Learning active quasistatic physics-based models from data

SANGEETHA GRAMA SRINIVASAN and QISI WANG, University of Wisconsin-Madison
JUNIOR ROJAS, University of Utah
GERGELY KLÁR, Weta Digital
LADISLAV KAVAN, University of Utah
EFTYCHIOS SIFAKIS, University of Wisconsin-Madison and Weta Digital

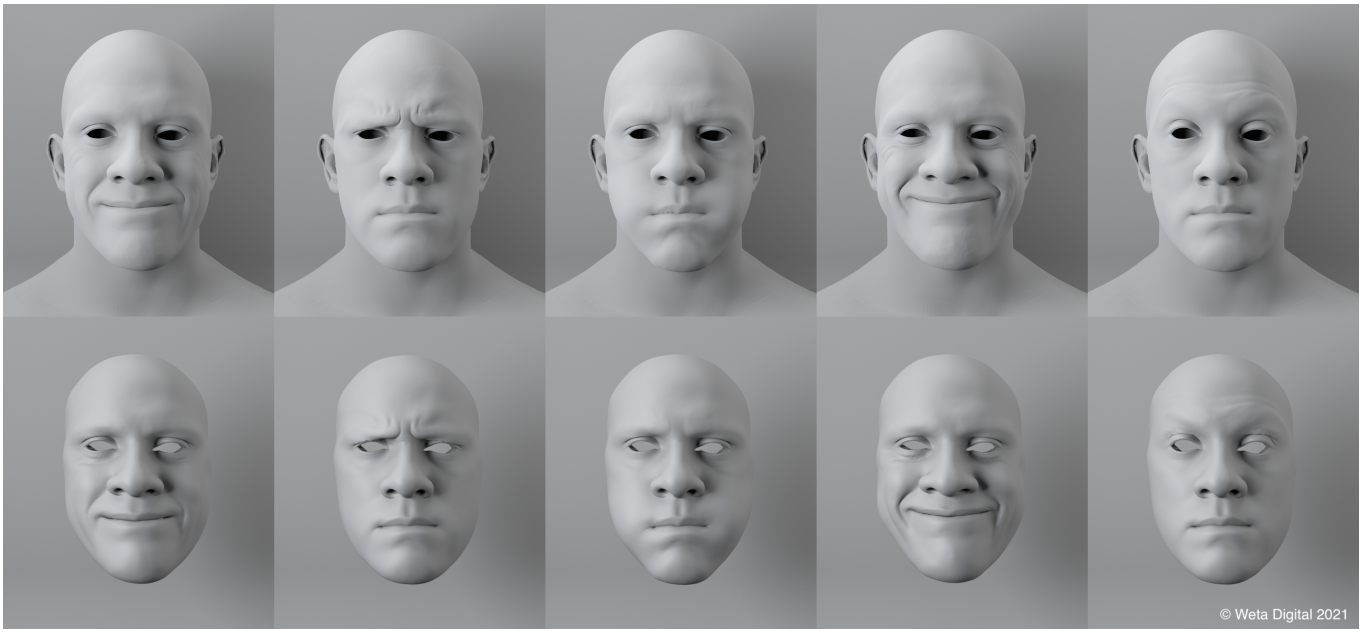

Fig. 1. This paper explores the following proposition: given a data set of target poses of an active deformable model (top), can we learn a low-dimensional control space that could reproduce them (bottom) in a physics-based way? The target poses are provided as surface meshes (top), and the reconstructions are produced by a physics-based quasistatic simulator using tetrahedral meshes (bottom). In contrast to modeling approaches where users build anatomical models from first principles, medical literature or medical imaging, we do not presume knowledge of the underlying musculature, but learn the structure and control of the actuation mechanism directly from the input data.

Humans and animals can control their bodies to generate a wide range of motions via low-dimensional action signals representing high-level goals. As such, human bodies and faces are prime examples of *active* objects, which can affect their shape via an internal actuation mechanism. This paper explores the following proposition: given a training set of example poses of an active deformable object, can we learn a low-dimensional control space that could reproduce the training set and generalize to new poses? In contrast to popular machine learning methods for dimensionality reduction such as auto-encoders, we model our active objects in a physics-based way. We utilize a differentiable, quasistatic, physics-based simulation layer and combine it with a decoder-type neural network. Our differentiable physics layer naturally fits into deep learning frameworks and allows the decoder network to learn actuations that reach the desired poses after physics-based simulation. In contrast to modeling approaches where users build anatomical models from first principles, medical literature or medical imaging, we do not presume knowledge of the underlying musculature, but learn the structure and control of the actuation mechanism directly from the input data. We present a training paradigm and several scalability-oriented enhancements that allow us to train effectively while accommodating high-resolution volumetric models, with as many as a quarter million simulation elements. The prime demonstration of the efficacy of our example-driven modeling framework targets facial animation, where we train on a collection of input expressions while generalizing to unseen poses, drive detailed facial animation from sparse motion capture input, and facilitate expression sculpting via direct manipulation.

Authors' addresses: Sangeetha Grama Srinivasan; University of Wisconsin-Madison, sgsrinivasa2@wisc.edu; Qisi Wang; University of Wisconsin-Madison, qisi.wang@gmail.com; Junior Rojas, University of Utah, jrojasdavalos@gmail.com; Gergely Klár, Weta Digital, gklar@wetafx.co.nz; Ladislav Kavan, University of Utah, ladislav.kavan@gmail.com; Eftychios Sifakis, University of Wisconsin-Madison and Weta Digital, sifakis@cs.wisc.edu.

0730-0301/2021/8-ART1 $15.00
https://doi.org/10.1145/3450626.3459883

CCS Concepts: • **Computing methodologies → Physical simulation**.

Additional Key Words and Phrases: physics-based animation, differentiable physics, neural networks

ACM Reference Format:
Sangeetha Grama Srinivasan, Qisi Wang, Junior Rojas, Gergely Klár, Ladislav Kavan, and Eftychios Sifakis. 2021. Learning active quasistatic physics-based models from data. *ACM Trans. Graph.* 40, 4, Article 1 (August 2021), 14 pages. https://doi.org/10.1145/3450626.3459883

1 INTRODUCTION

Many of the elastic deformable bodies used in graphics and visual effects applications are *active* objects, in that their deformation is the result of internal actuation mechanisms in addition to external constraints such as forces, boundary conditions, or collisions. Human bodies and faces are prime examples of such active models, as their apparent deformation is primarily driven by the action of the underlying musculature, which is effectively the built-in actuation mechanism. In graphics, it is common to construct such simulated objects from first principles, by incorporating muscle geometry and composition directly into Finite Element meshes and material models. This paper explores an alternative approach for learning how to create and simulate such active models from data, seeking to answer the question: *Given a collection of shapes that an active object takes in various scenarios and poses, and assuming they provide a good sampling of its possible behaviors, can we infer an actuation mechanism that explains and generalizes these demonstrations?*

Our approach is built around a crucial, fundamental hypothesis. We assume that the control parameters of the internal actuation mechanism we seek to learn are low-dimensional in nature. There is a clear motivation for this, inspired from anatomical models: the apparent function and shape of human bodies and faces is traced to the action of a finite number of active muscles. In our method, we start by endowing our active model with extremely fine-grained actuation mechanisms, akin to creating a completely independent "muscle" for every tetrahedral element of a Finite Element simulation mesh, similar in spirit to the work of Ichim et al. [2017] and Klár et al. [2020]. We use a neural network to generate these fine-grained actuation controls as a function of a low-dimensional set of latent variables, that parameterize the control space of the object's actuation mechanism (Figure 2). The output of this network is fed to a quasistatic, differentiable, Finite Element simulator that produces the equilibrium shape produced by such controls, while resolving additional physics constraints (e.g. boundary conditions, external forces, collisions). After training, the neural network that maps the latent parameters of the control space to the fine-grained actions effectively becomes the model of the internal actuation mechanism.

The proposed methodology explicitly seeks to model and parameterize the *control space* of the actuation mechanism rather than the space of shapes that the active model deforms into. In this aspect, it deviates from neural network architectures (e.g. autoencoders) that directly produce shape as output without incorporating a differentiable simulator in their design. There are two factors that motivate this design. First, by injecting a simulator layer into this network we reduce the burden for the neural network to learn fundamental simulation mechanics, such as rotation invariance, energy conservation, etc. Second, this architecture allows us to separate the effect that measurable/knowable simulation parameters have on the resulting simulated shapes, and focus the network on just capturing the intricacies of the actuation mechanism itself. Consider for example a training set consisting of the deformation of the skin surface in a human body, corresponding to varied skeletal poses and muscle action. If the skeletal motion was presumed to be known at all times, both for the training data and any poses that we wish to later generate, our architecture allows us to focus the expressive ability of the neural network on just capturing the mechanics of the active musculature. This would be in contrast to requiring additional parameters to capture the dimensionality of the apparent shape space of skin deformations which would be partly due to the known skeletal motion. We believe that this separation of known simulation factors or physics traits from the true latent parameters of the underlying actuation facilitates learning from a reasonably sized data set, and creates favorable conditions for generalization.

In this paper, we focus our investigation on physically-based models that produce deformation via *quasistatic* simulation, a well-represented approach in anatomy-driven animation for human bodies and faces. We presume that a volumetric mesh representation of the active object is available, e.g. a tetrahededral mesh of the flesh volume for face or body models, along with any kinematic constraints (e.g. bone attachments) that the user seeks to provide as input. We require no prior knowledge of the geometry, composition or control structure of the internal actuation mechanism – we do not presume any internal musculature, or constrain our learning process to an anatomical prior. Although our simulated models are volumetric, we operate on training data given as surface shapes; for the case of faces, we simulate volumetric flesh meshes but train them using only shape data of the exterior skin surface. In contrast to modeling approaches based on first principles, our scheme is able to capture the behavior of such active objects driven by data alone, without strict reliance on detailed knowledge of anatomy, material constitutive laws, or intricacies of muscle control.

Core contributions of our work include:

- The design concept of a neural network that generates fine-grained actuation signals as a function of a *low-dimensional* latent descriptor of the action/actuation space, combined with a quasistatic differentiable simulator.
- A training methodology for this network, bootstrapped by an autoencoder that maps surface shapes to an initial approximation of corresponding volumetric actions; this provides a warm start to training the network via backpropagation.
- A number of algorithmic enhancements aimed at scalability, that allow us to train volumetric face models with a quarter million elements, and accelerate batch processing during backpropagation.
- Demonstrations of the usability and applications of our system in the context of facial animation, including approximations of unseen expressions, motion-capture driven animation, and expression sculpting using direct manipulation.

Finally, it should be highlighted that, while producing an parametric animated model that accurately matches training demonstrations and generalizes well to unseen scenarios is a key objective, there are

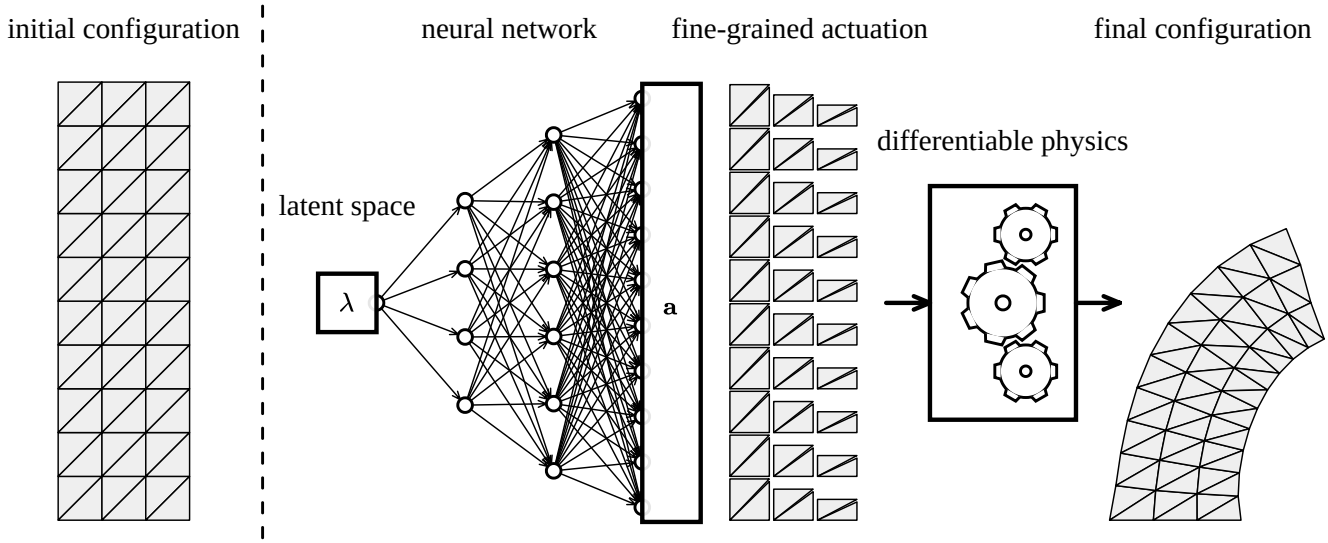

Fig. 2. Schematic view of our pipeline. We model elastic objects discretized as tetrahedralized elements and endow each element with a fine-grained actuator that can push it towards a prescribed target shape (*"fine-grained actuation"*); a differentiable simulator (*"differentiable physics"*) reconciles the elemental actuation inputs, and produces a quasistatic output shape (*"final configuration"*). We use a decoder-type neural network to generate the fine-grained controls from a low-dimensional latent space (λ) that effectively encodes the structure of the final (coarse-grained) actuators. The network is trained on a collection of final configurations representative of the desired behavior.

further reasons for pursuing a volumetric, physics-based description. After all, kinematic non-physics approaches, from traditional blendshape solvers to newer techniques that can provide good localization of actions [Neumann et al. 2013] and reproduce detailed, high-frequency features [Bailey et al. 2020] have an excellent record in this space; such methods offer opportunities for highly efficient animation and manipulation, and may also circumvent the need for a full volumetric mesh as part of the pipeline. A key added benefit of our proposed approach is the opportunity to further edit and affect the animations using physics-based effects, such as detailed resolution of collision, editing boundary conditions or mandible kinematics while retaining an expression, adding external forces, or modulating the activation signals themselves, while maintaining a physically consistent simulated face shape.

2 RELATED WORK

Physics-based modeling and simulation of anatomical systems have been studied for several decades [Barbarino et al. 2009; Flynn et al. 2015; Gladilin 2003; Stavness et al. 2014; Weiss et al. 1996; Zhang et al. 2019] and usually rely on expert operators who manually design individual organ shapes (bones, muscles, ligaments, etc.), including their connections and material parameters. Several modeling and simulation platforms have been developed, such as FEBio [Maas et al. 2017], SOFA [Faure et al. 2012], or ArtiSynth [Lloyd et al. 2012]. An important feature of physics-based simulation is the ability to faithfully model effects such as contact and volume conservation, which makes a difference in applications such as hand tracking [Smith et al. 2020]. In this paper we propose to depart from these traditional modeling paradigms and, instead, we aim to infer muscle models from data using machine learning techniques, specifically, a neural network coupled with a physics-based simulator.

Classical muscle models include the popular Hill-type models [Zajac 1989], which have been successfully used in both biomechanics [Blemker 2004] and computer graphics [Sifakis et al. 2005; Teran et al. 2003, 2005]. However, the behavior of muscles *in vivo* is poorly understood mainly due to difficulties of obtaining experimental data [Fan et al. 2014]. In this project we adopt a shape-targeting approach to muscle modeling [Fan et al. 2014; Ichim et al. 2017; Kadlecek and Kavan 2019; Klár et al. 2020; Nardinocchi and Teresi 2007]. In this family of models, muscle activations are modeled as volumetric, typically fiber-aligned contractions of the rest pose. Related models were also examined in physics-based animation to generate example-based materials [Martin et al. 2011] and to control active deformable objects to achieve specific animations [Coros et al. 2012; Tan et al. 2012]. The key difference of our approach is the ability to learn a low-dimensional action space. Although there is prior work applying dimensionality reduction methods to extract muscle synergies from EMG data [Spüler et al. 2016] and to obtain control policies for physics-based animation of articulated characters [Ding et al. 2015], in this paper we focus on dimensionality reduction via differentiable simulation for active deformable models, with an emphasis on facial animation.

Even though the expressions of the human face are driven by muscles, early methods for facial modeling and animation relied on direct, linear models [Blanz and Vetter 1999]. Blendshape-based methods dating back to FACS [Ekman and Friesen 1977] continue to be commonly used in the animation industries [Lewis et al. 2014].

Local models often provide higher expressivity and ability to generalize to even less common facial shapes [Tena et al. 2011; Wu et al. 2016]. Deep learning methods enabled a new generation of models that explain both geometry and appearance, in particular "codec avatars" based on variational auto-encoders [Bagautdinov et al. 2018; Lombardi et al. 2018], including advanced encoders using VR headsets [Wei et al. 2019] and explicit modeling of the eyes [Schwartz et al. 2020]. Most recent morphable face models feature impressive visual quality [Li et al. 2020a] even when learned from just a single scan [Li et al. 2020b].

Animating the face using physics-based simulation with explicitly modeled muscles is a difficult problem, even though early work on this topic demonstrated encouraging results [Sifakis et al. 2005]. About a decade later, the limitations of explicit muscle modeling in targeting 3D facial scans were discussed, along with more expressive muscle models such as "muscle tracks" [Cong et al. 2016] and fine-grained per-tetrahedron activations [Ichim et al. 2016]. These advances spurred renewed interest in physics-based animation of the face [Kozlov et al. 2017; Lan et al. 2017] with more recent approaches proposing inverse modeling combining MRI and 3D scans [Kadlecek and Kavan 2019]. Related to our approach is a differentiable physics-based model capable of targeting a single image [Bao et al. 2019], assuming that facial musculature is given. In this paper, we also propose a differentiable simulator, but we focus on learning the muscle model without making any anatomical assumptions.

We represent our muscle activation model by a neural network, with parameters trained from data. This is related to well-known generative models such as Generative Adversarial Networks [Arjovsky et al. 2017; Goodfellow et al. 2014] and Variational Autoencoders [Doersch 2016; Kingma and Welling 2014], which are generally applicable methods to discover latent structure in data, such as collections of images. The key difference of our methodology is that we explicitly endow our models with knowledge about physics and its numerical solution in the case of quasistatics. Even though physics invariants can be learned from data for simple mechanical systems [Schmidt and Lipson 2009], the problem becomes more challenging for complex anatomical systems. Our main thesis is that combining neural networks with physics-based priors will improve their training and ability to generalize. Similar questions have recently been studied also for other types of physics-based simulation such as cloth [Geng et al. 2020].

Differentiable physics-based simulators are an active research area because they enable the combination of the strengths of deep learning frameworks with physics. Differentiable soft-body simulators based on the material point method [Hu et al. 2020, 2019] have proven useful for training soft robots, but the explicit time integration scheme with small time steps may require a lot of memory and compute time to backpropagate through. Implicit integration enables stable simulation with large time steps, but both the forward and backward passes become much more complex to solve numerically [Geilinger et al. 2020; Hahn et al. 2019]. A recently published paper developed concurrently with our work extends Projective Dynamics (originally proposed only for forward simulation [Bouaziz et al. 2014]) to DiffPD, i.e., differentiable simulator which enjoys the fast numerics of Projective Dynamics also in the backward pass.

3 BACKGROUND

3.1 Simulation framework

Following standard practices for mesh discretization and simulation using tetrahedral elements [Sifakis and Barbič 2012], different elastic materials can be defined in terms of an energy density function $\Psi : \mathbb{R}^{3\times3} \rightarrow \mathbb{R}$ (assuming 3D simulations) that represents the stored elastic energy associated with the deformation gradient $\mathbf{F}_j(\mathbf{x}) \in \mathbb{R}^{3\times3}$ which characterizes the deformation of element j due to an arbitrary configuration of vertex positions $\mathbf{x} \in \mathbb{R}^{3n}$, where n is the number of vertices. More precisely, the deformation gradient $\mathbf{F_j}(\mathbf{X}, \mathbf{x})$ is computed assuming a rest pose $\mathbf{X} \in \mathbb{R}^{3n}$, but here we omit $\mathbf{X}$ since it can be considered as a constant associated to a particular model of interest. The total elastic energy of the body is computed by a summation over all mesh elements $E(\mathbf{x}) = \sum_j E_j(\mathbf{x})$, where $E_j = W_j\Psi(\mathbf{F_j}(\mathbf{x}))$ is the energy associated to element j and W_j is its rest-pose volume. Forces are obtained by computing the negative gradient of the energy with respect to vertex positions, i.e. $\mathbf{f}(\mathbf{x}) = -\nabla_x E(\mathbf{x})$. A quasistatic solution χ is a minimum of the energy E subject to boundary conditions C:

$$\chi = \underset{\mathbf{x}\in C}{\operatorname{argmin}}\ E(\mathbf{x}) \tag{1}$$

We employ *embedded* volumetric simulation in our framework [McAdams et al. 2011; Sifakis et al. 2007], motivated by the desire to have simulation elements that are as well conditioned as possible in their rest shape. We use simulation meshes that are derived from regular lattices (BCC or Cartesian, in our examples) and use virtual node methods to duplicate degrees of freedom to resolve topology when needed, e.g. around the lips [Mitchell et al. 2015].

3.2 Actuation model

We adopt shape targeting [Klár et al. 2020] for our actuation model, which can be defined by extending the conventional definition of energy density function Ψ to depend not only on the deformation gradient $\mathbf{F}$ but also on a shape target matrix $\mathbf{S_t} \in \mathbb{R}^{3\times3}$ for each tetrahedron:

$$\Psi : \mathbb{R}^{3\times3} \times \mathbb{R}^{3\times3} \rightarrow \mathbb{R} \tag{2}$$

$$\Psi(\mathbf{F}, \mathbf{S}_t) = \underset{\mathbf{R}\in SO(3)}{\operatorname{argmin}}\ \mu\|\mathbf{F} - \mathbf{R}\mathbf{S}_t\|_F^2 \tag{3}$$

It is our intent for $\mathbf{S}_t$ to be a rotationally-invariant descriptor of the shape that an element targets, as opposed to any specific orientation, and the minimization formulation reflects exactly that. With rotations factored away, we restrict $\mathbf{S_t}$ to be a symmetric 3×3 matrix, effectively reducing its degrees of freedom to 6. We represent $\mathbf{S_t}$ as a vector $\mathbf{b} \in \mathbb{R}^6$ using the following convention:

$$\mathbf{S_t} = \begin{pmatrix} 1+\mathbf{b_1} & \mathbf{b_2} & \mathbf{b_3} \\ \mathbf{b_2} & 1+\mathbf{b_4} & \mathbf{b_5} \\ \mathbf{b_3} & \mathbf{b_5} & 1+\mathbf{b_6} \end{pmatrix} \tag{4}$$

If we then concatenate the shape targets of all tetrahedra into a single action vector $\mathbf{a} \in \mathbb{R}^{6u}$, where u is the number of tetrahedra, we can reformulate the quasistatic solution χ from Equation 1 as a function of $\mathbf{a}$:

$$\chi(\mathbf{a}) = \underset{\mathbf{x} \in C}{\operatorname{argmin}}\, E(\mathbf{x}, \mathbf{a}) \tag{5}$$

which also implies that the force at the quasistatic equilibrium configuration is zero:

$$\mathbf{f}(\chi(\mathbf{a}), \mathbf{a}) = 0 \tag{6}$$

3.3 Learning-based pipeline

Our project focuses on finding tetrahedral actions $\mathbf{a} \in \mathbb{R}^{6u}$ that can reproduce desired poses $\mathbf{t}$ from a data set, specified as vertex positions. Note that because the number of tetrahedra u is typically large, directly controlling $\mathbf{a}$ could be problematic. Instead, we introduce $\mathbf{a} = \pi_\theta(\lambda)$, a mapping from a low-dimensional latent vector $\lambda \in \mathbb{R}^z$ to the high-dimensional action signal $\mathbf{a}$, with learnable parameters θ. We model π_θ using a neural network as shown in Figure 3.

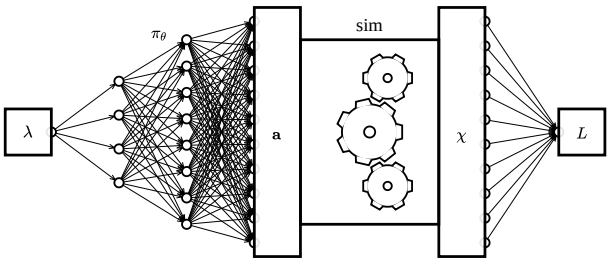

Fig. 3. Learning-based pipeline. A decoder-type network maps a low-dimensional latent vector λ to a full set of tetrahedral actions $\mathbf{a}$ that are fed to a quasistatic simulator. The simulator produces a resulting pose χ which is used to compute a loss L that measures the discrepancy between the pose produced by the simulator and a given target pose.

Our problem formulation involves finding optimal values for λ and θ to minimize a loss function L that penalizes the discrepancy between poses in the training set $\mathbf{t}_i$ and poses generated by the quasistatic solver $\chi(\mathbf{a}_i)$:

$$\lambda^*, \theta^* = \underset{\lambda, \theta}{\operatorname{argmin}} \sum_i L(\mathbf{a}_i, \mathbf{t}_i) \tag{7}$$

$$L(\mathbf{a}_i, \mathbf{t}_i) = \|\chi(\mathbf{a}_i) - \mathbf{t}_i\|^2 \tag{8}$$

$$\mathbf{a}_i = \pi_\theta(\lambda_i) \tag{9}$$

Note that θ^* is shared among all target poses, since it is a parameter of the mapping π, but λ^* includes one latent vector λ_i^* for each target pose $\mathbf{t_i}$. In other words, if the index i corresponds to time, λ_i expresses temporal variation while the network π_θ expresses spatial variation (by mapping $\lambda_i \in \mathbb{R}^z$ to $\mathbf{a}_i \in \mathbb{R}^{6u}$, where z is much smaller than $6u$).

3.4 Back-propagation through quasistatic solver

To solve the optimization problem formulated in Equation 7 using gradient-based optimization, we need to compute $\frac{dL}{d\mathbf{a}}$. We start by observing that $\frac{d\mathbf{f}}{d\mathbf{a}}$ must be zero when evaluated at $(\chi(\mathbf{a}), \mathbf{a})$ since the force $\mathbf{f}$ is zero at the quasistatic equilibrium configuration as originally stated in Equation 6:

$$\frac{d}{d\mathbf{a}}\mathbf{f}(\chi(\mathbf{a}), \mathbf{a}) = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{(\chi(\mathbf{a}),\mathbf{a})} \frac{d\chi}{d\mathbf{a}} + \left.\frac{\partial \mathbf{f}}{\partial \mathbf{a}}\right|_{(\chi(\mathbf{a}),\mathbf{a})} = 0 \tag{10}$$

Using Equation 10 we can then obtain a formula for $\frac{dL}{d\mathbf{a}}$ via the following derivations (omitting the explicit evaluations at $(\chi(\mathbf{a}), \mathbf{a})$ for conciseness):

$$\frac{d\chi}{d\mathbf{a}} = -\frac{\partial \mathbf{f}}{\partial \mathbf{x}}^{-1} \frac{\partial \mathbf{f}}{\partial \mathbf{a}} \tag{11}$$

$$\frac{dL}{d\mathbf{a}} = \frac{dL}{d\mathbf{x}}\frac{d\chi}{d\mathbf{a}} = -\frac{dL}{d\mathbf{x}}\frac{\partial \mathbf{f}}{\partial \mathbf{x}}^{-1} \frac{\partial \mathbf{f}}{\partial \mathbf{a}} \tag{12}$$

As a final step, we take advantage of the fact that the stiffness matrix $-\partial\mathbf{f}/\partial\mathbf{x}$ is *symmetric* to rewrite Equation 12 as:

$$\frac{dL}{d\mathbf{a}} = -\underbrace{\left(\underbrace{\frac{\partial \mathbf{f}}{\partial \mathbf{x}}^{-1}}_{\mathbb{R}^{3n\times 3n}} \underbrace{\frac{dL}{d\mathbf{x}}^{\mathsf{T}}}_{\mathbb{R}^{3n\times 1}}\right)^{\mathsf{T}}}_{\mathbb{R}^{1\times 3n}} \underbrace{\frac{\partial \mathbf{f}}{\partial \mathbf{a}}}_{\mathbb{R}^{3n\times 6u}} \tag{13}$$

Considering the dimensionality and sparsity of $\frac{\partial \mathbf{f}}{\partial \mathbf{a}}$, this last derivation is particularly useful to make the computation more efficient. Note that $\frac{\partial \mathbf{f}}{\partial \mathbf{a}}$ is sparse because the action associated with each tetrahedral element only affects the force on its four vertices. Although previous works have used similar strategies based on implicit differentiation to compute this derivative [Bern et al. 2017b, 2019, 2017a; McNamara et al. 2004; Sifakis et al. 2005; Wojtan et al. 2006], the learning-based approach that we propose to discover *low-dimensional* latent spaces for active models with unknown structure via neural networks and differentiable simulation has not been explored before, to our best knowledge.

To put this differentiation strategy in the context of deep learning and back-propagation, we observe that the "forward pass" $\chi(\mathbf{a})$ can be computed by running the physical simulator, and the "backward pass" (which computes $\frac{dL}{d\mathbf{a}}$ assuming that $\frac{dL}{d\mathbf{x}}$ has already been computed) is defined by Equation 13 and requires solving a linear system. Note that computing the backward pass is possible because we have access to an explicit definition of the force function $\mathbf{f}$ from our physics-based model. With the forward and backward passes in place, we can compose our physics-based quasistatic simulator with neural networks and run back-propagation through the simulation step.

4 SCALABILITY OPTIMIZATIONS

Some of the challenges associated with the effective training of our pipeline are related to the scale of the training set we aspire to accommodate. Specifically, we target high-resolution volumetric simulation meshes (up to 250K tetrahedral elements in our principal demonstrations), and training sets containing hundreds of frames of sample surface deformations. This section describes the design interventions and algorithmic enhancements that allowed us to

accommodate such training tasks on typical well-equipped GPU workstations.

4.1 Batch-optimized computation

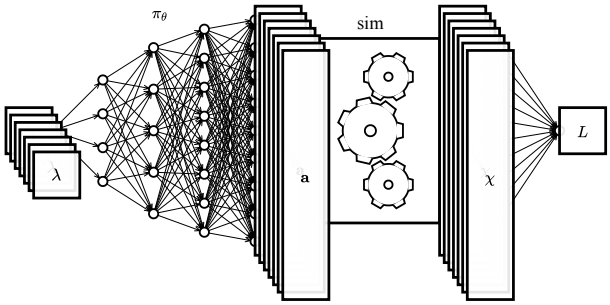

Fig. 4. Several of the state variables in our training pipeline are provided as batches. State variables in the batch include the latent parameters λ, the fine-grained actions $\mathbf{a}$, and the corresponding result of the quasistatic volumetric simulation χ. Our simulator is designed to support a batch of frames and can accommodate a persistent collection of states for each frame in the batch, thus speeding up training and inference.

We train the neural network on a collection of animation frames corresponding to example surface deformations, with the objective of optimizing the loss function collectively on all frames. As a consequence, several of the state variables in our training pipeline are provided as *batches*, with each instance in the batch corresponding to a specific targeted animation frame. Those include the latent parameters (λ_i), the fine-grained activations ($\mathbf{a}_i$), and the corresponding results of the quasistatic volumetric simulations (χ_i). During training, we conceptually need to maintain several concurrent simulator instances, both for the purposes of forward evaluation (e.g. update of the quasistatic simulation after any changes are made to the latent parameters, during training) as well as back-propagation. Note that, beyond such state variables that are directly referenced in our neural net pipeline, there are intermediate, simulation-related quantities (e.g. Polar Decomposition factors, force accumulators, bone attachment targets and boundary conditions) that are also maintained as a batch across animation frames.

In practice, instead of maintaining several concurrent and independent instances of a simulator, we have adapted a single quasistatic Projective Dynamics simulator engine to accommodate a persistent collection of states corresponding to the various frames of the animation used for training. Individual operations of this engine (e.g. update of elemental rotations via Polar Decomposition, calculation of elastic forces, etc.) can optionally be invoked on a specific frame (or "slice" of the batch), or collectively across all batches, if such in-tandem execution provides performance benefits. We will see that such aggregate execution of certain operations across the entire batch is particularly meaningful in the back-propagation stage, for the evaluation of gradients.

In practice, most operations associated with forward evaluation – corresponding to an update of the quasistatic shape as a result of change in the actuation parameters – are performed on one slice of the batch at a time. This is due to the fact that there are adequate opportunities for efficient parallelization of the relevant simulation kernels by concurrently processing elements of the simulation mesh, that we do not need to resort to processing different animation frames in parallel. This also allows us to iterate the Projective Dynamics loop for as many times as needed for convergence on each individual animation frame, and we do observe that some frames of the animation certainly converge more rapidly than others.

Back-propagation involves substantially different circumstances and operations, which demand more attention to yield good scalability. In particular, the most expensive operation in the pipeline is the inversion of the stiffness matrix $\mathbf{K} = \partial\mathbf{f}/\partial\mathbf{x}$ in equation (13) in the process of computing the gradient of the loss with respect to the fine grained actions $\mathbf{a}$. This situation is made more complex by the fact that the stiffness matrix is, in fact, a function $\mathbf{K}_i = \mathbf{K}(\chi_i)$ of the quasistatic shape under the present value of the actions $\mathbf{a}_i$. Essentially, for each instance in the training set we are called upon to independently solve a system $\mathbf{K}_i\mathbf{h}_i = \mathbf{g}_i$ (where $\mathbf{g}_i = \partial L(\chi_i)/\partial\mathbf{x}$) with a different coefficient matrix and right hand side for each frame.

Thankfully, we can leverage an iterative solution, in the spirit of local defect correction schemes, which is similar to the local-global iterative solution in forward simulation of Projective Dynamics. Specifically, if we denote by $\mathbf{K}_{PD}$ the (constant; not deformation-dependent) matrix used in the global step of Projective Dynamics, we can solve the system $\mathbf{K}_i\mathbf{h}_i = \mathbf{g}_i$ using the iterative process:

```
Initialize h_i ← 0;
// fixed iteration count
for k = 1, 2, ..., N do
    r ← g_i − K_i h_i;          // matrix-free multiply with K_i
    δh ← K_PD^{-1} r;           // via Cholesky factorization
    h_i ← h_i + δh
end
```

Du et al. [2021] provided a good intuitive explanation of this iterative process; its convergence can be rigorously proven, tracing its foundation to the stability properties of the local-global iteration in forward simulation of Projective Dynamics. This methodology alleviates the need to construct or re-factorize every shape-dependent stiffness matrix $\mathbf{K}_i$, as the matrix $\mathbf{K}_{PD}$ can be prefactorized in advance, and the product $\mathbf{K}_i\mathbf{h}_i$ can be computed in a matrix-free fashion, as we do in our simulator (the product is in essence a force differential, which can be computed according to the derivations in Klàr et al. [2020]). In fact, under this formulation the most computationally expensive part of this solution (and of the back-propagation algorithm, in general) is the solution of $\delta\mathbf{h} \leftarrow \mathbf{K}_{PD}^{-1}\mathbf{r}$ via forward- and back-substitution on a precomputed Cholesky factorization, which needs to be performed for every animation frame in the training set. However, since now the matrix to be inverted for all frames in a batch has become the same, constant matrix $\mathbf{K}_{PD}$, this task can be recast as a forward- and back-substitution with multiple right hand sides (as many as training frames). This is typically at least an order of magnitude faster than solving each system independently; the substitution operations are highly memory-bound when operating on a single right-hand-side, while using several allows this cost to be amortized, with significant performance gains.

4.2 Matrix-free multiplication with force/action Jacobian

Equation (13) involves a multiplication of the vector $\mathbf{h}_i = \mathbf{K}_i^{-1} \frac{\partial L(\chi_i)}{\partial \mathbf{x}}$ (computed using the optimizations of the preceeding section) with the Jacobian $\partial \mathbf{f}(\chi_i, \mathbf{a}_i)/\partial \mathbf{a}$ to form the product $\mathbf{h}_i^T \frac{\partial \mathbf{f}}{\partial \mathbf{a}}$. This is a challenging task in terms of efficiency; the Jacobian $\partial \mathbf{f}/\partial \mathbf{a}$ is a matrix with such large dimensions (remember that $\mathbf{a}$ includes 6 scalars for each tetrahedron) that could only realistically be stored in a sparse format. The assembly of such matrix would be both cumbersome and expensive, and would have to be repeated for each animation frame, as this Jacobian is dependent on the current quasistatic shape, which is different across the batch. All such matrices (which would be very impractical to store) would have to be recomputed at every back-propagation pass.

We circumvent such challenges by computing the product $\mathbf{h}^T \frac{\partial \mathbf{f}}{\partial \mathbf{a}}$ (we drop the animation frame indices i, for simplicity) without explicitly constructing the Jacobian $\partial \mathbf{f}/\partial \mathbf{a}$, but instead applying it to this algebraic operation in a matrix-free fashion. Let us start by revisiting the equation of the Piola stress in our specific actuation model [Klár et al. 2020]:

$$\mathbf{P}(\mathbf{F}, \mathbf{S}_t) = 2\mu(\mathbf{F} - \mathbf{R}_* \mathbf{S}_t).$$

Where $\mathbf{R}_*$ is the rotational component of $\mathbf{F}\mathbf{S}_t$. We observe that a tetrahedron-specific shape target $\mathbf{S}_t$ (corresponding to six of the degrees of freedom in $\mathbf{a}$, corresponding to the element in question) will only induce stress – and thus, elastic force – on the same element. This the Jacobian $\partial \mathbf{f}/\partial \mathbf{a}$ is conceptually assembled from *elemental* contributions: each will be a sub-matrix which will be non-zero only on the 6 columns corresponding to the actuation values of this element, and the 3×4 rows corresponding to the forces on the four tetrahedron vertices. It is thus sufficient to focus on the contribution of each individual simulation tetrahedron to the product $\mathbf{h}^T \frac{\partial \mathbf{f}}{\partial \mathbf{a}}$; all such contributions will be additively combined. In doing so, we can also restrict the vector $\mathbf{h}$ to just its values on the four vertices of the element in question.

Thus, the i-th component of the product $\mathbf{h}^T \frac{\partial \mathbf{f}}{\partial \mathbf{a}}$ is computed as:

$$\mathbf{h}^T \frac{\partial \mathbf{f}}{\partial \mathbf{a}_i} = \frac{\partial}{\partial \mathbf{a}_i} [\mathbf{h}^T \mathbf{f}(\mathbf{x}, \mathbf{S}_t(\mathbf{a}))] = \mathbf{h}^T \delta \mathbf{f}(\mathbf{x}, \mathbf{S}_t; \delta \mathbf{S}_t \leftarrow \frac{\partial \mathbf{S}_t}{\partial \mathbf{a}_i})$$

Here, the derivative $\partial \mathbf{S}_t/\partial \mathbf{a}_i$ – which is constant – is trivially computable from Equation 4, while force differentials are directly computable [Sifakis and Barbič 2012] from the differential of the Piola Stress with respect to a variation $\delta \mathbf{S}_t$ in the shape target. From the definition of $\mathbf{P}$ previously given, this differential is given as:

$$\delta \mathbf{P}(\mathbf{F}, \mathbf{S}_t; \delta \mathbf{S}_t) = -2\mu(\delta \mathbf{R}_*)\mathbf{S}_t - 2\mu \mathbf{R}_*(\delta \mathbf{S}_t)$$

and, finally, the differential of the Polar Decomposition factor $\mathbf{R}_*$ with respect to a variation in $\mathbf{S}_t$ is (following the derivations of Klár et al. [Klár et al. 2020]):

$$\delta \mathbf{R}_*(\mathbf{F}, \mathbf{S}_t; \delta \mathbf{S}_t) = \mathbf{U}_* \{ \mathcal{E} : [\mathbf{K}^{-1} \mathcal{E}^T (\mathbf{U}_*^T \mathbf{F} \delta \mathbf{S}_t \mathbf{V}_*)] \} \mathbf{V}_*^T$$

where $\mathbf{U}_*, \Sigma_*, \mathbf{V}_*$ are the SVD factors of the product $\mathbf{F}\mathbf{S}_t$, $\mathcal{E}$ is the alternating tensor, and $\mathbf{K} = \mathrm{tr}(\Sigma_*)\mathbf{I} - \Sigma_*$ [Klár et al. 2020]. All the aforementioned calculations are easy to parallelize over the elements of the tetrahedral mesh, share computation among the partial derivatives $\partial/\partial \mathbf{a}_i$ relative to the six components of the elemental action parameter, and can be aggregated to compute the overall product $\mathbf{h}^T \frac{\partial \mathbf{f}}{\partial \mathbf{a}}$ without explicit formation of the force Jacobian.

4.3 Sparse layers

The architecture we would conventionally use for the decoder network that is prefixed to the differentiable simulator would be one of multiple layers, with fully-connected topology between successive ones. The size, however, of our output layer (1.5M scalars, for our 250K element face model, with 6 scalars per tetrahedron) would yield a prohibitive size and density of the network connections in the last two layers (in our experience, a hierarchical, fully connected network would exhaust the memory even of large-memory GPUs with just a few tens of thousands of simulation elements).

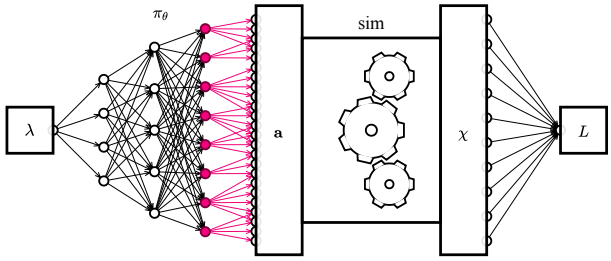

Fig. 5. The final layer in the decoder is a sparse layer with a custom connectivity determined from the topological neighborhood of the volumetric mesh. This design decision is important to allow us to scale the network to output 1.5M scalars, required to specify a full set of tetrahedral actions.

In order to avoid this prohibitive storage and complexity cost, we designed a custom, sparse topology, specifically for the last layer of the decoder network (Figure 5). Our design was motivated by the observation that, when training on more modest-resolution models that would be accommodated with a fully-connected network topology, the actuation signals that were learned in neighboring simulation elements appeared to be, in general, highly correlated. We hypothesized that an "upsampling" layer, that would have a sparse, but localized connectivity would perform well in this instance. We designed such a layer by using a regularly sampled subset of the elements in our simulation mesh (see the conceptual illustration in Figure 6; samples indicated by the small red circles), and associated each of these sparse samples with a node in the second-to-last layer of the decoder network. Since we used embedded simulation in our work, with lattice-derived embedding meshes, the chosen samples among the simulation elements were simply chosen by taking regular strides along the background Cartesian lattice.

For each element in this sparser set, we computed the topological neighborhood, specifically in our examples a 9×9 box surrounding each of the sparsely selected "centers". The extent of this topological neighborhood would be used to establish network connections between nodes in the last two layers of the decoder; a node on the second-to-last layer would be connected only to those nodes on the last layer that would be interpreted as belonging in the topological neighborhood of the sparsely sampled node. In a sense, the topology has a conceptual similarity to that of a convolutional network,

but with a distinct convolution stencil associated with each node in the interior layer, and weights that are learned during training. We found this paradigm to be very effective in our face simulation examples, while allowing high-resolution simulation models to be used within our pipeline.

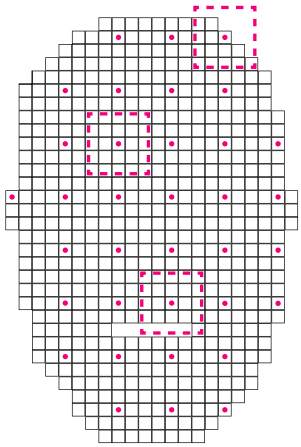

Fig. 6. Leveraging our observation that fully-connected networks when trained on modest resolution models learn highly correlated actuation signals around their neighborhood, we select centers (colored dots) in a sparse manner and a neighborhood around them (dotted squares) to establish custom connectivity for the sparse layer. Note that the number of neighbors, and hence the kernel size, is not the same for each cluster center.

5 TRAINING METHODOLOGY

In this section we present specific design choices for our network training strategy, mainly aimed at bootstrapping the training process with good quality initialization, and improving data fit and generalization. We highlight this process with a focus on our data set of facial expressions; this collection included 694 frames in varied facial expressions, which were separated into 533 frames for training, and 161 frames held aside for testing.

5.1 Initialization via an auto-encoder (no simulation)

Although our end-to-end training training procedure involves a decoder network followed by the differentiable simulator, as originally presented in Figure 3, we used an auto-encoder-type network to train the decoder without using the simulator as an initialization step. This network receives as input a surface with a facial expression, and outputs the fine-grained elemental actuations that should result in the same expression resulting from volumetric, physics-based simulation. Of course, we do not have ground truth data for these actions. Moreover, actions that would produce that (or any other) surface deformation are not uniquely defined: In our mesh the degrees of freedom in the fine-grained actuation descriptor $(\mathbf{a})$ are about 10 times as many as the ones in the quasistatic simulation result (χ). As a consequence, any given simulated shape corresponds to an entire (nonlinear, and multi-dimensional) manifold of elemental actions that would yield exactly the same equilibrium configuration

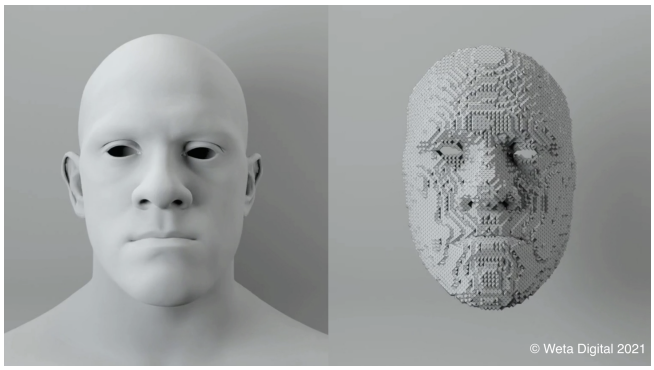

Fig. 7. The raw input to our pipeline is a surface mesh that specifies a target pose (left). In order to generate pairs of target poses and actions for our initial training procedure, we use a tetrahedral mesh (right) to target the surface pose (left) with the help of targeting springs with zero rest-length and compute the actuation for the corresponding pose from this deformed tetrahedral mesh.

– the difference being that different actions would reproduce the same shape with different degrees of *tension* throughout the model.

Even if ground-truth elemental actions are rather elusive, for the reasons we described, we still seek to furnish at least a *plausible approximation* of them, constructed from the input surface data. If we had such an approximation, and with the understanding that the result of this stage would simply be used for warm-starting the training of our end-to-end pipeline, we have the opportunity to train the auto-encoder network, as shown in Figure 8, without incorporating simulation in the training process. This training process is faster, and can be used to experimentally probe what is an appropriate dimension of the bottleneck of this auto-encoder, which will ultimately become the dimension of the latent parameter vector in our original (simulator-integrated) network.

We fabricate the approximation of elemental actions needed for this supervised training by a physics-based extrapolation of the surface deformation provided as input, into a corresponding deformation of the volumetric simulation mesh. We do so by attaching "targeting springs" with zero rest length between the surface vertex locations in the input animation, and the corresponding locations on the flesh surface embedded on our volumetric simulation mesh; we thus "drag and stretch" the volumetric simulation mesh to match the surface deformation given as input, see Figure 7. While doing so, we enforce any boundary conditions on skeletal attachments in the inner part of the flesh volume, and apply *no muscle action* in our actuation model; effectively the flesh material is pure corotated elasticity in this exercise. When we obtain a converged quasistatic simulation from this targeting operation, we extract the deformation gradient of each tetrahedron in the simulation mesh, and use the symmetric part from its polar decomposition as the approximate shape target $\mathbf{S}_t$ we will use to train our auto-encoder.

It should be emphasized that the fine-grained muscle actions thus computed are just an approximation of what true actions would have corresponded to a volumetric simulation that closely approximates the input surface shape. In particular, the actions thus computed

would bring the volumetric mesh towards the desired shape while exhibiting *zero stress or tension*. To put it in other terms, such fine-grained actions would have resulted in tetrahedral forces that are equal to zero, individually on every simulation element (as opposed to simply canceling out from the contributions of neighboring elements, but still exhibiting some tension). In our experiments we observed both positive and negative impacts of this approximation: the decoder trained via this process appeared to generally produce rather acceptable actions, especially in their ability to fit the training set, but was far from perfect when applied to data unseen in training (especially in poses with high flesh tension, as a wide opening of the mouth), as evaluated by simply taking the actions output by the auto-encoder and feeding them through a forward simulator. Nonetheless, despite the inaccuracies inherent in this first stage of training, the weights of the decoder network computed as part of this auto-encoder architecture proved to be a very effective initialization for the end-to-end training of our primary network architecture, which combines the decoder with a differentiable simulator.

We use a β-variational auto-encoder [Higgins et al. 2017] with 2 hidden layers in the encoder, 3 hidden layers in the decoder and a latent dimension of 60. All the layers of the encoder and all but the last layer of the decoder are fully-connected. The final layer of the decoder is designed to be a sparse layer by specifying custom connections between the neurons in the second to last layer and the last layer as illustrated in 5. The surface displacement is given as input to the encoder (76557 scalars, corresponding to 25519 surface vertices in 3D) and actions for each tetrahedral element are produced as output by the decoder (1475904 scalars). The encoder has 2 hidden layers with 4096 neurons in the first layer and 512 neurons in the second layer. Since we fixed the latent parameter dimension to 60, the output of the encoder is composed of 120 neurons (60 neurons for the mean and 60 neurons for the variance). The decoder has 3 hidden layers with 256, 4096 and 10224 neurons, followed by a sparse layer as described in Section 4.3. All the hidden layers use Leaky RELU activations. During every training epoch, the input is fed into the encoder which outputs 2 values: mean and variance, which are then used to sample from a normal distribution via the commonly used reparameterization trick [Kingma and Welling 2014]. The generated latent parameters λ are then fed into the decoder which outputs actions $\mathbf{a}$. To train the weights of this network, we compute the loss on the output actions and the KL-Divergence of the latent parameters, to measure how far their distribution is from the prior. This KL-Divergence term is also scaled by a hyperparameter β which encourages disentanglement between the latent parameters. We experimented with different β values for training and fixed β to be 0.01 for this dataset. We used Adam [Kingma and Ba 2015] to train the network weights, with a learning rate of 10^{-5}, and $\beta_1 = 0.9$ and $\beta_2 = 0.999$ for the running averages of the gradient and squared gradient. The sparsity introduced in the final layer of the decoder permitted us to train the network on a GPU using a batch size of 30, for a data set with 533 training samples. In the inference stage, the mean produced by the encoder for each input, is used as the latent parameter vector λ and fed into the decoder, followed by the simulator to reconstruct the input shape.

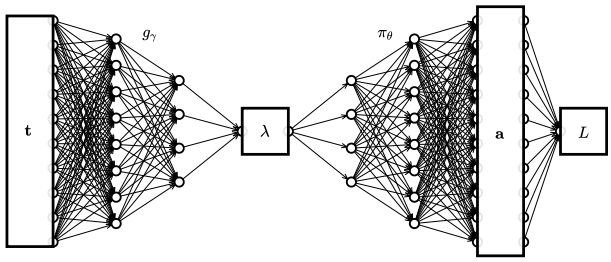

Fig. 8. As an initialization step, we used an auto-encoder network to train the decoder π_θ without using the simulator. This network takes as input a facial expression $\mathbf{t}$ (specified as vertex positions of the surface mesh), and outputs the fine-grained elemental actuations $\mathbf{a}$. We used a variational auto-encoder, which generates the latent variable λ by sampling from a normal distribution whose mean and variance are produced as outputs of the encoder g_γ (sampling procedure not shown in the figure).

5.2 Training via differentiable simulator

We use the weights for the decoder network from the auto-encoder trained via the procedure described in Section 5.1, and use them to warm-start an end-to-end training process on our intended network design that combines the decoder with a differentiable simulator, as depicted in Figure 4. During training of this network, both the weights of the decoder network and the values of the latent parmeters (λ) will be updated to better incorporate the effect of the simulator stage in the quasistatic shapes produced. We clarify that while only a single set of decoder network weights will result from this training, an entire batch of latent vectors will be optimized, each corresponding to a frame of the training set. The loss function itself is defined in terms of the discrepancy between the simulated and targeted surface vertex positions:

$$L = \sum_i L(\mathbf{a}_i, \mathbf{t}_i) \tag{14}$$

$$L(\mathbf{a}_i, \mathbf{t}_i) = \|\chi(\mathbf{a}_i) - \mathbf{t}_i\|^2 \tag{15}$$

$$\mathbf{a}_i = \pi_\theta(\lambda_i) \tag{16}$$

where $\mathbf{t}_i, \chi(\mathbf{a}_i), \mathbf{a}_i$, and λ_i are respectively the target surface vertex locations, their quasistatic simulated counterparts, the fine-grained actuations, and the low-dimensional latent parameters associated with the i-th frame of the training set. Exploiting the batch-optimized computation design of the simulator engine, we train the latent parameters and the decoder weights with a batch size of 30. To accommodate 533 training samples, we use 18 simulator instances. Since each simulator instance has persistent per-frame data required for simulation, maintaining a separate simulator instance for each batch speeds up the training pipeline. Another trick we use to speed up training is to customise the number of iterations for forward and backward passes in the simulator. Since we only need the direction of descent during training, with a small learning rate, we are able to train the model, without requiring the backward and forward passes through the simulator to converge completely. We used the Adam [Kingma and Ba 2015] optimizer to train the decoder network and used different learning rates for updating the decoder

weights and the latent parameters. We used a learning rate of 10^{-5} for the decoder weights and 10^{-3} for the latent parameters. We used the same values of β_1 and β_2 that we used for the auto-encoder training, and we also used L2-regularization, with a regularization factor of 10^{-1}, for both cases. Since we warm-start the training from the auto-encoder network, the end-to-end training of the decoder network requires fewer epochs. We train the decoder network on our dataset for 175 epochs. We also point out that the autoencoder network was trained with data (for its output) that were fabricated using a simplifying hypothesis, rather than corresponding to a hard ground-truth: we relaxed a volumetric mesh targeting the frontal surface and used the converged element shapes as shape targets. In doing so, we make a highly heuristic assumption that such shapes are reached with each element being tension-free (as opposed to having forces that balance out across neighboring elements). It is important to note that the loss function used in training this autoencoder was defined directly on these elemental shape targets, while for the end-to-end training, the loss is defined on the quasistatic simulated surface shape. In our examples, we observed that the prior auto-encoder training consistently generated a high-quality warm start for the training of this simulator-integrated network.

5.3 Optimizing for latent variables on unseen data

Our goal with the decoder training process was to capture some internal structure of the active object. To further explore the properties of the learned latent spaces, we perform additional experiments where we fix the parameters of the decoder network π_{θ^*} and run a new optimization process to find optimal latent variables λ^* that best match a new target expression $\mathbf{t}_*$

$$\lambda^* = \underset{\lambda}{\operatorname{argmin}} \, \|\chi(\pi_{\theta^*}(\lambda)) - \mathbf{t}_*\|^2 \tag{17}$$

Of course, the data target $\mathbf{t}_*$ involved in this optimization would be some unseen expression, not necessarily a part of the training set. Intuitively, here we keep the learnt actuation mechanism fixed, as resulting from the training of our network, and only focus on determining the latent parameters that drive this mechanism to a given pose; essentially a standard problem of inverse control for a given actuation mechanism. We note that the norm in Equation 17 could be taken either over the entire frontal surface vertices (if we wish to "project" an input expression to our action space), or a sparse subset thereof (e.g. for expression reconstruction from motion capture data, or sculpting from direct manipulation of a small number of vertices). For motion capture, we use 35 sparse markers as shown in Figure 11 and use unseen poses from the test data to simulate motion capture data. We include only these 35 markers in the norm in Equation 17 and find the set of latent parameters that best fit the specified marker positions. We use the Adam [Kingma and Ba 2015] optimizer with a learning rate of 10^{-1} to optimize the latent parameters in the inverse control setup.

6 EXPERIMENTS AND EVALUATION

We draw our primary set of examples from facial animation, with an additional controlled experiment on simulated, synthetic data on a human limb.

Fig. 9. Using our differentiable simulator, we can produce a more accurate reconstruction (right) of the target expression (left) than an auto-encoder which only operates on surface shapes and was not trained using the differentiable simulator (middle). The colors indicate reconstruction error, where blue means low error and red means high error.

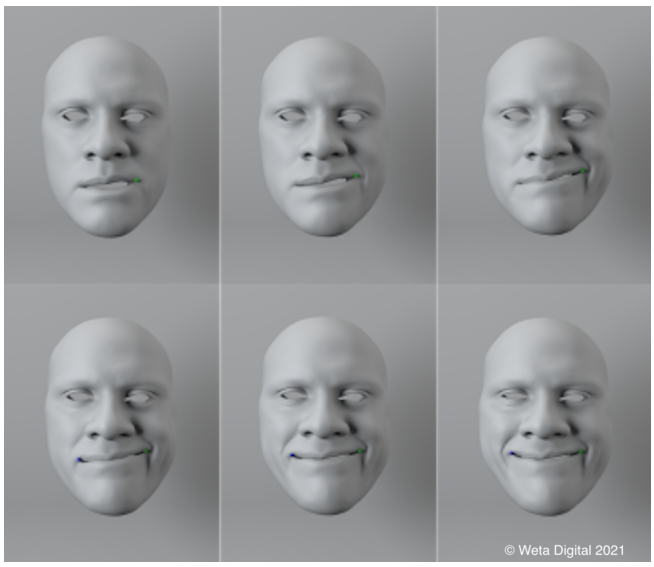

Fig. 10. Our trained model can be used to generate physically plausible expressions via direct manipulation of sparse targets (blue and green markers). The top 3 images (left-to-right) show results from our pipeline after manipulating the green marker. The bottom 3 images (left-to-right) show the results from our pipeline after manipulating the blue marker, while keeping the green marker fixed.

6.1 Facial animation examples

We draw upon a training set of facial performance data, acquired from an actor that was instructed to articulate their face between several expressive poses. Our training set includes 694 frames, 533 of which are used for training, and 161 frames for testing. The data set we had at our disposal had kinematic information for the mandible, thus boundary conditions associated with skeletal attachments were presumed known at all time. We employ embedded simulation on a background (embedding) simulation mesh constructed from a BCC

Fig. 11. Motion Capture. One of the applications of our pipeline is to target unseen poses using motion capture markers. The position of the markers (shown on the right) from the unseen pose (left) is given as input to our system. The output from our pipeline is on the right. The red markers correspond to vertices of the surface reconstructed by our embedded simulation and the blue markers correspond to the target positions.

lattice template, with a total of 246K tetrahedral elements and 52K vertices. Our supplemental video provides footage associated with the following experiments and evaluation exercises:

Construction of approximate shape targets. As detailed in Section 5.1, we perform a simulation-based extrapolation of the surface data into a volumetric deformation, by targeting the skin surface embedded on the simulation mesh towards the shape provided in the training data (using zero-restlength springs). Approximate shape targets are constructed from the symmetric part of the polar decomposition of the deformation gradient for each simulation tetrahedron. We observed that the use of highly regular, uniform tetrahedra with good aspect ratios was particularly important for our system to have good training and reconstruction performance, hence our choice to use embedding. Prior experiments with conforming meshes with non-ideal aspect ratios would create approximate actions with range beyond what is intutively expected – a way to interpret this is that a flat or sliver tetrahedron might need to scale one of its dimensions very drastically even to assume a shape that is not very remote from its rest shape in absolute distance terms. Our embedded simulation enjoyed very stable performance in this step. Our video depicts the results of this volumetric extrapolation of the training shapes.

Autoencoder training and evaluation. In accordance with Section 5.1 we train an autoencoder that maps surface shapes to estimated tetrahedral actions, supervised during training using the elemental shape targets just computed. Even though this network is trained without the intervention of a simulator stage, we can simply take the resulting fine-grain controls and feed them to a simulator, in order to visually assess how close the result is to the surface originally given as input (Figure 9). Each epoch of training for the autoencoder takes around 23s on an NVidia Quadro RTX 8000 GPU. In our video, we examine the reconstruction accuracy of this step, both for the training and test datasets.

End-to-end training via simulation. Using the decoder stage of the autoencoder just trained as initialization, we assemble our decoder-simulator network and continue to train it via backpropagation as in section Section 5.2. Training 533 frames on Intel i9-9940X CPU (28 cores, 3.30GHz) takes 1603.92s. Our video demonstrates the reconstruction error of this network, and compares it with the prior result of only using the autoencoder controls.

Inverse control. New poses can be fit against our trained model, by keeping the decoder weights fixed, and optimizing for the latent parameters that yield the best fit. We evaluate this task on our end-to-end trained network, and also contrast to an alternative autoencoder that could be crafted with no simulation (or volumetric deformation) in the loop, that simply performs dimensionality reduction of the animation samples to the same number of parameters as our latent control dimension. We observe our system to be generally superior, and especially so in frames that have mandible motion. It should also be noted that our system provides a full physics-based, simulation-oriented description of the expression, which can be manipulated in additional useful ways, such as incorporating collisions, constraints, volume conservation, etc.

Motion capture and direct manipulation. One very practical application of a parametric, controllable face puppet as the one we effectively learn from data would be the generation of detailed facial expressions from sparse motion capture data. We emulate such a scenario by capturing the trajectories of a small set of hand-selected "marker locations" from input shapes that were not part of the training set (effectively, a sparse sampling of unseen data that was also used in our prior reconstruction experiment). We then formulate a loss function that seeks to optimize the sum of squared distances between the target markers and their simulated counterparts, and optimize the latent parameters to compute a fit (Figure 11). Our results, which can be seen in our video, illustrate that even a sparse marker set articulates the face model in a way that allows detailed features such as folds, bulges and wrinkles to realistically emerge. Finally, we also demonstrate a "direct manipulation" session, where an artist might directly sculpt a facial expression by dragging a mesh vertex (Figure 10), while traversing the parameter space of this puppet (which, in our case, is action- and simulation-driven, as opposed to procedural blendshapes).

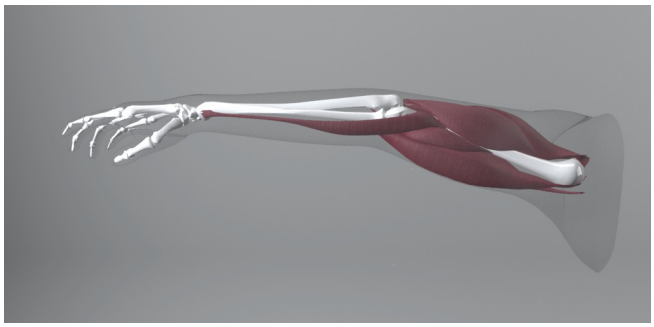

Fig. 12. Muscle-driven simulation of the arm. We included four active muscles: the *biceps, triceps, brachialis*, and *brachioradialis*.

6.2 Learning a synthetic elbow model

The facial dataset we had at our disposal included skeletal motion (for the mandible) to a relatively small degree, and for a smaller fraction of the available frames. We wanted to evaluate the ability of our model, which is action-centric as opposed to pose- or shape-centric, to learn the actuation mechanism even in the presence of substantial (albeit, known) skeletal motion. For this specialized example, we created our training set by muscle-driven simulation on the upper extremity; we included four active muscles: the *biceps, triceps, brachialis*, and *brachioradialis* shown in Figure 12. For this experiment we create a tetrahedral embedding mesh 130K elements and approximately 75k degrees of freedom in total. We simulated a sequence where the four muscles activate with a sinusoidal temporal pattern, with different period for each, and also a different period for the kinematic bending of the elbow joint. We repeated the same pipeline as in our facial examples, and demonstrate the performance of our model to fit unseen poses, beyond those in the training set.

7 LIMITATIONS AND FUTURE WORK

A notable characteristic of our approach that could variably be considered either as a limitation or a possible trait of versatility is that when we use our framework to approximate a model for which we have a strong anatomical prior (e.g. muscle-driven bodies or faces) we do not have explicit guarantees that the actuation system we will infer will be identical, or even similar enough to the anatomical ground-truth. A good example of this is the frontalis muscle, and its action in generating corrugations in the forehead. The muscle itself, structured largely as a sheet, is relatively simple in its operation (produces a contractile potential along largely parallel lines). The high-level observed effect (forehead wrinkles) is a consequence of many additional contributing factors: inextensibility of the skin surface, sliding on top of the cranium, and the effect of connective tissue. If our simulation model does not explicitly incorporate these anatomical factors (and our tetrahedral simulation mesh consciously does not), it is quite likely that the mechanism that will be deduced from data would be one that directly triggers the formation of wrinkles by shearing elements to create bending and folding of skin. This can be a disadvantage, when conforming to the anatomically accurate structures is important, which could be the case, for example if we want to have the ability to warp this actuation system to another human subject or non-human creature, or if we wanted the control parameters of such different models to be analogous. On the other hand, the fact that we are not constrained to be anatomically accurate alleviates the need for perfect geometric modeling of anatomy, and still gives us the possibility to generate a plausible physics-based expression space.

The use of shape targeting [Ichim et al. 2017; Klár et al. 2020] as the underlying fine-grained actuation model was motivated by its stability properties and the moderate nonlinearity it involves. It has also been argued [Klár et al. 2020] that it is a plausible generalization of fiber-based muscle actuation models. It does come, however, with certain limitations; principal among them is that the "action" of the contractile elements principally dictates their desired rest shape, *not their stiffness!* For certain parts of anatomical models, the distinction is substantial: tendons are characterized by their highly anisotropic stiffness, and modeling them as such is essential to their biomechanical function (including their ability to generate adequate torque to move the skeleton). Since our actuation mechanism is primarily geared towards creating *shapes* rather than *forces*, it is questionable if our approach would be directly applicable to a musculoskeletal application where the primary function would be supporting skeletal motion (as opposed to deforming the skin). It would be a fascinating topic of inquiry to explore learning-based models that would be capable of also attenuating their stiffness, and inferring such properties from data.

Finally, our examples in this iteration of our work do not include collision effects as a component of the training process, or the refinement process that determines the values of the latent parameters that best fit an unseen face shape. We do include some demonstrations of collision processing as a post-process (while replaying precomputed elemental actions), but not as a part of our optimization pipeline. We do, in fact, suffer some isolated consequences of this omission, for example, the absence of lip/gum collisions in the jaw makes it difficult to have a good match with the animation data in scenarios where the jaw is widely open. Although this is a feature that is presently not incorporated into our pipeline, we strongly believe that our physics-based approach creates much more favorable conditions for either training models through scenarios that involve collision, or reconstructing expressions (e.g. via motion capture) while respecting collisions. Since we have seen examples of inverse control in the presence of self- and body-collisions [Sifakis et al. 2005], it is reasonable to expect that tuning the latent parameters of a pre-learned mechanism to target poses through collisions should be quite possible. Training will be more challenging; one immediate impact in our approach would be compromising the opportunity to reuse the same, constant, pre-factorized matrix from the global step of Projective Dynamics when evaluating the gradients for back-propagation; this would, however, be a fascinating opportunity for research that would unlock significant new capabilities.

ACKNOWLEDGEMENTS

We thank Luca Fascione and Stephen Cullingford for valuable contributions on assets and administrative support. This research was supported in part by National Science Foundation grants CCF-1812944, IIS-1763638, IIS-1764071, IIS-2008915, IIS-2008564 and IIS-2008584.

REFERENCES

Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein generative adversarial networks. In *International conference on machine learning*. 214–223.

Timur Bagautdinov, Chenglei Wu, Jason Saragih, Pascal Fua, and Yaser Sheikh. 2018. Modeling Facial Geometry Using Compositional VAEs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

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).

Michael Bao, Matthew Cong, Stephane Grabli, and Ronald Fedkiw. 2019. High-Quality Face Capture Using Anatomical Muscles. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

GG Barbarino, M Jabareen, J Trzewik, A Nkengne, G Stamatas, and E Mazza. 2009. Development and validation of a three-dimensional finite element model of the face. *Journal of biomechanical engineering* 131, 4 (2009), 041006.

James Bern, Grace Kumagai, and Stelian Coros. 2017b. Fabrication, Modeling, and Control of Plush Robots. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 3739 – 3746. 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2017); Conference Location: Vancouver, Canada; Conference Date: September 24-28, 2017.

Fig. 13. Left-to-right. Tetrahedral mesh (column 1) deformed to match the surface poses (column 2). Output form the auto-encoder is shown in column 3. Reconstructions form the end-to-end trained decoder network is shown in column 4. Column 5 shows the result after refining the latent parameters on unseen poses from test data.

James M Bern, Pol Banzet, Roi Poranne, and Stelian Coros. 2019. Trajectory optimization for cable-driven soft robot locomotion. *Proceedings of Robotics: Science and Systems* (2019).

James M. Bern, Kai-Hung Chang, and Stelian Coros. 2017a. Interactive Design of Animated Plushies. *ACM Trans. Graph.* 36, 4, Article 80 (July 2017), 11 pages.

Volker Blanz and Thomas Vetter. 1999. A morphable model for the synthesis of 3D faces. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. 187–194.

Silvia Salinas Blemker. 2004. *3D modeling of complex muscle architecture and geometry.* Ph.D. Dissertation. Stanford University.

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. 2014. Projective dynamics: fusing constraint projections for fast simulation. *Proc. of ACM SIGGRAPH* 33, 4 (2014).

Matthew Cong, Kiran S Bhat, and Ronald Fedkiw. 2016. Art-directed muscle simulation for high-end facial animation. 119–127.

Stelian Coros, Sebastian Martin, Bernhard Thomaszewski, Christian Schumacher, Robert Sumner, and Markus Gross. 2012. Deformable objects alive! *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 69.

Kai Ding, Libin Liu, Michiel van de Panne, and KangKang Yin. 2015. Learning Reduced-Order Feedback Policies for Motion Skills. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (Los Angeles, California) *(SCA*

'15). Association for Computing Machinery, New York, NY, USA, 83–92.
Carl Doersch. 2016. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908* (2016).
Tao Du, Kui Wu, Pingchuan Ma, Sebastien Wah, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. 2021. DiffPD: Differentiable Projective Dynamics with Contact. arXiv:2101.05917 [cs.LG]
Paul Ekman and Wallace V Friesen. 1977. Facial action coding system. (1977).
Ye Fan, Joshua Litven, and Dinesh K Pai. 2014. Active volumetric musculoskeletal systems. *Proc. of ACM SIGGRAPH* 33, 4 (2014), 152.
François Faure, Christian Duriez, Hervé Delingette, Jérémie Allard, Benjamin Gilles, Stéphanie Marchesseau, Hugo Talbot, Hadrien Courtecuisse, Guillaume Bousquet, Igor Peterlik, et al. 2012. Sofa: A multi-model framework for interactive physical simulation. In *Soft tissue biomechanical modeling for computer assisted surgery*. Springer, 283–321.
Cormac Flynn, Ian Stavness, John Lloyd, and Sidney Fels. 2015. A finite element model of the face including an orthotropic skin model under in vivo tension. *Computer methods in biomechanics and biomedical engineering* 18, 6 (2015), 571–582.
Moritz Geilinger, David Hahn, Jonas Zehnder, Moritz Bächer, Bernhard Thomaszewski, and Stelian Coros. 2020. ADD: analytically differentiable dynamics for multi-body systems with frictional contact. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–15.
Zhenglin Geng, Daniel Johnson, and Ronald Fedkiw. 2020. Coercing machine learning to output physically accurate results. *J. Comput. Phys.* 406 (Apr 2020), 109099.
Evgeny Gladilin. 2003. Biomechanical modeling of soft tissue and facial expressions for craniofacial surgery planning. *Freien University, Berlin* (2003).
Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.
David Hahn, Pol Banzet, James M Bern, and Stelian Coros. 2019. Real2sim: Visco-elastic parameter estimation from dynamic motion. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–13.
I. Higgins, Loïc Matthey, A. Pal, C. Burgess, Xavier Glorot, M. Botvinick, S. Mohamed, and Alexander Lerchner. 2017. beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. In *ICLR*.
Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. 2020. DiffTaichi: Differentiable Programming for Physical Simulation. In *International Conference on Learning Representations*.
Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B Tenenbaum, William T Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. 2019. Chainqueen: A real-time differentiable physical simulator for soft robotics. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 6265–6271.
Alexandru Ichim, Ladislav Kavan, Merlin Nimier-David, and Mark Pauly. 2016. Building and Animating User-Specific Volumetric Face Rigs.
Alexandru-Eugen Ichim, Petr Kadleček, Ladislav Kavan, and Mark Pauly. 2017. Phace: Physics-based face modeling and animation. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 153.
Petr Kadlecek and Ladislav Kavan. 2019. Building Accurate Physics-Based Face Models from Data. In *Symposium on Computer Animation*.
Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.
Diederik P. Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.
Gergely Klár, Andrew Moffat, Ken Museth, and Eftychios Sifakis. 2020. Shape Targeting: A Versatile Active Elasticity Constitutive Model. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks (SIGGRAPH '20)*. Association for Computing Machinery, Article 59, 2 pages.
Yeara Kozlov, Derek Bradley, Moritz Bächer, Bernhard Thomaszewski, Thabo Beeler, and Markus Gross. 2017. Enriching Facial Blendshape Rigs with Physical Simulation. In *Computer Graphics Forum*, Vol. 36. Wiley Online Library, 75–84.
Lana Lan, Matthew Cong, and Ronald Fedkiw. 2017. Lessons from the evolution of an anatomical facial muscle model. In *Proceedings of the ACM SIGGRAPH Digital Production Symposium*. ACM, 11.
John P Lewis, Ken Anjyo, Taehyun Rhee, Mengjie Zhang, Frederic H Pighin, and Zhigang Deng. 2014. Practice and Theory of Blendshape Facial Models. *Eurographics (State of the Art Reports)* 1, 8 (2014), 2.
Jiaman Li, Zhengfei Kuang, Yajie Zhao, Mingming He, Karl Bladin, and Hao Li. 2020b. Dynamic facial asset and rig generation from a single scan. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–18.
Ruilong Li, Karl Bladin, Yajie Zhao, Chinmay Chinara, Owen Ingraham, Pengda Xiang, Xinglei Ren, Pratusha Prasad, Bipin Kishore, Jun Xing, et al. 2020a. Learning Formation of Physically-Based Face Attributes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3410–3419.
John E Lloyd, Ian Stavness, and Sidney Fels. 2012. ArtiSynth: A fast interactive biomechanical modeling toolkit combining multibody and finite element simulation. In *Soft tissue biomechanical modeling for computer assisted surgery*. Springer, 355–394.
Stephen Lombardi, Jason Saragih, Tomas Simon, and Yaser Sheikh. 2018. Deep appearance models for face rendering. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 68.
Steve A Maas, Gerard A Ateshian, and Jeffrey A Weiss. 2017. FEBio: History and advances. *Annual review of biomedical engineering* 19 (2017), 279–299.
Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus Gross. 2011. Example-based elastic materials. In *ACM Transactions on Graphics (TOG)*, Vol. 30. ACM, 72.
Aleka McAdams, Yongning Zhu, Andrew Selle, Mark Empey, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. 2011. Efficient elasticity for character skinning with contact and collisions. *ACM Transactions on Graphics (TOG)* 30, 4 (2011), 37.
Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. 2004. Fluid Control Using the Adjoint Method. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 449–456.
Nathan Mitchell, Court Cutting, and Eftychios Sifakis. 2015. GRIDiron: An interactive authoring and cognitive training foundation for reconstructive plastic surgery procedures. *ACM Trans. Graph. (Proceedings of ACM SIGGRAPH)* (2015).
Paola Nardinocchi and Luciano Teresi. 2007. On the active response of soft living tissues. *Journal of Elasticity* 88, 1 (2007), 27–39.
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.
Michael Schmidt and Hod Lipson. 2009. Distilling Free-Form Natural Laws from Experimental Data. *Science* 324, 5923 (2009), 81–85.
Gabriel Schwartz, Shih-En Wei, Te-Li Wang, Stephen Lombardi, Tomas Simon, Jason Saragih, and Yaser Sheikh. 2020. The eyes have it: an integrated eye and face model for photorealistic facial animation. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 91–1.
Eftychios Sifakis and Jernej Barbič. 2012. FEM Simulation of 3D Deformable Solids: A practitioner's guide to theory, discretization and model reduction. http://www.femdefo.org.
Eftychios Sifakis, Igor Neverov, and Ronald Fedkiw. 2005. Automatic determination of facial muscle activations from sparse motion capture marker data. In *Proc. of ACM SIGGRAPH*, Vol. 24. 417–425.
Eftychios Sifakis, Tamar Shinar, Geoffrey Irving, and Ronald Fedkiw. 2007. Hybrid simulation of deformable solids. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association, 81–90.
Breannan Smith, Chenglei Wu, He Wen, Patrick Peluse, Yaser Sheikh, Jessica K Hodgins, and Takaaki Shiratori. 2020. Constraining dense hand surface tracking with elasticity. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–14.
Martin Spüler, Nerea Irastorza Landa, Andrea Sarasola Sanz, and Ander Ramos-Murguialday. 2016. Extracting Muscle Synergy Patterns from EMG Data Using Autoencoders. In *Artificial Neural Networks and Machine Learning – ICANN 2016*. 47–54.
Ian Stavness, Mohammad Ali Nazari, Cormac Flynn, Pascal Perrier, Yohan Payan, John E Lloyd, and Sidney Fels. 2014. Coupled biomechanical modeling of the face, jaw, skull, tongue, and hyoid bone. In *3D Multiscale Physiological Human*. Springer, 253–274.
Jie Tan, Greg Turk, and C Karen Liu. 2012. Soft body locomotion. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 26.
J Rafael Tena, Fernando De la Torre, and Iain Matthews. 2011. Interactive region-based linear 3d face models. In *ACM SIGGRAPH 2011 papers*. 1–10.
Joseph Teran, Sylvia Blemker, V Hing, and Ronald Fedkiw. 2003. Finite volume methods for the simulation of skeletal muscle. Eurographics Association, 68–74.
Joseph Teran, Eftychios Sifakis, Silvia S Blemker, Victor Ng-Thow-Hing, Cynthia Lau, and Ronald Fedkiw. 2005. Creating and simulating skeletal muscle from the visible human data set. *IEEE TVCG* 11, 3 (2005), 317–328.
Shih-En Wei, Jason Saragih, Tomas Simon, Adam W Harley, Stephen Lombardi, Michal Perdoch, Alexander Hypes, Dawei Wang, Hernan Badino, and Yaser Sheikh. 2019. Vr facial animation via multiview image translation. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–16.
Jeffrey A Weiss, Bradley N Maker, and Sanjay Govindjee. 1996. Finite element implementation of incompressible, transversely isotropic hyperelasticity. *Computer methods in applied mechanics and engineering* 135, 1 (1996), 107–128.
Chris Wojtan, Peter J. Mucha, and Greg Turk. 2006. Keyframe control of complex particle systems using the adjoint method. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Vienna, Austria). Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 15–23.
Chenglei Wu, Derek Bradley, Markus Gross, and Thabo Beeler. 2016. An anatomically-constrained local deformation model for monocular face capture. *ACM transactions on graphics (TOG)* 35, 4 (2016), 1–12.
Felix E Zajac. 1989. Muscle and tendon Properties models scaling and application to biomechanics and motor. *Critical reviews in biomedical engineering* 17, 4 (1989), 359–411.
Xiaotian Zhang, Fan Kiat Chan, Tejaswin Parthasarathy, and Mattia Gazzola. 2019. Modeling and simulation of complex dynamic musculoskeletal architectures. *Nature communications* 10, 1 (2019), 1–12.