{t}_p)$. The root bone is not provided as an input. In total, each bone contributed 12 inputs to the neural network. The models were trained using the Adam optimization method [70] with the following values for the parameters: $\alpha = 0.01$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$.

Model Sparsification

The method presented so far works well to approximate the nonlinear deformation function, but similar results can be achieved with less computation per neural network. We can increase the speed of the approximation without significantly affecting the accuracy by identifying and removing extra computations in the models. We explored four approaches to reduce the size of the approximation model: reduce the size and number of the hidden layers in each neural network, reduce the dimension of the inputs, reduce the dimensions of the outputs, and reduce the total number of neural networks evaluated. Reducing the size and number of hidden layers can be done empirically, and we found that two layers each of 128 nodes in each neural network worked well for our character rigs.

Feed-forward neural networks are composed of a series of dense layers where the output $\mathbf{x}_{i+1}$ of layer i is used as the input of the next layer. The output for some layer i is computed as follows

$$\mathbf{x}_{i+1} = f(\mathbf{W}_i\mathbf{x}_i + \mathbf{b}_i) \tag{2.6}$$

where $\mathbf{W}_i$ and $\mathbf{b}_i$ are unknown parameters that are learned when the model is trained. The function $f(\mathbf{x})$ is a nonlinear function applied element-wise to the components of the input vector. The most time-consuming part of Equation 2.6 is the matrix-vector product $\mathbf{W}_i\mathbf{x}_i$. If the matrix $\mathbf{W}_i$ is $m \times n$, then the complexity of calculating the product is $\mathcal{O}(mn)$. Therefore, to reduce the computational complexity of evaluating the neural network models, we need to reduce the sizes of the weight matrices $\mathbf{W}_i$.

Input Reduction

Evaluating the first layer of the network involves a large amount of computation because the dimension of the input is large. For some set of vertices P_i, the nonlinear deformation $\mathbf{f}_{P_i}(\mathbf{S})$ is expressed locally with respect to bone i according to Equation 2.2. Because the vertices P_i are primarily deformed by bones near bone i, this deformation function depends only on the local bones near the vertices in this set and does not require all of the bones as input. This invariance to some of the input bones is a direct consequence of the formulation of $\mathbf{f}_{P_i}(\mathbf{S})$ in a local coordinate space of bone i.

Assuming that we have access to the original rig function $\mathbf{r}(\mathbf{p})$ and the skeleton $\mathbf{S} = \mathbf{m}(\mathbf{p})$, we can identify which bones most affect the vertices in P_i. Starting in an arbitrarily selected example pose $\mathbf{p}'$, we perturb rig controls that affect the bones one at a time and record which bones caused a change in the function $\mathbf{f}_{P_i}(\mathbf{S})$. This process is repeated with multiple example poses and with large perturbations to ensure that all bones affecting vertices in P_i are identified. We define a subset of the skeleton $\mathbf{S}_{P_i}$ as the set of all bones that influence any of the vertices in P_i. We then use this subset of bones as the input to the model approximating the nonlinear deformation function for these vertices. The rigs we tested contained between 100 and 200 bones. After reducing the number of input bones, each set of vertices tended to have around 20 bones that contributed to their deformation. By using this reduced input set, the computational cost of the first layer for each model can be significantly reduced.

Output Reduction

Next, we consider the size of the output layer. The output contains 3 values per vertex, and for the rigs that we tested, there were on the order of hundreds of vertex positions that each neural network approximated. Unlike the input layer, each dimension of the output needs to be predicted. However, these outputs are highly correlated with each other. With this in mind, we propose using a linear dimensionality reduction method to reduce the size of the output. Our approach is similar to the approach used by Laine et al. [78] in which they use PCA to initialize the final layer in their network to output vertex positions.

With the data used to train each model, we run PCA on each matrix $\mathbf{V}_{P_i}^{1...n}$ containing all of the vertex positions for set P_i across all n poses in the training set. The matrix $\mathbf{V}_{P_i}^{1...n}$ is a $3|P_i| \times n$ matrix where there are $|P_i|$ vertices and n training examples. PCA gives us a transformation $\mathbf{T}$ that maps the set of vertex positions to a lower dimensional space. Next, we need to determine how many principal components to use in the linear transformation $\mathbf{T}$. Keeping more components will increase the accuracy of the model at the cost of more computation time. We decide the number of components to keep by finding the minimum amount that keeps the reconstruction error $\left\|\mathbf{V}_{P_i}^{1...n} - \mathbf{T}^T\mathbf{T}\mathbf{V}_{P_i}^{1...n}\right\|_F^2$ below some user-specified threshold.

In our experiments, we found that keeping the average per-vertex distance error below 0.03 cm was sufficient to maintain the visual accuracy of the approximation without adding too many principal components to the transformation. This threshold choice lead on average to 20-30 principal components per model, which provided a reasonable balance between speed and accuracy. Once we found the transformation $\mathbf{T}$, we appended it to the end of the neural network model as a final dense layer with a linear activation. When the model is trained, the weights of this last layer are not optimized so that the transformation is maintained.

Model Count Reduction

One final approach to reduce the computation of the approximation is to reduce the total number of neural networks in the approximation model. In our method as currently described so far, one model is trained per bone; however, we found that some bones had few vertices assigned to them. As a result, we were training some models to predict the deformation of a small set of vertices. These neural networks can be removed, and their vertices can be reassigned to other bones.

To remove networks approximating small subsets of vertices, we greedily removed the bone with the fewest vertices assigned to it and iteratively recomputed the vertex subsets P_i. We continued this process until the average vertex assignment error

$$e = \sum_{i=1}^{n} \left\|\mathbf{V}^i - \hat{\mathbf{d}}(\mathbf{S^i})\right\|_F^2 \tag{2.7}$$

grew larger than some pre-defined threshold. Before removing any bone, we recorded the best average vertex assignment e_0 error given by Equation 2.7. Next, we removed bones one at a time. In each iteration, the bone with the fewest number of vertices was removed, and the vertices assigned to that bone were reassigned to the next best bone that minimized the error.

At each iteration, we recomputed the assignment error e_i, and we stopped this procedure when $e_i > \tau e_0$ for some scaling factor $\tau > 1$. In our experimental rigs we found that values of $\tau \in [1.1, 1.5]$ worked well. Higher values of τ will lead to fewer models that need to be trained, but fewer models could lead to larger approximation errors. If a small value of τ is chosen, then more models will be used, but the approximation errors will be smaller. Thus, the choice of τ provides a trade-off between speed and accuracy in the approximation.

Data Generation

The choice of training data is important for the rig approximator's accuracy when run on test inputs. Feed-forward neural networks do not extrapolate well from training data, and therefore, the training data needs to span the range of all possible poses that could be expected as inputs when the approximator is used. However, if the training set includes a large range of motion with improbable poses such as arms rotated into the torso or body parts stretched to twice their length, then these types of poses would represent large deformations that the approximator would need to learn. As a result, the neural network would learn these large deformations while sacrificing accuracy for smaller deformations. However, we desire a high accuracy for these smaller deformations because they are more likely to be encountered when the model is evaluated.

Here, we describe a method to create a data set that contains all of the probable poses while avoiding poses with large deformations that are unlikely to occur in an animation. First, we consider each joint in the skeleton independently. For each joint, we manually identify a reasonable range of motion for the rotation and scaling. For example, we might specify the range of the knee joint from 0 to 150 degrees. We define a range for each joint in the skeleton and generate new poses by randomly sampling independently from each joint range. Each value is sampled from a Gaussian distribution with 1.5 standard deviation aligned with and scaled to the specified range. Specifically for some rig parameter with a range $[a, b]$, the parameter values are drawn from the following Gaussian distribution:

$$\mathcal{N}\left(0.5 \cdot (a + b), 1.5 \cdot (b - a)\right) \tag{2.8}$$

We re-sample values that lie outside of the range $[a, b]$. This sampling method ensures that the full range of motion for each joint is contained in the training set. Samples near the ends of the range of motion occur in the data set less frequently. If we assume that poses near the ends of the joint range create poses that an animator typically will not use, then because there are fewer of these examples in the training set, the approximator will focus on learning the deformations near the middle of the range of motion.

Our sampling method creates poses that globally appear invalid. Because our approximation method learns deformations locally, the global appearance of the character is less important than the local deformations around the joints of the character. Because each joint is sampled within the user-defined range of motion, meaningful local deformations of the mesh are contained in the samples, and our approximator can accurately learn from these example poses. Figure 2.3 shows several examples of poses generated by our method.

Other sampling methods could be used where appropriate. For example, a character that has a walking mode and flying mode that it switches between could use bimodal sampling. Additionally, semantic knowledge about how a character moves could be used for customized sampling. If example animations for the character are available, then supersampling methods could be used [55] to generate example poses similar to the animation.

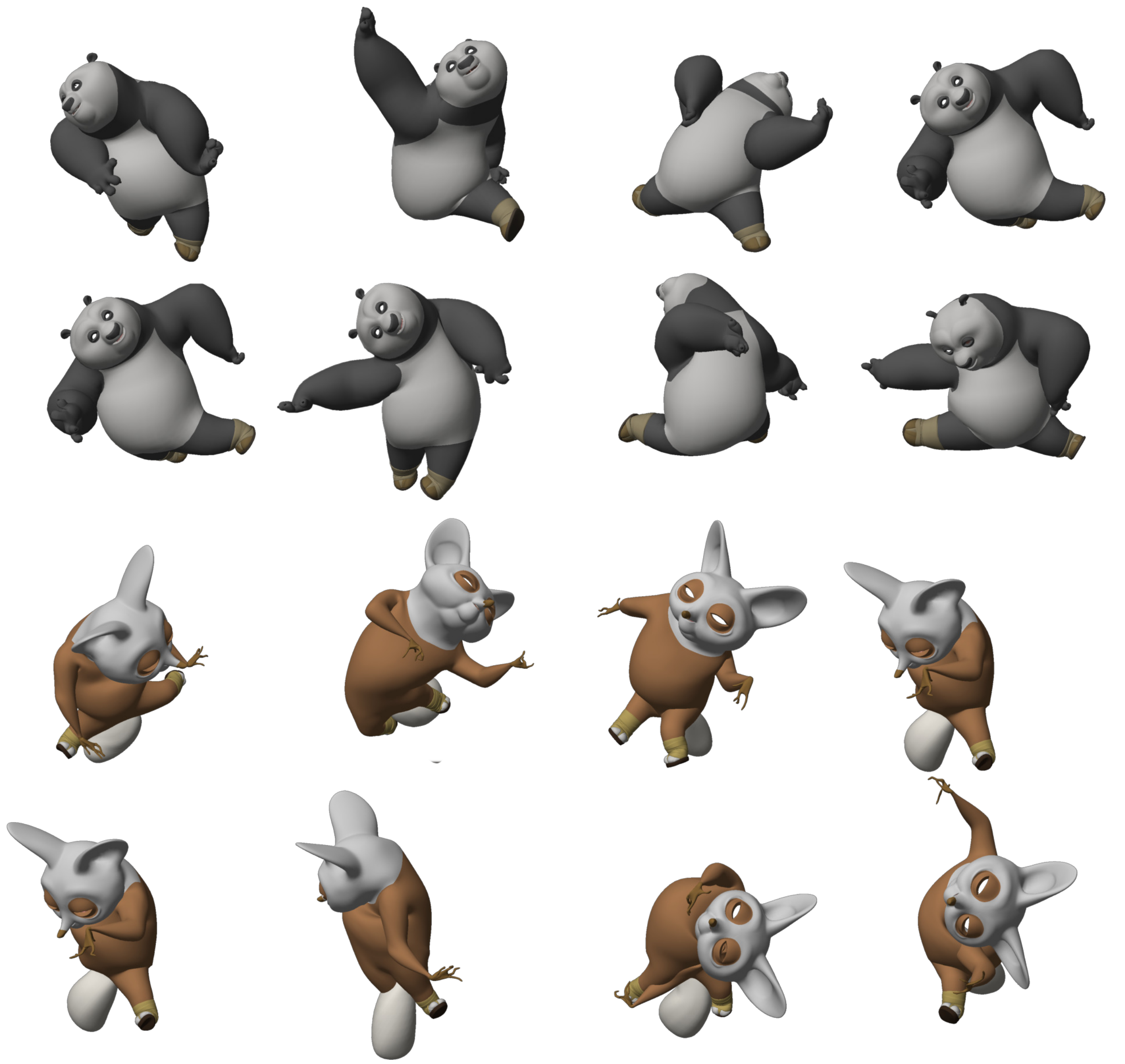


Figure 2.3: Example poses of Po and Shifu created by our data generation method. The poses do not look like anything an artist would create, but the local deformations of the mesh are still meaningful.[1]

Table 2.1: Statistics of the approximation models trained for the character rigs.

	Tigress	Shifu	Astrid	Po
Vertices	16,206	14,706	168,635	13,800
Character Height	182 cm	86 cm	194 cm	191 cm
Models	67	73	45	40
Avg. PCs used	22.6	19.5	24.5	29.7
Avg. input bones	26.9	22.3	14.6	57.75
Model memory size	11.5 MB	10.7 MB	67.5 MB	27.5 MB

2.4 Results

We are interested in two key aspects of our deformation approximation: model accuracy and model speed. Model accuracy is important because we want to minimize the visual differences between the approximated mesh and the original deformed mesh. If the approximation is noticeably different than the original, then this method might not be suitable for all applications. The speed of the approximation is also important. On the rigs that we tested, our method took significantly less time to evaluate the rig compared with the original rig function. With a fast rig approximation, a highly complicated character can be evaluated at interactive rates even on low-end machines and mobile devices. See the accompanying video[2] for animation examples.

We approximated the deformation functions of four film-quality character rigs: Po, Shifu, and Tigress from *Kung Fu Panda 3*, and Astrid from *How to Train Your Dragon 2*. All of these character rigs were originally optimized to run on high-end machines. Table 2.1 shows the size of the models trained for all four rigs including the total memory required to evaluate the approximation models. In all of the approximators, we used two nonlinear hidden layers with the tanh activation function, and each hidden layer consisted of 128 nodes. We found that generating between 10,000 and 20,000 example poses was sufficient to train accurate approximation models for each rig.

Model Accuracy

To evaluate the accuracy of our models, we measured the average per-vertex distance error of the approximated mesh as well as the largest single vertex error. We are interested in the largest error because even if the average error is small, a single misplaced vertex could create undesirable results. For each character, we evaluated a walk-cycle animation and computed the errors across all frames of the animation. The approximation errors are reported in Table 2.2. Figure 2.6 shows the distribution of vertex errors for each walking animation.

[2] `http://graphics.berkeley.edu/papers/Bailey-FDD-2018-08/Bailey-FDD-2018-08.mp4`

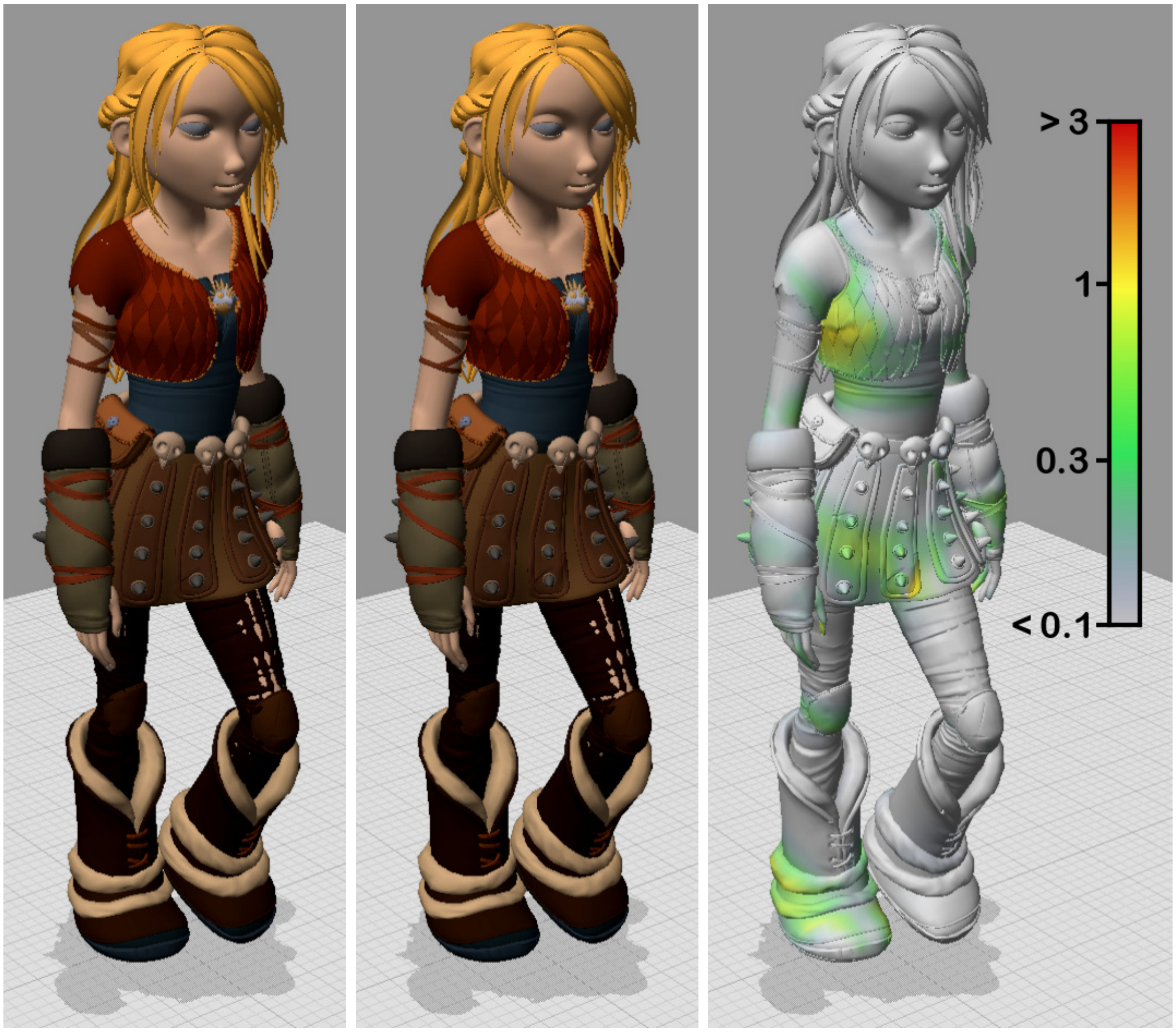


Figure 2.4: Side-by-side comparison of the ground truth mesh (left) and the approximation (center). The vertices of the approximated mesh are colored to indicate the per-vertex distance error (right). Errors above and below the range of the scale are clamped to the ends of the color range. All distances are measured in centimeters.[1]

From these plots, we observe the number of vertices falls off roughly exponentially with the distance error.

Figure 2.4 shows a side-by-side comparison of the mesh generated through the full rig evaluation and the approximated mesh for a frame of Astrid walking. Our method does not handle facial animation, and for each tested animation, we turned off all face controls. A region with large error in the approximation is found in the middle of Astrid's skirt. The deformation of this area is controlled by both legs, which appears to cause some inaccuracy

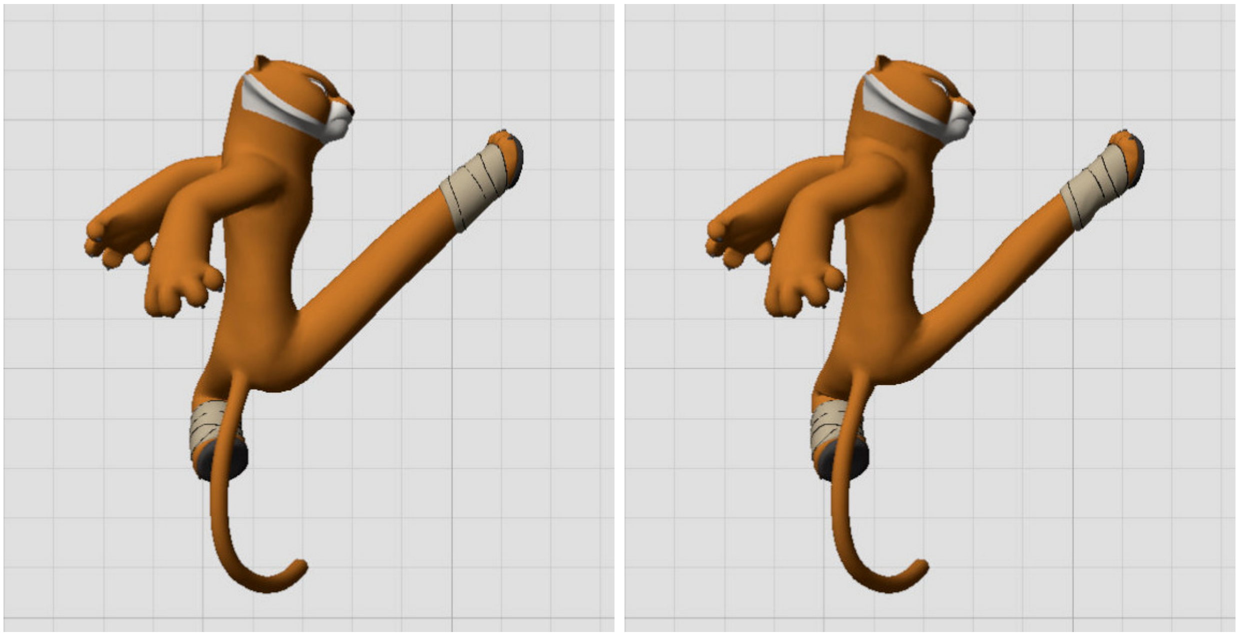


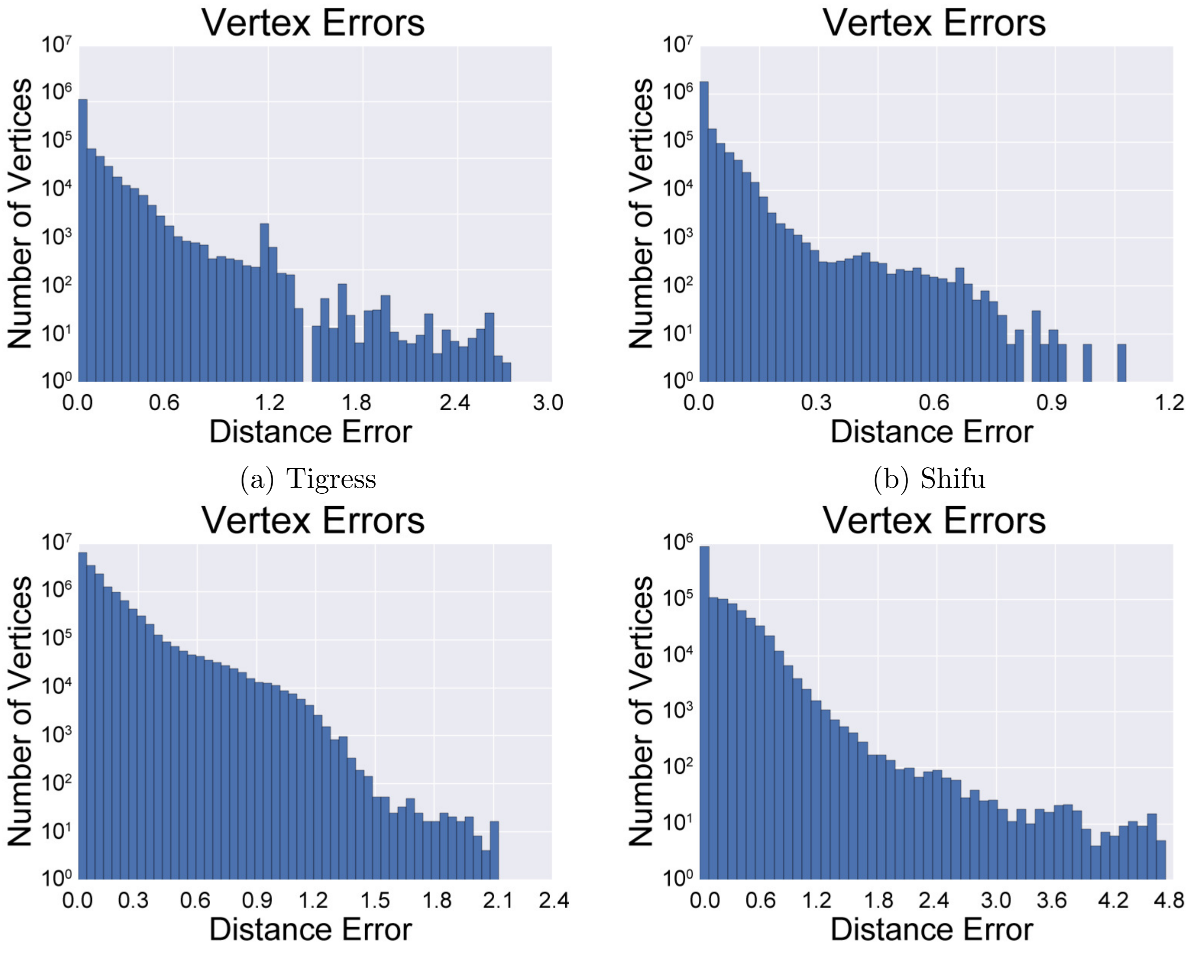
Figure 2.5: Side-by-side comparison of the ground truth mesh (left) and the approximation (right) for a frame of the dynamic motion of Tigress. The most noticeable difference is the shape of the stretched leg.[1]

Table 2.2: Mean and max approximation errors for each model tested on a walk cycle.

	Tigress	Shifu	Astrid	Po
Mean error	0.087 cm	0.016 cm	0.104 cm	0.143 cm
Max error	2.78 cm	1.10 cm	2.16 cm	4.80 cm

in the approximation.

We additionally evaluated our method on more dynamic motions that contain poses near the edge or beyond the range of motion that the approximation models were trained on. We tested animations of martial arts moves for both Shifu and Tigress. The mean and max errors are presented in Table 2.3, and the distribution of errors is shown in Figure 2.7. We found that the largest errors tend to occur in the legs during kicking motions. Specifically, the legs are stretched beyond what the approximator was trained on. Because the model did not learn from example poses with a large amount of stretch, it fails on this particular pose from the animation. However, because the error is still small relative to the scale of the character and because the leg is stretched for a brief, dynamic moment, the approximation error is barely noticeable when viewing the animation. Figure 2.5 shows the results of our approximation on a frame of animation with the stretched leg.



(a) Tigress

(b) Shifu

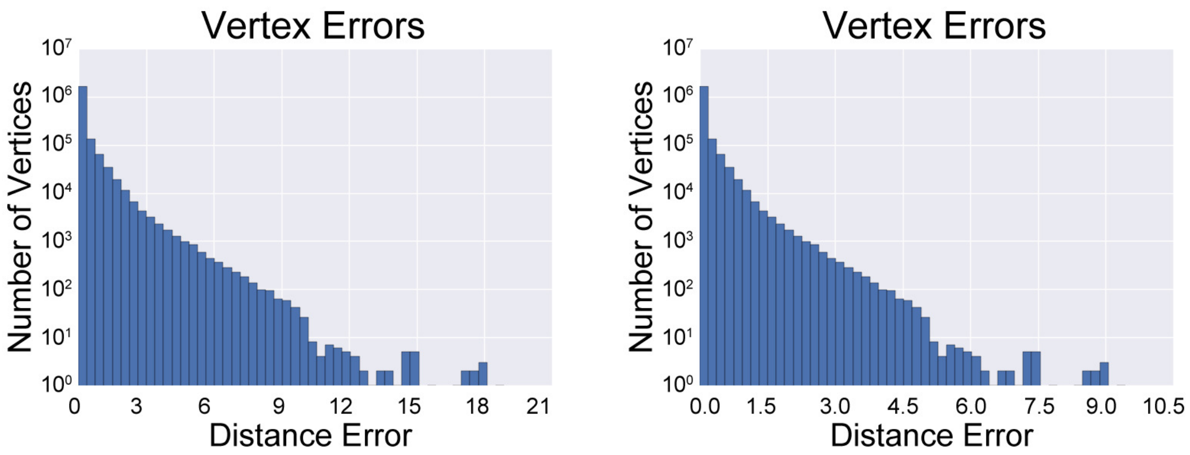
(c) Astrid

(d) Po

Figure 2.6: Log histogram plot of the distribution of per-vertex approximation errors for the walk cycle animations. All distances are measured in centimeters.

Table 2.3: Mean and max approximation errors for dynamic animations tested on the Tigress and Shifu rigs.

	Tigress	Shifu
Mean error	0.208 cm	0.041 cm
Max error	19.17 cm	8.14 cm



(a) Tigress

(b) Shifu

Figure 2.7: Log histogram plot of the distribution of per-vertex approximation errors for the dynamic animations for Tigress and Shifu. All distances are measured in centimeters.

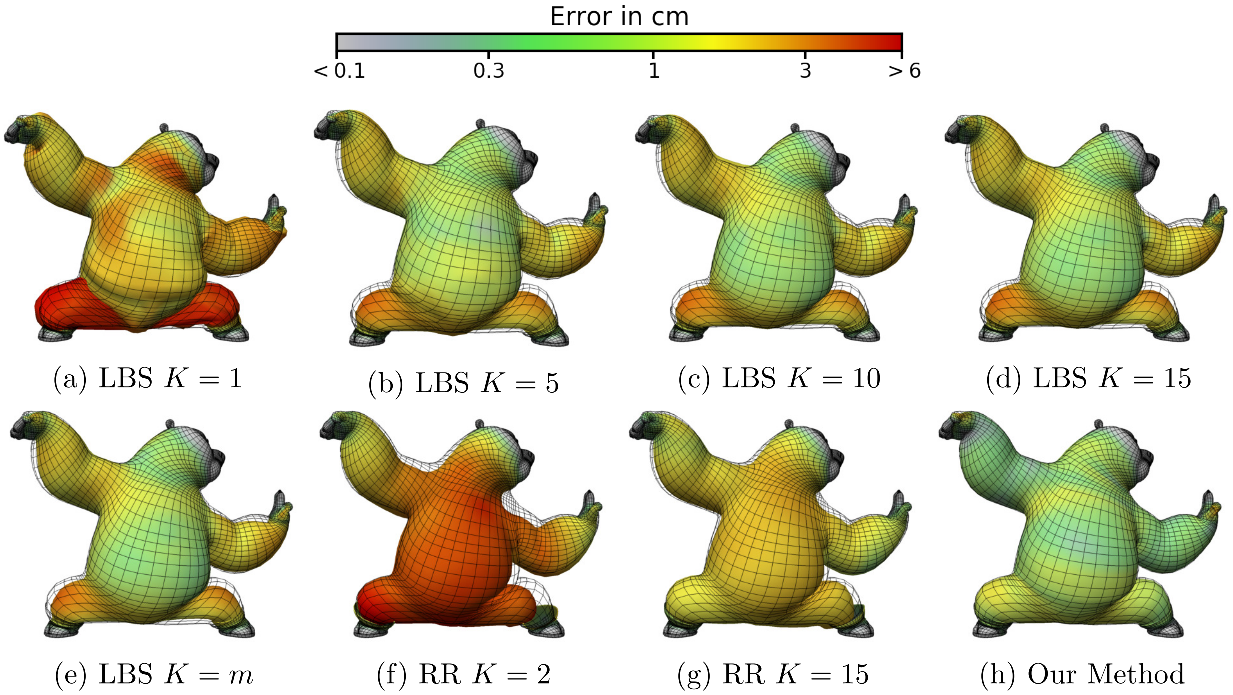
Comparison

We compare our approach with linear blend skinning (LBS) and the rotational regression (RR) method of Wang, Pulli, and Popović [150]. Like our algorithm, these two other methods can approximate the deformation function given any possible input pose. Thus, we can directly compare the accuracy of our approximation with these two methods.

For LBS, we estimate the bone weights for each vertex using example-based skinning decomposition [109]. We do not add any additional bones to the skeleton when we solve for the vertex weights. We experimented with many different values of the sparseness constraint K, which determines the total number of joints that can influence each vertex.

The rotational regression method described in Wang, Pulli, and Popović [150] approximates the deformation gradients of the triangles in a character's mesh. Their model uses linear regression to approximate deformation gradients from bone transformations. In their method, they only used at most two bones to approximate the gradients. However, we found that using only two bones was insufficient to approximate the deformations of our character rigs. We thus show results with the original method using $K = 2$ bones per triangle and with the method using $K = 15$ bones to approximate the gradients. To achieve the most accurate results, our implementation does not include the reduced formulation presented in their work.

In Figure 2.8, we show a comparison of deformations approximated with linear blend skinning, rotational regression, and our method. We show the deformations using LBS computed with different values of K ranging from $K = 1$ to $K = m$ where m is the total number of bones in the skeleton as well as RR using the original $K = 2$ as well as $K = 15$ bones as inputs. Visually, we can see that our method outperforms LBS and RR for this



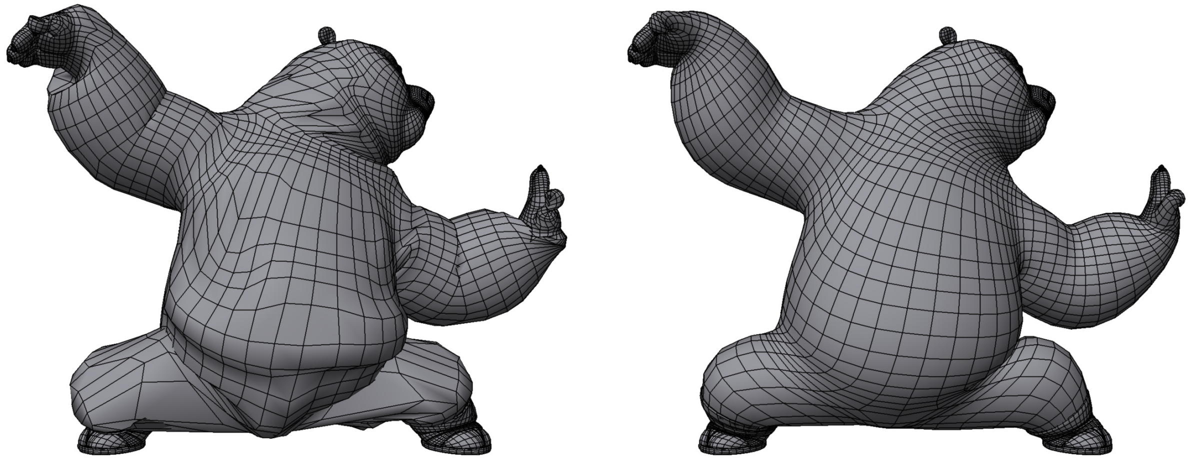
(a) LBS $K = 1$ (b) LBS $K = 5$ (c) LBS $K = 10$ (d) LBS $K = 15$

(e) LBS $K = m$ (f) RR $K = 2$ (g) RR $K = 15$ (h) Our Method

Figure 2.8: Our method compared with linear blend skinning using at most k bone weights per vertex and rotational regression using $K = 2$ and $K = 15$ input bones per deformation gradient. The deformation errors are denoted by the vertex color. Gray indicates no error while red indicates large error. The wire-frame of the ground truth mesh is rendered on top of each image to help visualize the errors.[1]

example pose. For LBS with $K > 1$, the deformations suffer from significant volume loss in the legs and the arms. In our method, this volume loss problem is not apparent, and the approximated deformation is closer to the original mesh than any of the meshes generated with LBS. In the case of $K = 1$ (Figure 2.8a), volume loss is not seen in the deformation because no transformations matrices are blended together. However, most of the errors occur from the vertices moving tangentially to the surface of the target deformation as shown in Figure 2.9. This type of error in the deformation would cause undesirable stretching and distortion of any texture applied to the mesh.

We further compared our method with LBS and RR for each walking and kungfu animation. In Table 2.4, we present the average approximation errors for all of the animations using our method compared with the other methods. Our algorithm learns from a separate training set described in Section 2.3 while LBS and RR are trained on the same animation on which the errors are measured, a situation which benefits LBS and RR in the comparison.



(a) LBS $K = 1$

(b) Ground Truth

Figure 2.9: Side by side comparison of LBS with $K = 1$ (left) and the target deformation (right). Although the shape of the meshes appear similar, the vertices in the LBS deformation are moved tangentially along the surface, which can cause undesirable effects when applying textures to the mesh.[1]

Table 2.4: Mean approximation errors, measured in cm, and enveloping errors (EE) using our method compared with linear blend skinning (LBS) with $K = 4$ and $K = m$ and rotational regression (RR) using the original choice of $K = 2$ as well as $K = 15$ input bones. The comparison is shown for all of the test animations with all of the rigs. EE is defined in Equation 2.9.

	Our Method		LBS ($K = 4$)		LBS ($K = m$)		RR ($K = 2$)		RR ($K = 15$)	
	Mean	EE	Mean	EE	Mean	EE	Mean	EE	Mean	EE
Tigress Walk	0.085	18.47	0.365	85.51	0.079	20.29	0.188	37.07	0.063	11.78
Tig. Kungfu	0.207	21.35	0.788	65.12	0.640	52.75	1.052	86.95	1.033	84.85
Shifu Walk	0.015	6.88	0.171	57.58	0.061	19.83	0.145	38.62	0.110	26.85
Shifu Kungfu	0.043	17.55	0.331	66.71	0.283	59.19	0.349	56.93	0.336	52.85
Po Walk	0.143	19.03	0.307	51.02	0.155	23.01	0.321	41.03	0.163	23.46
Astrid Walk	0.104	21.53	0.270	62.35	0.116	30.41	0.279	66.74	0.257	64.44

Following Wang, Pulli, and Popović [150], we also compute the envoloping error (EE)

$$EE = 100\sqrt{\frac{\sum_{i=1}^{N}\sum_{k=1}^{V}\left\|\mathbf{v}_k^i - \hat{\mathbf{v}}_k^i\right\|^2}{\sum_{i=1}^{N}\sum_{k=1}^{V}\left\|\mathbf{v}_k^i - \mathbf{c}\left(\mathbf{v}_k^0\right)\right\|^2}} \tag{2.9}$$

where $\hat{\mathbf{v}}_k^i$ is the approximated vertex position and

$$\mathbf{c}\left(\mathbf{v}_k^0\right) = \mathbf{X}_{b_k}^i \left(\mathbf{X}_{b_k}^0\right)^{-1}\left(\mathbf{v}_k^0 - \mathbf{t}_{b_k}^0\right) + \mathbf{t}_{b_k}^i \tag{2.10}$$

is the function that rigidly transforms the rest pose vertex position by the single best bone that explains its deformation. The envoloping error measures only local errors as opposed to global errors. We present the average error and the enveloping error for each animation using each approximation method in Table 2.4.

For each animation, our method is more accurate than LBS with both $K = 4$ and $K = m$. Furthermore, our method is more accurate than rotational regression in each tested animation with the exception of the walk cycle for Tigress when $K = 15$. The mesh deformations on the characters we tested have regions where the vertex positions depend on more than two bones such as in the hands and in the torso, and our results show that rotational regression performs better when more bones are provided as input. In Astrid's mesh, there are many small, unconnected meshes such as the spikes in her skirt. No vertex on these meshes can be accurately placed using rigid skinning with a single bone, which the rotational regression method relies on. Thus, the error from the rotational regression approximation is clearly visible as seen in Figure 2.10.

We did not compare our deformation approximation with skinning decomposition methods that add bones to a character rig. Skinning decomposition methods solve the problem of optimally fitting bones to a mesh when an existing animation is provided. The advantages of these types of methods are that they can have arbitrarily high accuracy when reproducing the example animations [66] and that they can play back the example animations at a fast rate. Our method, in contrast, solves the problem of approximating mesh deformations given a character rig with an existing skeleton but without any example animations. Because of the difference in the type of problem that our method solves and the type that skinning decomposition methods solve, we do not compare our approach with these algorithms.

Model Speed

As seen in Table 2.5, the run-time of our approximation compared to the run-time of the original rig evaluation demonstrates the computational savings that can be achieved with our method. To train our models, we used Theano [138] in a Python environment. Once the models were trained, we evaluated them in our own multi-threaded C++ implementation. During training and testing, we use only the CPU to evaluate the networks. We found that running the models on the GPU was slower than the CPU. This slower performance is caused by the models having large inputs and outputs compared to the size of the hidden layers in

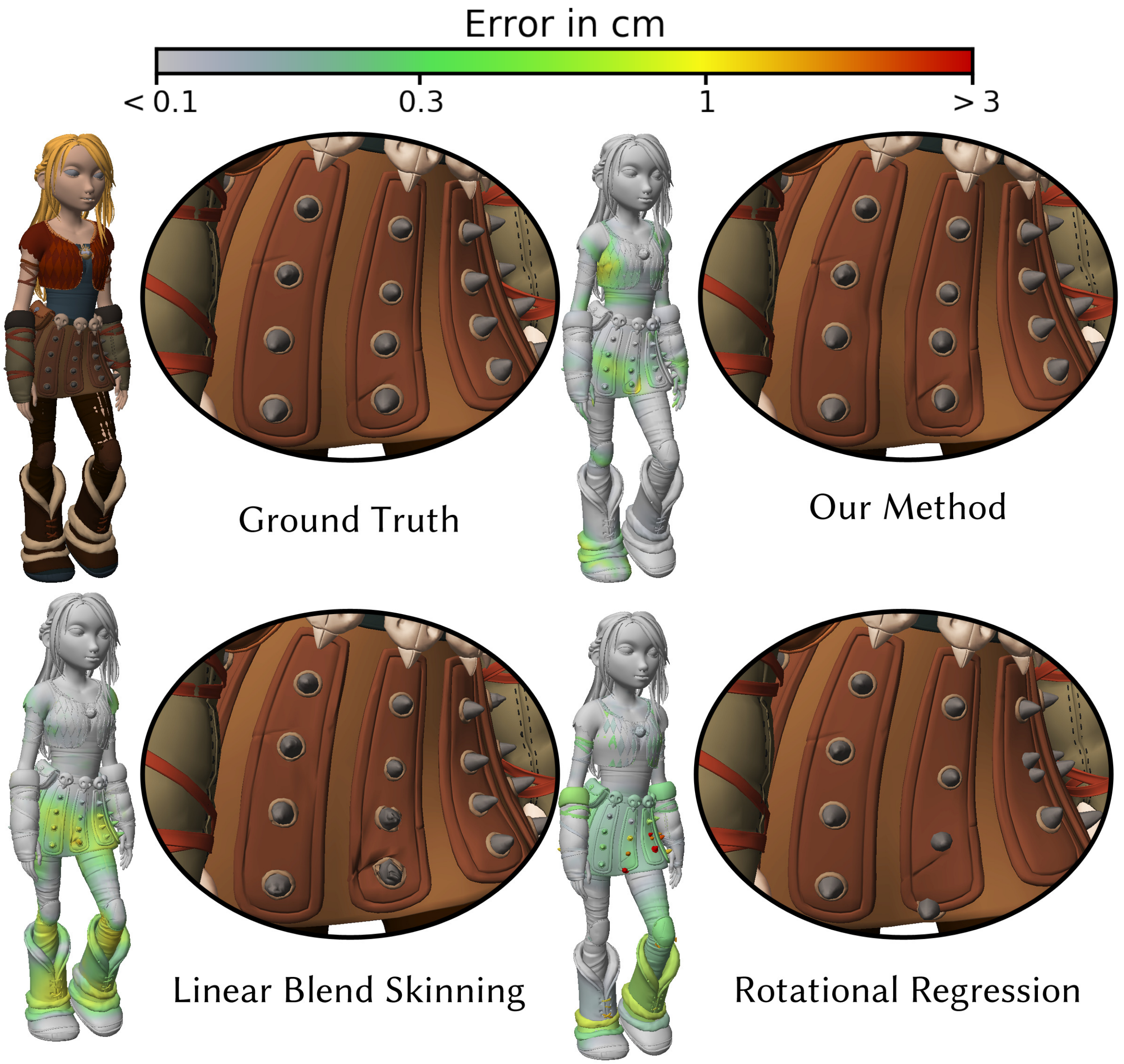


Figure 2.10: Close-up of Astrid's skirt for the original mesh compared with our method, LBS, and RR. Our method more accurately approximated the vertices on the skirt. LBS produced visible errors in the middle of the skirt because those vertices cannot be placed accuratly as a linear combination of the bones. RR produced visible errors because the spikes on the skirt are separate meshes and cannot be placed accurately because RR relies on rigid skinning to fix the location of at least one vertex in each mesh.[1]

the network. Thus, most of the time spent evaluating the network on the GPU was spent transferring data.

The training time for each approximation model took approximately 2-3 hours with the majority of the time spent evaluating the original character rig to generate the training data. Once trained, we evaluated the speed of the model by measuring the evaluation time through the model for a single input pose. Multiple input poses can be passed into the model for a single evaluation, which would utilize matrix-matrix multiplications through the neural network as opposed to matrix-vector. Evaluating the model with multiple input poses would have a faster run-time per input compared with evaluation on poses one at a time. This speed increase comes from matrix-matrix multiplication running faster than separate matrix-vector multiplications for each input when using highly optimized linear algebra packages.

Despite the performance gains from evaluating multiple input poses simultaneously, we timed the approximation models evaluating inputs one at a time to demonstrate the applicability of our method for interactive applications. In Table 2.5, we compare the run-time of our method with Libee [153], a highly optimized, multi-threaded rig evaluation engine, on four different character rigs. All of the character rigs have been optimized to evaluate as fast as possible in Libee. Because our method approximates only the deformation system of the character, the times we report from Libee are measured by the difference in time between when the rig evaluation starts and finished all computations for the deformation system. We present times for running the rig evaluation in both parallel and single-threaded implementations. We ran our experiments on a machine with an Intel Xeon Processor E5-2690 v2 running at 3.0GHz with 10 cores and 92GB of RAM. In both cases, our approximation method outperforms Libee by up to a factor of 10. The largest performance gains are observed when comparing the parallel implementations.

In addition to Libee, we compare our method with the weighted variant [76] of pose space deformation [94]. Like our method, PSD can be used to add a corrective offset to overcome the limitations of linear skinning. To compare the timing of our method with WPSD, we replace the neural networks in each nonlinear deformer and use WPSD to predict the same vertex offsets given the same input bones from the skeleton. We test WPSD using 10, 50, and 100 example poses from the test animations. The timing results using WPSD as well as the timing results evaluating the linear only skinning are shown in Table 2.5. Although the timing of our method is comparable to WPSD using 100 example poses, we would like to point out that the speed of WPSD depends on the number of example poses.

Our method provides a fast approximation to the deformation system, which allows a high-quality character rig to be evaluated in real-time on a low-end system or even a mobile device. To demonstrate our approach, we implemented the approximation on an iPad and evaluated both Astrid's and Po's character rigs on the device. Table 2.5 shows the timed results on the iPad. For both rigs, our approximation runs faster on the mobile device compared with the full evaluation of the deformation system running in parallel on a high-end machine using Libee.

Table 2.5: Timing comparison in milliseconds for the deformation systems of the characters evaluated with Libee and our approximation using both a parallel implementation and a single-threaded implementation as well as the timing for the approximation run on a mobile device for the Astrid and Po character rigs. We also provide timing for WPSD using 100, 50, and 10 example poses and timing for the linear only skinning. The timings for the iPad, WPSD, and linear skinning are all evaluated on a parallel implementation on the CPU.

	Tigress	Shifu	Astrid	Po
Libee serial	65.2 ms	43.1 ms	142.5 ms	89.6 ms
Our approx. serial	10.6 ms	10.2 ms	62.0 ms	9.8 ms
Libee parallel	20.6 ms	8.7 ms	32.8 ms	28.2 ms
Our approx. parallel	2.7 ms	1.5 ms	7.7 ms	2.2 ms
iPad	N/A	N/A	28.6 ms	7.7 ms
WPSD 100	1.5 ms	1.5 ms	9.0 ms	1.4 ms
WPSD 50	1.0 ms	1.0 ms	7.6 ms	0.9 ms
WPSD 10	0.7 ms	0.6 ms	6.7 ms	0.6 ms
Linear only	0.5 ms	0.4 ms	3.2 ms	0.4 ms

2.5 Discussion

We have presented a method that can accurately approximate mesh deformations for film-quality character rigs in real-time. Our method relies on defining the deformations in a local coordinate system to reduce the complexity of the nonlinear deformation function that we approximate. We use deep learning methods to learn these deformations and are able run the approximation in real-time.

Limitations

Our method assumes that mesh deformations are a function only of the skeleton. However, character rigs for feature films may have additional deformations that rely on rig parameters that are not associated with bones. Currently, our approach is unable to learn these types of deformations, but the algorithm we have described can be modified if the additional rig parameters influencing these deformations are given as inputs to the approximation. Additionally, because our method only computes the approximation per pose, it cannot handle dynamics or non-deterministic behavior. Approximating these types of behaviors could make for an interesting extension to our method.

Deformations of a character's face is an example of deformations that rely both on bone transformations and additional rig parameters. In film-quality rigs, the face is animated with high-level artistic controls. Unlike the deformations on a character's body, the face deformations rely mostly on the high-level controls rather than the underlying skeleton.

This difference would create a significant problem when directly applying our method to approximate the face. If our method were used to approximate the face deformation, the vertices on the face would be assigned to a small set of head bones that do not explain most of the deformation of the mesh. Furthermore each vertex might be affected by large number of rig parameters, which leads to a high-dimensional input for each vertex. Because the input dimension would be large and most of the facial deformation would need to be learned, training an approximator with our deep learning method would be challenging.

Linear Component Alternatives

From Figure 2.8, we can see that the approximation grows closer to the original mesh as K increases. Because LBS can be computed quickly, our method could compute the linear deformation component from Equation 2.1 using some $K > 1$, and this would reduce the residual error that the nonlinear function approximators need to learn. However, we found that in practice using a larger K does not have a significant visual impact on the results.

Delta Mush [104] is a type of deformation that aims to preserve the volume of a deformed mesh. Additionally, Delta Mush can easily be applied to a character rig without requiring any fine-tuning. Although this method is not a linear deformation, by preserving volume, Delta Mush could bring the deformed mesh closer to the target deformation. As a result, the remaining nonlinear deformation could be easier to learn and could be approximated with smaller and faster neural networks. Using Delta Mush as an alternative for the linear component of our method could be an interesting area of exploration.

Nonlinear Component Alternatives

Although pose space deformation can approximate mesh deformations faster than our method if sufficiently few example poses are provided, the quality of the approximation depends heavily on the selected poses. We found that using poses generated from our training set described in Section 2.3 as example poses for PSD does not result in an accurate deformation approximator. Better results could be achieved by manually selecting example poses to ensure a more accurate approximation. Our method, in contrast, is able to learn an accurate approximation from this randomly generated dataset.

Potential Applications

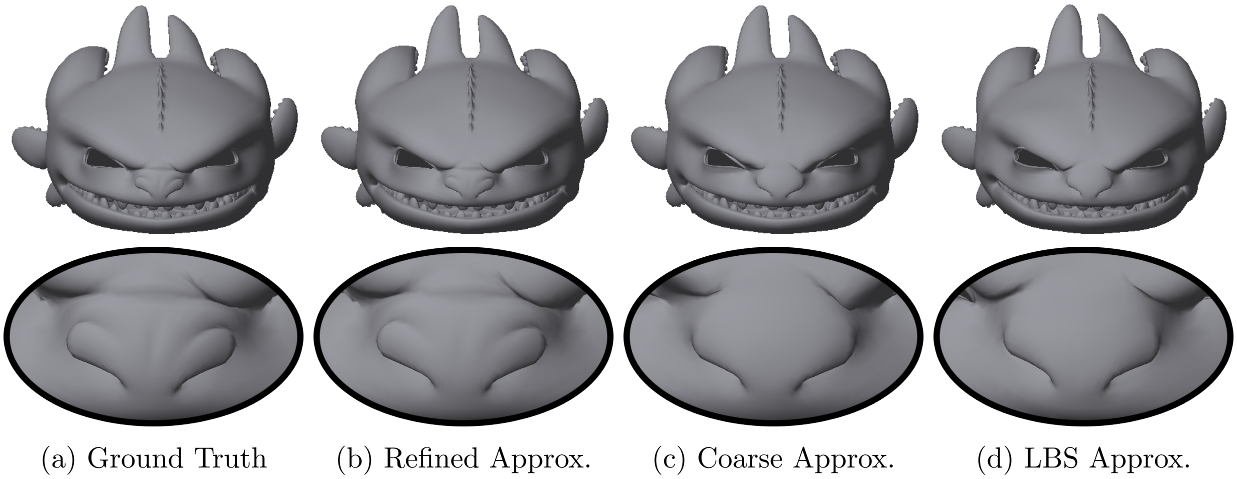
Our method can be combined with other approaches that can provide a character skeleton in real-time to create interesting real-time experiences. For example, motion capture recordings can be used to drive a character's skeleton, and our method can use the skeleton to compute the final deformed mesh of a character. Furthermore, prior research has explored animation synthesis techniques. Motion graphs [74] can be used to synthesize controllable animation at interactive rates. The animation generated by this approach is a sequence of skeletons, which our method can use to compute a character's mesh deformation. Other synthesis methods use generative models such as Gaussian processes [43, 148, 92] or deep learning models [53, 52]. All of these motion synthesis methods output bone positions and rotations for a character, which form the inputs to our method. Because character skeletons can be generated using many different techniques, our method can readily be applied to the outputs of these synthesis algorithms to animate a film-quality rig in real-time.

Chapter 3

Fast and Deep Facial Deformations

3.1 Introduction

Character facial rigs for video games and other real-time applications are often controlled by sets of bones or blendshapes. Although these rigging methods can be computed quickly, they generally sacrifice fine-scale details for speed. Expressing nuanced deformations with these real-time rigs is challenging and often requires additional computational layers added to the underlying rig. Some such additions for increasing the level of detail in the mesh



(a) Ground Truth (b) Refined Approx. (c) Coarse Approx. (d) LBS Approx.

Figure 3.1: Side-by-side comparison of facial mesh deformations using our coarse and refined approximations as well as an approximation generated by linear blend skinning. The most noticeable difference, shown on the second row, is observed around the nasal region of the mesh.[1]

[1] These images are Property of DreamWorks Animation L.L.C., used with permission.

deformation include pose space deformations [94] and wrinkle maps. However, despite these improvements, the level of detail in film-quality facial rigs is noticeably better when compared with real-time rigs. Because these rigs are not constrained by real-time requirements, artists can utilize more computationally expensive deformations to achieve sophisticated and detailed results.

Facial rigs for film require a significant amount of computation to create realistic and expressive mesh deformations. When evaluated on a high-end machine, these facial rigs might run at a rate of only 10-30 FPS, and would run even slower on consumer-level devices. Furthermore, animation studios typically develop in-house rigging software on which their characters are developed. These rigs are limited to their custom animation engines, and porting a character for use outside of the in-house software can be challenging and time-consuming. Thus, due to computational limits and portability, film-quality characters are rarely used outside of the film for which they are designed.

Recently, Bailey et al. [5] proposed a method to approximate body deformations of film-quality character rigs. However, their method relies on an underlying skeleton to approximate the deformations. Unlike body rigs, facial rigs do not have an extensive skeletal structure that can be utilized for the deformation approximation. To overcome this limitation, we propose a novel method for approximating deformations of facial meshes. Our approximation accurately computes deformations of the facial mesh, including skin, mouth interior, teeth, and other structures. Our approach uses convolutional neural networks (CNNs) to take advantage of the types of deformations found in facial meshes. The method computes the deformation in three separate parts: a coarse approximation, a refined approximation, and an approximation for rigid components of the mesh. Our method preserves high-frequency detail in the mesh (Figure 3.1) while running up to 17x faster than the production rigs that we tested.

Furthermore, we implement our approximation model in TensorFlow [105], an open-source machine learning library, which allows the facial rig to transcend the proprietary limitations of the original rigging software used to author the character and allows the model to be deployed on a wider variety of devices and applications. In addition, the rig approximation can easily be evaluated on both the CPU and the GPU without any additional effort.

3.2 Related Work

Facial deformation systems for animated characters vary widely in their methods and complexity. Often, facial models combine multiple deformation methods to achieve their final result. One of the simplest and fastest ways to compute deformation is from an underlying skeleton. Skeleton subspace deformation (SSD) [102], also called linear blend skinning (LBS), is popular due to its simplicity and speed. Due to the well-known shortcomings of SSD, like the candy-wrapper effect, improvements have been investigated such as multi-weight enveloping [151] and dual-quaternion skinning [68] which improve the quality without noticeably impacting the evaluation speed. While this class of methods is often used as the base deformation system for a character's body, it is often combined with other methods to rig a character's face. A more common approach for a facial deformation system is blendshapes [117, 116, 72] which linearly combine a set of artist-created facial expressions. This method is also fast to evaluate but is too limiting by itself for film-quality character rigs that could require hundreds of blendshapes to be keyed every frame in an animation. Another approach to construct a facial model is through physically-based deformation for better realism and ease of generating realistic poses [132, 29, 58]. In complex facial models, all of these techniques and others may be combined which generally results in a high cost and low evaluation speed.

For real-time applications, it is necessary to construct a facial deformation model which preserves detail without incurring too great a computational cost. One approach [11] utilizes pose-space deformation [94] in a hybrid approach which computes the base deformation using SSD and learns a model to compute high-fidelity, nonlinear details, like wrinkles, which are applied on top of the base. For efficient computation of physically-based deformation, Hahn et al. [45] improves on rig-space physics [47] for real-time results on production-quality character rigs. These approaches are sufficient in achieving high performance for the systems they are built upon, but we seek to find an efficient representation for an existing high-quality rig, whose deformation model may be slow to compute on lower-powered hardware without needing to optimize the complex character rig.

There exist many different approaches to approximate an existing deformation model given a set of example poses. A goal of most of these approaches is to construct a more computationally efficient representation of the deformation function. One of the skinning decomposition methods [85] finds the bone transformations and skin weights for a skeleton subspace deformation model given a set of example poses. Similarly, Le and Deng [84] also finds a SSD representation of the deformation, but organized in a skeletal hierarchy for easier animation afterward. Feng, Kim, and Yu [33] learns a skinned mesh via SSD from example data in order to animate with control points. Because a bone-based deformation system is not the best way to represent facial deformations, these methods alone are not suitable for our purposes. Sphere-Meshes [139] decompose a mesh animation into a set of animated spheres, which can be keyframed afterwards for animation. This approach is also unsuitable for high-quality character animation due to the difficulty of representing fine details. Specifically targeted at facial animation, Li, Weise, and Pauly [97] and Neumann et al. [115] learn new parametric rig models like blendshapes from example poses. Garrido et al. [37] create facial rigs based on statistical models such that the appearance of the rig closely matches the appearance of a recorded actor. All of these methods learn a completely new rig representation with different controls than those present in the original rig. Our goal is to approximate an existing facial rig and maintain the same controls so an artist would not have to re-learn the control parameters.

Past research that attempts to approximate an existing rig function often assumes an underlying blendshape model [129] or an underlying skeletal structure, while our method does not make such strong assumptions about the facial rig. EigenSkin [75] efficiently computes high-fidelity nonlinear deformation on GPU via an error-optimal pose-dependent displacement basis constructed from example meshes. This method assumes an underlying SSD representation of a given rig and uses it in its computation. Mohr and Gleicher [109] learns an augmented SSD skinning model with additional joints from an existing SSD rig. The work of Bailey et al. [5] assumes an underlying skeleton and uses the skeletal deformation as a base on which a fine-detail nonlinear displacement is overlaid. In this chapter, we learn a deformation model without the assumption of a skeletal system, which is appropriate for complex facial rigs.

In order to support inverse kinematics (IK) for a facial rig in real-time, an efficient and accurate inversion of the rig function is necessary to compute character poses given a set of constraints. Due to the complexity of facial rigs, traditional solutions for the IK problem, which has been well-studied [21, 124, 113, 157, 40], are not easily applicable to film-quality facial models due to the requirement of a differentiable rig function. There is existing research in computing blendshape parameters from landmarks [162, 11, 93], but our work seeks to allow for the inversion of an arbitrary black-box rig function. Prior work has explored solutions to this problem. Xiao et al. [159] utilizes an iterative optimization approach; however, their method is not entirely rig-agnostic as it is designed to optimize the inversion of a pose-space deformation rig. Holden, Komura, and Saito [52] successfully inverts a black-box rig function using two nonlinear methods: Gaussian process regression and feed-forward neural networks. In contrast, our approach uses deep-learning methods to approximate the original rig function. Due to the neural network, the gradient of the rig function can be estimated through the rig approximation, which can then be used to estimate the inverse rig function.

Recently, deep convolutional methods have been developed for data-driven mesh regression problems. These methods utilize the power and flexibility of deep neural networks for applications ranging from facial reconstruction [34] and facial animation [78] to cloth simulation [61]. One way to apply CNNs to meshes is by defining mesh convolution operations. Litany et al. [99] introduces graph convolutional autoencoders, while Ranjan et al. [122] uses a similar idea to generate 3D faces. MeshCNN [48] defines specialized convolution and pooling operations on triangle meshes. Because our applications are centered around efficiency, using these mesh convolutions would be too computationally expensive. Traditional CNNs operate on 2D images and feature maps. In order to reconstruct 3D deformations using these models, a mapping must be created between the feature map space and vertex positions. Masci et al. [106] and Boscaini et al. [14] apply convolutions to a mesh by parameterizing the mesh around a small local area. Sinha, Bai, and Ramani [134] applies CNNs by projecting a mesh onto a spherical domain then “cutting up” the projection. Other works [77, 34, 61] use texture (UV) maps to map the vertex positions to 2D space. In this way, the networks learn to predict 2-dimensional feature maps but they represent 3-dimensional coordinates. Convolutional neural networks have seen success in prior work due to the spatial coherence of vertex positions being preserved in transformed space. Previous work has generated UV maps from perspective projections [34] or scans [61, 77]. Because our work assumes a complete character rig, we use a UV map created by an artist to compute vertex positions from

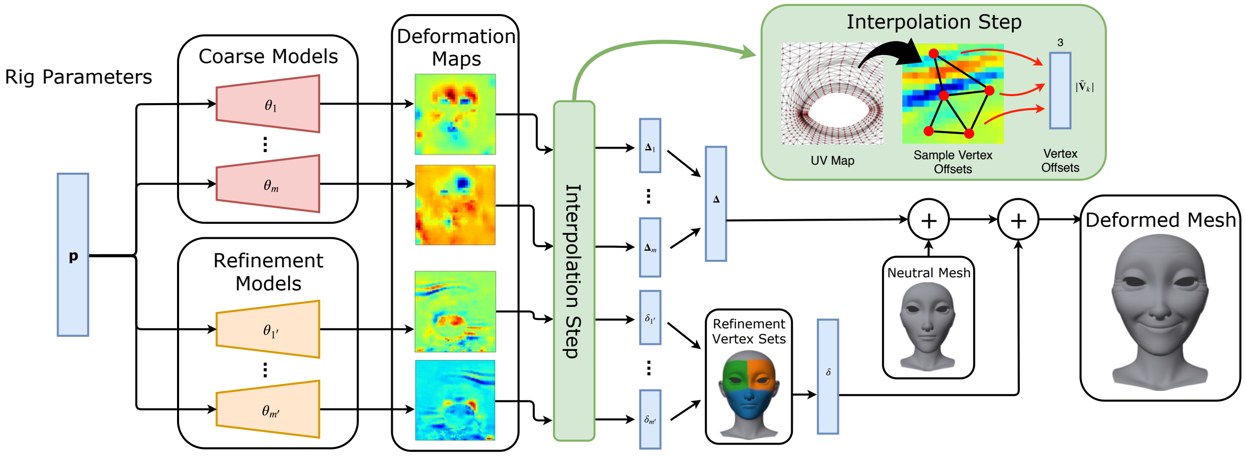


Figure 3.2: Diagram of the approximation model. Rig parameters are used as input to convolutional networks which generate a deformation map for each mesh segment. Vertex offsets are extracted by bilinear interpolation of the deformation map at each vertex position in texture coordinate space. These offsets are added to the neutral pose to reach the desired deformation. For the refinement model, only a subset of the total active vertices is used.[1]

a 2D feature space.

3.3 Facial Approximation

Given a character's facial rig with a polygonal mesh, let $\mathbf{V}$ denote the set of vertex coordinates in the mesh with $|\mathbf{V}| = n$ vertices. Let $\mathbf{p}$ represent the rig parameters of the characters, and let $\mathbf{V} = \mathbf{r}(\mathbf{p})$ be the rig function that maps the parameters to a deformed mesh. Our method focuses on approximating this rig function $\mathbf{r}(\mathbf{p})$.

Our approach utilizes artist-created texture coordinates $\mathbf{U} \in \mathbb{R}^{n\times 2}$ of the facial mesh. The approximation relies on convolutional neural networks, which generate deformation maps given input rig parameters. The deformation maps are sampled at texture coordinates to approximate vertex positions in the mesh. Many parameters for a facial rig deform local regions of the mesh, and the rig parameters can be viewed as local operations on the mesh. By design, a CNN performs local computations on a feature map. Assuming that the local information in a mesh is preserved in the texture coordinates, a CNN is ideal for approximating the rig function.

Our model is divided into two stages: a coarse approximation and a refined approximation (Figure 3.2). The coarse approximation operates on the entire mesh. To ensure that the model executes quickly, the coarse approximation is comprised of multiple CNNs that output low resolution deformation maps. As a result, high-frequency details in the deformation are lost. To handle this detail loss, we propose a refined approximation composed of additional

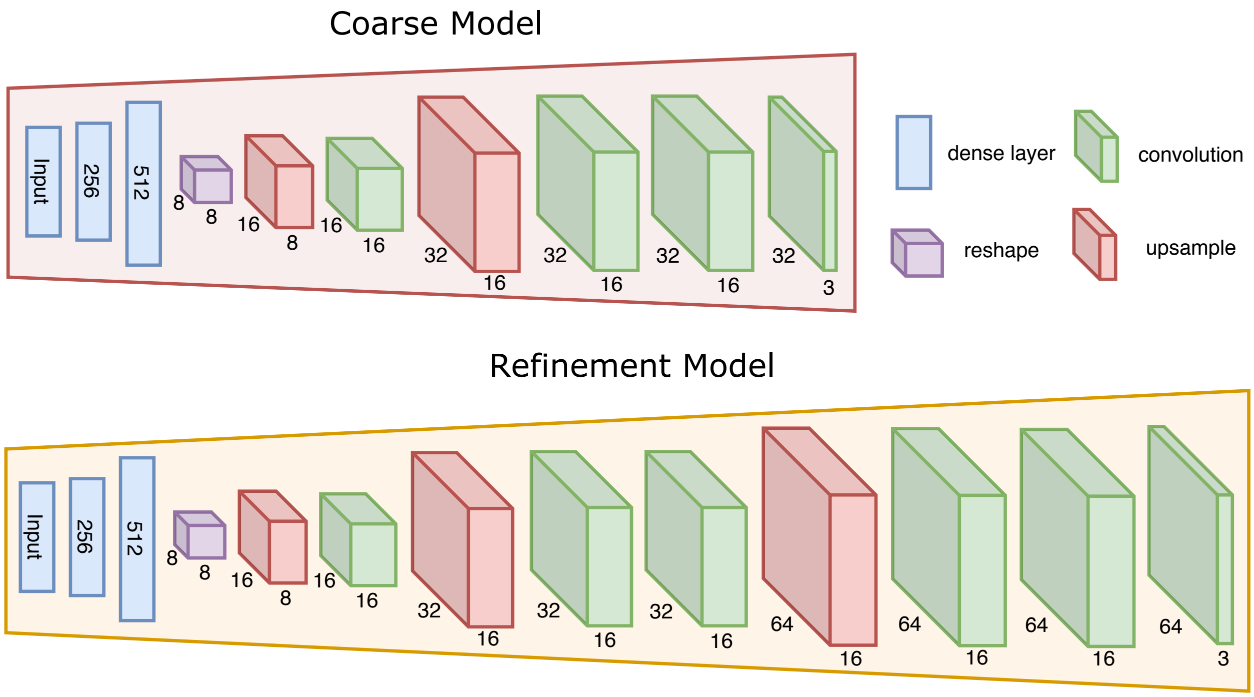


Figure 3.3: Detail of course and refine models. All convolutions use 3x3 kernels except for the last layer, which uses a 1x1 kernel. All layers but the last use the leaky ReLU activation function, and no activation function is applied to the last layer. All non-dense layers are square in the image plane. Upsampling is achieved through nearest-neighbor interpolation.

CNNs that output higher resolution deformation maps. These models focus only on vertex-dense regions of the mesh to approximate these high-frequency deformations. To further improve the model's efficiency, we identify segments of the mesh that only undergo rigid rotations and translations within the rig function. These segments are approximated with a faster rigid approximation instead of the more complex CNN approximation.

Coarse Model

We assume that the facial mesh is divided into multiple segments, which is common for artist-created facial rigs. Each vertex is assigned to a single mesh segment. Let m indicate the number of segments in the mesh, and let $\mathbf{V}_k$ and $\mathbf{U}_k$ denote the set of vertex positions and texture coordinates in mesh segment k. The coarse approximation also works with facial rigs that are not segmented, and in this case, we would set $m = 1$, and the full mesh is contained in the single segment.

The coarse model computes the deformed mesh by first generating a deformation map for each mesh segment in the facial rig and then computing vertex positions through the maps. The function $\mathcal{I}_k = f(\mathbf{p}; \boldsymbol{\theta}_k)$ computes a deformation map for mesh segment k given rig parameters $\mathbf{p}$. The function f is a neural network consisting of several dense layers and convolutional layers and is parameterized by $\boldsymbol{\theta}_k$ (Figure 3.3). Vertex offsets $\boldsymbol{\Delta}_k$ are computed by sampling deformation map $\mathcal{I}_k$ at texture coordinates $\mathbf{U}_k$. We represent this sampling step as $\boldsymbol{\Delta}_k = \mathbf{g}(\mathcal{I}_k; \mathbf{U}_k)$, which outputs the vertex offsets. Because each vertex is assigned to a single mesh segment, the vertex offsets for the full mesh are obtained by concatenating the offsets for each segment such that $\boldsymbol{\Delta} = \cup_{k \in \{1,\ldots,m\}} \boldsymbol{\Delta}_k$. The approximation computes the final vertex positions for the mesh by adding the offsets to the mesh's neutral pose.

Given the approximation model, we next define a loss function to find the optimal model parameters $\boldsymbol{\theta}_k$. We propose a loss function that both penalizes inaccuracies in approximated vertex positions as well as inaccuracies in face normals on the mesh. Given a target mesh $\mathbf{V}$, and the approximated vertex offsets $\boldsymbol{\Delta}$, the loss function is defined as

$$\mathcal{L}(\mathbf{V}, \boldsymbol{\Delta}) = \frac{1}{n} \sum_{i=1}^{n} \left\| \mathbf{v}_i - (\mathbf{V}^0 + \boldsymbol{\Delta})_i \right\|_1 + \alpha_n \frac{1}{f} \sum_{i=1}^{f} \left\| \mathbf{n}_i - \widetilde{\mathbf{n}}_i \right\|_1 \tag{3.1}$$

where α_n is a scaling factor hand-tuned by the user. Experimentally, we found that $\alpha_n = 5$ works well. In the loss, $\mathbf{n}_i$ is the normal of face i in the mesh $\mathbf{V}$, and $\widetilde{\mathbf{n}}_i$ is the normal of face i in the approximated mesh with vertex positions $\mathbf{V}^0 + \boldsymbol{\Delta}$ and a total of f faces in the mesh topology. We use the L1 loss instead of the L2 loss because it produces sharper features. We learn the mapping from rig parameters to vertex offsets end-to-end without supervision on the intermediary deformation maps. Furthermore, we do not optimize the texture coordinates and rely instead on the artist-created coordinates.

Because the approximation model works on separate mesh segments, the model could produce discontinuities across mesh segment boundaries and seams. To minimize this potential problem, the error function strongly penalizes inaccurate face normals, which encourages smooth results along mesh segment boundaries. Penalizing normal errors also suppresses low-amplitude, high-frequency errors that are visually disturbing.

To help with model training, each network is provided only a subset of the rig parameters. The subset contains all of the rig parameters that can deform any vertex within the mesh segment that the model is approximating. All other rig parameters are excluded from the network's inputs. As a result, the network does not need to learn which parameters to ignore and will avoid being negatively affected by the noise provided by inputs that have no influence on the outputs.

Refinement Model

In the coarse approximation model, the resolutions of the deformation maps $\mathcal{I}_k$ are intentionally kept small to reduce the computational complexity. However, in texture coordinate space, vertices in dense regions of the mesh could be placed less than a pixel apart in the small deformation maps. If these vertices undergo a high-frequency deformation, such as skin wrinkle, then the approximation model will be unable to recreate this deformation accurately. The bottleneck in this case is the resolution of the map output by the CNN. To overcome this limitation, we propose applying refinement models that focus exclusively on these vertex-dense regions of the mesh.

First, we identify sets of vertices that correspond with regions of large error in the approximation. We discuss vertex selection for the refinement model in Section 3.3. Each set is then defined as a new mesh segment. The texture coordinates for each vertex in the new mesh segments are scaled to fill the full resolution of the refinement deformation maps. As in the coarse approximation, no vertex is assigned to multiple mesh segments. Additionally, not every vertex is assigned to a mesh segment for the refinement stage. Only vertices in regions of the mesh with a high approximation error are divided into mesh segments.

Let m' indicate the number of mesh segments in the refinement stage and let $\mathbf{U}'_{k'}$ be the new texture coordinates for segment k'. Similar to the notation for the coarse model, the refinement model for segment k' can be expressed as $\boldsymbol{\delta}_{k'} = \mathbf{g}(\mathbf{f}(\mathbf{p}; \boldsymbol{\theta}^r_{k'}); \mathbf{U}'_{k'})$ where $\boldsymbol{\theta}^r_{k'}$ are the parameters for the refinement model. The output $\boldsymbol{\delta}_{k'}$ approximates the residual between the vertex positions in the mesh and the output of the coarse model within mesh segment k'. The refinement approximation for vertices not contained in any mesh segment in this stage is set to zero, and we denote this set as $\boldsymbol{\delta}_{m'+1} = \mathbf{0}$. Let $\boldsymbol{\delta}$ represent the combined set of outputs $\boldsymbol{\delta}_{k'}$. The refinement models are trained using the same loss as the coarse model from Equation 3.1 where the loss is evaluated as $\mathcal{L}(\mathbf{V}, \boldsymbol{\Delta} + \boldsymbol{\delta})$.

In our implementation, the refinement models produce deformation maps with a higher resolution than those produced by the coarse models. Alternatively, the entire approximation could be computed by only applying these higher resolution refinement models across the entire mesh and foregoing use of a coarse approximation. However, applying the refinement model across the entire mesh would have a much higher computational cost both because of a global resolution increase and because the refinement model uses a deeper network.

Refinement Boundary Selection

To identify the vertex sets used for refinement, we estimate for each vertex the minimum approximation error given the coarse deformation map resolution and texture coordinates of each mesh segment. Next, we perform clustering on the texture coordinates with each vertex weighted by its estimated approximation error. The vertices near each cluster become the mesh segments for the refinement models while vertices far from cluster centroids are omitted from the refinement step.

We estimate the minimum approximation error by first mapping vertex positions to a deformation map through the texture coordinates and then sampling the deformation map at the coordinates to generate vertex positions. The map is computed through polyharmonic interpolation with a linear kernel by interpolating values at pixel coordinates from the texture coordinates. Vertex positions are computed from the deformation maps through bilinear interpolation. Let $\mathbf{v}_i$ be the original vertex position, and let $\widetilde{\mathbf{v}}_i$ be the sampled vertex position from the deformation map. We estimate the approximation error over a set of n

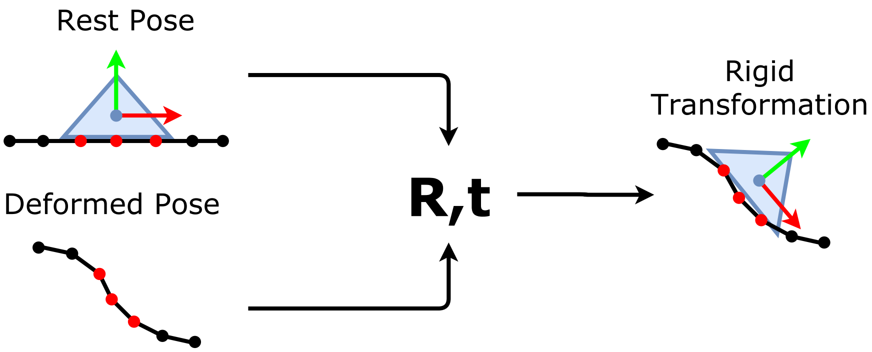


Figure 3.4: Illustration of the rigid components. The blue triangle represents a rigid mesh segment identified by Equation 3.3. The black line represents a nonlinearly deformed mesh segment, and the dots on the line represent vertices on the surface. The red dots represent the set of vertices identified by Equation 3.4 that best match the rigid transformation of the blue triangle across a large set of examples. Given the rest pose and the positions of the vertices on the nonlinear segment in a deformed pose, the transformation $\mathbf{R}, \mathbf{t}$ is computed from the red vertices. The transformation is then applied to the blue triangle to compute its position in the deformed pose.

samples $\mathcal{V}_i = \{\mathbf{v}_i^1, \mathbf{v}_i^2, ..., \mathbf{v}_i^n)$ as

$$e_i = \frac{1}{n} \sum_{j=1}^{n} \left\| \mathbf{v}_i^j - \widetilde{\mathbf{v}}_i^j \right\|_2^2 . \tag{3.2}$$

We then run k-means clustering on the texture coordinates with each vertex weighted by its corresponding approximation error e_i. The number of clusters is determined by the elbow method. Each vertex is assigned to the nearest cluster centroid up to a user-specified distance. In our experiments, we assigned vertices within a square with a length of 1/4 of the width of the original texture coordinate space and centered on the cluster means. This approach worked well for the characters we tested. As with the coarse approximation, we compute the set of rig parameters that can deform any vertex contained in these new mesh segments and provide each refinement model with only those input parameters.

Rigid Components

In character faces there could be sections of the mesh that move rigidly, such as teeth. In the case of characters that we tested, each tooth was modeled as a separate segment. Because the deformation of each tooth in the rig could be expressed as a rotation and translation, approximating the linear transformation with a CNN model for each tooth would produce unnecessary computation. Instead, we estimate the rigid movement by computing a linear transformation from nearby vertices in the approximated mesh as illustrated in Figure 3.4.

Each rigid mesh segment is assigned to a subset of vertices approximated by the CNN models. The motions of these rigid segments are then estimated by solving for the rigid transformation that best explains the movement of the corresponding subset of vertices from the CNN approximation. The rigid transformations are computed after the coarse and refinement approximations have been evaluated because the computation relies on the results of the approximation.

To identify the rigidly deformed segments of the mesh, we consider all k mesh segments that are provided by the author of the facial rig. Next, we collect a set of n example mesh deformations $\mathcal{V} = \{\mathbf{V}^1, \mathbf{V}^2, ..., \mathbf{V}^n\}$. Given the mesh in a rest pose $\mathbf{V}^0$, we compute the approximation error of rigidly transforming vertex positions $\mathbf{V}_k^0$ to $\mathbf{V}_k^i$ as

$$e_k^i = \min_{\mathbf{t}_k^i, \mathbf{R}_k^i} \left\| \mathbf{V}_k^0 \mathbf{R}_k^i + \mathbf{t}_k^i - \mathbf{V}_k^i \right\|_F^2 \text{ s.t. } \mathbf{R}_k^i \in \mathrm{SO}(3). \tag{3.3}$$

This equation indicates the difference in vertex positions for mesh segment k in sample i when applying a rigid rotation $\mathbf{R}_k^i$ and translation $\mathbf{t}_k^i$. We average the error across samples $e_k = \frac{1}{n} \sum_{i=1}^n e_k^i$. Rigidly deformed mesh segments can then be identified where $e_k < \tau$. In our experiments, we used $\tau = 0.3$mm.

Let $\mathbf{V}_r^i$ be a rigidly deformed mesh segment (i.e. $e_r < \tau$) for sample i. Let $\mathbf{R}_r^i$ and $\mathbf{t}_r^i$ be the minimizers of Equation 3.3. Let $\mathcal{P}$ be the set of vertex indices in the mesh that are not contained in any rigidly deformed segment. For each vertex $j \in \mathcal{P}$, we compute the approximation error under the transformation $\mathbf{R}_r^i$, $\mathbf{t}_r^i$ across all samples i

$$\epsilon_{r,j} = \frac{1}{n} \sum_{i=1}^n \left\| \mathbf{v}_j^0 \mathbf{R}_r^i + \mathbf{t}_r^i - \mathbf{v}_j^i \right\|_2^2. \tag{3.4}$$

For the rigid mesh segment r, let $\mathbf{V}_\delta^0$ and $\mathbf{V}_\delta^i$ be the set of vertices with the c smallest approximation errors $\epsilon_{r,j}$ where $|V_\delta^0| = c$. In our experiments, we chose $c = 10$. Given the nonlinearly deformed vertices of a mesh $\mathbf{V}'_\mathcal{P}$, the vertex positions for rigid mesh segment r can be approximated as $\mathbf{V}'_r = \mathbf{V}_r^0 \mathbf{R}'_\delta + \mathbf{t}'_\delta$ where $\mathbf{R}'_\delta$ and $\mathbf{t}'_\delta$ are the minimizers of Equation 3.3 for the vertex positions $\mathbf{V}'_\delta$.

Implementation Details

All of the models $\mathbf{f}(\mathbf{p}; \boldsymbol{\theta}_k)$ and $\mathbf{f}(\mathbf{p}; \boldsymbol{\theta}_{k'}^r)$ for the coarse approximation and the refinement stage are implemented as deep neural networks with a series of dense layers followed by convolutional layers. Figure 3.3 shows the structure of both coarse and refinement networks. The networks are trained across two stages. The parameters $\boldsymbol{\theta}_k$ corresponding with the coarse approximation are trained in the first stage to minimize the loss $\mathcal{L}(\mathbf{V}, \boldsymbol{\Delta})$ from Equation 3.1. These models are trained with the Adam optimizer [71] using the momentum parameters suggested by the authors and with a batch size of 8. Optimization starts with the learning rate at 10^{-3}. After the model converges, the learning rate is reduced to 10^{-4}. After convergence again, we reduce the rate to 10^{-5} and run until convergence once more. Once the parameters $\boldsymbol{\theta}_k$ from the coarse approximation are fully optimized, they are held constant while the refinement model parameters $\boldsymbol{\theta}^r_{k'}$ are optimized with the loss $\mathcal{L}(\mathbf{V}, \boldsymbol{\Delta} + \boldsymbol{\delta})$. The same hyper-parameters and training schedule are used for optimization of the refinement model.

When training the approximation models, we compute the rigid mesh segments (Equation 3.3) and the sets of vertices assigned to each rigid segment (Equation 3.4) using the original rig function. During model evaluation, the rigid transformations are computed after the coarse and refinement models are evaluated. The approximated vertex positions are used to compute the rotation matrices and translation vectors, which are then applied to the rigid mesh segments to create the resulting approximated mesh deformation.

To train the facial approximation model, a large set of training data is needed. The training data consist of pairs $(\mathbf{p}, \mathbf{V})$ of rig parameters $\mathbf{p}$ and the vertex positions of the deformed mesh output by the rig function $\mathbf{V} = \mathbf{r}(\mathbf{p})$. To generate the training data, we augment existing animation with multiplicative noise and apply data balancing to prevent common poses found in the animation data from being over-represented in the training data.

Let $\mathcal{A}$ be the set of poses from the training animation, and let m be the number of rig parameters in each pose. We construct the training set as

$$\mathcal{T} = \{\mathbf{u} \odot \mathbf{p} — \mathbf{u} \sim U(0.25, 3.0)^m, \mathbf{p} \sim \mathcal{A}\} \tag{3.5}$$

where $\mathbf{u} \in \mathbb{R}^m$ is a vector of random values with each component drawn uniformly at random in the range $[0.25, 3.0]$ and $\mathbf{p}$ is drawn uniformly at random from the set of poses $\mathcal{A}$. The operation $\odot$ denotes component-wise multiplication of vectors. In our experiments, we generate $|\mathcal{T}| = 50{,}000$ samples for our training set. Figure 3.5 shows example poses from the training data $\mathcal{T}$.

Given the training set, we next balance the data. The training data is generated from existing animation, and certain expressions, such as a neutral expression, might occur more frequently than other poses in the data. A model trained with this dataset would overfit to frequently occurring expressions and would perform poorly when approximating other types of expressions. Taking inspiration from Feng et al. [35], we sort the training examples into bins and draw random samples by picking a bin uniformly at random and then picking a sample within the bin uniformly at random.

To divide the data into bins, we first manually label a small set of landmark vertices around key facial features such as the mouth, eyes, and nose. In our experiments, we manually identified roughly 20-30 landmark points for each character. For each pose $\mathbf{p}^i \in \mathcal{T}$, we gather the positions of the landmark vertices $\mathbf{V}^i_l$ in the deformed mesh. We then use PCA to project the set of landmark positions $\{\mathbf{V}^1_l, \mathbf{V}^2_l, ..., \mathbf{V}^{|\mathcal{T}|}_l\}$ onto a one dimensional space. This one dimensional space is segmented into intervals of equal length along the range of the projected data. The samples are then sorted into bins according to the interval in which they lie. When drawing samples for training, a bin is selected uniformly at random, and from that bin, a sample is selected uniformly at random. In our experiments, we divided the data into 16 bins.

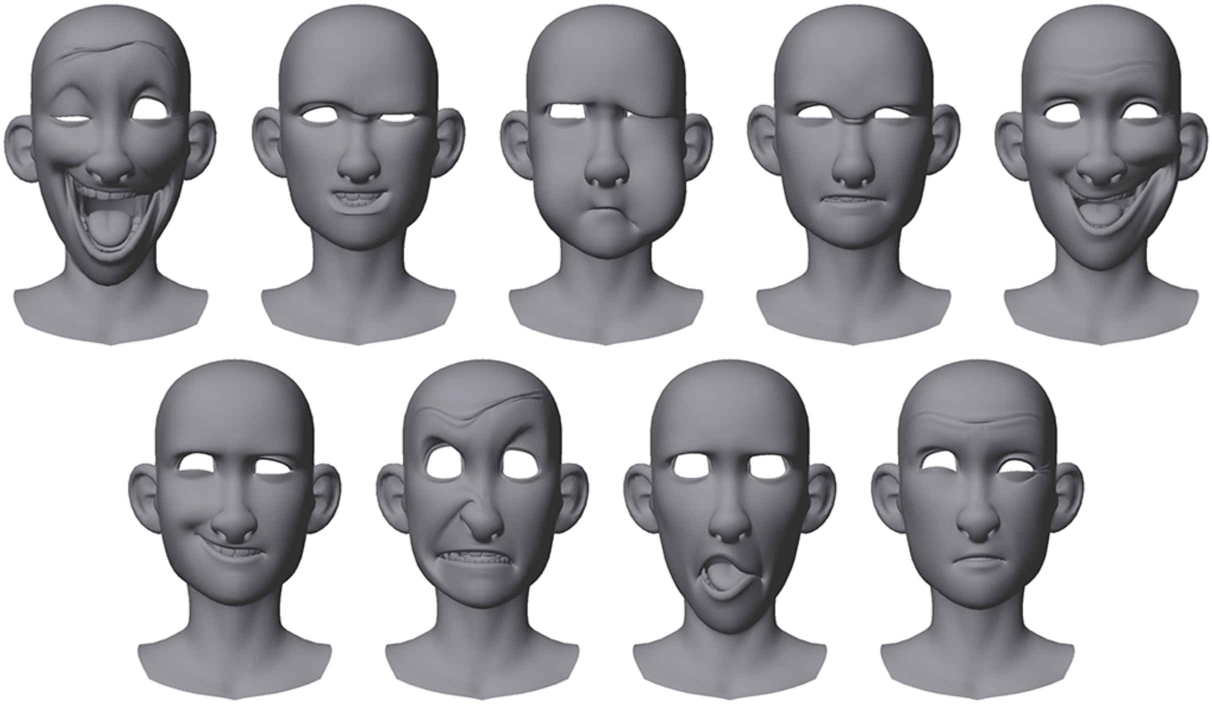


Figure 3.5: Example poses from the training data.[1]

3.4 Results

We built our approximation to work with film-quality facial rigs that are used in computer-animated film production. The rigs are deformed through a combination of a free-form shaping system and a curve-based pose interpolation system. These deformers are layered for coarse to fine control of the mesh to facilitate art-directable facial rigging of the character [119]. The rigs are implemented as node-based computational graphs with more than 10,000 nodes used to compute facial deformations. The nodes implement a wide variety of functions such as basic arithmetic operators and spline interpolation. The rig system also supports custom-written nodes that can execute arbitrary code.

We demonstrate our method using four example facial rigs. Three of these rigs are the proprietary facial rigs used in the feature film *How to Train Your Dragon: The Hidden World* for the characters Hiccup, Valka, and Toothless. The fourth example is the facial rig from the publicly available open-source character, Ray, published by the CGTarian Animation and VFX Online School [9].

We compare our approximation models against linear blend skinning (LBS) approximations and a dense feed-forward version of our approximation models. We observe that our method preserves high-frequency details, which are lost in the LBS approximations, and it is more accurate than the dense models for three out of four character rigs. Furthermore,

Table 3.1: Approximation model statistics for each character rig.

	Hiccup	Valka	Toothless	Ray
Vertices	12,510	12,828	14,080	4,922
Rig Parameters	258	265	286	99
Coarse Segments	10	9	18	6
Refinement Segments	4	3	4	3
Rigid Segments	26	24	110	2
Model Size	8.66 MB	7.39 MB	12.97 MB	5.08 MB
Dense Size	25.96 MB	27.32 MB	31.66 MB	24.19 MB

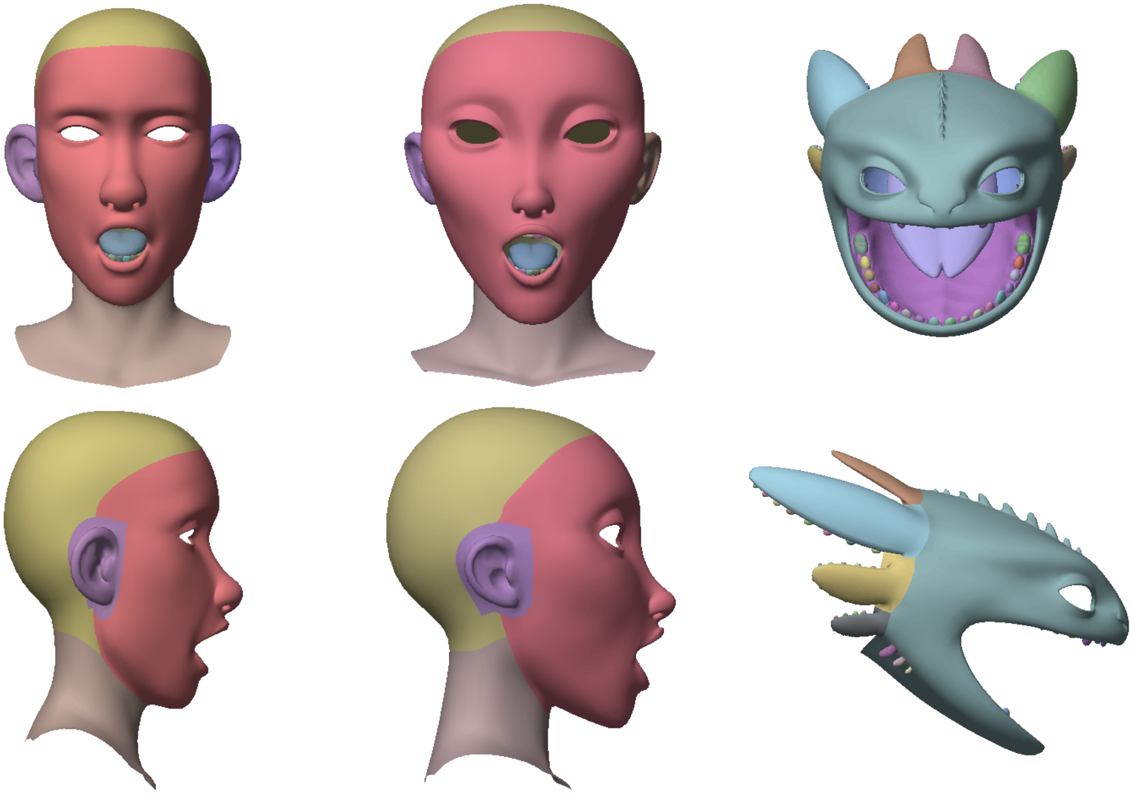
unlike the LBS approximations, our model preserves the mapping from rig parameters to the deformed mesh, which allows our method to approximate novel animations without access to the original rig function.

Table 3.1 shows statistics of each model trained on these characters. The models do not approximate the characters' hair nor their eyeballs. However, the models do approximate the interior of the mouth as well as the teeth. Figure 3.6 visualizes the mesh segments for the facial models of Hiccup, Valka, and Toothless, and Figure 3.7 shows the mesh segments used during the refinement stage of the approximation.

In our results, the dense model runs faster than our approximation model, and in one case is more accurate than our model when approximating artist-created animations. However, the dense approximation's faster speed does come at the cost of more model parameters, which translates to higher memory storage costs as seen in Table 3.1. When the dense model fails, there are visible and undesirable artifacts in the deformed mesh as seen in the facial meshes of Hiccup and Valka in Figure 3.9. These artifacts appear as high-frequency noise on the surface of the mesh and are caused by the dense approximation modeling each component of each vertex as an independent output. Our approximation, on the other hand, models local neighborhoods in the mesh through the use of CNNs, and inaccuracies in the approximation are less likely to manifest as high-frequency noise as in the dense approximation. Furthermore, our model is more accurate than the dense approximation on poses generated through inverse kinematics for all characters.

Comparison

We compare the accuracy of our approximation models to a LBS model and a dense feed-forward network with fully connected layers instead of convolutional layers. We estimate the LBS weights and bone transformations using the method of Le and Deng [85]. The dense model is trained to approximate vertex offsets, and we train a separate network for each mesh segment. Each model is comprised of two hidden layers each of 256 nodes, and



(a) Hiccup (b) Valka (c) Toothless

Figure 3.6: Visualization of the mesh segments of the three characters. Each mesh segment is represented as a continuous region of the same color.[1]

the final output layer is the offsets for each vertex in the mesh segment. Because the dense network is not constrained by deformation map resolution, we do not train an additional refinement model, but we do deform the rigid segments using the same method as our CNN approximation. The dense model most closely resembles the method described by Bailey et al. [5]. The primary difference is that the facial mesh is not linearly deformed by a set of bones before applying the dense neural network.

For each character, we collect all available animation for the rig and randomly split the data 90%/10% into training and test data. We generate training data using only poses from the training set according to Equation 3.5. In the case of Ray's rig, we do not have access to existing facial animation. Instead, we generate the training and test sets by sampling the rig parameters in each pose independently from a uniform distribution covering a user-specified range of values for each parameter. This random sampling method does not work when training the approximation models for the other character rigs due to a higher level of complexity in their mesh deformations. To train our approximation models as well as the

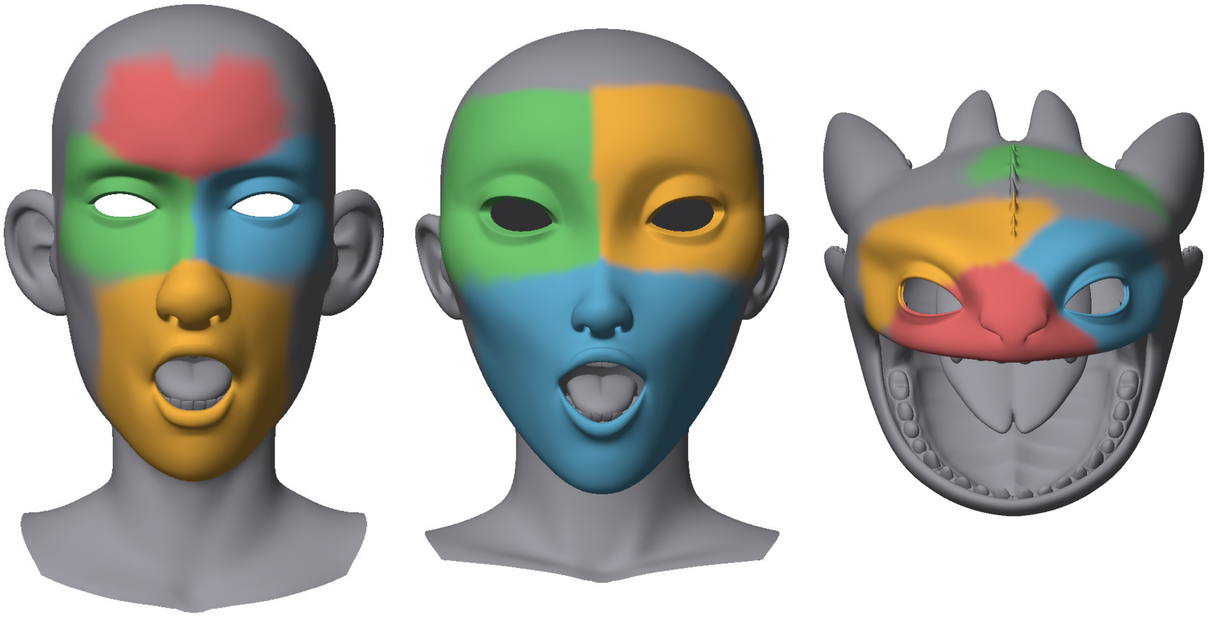


Figure 3.7: Visualization of the mesh segments used for the refinement stage of the approximation model. Gray regions of the mesh indicate segments that are unused in the refinement model.[1]

dense models, we generate 50,000 samples for each character. For the LBS models, we fit 16, 24, and 32 bones to the mesh and allow each vertex to have 8 non-zero weights. In addition, we generate 1,000 samples in order to estimate the vertex weights. We utilize fewer training examples due to memory and computational constraints.

The test sets for Hiccup, Valka, and Toothless are constructed by taking all unique poses from the test data that were unused for training. We measure both the vertex position error and the face normal error in Table 3.2. The vertex error is the mean distance between the approximated and target vertex positions across the test set. The face normal error is the angle between the approximated and target face normals in the mesh. Specifically,

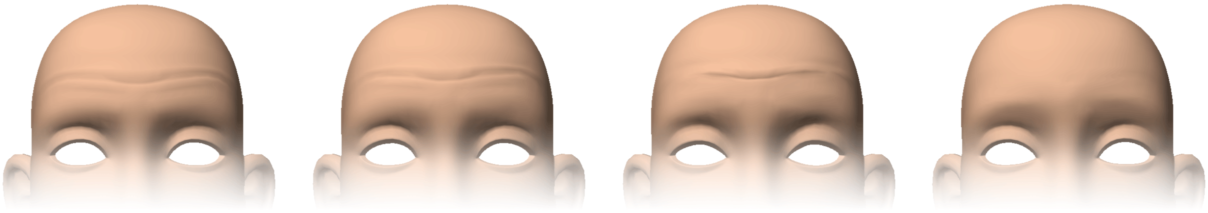
$$E_{\text{normal}} = \frac{1}{f}\sum_{i=1}^{f} \arccos\left(\mathbf{n}_i \cdot \mathbf{n}_i'\right) \tag{3.6}$$

where $\mathbf{n}_i$ is the normal of face i in the ground truth mesh, and $\mathbf{n}_i'$ is the normal of face i in the approximated mesh with a total of f faces.

From the results, we see that most approximations achieve submillimeter accuracy on average. However, the average vertex position error is not a good indicator for the accuracy of fine-scale details in the approximations. Figure 3.1 and Figure 3.8 show approximated deformations for a poses of Toothless and Hiccup containing wrinkles. As seen in Table 3.2, the refined approximation produces the smallest normal error for Hiccup, Valka, and Ray, but

Table 3.2: Average vertex position error measured in mm and average normal angle error measured in degrees.

	Hiccup	Valka	Toothless	Ray
	Distance Error			
Coarse	0.36	0.43	2.01	1.00
Refined	0.27	0.37	1.81	0.40
Dense	0.37	0.98	1.55	5.00
LBS: 16 Bones	0.49	0.33	4.36	0.64
LBS: 24 Bones	0.35	0.25	2.77	0.46
LBS: 32 Bones	0.24	0.21	2.35	0.44
	Normal Angle Error			
Coarse	1.6	1.7	3.1	3.8
Refined	0.9	1.1	2.4	1.5
Dense	1.9	5.5	1.9	8.9
LBS: 16 Bones	2.0	1.9	5.2	4.2
LBS: 24 Bones	1.9	1.8	4.3	3.7
LBS: 32 Bones	1.6	1.6	4.5	4.1



(a) Ground Truth (b) Refined Approx. (c) Coarse Approx. (d) LBS Approx.

Figure 3.8: Comparison of forehead wrinkles on Hiccup's mesh using our approximation and LBS.[1]

the dense model produces the smallest error for Toothless. This smaller error indicates that both the refined approximation and the dense approximation can reproduce fine-scale details in the deformed mesh when compared to our coarse approximation and LBS. Figure 3.9 shows side-by-side comparisons with a visualization of the normal error. See the supplementary video for more results. The trained network for Ray, along with other supporting files needed for benchmarking and comparison, are available as supplemental materials [2].

[2] http://graphics.berkeley.edu/papers/Bailey-FDF-2020-07

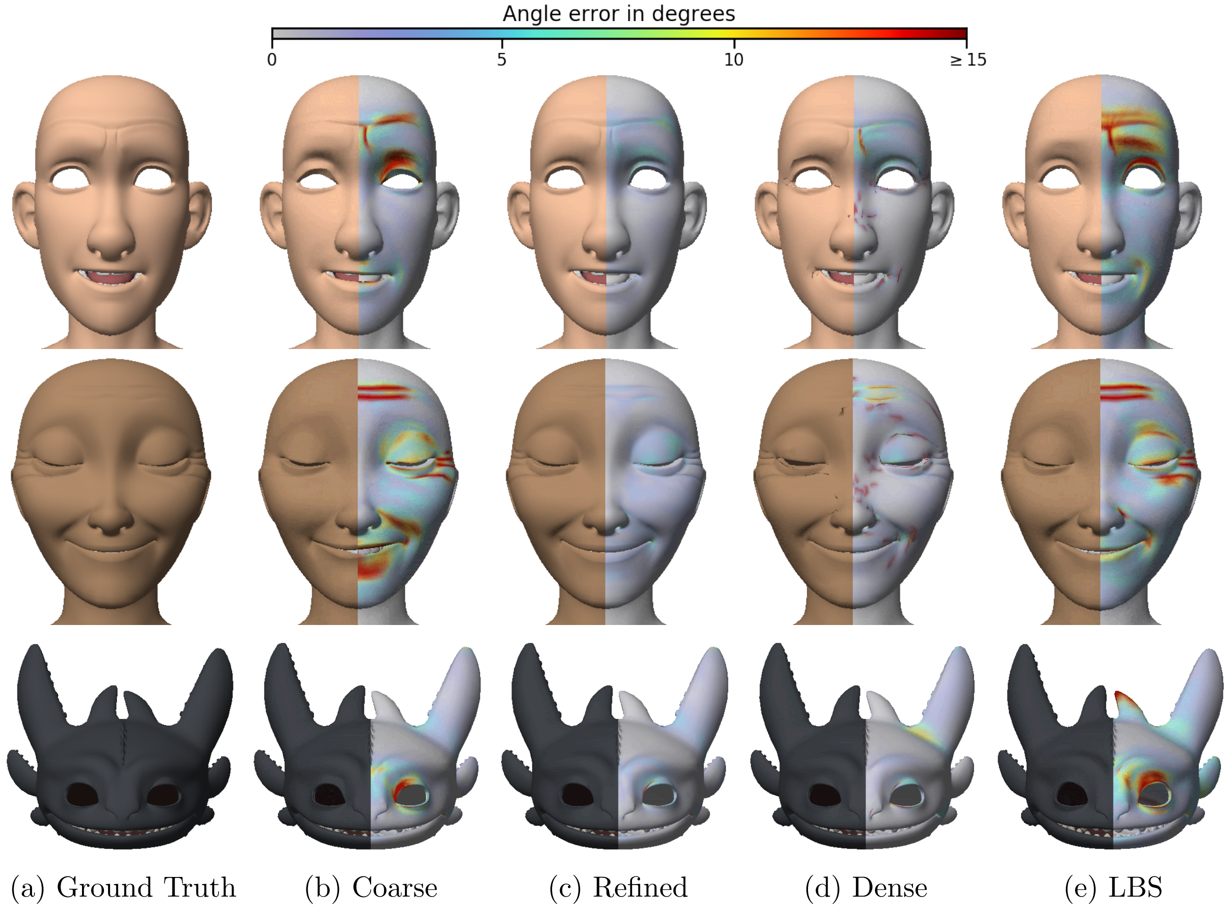


Figure 3.9: Visual difference between the ground truth mesh evaluated through the original rig function and the rig approximation methods. The heatmap on the right half of each approximation visualizes the angle between the normal vector on the approximation and the corresponding normal on the ground truth mesh. Smaller angles are better.[1]

Timing

We implement our approximation models in Python with TensorFlow. We evaluate their execution times on both a high-end machine and a consumer-quality laptop using both the CPU and the GPU. For the high-end machine, we use an Intel Xeon E5-2697 v3 processor with 28 threads running at 2.60 GHz along with an NVIDIA Quadro K5200 GPU. On the laptop, we use an Intel Core i7-7700HQ processor with 8 threads running at 2.80 GHz along with an NVIDIA GeForce GTX 1060. The rotation for the rigid segments is computed by minimizing Equation 3.3 with the SVD. When evaluating the full approximation with the GPU, we solve this minimization problem on the CPU due to TensorFlow's slow implementation of the SVD on the GPU. Model training time consisted of 2-4 hours spent generating

Table 3.3: Average evaluation time in milliseconds on both the high-end machine and the consumer-quality machine. The coarse approximation is timed by evaluating the coarse model and the rigid deformations. The full approximation is timed by evaluating the coarse model, the refinement model, and the rigid deformations. Where indicated, the neural network is evaluated on the GPU, but the rigid components are always evaluated on the CPU.

		Hiccup	Valka	Toothless
Original Rig	High-end	75	66	30
Coarse w/ GPU		6.4	6.0	10.4
Full w/ GPU		8.7	7.5	12.6
Dense w/ GPU		2.6	2.6	4.2
Coarse		2.9	2.7	4.4
Full		4.2	3.8	5.6
Dense		2.1	2.2	2.9
Coarse w/ GPU	Consumer	5.8	3.2	7.3
Full w/ GPU		4.3	4.5	9.1
Dense w/ GPU		1.8	1.7	7.3
Coarse		6.9	2.7	5.7
Full		3.5	5.2	9.0
Dense		3.4	2.5	3.35

training data through the original rig evaluation engine followed by 2-3 hours of training the coarse approximation model and 2-3 hours for the refined approximation model.

We compare the timing of our approximation against the original rig evaluation software for Hiccup, Valka, and Toothless. These three character rigs are designed for Libee [153], a multi-threaded rig evaluation engine. Character artists optimized these rigs to run as fast as possible on the engine. Unlike our method, Libee can only evaluate character rigs on the CPU. Table 3.3 shows the evaluation times using Libee and our method running both on the CPU and the GPU. We time our models by taking the average execution time across 1,000 evaluations on single poses.

From these results, we observe that our approximation models runs from 5 to 17 times faster than the original rig evaluation engine. In the case of the high-end machine, the approximation runs slower on the GPU because the model is evaluated on a single pose and because the convolutions operate on feature maps with low resolution. Thus, the GPU is underutilized in this case, which leads to slower performance. Furthermore, we find that the GeForce GPU on the consumer-quality machine evaluates the approximation models faster than the Quadro GPU on the high-end desktop. This difference can be attributed to the slower clock speed of the Quadro compared with the GeForce GPU.

3.5 Discussion

Our method provides a fast and accurate approximation of film-quality facial rigs. We have shown that our approximation can preserve details of fine-grain mesh deformations where bone-based approximations are unable. In addition, our approach provides a differentiable rig approximation, which allows for a wide range of potential new applications for the character rig. Additionally, once the model is trained, our method no longer requires the original rig function to evaluate mesh deformations. Because the approximations can be implemented with open-source machine learning libraries, the models can be easily distributed and deployed on many different systems without requiring the complex or proprietary software that was initially used to build the facial rig. Thus, our approximation model provides a common format in which facial rigs can be shared without a dependency on the original rigging software. Furthermore, the approximation model parameters can be viewed as a form of rig obfuscation such that the underlying rigging techniques used to create the character are hidden when the model is shared.

Because our method is built upon convolutional layers the model is not restricted to a single mesh topology. The approximation model trained on a certain mesh can deform a novel mesh not seen during training. As long as the texture coordinates of the facial features in the new mesh align with the texture coordinates of the original, the approximation rig can be transferred to the new facial mesh. In this case, the approximation models output the deformation maps using the same set of input rig parameters. Vertex offsets for the new mesh are computed by sampling deformation maps at new texture coordinates corresponding with the new mesh. Figure 3.10 shows an example of transferring one mesh segment of the coarse approximation model onto a new mesh with a different topology. In this example, the texture coordinates are manually aligned to those of the original mesh.

The approximation method outputs vertex offsets in a world coordinate system. As a result, the deformations applied to a new mesh might appear undesirable if the facial proportions of the mesh differ significantly from the original model. A different parameterization of the offsets output by the approximation model could help alleviate this issue and allow our method to transfer the approximation from one rig to a facial mesh with significantly different proportions.

In our examples, vertex normals are computed separately, and are not considered as part of the approximation model. However, in certain real-time applications, recomputing normals from a deformed mesh is avoided to save computational time. Although we did not experiment with approximating vertex normals in our model, the method could easily be extended to approximate normals as well. Instead of outputting 3 channels in the deformation maps, the network could output additional channels for the normal directions, and an additional loss term could be included to train the model to output accurate normal vectors. Due to the small resolution of the intermediary feature maps, this approach would only be appropriate for approximating vertex or face normals. Normal maps or other high-resolution maps such as ambient occlusion maps would need to be created using other means.

In our implementation, we used the texture coordinates provided with each character rig to interpolate from deformation maps to vertex offsets. Although these coordinates work well for mapping textures to the mesh surface, they might not be well-suited for our method. For example, the texture coordinates for the upper lip and lower lip of a character's mouth could be near each other. Vertices on the lower lip can move far away from the upper lip when the mouth opens. If the texture coordinates are close enough together, then vertices on both lips might lie on the same pixel in the deformation map. If this were the case, then visually the lips would appear stuck together when the mouth opens, which would be an inaccurate deformation. To avoid these types of issues, new deformation map coordinates could be generated specifically for this approximation task rather than relying on pre-existing texture coordinates.

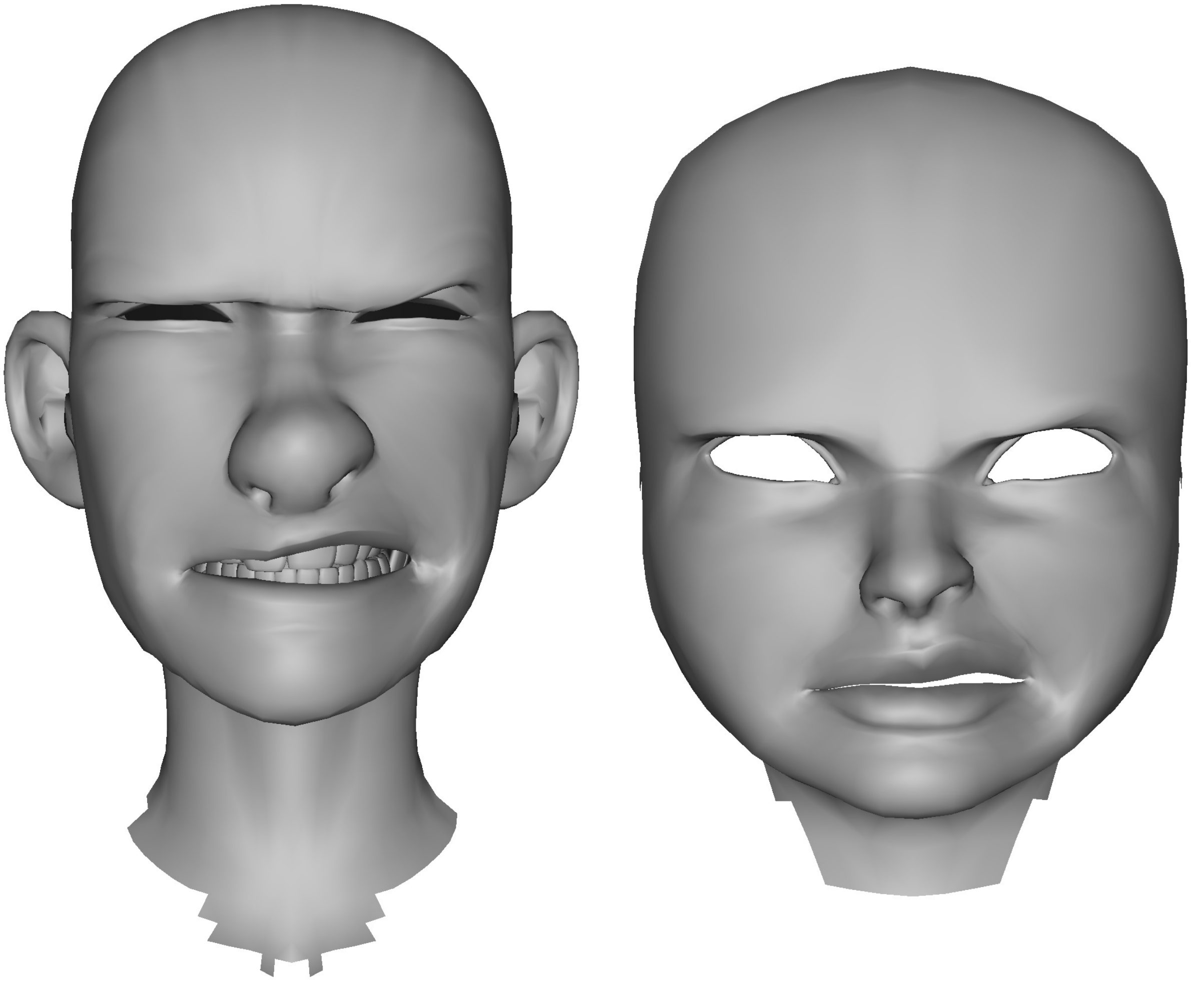


Figure 3.10: Rig approximation of Hiccup transferred to a new mesh with a different topology. A single mesh segment from the coarse approximation is applied to the new mesh on the right. The facial mesh on the right is from the freely available Mathilda Rig developed by Leon Li-Aun Sooi and Xiong Lin.[1]

Chapter 4

Inverse Kinematics with Mesh Approximations

4.1 Introduction

Character rigs are manipulated through artist-friendly parameters that give an animator low-level control over a character's deformation. For example, an artist posing a character's arm and hand might specify the rotations for every bone in the arm and hand. However, specifying bone rotations becomes challenging when a character needs to interact with an environment such as fixing a foot in place on the ground while it walks. Inverse Kinematics (IK) helps solve this problem by allowing an animator to specify the location of a control point (such as a foot) and then automatically computing the necessary bone rotations to deform a character to match the control point's location.

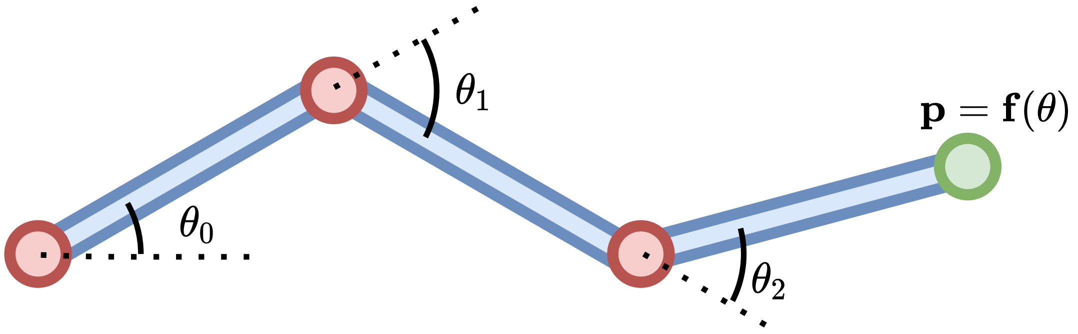


Figure 4.1: Example of a kinematic chain with three bones. The bone configurations are parameterized by θ_0, θ_1, and θ_2. The forward kinematic function $\mathbf{p} = \mathbf{f}(\boldsymbol{\theta})$ provides the position of the control point on the end of the bone on the right.

Inverse kintematics is well-studied in the field of robotics, and Aristidou et al. [3] provide a comprehensive survey of the use IK for computer animation. Given a kinematic chain of bones parameterized by bone configurations $\boldsymbol{\theta}$ such than the position of a control point on the chain is a function of the bone configurations $\mathbf{p} = \mathbf{f}(\boldsymbol{\theta})$, which we refer to as the forward kinematic function. Figure 4.1 illustrates an example of a kinematic chain with three bones.

The IK function is represented as the inverse of the forward kinematic function such that $\boldsymbol{\theta} = \mathbf{f}^{-1}(\mathbf{p})$. Given some target $\mathbf{t}$ for the control point, the solution to the IK problem seeks to find an acceptable bone configuration $\boldsymbol{\theta'}$ such that $\mathbf{t} = \mathbf{f}(\boldsymbol{\theta'})$. Iterative methods are popular solutions to this inverse problem, especially for computer graphics and animation. First, an error function is defined as

$$\mathbf{e}(\boldsymbol{\theta}) = \mathbf{t} - \mathbf{f}(\boldsymbol{\theta}). \tag{4.1}$$

To optimize this function, the Jacobian of the forward kinematics function is computed or estimated $J(\boldsymbol{\theta})_{ij} = \frac{\partial e_i}{\partial \theta_j}$. Depending on the complexity of the forward kinematic function, the Jacobian can be computed analytically or numerically. Finally, an update step for the bone configurations can be estimated as $\Delta\boldsymbol{\theta} \approx \mathbf{J}(\boldsymbol{\theta})\mathbf{e}(\boldsymbol{\theta})$.

For many IK problems, the loss function might be underconstrained or the Jacobian might not be a square matrix. Many methods have been proposed to compute an update step given a non-invertible Jacobian [3]. One popular approach is to use the pseudo-inverse of $\mathbf{J}(\boldsymbol{\theta})$ such that

$$\Delta\boldsymbol{\theta} = \mathbf{J}(\boldsymbol{\theta})^\top(\mathbf{J}(\boldsymbol{\theta})\mathbf{J}(\boldsymbol{\theta})^\top)^{-1}\mathbf{e}(\boldsymbol{\theta}). \tag{4.2}$$

Because our rig approximation methods compute mesh deformations in real-time, fast IK methods could be used with our approximations to provide interactive character control. Because the body deformation approximation relies on an underlying character skeleton, we can use traditional IK methods to manipulate characters' limbs. However, this approach is not applicable to our facial approximation because it does not use bones as input. We present our real-time solutions to the IK problem for both our body and facial rig approximation methods.

4.2 Solutions for the Body

Often, IK problems are underconstrained, and the solution is not unique. These iterative methods converge to a single solution, but the result depends on the initial configuration when optimizing the error function. For character rigs, the IK problem for positioning hands and feet are usually underconstrained. However, artists want the solution of the IK problem to be deterministic such that the same input rig parameters always produce the same deformed mesh. Borrowing solutions for character rigs in production, we deterministically solve the IK problem for the hands and feet.

Our solution relies on the fact that human rigs and human-like rigs consist of two bones between the torso and the hands and feet with a hinge joint connecting the two bones and

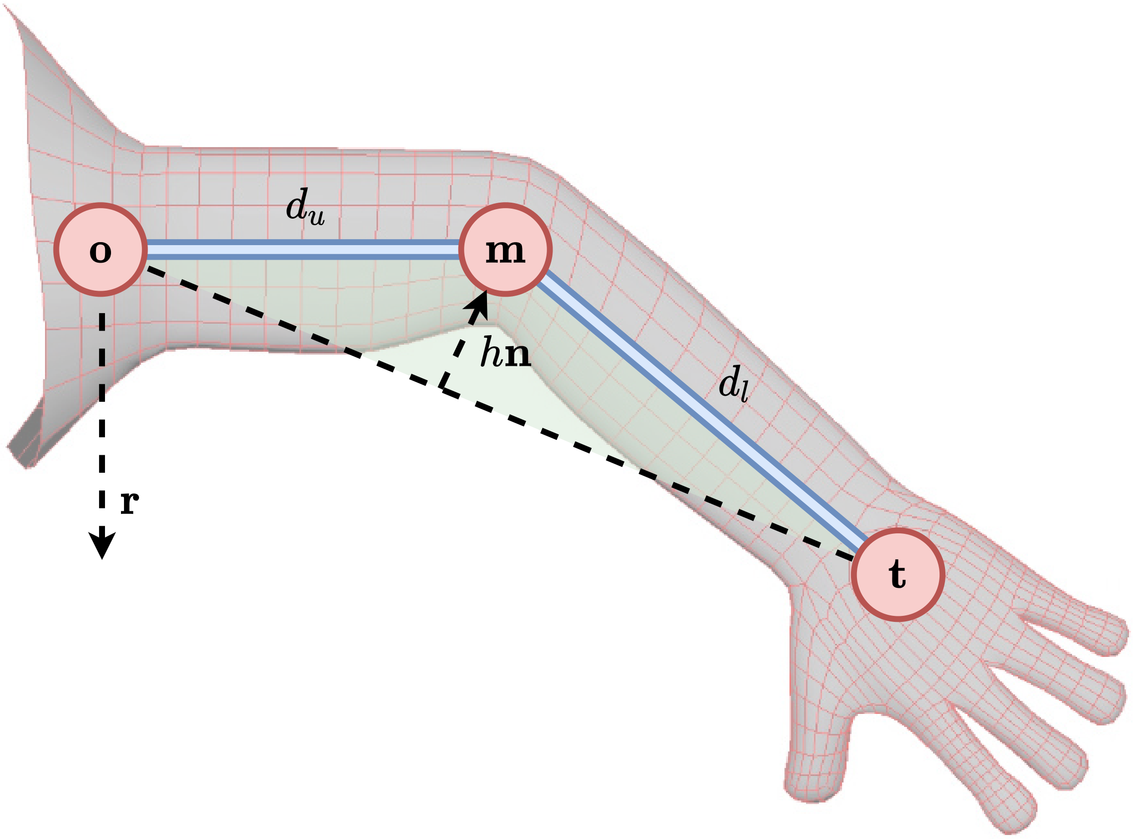


Figure 4.2: Illustration of aligning bones to a triangle specified by target control point **t**.

a ball joint connecting one of the bones to the torso. Using the arm as an example, we construct a triangle with the shoulder, elbow, and hand as vertices. We then rotate the upper arm such that the elbow lies on the corresponding vertex, and then we rotate the hand to align it with the remaining vertex. Assuming that the triangle can be computed deterministically, then the solution to this IK problem with two bones has a closed-form solution.

We describe our IK solution for a character's arm, but this approach generalizes to legs with two bones as well. Let d_u be the length of the upper arm bone, let d_l be the length of the lower arm bone, and let $\mathbf{o}$ be the position of the base of the upper arm bone. The user inputs the target position of the hand $\mathbf{t}$ as well as a reference direction $\mathbf{r}$, which controls the orientation the triangle that the bones are aligned to. We compute the normalized vector

$$\mathbf{n} = \frac{(\mathbf{t} - \mathbf{o}) \times \mathbf{r}}{\|(\mathbf{t} - \mathbf{o}) \times \mathbf{r}\|} \tag{4.3}$$

and place the elbow joint on the plane spanned by the vectors $\mathbf{t} - \mathbf{o}$ and $\mathbf{n}$. The joint is placed on the remaining vertex of the triangle whose vertices contain $\mathbf{o}$ and $\mathbf{t}$ with lengths $\|\mathbf{t}-\mathbf{o}\|$, d_u, and d_l. Using Heron's formula, we first compute the area of the triangle with base $\mathbf{t}-\mathbf{o}$ as

$$A = \frac{1}{4}\sqrt{4d_u^2 d_l^2 - (d_u^2 + d_l^2 - \|\mathbf{t}-\mathbf{o}\|^2)^2} \tag{4.4}$$

and the height h of the triangle with base $\mathbf{t}-\mathbf{o}$ is given as $h = 2A/\|\mathbf{t}-\mathbf{o}\|$. Finally, the position $\mathbf{m}$ of the elbow joint can be computed as

$$\mathbf{m} = \frac{\sqrt{d_u^2 - h^2}(\mathbf{t}-\mathbf{o})}{\|\mathbf{t}-\mathbf{o}\|} + h\mathbf{n} + \mathbf{o}. \tag{4.5}$$

Given the positions $\mathbf{o}$, $\mathbf{m}$, and $\mathbf{t}$, the two bone configurations can be computed in closed-form. In the case where the IK problem has no solution when the control point $\mathbf{t}$ is out of reach of the arm, we leave the elbow joint unbent and point the upper arm in the direction $\mathbf{t}-\mathbf{o}$.

Our IK method is a fast alternative to iterative methods when posing a two-bone kinematic system, which is suitable for common IK problems with human-like character rigs. We combine this IK method with our fast body deformation approximation, which allows a high-quality character rig to be evaluated in real-time on a low-end system or even a mobile device. We implemented a posing application for the iPad in which the user can pose the arms and legs of the character using IK controls. Figure 4.3 shows a screenshot of a user interacting with the application to pose Po.

We use this IK method to compute the input skeleton as a simplified skeletal system. The user can manipulate five points: one each for the hands and feet and one control for the torso position. To compute the resulting skeleton, we apply the torso position as a global offset to all bones in the skeleton. Next, the system only manipulates the two bones in each arm and leg. The resulting skeleton is then passed to the deformation approximation.

4.3 Solutions for the Face

Facial character rigs for production use are typically constructed in such a manner that computing the gradient of the vertex positions with respect to the rig parameters $\partial\mathbf{V}/\partial\mathbf{p}$ would be difficult and extremely slow. Using the approximation model that we propose, estimating the gradient becomes possible and is trivial with automatic differentiation, which is a common feature in deep learning libraries. One useful application of this gradient is inverse kinematics where rig parameters are estimated to best deform the mesh in order to match user-specified control point positions. Common solutions to inverse kinematics formulate it as an iterative optimization problem [3]. These types of solutions would require multiple gradient evaluations before converging on the optimal rig parameters. Although the approximation model can be used to estimate $\partial\mathbf{V}/\partial\mathbf{p}$, computing the gradient multiple times

[1] These images are Property of DreamWorks Animation L.L.C., used with permission.

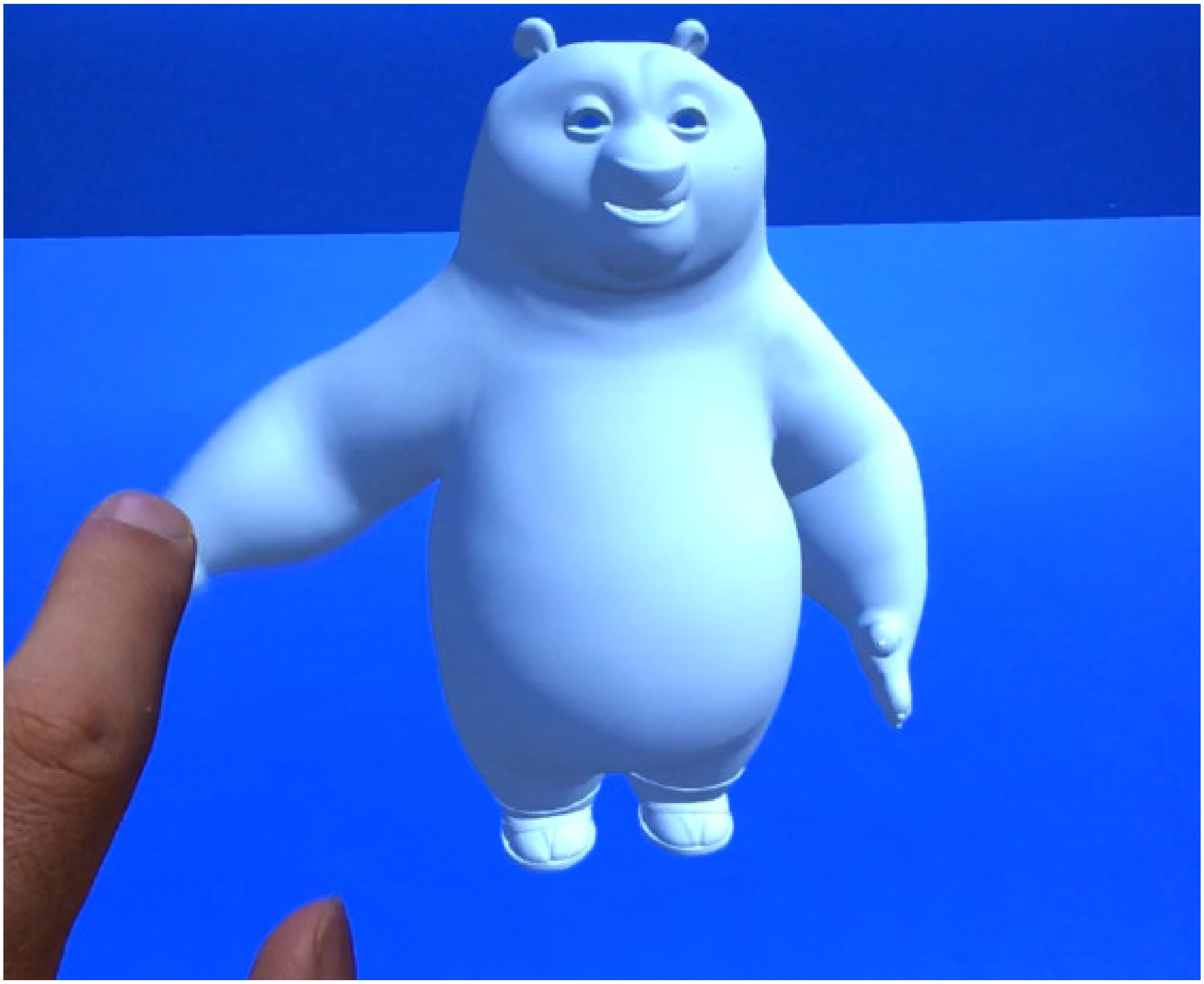


Figure 4.3: Posing example on iPad.[1]

through our approximation model for an iterative optimization method requires too much computation to run in real-time. Instead, we propose a feed-forward neural network that takes the IK control points as input and outputs the corresponding rig parameters. During training, the network utilizes the approximation gradient, but does not require $\partial \mathbf{V}/\partial \mathbf{p}$ when evaluated on new inputs. As a result, the feed-forward network can compute the desired rig parameters in real-time.

Model Details

Let $\mathcal{C}$ be the set of indices of vertices corresponding to IK control points, and let $\mathbf{r}_{\mathcal{C}}(\mathbf{p}) : \mathbb{R}^m \rightarrow \mathbb{R}^{|\mathcal{C}| \times 3}$ be the rig function that maps the rig parameters $\mathbf{p}$ to the subset of vertices

$\mathbf{V}_{\mathcal{C}}$. Then the inverse kinematics problem can be expressed as

$$\mathbf{p}' = \arg\min_{\mathbf{p}} \left\| \mathbf{r}_{\mathcal{C}}(\mathbf{p}) - \overline{\mathbf{V}}_{\mathcal{C}} \right\|_F^2 \tag{4.6}$$

where $\overline{\mathbf{V}}_{\mathcal{C}}$ are the target control points provided by the user. Due to the assumption that the rig function $\mathbf{r}$ is not differentiable, we replace it with the approximation, denoted as $\tilde{\mathbf{r}}$. Furthermore, instead of solving the minimization problem with an iterative algorithm, we introduce a feed-forward network $\mathbf{f}_{IK} : \mathbb{R}^{|\mathcal{C}| \times 3} \to \mathbb{R}^m$ to approximate the minimization problem through a fixed-length computation such that

$$\mathbf{f}_{IK}(\overline{\mathbf{V}}_{\mathcal{C}}; \boldsymbol{\theta}_{IK}) = \arg\min_{\mathbf{p}} \left\| \widetilde{\mathbf{r}}_{\mathcal{C}}(\mathbf{p}) - \overline{\mathbf{V}}_{\mathcal{C}} \right\|_F^2 \tag{4.7}$$

where $\boldsymbol{\theta}_{IK}$ are the network parameters that require training. The model is trained on a specific set of control points and vertices $\mathbf{V}_{\mathcal{C}}$, and a new network would need to be trained for any different set of vertices.

The loss function used to train the model contains both a point-matching component to ensure that the deformed mesh closely matches the control points as well as a regularization component to avoid large rig parameters that would create unnatural poses. The loss is expressed as

$$\mathcal{L}(\overline{\mathbf{V}}_{\mathcal{C}}) = \mathcal{L}_{point}(\overline{\mathbf{V}}_{\mathcal{C}}) + \lambda_{reg}\mathcal{L}_{reg}(\overline{\mathbf{V}}_{\mathcal{C}}) \tag{4.8}$$

where $\lambda_{reg} \in \mathbb{R}$ is a user-defined regularization weight. The point-matching loss computes the distance between the points generated by the estimated pose and the corresponding control points

$$\mathcal{L}_{point}(\overline{\mathbf{V}}_{\mathcal{C}}) = \frac{1}{|\mathcal{C}|} \sum_{i \in \mathcal{C}} \left\| \widetilde{\mathbf{r}}_i(\mathbf{f}_{IK}(\overline{\mathbf{V}}_{\mathcal{C}}; \boldsymbol{\theta}_{IK})) - \overline{\mathbf{v}}_i \right\|_1 . \tag{4.9}$$

The regularization term penalizes large parameter values as

$$\mathcal{L}_{reg}(\overline{\mathbf{V}}_{\mathcal{C}}) = \left\| (\mathbf{f}_{IK}(\overline{\mathbf{V}}_{\mathcal{C}}; \boldsymbol{\theta}_{IK}) - \mathbf{p}^0) \odot \mathbf{s} \right\|_1 \tag{4.10}$$

where $\mathbf{p}^0$ defines the neutral expression of the character and $\mathbf{s} \in \mathbb{R}^m$ defines individual scaling values for each rig parameter. For rig parameter i, the scale is given by $s_i = 1/(p_{i,max} - p_{i,min})$ where $p_{i,max}$ and $p_{i,min}$ are the maximum and minimum values for rig parameter i in the animation data $\mathcal{A}$. Scaling each parameter separately ensures that regularization is applied equally to each parameter regardless of the difference in their ranges of values. Furthermore, we use the L1 regularization loss to encourage sparsity in the estimated pose $\mathbf{p}$.

An ideal IK approximation model $\mathbf{f}_{IK}$ would avoid learning incorrect correlations between certain rig parameters and control points. For example, if a user were to adjust a control point on the left eye of a character, the approximation model should avoid changing rig parameters related to the mouth. We guarantee this property by designing the IK approximation model as a combination of multiple networks. The control points are divided into separate sets based on the regions of the face. For example, all of the points on the right eye of the

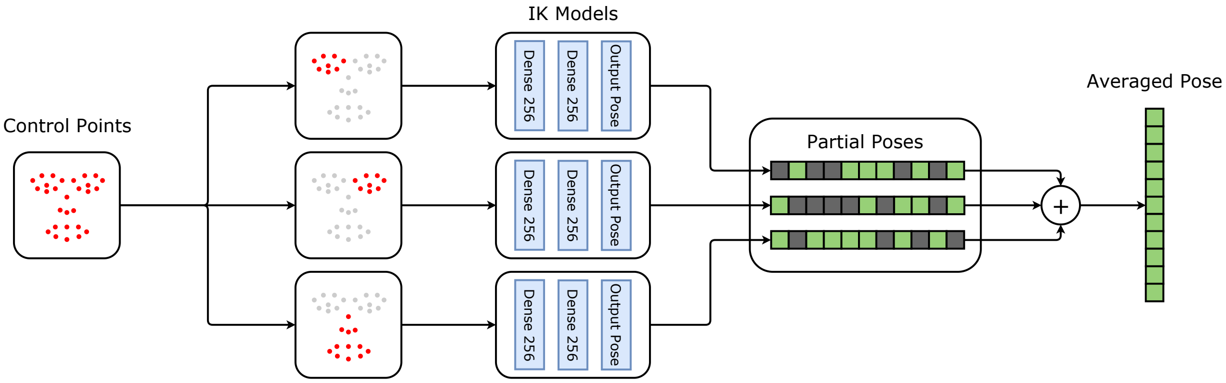


Figure 4.4: Diagram of IK model. Control points are divided into disjoint subsets and provided to separate dense neural networks. Each network outputs a subset of the pose. The valid values from the outputs are averaged together to produce the final averaged rig parameter pose.

character define one subset, and all of the points on the mouth define a separate subset. In our experiments, the points are divided manually.

Let the control points be divided into k subsets, and let $\mathcal{C}_j$ denote subset j. In total, the IK approximation model consists of k separate feed-forward networks. The input to model j is the subset of control points $\overline{\mathbf{V}}_{\mathcal{C}_j}$, and the output is the set of rig parameters that can deform any of the vertices corresponding to the control points. Rig parameters can be estimated by multiple models. In this case, the final estimated value is the average of the outputs. More sophisticated methods could be used to compute the final value of rig parameters predicted by multiple networks. However, averaging the values worked well with our rigs. For the character faces, only a small fraction of the rig parameters are shared between IK models. Of the shared parameters, almost all of them control large-scale deformations of the face such as squash and stretch of the entire head. Because these controls drive large deformations across all regions of the mesh, IK models trained on control points for small portions of the mesh will generally agree on parameter values for these types of global deformations. Thus, we can achieve reasonable results by simply averaging these parameters.

Implementation Details

Each network of the IK approximation model consists of three dense layers with 256 nodes in the first two layers and $|\mathcal{R}_j|$ nodes in the final layer where $\mathcal{R}_j$ is the set of rig parameters estimated by IK model j. The leaky ReLU activation function is applied after the first and second layers. No activation is applied to the output of the final layer so that the network can output any value for the rig parameters. A diagram of the network is shown in Figure 4.4. Similar to the facial approximation model, the IK model is optimized with Adam using the same training schedule and balanced dataset described in Section 3.3.

As described, the IK model is trained using control points from deformed meshes computed through the rig function. Thus, the training data only contains examples of control points that can be matched exactly with the appropriate rig parameters. However, when evaluating the IK model, a user might configure the control points in a way such that rig cannot precisely match the points. To account for this use case, we add noise to the control points during training. Given a training sample $\overline{\mathbf{V}}_{\mathcal{C}}$, a new sample is computed as $\overline{\mathbf{V}}'_{\mathcal{C}} = \overline{\mathbf{V}}_{\mathcal{C}} + \mathrm{U}(-\delta, \delta)^{|\mathcal{C}| \times 3}$ for some user-defined $\delta > 0$. This new data point is created by adding uniformly random noise to each control point's position. In our experiments, we found that $\delta = 4.5$mm produces reasonable results. The IK model is trained with this new data $\overline{\mathbf{V}}'_{\mathcal{C}}$, but all other aspects of model training remain identical.

4.4 Results

The facial approximation method provides a differentiable model that maps rig parameters to the deformed mesh, which can be used for IK applications. We demonstrate the uses of our approximation through an interactive posing application and a facial landmark-based performance capture system.

Character Posing

We develop a real-time posing application in which the user manipulates a sparse set of control points, and our IK model computes rig parameters that deform the mesh to match the control points. The user drags the points across the screen, and the mesh is updated interactively. The control points are provided to the system as 2D image coordinates. We train the IK model to match the points by projecting the mesh onto the image plane and express the point loss term in Equation 4.9 in terms of distance in image coordinates. We project the mesh onto the image plane through an orthographic projection with the camera pointing along the Z axis. Thus, the distance in image coordinates can be computed by only the X and Y coordinates of the vertex positions.

The IK model is trained on meshes generated from the same augmented dataset used to train the approximation models. Excluding the time taken to generate the meshes from the original rig function, training takes 1-2 hours.

We compare our approximation method with the dense neural network approach. IK models are trained using both methods as the rig approximation $\widetilde{\mathbf{r}}(\mathbf{p})$ from Equation 4.7. In our experiments, the IK models trained with gradients from the dense model for Hiccup, Valka, and Ray produce poses for which the dense approximation generates significantly inaccurate deformations with blatant visual artifacts. For these three characters, we instead use poses generated from the IK model trained using gradients from our approximation method. In the case of Toothless's rig, we evaluate the dense model with poses generated from an IK model trained with gradients from the dense approximation. To evaluate the models, we collect 25 user-generated control point configurations. There is no guarantee that

Table 4.1: Posing errors measured in mm and degrees. For Toothless, the IK models are trained using gradients from the corresponding approximation. For Hiccup, Valka, and Ray, the IK model is trained with gradients from our method and generates rig parameters for both our approach and the dense method.

	Hiccup	Valka	Toothless	Ray
	Distance Error			
CNN (ours)	0.94	0.70	5.49	0.58
Dense	1.92	2.19	11.19	4.19
	Normal Angle Error			
CNN (ours)	2.8	1.7	3.0	1.5
Dense	5.6	8.9	4.2	8.5

these control point configurations can be matched exactly by the original rig. Next, the IK model computes rig parameters for the control points. Finally, a mesh is generated using the approximation method, and a ground truth mesh is generated using the original rig function evaluated on the same rig parameters. We measure the per-vertex distance error and the per-face normal error between the approximated and ground truth meshes. For Toothless, the approximation model is fed poses generated from the IK model trained on its gradients. For Hiccup, Valka, and Ray, both our model and the dense model are fed poses from the IK model trained on gradients from our method. As seen in Table 4.1, our method more closely matches the ground truth mesh evaluated on rig parameters output by the IK model. Figure 4.5 shows a side-by-side comparison of the ground truth mesh and the approximated deformation for several example control point configurations.

The larger difference in accuracy between our approximation and the dense approximation for Hiccup, Valka, and Toothless can be explained by the types of poses output by the IK model. The IK model is trained in an unsupervised setting, and the distribution of poses output by the model does not exactly match the distribution of poses from the training data. Thus, some poses output by the IK model are dissimilar from the original training data. Higher accuracy on these poses suggests that our approximation model generalizes to new poses better than the dense model. The results from Ray further support this conclusion. Both the CNN and dense models for Ray are trained on poses sampled uniformly at random. Any pose output by the IK model will lie somewhere within this distribution. As seen in these results, the average approximation error for both the CNN and the dense model for Ray are similar when evaluated on a uniformly random set of poses (Table 3.2) and on the set of poses output by the IK model (Table 4.1).

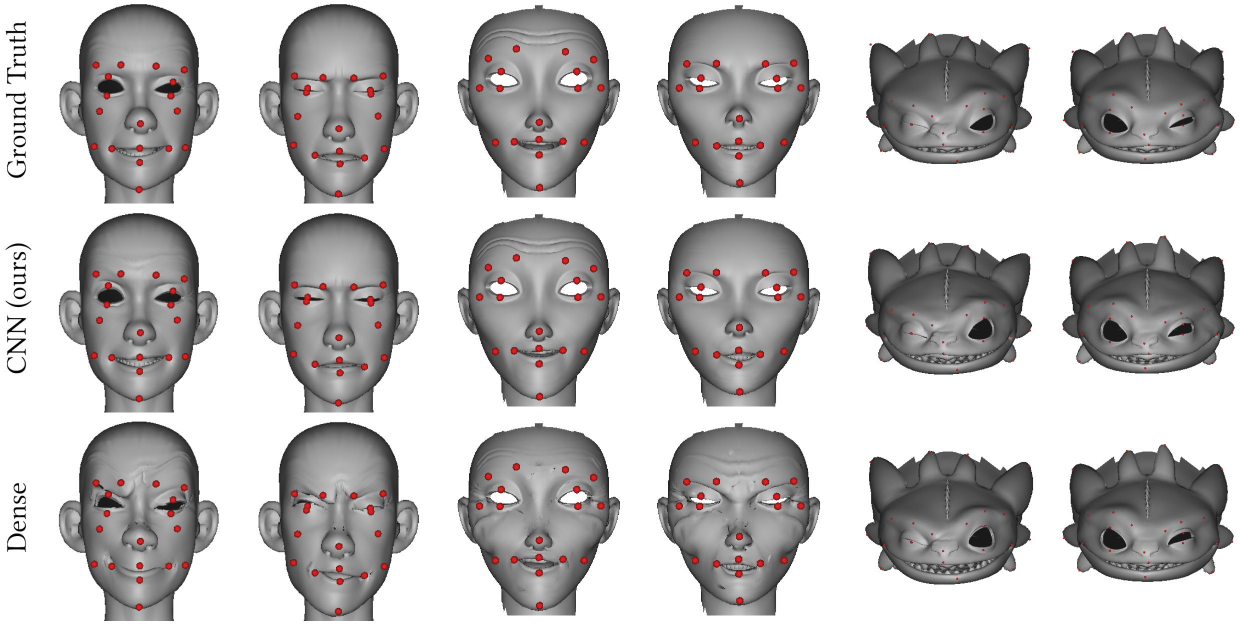
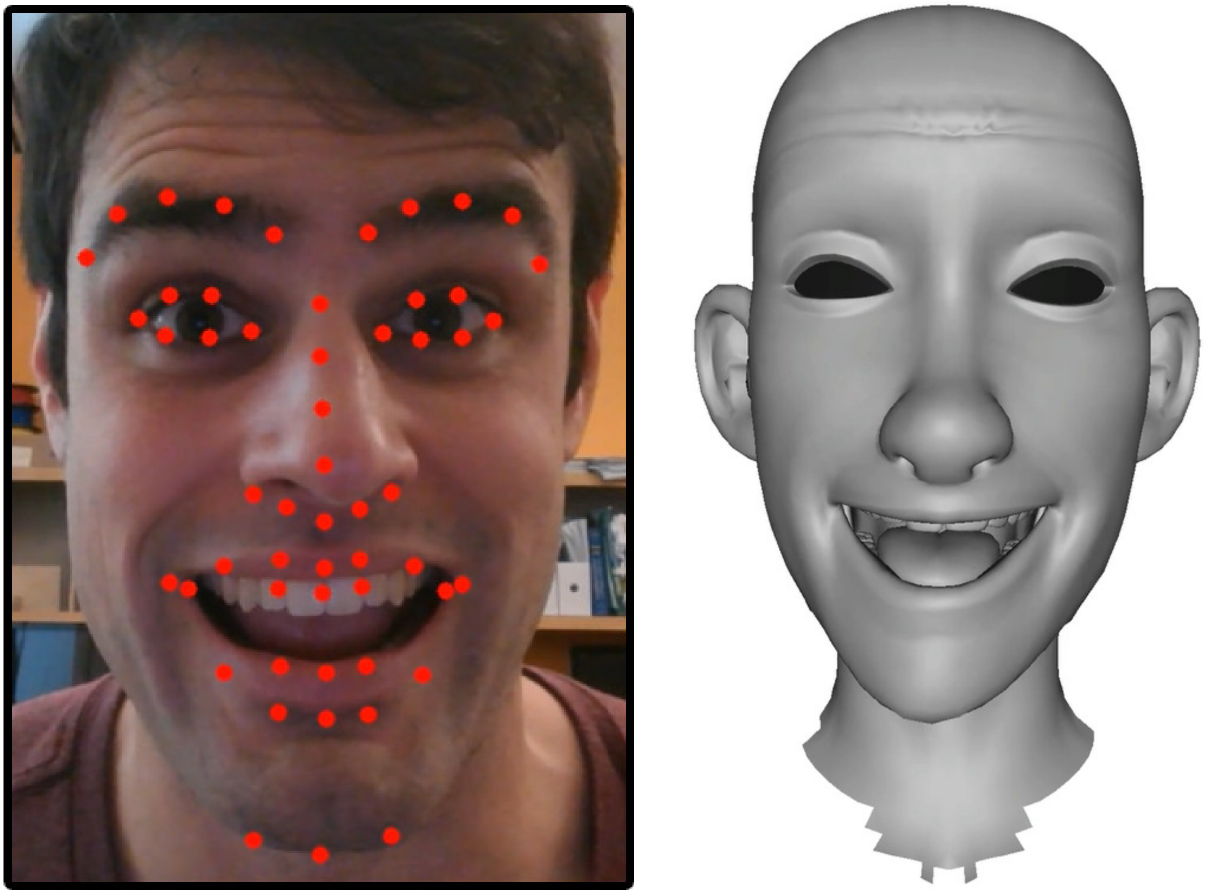


Figure 4.5: Comparison of meshes deformed by rig parameters computed through the IK model. The red dots represent the control points provided to the IK model.[1]

Facial Performance Capture

Real-time monocular facial performance capture systems rely on a differentiable rig to map a video recording to an animation sequence. Zollhöfer et al. [165] provide a survey of current state of the art methods in monocular facial tracking. Because the physical appearance of an actor will not match the appearance of our animated characters, our system animates the character by tracking a sparse set of facial landmark points. To track the facial landmarks on an actor, we use our own implementation of the method described in [35], and we train the model on the same dataset described by the authors. In our facial tracking system, we use 54 out of the 68 landmark points from the dataset. We manually identify the corresponding points on the facial model.

To animate the mesh, we track the movement of the detected landmarks in the recording and use the IK model to estimate the rig parameters required to match the new landmark configuration. Because the facial proportions of the actor might differ from those of the animated character, we track the difference between the actor's expression and the actor's neutral pose. This difference is then applied to the control points for the IK model. Specifically, let $\mathbf{l}^0$ be the detected landmark points on an image of the actor in a neutral expression and let $\mathbf{l}$ be the coordinates of the detected landmarks in the current expression of the actor. The control points $\mathbf{c}$ given to the IK model are then computed as $\mathbf{c} = \mathbf{c}^0 + \mathbf{l} - \mathbf{l}^0$ where $\mathbf{c}_0$ is the control point positions of the mesh in the neutral expression. Figure 4.6 shows a frame from a recording and the resulting deformed mesh from the input.



(a) Input frame

(b) Target mesh

Figure 4.6: Example of our facial performance capture method. The facial landmarks are detected on the input image. The landmark information is passed to the IK model, which computes rig parameter values. The rig parameters are then passed to our approximation model to produce the deformed target mesh.[1]

4.5 Discussion

Because our body approximation model uses a character's skeleton as input, our method can be seamlessly combined with IK by first using IK methods to pose the skeleton then computing the mesh deformations from the resulting bone configurations. However, combining IK with our facial approximation model provides additional complications. The main difference is that the facial deformations do not depend solely on a character's skeleton. Thus, gradients must be computed through the mesh deformations of the facial rig rather than through bone transformations. In the case of complex facial rigs, the computational cost of computing gradients with respect to rig parameters is significantly larger than computing gradients through a skeleton. As previously indicated, gradient computation through our fast facial approximation method is too slow for iterative IK method to evaluate in real-time. To overcome this limitation, we have presented an additional deep-learning model that approximates solutions to the IK problem through fixed-length feed-forward neural network.

In this work, we have presented a facial performance capture method that poses a character based on detected facial landmarks in a recorded video. Prior methods that animate facial rigs through performance capture typically use simple rigs, such as blendshape models [156] or morphable models [12], due to the ease of computing gradients through the rig. However, as demonstrated through our IK solution, the facial approximation model easily provides gradients for more complex facial rigs. Thus, facial capture methods that have previously been limited to differentiable character rigs can now also be applied to production-level characters through our deformation approximation method.

Chapter 5

Repurposing Artist-Created Facial Animation

5.1 Introduction

Feature animation is a labor and time intensive process that creates characters with compelling and unique personalities. Taking one of these characters into an interactive application presents a challenge. The traditional approach is to hand animate large numbers of motion clips which can then be evaluated in a motion graph. This becomes expensive due to the large number of possible actions required. Even a single action can require multiple clips to avoid obvious visual repetition when idling in a specific pose.

In this chapter we repurpose the original hand animated content from a film by using it as a training set which is then used to generate new animation in real time that can retain much of the personality and character traits of the original animation. Due to this choice of training data, we assume that we will have tens of minutes of usable animation.

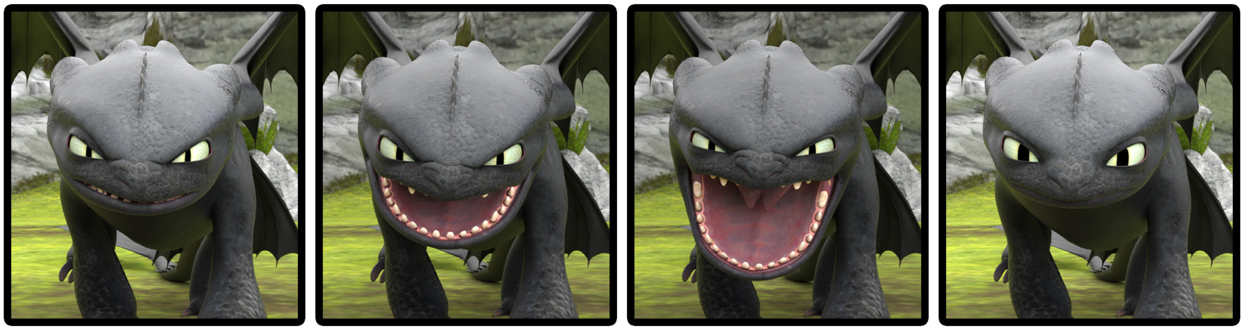


Figure 5.1: Four frames of a synthesized roar animation for Toothless the dragon.[1]

[1] These images are Property of DreamWorks Animation L.L.C., used with permission.

Furthermore, because we use animation for a film-quality character, there is a large number of rig parameters that our synthesis algorithm will need to control. Thus, we use a form of the Gaussian Process Latent Variable Model (GPLVM) to embed the rig parameters of the animation in a lower dimensional space, and we synthesize new animations using this model.

Our work presents a new method to scale the input data to the GPLVM to account for the nonlinear mapping between a character's rig parameters and its evaluated surface mesh. Further, we present a novel method to synthesize new animation using the GPLVM. Our method is based on a particle simulation, and we demonstrate its effectiveness at generating new facial animation for a non-human character. We found that GPLVMs trained with a few homogeneous animations produce visually better results than one trained with many animations of varying types of motions. Our method uses multiple GPLVMs, and we present a novel method to synthesize smooth animations that transition between models. To demonstrate the effectiveness of our work, we develop an interface for our method to receive directions to control the animation in real-time. We developed an interactive application to interface with our method to show that our algorithm can synthesize compelling and expressive animation in real-time.

5.2 Related Work

Statistical methods have been used to analyse and synthesize new motion data [19, 114, 79]. In particular, the Gaussian Process Latent Variable Model (GPLVM) [82] has been used for a number of applications in animation such as satisfying constraints or tracking human motion [44, 146, 149] as well as interactive control [160, 91]. This model is used to reduce the dimension of the motion data and to create a statistical model of the animation. Modifications to the GPLVM have been proposed to make it better suited for modeling motion data. The GPLVM tends to keep far data separated in the reduced dimensional space, but it makes no effort to keep similar data points close together. A number of methods have been proposed to address this limitation. Back constraints [83] have been applied to the GPLVM to preserve local distances. Dynamic models [147, 81] have also been introduced to model the time dependencies in animation data. A connectivity prior [91] has been proposed to ensure a high degree of connectivity among the animation data embedded in the low-dimensional latent space. Prior methods that model animation data with a GPLVM have been applied to full-body motion capture data. In contrast with past work, we apply a similar technique to hand-crafted animation for a film-quality character. One key difference between motion capture data and film-quality hand animation is that the hand animation lies in a significantly higher dimensional space than the motion capture data in terms of the number of parameters needed to specify a pose.

Data-driven approaches to character control and animation synthesis have focused on full-body tasks, which are based on motion graphs [2, 73, 88, 143, 100, 90, 108]. These methods use a graph structure to describe how motion clips from a library can be connected and reordered to accomplish a task. These approaches perform well with large training set;

however, smaller data sets might not be well-suited for motion graphs because a lack of variety and transitions in the motions. Other methods for character control include data-driven and physics-based approaches [30, 111, 91, 136]. All of these methods are applied to full-body human motion or hand motion [1]. The tasks the controllers are trained can be quantifiably measured such as locomotion or reaching tasks. In contrast, we use our method to animate a non-human character's face. Tasks for facial animation are not as easy to quantify, and we therefore develop a novel particle simulation-based method to control facial animation.

Facial animation of non-human characters can be controlled by retargetting recorded expressions. A commonly used method is blendshape mapping [20, 26, 130, 15, 22], which maps expressions from an input model onto corresponding expressions from the target character. Motion is generated by then blending between the different facial shapes of the character. This approach uses an input model such as a video recording of a human to drive the animation of the character. Unlike the blenshape mapping approaches, our method does not control facial animation with recordings of a model. Furthermore, we do not require that the character's face be animated with blendshapes. We make no assumptions about the character's rig, but specifically the face rig we used in our results is animated using a combination of bones, blendshapes, and free-form deformations. Other methods use speech recordings to control the facial animation [96, 152, 32, 18]. Our method does not use video or speech recordings to control the facial animation. Instead we use user interaction with an interactive application as input for our animation synthesis algorithm. Another method for modeling facial expressions allows users to manipulate the face directly and avoids unnatural faces by learning model priors [80].

Animated characters are controlled through an underlying rig, which deforms a surface mesh that defines the character. A variety of methods exist to map a character's rig controls to deformations of the surface mesh [7, 128, 103, 133, 95] as well as the inverse from a skeleton to rig space [54]. Our method makes no assumptions about rig controls and treats mapping from the character rig to the surface mesh as an arbitrary nonlinear function, similar to the assumptions made in [46].

5.3 Overview

Our work computes a low dimensional embedding for a set of training animation and uses the resulting model to generate new animation. The animation data is represented as character rig parameters, which can be evaluated to generate a surface mesh of the character. We make no assumptions about the mapping from rig parameters to the mesh. Because the mapping is typically nonlinear, variation in the rig controls might not necessarily correspond with a similar variation in the surface mesh. We therefore scale each component of the rig parameters based on an approximation of the influence each control has on the mesh.

Next, we embed the scaled rig parameters in a low dimensional space. We first use principal component analysis (PCA) to reduce the data to an intermediate space. We then use then use a form of the GPLVM to further reduce the dimension of the data. Our GPLVM variant keeps similar poses in the animation close in the latent space and keeps temporally close poses near each other as well. For pose synthesis, we compute the maximum a posteriori estimate for the most likely rig parameters given a low-dimensional latent point. We use the learned models to synthesize new animations in real-time. The current pose of a synthesized animation is represented as a particle in the latent space. We apply forces to the particle to push it towards user-defined targets. At each time step in the simulation, we use the current location of the particle in the latent space to generate the next pose in the animation using the GPLVM. We found that this method creates expressive facial animations.

Because we train a separate GPLVM for each type of action, the particle simulation by itself cannot generate animations that transition between models. To overcome this limitation, we compute matching points between the models. These matching points are locations in the latent spaces that map to similar rig parameters. Transitions between models are performed by moving the particle to one of these matching points, switching models, and starting a new simulation at the corresponding matching point in the new model.

5.4 Low Dimensional Embedding

Given a large set of training animation, represented as a sequence of rig control parameters, our method learns a mapping between a low dimensional latent space and rig parameters. This mapping is generated in three stages. First, each rig control in the training animation is scaled to weight the controls proportional to changes in the final mesh. Second, the training animation is reduced linearly using Principal Component Analysis (PCA). Finally, the data is mapped to a lower dimensional latent space using a form of the Gaussian Process Latent Variable Model (GPLVM). After we have found an embedding of the training data in the latent space, we can then map any arbitrary point in the low dimensional space to values for the rig controls.

Scaling Rig Controls

We assume that the character rig parameters $\mathbf{p}$, when evaluated, produces a surface mesh. The i^{th} vertex of this mesh is given by the function $\mathbf{e}_i(\mathbf{p})$. We only assume that the rig evaluation function $\mathbf{e}(\mathbf{p})$ is continuous. Otherwise, we make no other assumptions about the function to keep our method as general as possible. Thus, the evaluation function will typically be highly nonlinear.

Depending on how the evaluation function $\mathbf{e}(\mathbf{p})$ is defined, large changes in some rig parameters might result in small changes in the output surface mesh while small changes for other parameters might result in large changes in the mesh. Specifically for some setting of the rig parameters $\mathbf{p}$, the value $\left\|\frac{\partial \mathbf{e}(\mathbf{p})}{\partial p_i}\right\|$ might be large for the i^{th} rig parameter, but the value $\left\|\frac{\partial \mathbf{e}(\mathbf{p})}{\partial p_j}\right\|$ might be small for some other rig control. Thus, there could exist some rig controls that have a very small effect on the surface mesh but have a large variance across the training animation. Because we will be using PCA, we want to scale each component of the data so that the principal axes of the transformation do not align with these controls with high variance but low influence on the mesh.

To avoid this situation, we want to scale the rig parameters about the sample average to obtain $\mathbf{z} = \mathbf{W}(\mathbf{p} - \bar{\mathbf{p}}) + \bar{\mathbf{p}}$ where $\mathbf{W}$ is a diagonal matrix and w_i is the amount to scale the i^{th} rig parameter. We choose $\mathbf{W}$ such that a unit change in the scaled rig parameter space corresponds with approximately a unit change in the surface mesh. Specifically for the i^{th} rig parameter,

$$\left\| \frac{\partial}{\partial z_i} \mathbf{e}(\mathbf{W}^{-1}(\mathbf{z} - \bar{\mathbf{p}}) + \bar{\mathbf{p}}) \right\| = 1 \tag{5.1}$$

where $\mathbf{z}$ is any possible value of the scaled rig parameters.

We use $\mathbf{p} = \mathbf{W}^{-1}\mathbf{z}$ and the chain rule to find that

$$\left\| \frac{\partial \mathbf{e}(\mathbf{p})}{\partial p_i} \frac{\partial}{\partial z_i} \left[w_i^{-1}(z_i - \bar{p}_i) + \bar{p}_i \right] \right\| = 1. \tag{5.2}$$

We can use Equation 5.2 to solve for the weights and find that $w_i = \left\| \frac{\partial \mathbf{e}(\mathbf{p})}{\partial p_i} \right\|$. Because $\mathbf{e}(\mathbf{p})$ is a generally nonlinear function, Equation 5.2 cannot be satisfied for all possible values of $\mathbf{p}$ for a fixed $\mathbf{W}$. Instead, we approximate the norm of the partial derivative by evaluating the rig at the sample mean $\bar{\mathbf{p}}$ of the training data and at several points about the mean. For rig parameter i, we construct a least squares error problem to approximate the norm of the partial derivative by

$$\left\| \frac{\partial \mathbf{e}(\mathbf{p})}{\partial p_i} \right\| \approx \arg\min_{w} \sum_{n=-2}^{2} \left(\| \mathbf{e}(\bar{\mathbf{p}}) - \mathbf{e}(\bar{\mathbf{p}} + n\boldsymbol{\sigma}_i) \| - w \, \| n\boldsymbol{\sigma}_i \| \right)^2 \tag{5.3}$$

where $\boldsymbol{\sigma}_i$ is a vector with the sample standard deviation of the i^{th} rig parameter in the i^{th} position and zeros elsewhere. The values $n \in \{-2, -1, 0, 1, 2\}$ were chosen experimentally, and this set was found to produce good results. We solve this least squares problem separately for each w_i.

Linear Dimensionality Reduction

Typically, a fully-rigged main character for a feature film will have on the order of thousands of rig controls. Some of these rig controls might not be used in the training data, and some might have a small, almost imperceptible effect on the animation. To remove these controls and simplify the data, we linearly reduce the dimension of the data by using Principal Component Analysis. This method will treat the small variations in the data as noise and remove it. This initial linear reduction helps improve the results of the GPLVM that is used later.

Let $\mathbf{z}$ represent the scaled rig parameters of a single frame of animation. Suppose that there are D_{rig} parameters and that there are N total number of frames of animation in the training set. The scaled animation data can be represented as $\mathbf{Z} = [\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3, ..., \mathbf{z}_N]$. We then compute the singular value decomposition of the data $\bar{\mathbf{Z}} = \mathbf{U}\Sigma\mathbf{V}^T$ where the matrix $\bar{\mathbf{Z}}$ is the matrix $\mathbf{Z}$ with the sample mean subtracted from each column of the matrix. We choose the number of principal components d_{pca} to use by considering the explained variance of the model. The explained variance is given by $v(d) = \sum_{i=1}^{d} \sigma_i^2 / \sum_{i=1}^{k} \sigma_i^2$, where σ_i^2 is the i^{th} singular value of the normalized matrix $\bar{\mathbf{Z}}$ and k is the rank of the matrix. In our experiments for our models, we chose d_{pca} such that $v(d_{pca}) \approx 0.85$. With the number of principal components chosen, we define the transformation matrix $\mathbf{T}_{pca}$, which contains the first d_{pca} columns of the matrix $\mathbf{U}$. We then represent the training data as the matrix $\mathbf{Y} = \mathbf{T}_{pca}^T \bar{\mathbf{Z}}$.

We evaluated the difference between running PCA on the original and scaled rig parameters to determine the effect scaling the parameters has on the quality of the dimensionality reduction. We found that when enough principal components are used to ensure that the explained variance is at or above 85%, there is no discernible difference quality of the animations between the scaled and original rig parameters, but the GPLVMs described in the following section tended to perform better with the scaled rig parameters. The difference between the original rig parameters and the compressed data, measured as $\left\|\mathbf{z} - \mathbf{T}_{pca}\mathbf{T}_{pca}^T\mathbf{z}\right\|$, is much larger when using the scaled rig parameters compared to the unscaled parameters. When we use a small number of principal components, animations compressed with the scaled rig parameters are visually better than the animations compressed with the unscaled data. Furthermore, the unscaled version often contains objectively undesirable meshes, such as the jaw of a character passing through the roof of its mouth. Therefore, we conclude that quantitative comparisons in the rig parameter space will not be sufficient to evaluate the effectiveness of our method.

Nonlinear Dimensionality Reduction

Given the linearly reduced data in the matrix $\mathbf{Y}$, we now compute a low-dimensional embedding through the use of a Gaussian Process Latent Variable Model [82]. The GPLVM is a generative, probabilistic model that we use to map nonlinearly the PCA transformed data $\mathbf{Y}$ to a set of points $\mathbf{X}$ in a latent space of dimension d_{gplvm} where $d_{gplvm} < d_{pca}$. We model dynamics in the latent space by placing a Gaussian process prior on the points $\mathbf{X}$ as described in [81]. This dynamics prior will thus keep temporally close data points close together spatially. Because we train our models using multiple segments of animation, the GPLVM with a dynamics prior will tend to keep separate segments far apart in the latent space. This separation is caused by the GPLVM placing dissimilar frames of animation far apart without trying to place similar frames near each other. Therefore, we use the connectivity prior described in [91] in order to pull together similar frames of animation from separate segments.

The GPLVM models the training data $\mathbf{Y}$ as the outputs of a Gaussian process from the low dimensional embedding of the points $\mathbf{X}$. We assume that each output of the GP is independent so that

$$\begin{aligned}\log p(\mathbf{Y}|\mathbf{X}) &= \sum_{i=1}^{d_{pca}} \log N(\mathbf{y}_{i,:}|\mathbf{0}, \mathbf{K}_x) \\ &= -\frac{d_{pca}}{2}|\mathbf{K}_x| - \frac{1}{2}\operatorname{tr}\left(\mathbf{K}_x^{-1}\mathbf{Y}\mathbf{Y}^T\right) + \text{const.}\end{aligned} \tag{5.4}$$

We denote the i^{th} row of $\mathbf{Y}$ as $\mathbf{y}_{i,:}$. For the entries in the kernel matrix $\mathbf{K}_x$, we use the radial basis function, which is given by:

$$k_X(\mathbf{x}_i, \mathbf{x}_j) = \sigma_{rbf}^2 \exp\left(-\frac{1}{2l_x^2}\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2\right) + \delta_{ij}\sigma_{white}^2. \tag{5.5}$$

The kernel parameters σ_{rbf}^2, σ_{white}^2, and l^2 are optimized when the GPLVM is trained.

Our input data is composed of multiple segments of animation, and we would like to model the dynamics of each segment. We place a Gaussian process prior on the latent points $\mathbf{X}$. The input to the GP is time $\mathbf{t}$ of each frame. Each segment of animation is independent from all others; thus, the prior places a Gaussian process on each segment separately. The dynamics prior is given by

$$\psi_D(\mathbf{X}, \mathbf{t}) = \sum_{i=1}^{d_{gplvm}} \log N(\mathbf{X}_{\mathbf{i},:}|\mathbf{0}, \mathbf{K}_{\mathbf{t}}). \tag{5.6}$$

The entries of the kernel matrix $\mathbf{K}_t$ are computed by the radial basis function. Furthermore, $K_t^{ij} = 0$ when frames i and j belong to separate animation segments. See the description of the simple hierarchical model in [81] for more details.

The connectivity prior provides a method to model the degree of connectivity among the latent points $\mathbf{X}$ by using graph diffusion kernels. We denote this prior with $\psi_C(\mathbf{X})$. See the description of the connectivity prior in [91] for more details.

Combining the dynamics and connectivity priors, we can express the conditional probability of $\mathbf{X}$ as $p(\mathbf{X}|\mathbf{t}) \propto \exp\psi_D(\mathbf{X}, \mathbf{t})\exp\psi_C(\mathbf{X})$. We estimate the latent points $\mathbf{X}$ and the hyper-parameters σ_{rbf}, σ_{white}, and l_x through maximum a posteriori (MAP) estimation. Thus, we want to maximize

$$\log p(\mathbf{X}, \sigma_{rbf}, \sigma_{white}, l_x|\mathbf{Y}, \mathbf{t}) = \log p(\mathbf{Y}|\mathbf{X}) + \psi_D(\mathbf{X}, \mathbf{t}) + \psi_C(\mathbf{X}). \tag{5.7}$$

To maximize Equation (5.7), we use scaled conjugate gradient. The initial guess for the latent points is the first d_{gplvm} rows of $\mathbf{Y}$. We manually set the hyper-parameters for the dynamics prior and do not optimize these values. In Figure 5.2, we show a plot of several animation curves embedded in a three dimensional latent space.

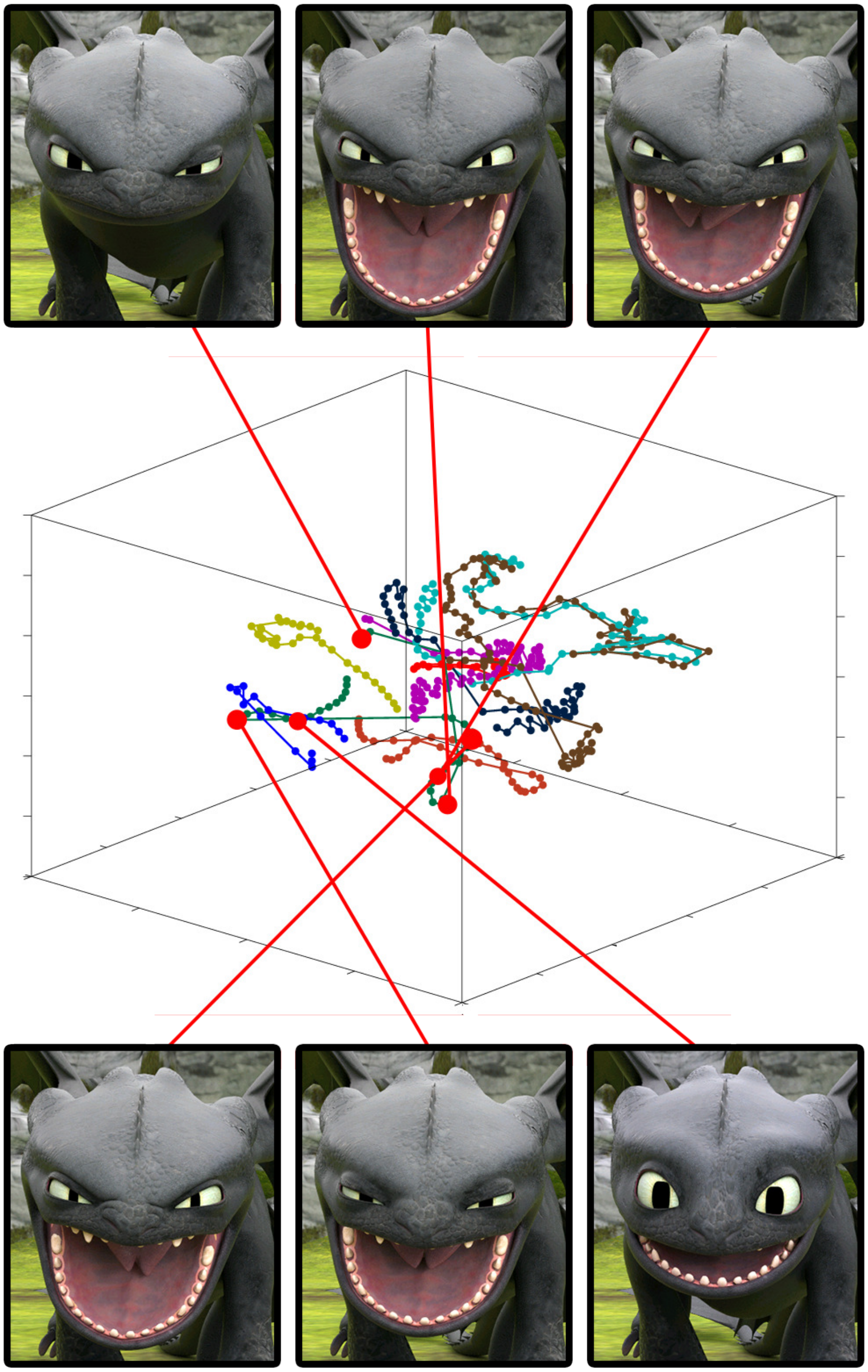


Figure 5.2: Three dimensional latent space learned for a training set of 9 examples of a roar with a total of 393 frames.[1]

Mapping to Rig Controls

Once we have a trained a model, we are now able to reconstruct rig control values from a new point $\mathbf{x}'$ in the latent space. We first find the most likely point in the d_{pca} dimensional space given the new point and the GPLVM model. Next, we multiply by the matrix of principal components to obtain the scaled rig parameters. Finally, we divide by the scaling factors and add the mean to each parameter.

The distribution of a new point $\mathbf{y}$ given the corresponding latent point $\mathbf{x}$ and the GPLVM model M is a Gaussian distribution where

$$p(\mathbf{y}|\mathbf{x}, M) = N(\mathbf{y}|\mathbf{Y}\mathbf{K}_x^{-1}\mathbf{k}_x(\mathbf{x}), k_x(\mathbf{x}, \mathbf{x}) - \mathbf{k}_x(\mathbf{x})^T\mathbf{K}_x\mathbf{k}_x(\mathbf{x})) \tag{5.8}$$

where $\mathbf{k}_x(\mathbf{x})$ is a column vector whose i^{th} entry is given by $\mathbf{k}_x(\mathbf{x})_i = k_x(\mathbf{x}_i, \mathbf{x})$. Because the distribution is Gaussian, the most likely point in the d_{pca} dimensional space is given by the mean $\mathbf{Y}\mathbf{K}_x^{-1}\mathbf{k}_x(\mathbf{x})$. The product $\mathbf{Y}\mathbf{K}_x^{-1}$ can be precomputed, which would allow this pose reconstruction problem to run in time linear to the size of the training data for the model.

5.5 Animation Synthesis in Latent Space

New animations can be synthesized by generating a new path $\mathbf{P} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_t]$ through the latent space. The rig parameters for each point in the path can be computed by mapping the point from the latent space to the high dimensional rig control space. Because the latent space provides a continuous mapping any smooth curve in this low-dimensional space will result in smooth animation curves for each rig parameter.

To synthesize a new path, we simulate a particle moving through the latent space and track its position over time. We control the particle using a Lagrange multiplier method to enforce constraints on the system. For example, if we desire a path that does not stray too far from a user-defined point, we define a constraint to enforce this behavior. To add variations and noise to the path, we apply a random force. We found that this particle simulation method works well for synthesizing facial animations.

In order to achieve real-time performance, the number of training points in the GPLVM must be small. Therefore, the training animation needs to be divided into sufficiently small subsets. Each subset of animation corresponds with a specific type of expression or facial action such as a roar. A separate GPLVM is trained on each subset of animation. Because these latent spaces are separate, we need a method to map points from one model to another. With such a mapping, the particle simulation can transition between models, which allows for the synthesis of facial animations across multiple subsets of the animation.

We conclude this sections with a descriptions of a set of low-level "commands" to provide control of the synthesized animation. These commands are used to control the particle in the latent space, which thus gives control of the synthesized animation. The motivation for these commands is to develop a system reminiscent of the method an artist might use to plan an animation of a character's face. These commands allow for a user or an application to specify key poses in time, and our animation synthesizer generates motion that transitions between the poses.

Particle Simulation

We synthesize curves in the latent space by tracking the position of a particle in this space over time.

The input to our simulation is a path $\mathbf{p}(t)$ that the particle follows through time. We apply two constraints to the system and a "random" force to add noise to the path. The first constraint ensures that the particle does not move too far from the path. The second constraint ensures that the particle remains in areas of high probability in the GPLVM. Because there could be times when both constraints cannot be satisfied simultaneously, we model the path-following constraint as a hard constraint that must be satisfied, and the other constraint is modeled as a soft constraint that can be violated.

Given some path $\mathbf{p}(t)$ parametrized by time, we want to ensure that the particle does not drift too far away from the curve. To enforce this requirement, we apply the inequality constraint $\|\mathbf{x} - \mathbf{p}(t)\|^2 - r^2 \leq 0$ to ensure that the particle at location $\mathbf{x}$ stays within a distance r of the point $\mathbf{p}(t)$ at time t. Forward simulation with this constraint is computed using the Lagrange multiplier method described in [8].

Let $\mathbf{F}$ be the force acting on the particle at time t. We use the Lagrange multiplier method to compute an additional force $\mathbf{F}_c$ that we apply to the particle to ensure that the constraint is satisfied. The constraint force is given by $\mathbf{F}_c = \lambda \mathbf{g}$ where $\mathbf{g} = \mathbf{x}(t) - \mathbf{p}(t)$. The multiplier λ for a particle of unit mass is given by

$$\lambda = \frac{-\mathbf{g}^T\mathbf{F} + G}{\mathbf{g}^T\mathbf{g}}. \tag{5.9}$$

The scalar G is given by

$$G = (\dot{\mathbf{x}}(t) - \dot{\mathbf{p}}(t))^T(\dot{\mathbf{x}}(t) - \dot{\mathbf{p}}(t)) + 2\alpha(\mathbf{g}^T\dot{\mathbf{x}}(t) - \mathbf{g}^T\dot{\mathbf{p}}(t)) + \frac{1}{2}\beta^2(\mathbf{g}^T\mathbf{g} - r^2). \tag{5.10}$$

The parameters α and β are selected by the user to control how quickly a system violating the constraints returns to a state satisfying them. We set $\beta = \alpha^2$, which is suggested in [8]. The term $\mathbf{F}_c$ described above will apply a force to satisfy the equality constraint $\|\mathbf{x}(t) - \mathbf{p}(t)\|^2 - r^2 = 0$. To allow the particle to move freely within the radius around the target point, we constrain the force $\mathbf{F}_c$ to only point towards the target point $\mathbf{p}(t)$. This is accomplished by setting $\lambda = 0$ whenever $\lambda > 0$.

Our second constraint pushes the particle towards high probability regions in the latent space. The GPLVM provides a probability distribution over the latent space $p(\mathbf{x}(t)|M)$, and we use this distribution to push the particle towards "probable" regions, which can provide better reconstructed poses than less probable regions of the latent space. However, we found that models trained with facial animations can synthesize reasonable poses from less likely regions of the latent space. We found that generally these lower probability poses do not contain visual defects such as an overly stretched face or interpenetrating meshes. Therefore, keeping the particle in a high probability region is not critical and can be violated if necessary to satisfy the path constraint. We model this likelihood constraint as a force applied to the particle that points in the direction of the gradient of the PDF. The magnitude of the force is determined by the value of the PDF evaluated at the particle's current location. If the value is above some empirically chosen quantity v, the magnitude is small, and if the value is below v, the magnitude is large. We model this as a sigmoid function so that the force function is continuous for numerical integration. The magnitude is expressed as

$$S(t) = a\left(1 + \exp\left(\frac{p(\mathbf{x}(t)|M) - v}{l}\right)\right)^{-1}, \tag{5.11}$$

and the constraint force is expressed as

$$\mathbf{F}_{GPLVM}(t) = S(t)\frac{\partial p(\mathbf{x}(t)|M)}{\partial \mathbf{x}} \bigg/ \left\|\frac{\partial p(\mathbf{x}(t)|M)}{\partial \mathbf{x}}\right\|. \tag{5.12}$$

The parameters a and l are defined by the user, and control the magnitude of the force when the constraint is not satisfied and how quickly the magnitude approaches a. Computing the partial derivatives of the Gaussian process takes time quadratic to the size of the training data. If the size of the training set is small, this can be computed in real-time.

In addition to these constraint forces, we apply a random force $\mathbf{F}_{rand}(t)$ to add variation to the particle's path. We model this force as a randomly drawn, zero-mean Gaussian process: $\mathbf{F}_{rand}(t) \sim \mathcal{GP}(0, k(t,t'))$. Each component of $\mathbf{F}_{rand}(t)$ is independent of all others. The covariance function is given by $k(t,t') = \alpha \exp\left(-(2\gamma)^{-1}(t-t')^2\right)$, where α and γ are user-defined parameters that control the magnitude and smoothness of the random force.

This random force adds noise and variations to the particle's movement through the latent space. Thus, a particle following the same path multiple times will have slight variations in each repetition, which will generate unique animations with small but noticeable differences. Variations in the animation could be achieved through other means such as perturbing the path $\mathbf{p}(t)$; however, we did not evaluate these other possibilities.

In our experiments, we simulate the particle forward in time using a fourth-order Runge-Kutta integration method. We used a piecewise linear function for the path $\mathbf{p}(t)$, which is defined by a set of points $[\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n]$ such that $\mathbf{p}(t_i) = \mathbf{p}_i$ and t_i is the time of the i^{th} frame of animation. We do not integrate across multiple frames of animation to avoid integrating over discontinuities in the piecewise path function $\mathbf{p}(t)$. Section 5.5 describes methods to define $\mathbf{p}(t)$.

Mapping Between Models

A large set of heterogeneous motions cannot be accurately embedded in a low dimensional ($d \leq 5$) latent space. Therefore, we divide the training animation into small sets of similar expressions and compute the embedding in the latent space for each subset separately. The drawback of training separate models is that animations transitioning between multiple models cannot be synthesized using our particle simulation method. This problem arises because a continuous path between models does not exist. In this section, we describe a method to synthesize smooth animations that transition between latent spaces.

To create a path between two models M_1 and M_2, we first precompute a set S of corresponding points in both latent spaces. A pair of matching points $(\mathbf{x}_1, \mathbf{x}_2)$ where $\mathbf{x}_1 \in M_1$ and $\mathbf{x}_2 \in M_2$ is included in S if $\|\mathbf{g}(\mathbf{x}_1; M_1) - \mathbf{g}(\mathbf{x}_2; M_2)\|^2 < \epsilon$ where $\mathbf{g}(\mathbf{x}; M)$ is the function that maps $\mathbf{x}$ to the rig parameter space. Thus, we want to identify pairs of points in the latent spaces whose reconstructed poses are similar. The set of matching points identifies points in the two models, which can be used as bridges between the two models. To create a curve that moves between model M_1 to M_2, we create a path in M_1 that ends at a point in S for the model and then create a path that starts at the matching point in M_2.

To identify a pair of matching points for models M_1 and M_2, we fix a point $\mathbf{x}_1 \in M_1$ and compute the reconstructed rig parameters $\mathbf{z}_1 = \mathbf{g}(x_1; M_1)$. The point $\mathbf{x}_1$ can be any point; however, in our implementation, we restricted $\mathbf{x}_1$ to be from the set of latent points corresponding to the training animation for the model. Next, the point $\mathbf{z}_1$ is transformed by the linear dimensionality reduction specified by model M_2

$$\hat{\mathbf{y}}_1 = \mathbf{T}_2^T[\mathbf{W}_2(\mathbf{z}_1 - \mathbf{m}_2)] \tag{5.13}$$

where $\mathbf{T}_2$ is the first d principal components of the PCA transformation given in model M_2, $\mathbf{W}_2$ is the diagonal matrix of scale values for each component, and $\mathbf{m}_2$ is the mean of the training data used in model M_2.

The next step is to find the point $\mathbf{x}_2$ in the latent space of model M_2 such that

$$\mathbf{x}_2 = \arg\min_{\mathbf{x}} \left\| \hat{\mathbf{y}}_1 - \operatorname*{argmax}_{\mathbf{y}} \log p(\mathbf{y}|\mathbf{x}, M_2) \right\|^2. \tag{5.14}$$

Because $y_i = f(\mathbf{x}) + \epsilon$ where ϵ is additive Gaussian white noise, the maximum of $p(\mathbf{y}|\mathbf{x}, M_2)$ occurs when $\mathbf{y} = \mathbf{f}_*$ where $f_* = \mathbf{K}_*[\mathbf{K}_x]^{-1}\mathbf{Y_2}$ is the noise-free output for the test point $\mathbf{x}$. Therefore, Equation (5.14) can be written as

$$\mathbf{x}_2 = \arg\min_{\mathbf{x}} \left\| \hat{\mathbf{y}}_1 - \mathbf{K}_*[\mathbf{K}_x]^{-1}\mathbf{Y_2} \right\|^2. \tag{5.15}$$

The problem of finding the best matching $\mathbf{x}_2 \in M_2$ giving the point $\mathbf{x}_1 \in M_1$ is now formulated as a nonlinear optimization problem. We solve this problem by using the scaled conjugate gradient algorithm. However, because the function is multi-modal, we run the optimization algorithm multiple times with randomly selected initial values to attempt to find the global minimizer. Furthermore, care needs to be taken not to take large steps during the optimization routine because the gradient of the objective function quickly goes to zero as $\mathbf{x}_2$ moves away from the training points in the model.

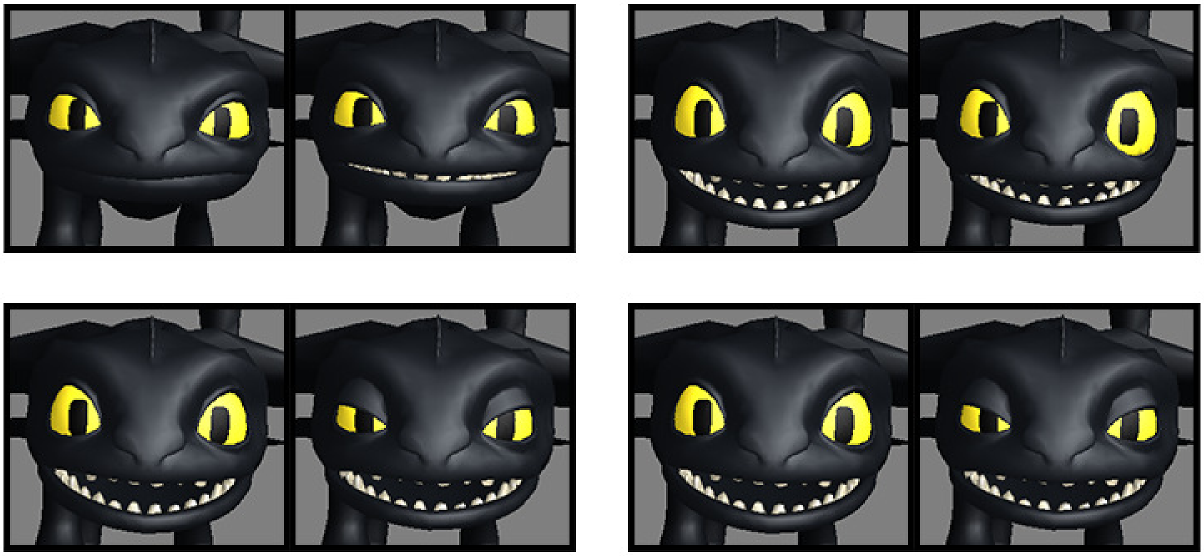


Figure 5.3: Four examples of the best-matching poses found between two models. In each pair, the pose on the left is generated from a model trained on animations with grumpy-looking animations, and the pose on the right is generated from happy-looking animations.[1]

In our implementation, we identified pairs of matching points between models M_1 and M_2 by computing matching points $\mathbf{x}_2$ for each latent point of the training data for model M_1. We then evaluated the Euclidean distance between the reconstructed rig space poses for each pair of matching points. Pairs with distances below some user-defined threshold were kept while all other pairs were discarded. With this method, we obtained between 10-50 transition points between each pair of models. For models trained on similar-looking animations, the transition points were spread throughout the latent space. Models trained with distinct animations tended to have the transition points clustered around one or two small regions of the latent space.

To create an animation that transitions between two models, we generate a curve in the first model that ends at one of the precomputed transition points and a curve in the second model that starts at the corresponding transition point from the first model. The animation is synthesized by reconstructing the poses along the curves and placing the animation from the second model right after the first. As seen in Figure 5.3, the poses reconstructed from matching latent points in two models are not necessarily identical. As a result, there will be a discontinuity in the animation at the transition between the two models. To overcome this problem, we perform a short blend between the two poses in the rig parameter space at the transition point.

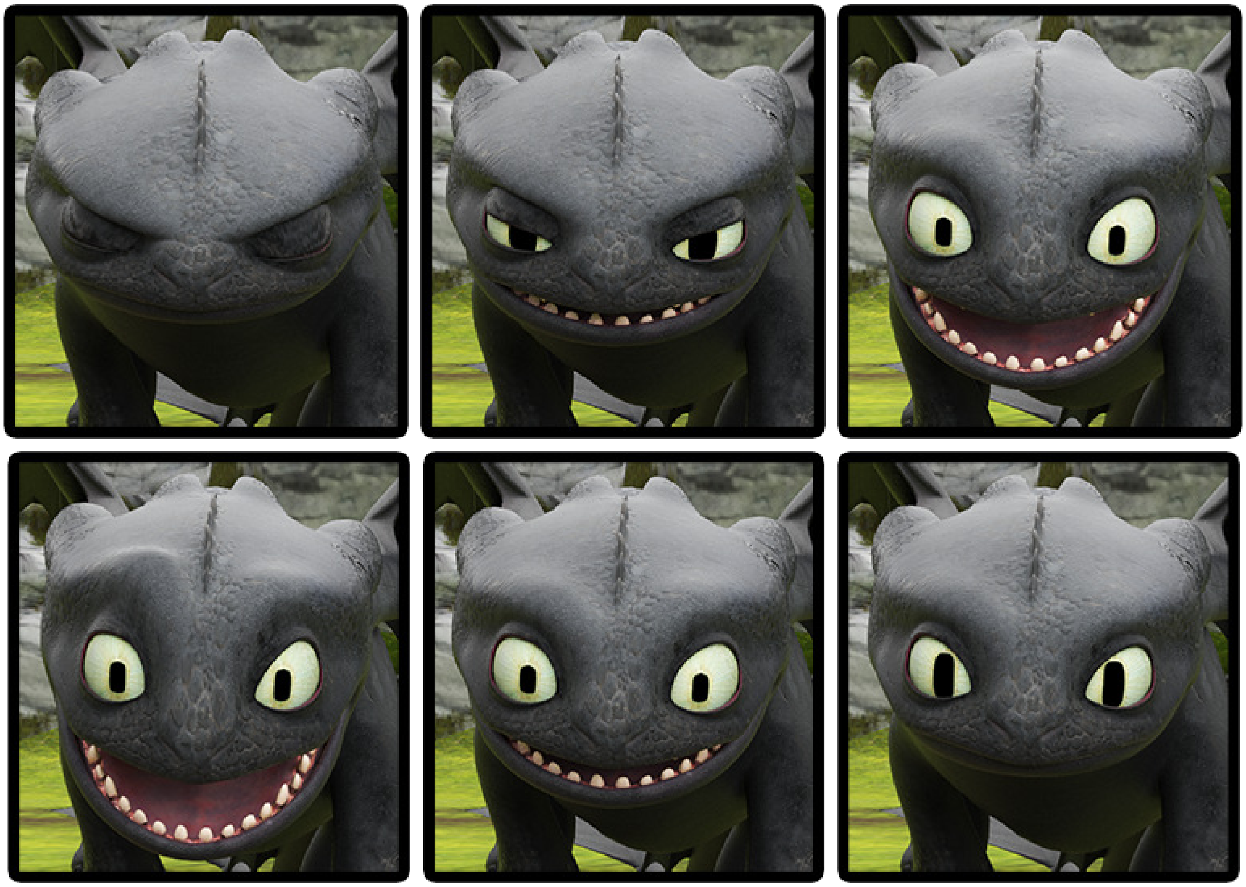


Figure 5.4: Set of frames from an animation synthesized using a model trained on a set of ”surprise” expressions.[1]

Synthesis Control

We use the particle simulation method described above to synthesize animation for the face of a non-human character and develop a set of commands to provide intuitive control of the character’s expression. The high-level reasoning for using these commands is that we want to provide control over what pose the character has at a specific time in an animation. With these poses, our synthesis algorithm then generates transitions between the poses and models specified in the commands.

MOVE: The move command takes a target point $\mathbf{t}$ in the latent space as input. The synthesized animation is controlled by moving the particle from its current position in the latent space to the target point. This is accomplished by setting the particle’s path function $\mathbf{p}(t)$. We tested two methods to generate the path. The first method creates a straight line from the current point to the target. The second method uses the shortest path in a complete weighted graph G of the training data. In the graph, we represent each frame of data as a vertex, and the weights between vertices are computed by $w(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|^{-p}$, which is similar to the graph constructed for the connectivity prior [91]. In our implementation, we found that setting $p = 4$ yielded good results. We also add the start and end points as vertices in the graph G. We re-sample the resulting path so that $\left\|\frac{\partial \mathbf{p}(t)}{\partial t}\right\|$ is constant for all t. This ensures that the particle follows the path at a consistent speed. We found that both path-generating methods create compelling animation. The only difference between the two is that the straight line path is shorter, and thus a particle following this path will reach the target in less time.

IDLE: When the animated character is not performing an action, we would like for the character to have an "idling" animation, and we would like to control the expression of the character as it idles. We found that we can synthesize idling animations by picking a point $\mathbf{p}$ in the latent space corresponding with a user-specified pose. This pose is a hand-selected expression. We let the particle move randomly within a radius r about the point to create variations of that pose. Keeping the particle within the radius is accomplished by setting the particle's path following function to $\mathbf{p}(t) = \mathbf{p}$ for the time we want idle about the point. To add variety to the animation, multiple user-specified points can be used. With multiple points, the synthesis can be controlled by first picking a point from the set to move to. Next, the particle hovers about that point for a fixed amount of time. Finally, an new point is selected, and the simulation repeats by moving to this new point and hovering. See the accompanying video[2] for examples of synthesized idling animations.

TRANSITION: The transition command is used to generate a continuous animation between two models. This command uses the previously described MOVE and IDLE commands. To transition from model M_1 to model M_2, our method moves the particle from its current position in model M_1 to the nearest precomputed matching point in the latent space. When the particle is close to the point, it then idles about that point and the particle in M_2 also begins to idle about the corresponding matching point. We finish the transition by performing a blend between the high-dimensional rig parameters from the two models while the particles are idling. Please see the video[2] for examples of transitions.

PLAY SEGMENT: Occasionally, we might want to play part of an animation unmodified directly from the training set. We play the animation by using the embedding of the sequence in the latent space. We use the MOVE command to position the particle near the starting pose of the animation. When the particle is close enough, we stop the simulation and move the particle along the path of the embedded animation. When moving the particle to the start, we adjust the radius r to ensure that it has moved close to the start to avoid discontinuities when the animation segment starts playing.

5.6 Results

We used the method described above to synthesize animations at interactive frame rates. The input to our algorithm is film-quality hand animation. For a feature film, a main character might have about 20 minutes of animation. We manually separated the data into sets of

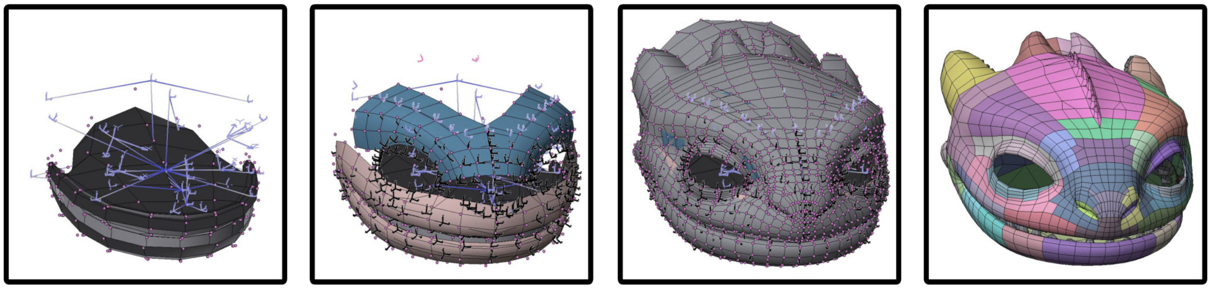


Figure 5.5: A visualization of the layered Deformation System for Toothless's facial rig that enables real time free-form facial control shaping.[1]

similar expressions and also removed any visually bad data. For example, a character might be off screen and is not animated, or a character might be animated for one specific camera angle and does not look acceptable from all possible viewing angles. Using our method, we trained a separate model for each type of expression that we manually labeled in the training data. To evaluate the effectiveness of our method, we compared transitions synthesized with our method to transitions generated using Motion Graphs [73]. Additionally, we synthesized scripted animations off-line and created an interactive game featuring synthesized real-time animation using our algorithm to demonstrate the application of our method.

We used the animation data from the hero dragon character Toothless in the feature film *How to Train Your Dragon 2*. This data is sampled at 24 FPS, and 742 face rig controls are used in our algorithm. Toothless's facial rig is a multi-layered design [120], which provides control ranging from coarse to fine deformations. Figure 5.5 shows the layers of the face rig. There are four main layers of the face rig that involve both bones and blenshapes. First, the bones control large, gross deformations of the mesh. Second, intermediate blendshapes are applied for coarse control. Third, fine-control blendshapes are used. Finally, free-form deformations are applied to allow custom shapes after the first three layers have been evaluated.

To demonstrate how well our method can reuse previous animation, we use only data from this film and do not hand animate any data specific for our applications. We identified eight expression sets: happy, grumpy, bored, curious, and neutral, roar, head shake, and surprise. We manually labeled animations that fit into these categories and trained a GPLVM on each one separately. The labeling task required several hours to complete. Each model contained between 100 to 800 frames of animation, and the latent space for each model has three dimensions. We chose three dimensions experimentally by training models with different dimensions. We found that for our small data sets, the quality of animations synthesized with models of three dimensions or higher were perceptually similar. Therefore, we chose the smallest dimension to minimize the number of unknown variables we solve for when training the GPLVM. In total, we included 3745 usable frames of animation in our training data, which is equivalent to 156 seconds of animation.

Because our method solves a problem similar to Motion Graphs and methods based on Motion Graphs, we compare expression transitions synthesized with our method to those we synthesized using Motion Graphs described in [73]. In our method, we used on average 12 frames to blend between two models. Therefore, we used the same number of frames to synthesize the blends between segments of animation using Motion Graphs for comparison. In the accompanying video[2], we show transitions synthesized using both methods. For Motion Graphs, we picked transitions between two sets of animation by picking transitions points between animation sequences with small distances in the rig parameter space as described in their work. Visually, we found that in some cases, transitions synthesized using Motion Graphs appear sudden and unnatural. We found that these sudden transitions occur when the two animations do not contain large movements. However, Motion Graph blends are not noticeable when transitioning between motions containing large movements. Our method, on the other hand is able to synthesize smooth transitions between different expressions regardless of the amount of motion before and after the transition.

We found that because our sets of training animation are small and contain heterogeneous motions, the Motion Graph algorithm was unable to find transitions with small distances going towards or away from most animation segments. Thus, a motion graph built on this data would use a small fraction of the data. Our method, however, makes use of the entire data set and is capable of transitioning to and from any pose.

We also evaluate our method by synthesizing scripted animations. We directly used our interface for the synthesis algorithm. We provided control over which command is sent to the system and when. This give the user the ability to specify poses that the character needs to make at a scripted time. Because the animation can be computed in real-time, the user can quickly see how changes in the script affect the animation. All of the off-line animations shown in our accompanying video are synthesized with this method. We found that scripting an animation allows for someone without an artistic background to author novel and expressive animations quickly.

We demonstrate the effectiveness of our algorithm through an interactive game of Tic-Tac-Toe, in which the user plays against the computer. We synthesize animation for Toothless's face to react in real time with the results of the game. During Toothless's turn, he holds a ponderous expression. Although the computer logic for Tic-Tac-Toe strategy can be computed in milliseconds, we intentionally extend Toothless's deliberation time to allow for expressions as if he were playing a cognitively difficult game. During the player's turn, he squints and scowls as if he were intimidating the player. When Toothless loses a round in the game, he roars and expresses anger, and when the he wins, he expresses happiness. If Toothless misses a move to block the player from winning, he displays an expression of surprise. All of these expressions are scripted using commands described in Section 5.5.

We found that eye movement is context specific. Because synthesizing new animation with eye movement lead to unrealistic animation, we fixed the eyes to look forward and do

[2] `http://graphics.berkeley.edu/papers/Bailey-RHA-2016-07/Bailey-RHA-2016-07.mp4`

not include the eyes' rig parameters in the synthesis model.

For each emotional state animated in the game, we created a set of scripts containing specific commands. When the game needed to synthesize a particular emotional expression, it randomly picked a script from the corresponding set to run. Only the head shaking animation was scripted using the PLAY command. All other animations are scripted using TRANSITION, MOVE, and IDLE.

We tested our application on an HP Z840 workstation with two Intel Xeon E5-2687w processors running at 3.1GHz, providing 16 cores in total. The machine has 32GB RAM. To compute the surface meshes, we use LibEE [154], a multithreaded evaluation engine for calculating Toothless's surface mesh.

To achieve interactive frame rates for rig evaluation, the resolution of Toothless's final skin mesh was reduced by a factor of 5. This was done non-uniformly to ensure resolution was retained in the most critical areas for expression, e.g. eyes and wrinkles around the nose. Apart from the mesh resolution reduction, no other changes were made to the face rig compared with the original production rig used in the film. LibEE is also the same engine used to evaluate the rig during the production of the film; therefore, the animation and deformations are all the same as used in production. We render the mesh for the real-time application using OpenGL. The application runs successfully at 24 frames per second. Please see the supplementary video for a recording of the application running in real time.

5.7 Discussion

Our labeled training data for each expression formed small sets ranging from 100 to 800 frames of animation. Because of the small size of these sets, GPLVMs worked well to model the variation in the motion for each expression. However, dividing the data into separate sets of expressions has limitations. We cannot mix expressions because the models are separate. For example, our method is unable to combine "happy" and "surprise" expressions to synthesize a hybrid expression from both models. Generating these mixed expressions could be possible by training a GPLVM on a large, combined data set. However, we found that a GPLVM trained on this mixed set did not perform well because of the dissimilarities in the motions from the separate expressions. Additionally, the computation time required to train the model grows cubically with the size of the training data, and we found that the training times were unfeasibly long without using Gaussian process approximation techniques.

Our method's ability to synthesize transitions between models depends on its ability to find matching points between two expression models. Suppose that two GPLVM models are so different that no pair of similar points can be found. Then synthesizing transitions between the two might need to pass through a third model that has matching points with the two. For example, a transition going from happy to grumpy expressions might need to pass through a neutral expression if the happy and grumpy models share no similar points.

Chapter 6

Facial Performance Capture

6.1 Introduction

Facial performances are a key component of character animation. The expressiveness of a facial animation as well as the timing are crucial for a sense of realism. Achieving this high level of quality in a facial animation can be a time-consuming and expensive process. For animated film, the high-quality, high-end solution involves animating a character's face by hand, in which artists have full control of the appearance and timing of a facial performance. Although every aspect of the performance is directable, an artist working for a week might only produce several seconds of animation.

For live action film, visual effects artists create highly accurate and detailed virtual representations of actors' heads. These facial models are typically generated through scans of an actor's face, which require a highly sophisticated capture rig consisting of a large array

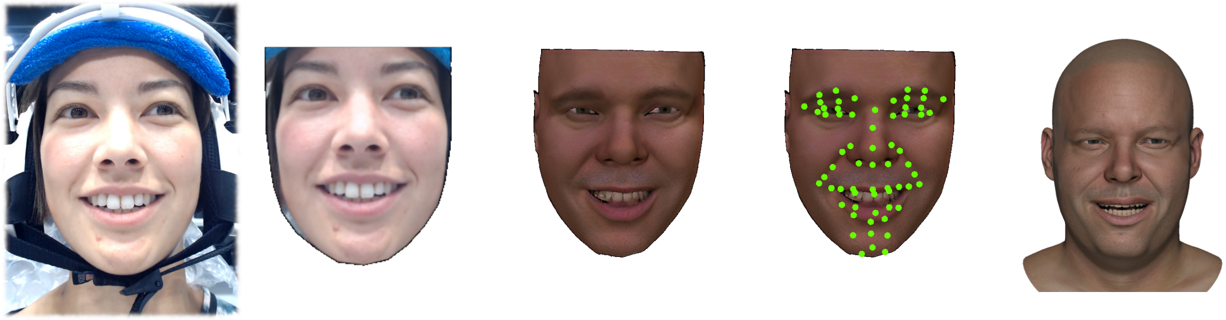


Figure 6.1: Visualization of the full blendweight solving process. Starting with a frame from a recorded video on the left, we employ style transfer to produce the middle image that has the appearance of the blendshape model but preserves the expression of the actor in the recorded frame. We then apply our blendweight optimization method to produce the deformed facial model shown on the right.

of carefully calibrated cameras [123]. Recently, data-driven approaches have been introduced that produce similar high-quality results for actors wearing head-mounted cameras [110]. Additional work has allowed for facial performance capture to transfer an actor's performance onto a facial rig whose appearance does not match the actor's [51].

Although these types of facial animations are impressive, the results take a significant amount of time and resources to produce, which limits their use to high-end productions such as big budget feature films or Triple-A video games. Lower budget productions have significantly fewer options for facial animation, and there is a noticeable difference in the quality. Our work aims to improve the accuracy of facial performance capture methods while using inexpensive equipment. Specifically, our method does not require multi-camera facial capture camera arrays nor does it require high-resolution scans of actors' faces. Instead, our approach utilizes an artist-modeled blendshape facial rig, a set of manually posed expressions on the rig, and a set of recordings captured with a helmet-mounted camera. In our experiments, we record performances with an inexpensive webcam. In total, the only equipment required to generate facial performance capture animations with our method is a helmet and a single camera. The cost of these two items is a fraction of the cost of a multi-camera performance capture system.

Given a recording of an actor, our work generates an animation to match a recorded performance. The animated facial rig is an artist-created blendshape model, and we do not require the appearance and facial features of the model to match those of the actor. The resulting animation consists only of blendweights used to deform the facial rig. We do not generate any corrective deformations to better match the recorded performance. The main benefit of animating the rig through blendshapes only is that artists can manipulate and edit the solved animation via the original controls of the facial rig.

Our facial performance capture method operates in several stages. First, the face is detected in a frame, and the image is cropped and resized while the background is removed. The image is then passed to a style transfer model that manipulates the image in pixel-space to morph the appearance of the actor to match the appearance of the blendshape model while preserving the actor's expression. Facial landmarks are then detected on the resulting image. The image and the points are passed to an iterative optimizer that estimates the rig parameters for the given frame. Finally, eye rotations are computed after the mesh and camera parameters have been fitted to the input frame. Figure 6.1 shows images from each stage of our method.

Our contributions include a style transfer method designed specifically for transferring facial expressions as well as an iterative blendweight optimizer for processing the style-transferred images. For style transfer, we adapt the first order motion model of Siarohin et al. [131] by segmenting the face and training a model for each component. We also introducing additional loss terms to improve the quality of results. We design our iterative blendweight optimizer to take advantage of constant features contained in the style transferred images. Because the facial geometry, texture, and lighting are constant in the images, the optimzer can solve for blendweight values and camera position without needing to estimate these constant parameters.

We evaluate our approach on recordings of actors whose facial features are visibly different from the animated facial rig. Furthermore, we show that our iterative optimizer produces the best results from images generated through our style transfer method when compared to other approaches. All of our results are produced with an artist-created blendshape model. The style transfer method and separate landmark detectors are the only components that requires training. We train the model with a small set of facial recordings, and we demonstrate that our method successfully generalizes to new actors and performances.

6.2 Related Work

A common approach for facial performance capture is first to fit a parametric model to an image of the actor's face and then to match the expression in the recorded face by finding the optimal rig parameters that minimize some objective function on the image. We will refer to this class of methods as optimization-based methods. These approaches start with a parametric facial model (or morphable model) [12, 13], which solves the underconstrained problem of estimating facial geometry given a static image of a head. This type of parametric model can be created by scanning several hundred heads and then correlating their geometric differences through PCA. FaceWarehouse [23] extends the morphable model by also including a statistical model of facial expressions. Furthermore, morphable models can be fit to a source and target actor to transfer expressions from one recording to another [140]. Because our method transfers expressions through manipulations in image space, our algorithm does not require a morphable model and can instead work with artist-created facial rigs.

Once a 3D model has been fit to the actor's head, the blendweights are updated according to the displacement of image features from frame to frame by solving an optimization problem [97, 156, 98, 16]. Although fitting a blendshape model to a video captures the general motion of the face, the eyes and the mouth are difficult to capture accurately. Methods to improve lip animations use an example-based approach to animate the mouth [36] or use multi-camera setups [17, 10]. Facial performance capture methods that rely on a single RGB camera tend to miss high-frequency information in the appearance of the actor's head such as wrinkles. These details can be added through texture [57] or through direct manipulation of the geometry [38]. Because our method is designed to allow artists to work with the resulting facial animation, we restrict our algorithm to blendweight manipulation only and avoid adjusting the facial model's geometry to better match a recording.

Recently, deep learning methods have helped improve results in facial reconstruction and performance capture. Deep video portraits [69] transfers the facial performance of an actor onto a target face by fitting a morphable model to the recordings and then applying deep learning models to synthesize a photorealistic image from a rasterized version of the facial model. Other methods [141, 137] train deep learning models to regress parameters of a morphable model given input images or video. In contrast, Laine et al. [78] use a neural network to output vertex positions of a facial model and avoid using a morphable model. They collect data from high-quality facial scans for use during model training. Similarly, the codec avatar [101, 155, 127] generates facial geometry and view-dependent textures from a set of input videos. Unlike these approaches, our method instead transfers the expression of a recorded actor onto an animatable facial rig in image space. As a result, our approach does not require facial scans of actors.

Style transfer is a key component of our method and can be loosely defined as rendering the content of one image in the style of another. Gatys, Ecker, and Bethge [39] introduced an iterative method for style transfer by optimizing correlations between features from intermediate layers in pretrained convolutional neural networks. Subsequent work replaced the iterative optimization step with deep learning models to allow for real-time style transfer [62, 145, 144]. Other work includes pix2pix [59], which proposes a solution to the image-to-image translation problem with paired training data. CycleGAN [164] addresses the same image-to-image translation problem but works in an unsupervised setting. StarGAN [28] addresses the multi-domain problem by utilizing pre-defined labels for images across many different style domains. More recently, StarGAN v2 [27] replaces the domain labels from StarGAN with style codes that are learned during model training.

Additional research addresses style transfer specifically for images of human faces. These approaches generate facial images through image processing alone and do not rely on geometric priors such as the morphable model. X2Face [158] develops an encoder/decoder network to learn a latent code for facial images. Similarly, DR-GAN [142] learns latent codes from facial images in which facial variations such as expression are separated from facial identity. Other work [161] synthesizes novel facial images given a source photo and the landmark positions of the target pose. CarioGAN [24] synthesizes caricature images through a style transfer component and a geometric component, which warps the style-transferred image based on exaggerated movement of facial landmarks.

For the style transfer component of our method, we use the first order motion model [131]. This approach detects motion between pairs of images through an unsupervised landmark detector. An image generator then uses the motion to warp image features in order to synthesize a new image of some target style with the driving motion from an image pair. We found this approach works well with our method because it preserves the geometry of facial images during style transfer.

6.3 Blendweight Optimization

Given a video of a facial performance $[\mathcal{I}^1, \mathcal{I}^2, ..., \mathcal{I}^T]$, where frame $\mathcal{I}^t$ is an RGB image, our method outputs a sequence of deformed meshes $\mathbf{M} = [\mathbf{m}^1, \mathbf{m}^2, ..., \mathbf{m}^T]$, where mesh $\mathbf{m}^t$ matches the facial expression depicted in frame $\mathcal{I}^t$. Here, we assume that the face depicted in the video sequence matches the appearance of the facial mesh that is fit to the images. We parameterize the deformed mesh with a blenshape model. The blendshape model is defined by a set of meshes $\mathbf{B} = [\mathbf{b}^0, \mathbf{b}^1, ...\mathbf{b}^m]$, where the mesh $\mathbf{b}^0$ is a neutral expression and mesh $\mathbf{b}^i$, $i > 0$, is some artist-defined expression such as “open mouth”. The mesh deformation can be parameterized by a vector of blendweights $\mathbf{x} \in \mathbb{R}^m$ where $x_i \in [0, 1]$ and is computed

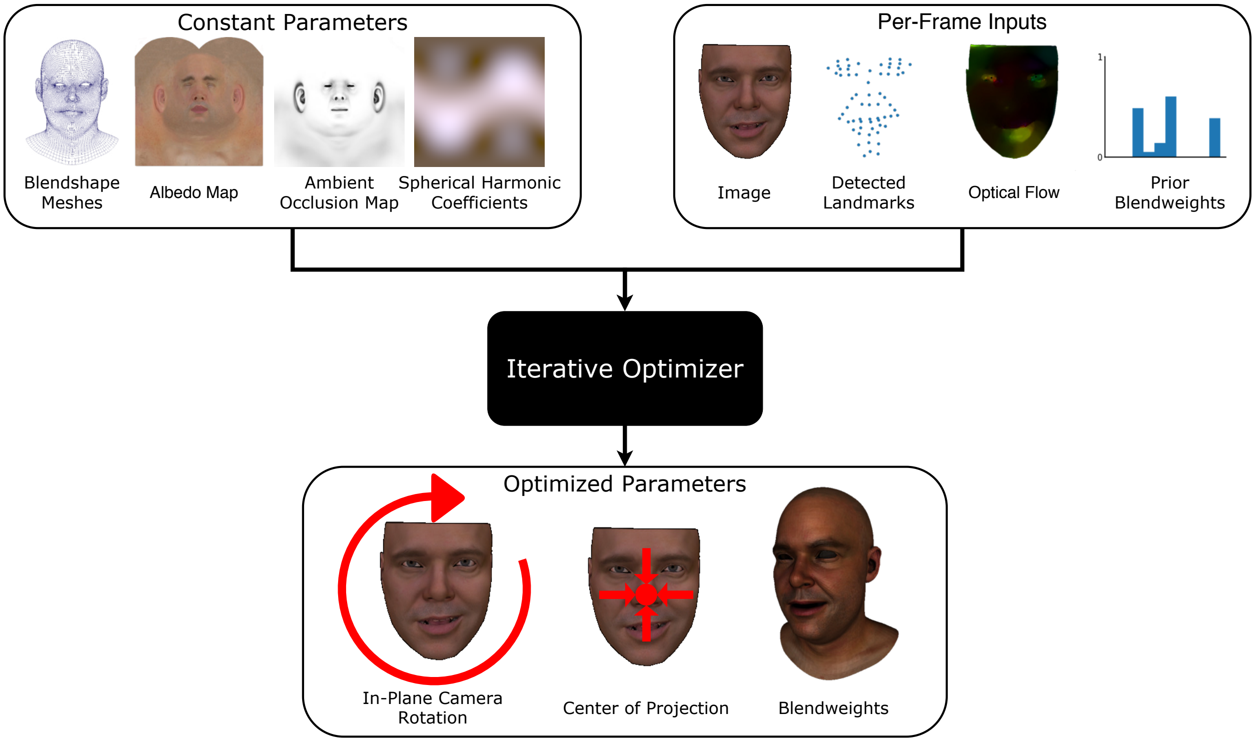


Figure 6.2: Graphical representation of our blendweight optimization method. When evaluated on a video sequence, the geometry of the blendshape model, texture maps, and lighting coefficients are held constant. For each frame sequence, the iterative optimizer is provided the image, detected facial landmarks, optical flow between the past and current frame, and blendweight values from the previous frame. The optimizer outputs blendweight values that best deform the mesh to match the image. The optimizer also outputs in-plane camera rotation and the camera's center of projection, which best aligns the blendshape model to the input image.

as

$$\mathbf{m}(\mathbf{x}) = \mathbf{b}^0 + \sum_{i=1}^{m} \left(\mathbf{b}^i - \mathbf{b}^0\right) x_i. \tag{6.1}$$

Thus, to produce a sequence of meshes given a video sequence, our method needs to estimate blendweight values $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T]$ that correspond with the facial expressions seen in the recorded video sequence $[\mathcal{I}^1, \mathcal{I}^2, ..., \mathcal{I}^T]$. Our method estimates blendweights through an iterative optimization method (Figure 6.2). Given an image frame $\mathcal{I}$ in a "rendered" style (as seen in the middle image of Figure 6.1), we minimize an objective function $\mathcal{L}(\mathbf{x}, \mathcal{I})$ to estimate the optimal blendweight values $\mathbf{x}$ for the given image $\mathcal{I}$. After the blendweights have been computed, we estimate eye rotations on the facial model to match the appearance in the image.

With an image $\mathcal{I}$ of a facial expression matching the appearance of the blendshape model, we want to estimate the blendweights $\mathbf{x}$ that best deform the mesh to match the facial expression depicted in the image. Our approach formulates blendweight estimation as an inverse rendering problem in which we identify the blendweights and environment parameters that can be used to render an image that closely matches the target image $\mathcal{I}$. Our blendweight estimation method is based on prior optimization-based approaches [97, 156, 98, 16] that estimate geometry, skin reflectance, and illumination from images. Because we have control over the "rendered" image style, we can keep the skin reflectance and illumination constant across all images and only need to estimate a small set of camera parameters and the deformed geometry through the blendweights.

Model Appearance

The "rendered" image style is generated with diffuse reflectance on the geometry and is illuminated with a spherical harmonic (SH) lighting model. Furthermore, the same albedo map and ambient occlusion map are used across all examples. Given a point $\mathbf{v}$ on the geometry's surface with normal $\mathbf{n}$ and texture coordinates $\mathbf{u}$, the radiance at point $\mathbf{v}$ can be computed as

$$\mathbf{c}(\mathbf{n}, \mathbf{u}) = \mathcal{I}_{ao}[\mathbf{u}]\mathcal{I}_{alb}[\mathbf{u}] \sum_{b=1}^{B^2} \gamma_b Y_b(\mathbf{n}) \tag{6.2}$$

The illumination is approximated by the first $B = 3$ SH bands [121]. The SH coefficients γ_b are kept constant across all images. The values $I_{ao}[\mathbf{u}]$ and $I_{alb}[\mathbf{u}]$ are the bilinearly interpolated values of the ambient occlusion map and the albedo map evaluated at texture coordinate $\mathbf{u}$.

Points on the geometry are mapped to the local coordinate frame of the camera and then to the image plane through perspective projection. Let $\mathbf{v} \in \mathbb{R}^3$ be a point in world space, let $\mathbf{R} \in \mathrm{SO}(3)$ and $\mathbf{t} \in \mathbb{R}^3$ be the camera's orientation parameters, and let $\mathbf{\Pi} : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ be the camera's perspective projection function. The position of the point $\mathbf{v}$ on the image plane can be computed as

$$\mathbf{p}(\mathbf{v}) = \mathbf{\Pi}(\mathbf{R}^{-1}(\mathbf{v} - \mathbf{t})) \tag{6.3}$$

Thus, the image coordinates of some point $\mathbf{m}_i(\mathbf{x})$ on the deformed mesh can be computed as $\mathbf{p}(\mathbf{m}_i(\mathbf{x}))$. During optimization, all camera parameters are held constant except for the center of projection as well as in-plane camera rotation.

Objective Function

Given a frame $\mathcal{I}$ from a sequence of images, we develop an objective function to estimate blendweight values $\mathbf{x}$ as well as camera parameters such that the resulting deformed mesh best matches the image when it is rendered on the image plane. For the camera parameters, we optimize the in-plane rotation θ and the center of projection $\mathbf{o}$ in the objective function. All other camera parameters remain fixed. We use the following loss:

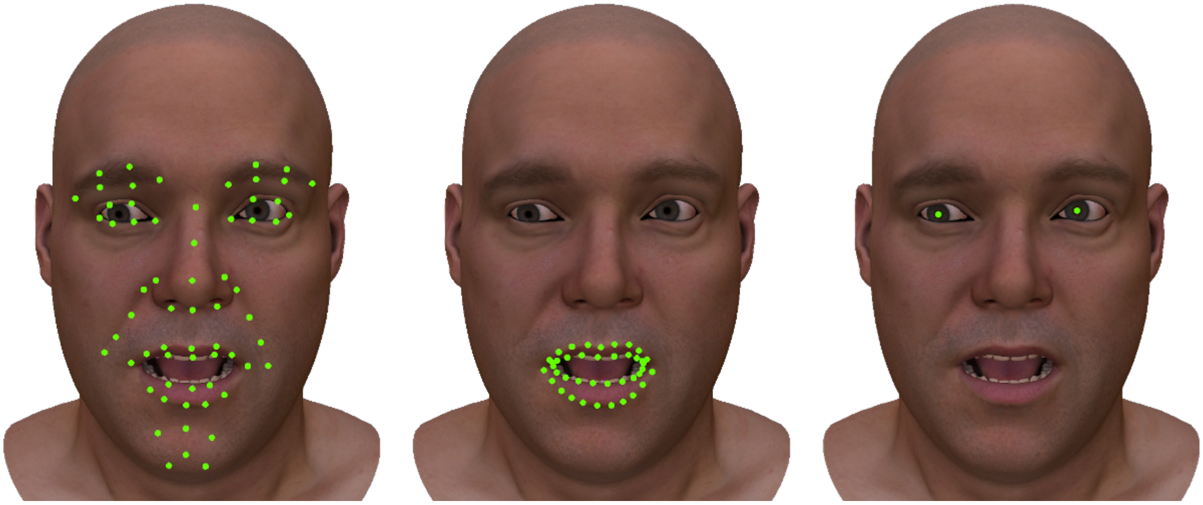


Figure 6.3: Landmark points for the full face (left), the mouth contours (middle), and pupil centers (right) plotted on the rendered image. The points on the full face plot, the outer contour of the lips on the mouth plot, and the eye plot are defined as fixed points on the mesh topology. The points along the inner contour of the mouth plot lie on the silhouette edges of the mouth, and their location on the mesh depends on the pose and camera position.

$$\begin{aligned}\mathcal{L} =& w_{photo}E_{photo} + w_{land}E_{land} + \\ & w_{smooth}E_{smooth} + w_{reg}E_{reg}\end{aligned} \tag{6.4}$$

where w_{photo}, w_{land}, w_{mouth}, w_{smooth}, and w_{reg} are user-defined positive scalars to weight each error term in the objective function.

The photometric error E_{photo} penalizes differences in the radiance on points on the mesh compared to what is observed in the image. Alternatively, this error can be viewed as rendering the mesh onto the image plane and comparing the color at specific points with the image. We define the following photometric error:

$$E_{photo} = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \|\mathcal{I}[\mathbf{p}(\mathbf{m}_i(\mathbf{x}))] - \mathbf{c}(\mathbf{n}_i, \mathbf{u}_i)\|_1 \tag{6.5}$$

The mesh is sampled at a set $|\mathcal{S}|$ of points that are picked uniformly at random across a user-defined region on the surface of the mesh. This region corresponds with the visible part of the front of the face.

The landmark error term E_{land} is designed to align a sparse set of points on the facial mesh with corresponding points on the image plane. These points are manually identified on the mesh and correspond with prominent facial features such as the eyes, nose, and mouth.

Given a set of landmark points $\mathcal{P}$, as shown in the left and middle images in Figure 6.3, and a landmark detector $D(\mathcal{I})$ that identifies landmarks in an RGB image, the landmark error is expressed as

$$E_{land} = \frac{1}{|\mathcal{P}|} \sum_{i \in \mathcal{P}} \left\| \mathbf{p}(\mathbf{m}_i(\mathbf{x})) - D_i^f(\mathcal{I}) \right\|_2^2 . \tag{6.6}$$

The landmark detector $D^f(\mathcal{I})$ outputs the landmarks as 2D image coordinates. We implement the CNN-based landmark detector described by Feng et al. [34]. We construct the training data for the detector by rendering a large set of images of the facial mesh with different poses. Because the images are rendered from the mesh, we can compute exactly where each landmark is located in the images, which we use to train the landmark detector.

To ensure that mouth shapes are accurately reconstructed from the input image, we include a set of landmark points following the contours of the mouth. We evenly space the points along the contours of the outer lips and the inner lips. The outer lip contours are defined by the boundary between the lips and the skin around the mouth. These landmarks are computed as fixed points on the mesh topology. The inner lip contours are defined as the silhouette edge of the mouth as seen from the view of the camera. Because the silhouette edge can change with respect to different mouth shapes, the landmarks for the inner lips are not fixed to specific points on the mesh. We evenly distribute the points along the contour according to the euclidean distance between the vertices on the silhouette edges in the deformed mesh $\mathbf{m}(\mathbf{x})$. The middle image of Figure 6.3 depicts the landmarks on the outer and inner lip contours. To identify the corresponding points in an image, we train a separate landmark detector.

To promote temporal coherence between frames, we utilize optical flow maps to penalize mesh movement in regions with little to no detected motion. For a given frame t, let $\mathcal{F}^{t-1}$ represent the optical flow map from frame $\mathcal{I}^{t-1}$ to frame $\mathcal{I}^t$. Let $\mathbf{x}^{t-1}$, θ^{t-1}, and $\mathbf{o}^{t-1}$ be the optimized blendweights and camera parameters estimated for frame $\mathcal{I}^{t-1}$. First, for each point in the set $\mathcal{S}$ of sampled points from Equation 6.5, we compute the influence

$$\alpha_{i,k} = \frac{\left\| \mathbf{b}_i^0 - \mathbf{b}_i^k \right\|_2}{\sum_{j \in \mathcal{S}} \left\| \mathbf{b}_j^0 - \mathbf{b}_j^k \right\|_2} \tag{6.7}$$

for some point $i \in \mathcal{S}$ on the mesh and blendshape $k > 0$ in the facial rig. Next, we compute the magnitude of movement for each sampled point through the optical flow map. This magnitude is represented as $\mu_i = \|\mathcal{F}^{t-1}[\mathbf{p}(\mathbf{m}_i(\mathbf{x}^{t-1}))]\|_2$. The values in the optical flow map are bilinearly sampled according to the projection of the mesh $\mathbf{m}(\mathbf{x}^{t-1})$ onto the image plane. We can then penalize changes in blendweights according to flow magnitudes as follows:

$$E_{smooth} = \sum_{k=1}^{m} \exp\left(\frac{-\sum_{i \in \mathcal{S}} (\alpha_{i,k}\mu_i)}{\sigma} \right) \left(x_k^{t-1} - x_k \right)^2 \tag{6.8}$$

where $\sigma > 0$ is a user-specified hyperparamter.

Finally, to enforce sparsity in the blendweights $\mathbf{x}$, we use L1 regularization: $E_{reg} = \|\mathbf{x}\|_1$. This sparsity-inducing regularization encourages the optimizer to activate only blendshapes that are required to minimize the objective.

Optimization

To compute the optimal blendweight values, we minimize the objective function (Equation 6.4) for the blendweights $\mathbf{x}$ and the camera parameters θ and $\mathbf{o}$ simultaneously. When solving for blendweights across a video sequence, we use the optimal values from the previous frame as the initial guess for the next frame. To minimize the objective, we use the Newton-Raphson method with box constraints on the blendweights to ensure that each value lies in the range $[0, 1]$. At each iteration, we fit a quadratic to the current guess and solve for the optimal value of the constrained problem. We use the method of Goldfarb and Idnani [42] to solve the quadratic program. We perform a line search in the direction of the solution using the bisection method. For a given frame, we run up to 10 iterations of Newton-Raphson and terminate early if the line search fails to reduce the value of the error function.

Computing the silhouette edges for the landmark points along the mouth in Equation 6.6 during optimization could introduce discontinuities in the loss function. To avoid these complications, we compute the silhouette edges from the initial guess for the blendweights. The edges are held constant throughout the optimization process for each frame and are only updated when optimization begins for the next frame in a sequence.

Eye Tracking

After the blendweights and camera parameters have been fit to an image, we have sufficient information to rotate the facial rig's eyes to match the image. Given the centers of the pupils in the image and in the mesh, we compute the eye rotations such that when the mesh is projected onto the image plane the pupil's center aligns with the corresponding point in the detected image as depicted on the right in Figure 6.3. Furthermore, to ensure that the resulting eye rotations appear realistic, we compute the transformation as two rotations: first about the X axis then about the Y axis, assuming that the front of the facial mesh is pointing along the Z axis.

For the eye, let $\mathbf{e}$ be the center of rotation for the mesh and let $\mathbf{p}$ be the point on the surface of the mesh representing the center of the pupil. We do not assume that $\mathbf{p} - \mathbf{e}$ aligns with the Z axis. Given the optimized camera parameters and the image coordinates of the center of the pupil detected in the frame, we construct a ray that deprojects the point into the 3D scene. Let $\mathbf{t}$ be the intersection of this ray with a sphere centered on $\mathbf{e}$ with radius $r = \|\mathbf{p} - \mathbf{e}\|_2$. The eye rotations are then computed as the rotations about the X and Y axes that align $\mathbf{p} - \mathbf{e}$ with $\mathbf{t} - \mathbf{e}$.

We first calculate the rotation about the X axis as the angle between $\mathbf{p} - \mathbf{e}$ and an intermediate vector $\mathbf{a}$. The Y rotation is then computed as the angle between $\mathbf{a}$ and $\mathbf{t} - \mathbf{e}$. This intermediate vector is the intersection point of the plane $x = \mathbf{p}_x - \mathbf{e}_x$, the plane

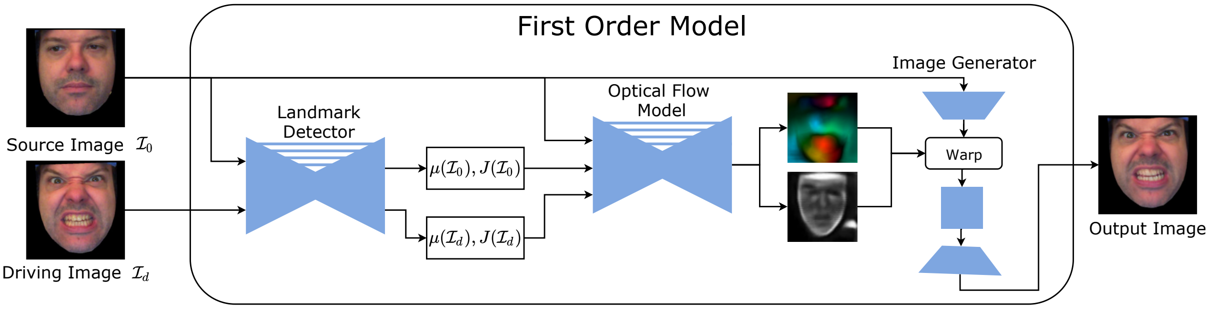

First Order Model
Image Generator
Optical Flow Model
Landmark Detector
Source Image $\mathcal{I}_0$
Driving Image $\mathcal{I}_d$
$\mu(\mathcal{I}_0), J(\mathcal{I}_0)$
$\mu(\mathcal{I}_d), J(\mathcal{I}_d)$
Warp
Output Image

Figure 6.4: Style transfer network evaluated with a source image $\mathcal{I}_0$ and a driving image $\mathcal{I}_d$ from the same video. The landmark detector computes the landmark positions $\boldsymbol{\mu}$ and the transformation matrices $\mathbf{J}$. These outputs are then given to the optical flow model along with the source image. This model produces a dense flow field and an occlusion map, which are given to the generator along with the source image. The image generator then outputs an image that closely matches the driving image $\mathcal{I}_d$.

$y = \mathbf{t}_y - \mathbf{e}_y$, and the sphere centered on the origin with radius r. If there are two points in this intersection, we select $\mathbf{a}$ as the point on the same side of the plane $x = 0$ as $\mathbf{p} - \mathbf{e}$. If there is no intersection, then the deprojected ray from the image plane does not intersect the sphere, and we cannot estimate eye rotations for this frame with this approach.

Our eye tracking approach works well when the pupils are correctly positioned in the style transferred input image. However, the style transfer method that we use occasionally misplaces the pupils in the image. When this error occurs, our eye tracking method produces undesirable cross-eyed results. In these cases, we compute the rotation for only one of the eyes and apply this transformation to both eyes.

6.4 Style Transfer

Given a recording of an actor, we want to synthesize a new recording that preserves the expressions of the actor but swaps the identity with that of the facial rig so that our iterative optimizer can fit the blendshape model to the image. Our style transfer method directly manipulating images without knowledge of the underlying digital face.

Our approach is based on the first order motion model [131], which transfers motion from a video sequence onto a separate subject. We use this method to transfer the motion in a recording of an actor onto an image in the "rendered" style for blendweight optimization. The model uses a source image in the "rendered" style and manipulates it according to the motion observed between two frames from a video recording of an actor. Thus, the motion of an actor's facial features are directly transferred onto an image of the facial model without any direct knowledge of the underlying blendshapes.

Style Transfer Model

Our style transfer method follows the same architecture described by Siarohin et al. [131]. The model consists of three components: an unsupervised landmark detector, an optical flow field generator, and an image generator. Given an input image $\mathcal{V}$, the landmark detector outputs k image points $\boldsymbol{\mu}(\mathcal{V}) \in \mathbb{R}^{k\times 2}$ as well as a 2×2 transformation matrix $\mathbf{J}(\mathcal{V}) \in \mathbb{R}^{k\times 2\times 2}$ for each point. The landmark detector processes a source image and a frame from a driving video, and the optical flow model processes the outputs from the landmark detector. The optical flow model produces a dense motion map along with an occlusion map, which is then used by the image generator module. The image generator processes the source image through several convolutional layers. Next, the the intermediate feature map is warped according to the dense motion map, and the features are masked by the occlusion map. The warped features are then passed through several residual layers and finally several convolutional and upsampling layers to produce an output image. Figure 6.4 depicts the first order model evaluated on a source and driving image of the same style. We only describe in detail the components of the first order model related to our modifications of the original method.

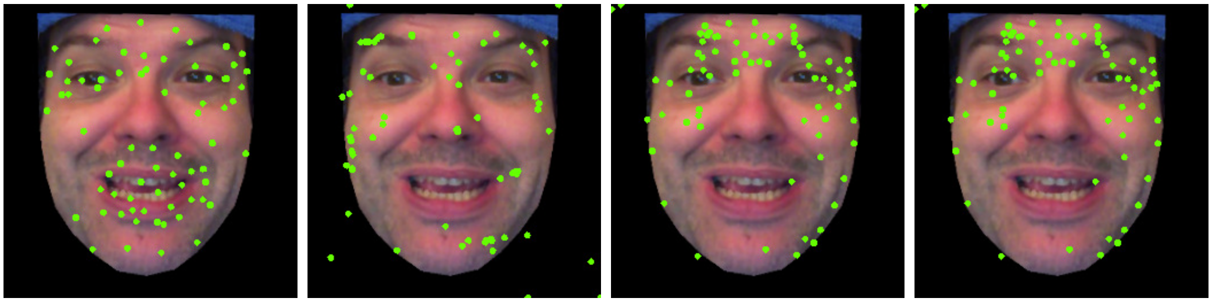
The model is trained to reconstruct a driving image $\mathcal{V}_d$ from two components: an image $\mathcal{V}_0$ of the same style and a sparse set of features computed from both images. We optimize the model parameters using the perceptual loss as well as the equivariance constraint proposed by the authors of the first order model. Additionally, we introduce a landmark distance loss, a background-deterring loss, and a regularization term to promote a better distribution of landmark points and transformation matrices across human faces in input images. Figure 6.5 shows a comparison of landmarks points detected by models trained with and without these additional loss terms. Given a driving image $\mathcal{V}_d$ and the reconstructed image $\hat{\mathcal{V}}_d$ generated by the first order model, the loss function is defined as

$$\begin{aligned}\mathcal{L}(\mathcal{V}_d, \hat{\mathcal{V}}_d) = & L_{percep}(\mathcal{V}_d, \hat{\mathcal{V}}_d) + L_{equi}(\mathcal{V}_d) + \\ & L_{dist}(\mathcal{V}_d) + L_{bg}(\mathcal{V}_d) + L_{reg}(\mathcal{V}_d).\end{aligned} \tag{6.9}$$

Our proposed loss terms primarily affect the unsupervised landmark detector. To generate k landmark coordinates, the model first produces one heatmap for each landmark. The softmax operation is then applied to the heatmap to produce a set of confidence maps $C(\mathcal{V}_d) \in [0,1]^{h\times w\times k}$. The 2D positions are then computed through the summation of the confidence maps $\boldsymbol{\mu}_i(\mathcal{V}_d) = \sum_{y=1}^{h}\sum_{x=1}^{w} C_i(\mathcal{V}_d)$. Because our use-case only requires that the face in an image be accurately reconstructed, the landmark detector should place all points on the face and ignore the background in an image. To achieve this goal, we first replace the background in the input image with solid black to remove any unnecessary image features. Second, we apply a background-deterring constraint to the confidence maps as follows:

$$L_{bg}(\mathcal{V}_d) = \lambda_{bg} \sum_{i=1}^{k}\sum_{y=1}^{h}\sum_{x=1}^{w} C_i(\mathcal{V}_d)[x,y] \cdot \mathcal{M}(\mathcal{V}_d)[x,y] \tag{6.10}$$

where $\lambda_{bg} > 0$ is a user-defined hyperparameter. $C_i(\mathcal{V}_d)[x,y]$ indicates the pixel value at the coordinate (x,y) in the confidence map for landmark i, and $\mathcal{M}(\mathcal{V}_d) \in \{0,1\}^{h\times w}$ is a binary



(a) Full loss function (b) No loss terms L_{dist}, L_{bg}, and L_{reg} (c) No Segmentation (d) Original First Order Motion Model

Figure 6.5: Visualization of landmark points generated by a model trained on the full objective function, a model trained without the distance, background, and regularization loss, a model trained without facial segmentation, and the original first order motion model.

mask that is nonzero for pixels lying on the background of the image. Figure 6.6 shows an example of a video frame with the background removed along with the mask $\mathcal{M}$.

The distance error $L_{dist}(\mathcal{V}_d)$ penalizes overlapping landmarks and landmarks that are in close proximity. Given the set of k landmarks $\boldsymbol{\mu}(\mathcal{V}_d)$, we compute the pair-wise squared euclidean distance between the landmarks $d_{ij} = \|\boldsymbol{\mu}_i(\mathcal{V}_d) - \boldsymbol{\mu}_j(\mathcal{V}_d)\|_2^2$. The distance loss is then computed as

$$L_{dist}(\mathcal{V}_d) = \lambda_{dist} \sum_{i=1}^{k} \sum_{j=i+1}^{k} \exp(-d_{ij}T) \tag{6.11}$$

where $\lambda_{dist} > 0$ and $T > 0$ are user-defined hyperparameters.

The regularization term $L_{reg}(\mathcal{V}_d)$ penalizes large eigenvalues in the transformation matrices $\mathbf{J}(\mathcal{V}_d)$. These matrices are part of affine transformations in the image space. Large eigenvalues would indicate regions undergoing significant stretching and scaling deformations. On human faces, the mouth region typically exhibits the largest deformations across various poses, and the transformation matrices can reflect information about stretched and deformed parts of the mouth. However, we found that penalizing large transformations in $\mathbf{J}(\mathcal{V}_d)$ produces visually better results when the first order model is used for style transfer. To regularize these matrices, we apply the following loss:

$$L_{reg}(\mathcal{V}_d) = \lambda_{reg} \sum_{i=1}^{k} \|\mathbf{J}_i(\mathcal{V}_d) - I\|_F^2 \tag{6.12}$$

where I is the 2×2 identity matrix. Although we could penalize large eigenvalues in $\mathbf{J}(\mathcal{V}_d)$ with a rotation-invariant loss, we found that Equation 6.12 is simple to implement, produces good results, and can easily be differentiated.

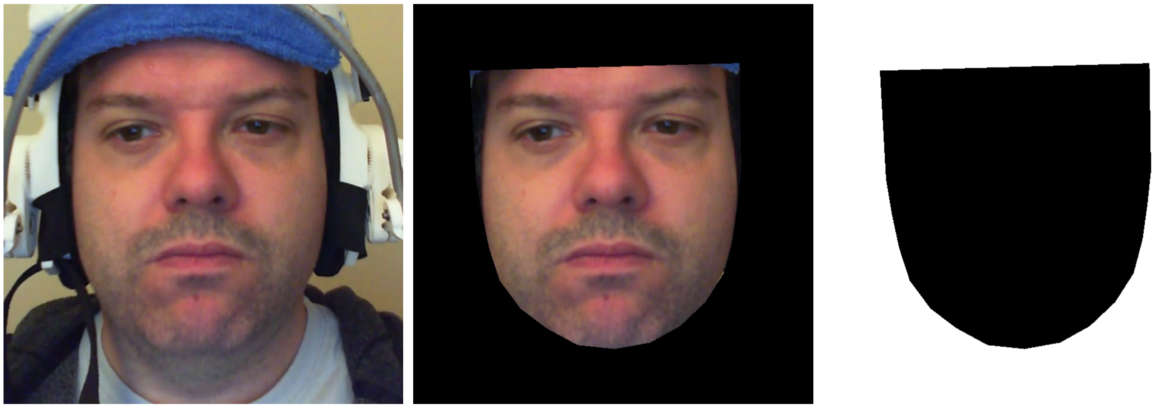


Figure 6.6: Example of a recorded frame (left) with the background removed (middle) and the mask $\mathcal{M}$ used for the background-deterring objective.

Model Evaluation

During training, the style transfer network is optimized to reconstruct a frame $\mathcal{V}_d$ in a driving video from a source frame $\mathcal{V}_0$ taken from the same recording. However, we want the network to transfer the expression of a face in a recorded video onto an image with a "rendered" style. Given a frame $\mathcal{V}_d$ from a driving video along with a reference frame $\mathcal{V}_0$ and a source image $\mathcal{I}_0$ that both depict the same facial expression (such as a neutral expression), the model produces an image $\mathcal{I}_d$ that preserves the style of the source image $\mathcal{I}_0$ but contains the expression in the video frame $\mathcal{V}_d$.

To use the first order motion model for style transfer, we manipulate the coordinates and transformation matrices output by the landmark detector module. These outputs serve as a low-dimensional representation of the input images and functions as a bottleneck for the image generator. Given a driving image $\mathcal{V}_d$ and a source image $\mathcal{V}_0$, the optical flow module receives the relative motion of the landmarks as well as the relative change in the transformation matrices. The relative motion for the landmarks is given by $\boldsymbol{\mu}_{0\to d} = \boldsymbol{\mu}(\mathcal{V}_d) - \boldsymbol{\mu}(\mathcal{V}_0)$, and the change in the transformation matrices is given by $\mathbf{J}_{0\to d} = \mathbf{J}(\mathcal{V}_d)\mathbf{J}(\mathcal{V}_0)^{-1}$.

The first order motion model transfers facial expressions by applying the relative motion in landmarks between the frames $\mathcal{V}_0$ and $\mathcal{V}_d$ onto the source image $\mathcal{I}_0$. We approximate landmark positions for the to-be-estimated image $\mathcal{I}_d$ as follows:

$$\boldsymbol{\mu}(\mathcal{I}_d) \approx \boldsymbol{\mu}(\mathcal{I}_0) + \boldsymbol{\mu}_{0\to d}. \tag{6.13}$$

Because the image $\mathcal{I}_d$ does not exist, the landmarks $\boldsymbol{\mu}(\mathcal{I}_d)$ are estimated by applying the motion of the landmarks between video frames $\mathcal{V}_0$ and $\mathcal{V}_d$ onto the landmarks from the "rendered" source image $\mathcal{I}_0$. Similarly, we transfer the relative change in the transformation matrices produced by the landmark detector. This change is expressed as follows:

$$\mathbf{J}(\mathcal{I}_d) \approx \mathbf{J}_{0\to d} \cdot \mathbf{J}(\mathcal{I}_0). \tag{6.14}$$

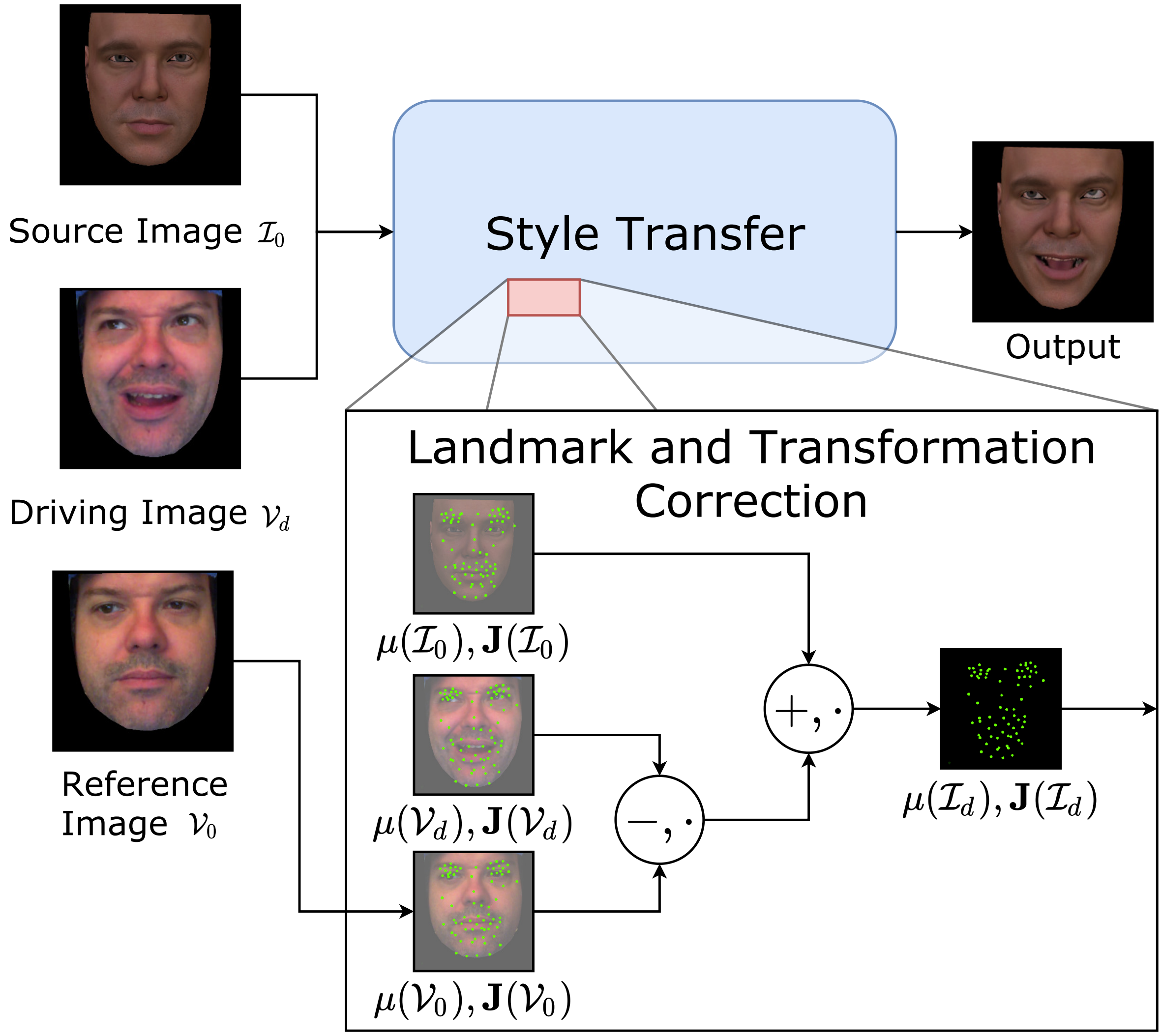

Source Image $\mathcal{I}_0$
Driving Image $\mathcal{V}_d$
Reference Image $\mathcal{V}_0$
Style Transfer
Output
Landmark and Transformation Correction
$\mu(\mathcal{I}_0), \mathbf{J}(\mathcal{I}_0)$
$\mu(\mathcal{V}_d), \mathbf{J}(\mathcal{V}_d)$
$\mu(\mathcal{V}_0), \mathbf{J}(\mathcal{V}_0)$
$-, \cdot$
$+, \cdot$
$\mu(\mathcal{I}_d), \mathbf{J}(\mathcal{I}_d)$

Figure 6.7: Landmark and transformation matrix correction during style transfer evaluation with a source image $\mathcal{I}_0$ in the "rendered" style. Evaluation proceeds as illustrated in Figure 6.4 with images $\mathcal{I}_0$ and $\mathcal{V}_d$. Landmark and transformation correction occurs between the landmark detector and the optical flow model. The landmark and transformation correction component provides the optical flow model with an approximation of the coordinates for $\boldsymbol{\mu}(\mathcal{I}_d)$ and the transformations $\mathbf{J}(\mathcal{I}_d)$.

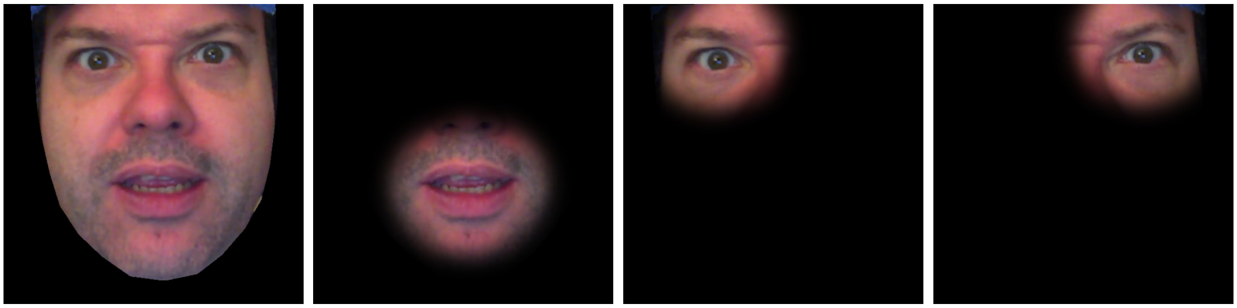


Figure 6.8: Example of an input frame (left) with the a mask applied to the mouth (middle-left), the right eye (middle-right), and the left eye (right).

Figure 6.7 graphically depicts how the first order model is used to produce style transferred image.

Facial Segmentation

Although the style transfer network can operate on the full image of a face, we found that more accurate results were achieved by segmenting the face into regions and training a separate model on each facial region. Specifically, we have a model for the full face, a model for the mouth, one model for the left eye and eyebrow, and one for the right eye and eyebrow. These facial regions are identified in the input images through a separate facial landmark detector. Figure 6.8 shows an example frame along with masks applied to the three separate regions of the face.

For each region in an image, we identify corresponding landmarks and compute the convex hull containing those points. The convex hull is then expanded by a user-defined margin, and the shape is used to mask the image so that only the relevant facial region is visible. During model training, the unmasked images are given as input. When evaluating the objective function (Equation 6.9), the driving image $\mathcal{V}_d$ and the reconstructed image $\hat{\mathcal{V}}_d$ are masked. Thus, the model has access to the full image but only needs to output the image within the masked region. Each style transfer model is trained separately. Figure 6.5 shows a comparison of detected landmark positions for models trained with and without facial segmentation. For the style transfer models, we allow 24 landmark points for the models trained on the full face and mouth. We allow 12 points each for the models trained on the eyes.

When evaluating the model for style transfer, we merge the results together to construct a single image. First, the model of the full face is evaluated. Next, the remaining style transfer models are evaluated separately and merged onto the output of the model of the full face. The images are merged according to the mask of the driving image $\mathcal{V}_d$. Pixels within the mask are copied onto the combined image while pixels outside the mask remain unchanged. Although the output image is of the “rendered” style, we found that reasonable results are achieved using masks from the driving image $\mathcal{V}_d$ to merge the results of the style transfer models onto image $\mathcal{I}_d$.

Image Super-Resolution

Due to memory and computation restrictions, our implementation of the style transfer network outputs images of size 256 × 256. However, the blendweight optimization method benefits from higher resolution images. To increase the resolution of the image, we train SR-GAN [87] on images of the "rendered" style only. Because we can produce rendered images of arbitrary size, generating a dataset with higher resolution images is trivial. We use the super-resolution model to double the image resolution to 512 × 512.

6.5 Model Training

For training, the style transfer networks require a set of images in the "rendered" style, featuring a diverse set of facial expressions as well as a set of images of actors covering many head shapes, facial appearances, and expressions. To create the "rendered" style images, we rasterize images of the blendshape model posed with randomly generated expressions. The camera parameters and lighting conditions are held constant across all images. The random expressions are generated by augmenting a small set of artist-created poses. To synthesize a new pose, we first divide the blendweights into two sets: one set for the eyes and upper half of the face and another set for the mouth and the lower half of the face. Next, we select two poses uniformly at random from the set of artist created poses. For each pose with probability 0.5, we randomly choose to mirror the expression or use the original pose. A new pose is constructed by combining the set of upper half blendweights from one pose with the set of lower half blendweights from the other pose. Finally, we add uniformly random noise to all non-zero blendweights in the synthesized pose. The noise is drawn i.i.d. from the distribution $U(-0.15, 0.15)$, and the resulting blendweight values are clamped to the range $[0, 1]$.

We construct the image set of actors by recording performers through a helmet-mounted webcam. The camera records the performance at 30 FPS. Each recording consists of the actor physically expressing various emotional states as well as the actor speaking both extemporaneously as well as from a script.

For rendered images and images of actors, we crop the image around the face, replace the background with black and resize the image to a resolution of 256 × 256. We use a landmark detector trained on a distribution of 68 facial landmarks as defined in the iBUG 300-W dataset [125]. With the detected landmarks, we crop each image so that the image is square and so that the face occupies the center of the image. We then compute the convex hull of the landmarks and set to black any pixel that lies outside of the convex hull.

During training of the style transfer model for each batch sample, we randomly pick with 0.5 probability to use the rendered image set or any of the recorded image. If we choose to use an example from the recorded images, we pick one recording uniformly at random from the set of all recordings of all actors. Then two images from the same recording are selected uniformly at random with one image as the source and the other as the driving image. For the landmark detection models used for blendweight optimization, we compute landmark positions for all of the “rendered” images in the training set. These models and the SRGAN model are only trained on this “rendered” image set, and the images of recorded actors are not used. When training all of the models, we augment the data by applying random rotations, translations, and thin plate spline warps to the input samples.

6.6 Results

To evaluate our method, we construct a dataset of video recordings and one blendshape model along with a set of artist-created poses for the facial rig. Our training set of video recordings consists of 92 minutes of facial performances across 14 actors, including 12 male actors and 2 female actors. One additional male and one female actor are recorded and used for evaluation purposes. The videos are captured with an inexpensive webcam and are recorded at a rate of 30 FPS with a resolution of 1280 × 720. The camera is mounted on a helmet worn by the actor to eliminate all head rotation relative to the camera.

The artist-created facial blendshape rig is modeled after one of the actors in the training set of our experiments. The model consists of 11,406 vertices. In the blendweight optimizer, we deform the mesh with 92 blendshapes. To generate the training images for the style transfer model, an artist manually creates 104 poses that cover the full range of emotional expressions and the full range of visemes of the blendshape model. We augment the poses using the method described in the previous section to produce a set of 2,000 samples.

We implement our method in Python and use TensorFlow for the deep learning components. We run our method on a 20 core machine running at 2.20 GHz with one Nvidia Titan RTX GPU. Training the four style transfer networks takes roughly one day in total. When solving for blendweights, our iterative optimizer runs at a rate of roughly 20 seconds per frame.

The quality of the style transferred image has a significant impact on the results from the blendweight optimization process. Our blendweight solver assumes a fixed head shape for the blendshape model. Thus, the appearance of the physical structure of the face must be preserved in the stylized image while also accurately transferring the expression from a recorded actor. In our experiments, we first compare our style transfer method’s ability to reconstruct images from the same style domain. For our second experiment, we construct a small set of artist-posed expressions from recorded video frames and use these rendered images as an approximation for the ground truth blendweight values. In this evaluation, we compare deformed meshes estimated through our style transfer results and through other style transfer approaches.

Table 6.1: Average video reconstruction errors using our method, our method without segmentation, our method without our additional loss terms, the original first order method, and X2Face.

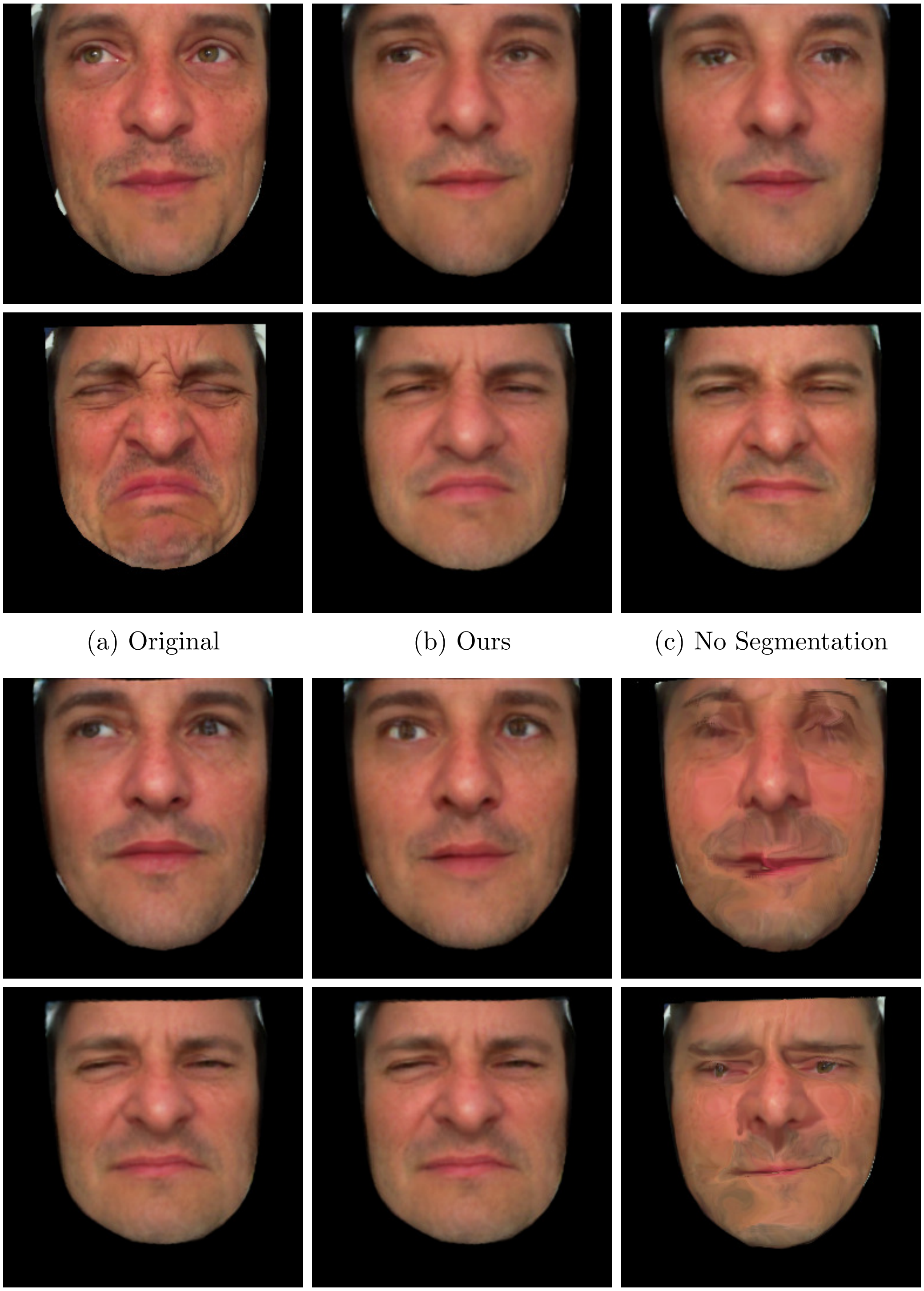
	Test		Train	
	RMSE	LPIPS	RMSE	LPIPS
Our Method	10.9	0.277	6.8	0.176
No Segmentation	11.8	0.269	7.2	0.182
No L_{dist}, L_{bg}, and L_{reg}	11.4	0.279	7.1	0.181
First Order	11.6	0.273	7.5	0.180
X2Face	17.3	0.370	13.5	0.291

Style Transfer Results

We first compare the accuracy of our method when reconstructing an image from the same style domain. We evaluate the performance of our method against four other approaches. First, we test a variant of our approach in which a single network generates the whole image instead of using multiple networks that focus on specific regions of the face. The number of landmarks tracked by this variant is the same as the sum of the landmarks across the four networks in our approach. Second, we compare against another variant of our model that is trained with image segmentation but without our proposed loss terms in Equation 6.9. Finally, we evaluate our method against the original first order motion model [131] as well as X2Face [158], a model that produces a latent representation of an image and reconstructs it through image warps on a different image. Thus, all of these methods take a source image and a driving image as inputs and reconstruct the driving image through some latent representation. For the source image, we use a recorded frame of the actor in a neutral expression.

We train all of the models with the previously described image dataset. For evaluation, we run the models on roughly 6 minutes of recordings of a male actor and a female actor not seen during training. We also evaluate the reconstruction of recordings of one male and one female actor contained in the training set to evaluate each approach's ability to generalize. Each model reconstructs the recordings, and we compute the difference between the reconstruction and the original. We measure the differences with both the root mean squared error (RMSE) and the Learned Perceptual Image Patch Similarity (LPIPS) metric [163], which measures image similarities through deep visual features. RMSE is measured on the images with colors scaled to the range $[0, 255]$. As seen in Table 6.1, our method shows a slight improvement on the root mean squared error over the original first order model as well as X2Face. Lower values are better for both error measurements.

Figure 6.9 shows the reconstructions for two recorded frames from the test set. The images from X2Face show the largest visual differences. The clear difference in the reconstruction quality suggests that the X2Face method might not generalize well to new faces



(a) Original

(b) Ours

(c) No Segmentation

(d) No L_{dist}, L_{bg}, L_{reg}

(e) First Order

(f) X2Face

Figure 6.9: Side-by-side comparison of reconstructed images.

Table 6.2: Posing errors (average distance error in cm). Errors are measured on sets of artist-created poses for recordings of two male and two female actors. For each recording, the point-wise distance errors are measured on a set of locations sampled uniformly at random across the front of the facial model. The second set of errors are measured from the point-wise distances of a specific set of facial landmarks on the mesh.

	Male Actors			
	Test Set		Training Set	
	Front Face	Landmarks	Front Face	Landmarks
Our Method	0.25	0.27	0.19	0.25
First Order	0.26	0.28	0.23	0.29
CycleGAN	0.26	0.30	0.32	0.36
Neural Style	0.62	0.76	0.83	1.15

	Female Actors			
	Test Set		Training Set	
	Front Face	Landmarks	Front Face	Landmarks
Our Method	0.28	0.30	0.30	0.33
First Order	0.28	0.30	0.34	0.35
CycleGAN	0.37	0.40	0.35	0.37
Neural Style	0.75	0.87	0.83	0.97

when trained on our small dataset with a small number of unique actors. The bottom row in the figure shows a limitation of our method and the first order method. The models are unable to reconstruct the wrinkles in the actor's face. Because the models generate the images partially through image warps, they are not well-suited for reconstructing pose-specific features, such as wrinkles, that cannot be reconstructed through image warps alone. We observe this behavior when using the model to transfer facial expressions onto the "rendered" style. However, we found that our blendweight optimization method produces reasonable results despite the lack of wrinkles in certain expressions.

Blendweight Optimization Results

Next, we evaluate the quality of solved blendweights given a style-transferred image. We found that preserving the visual proportions of the blendshape model in the "rendered" image style is essential for the accurate performance of our blendweight optimization method. To test the quality of the style-transferred images, an artist manually poses the blendshape model to match several frames from both the training set and test set of recordings. We then use the blendweight optimizer to solve for the poses in the set of images. An error is then computed from the distance between the surface of the artist-posed mesh and the blendshape face deformed by the optimized pose.

An ideal comparison would have a ground truth animation fit to the evaluation recording. However, because the geometry of the blendshape model does not match the geometry of the recorded actor, we cannot compare a solved animation to a 3D reconstruction of the actor's performance. As an alternative, an artist poses the facial model to a subset of frames from recordings of both male and female actors in the training and test sets that we use for evaluation. We assume that the professional artist can accurately reconstruct a facial expression from a reference image, and we use these manually posed expressions as a proxy for the ground truth. Animating and keyframing the blendshape model for the full recording would provide the most data for comparison. However, this process is too time-consuming, and blended poses between keyframes in the animation might not precisely match the corresponding facial expressions in the recordings. Thus, we settle for a subset of static poses modeled after specific frames in the video recordings.

In our experiment, we compare the results of the optimization method applied to images generated by our style transfer method, the first order motion model [131], CycleGAN [164], and A Neural Algorithm of Artistic Style [39]. Because CycleGAN is designed to translate images from only one predefined style domain to another, we train the model on frames of the recordings in which it is evaluted. In contrast, our style transfer approach is not limited to a pre-determined style domain, and the model does not see frames from the recordings in the test set during training. Although StarGAN v2 [27] would be an ideal method for comparison, we were unable to train the model successfully with our dataset.

We use each style transfer method to produce a sequence of 256×256 images. The images are then upsampled via our trained SRGAN model. Next, we detect annotated landmark points on the "renered" style images (Figure 6.3). Finally, we generate an animation for the full sequence of images and collect a subset of poses corresponding with the frames in which we have ground truth data.

To compare individual frames of an animation quantitatively, we compute the distance of a set of points. Because the topology of the mesh never changes when deformed by blendshapes, we can compute corresponding points on the surfaces of two deformed meshes. Given a set of n frames and K points, let $\mathbf{p}_k^i$ represent the point k on the ground truth deformed mesh for frame i, and let $\widetilde{\mathbf{p}}_k^i$ represent the point k on the deformed mesh computed through solved blendweights. We compute the error as the average distance between the corresponding points

$$D = \frac{1}{nK} \sum_{i=1}^{n} \sum_{k=1}^{K} \left\| \mathbf{p}_k^i - \widetilde{\mathbf{p}}_k^i \right\|_2 . \tag{6.15}$$

We evaluate the error on two different sets of points. First, we construct as set of points by sampling 1,000 points uniformly at random across the surface of the front of the facial mesh. The second set of points consists of the landmarks for the full face as depicted on the left in Figure 6.3. We evaluate our method across four recordings consisting of two male and two female actors. One of the male actors and one of the female actors are included in the training set used to train our style transfer models. For CycleGAN, we train a separate model for each recording. These CycleGAN models are trained on a dataset consisting of rendered images of the blendshape model for one style domain, and all frames from a single recorded actor as the other style domain.

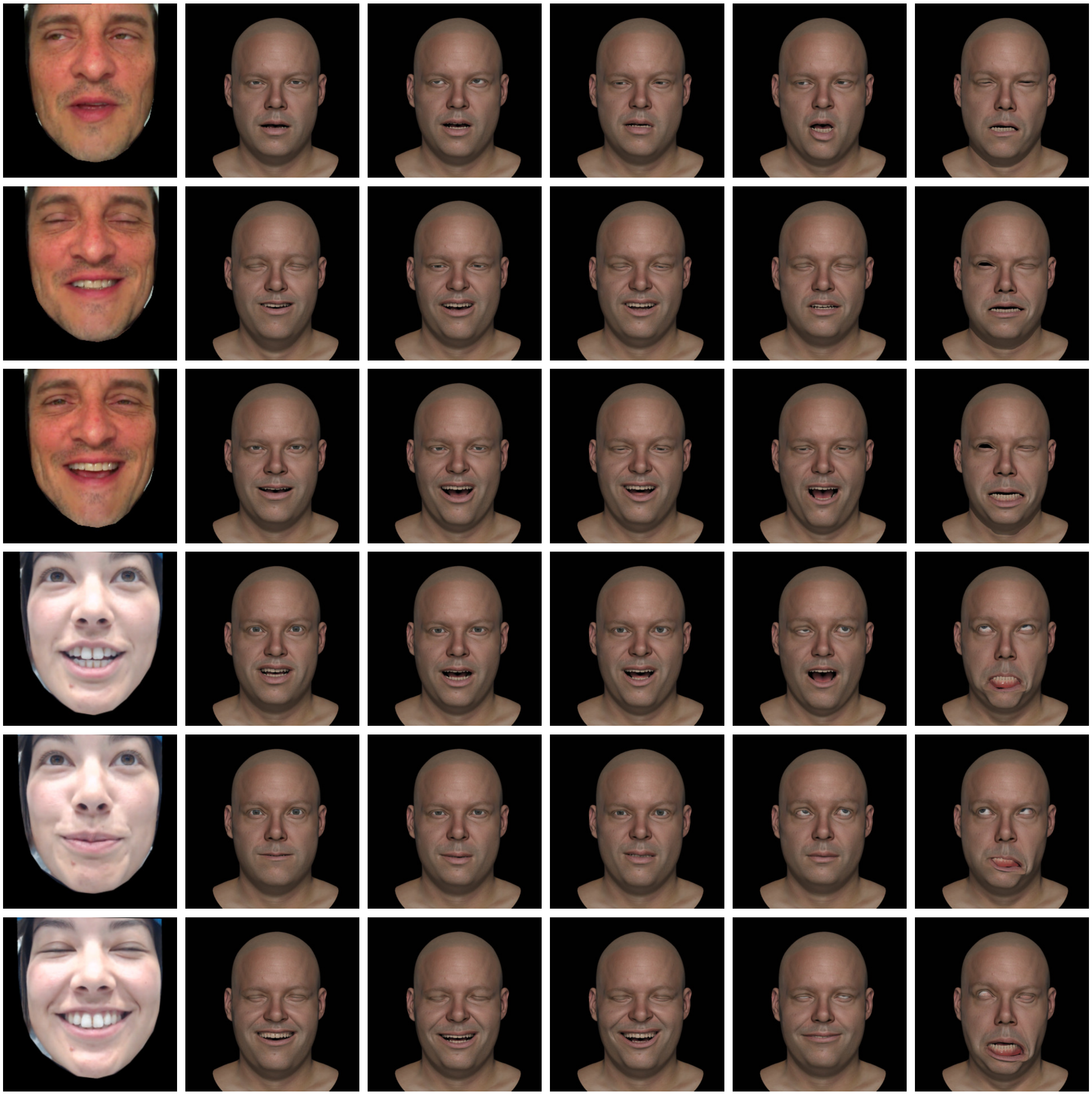
Table 6.2 shows numerical results of our experiment. In all cases, our method performs the best while the Neural Algorithm of Artistic Style produces the largest error. For the neural style method, these large errors can be attributed to the method transferring the textures in the images without warping the image to match the facial structure of the rendered blendshape model. Our blendweight optimization method assumes a fixed facial structure in the input image. The neural style method generally preserves the positions of facial features in the image, and our blendweight solver cannot compensate for these resulting differences in facial feature proportions. The error from the neural style images becomes even larger when the actor's facial geometry is significantly different from the blendshape model's geometry.

Figure 6.10 shows side-by-side examples of recorded frames with the solved pose from images of the different style transfer methods. The frames are selected from recordings of the male and female actors in our test set. In comparison to CycleGAN, we can see that poses solved from images generated through our style transfer method more closely match the the shape of the mouth as posed by the artist. However, in some cases, we can see that poses solved from the CycleGAN results more accurately match the eyes when they are fully closed in a recorded frame. Figure 6.11 shows a side-by-side comparison of the same recorded frames with the style transferred results. In this set of images, we also observe these subtle differences in the eyes and the mouth shapes.

6.7 Discussion

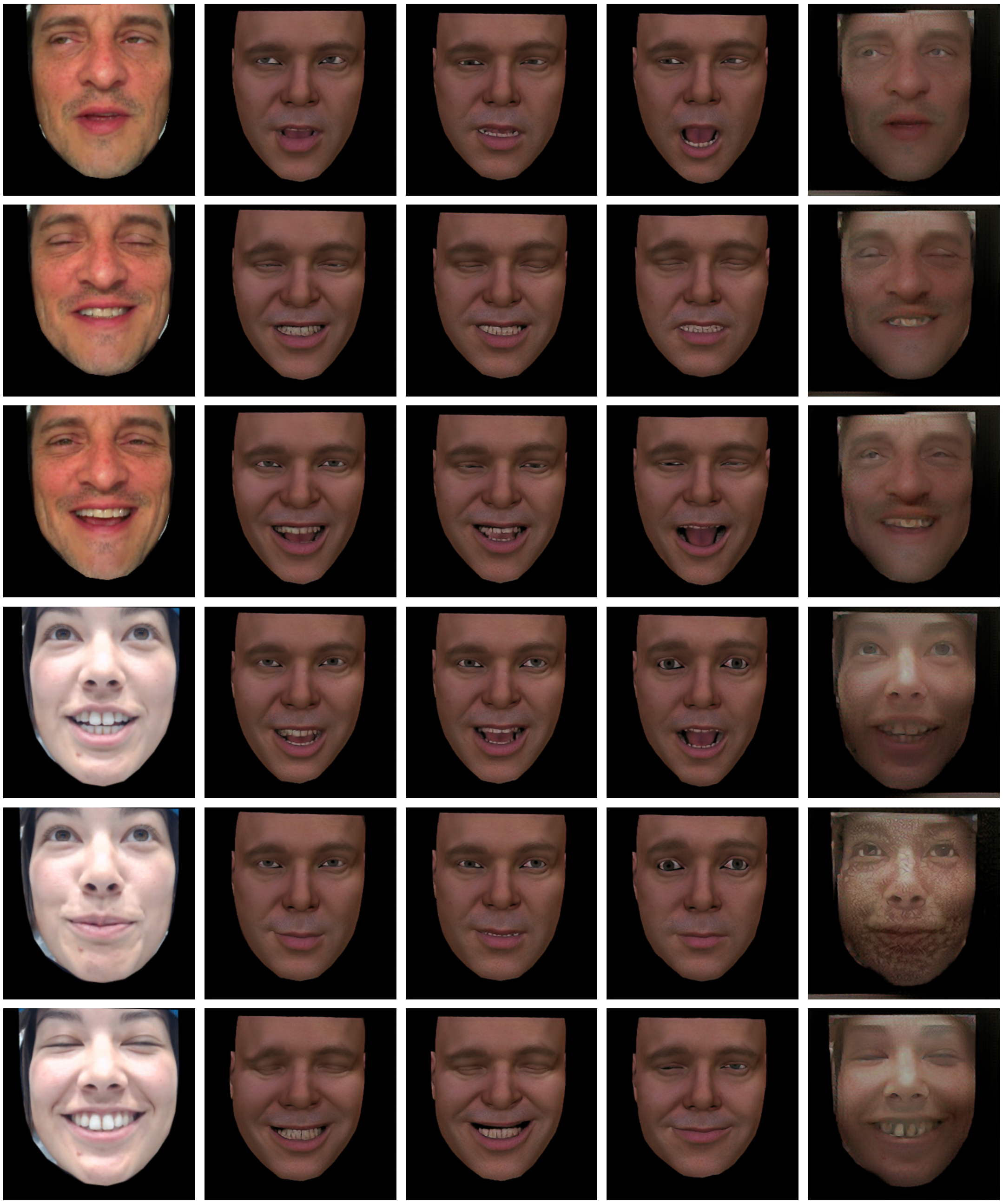
We have presented a method for facial performance capture consisting of a style transfer component and a blendweight optimization component. Our method does not require high-quality facial scans nor does it require that a recorded actor's appearance match that of the target facial rig. Instead, our method requires a blendshape model, example poses covering a full range of expression on the facial rig, and a training set of facial performances recorded from a helmet-mounted camera. As a result, any actor, regardless of appearance, can drive a facial rig through our performance capture method. Furthermore, the resulting animation is expressed only as a sequence of blendweights, which artists can use to edit an animation to their liking.

Because the blendweight optimizer seeks a solution that best matches the appearance of a face in a single frame, the resulting solved pose might use a different set of blendweights compared to what an artist might use. However, the appearance of an expression would be similar to what an artist would create. This difference is a consequence of a large set of blendshapes in a facial rig. Similar facial expressions can be created through different combinations of blendweights. Additionally, the optimizer considers a single frame in isolation when solving for blendweights. This approach loses the context of a facial performance that could provide vital cues as to which set of blendweights to activate. For example, in a recording of an actor talking, an animator can identify phonemes through the audio and



(a) Recorded (b) Artist Pose (c) Our Method (d) First Order (e) CycleGAN (f) Neural Style

Figure 6.10: Side-by-side comparison of the deformed blendshape model for various frames of the male actor and the female actor in the test set recordings. From left to right, the columns show the original recorded frame, the facial rig posed by an artist to match the recording, the posed rig generated from our style transfer method, the original first order motion model, CycleGAN, and the neural algorithm of artistic style.



(a) Recorded (b) Our Method (c) First Order (d) CycleGAN (e) Neural Style

Figure 6.11: Side-by-side comparison of style-transferred results for various frames of the male actor and female actor in the test set recordings. From left to right, the columns show the original recorded frame, results from our method, the original first order motion model, CycleGAN, and the neural algorithm for artistic style.

activate corresponding shapes to produce the desired mouth appearance. Our method does not utilize any contextual information in a recording and might match appearances through different blendshapes. Future work could incorporate additional information, such as audio, to help produce animations more similar to what an artist would create.

Although our method produces believable animations from recordings, future improvements could be made to both the style transfer method as well as the blendweight optimization method. Because our style transfer approach relies on image warping through landmarks, the method has difficulties reproducing pose-specific image features such as facial wrinkles caused by smiling. Furthermore, changes in lighting create challenges for the style transfer method because is reduces an image to a set of landmarks and matrix transformations. Changes in lighting or color cannot easily be encoded in the landmarks. We avoid this issue by ensuring that videos in our training data are captured under constant lighting as well as using a fixed lighting environment for the "rendered" style images in the training data. When performing style transfer on new videos, the actor does not need to be recorded under constant lighting conditions. Because the style transfer method cannot easily encode lighting variations in the landmarks and linear transformations, any changes in lighting in a new recording are lost. When an expression is transferred to the "rendered" style, the constant lighting is preserved, which is the behavior that we desire.

As presented, our blendweight optimization method does not consider the interior of the mouth when matching the facial rig to a performance. One major consequence of this formulation is that our method can produce an animation where the teeth penetrate the lips, which produces an undesirable visual appearance. Future work could explore methods that track teeth as well to prevent intersection between the teeth and lips of the model. Furthermore, our method does not track the tongue. Animating this part of the mouth could further improve the realism and accuracy of our facial performance capture system.

Finally, our method requires an actor's performance to be recorded from a helmet-mounted camera. This setup ensures that the actor's head does not rotate relative to the camera. Avoiding head rotations is necessary for our blendweight optimization method because our approach only estimates in-plane camera rotations and the camera's center of projection. It would be unable to handle any arbitrary camera rotation. Future work could explore tracking head rotations so that actors will not need to wear a helmet-mounted camera. However, the style transfer method would need to preserve the visual appearance of the facial geometry across head rotations. If the proportions are not preserved, then the blendweight optimizer will not produce accurate results because it cannot deform the geometry of the mesh outside of the given blendshapes.

Chapter 7

Conclusion

7.1 Summary of Contributions

Recent advances in machine learning, and deep learning in particular, have provided new tools to apply to problems in character animation. To address growing complexity of film-quality character rigs, I have proposed methods to compress the computational cost of evaluating mesh deformations. Previously, these types of rigs have been specialized to individual films. In some film studios, these characters might even be inaccessible in future projects due to incompatibilities with updated animation software.

However, my proposed methods offer a solution to these common challenges with character rigs. First, my approach reduces the computational complexity of character rigs so that they can be evaluated real-time on low-powered, consumer-quality devices. As a result, my approach can increase the level of complexity of characters in games and interactive applications. Second, because these rig approximations are implemented as neural networks, character rigs can now be expressed as a fixed-length set of model parameters. This representation provides a common format in which any character can be expressed. Because deep learning libraries and packages are readily available, applications that evaluate these models can easily be written. Once trained, these approximation models no longer depend on the original animation software used to create the rig. The model parameters can also be used as an archival method for characters authored on outdated rigging software. As another benefit, the models allow for character sharing between animation studios in cases where sharing their proprietary rigging software is an impracticality.

Additionally, I have proposed tools for character control and assisted animation authoring. As digital characters continue to grow in complexity, animators continually spend more effort to control additional character details to achieve an ever-increasing level realism and expressiveness. Although automated methods may never match the quality of artist-created animations, I have developed methods that allow artists to control coarse movements and deformations of a character so that they can focus their energy on finer-scale details that make an animation believable. The inverse kinematics methods allow artists to pose characters quickly through manipulation of a small set of control points rather than a long list of rig parameters. Additionally, my proposed facial performance capture method generates animations through the original controls of the rig. Afterwards, an animator can then use the familiar rig parameters to clean-up and fine-tune the performance. Finally, my animation synthesis approach aims to extend the lifetime of film-quality animation. Artists spend a significant amount of effort to produce character motions for a movie. As a way to reuse these high-quality animations, I have proposed a method to learn from these examples and produce controllable facial animations that follow a similar artistic style of the original animations.

7.2 Limitations and Future Work

My proposed research has shown promising results, but there are always opportunities for improvements and extensions. In each chapter, I have listed limitations specific to each project separately. Here, I will discuss limitations and future work relating to my proposed research when considered as a whole.

The rig approximations for both the body and the face operate only on a single pose to compute mesh deformations. When posing characters, animators typically work with rigs that operate on single-frame poses and exclude time-dependent motions such as cloth and hair simulations. These types of simulations are often expensive to compute, but do provide an important level of detail and realism to an animation. Incorporating these simulations in a deformation approximation would require extending the models to handle temporal data. Furthermore, collisions with the cloth, hair, and body meshes could pose additional challenges for a real-time approximation method.

In addition to improving rig evaluation times, an approximation method has the potential to help artists with the character rigging process. As demonstrated with the facial approximation method, the deformations can be transferred to an entirely different facial mesh. However, this is not possible with my proposed body approximation because it relies on a fixed mesh topology for the character. Furthermore, the approximation models learn deformations specific to the rig on which they are trained. If the models were transferred to a new character with significantly different body and facial proportions, the approximations would produce undesirable results. For example, if a new character's eyes were twice as large as the character on which an approximation model is trained, then a fully closed eye on the original character would produce only half-closed eyes on the new character. Challenges like this need to be addressed in order to transfer deformations learned from one rig onto a character mesh with different topology and proportions.

For animations, artists could benefit the most from methods that generate animation, which provides a starting point for the artist. Thus, artists can spend their efforts focusing on fine-scale stylistic details that are expected in high-quality animations. The facial performance capture method that I have proposed is capable of providing this starting point for artists because it produces animations only through the existing rig controls of the character. However, my approach, like similar related solutions, prioritizes matching the physical appearance and movements of a recorded actor. In some cases, certain desired emotional expressions might not be captured in the generated animation, and an artist might choose to rework the animation from scratch if the performance capture version is significantly different from the desired look.

In contrast, my animation synthesis method learns to generate new motions from existing artist-created examples. Thus, producing an animation segment with a desired emotional expression is straightforward. However, the high-level controls of the system do not provide fine-scale influence over the synthesized motion. For example, the method would struggle to produce a timed sequence of visemes for a talking character. If combined with the facial performance capture method, this approach could provide physical emotional expressions and stylistic movements for characters while the performance capture approach provides controls for physical movement of facial features such as eyes and mouth. The system would give an artist more control over the generated animation and would not be limited to the exact physical motion of a recorded performance. This combined system could either detect the emotional expression from a recording or allow an artist to script times when certain emotions should be triggered, which would help generate animations that are closer to an artist's desired result.

Finally, in each chapter, I have considered either body or facial motion in isolation. Nevertheless, a fully immersive performance would need to consider both body language and facial expressions together. Animations of talking heads can provide some sense of realism. However, combining facial motion with context-specific arm gestures and body language would further increase the quality of an animation beyond current capabilities when controlling the face separately. Future work should address animating and synthesizing full body and facial motions as a whole in order to create more compelling and believable experiences.

Bibliography

[1] S. Andrews and P.G. Kry. “Goal directed multi-finger manipulation: Control policies and analysis”. In: *Computers & Graphics* 37.7 (2013), pp. 830–839. ISSN: 0097-8493.

[2] Okan Arikan and D. A. Forsyth. “Interactive Motion Generation from Examples”. In: *ACM Trans. Graph.* 21.3 (July 2002), pp. 483–490. ISSN: 0730-0301.

[3] Andreas Aristidou et al. “Inverse Kinematics Techniques in Computer Graphics: A Survey”. In: *Computer Graphics Forum* 37 (Sept. 2018), pp. 35–58. DOI: 10.1111/cgf.13310.

[4] Stephen W. Bailey, Martin Watt, and James F. O’Brien. “Repurposing Hand Animation for Interactive Applications”. In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* SCA ’16. Zurich, Switzerland: Eurographics Association, 2016, pp. 97–106. ISBN: 9783905674613.

[5] Stephen W. Bailey et al. “Fast and Deep Deformation Approximations”. In: *ACM Trans. Graph.* 37.4 (July 2018), 119:1–119:12. ISSN: 0730-0301. DOI: 10.1145/3197517.3201300. URL: http://doi.acm.org/10.1145/3197517.3201300.

[6] Stephen W. Bailey et al. “Fast and Deep Facial Deformations”. In: *ACM Trans. Graph.* 39.4 (July 2020). ISSN: 0730-0301. DOI: 10.1145/3386569.3392397. URL: https://doi.org/10.1145/3386569.3392397.

[7] Alan H. Barr. “Global and Local Deformations of Solid Primitives”. In: *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH ’84. New York, NY, USA: ACM, 1984, pp. 21–30. ISBN: 0-89791-138-5.

[8] J. Baumgarte. “Stabilization of constraints and integrals of motion in dynamical systems”. In: *Computer Methods in Applied Mechanics and Engineering* 1 (June 1972), pp. 1–16.

[9] Vadim Besedin et al. *Ray Character Rig.* CGTarian Animation & VFX Online School. 2018. URL: https://www.cgtarian.com/maya-character-rigs/download-free-3d-character-ray.html.

[10] Kiran S. Bhat et al. “High Fidelity Facial Animation Capture and Retargeting with Contours”. In: *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’13. Anaheim, California: ACM, 2013, pp. 7–14. ISBN: 978-1-4503-2132-7. DOI: `10.1145/2485895.2485915`. URL: `http://doi.acm.org/10.1145/2485895.2485915`.

[11] Bernd Bickel et al. “Pose-space Animation and Transfer of Facial Details”. In: *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’08. Dublin, Ireland: Eurographics Association, 2008, pp. 57–66. ISBN: 978-3-905674-10-1. URL: `http://dl.acm.org/citation.cfm?id=1632592.1632602`.

[12] Volker Blanz and Thomas Vetter. “A Morphable Model for the Synthesis of 3D Faces”. In: *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’99. USA: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 187–194. ISBN: 0201485605. DOI: `10.1145/311535.311556`. URL: `https://doi.org/10.1145/311535.311556`.

[13] James Booth et al. “Large Scale 3D Morphable Models”. In: *Int. J. Comput. Vision* 126.2–4 (Apr. 2018), pp. 233–254. ISSN: 0920-5691. DOI: `10.1007/s11263-017-1009-7`. URL: `https://doi.org/10.1007/s11263-017-1009-7`.

[14] Davide Boscaini et al. “Learning Shape Correspondence with Anisotropic Convolutional Neural Networks”. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS’16. Barcelona, Spain: Curran Associates Inc., 2016, pp. 3197–3205. ISBN: 9781510838819.

[15] Sofien Bouaziz, Yangang Wang, and Mark Pauly. “Online Modeling for Realtime Facial Animation”. In: *ACM Trans. Graph.* 32.4 (July 2013), 40:1–40:10. ISSN: 0730-0301.

[16] Sofien Bouaziz, Yangang Wang, and Mark Pauly. “Online Modeling for Realtime Facial Animation”. In: *ACM Trans. Graph.* 32.4 (July 2013), 40:1–40:10. ISSN: 0730-0301. DOI: `10.1145/2461912.2461976`. URL: `http://doi.acm.org/10.1145/2461912.2461976`.

[17] Derek Bradley et al. “High Resolution Passive Facial Performance Capture”. In: *ACM Trans. Graph.* 29.4 (July 2010), 41:1–41:10. ISSN: 0730-0301. DOI: `10.1145/1778765.1778778`. URL: `http://doi.acm.org/10.1145/1778765.1778778`.

[18] Matthew Brand. “Voice Puppetry”. In: *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’99. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 21–28. ISBN: 0-201-48560-5.

[19] Matthew Brand and Aaron Hertzmann. “Style Machines”. In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 183–192. ISBN: 1-58113-208-5.

[20] Ian Buck et al. “Performance-Driven Hand-Drawn Animation”. In: *NPAR 2000 : First International Symposium on Non Photorealistic Animation and Rendering*. June 2000, pp. 101–108.

[21] Samuel R. Buss and Jin-Su Kim. “Selectively Damped Least Squares for Inverse Kinematics”. In: *Journal of Graphics Tools* 10.3 (2005), pp. 37–49. DOI: 10.1080/2151237X.2005.10129202. eprint: https://doi.org/10.1080/2151237X.2005.10129202. URL: https://doi.org/10.1080/2151237X.2005.10129202.

[22] Chen Cao et al. “3D Shape Regression for Real-time Facial Animation”. In: *ACM Trans. Graph.* 32.4 (July 2013), 41:1–41:10. ISSN: 0730-0301.

[23] Chen Cao et al. “FaceWarehouse: A 3D Facial Expression Database for Visual Computing”. In: *IEEE Transactions on Visualization and Computer Graphics* 20.3 (Mar. 2014), pp. 413–425. ISSN: 1077-2626. DOI: 10.1109/TVCG.2013.249. URL: https://doi.org/10.1109/TVCG.2013.249.

[24] Kaidi Cao, Jing Liao, and Lu Yuan. *CariGANs: Unpaired Photo-to-Caricature Translation*. 2018.

[25] Steve Capell et al. “Interactive Skeleton-driven Dynamic Deformations”. In: *ACM Trans. Graph.* 21.3 (July 2002), pp. 586–593. ISSN: 0730-0301. DOI: 10.1145/566654.566622. URL: http://doi.acm.org/10.1145/566654.566622.

[26] Jin-xiang Chai, Jing Xiao, and Jessica Hodgins. “Vision-based Control of 3D Facial Animation”. In: *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’03. San Diego, California: Eurographics Association, 2003, pp. 193–206. ISBN: 1-58113-659-5.

[27] Yunjey Choi et al. “StarGAN v2: Diverse Image Synthesis for Multiple Domains”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[28] Yunjey Choi et al. “StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation”. In: *CoRR* abs/1711.09020 (2017). arXiv: 1711.09020. URL: http://arxiv.org/abs/1711.09020.

[29] Matthew Cong et al. “Fully Automatic Generation of Anatomical Face Simulation Models”. In: *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. SCA ’15. Los Angeles, California: ACM, 2015, pp. 175–183. ISBN: 978-1-4503-3496-9. DOI: 10.1145/2786784.2786786. URL: http://doi.acm.org/10.1145/2786784.2786786.

[30] Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. “Robust Task-based Control Policies for Physics-based Characters”. In: *ACM Trans. Graph.* 28.5 (Dec. 2009), 170:1–170:9. ISSN: 0730-0301.

[31] Edilson De Aguiar et al. "Automatic Conversion of Mesh Animations into Skeleton-based Animations". In: *Computer Graphics Forum* 27.2 (2008), pp. 389–397. ISSN: 1467-8659. DOI: 10.1111/j.1467-8659.2008.01136.x. URL: http://dx.doi.org/10.1111/j.1467-8659.2008.01136.x.

[32] Marc Escher and Nadia Magnenat Thalmann. "Automatic 3D Cloning and Real-Time Animation of a Human Face". In: *Proceedings of the Computer Animation.* CA '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 58–. ISBN: 0-8186-7984-0.

[33] Wei-Wen Feng, Byung-Uck Kim, and Yizhou Yu. "Real-time Data Driven Deformation Using Kernel Canonical Correlation Analysis". In: *ACM Trans. Graph.* 27.3 (Aug. 2008), 91:1–91:9. ISSN: 0730-0301. DOI: 10.1145/1360612.1360690. URL: http://doi.acm.org/10.1145/1360612.1360690.

[34] Yao Feng et al. "Joint 3D Face Reconstruction and Dense Alignment with Position Map Regression Network". In: *CoRR* abs/1803.07835 (2018). arXiv: 1803.07835. URL: http://arxiv.org/abs/1803.07835.

[35] Zhen-Hua Feng et al. "Wing Loss for Robust Facial Landmark Localisation with Convolutional Neural Networks". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2017), pp. 2235–2245.

[36] Pablo Garrido et al. "Corrective 3D Reconstruction of Lips from Monocular Video". In: *ACM Trans. Graph.* 35.6 (Nov. 2016), 219:1–219:11. ISSN: 0730-0301. DOI: 10.1145/2980179.2982419. URL: http://doi.acm.org/10.1145/2980179.2982419.

[37] Pablo Garrido et al. "Reconstruction of Personalized 3D Face Rigs from Monocular Video". In: *ACM Trans. Graph.* 35.3 (May 2016). ISSN: 0730-0301. DOI: 10.1145/2890493. URL: https://doi.org/10.1145/2890493.

[38] Pablo Garrido et al. "Reconstruction of Personalized 3D Face Rigs from Monocular Video". In: *ACM Trans. Graph.* 35.3 (May 2016), 28:1–28:15. ISSN: 0730-0301. DOI: 10.1145/2890493. URL: http://doi.acm.org/10.1145/2890493.

[39] L. A. Gatys, A. S. Ecker, and M. Bethge. "Image Style Transfer Using Convolutional Neural Networks". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2016, pp. 2414–2423.

[40] Michael Girard and A. A. Maciejewski. "Computational Modeling for the Computer Animation of Legged Figures". In: *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH '85. New York, NY, USA: ACM, 1985, pp. 263–270. ISBN: 0-89791-166-0. DOI: 10.1145/325334.325244. URL: http://doi.acm.org/10.1145/325334.325244.

[41] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 315–323. URL: `http://proceedings.mlr.press/v15/glorot11a.html`.

[42] D. Goldfarb and A. Idnani. "A Numerically Stable Dual Method for Solving Strictly Convex Quadratic Programs". In: *Math. Program.* 27.1 (Sept. 1983), pp. 1–33. ISSN: 0025-5610. DOI: `10.1007/BF02591962`. URL: `https://doi.org/10.1007/BF02591962`.

[43] Keith Grochow et al. "Style-based Inverse Kinematics". In: *ACM Trans. Graph.* 23.3 (Aug. 2004), pp. 522–531. ISSN: 0730-0301. DOI: `10.1145/1015706.1015755`. URL: `http://doi.acm.org/10.1145/1015706.1015755`.

[44] Keith Grochow et al. "Style-based Inverse Kinematics". In: *ACM Trans. Graph.* 23.3 (Aug. 2004), pp. 522–531. ISSN: 0730-0301.

[45] Fabian Hahn et al. "Efficient Simulation of Secondary Motion in Rig-space". In: *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '13. Anaheim, California: ACM, 2013, pp. 165–171. ISBN: 978-1-4503-2132-7. DOI: `10.1145/2485895.2485918`. URL: `http://doi.acm.org/10.1145/2485895.2485918`.

[46] Fabian Hahn et al. "Efficient Simulation of Secondary Motion in Rig-space". In: *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '13. Anaheim, California: ACM, 2013, pp. 165–171. ISBN: 978-1-4503-2132-7.

[47] Fabian Hahn et al. "Rig-space Physics". In: *ACM Trans. Graph.* 31.4 (July 2012), 72:1–72:8. ISSN: 0730-0301. DOI: `10.1145/2185520.2185568`. URL: `http://doi.acm.org/10.1145/2185520.2185568`.

[48] Rana Hanocka et al. "MeshCNN: A Network with an Edge". In: *ACM Trans. Graph.* 38.4 (July 2019). ISSN: 0730-0301. DOI: `10.1145/3306346.3322959`. URL: `https://doi.org/10.1145/3306346.3322959`.

[49] Nils Hasler et al. "Learning Skeletons for Shape and Pose". In: *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '10. Washington, D.C.: ACM, 2010, pp. 23–30. ISBN: 978-1-60558-939-8. DOI: `10.1145/1730804.1730809`. URL: `http://doi.acm.org/10.1145/1730804.1730809`.

[50] Jim Hejl. "Hardware Skinning with Quaternions". In: *Game Programming Gems 4*. Ed. by Andrew Kirmse. Charles River Media, 2004, pp. 487–495.

[51] Darren Hendler et al. "Avengers: Capturing Thanos's Complex Face". In: *ACM SIGGRAPH 2018 Talks*. SIGGRAPH '18. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2018. ISBN: 9781450358200. DOI: `10.1145/3214745.3214766`. URL: `https://doi.org/10.1145/3214745.3214766`.

[52] Daniel Holden, Taku Komura, and Jun Saito. "Phase-functioned Neural Networks for Character Control". In: *ACM Trans. Graph.* 36.4 (July 2017), 42:1–42:13. ISSN: 0730-0301. DOI: `10.1145/3072959.3073663`. URL: `http://doi.acm.org/10.1145/3072959.3073663`.

[53] Daniel Holden, Jun Saito, and Taku Komura. "A Deep Learning Framework for Character Motion Synthesis and Editing". In: *ACM Trans. Graph.* 35.4 (July 2016), 138:1–138:11. ISSN: 0730-0301. DOI: `10.1145/2897824.2925975`. URL: `http://doi.acm.org/10.1145/2897824.2925975`.

[54] Daniel Holden, Jun Saito, and Taku Komura. "Learning an Inverse Rig Mapping for Character Animation". In: *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. SCA '15. Los Angeles, California: ACM, 2015, pp. 165–173. ISBN: 978-1-4503-3496-9.

[55] Daniel Holden, Jun Saito, and Taku Komura. "Learning Inverse Rig Mappings by Nonlinear Regression". In: *IEEE Transactions on Visualization and Computer Graphics* 23.3 (Mar. 2017), pp. 1167–1178. ISSN: 1077-2626. DOI: `10.1109/TVCG.2016.2628036`. URL: `https://doi.org/10.1109/TVCG.2016.2628036`.

[56] Kurt Hornik. "Approximation Capabilities of Multilayer Feedforward Networks". In: *Neural Netw.* 4.2 (Mar. 1991), pp. 251–257. ISSN: 0893-6080. DOI: `10.1016/0893-6080(91)90009-T`. URL: `http://dx.doi.org/10.1016/0893-6080(91)90009-T`.

[57] Alexandru Eugen Ichim, Sofien Bouaziz, and Mark Pauly. "Dynamic 3D Avatar Creation from Hand-held Video Input". In: *ACM Trans. Graph.* 34.4 (July 2015), 45:1–45:14. ISSN: 0730-0301. DOI: `10.1145/2766974`. URL: `http://doi.acm.org/10.1145/2766974`.

[58] Alexandru-Eugen Ichim et al. "Phace: Physics-based Face Modeling and Animation". In: *ACM Trans. Graph.* 36.4 (July 2017), 153:1–153:14. ISSN: 0730-0301. DOI: `10.1145/3072959.3073664`. URL: `http://doi.acm.org/10.1145/3072959.3073664`.

[59] Phillip Isola et al. "Image-to-Image Translation with Conditional Adversarial Networks". In: July 2017, pp. 5967–5976. DOI: `10.1109/CVPR.2017.632`.

[60] Doug L. James and Christopher D. Twigg. "Skinning Mesh Animations". In: *ACM Trans. Graph.* 24.3 (July 2005), pp. 399–407. ISSN: 0730-0301. DOI: `10.1145/1073204.1073206`. URL: `http://doi.acm.org/10.1145/1073204.1073206`.

[61] Ning Jin et al. "A Pixel-Based Framework for Data-Driven Clothing". In: *CoRR* abs/1812.01677 (2018). arXiv: `1812.01677`. URL: `http://arxiv.org/abs/1812.01677`.

[62] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. "Perceptual losses for real-time style transfer and super-resolution". In: *European Conference on Computer Vision*. 2016.

[63] Pushkar Joshi et al. "Harmonic Coordinates for Character Articulation". In: *ACM Trans. Graph.* 26.3 (July 2007). ISSN: 0730-0301. DOI: `10.1145/1276377.1276466`. URL: `http://doi.acm.org/10.1145/1276377.1276466`.

[64] Tao Ju et al. “Reusable Skinning Templates Using Cage-based Deformations”. In: *ACM Trans. Graph.* 27.5 (Dec. 2008), 122:1–122:10. ISSN: 0730-0301. DOI: 10.1145/1409060.1409075. URL: http://doi.acm.org/10.1145/1409060.1409075.

[65] L. Kavan, P.-P. Sloan, and C. O Sullivan. “Fast and Efficient Skinning of Animated Meshes”. In: *Computer Graphics Forum* (2010). ISSN: 1467-8659. DOI: 10.1111/j.1467-8659.2009.01602.x.

[66] Ladislav Kavan, Steven Collins, and Carol O’Sullivan. “Automatic Linearization of Nonlinear Skinning”. In: *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*. I3D ’09. Boston, Massachusetts: ACM, 2009, pp. 49–56. ISBN: 978-1-60558-429-4. DOI: 10.1145/1507149.1507157. URL: http://doi.acm.org/10.1145/1507149.1507157.

[67] Ladislav Kavan and Jiří Žára. “Spherical Blend Skinning: A Real-time Deformation of Articulated Models”. In: *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*. I3D ’05. Washington, District of Columbia: ACM, 2005, pp. 9–16. ISBN: 1-59593-013-2. DOI: 10.1145/1053427.1053429. URL: http://doi.acm.org/10.1145/1053427.1053429.

[68] Ladislav Kavan et al. “Skinning Arbitrary Deformations”. In: *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*. I3D ’07. Seattle, Washington: ACM, 2007, pp. 53–60. ISBN: 978-1-59593-628-8. DOI: 10.1145/1230100.1230109. URL: http://doi.acm.org/10.1145/1230100.1230109.

[69] Hyeongwoo Kim et al. “Deep Video Portraits”. In: *ACM Trans. Graph.* 37.4 (July 2018). ISSN: 0730-0301. DOI: 10.1145/3197517.3201283. URL: https://doi.org/10.1145/3197517.3201283.

[70] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2014). URL: http://arxiv.org/abs/1412.6980.

[71] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations* (Dec. 2014).

[72] J. Kleiser. “A fast, efficient, accurate way to represent the human face.” In: *SIGGRAPH ’89 Course Notes 22: State of the Art in Facial Animation*. 1989.

[73] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. “Motion Graphs”. In: *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’02. San Antonio, Texas: ACM, 2002, pp. 473–482. ISBN: 1-58113-521-1.

[74] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. “Motion Graphs”. In: *ACM Trans. Graph.* 21.3 (July 2002), pp. 473–482. ISSN: 0730-0301. DOI: 10.1145/566654.566605. URL: http://doi.acm.org/10.1145/566654.566605.

[75] Paul G. Kry, Doug L. James, and Dinesh K. Pai. “EigenSkin: Real Time Large Deformation Character Skinning in Hardware”. In: *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’02. San Antonio, Texas: ACM, 2002, pp. 153–159. ISBN: 1-58113-573-4. DOI: `10.1145/545261.545286`. URL: `http://doi.acm.org/10.1145/545261.545286`.

[76] Tsuneya Kurihara and Natsuki Miyata. “Modeling Deformable Human Hands from Medical Images”. In: *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’04. Grenoble, France: Eurographics Association, 2004, pp. 355–363. ISBN: 3-905673-14-2. DOI: `10.1145/1028523.1028571`. URL: `http://dx.doi.org/10.1145/1028523.1028571`.

[77] Zorah Lähner, Daniel Cremers, and Tony Tung. “DeepWrinkles: Accurate and Realistic Clothing Modeling”. In: *CoRR* abs/1808.03417 (2018). arXiv: `1808.03417`. URL: `http://arxiv.org/abs/1808.03417`.

[78] Samuli Laine et al. “Production-level Facial Performance Capture Using Deep Convolutional Neural Networks”. In: *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. SCA ’17. Los Angeles, California: ACM, 2017, 10:1–10:10. ISBN: 978-1-4503-5091-4. DOI: `10.1145/3099564.3099581`. URL: `http://doi.acm.org/10.1145/3099564.3099581`.

[79] Manfred Lau, Ziv Bar-Joseph, and James Kuffner. “Modeling Spatial and Temporal Variation in Motion Data”. In: *ACM Trans. Graph.* 28.5 (Dec. 2009), 171:1–171:10. ISSN: 0730-0301.

[80] Manfred Lau et al. “Face Poser: Interactive Modeling of 3D Facial Expressions Using Facial Priors”. In: *ACM Trans. Graph.* 29.1 (Dec. 2009), 3:1–3:17. ISSN: 0730-0301.

[81] Neil D. Lawrence. “Hierarchical Gaussian process latent variable models”. In: *In International Conference in Machine Learning*. 2007.

[82] Neil D. Lawrence. “The Gaussian process latent variable model”. In: *Technical Report no CS-06-05* (2006).

[83] Neil D. Lawrence and Joaquin Quiñonero-Candela. “Local Distance Preservation in the GP-LVM Through Back Constraints”. In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML ’06. Pittsburgh, Pennsylvania, USA: ACM, 2006, pp. 513–520. ISBN: 1-59593-383-2.

[84] Binh Huy Le and Zhigang Deng. “Robust and Accurate Skeletal Rigging from Mesh Sequences”. In: *ACM Trans. Graph.* 33.4 (July 2014), 84:1–84:10. ISSN: 0730-0301. DOI: `10.1145/2601097.2601161`. URL: `http://doi.acm.org/10.1145/2601097.2601161`.

[85] Binh Huy Le and Zhigang Deng. “Smooth Skinning Decomposition with Rigid Bones”. In: *ACM Trans. Graph.* 31.6 (Nov. 2012), 199:1–199:10. ISSN: 0730-0301. DOI: `10.1145/2366145.2366218`. URL: `http://doi.acm.org/10.1145/2366145.2366218`.

[86] Binh Huy Le and Zhigang Deng. “Two-layer Sparse Compression of Dense-weight Blend Skinning”. In: *ACM Trans. Graph.* 32.4 (July 2013), 124:1–124:10. ISSN: 0730-0301. DOI: 10.1145/2461912.2461949. URL: http://doi.acm.org/10.1145/2461912.2461949.

[87] Christian Ledig et al. “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network”. In: July 2017, pp. 105–114. DOI: 10.1109/CVPR.2017.19.

[88] Jehee Lee et al. “Interactive Control of Avatars Animated with Human Motion Data”. In: *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’02. San Antonio, Texas: ACM, 2002, pp. 491–500. ISBN: 1-58113-521-1.

[89] Sung-Hee Lee, Eftychios Sifakis, and Demetri Terzopoulos. “Comprehensive Biomechanical Modeling and Simulation of the Upper Body”. In: *ACM Trans. Graph.* 28.4 (Sept. 2009), 99:1–99:17. ISSN: 0730-0301. DOI: 10.1145/1559755.1559756. URL: http://doi.acm.org/10.1145/1559755.1559756.

[90] Yongjoon Lee, Seong Jae Lee, and Zoran Popović. “Compact Character Controllers”. In: *ACM Trans. Graph.* 28.5 (Dec. 2009), 169:1–169:8. ISSN: 0730-0301.

[91] Sergey Levine et al. “Continuous Character Control with Low-Dimensional Embeddings”. In: *ACM Transactions on Graphics* 31.4 (2012), p. 28.

[92] Sergey Levine et al. “Continuous Character Control with Low-dimensional Embeddings”. In: *ACM Trans. Graph.* 31.4 (July 2012), 28:1–28:10. ISSN: 0730-0301. DOI: 10.1145/2185520.2185524. URL: http://doi.acm.org/10.1145/2185520.2185524.

[93] J. P. Lewis and K. Anjyo. “Direct Manipulation Blendshapes”. In: *IEEE Computer Graphics and Applications* 30.4 (July 2010), pp. 42–50. ISSN: 1558-1756. DOI: 10.1109/MCG.2010.41.

[94] J. P. Lewis, Matt Cordner, and Nickson Fong. “Pose Space Deformation: A Unified Approach to Shape Interpolation and Skeleton-driven Deformation”. In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 165–172. ISBN: 1-58113-208-5. DOI: 10.1145/344779.344862. URL: http://dx.doi.org/10.1145/344779.344862.

[95] J. P. Lewis, Matt Cordner, and Nickson Fong. “Pose Space Deformation: A Unified Approach to Shape Interpolation and Skeleton-driven Deformation”. In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 165–172. ISBN: 1-58113-208-5.

[96] J. P. Lewis and F. I. Parke. “Automated Lip-synch and Speech Synthesis for Character Animation”. In: *SIGCHI Bull.* 17.SI (May 1986), pp. 143–147. ISSN: 0736-6906.

[97] Hao Li, Thibaut Weise, and Mark Pauly. "Example-based Facial Rigging". In: *ACM SIGGRAPH 2010 Papers*. SIGGRAPH '10. Los Angeles, California: ACM, 2010, 32:1–32:6. ISBN: 978-1-4503-0210-4. DOI: `10.1145/1833349.1778769`. URL: `http://doi.acm.org/10.1145/1833349.1778769`.

[98] Hao Li et al. "Realtime Facial Animation with On-the-fly Correctives". In: *ACM Trans. Graph.* 32.4 (July 2013), 42:1–42:10. ISSN: 0730-0301. DOI: `10.1145/2461912.2462019`. URL: `http://doi.acm.org/10.1145/2461912.2462019`.

[99] Or Litany et al. "Deformable Shape Completion with Graph Convolutional Autoencoders". In: *CoRR* abs/1712.00268 (2017). arXiv: `1712.00268`. URL: `http://arxiv.org/abs/1712.00268`.

[100] Wan-Yen Lo and Matthias Zwicker. "Real-time Planning for Parameterized Human Motion". In: *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '08. Dublin, Ireland: Eurographics Association, 2008, pp. 29–38. ISBN: 978-3-905674-10-1.

[101] Stephen Lombardi et al. "Deep Appearance Models for Face Rendering". In: *ACM Trans. Graph.* 37.4 (July 2018). ISSN: 0730-0301. DOI: `10.1145/3197517.3201401`. URL: `https://doi.org/10.1145/3197517.3201401`.

[102] N. Magnenat-Thalmann, R. Laperrière, and D. Thalmann. "Joint-dependent Local Deformations for Hand Animation and Object Grasping". In: *Proceedings on Graphics Interface '88*. Edmonton, Alberta, Canada: Canadian Information Processing Society, 1988, pp. 26–33. URL: `http://dl.acm.org/citation.cfm?id=102313.102317`.

[103] N. Magnenat-Thalmann, R. Laperrière, and D. Thalmann. "Joint-dependent Local Deformations for Hand Animation and Object Grasping". In: *Proceedings on Graphics Interface '88*. Edmonton, Alberta, Canada: Canadian Information Processing Society, 1988, pp. 26–33.

[104] Joe Mancewicz et al. "Delta Mush: Smoothing Deformations While Preserving Detail". In: *Proceedings of the Fourth Symposium on Digital Production*. DigiPro '14. Vancouver, British Columbia, Canada: ACM, 2014, pp. 7–11. ISBN: 978-1-4503-3044-2. DOI: `10.1145/2633374.2633376`. URL: `http://doi.acm.org/10.1145/2633374.2633376`.

[105] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[106] Jonathan Masci et al. "Geodesic Convolutional Neural Networks on Riemannian Manifolds". In: *Proceedings of the 2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*. ICCVW '15. USA: IEEE Computer Society, 2015, pp. 832–840. ISBN: 9781467397117. DOI: `10.1109/ICCVW.2015.112`. URL: `https://doi.org/10.1109/ICCVW.2015.112`.

[107] Tim McLaughlin, Larry Cutler, and David Coleman. “Character Rigging, Deformations, and Simulations in Film and Game Production”. In: *ACM SIGGRAPH 2011 Courses*. SIGGRAPH ’11. Vancouver, British Columbia, Canada: ACM, 2011, 5:1–5:18. ISBN: 978-1-4503-0967-7. DOI: 10.1145/2037636.2037641. URL: http://doi.acm.org/10.1145/2037636.2037641.

[108] Jianyuan Min and Jinxiang Chai. “Motion Graphs++: A Compact Generative Model for Semantic Motion Analysis and Synthesis”. In: *ACM Trans. Graph.* 31.6 (Nov. 2012), 153:1–153:12. ISSN: 0730-0301.

[109] Alex Mohr and Michael Gleicher. “Building Efficient, Accurate Character Skins from Examples”. In: *ACM Trans. Graph.* 22.3 (July 2003), pp. 562–568. ISSN: 0730-0301. DOI: 10.1145/882262.882308. URL: http://doi.acm.org/10.1145/882262.882308.

[110] Lucio Moser, Darren Hendler, and Doug Roble. “Masquerade: Fine-Scale Details for Head-Mounted Camera Motion Capture Data”. In: *ACM SIGGRAPH 2017 Talks*. SIGGRAPH ’17. Los Angeles, California: Association for Computing Machinery, 2017. ISBN: 9781450350082. DOI: 10.1145/3084363.3085086. URL: https://doi.org/10.1145/3084363.3085086.

[111] Uldarico Muico et al. “Contact-aware Nonlinear Control of Dynamic Characters”. In: *ACM SIGGRAPH 2009 Papers*. SIGGRAPH ’09. New Orleans, Louisiana: ACM, 2009, 81:1–81:9. ISBN: 978-1-60558-726-4.

[112] Tomohiko Mukai and Shigeru Kuriyama. “Efficient Dynamic Skinning with Low-rank Helper Bone Controllers”. In: *ACM Trans. Graph.* 35.4 (July 2016), 36:1–36:11. ISSN: 0730-0301. DOI: 10.1145/2897824.2925905. URL: http://doi.acm.org/10.1145/2897824.2925905.

[113] Tomohiko Mukai and Shigeru Kuriyama. “Geostatistical Motion Interpolation”. In: *ACM SIGGRAPH 2005 Papers*. SIGGRAPH ’05. Los Angeles, California: ACM, 2005, pp. 1062–1070. ISBN: 978-1-4503-7825-3. DOI: 10.1145/1186822.1073313. URL: http://doi.acm.org/10.1145/1186822.1073313.

[114] Tomohiko Mukai and Shigeru Kuriyama. “Geostatistical Motion Interpolation”. In: *ACM SIGGRAPH 2005 Papers*. SIGGRAPH ’05. Los Angeles, California: ACM, 2005, pp. 1062–1070.

[115] Thomas Neumann et al. “Sparse Localized Deformation Components”. In: *ACM Trans. Graph.* 32.6 (Nov. 2013), 179:1–179:10. ISSN: 0730-0301. DOI: 10.1145/2508363.2508417. URL: http://doi.acm.org/10.1145/2508363.2508417.

[116] Frederic Ira Parke. “A Parametric Model for Human Faces.” AAI7508697. PhD thesis. 1974.

[117] Frederick I. Parke. “Computer Generated Animation of Faces”. In: *Proceedings of the ACM Annual Conference - Volume 1*. ACM ’72. Boston, Massachusetts, USA: ACM, 1972, pp. 451–457. ISBN: 978-1-4503-7491-0. DOI: 10.1145/800193.569955. URL: http://doi.acm.org/10.1145/800193.569955.

[118] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. 3rd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2016. ISBN: 0128006455.

[119] Sven Pohle et al. “DreamWorks Animation Facial Motion and Deformation System”. In: *Proceedings of the 2015 Symposium on Digital Production*. DigiPro ’15. Los Angeles, California: Association for Computing Machinery, 2015, pp. 5–6. ISBN: 9781450337182. DOI: 10.1145/2791261.2791262. URL: https://doi.org/10.1145/2791261.2791262.

[120] Sven Pohle et al. “DreamWorks Animation Facial Motion and Deformation System”. In: *Proceedings of the 2015 Symposium on Digital Production*. DigiPro ’15. Los Angeles, California: ACM, 2015, pp. 5–6. ISBN: 978-1-4503-3718-2.

[121] Ravi Ramamoorthi and Pat Hanrahan. “An Efficient Representation for Irradiance Environment Maps”. In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’01. New York, NY, USA: ACM, 2001, pp. 497–500. ISBN: 1-58113-374-X. DOI: 10.1145/383259.383317. URL: http://doi.acm.org/10.1145/383259.383317.

[122] Anurag Ranjan et al. “Generating 3D Faces Using Convolutional Mesh Autoencoders”. In: *Computer Vision – ECCV 2018*. Ed. by Vittorio Ferrari et al. Cham: Springer International Publishing, 2018, pp. 725–741. ISBN: 978-3-030-01219-9.

[123] Jérémy Riviere et al. “Single-Shot High-Quality Facial Geometry and Skin Appearance Capture”. In: *ACM Trans. Graph.* 39.4 (July 2020). ISSN: 0730-0301. URL: https://doi.org/10.1145/3386569.3392464.

[124] Charles F. Rose III, Peter-Pike J. Sloan, and Michael F. Cohen. “Artist-Directed Inverse-Kinematics Using Radial Basis Function Interpolation”. In: *Computer Graphics Forum* 20.3 (2001), pp. 239–250. DOI: 10.1111/1467-8659.00516. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/1467-8659.00516. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-8659.00516.

[125] Christos Sagonas et al. “300 Faces In-The-Wild Challenge: database and results”. In: *Image and Vision Computing* 47 (Jan. 2016). DOI: 10.1016/j.imavis.2016.01.002.

[126] S. Schaefer and C. Yuksel. “Example-based Skeleton Extraction”. In: *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*. SGP ’07. Barcelona, Spain: Eurographics Association, 2007, pp. 153–162. ISBN: 978-3-905673-46-3. URL: http://dl.acm.org/citation.cfm?id=1281991.1282013.

[127] Gabriel Schwartz et al. "The Eyes Have It: An Integrated Eye and Face Model for Photorealistic Facial Animation". In: *ACM Trans. Graph.* 39.4 (July 2020). ISSN: 0730-0301. DOI: `10.1145/3386569.3392493`. URL: `https://doi.org/10.1145/3386569.3392493`.

[128] Thomas W. Sederberg and Scott R. Parry. "Free-form Deformation of Solid Geometric Models". In: *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '86. New York, NY, USA: ACM, 1986, pp. 151–160. ISBN: 0-89791-196-2.

[129] Jaewoo Seo et al. "Compression and Direct Manipulation of Complex Blendshape Models". In: *ACM Trans. Graph.* 30 (Dec. 2011), p. 164. DOI: `10.1145/2070781.2024198`.

[130] Yeongho Seol et al. "Artist Friendly Facial Animation Retargeting". In: *ACM Trans. Graph.* 30.6 (Dec. 2011), 162:1–162:10. ISSN: 0730-0301.

[131] Aliaksandr Siarohin et al. "First Order Motion Model for Image Animation". In: *Conference on Neural Information Processing Systems (NeurIPS).* Dec. 2019.

[132] Eftychios Sifakis, Igor Neverov, and Ronald Fedkiw. "Automatic Determination of Facial Muscle Activations from Sparse Motion Capture Marker Data". In: *ACM SIGGRAPH 2005 Papers*. SIGGRAPH '05. Los Angeles, California: ACM, 2005, pp. 417–425. ISBN: 978-1-4503-7825-3. DOI: `10.1145/1186822.1073208`. URL: `http://doi.acm.org/10.1145/1186822.1073208`.

[133] Karan Singh and Eugene Fiume. "Wires: A Geometric Deformation Technique". In: *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '98. New York, NY, USA: ACM, 1998, pp. 405–414. ISBN: 0-89791-999-8.

[134] Ayan Sinha, Jing Bai, and Karthik Ramani. "Deep Learning 3D Shape Surfaces Using Geometry Images". In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe et al. Cham: Springer International Publishing, 2016, pp. 223–240.

[135] Peter-Pike J. Sloan, Charles F. Rose III, and Michael F. Cohen. "Shape by Example". In: *Proceedings of the 2001 Symposium on Interactive 3D Graphics*. I3D '01. New York, NY, USA: ACM, 2001, pp. 135–143. ISBN: 1-58113-292-1. DOI: `10.1145/364338.364382`. URL: `http://doi.acm.org/10.1145/364338.364382`.

[136] Jie Tan et al. "Learning Bicycle Stunts". In: *ACM Trans. Graph.* 33.4 (July 2014), 50:1–50:12. ISSN: 0730-0301.

[137] Ayush Tewari et al. "MoFA: Model-Based Deep Convolutional Face Autoencoder for Unsupervised Monocular Reconstruction". In: Oct. 2017, pp. 1274–1283. DOI: `10.1109/ICCVW.2017.153`.

[138] Theano Development Team. "Theano: A Python framework for fast computation of mathematical expressions". In: *arXiv e-prints* abs/1605.02688 (May 2016). URL: `http://arxiv.org/abs/1605.02688`.

[139] Jean-Marc Thiery et al. "Animated Mesh Approximation With Sphere-Meshes". In: *ACM Trans. Graph.* 35.3 (May 2016), 30:1–30:13. ISSN: 0730-0301. DOI: 10.1145/2898350. URL: http://doi.acm.org/10.1145/2898350.

[140] Justus Thies et al. "Face2Face: Real-Time Face Capture and Reenactment of RGB Videos". In: *Commun. ACM* 62.1 (Dec. 2018), pp. 96–104. ISSN: 0001-0782. DOI: 10.1145/3292039. URL: https://doi.org/10.1145/3292039.

[141] Anh Tran et al. "Regressing Robust and Discriminative 3D Morphable Models with a Very Deep Neural Network". In: July 2017, pp. 1493–1502. DOI: 10.1109/CVPR.2017.163.

[142] Luan Tran, Xi Yin, and Xiaoming Liu. "Disentangled Representation Learning GAN for Pose-Invariant Face Recognition". In: *In Proceeding of IEEE Computer Vision and Pattern Recognition.* Honolulu, HI, July 2017.

[143] Adrien Treuille, Yongjoon Lee, and Zoran Popović. "Near-optimal Character Animation with Continuous Control". In: *ACM Trans. Graph.* 26.3 (July 2007). ISSN: 0730-0301.

[144] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. "Instance Normalization: The Missing Ingredient for Fast Stylization". In: *CoRR* abs/1607.08022 (2016). arXiv: 1607.08022. URL: http://arxiv.org/abs/1607.08022.

[145] Dmitry Ulyanov et al. "Texture Networks: Feed-Forward Synthesis of Textures and Stylized Images". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48.* ICML'16. New York, NY, USA: JMLR.org, 2016, pp. 1349–1357.

[146] Raquel Urtasun et al. "Priors for People Tracking from Small Training Sets". In: *Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1 - Volume 01.* ICCV '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 403–410. ISBN: 0-7695-2334-X-01.

[147] Jack M. Wang, David J. Fleet, and Aaron Hertzmann. "Gaussian process dynamical models". In: *In NIPS.* MIT Press, 2006, pp. 1441–1448.

[148] Jack M. Wang, David J. Fleet, and Aaron Hertzmann. "Gaussian Process Dynamical Models for Human Motion". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 30.2 (Feb. 2008), pp. 283–298. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2007.1167. URL: http://dx.doi.org/10.1109/TPAMI.2007.1167.

[149] Jack M. Wang, David J. Fleet, and Aaron Hertzmann. "Gaussian Process Dynamical Models for Human Motion". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 30.2 (Feb. 2008), pp. 283–298. ISSN: 0162-8828.

[150] Robert Y. Wang, Kari Pulli, and Jovan Popović. "Real-time Enveloping with Rotational Regression". In: *ACM Trans. Graph.* 26.3 (July 2007). ISSN: 0730-0301. DOI: 10.1145/1276377.1276468. URL: http://doi.acm.org/10.1145/1276377.1276468.

[151] Xiaohuan Corina Wang and Cary Phillips. “Multi-weight Enveloping: Least-squares Approximation Techniques for Skin Animation”. In: *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’02. San Antonio, Texas: ACM, 2002, pp. 129–138. ISBN: 1-58113-573-4. DOI: `10.1145/545261.545283`. URL: `http://doi.acm.org/10.1145/545261.545283`.

[152] K. Waters and T. Levergood. “An Automatic Lip-synchronization Algorithm for Synthetic Faces”. In: *Proceedings of the Second ACM International Conference on Multimedia*. MULTIMEDIA ’94. San Francisco, California, USA: ACM, 1994, pp. 149–156. ISBN: 0-89791-686-7.

[153] Martin Watt et al. “LibEE: A Multithreaded Dependency Graph for Character Animation”. In: *Proceedings of the Digital Production Symposium*. DigiPro ’12. Glendale, California: ACM, 2012, pp. 59–66. ISBN: 978-1-4503-1649-1. DOI: `10.1145/2370919.2370930`. URL: `http://doi.acm.org/10.1145/2370919.2370930`.

[154] Martin Watt et al. “LibEE: A Multithreaded Dependency Graph for Character Animation”. In: *Proceedings of the Digital Production Symposium*. DigiPro ’12. Glendale, California: ACM, 2012, pp. 59–66. ISBN: 978-1-4503-1649-1.

[155] Shih-En Wei et al. “VR Facial Animation via Multiview Image Translation”. In: *ACM Trans. Graph.* 38.4 (July 2019). ISSN: 0730-0301. DOI: `10.1145/3306346.3323030`. URL: `https://doi.org/10.1145/3306346.3323030`.

[156] Thibaut Weise et al. “Realtime Performance-based Facial Animation”. In: *ACM Trans. Graph.* 30.4 (July 2011), 77:1–77:10. ISSN: 0730-0301. DOI: `10.1145/2010324.1964972`. URL: `http://doi.acm.org/10.1145/2010324.1964972`.

[157] Chris J. Welman. “Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation”. PhD thesis. Simon Fraser University, 1993.

[158] Olivia Wiles, A. Sophia Koepke, and Andrew Zisserman. “X2Face: A network for controlling face generation using images, audio, and pose codes”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Sept. 2018.

[159] Xian Xiao et al. “A Powell Optimization Approach for Example-Based Skinning in a Production Animation Environment”. In: *Computer Animation and Social Agents* (Dec. 2006).

[160] Yuting Ye and C. Karen Liu. “Synthesis of Responsive Motion Using a Dynamic Model”. In: *Computer Graphics Forum* 29.2 (2010), pp. 555–562. ISSN: 1467-8659.

[161] E. Zakharov et al. “Few-Shot Adversarial Learning of Realistic Neural Talking Head Models”. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 9458–9467.

[162] Li Zhang et al. “Spacetime Faces: High Resolution Capture for Modeling and Animation”. In: *ACM Trans. Graph.* 23.3 (Aug. 2004), pp. 548–558. ISSN: 0730-0301. DOI: `10.1145/1015706.1015759`. URL: `http://doi.acm.org/10.1145/1015706.1015759`.

[163] Richard Zhang et al. “The Unreasonable Effectiveness of Deep Features as a Perceptual Metric”. In: *CVPR*. 2018.

[164] Jun-Yan Zhu et al. “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks”. In: *Computer Vision (ICCV), 2017 IEEE International Conference on*. 2017.

[165] Michael Zollhöfer et al. “State of the Art on Monocular 3D Face Reconstruction, Tracking, and Applications”. In: *Comput. Graph. Forum* 37 (2018), pp. 523–550.