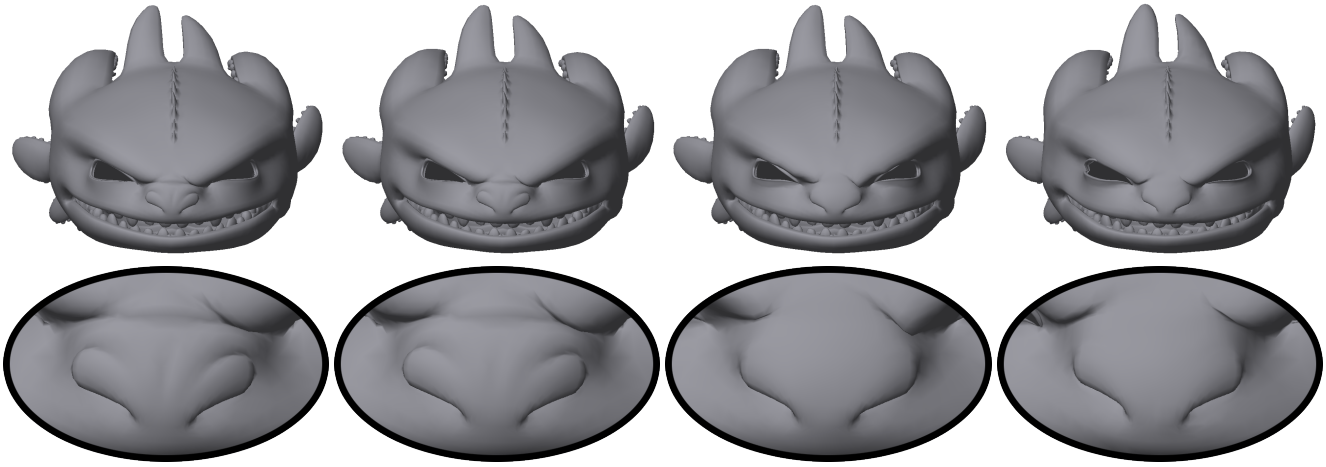
Fast and Deep Facial Deformations

STEPHEN W. BAILEY, University of California, Berkeley and Unity Technologies
DALTON OMENS, University of California, Berkeley
PAUL DILORENZO, DreamWorks Animation
JAMES F. O'BRIEN, University of California, Berkeley



(a) Ground Truth (b) Refined Approximation (c) Coarse Approximation (d) LBS Approximation

Fig. 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.

Film-quality characters typically display highly complex and expressive facial deformation. The underlying rigs used to animate the deformations of a character's face are often computationally expensive, requiring high-end hardware to deform the mesh at interactive rates. In this paper, we present a method using convolutional neural networks for approximating the mesh deformations of characters' faces. For the models we tested, our approximation runs up to 17 times faster than the original facial rig while still maintaining a high level of fidelity to the original rig. We also propose an extension to the approximation for handling high-frequency deformations such as fine skin wrinkles. While the implementation of the original animation rig depends on an extensive set of proprietary libraries making it difficult to install outside of an in-house development environment, our fast approximation relies on the widely available and easily deployed TensorFlow libraries. In addition to allowing high frame rate evaluation on modest hardware and in a wide range of computing environments, the large speed increase also enables interactive inverse kinematics on the animation rig. We demonstrate our approach and its applicability through interactive character posing and real-time facial performance capture.

CCS Concepts: • **Computing methodologies** → **Neural networks**; **Shape modeling**; **Animation**.

Additional Key Words and Phrases: character rig, facial animation, mesh deformations, deep learning, function approximation

ACM Reference Format:
Stephen W. Bailey, Dalton Omens, Paul DiLorenzo, and James F. O'Brien. 2020. Fast and Deep Facial Deformations. *ACM Trans. Graph.* 39, 4, Article 94 (July 2020), 15 pages. https://doi.org/10.1145/3386569.3392397

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 deformation include pose space deformations [Lewis et al. 2000] 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. The primary reason why film-quality facial rigs contain more sophisticated mesh deformations is because they are not constrained by real-time requirements.

Authors' addresses: Stephen W. Bailey, University of California, Berkeley, Unity Technologies; Dalton Omens, University of California, Berkeley; Paul DiLorenzo, DreamWorks Animation; James F. O'Brien, University of California, Berkeley.

0730-0301/2020/7-ART94 $15.00
https://doi.org/10.1145/3386569.3392397

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. [2018] 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 1) while running up to 17x faster than the production rigs that we tested.

Furthermore, we implement our approximation model in TensorFlow [Abadi et al. 2015], 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. Because our model is neural network-based, the approximation is fully differentiable. We demonstrate the usefulness of this property by developing an inverse kinematics-based posing application as well as a facial performance capture system.

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) [Magnenat-Thalmann et al. 1988], 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 [Wang and Phillips 2002] and dual-quaternion skinning [Kavan et al. 2007] 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 [Kleiser 1989; Parke 1972, 1974] 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 [Cong et al. 2015; Ichim et al. 2017; Sifakis et al. 2005]. 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 [Bickel et al. 2008] utilizes pose-space deformation [Lewis et al. 2000] 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. [2013] improves on rig-space physics [Hahn et al. 2012] 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 [Le and Deng 2012] finds the bone transformations and skin weights for a skeleton subspace deformation model given a set of example poses. Similarly, Le and Deng [2014] also finds a SSD representation of the deformation, but organized in a skeletal hierarchy for easier animation afterward. Feng et al. [2008] 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 [Thiery et al. 2016] 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 et al. [2010] and Neumann et al. [2013] learn new parametric rig models like blendshapes from example poses. Garrido et al. [2016] 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 [Seo et al. 2011] or an underlying skeletal structure, while our method does not make such strong assumptions about the facial rig. EigenSkin [Kry et al. 2002] 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 [2003] learns an augmented SSD skinning model with additional joints from an existing SSD rig. The work of Bailey et al. [2018] assumes an underlying skeleton and uses the skeletal deformation as a base on which a fine-detail nonlinear displacement is overlaid. In this paper, 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 [Buss and Kim 2005; Girard and Maciejewski 1985; Mukai and Kuriyama 2005; Rose III et al. 2001; Welman 1993], 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 [Bickel et al. 2008; Lewis and Anjyo 2010; Zhang et al. 2004], 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. [2006] 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 et al. [2017] 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 [Feng et al. 2018] and facial animation [Laine et al. 2017] to cloth simulation [Jin et al. 2018]. One way to apply CNNs to meshes is by defining mesh convolution operations. Litany et al. [2017] introduces graph convolutional autoencoders, while Ranjan et al. [2018] uses a similar idea to generate 3D faces. MeshCNN [Hanocka et al. 2019] 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. [2015] and Boscaini et al. [2016] apply convolutions to a mesh by parameterizing the mesh around a small local area. Sinha et al. [2016] applies CNNs by projecting a mesh onto a spherical domain then "cutting up" the projection. Other works [Feng et al. 2018; Jin et al. 2018; Lähner et al. 2018] 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 [Feng et al. 2018] or scans [Jin et al. 2018; Lähner et al. 2018]. Because our work assumes a complete character rig, we use a UV map created by an artist to compute vertex positions from a 2D feature space.

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 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 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.

3.1 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). 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,

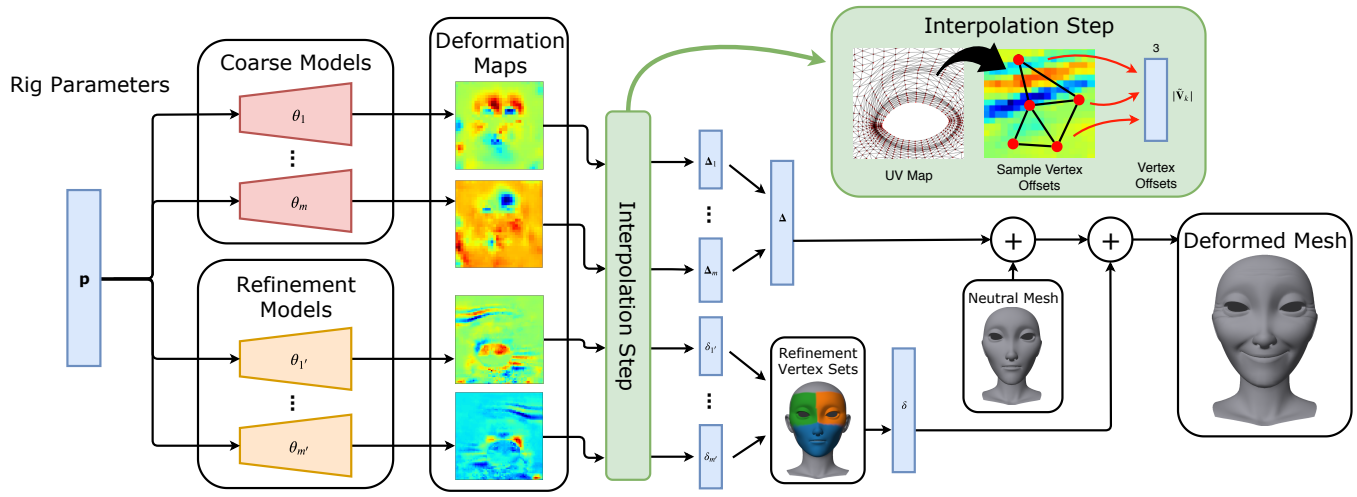


Fig. 2. Diagram of the approximation model. Rig parameters are used as input to convolutional networks which generate a deformation map for each mesh segment. Vertex offsets are extracted by bilinear interpolation of the deformation map at each vertex position in texture coordinate space. These offsets 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.

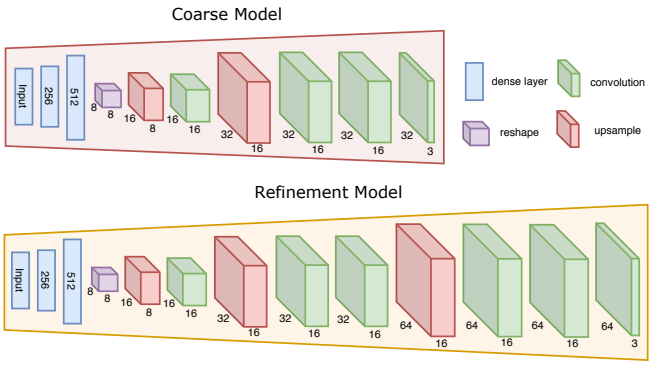


Fig. 3. Detail of course and refine models. All convolutions use 3x3 kernels except for the last layer, which uses a 1x1 kernel. All layers but the last use the leaky ReLU activation function, and no activation function is applied to the last layer. All non-dense layers are square in the image plane.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

the vertex offsets for the full mesh are obtained by concatenating the offsets for each segment such that $\Delta = \cup_{k\in\{1,\ldots,m\}}\Delta_k$. 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

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

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.

3.2 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 1 where the loss is evaluated as $\mathcal{L}(\mathbf{V}, \Delta + \boldsymbol{\delta})$.

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.

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

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

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.

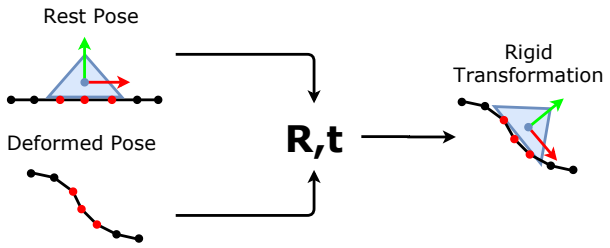


Fig. 4. Illustration of the rigid components. The blue triangle represents a rigid mesh segment identified by Equation 3. The black line represents a nonlinearly deformed mesh segment, and the dots on the line represent vertices on the surface. The red dots represent the set of vertices identified by Equation 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.

3.4 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 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 $\mathbf{V}^0_k$ to $\mathbf{V}^i_k$ as

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

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

Let $\mathbf{V}^i_r$ be a rigidly deformed mesh segment (i.e. $e_r < \tau$) for sample i. Let $\mathbf{R}^i_r$ and $\mathbf{t}^i_r$ be the minimizers of Equation 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 $\mathbf{R}^i_r$, $\mathbf{t}^i_r$ across all samples i

$$\epsilon_{r,j} = \frac{1}{n}\sum_{i=1}^{n} \left\| \mathbf{v}^0_j \mathbf{R}^i_r + \mathbf{t}^i_r - \mathbf{v}^i_j \right\|^2_2. \tag{4}$$

For the rigid mesh segment r, let $\mathbf{V}^0_\delta$ and $\mathbf{V}^i_\delta$ be the set of vertices with the c smallest approximation errors $\epsilon_{r,j}$ where $|V^0_\delta| = c$. 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 $\mathbf{V}'_r = \mathbf{V}^0_r \mathbf{R}'_\delta + \mathbf{t}'_\delta$ where $\mathbf{R}'_\delta$ and $\mathbf{t}'_\delta$ are the minimizers of Equation 3 for the vertex positions $\mathbf{V}'_\delta$.

3.5 Implementation Details

All of the models $\mathbf{f}(\mathbf{p}; \boldsymbol{\theta}_k)$ and $\mathbf{f}(\mathbf{p}; \boldsymbol{\theta}^r_{k'})$ 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 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 1. These models are trained with the Adam optimizer [Kingma and Ba 2014] 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) and the sets of vertices assigned to each rigid segment (Equation 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

$$\mathcal{T} = \{\mathbf{u} \odot \mathbf{p} \mid \mathbf{u} \sim U(0.25, 3.0)^m, \mathbf{p} \sim \mathcal{A}\} \tag{5}$$

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 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. [2017], 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.

4 INVERSE KINEMATICS

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 [Aristidou et al. 2018]. 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 through our

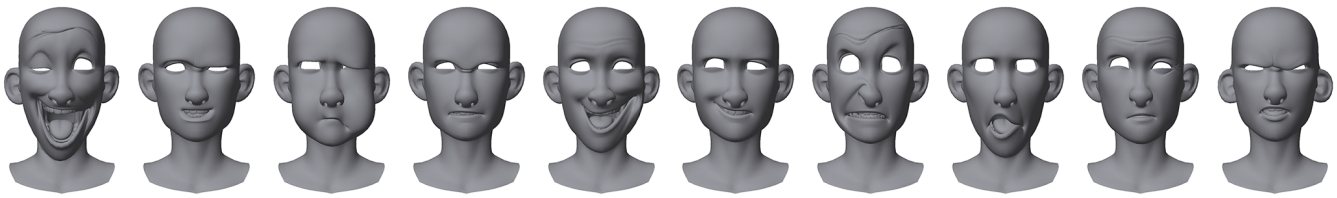


Fig. 5. Example poses from the training data.

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.

4.1 Model Details

Let C be the set of indices of vertices corresponding to IK control points, and let $\mathbf{r}_C(\mathbf{p}) : \mathbb{R}^m \rightarrow \mathbb{R}^{|C|\times 3}$ be the rig function that maps the rig parameters $\mathbf{p}$ to the subset of vertices $\mathbf{V}_C$. Then the inverse kinematics problem can be expressed as

$$\mathbf{p}' = \arg\min_{\mathbf{p}} \left\| \mathbf{r}_C(\mathbf{p}) - \overline{\mathbf{V}}_C \right\|_F^2 \tag{6}$$

where $\overline{\mathbf{V}}_C$ 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 $\mathbf{f}_{IK} : \mathbb{R}^{|C|\times 3} \rightarrow \mathbb{R}^m$ to approximate the minimization problem through a fixed-length computation such that

$$\mathbf{f}_{IK}(\overline{\mathbf{V}}_C; \boldsymbol{\theta}_{IK}) = \arg\min_{\mathbf{p}} \left\| \tilde{\mathbf{r}}_C(\mathbf{p}) - \overline{\mathbf{V}}_C \right\|_F^2 \tag{7}$$

where $\boldsymbol{\theta}_{IK}$ are the network parameters that require training. The model is trained on a specific set of control points and vertices $\mathbf{V}_C$, 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

$$\mathcal{L}(\overline{\mathbf{V}}_C) = \mathcal{L}_{point}(\overline{\mathbf{V}}_C) + \lambda_{reg}\mathcal{L}_{reg}(\overline{\mathbf{V}}_C) \tag{8}$$

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

$$\mathcal{L}_{point}(\overline{\mathbf{V}}_C) = \frac{1}{|C|} \sum_{i \in C} \left\| \tilde{\mathbf{r}}_i(\mathbf{f}_{IK}(\overline{\mathbf{V}}_C; \boldsymbol{\theta}_{IK})) - \overline{\mathbf{v}}_i \right\|_1 . \tag{9}$$

The regularization term penalizes large parameter values as

$$\mathcal{L}_{reg}(\overline{\mathbf{V}}_C) = \left\| (\mathbf{f}_{IK}(\overline{\mathbf{V}}_C; \boldsymbol{\theta}_{IK}) - \mathbf{p}^0) \odot \mathbf{s} \right\|_1 \tag{10}$$

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 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 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 $\overline{\mathbf{V}}_{C_j}$, 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.

4.2 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

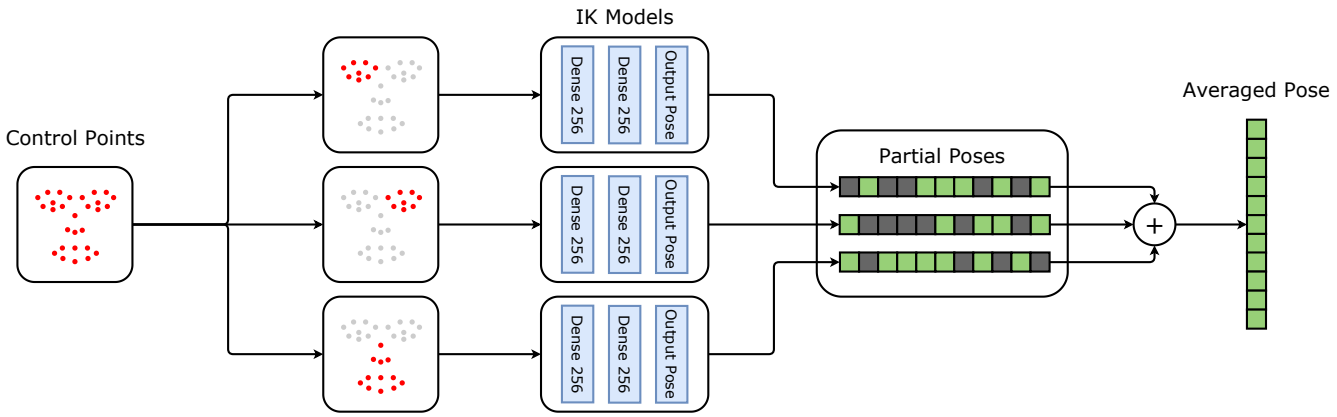


Fig. 6. Diagram of IK model. Control points are divided into disjoint subsets and provided to separate dense neural networks. Each network outputs a subset of the pose. The valid values from the outputs are averaged together to produce the final averaged rig parameter pose.

the final layer so that the network can output any value for the rig parameters. A diagram of the network is shown in Figure 6. Similar to the facial approximation model, the IK model is optimized with Adam using the same training schedule and balanced dataset described in Section 3.5.

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 $\overline{\mathbf{v}}_C$, a new sample is computed as $\overline{\mathbf{v}}'_C = \overline{\mathbf{v}}_C + \mathrm{U}(-\delta, \delta)^{|C|\times 3}$ 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 $\overline{\mathbf{v}}'_C$, but all other aspects of model training remain identical.

5 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 [Pohle et al. 2015]. 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 [Besedin et al. 2018].

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, 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 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 7 visualizes the mesh segments for the facial models of Hiccup, Valka, and Toothless, and Figure 8 shows the mesh segments used during the refinement stage of the approximation.

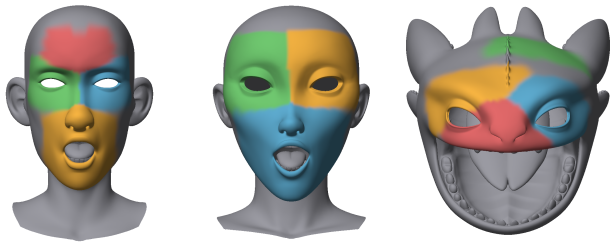
In our results, the dense model runs faster than our approximation model, and in one case is more accurate than our model when approximating artist-created animations. However, the dense approximation's faster speed does come at the cost of more model parameters, which translates to higher memory storage costs as seen in Table 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 10. 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.

Table 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

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

Fig. 7. Visualization of the mesh segments of the three characters. Each mesh segment is represented as a continuous region of the same color.



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

Fig. 8. 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.

5.1 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 [2012]. 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 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. [2018]. 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 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 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 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,

$$E_{\text{normal}} = \frac{1}{f}\sum_{i=1}^{f} \arccos\left(\mathbf{n}_i \cdot \mathbf{n}_i'\right) \tag{11}$$

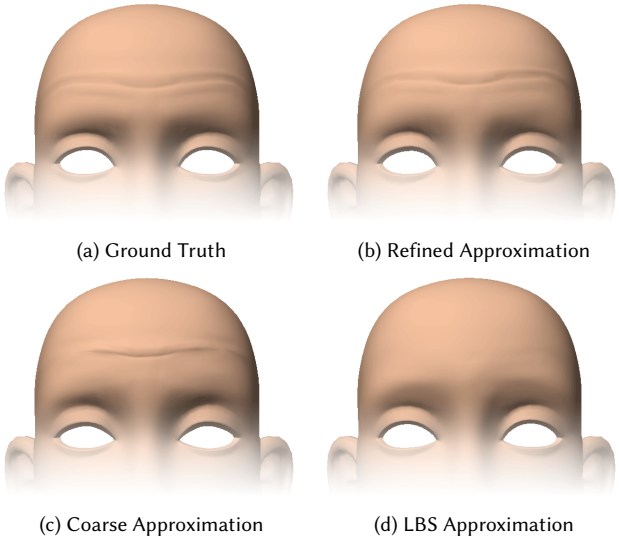
where $\mathbf{n}_i$ is the normal of face i in the ground truth mesh, and $\mathbf{n}_i'$ is the normal of face i in the approximated mesh with a total of f faces.

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 1 and Figure 9 show approximated deformations for a poses of Toothless and Hiccup containing wrinkles. As seen in Table 2, the refined approximation produces the smallest normal error for Hiccup, Valka, and Ray, but 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 10 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 [1].

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

Table 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 Approximation

(c) Coarse Approximation (d) LBS Approximation

Fig. 9. Comparison of forehead wrinkles on Hiccup's mesh using our approximation and LBS.

5.2 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 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 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.

Table 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.

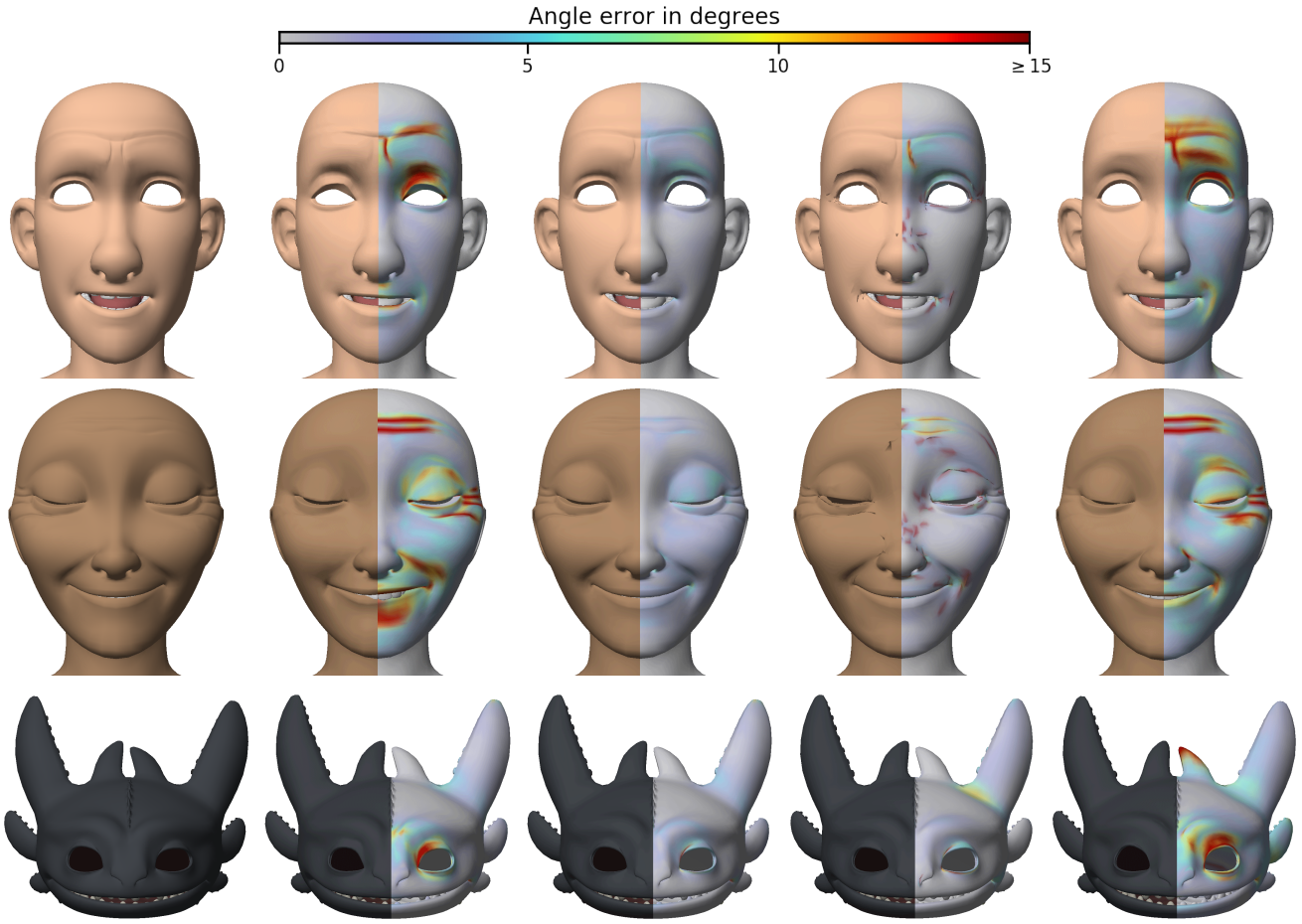
		Hiccup	Valka	Toothless
Original Rig	High-end	75	66	30
Coarse w/ GPU		6.4	6.0	10.4
Full w/ GPU		8.7	7.5	12.6
Dense w/ GPU		2.6	2.6	4.2
Coarse		2.9	2.7	4.4
Full		4.2	3.8	5.6
Dense		2.1	2.2	2.9
Coarse w/ GPU	Consumer	5.8	3.2	7.3
Full w/ GPU		4.3	4.5	9.1
Dense w/ GPU		1.8	1.7	7.3
Coarse		6.9	2.7	5.7
Full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

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 [Watt et al. 2012], 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 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.

5.3 Applications

Our 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.



(a) Ground Truth (b) Coarse (c) Refined (d) Dense (e) LBS

Fig. 10. 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.

5.3.1 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 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 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 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

Table 4. 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

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, our method more closely matches the ground truth mesh evaluated on rig parameters output by the IK model. Figure 11 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 2) and on the set of poses output by the IK model (Table 4).

5.3.2 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. [2018] 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 [Feng et al. 2017], 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 12 shows a frame from a recording and the resulting deformed mesh from the input.

6 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. As examples, we introduce a real-time IK-based posing method as well as a performance facial capture system built on top of the IK solver. 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 13 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.

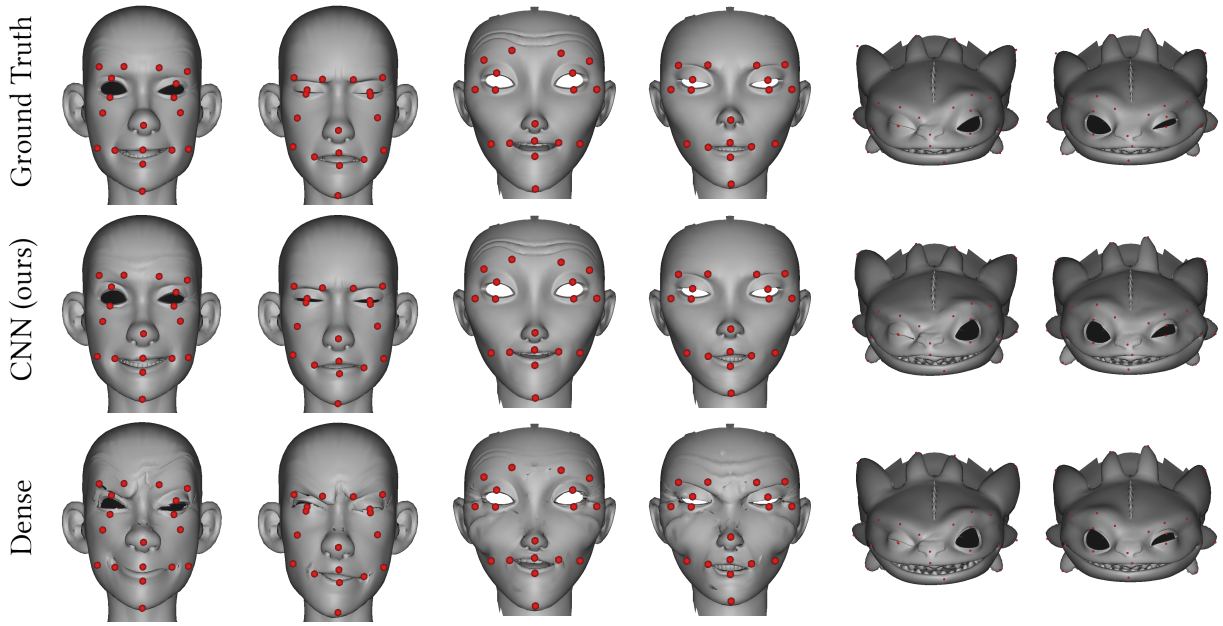
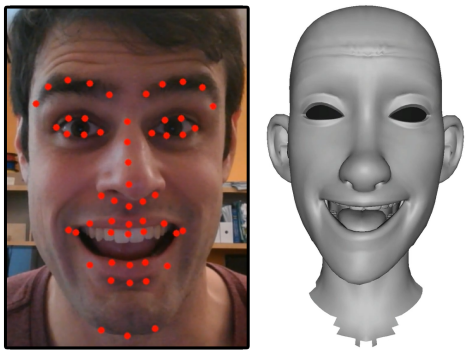


Fig. 11. Comparison of meshes deformed by rig parameters computed through the IK model. The red dots represent the control points provided to the IK model.



(a) Input frame (b) Target mesh

Fig. 12. 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.

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.

ACKNOWLEDGMENTS

We would like to thank Dave Otte for narrating the supplementary video. We would also like to thank all of the DreamWorks character artists, animators, and technical directors who helped create the character rigs and animations that were used for this paper. Finally, we would like to thank the anonymous reviewers for their helpful feedback.

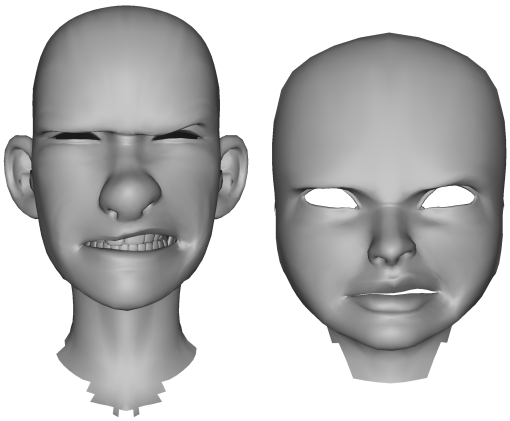


Fig. 13. Rig approximation of Hiccup transferred to a new mesh with a different topology. A single mesh segment from the coarse approximation is applied to the new mesh on the right. The facial mesh on the right is from the freely available Mathilda Rig developed by Leon Li-Aun Sooi and Xiong Lin.

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

Andreas Aristidou, Joan Lasenby, Yiorgos Chrysanthou, and Ariel Shamir. 2018. Inverse Kinematics Techniques in Computer Graphics: A Survey. *Computer Graphics Forum* 37 (09 2018), 35–58. https://doi.org/10.1111/cgf.13310

Stephen W. Bailey, Dave Otte, Paul Dilorenzo, and James F. O'Brien. 2018. Fast and Deep Deformation Approximations. *ACM Trans. Graph.* 37, 4, Article 119 (July 2018), 12 pages. https://doi.org/10.1145/3197517.3201300

Vadim Besedin, Jalil Sadool, Ludovic bouancheau, Shannon Thomas, Victor Vinyals, Mike Safianoff, Joe Bowers, and David Stodolny. 2018. Ray Character Rig. CGTarian Animation & VFX Online School. https://www.cgtarian.com/maya-character-rigs/download-free-3d-character-ray.html

Bernd Bickel, Manuel Lang, Mario Botsch, Miguel A. Otaduy, and Markus Gross. 2008. Pose-space Animation and Transfer of Facial Details. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '08)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 57–66. http://dl.acm.org/citation.cfm?id=1632592.1632602

Davide Boscaini, Jonathan Masci, Emanuele Rodoià, and Michael Bronstein. 2016. Learning Shape Correspondence with Anisotropic Convolutional Neural Networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS'16)*. Curran Associates Inc., Red Hook, NY, USA, 3197–3205.

Samuel R. Buss and Jin-Su Kim. 2005. Selectively Damped Least Squares for Inverse Kinematics. *Journal of Graphics Tools* 10, 3 (2005), 37–49. https://doi.org/10.1080/2151237X.2005.10129202 arXiv:https://doi.org/10.1080/2151237X.2005.10129202

Matthew Cong, Michael Bao, Jane L. E, Kiran S. Bhat, and Ronald Fedkiw. 2015. Fully Automatic Generation of Anatomical Face Simulation Models. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA '15)*. ACM, New York, NY, USA, 175–183. https://doi.org/10.1145/2786784.2786786

Wei-Wen Feng, Byung-Uck Kim, and Yizhou Yu. 2008. Real-time Data Driven Deformation Using Kernel Canonical Correlation Analysis. *ACM Trans. Graph.* 27, 3, Article 91 (Aug. 2008), 9 pages. https://doi.org/10.1145/1360612.1360690

Yao Feng, Fan Wu, Xiaohu Shao, Yanfeng Wang, and Xi Zhou. 2018. Joint 3D Face Reconstruction and Dense Alignment with Position Map Regression Network. *CoRR* abs/1803.07835 (2018). arXiv:1803.07835 http://arxiv.org/abs/1803.07835

Zhen-Hua Feng, Josef Kittler, Muhammad Awais, Patrik Huber, and Xiaojun Wu. 2017. Wing Loss for Robust Facial Landmark Localisation with Convolutional Neural Networks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2017), 2235–2245.

Pablo Garrido, Michael Zollhöfer, Dan Casas, Levi Valgaerts, Kiran Varanasi, Patrick Pérez, and Christian Theobalt. 2016. Reconstruction of Personalized 3D Face Rigs from Monocular Video. *ACM Trans. Graph.* 35, 3, Article Article 28 (May 2016), 15 pages. https://doi.org/10.1145/2890493

Michael Girard and A. A. Maciejewski. 1985. Computational Modeling for the Computer Animation of Legged Figures. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '85)*. ACM, New York, NY, USA, 263–270. https://doi.org/10.1145/325334.325244

Fabian Hahn, Sebastian Martin, Bernhard Thomaszewski, Robert Sumner, Stelian Coros, and Markus Gross. 2012. Rig-space Physics. *ACM Trans. Graph.* 31, 4, Article 72 (July 2012), 8 pages. https://doi.org/10.1145/2185520.2185568

Fabian Hahn, Bernhard Thomaszewski, Stelian Coros, Robert W. Sumner, and Markus Gross. 2013. Efficient Simulation of Secondary Motion in Rig-space. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '13)*. ACM, New York, NY, USA, 165–171. https://doi.org/10.1145/2485895.2485918

Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. 2019. MeshCNN: A Network with an Edge. *ACM Trans. Graph.* 38, 4, Article Article 90 (July 2019), 12 pages. https://doi.org/10.1145/3306346.3322959

Daniel Holden, Jun Saito, and Taku Komura. 2017. Learning Inverse Rig Mappings by Nonlinear Regression. *IEEE Transactions on Visualization and Computer Graphics* 23, 3 (March 2017), 1167–1178. https://doi.org/10.1109/TVCG.2016.2628036

Alexandru-Eugen Ichim, Petr Kadleček, Ladislav Kavan, and Mark Pauly. 2017. Phace: Physics-based Face Modeling and Animation. *ACM Trans. Graph.* 36, 4, Article 153 (July 2017), 14 pages. https://doi.org/10.1145/3072959.3073664

Ning Jin, Yilin Zhu, Zhenglin Geng, and Ronald Fedkiw. 2018. A Pixel-Based Framework for Data-Driven Clothing. *CoRR* abs/1812.01677 (2018). arXiv:1812.01677 http://arxiv.org/abs/1812.01677

Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. 2007. Skinning with Dual Quaternions. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games (I3D '07)*. ACM, New York, NY, USA, 39–46. https://doi.org/10.1145/1230100.1230107

Diederik Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations* (12 2014).

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

Paul G. Kry, Doug L. James, and Dinesh K. Pai. 2002. EigenSkin: Real Time Large Deformation Character Skinning in Hardware. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '02)*. ACM, New York, NY, USA, 153–159. https://doi.org/10.1145/545261.545286

Zorah Lähner, Daniel Cremers, and Tony Tung. 2018. DeepWrinkles: Accurate and Realistic Clothing Modeling. *CoRR* abs/1808.03417 (2018). arXiv:1808.03417 http://arxiv.org/abs/1808.03417

Samuli Laine, Tero Karras, Timo Aila, Antti Herva, Shunsuke Saito, Ronald Yu, Hao Li, and Jaakko Lehtinen. 2017. Production-Level Facial Performance Capture Using Deep Convolutional Neural Networks. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA '17)*. Association for Computing Machinery, New York, NY, USA, Article Article 10, 10 pages. https://doi.org/10.1145/3099564.3099581

Binh Huy Le and Zhigang Deng. 2012. Smooth Skinning Decomposition with Rigid Bones. *ACM Trans. Graph.* 31, 6, Article 199 (Nov. 2012), 10 pages. https://doi.org/10.1145/2366145.2366218

Binh Huy Le and Zhigang Deng. 2014. Robust and Accurate Skeletal Rigging from Mesh Sequences. *ACM Trans. Graph.* 33, 4, Article 84 (July 2014), 10 pages. https://doi.org/10.1145/2601097.2601161

J. P. Lewis and K. Anjyo. 2010. Direct Manipulation Blendshapes. *IEEE Computer Graphics and Applications* 30, 4 (July 2010), 42–50. https://doi.org/10.1109/MCG.2010.41

J. P. Lewis, Matt Cordner, and Nickson Fong. 2000. 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)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 165–172. https://doi.org/10.1145/344779.344862

Hao Li, Thibaut Weise, and Mark Pauly. 2010. Example-based Facial Rigging. In *ACM SIGGRAPH 2010 Papers (SIGGRAPH '10)*. ACM, New York, NY, USA, Article 32, 6 pages. https://doi.org/10.1145/1833349.1778769

Or Litany, Alexander M. Bronstein, Michael M. Bronstein, and Ameesh Makadia. 2017. Deformable Shape Completion with Graph Convolutional Autoencoders. *CoRR* abs/1712.00268 (2017). arXiv:1712.00268 http://arxiv.org/abs/1712.00268

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

Jonathan Masci, Davide Boscaini, Michael M. Bronstein, and Pierre Vandergheynst. 2015. Geodesic Convolutional Neural Networks on Riemannian Manifolds. In *Proceedings of the 2015 IEEE International Conference on Computer Vision Workshop*

(ICCVW) (ICCVW '15). IEEE Computer Society, USA, 832–840. https://doi.org/10.1109/ICCVW.2015.112

Alex Mohr and Michael Gleicher. 2003. Building Efficient, Accurate Character Skins from Examples. In *ACM SIGGRAPH 2003 Papers (SIGGRAPH '03)*. ACM, New York, NY, USA, 562–568. https://doi.org/10.1145/1201775.882308

Tomohiko Mukai and Shigeru Kuriyama. 2005. Geostatistical Motion Interpolation. In *ACM SIGGRAPH 2005 Papers (SIGGRAPH '05)*. ACM, New York, NY, USA, 1062–1070. https://doi.org/10.1145/1186822.1073313

Thomas Neumann, Kiran Varanasi, Stephan Wenger, Markus Wacker, Marcus Magnor, and Christian Theobalt. 2013. Sparse Localized Deformation Components. *ACM Trans. Graph.* 32, 6, Article 179 (Nov. 2013), 10 pages. https://doi.org/10.1145/2508363.2508417

Frederick I. Parke. 1972. Computer Generated Animation of Faces. In *Proceedings of the ACM Annual Conference - Volume 1 (ACM '72)*. ACM, New York, NY, USA, 451–457. https://doi.org/10.1145/800193.569955

Frederic Ira Parke. 1974. *A Parametric Model for Human Faces*. Ph.D. Dissertation. AAI7508697.

Sven Pohle, Michael Hutchinson, Brent Watkins, Dick Walsh, Stephen Candell, Frederick Nilsson, and Jason Reisig. 2015. DreamWorks Animation Facial Motion and Deformation System. In *Proceedings of the 2015 Symposium on Digital Production (DigiPro '15)*. Association for Computing Machinery, New York, NY, USA, 5–6. https://doi.org/10.1145/2791261.2791262

Anurag Ranjan, Timo Bolkart, Soubhik Sanyal, and Michael J. Black. 2018. Generating 3D Faces Using Convolutional Mesh Autoencoders. In *Computer Vision – ECCV 2018*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Springer International Publishing, Cham, 725–741.

Charles F. Rose III, Peter-Pike J. Sloan, and Michael F. Cohen. 2001. Artist-Directed Inverse-Kinematics Using Radial Basis Function Interpolation. *Computer Graphics Forum* 20, 3 (2001), 239–250. https://doi.org/10.1111/1467-8659.00516 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/1467-8659.00516

Jaewoo Seo, Geoffrey Irving, J.P. Lewis, and Junyong Noh. 2011. Compression and Direct Manipulation of Complex Blendshape Models. *ACM Trans. Graph.* 30 (12 2011), 164. https://doi.org/10.1145/2070781.2024198

Eftychios Sifakis, Igor Neverov, and Ronald Fedkiw. 2005. Automatic Determination of Facial Muscle Activations from Sparse Motion Capture Marker Data. In *ACM SIGGRAPH 2005 Papers (SIGGRAPH '05)*. ACM, New York, NY, USA, 417–425. https://doi.org/10.1145/1186822.1073208

Ayan Sinha, Jing Bai, and Karthik Ramani. 2016. Deep Learning 3D Shape Surfaces Using Geometry Images. In *Computer Vision – ECCV 2016*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Springer International Publishing, Cham, 223–240.

Jean-Marc Thiery, Émilie Guy, Tamy Boubekeur, and Elmar Eisemann. 2016. Animated Mesh Approximation With Sphere-Meshes. *ACM Trans. Graph.* 35, 3, Article 30 (May 2016), 13 pages. https://doi.org/10.1145/2898350

Xiaohuan Corina Wang and Cary Phillips. 2002. Multi-weight Enveloping: Least-squares Approximation Techniques for Skin Animation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '02)*. ACM, New York, NY, USA, 129–138. https://doi.org/10.1145/545261.545283

Martin Watt, Lawrence D. Cutler, Alex Powell, Brendan Duncan, Michael Hutchinson, and Kevin Ochs. 2012. LibEE: A Multithreaded Dependency Graph for Character Animation. In *Proceedings of the Digital Production Symposium (DigiPro '12)*. ACM, New York, NY, USA, 59–66. https://doi.org/10.1145/2370919.2370930

Chris J. Welman. 1993. *Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation*. Ph.D. Dissertation. Simon Fraser University.

Xian Xiao, John P. Lewis, Seah Hock Soon, Nickson Fong, and Tian Feng. 2006. A Powell Optimization Approach for Example-Based Skinning in a Production Animation Environment. *Computer Animation and Social Agents* (12 2006).

Li Zhang, Noah Snavely, Brian Curless, Brian Curless, and Steven M. Seitz. 2004. Space-time Faces: High Resolution Capture for Modeling and Animation. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 548–558. https://doi.org/10.1145/1015706.1015759

Michael Zollhöfer, Justus Thies, Pablo Garrido, Derek Bradley, Thabo Beeler, Patrick Pérez, Marc Stamminger, Matthias Nießner, and Christian Theobalt. 2018. State of the Art on Monocular 3D Face Reconstruction, Tracking, and Applications. *Comput. Graph. Forum* 37 (2018), 523–550.