Analytically Integratable Zero-restlength Springs for Capturing Dynamic Modes unrepresented by Quasistatic Neural Networks

YONGXU JIN, Stanford University, USA and Epic Games, USA
YUSHAN HAN, University of California, Los Angeles, USA and Epic Games, USA
ZHENGLIN GENG, Epic Games, USA
JOSEPH TERAN, University of California, Davis, USA, University of California, Los Angeles, USA, and Epic Games, USA
RONALD FEDKIW, Stanford University, USA and Epic Games, USA

Fig. 1. Our method enhances standard skinning with a configuration-only quasistatic neural network (QNN) that approximates quasistatic hyperelasticity as well as analytically integratable zero-restlength springs trained that approximates inertial effects. The QNN fixes well-known skinning artifacts (e.g. in the shoulder regions) and the zero-restlength springs add ballistic motion (e.g. in the belly region). We refer readers to our supplementary video which is far more compelling than still images.

We present a novel paradigm for modeling certain types of dynamic simulation in real-time with the aid of neural networks. In order to significantly reduce the requirements on data (especially time-dependent data), as well as decrease generalization error, our approach utilizes a data-driven neural network only to capture quasistatic information (instead of dynamic or time-dependent information). Subsequently, we augment our quasistatic neural network (QNN) inference with a (real-time) dynamic simulation layer. Our key insight is that the dynamic modes lost when using a QNN approximation can be captured with a quite simple (and decoupled) zero-restlength spring model, which can be integrated analytically (as opposed to numerically) and thus has no time-step stability restrictions. Additionally, we demonstrate that the spring constitutive parameters can be robustly learned from a surprisingly small amount of dynamic simulation data. Although we illustrate the efficacy of our approach by considering soft-tissue dynamics on animated human bodies, the paradigm is extensible to many different simulation frameworks.

CCS Concepts:
- Computing methodologies → Animation: Neural networks; Physical simulation.

Additional Key Words and Phrases: Zero-restlength spring, soft-tissue dynamics, human body animation

ACM Reference Format:
Yongxu Jin, Yushan Han, Zhenglin Geng, Joseph Teran, and Ronald Fedkiw. 2022. Analytically Integratable Zero-restlength Springs for Capturing Dynamic Modes unrepresented by Quasistatic Neural Networks. 1, 1 (January 2022), 8 pages. https://doi.org/XXXXXXX.XXXXXXX

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2022 Association for Computing Machinery.
1 INTRODUCTION

Recently, there has been a lot of interest in using neural networks to approximate dynamic simulation (see e.g. [Holden et al. 2019; Pfaff et al. 2020; Sanchez-Gonzalez et al. 2020; Santesteban et al. 2020]), especially because neural network inference has the potential to run in real-time (on high-end GPUs). Unfortunately, one requires an exorbitant amount of training data in order to represent all the possible temporal transitions between states that these networks aim to model. These networks do not typically generalize well when not enough training data is used. Even if one had access to the exorbitant amount of training data required, an unwieldy amount of network parameters would be required to prevent underfitting.

Some aspects of a dynamic simulation depend mostly on the configuration, whereas others more strongly depend on the time history of configuration to configuration transitions; thus, we propose the following paradigm. Firstly, we construct a neural network that depends only on the configurations (and as such cannot capture dynamic modes). Secondly, we subtract this configuration-only model from the full dynamics in order to obtain a dynamics layer. Thirdly, we propose a dynamic simulation model that can approximate the dynamics layer. Theoretically, a well-approximated dynamics layer has the potential to augment the configuration-only neural network in a way that exactly matches the original simulations. Moreover, if the configuration-only neural network can capture enough of the non-linearities, then the dynamics layer has the potential to be quite simple (and thus real-time).

In this paper, we propose using a quasistatic physics simulation neural network (see e.g. [Bertiche et al. 2020; Geng et al. 2020; Jin et al. 2020; Luo et al. 2018]) as the configuration-only neural network. Since quasistatic neural networks (QNNs) do not have time dependency, they require far less training data and as such can use a much simpler network structure with far fewer parameters than a network that attempts to model temporal transitions. Using less training data on a network designed to capture temporal transitions leads to overfitting and poor generalization to unseen data. Using a simpler network structure with less parameters on a network designed to capture transitions leads to underfitting of the training data (and poor generalization).

Although we expect that an entire cottage industry could be developed around the modelling and real-time simulation of dynamics layers, we propose only a very simple demonstrational model here (but note that it works surprisingly well). Importantly, the zero-restlength spring approximation to the dynamics layer can be integrated analytically and thus has zero truncation error and no time step stability restrictions, making it quite fast and accurate. Furthermore (as shown in Section 7), one can (automatically) robustly learn spring constitutive parameters from a very small amount of dynamic simulation data.

2 RELATED WORK

Stated-based methods. We first discuss prior works that generate elastic deformation directly from spatial state without considering temporal or configurational history. Many works aim to upsample a low-resolution simulation to higher resolution: [Feng et al. 2010] trains a regressor to upsample. [Kavan et al. 2011] learns an upsampling operator, and [Chen et al. 2018] rasterizes the vertex positions before upsampling it and interpolating new vertex positions. [Wang et al. 2010; Xu et al. 2014; Zurdo et al. 2012] use example-based methods to synthesize fine-scale wrinkles from a database. [Patel et al. 2020] predicts a low-frequency mesh with a fully connected network and uses a mixture model to add wrinkles. [Chentanez et al. 2020] upsamples with graph convolutional neural networks. [Wu et al. 2021b] recovers high-frequency geometric details with perturbations of texture. [Bailey et al. 2020, 2018] use neural networks to drive fine scale details from a coarse character rig. Many works aim to learn equilibrium configurations from boundary conditions: [Luo et al. 2018] uses a neural network to add non-linearity to a linear elasticity model. [Mendizabal et al. 2020] learns the non-linear mapping from contact forces to displacements. Such approaches are particularly common in virtual surgery applications, e.g. [De et al. 2011; Liu et al. 2020; Pfeiffer et al. 2019; Roewer-Despres et al. 2018; Salehi and Giannacopoulos 2021]. [Jin et al. 2020] trains a CNN to infer a displacement map which adds wrinkles to skinned cloth, and [Wu et al. 2020] improves the accuracy of this approach by embedding the cloth into a volumetric tetrahedral mesh. [Bertiche et al. 2020] adds physics to the loss function, a common approach in physics-inspired neural networks (PINNs), see e.g. [Raissi et al. 2019]. To avoid the soft constraints of PINNs that only coerce physically-inspired behaviour, [Geng et al. 2020; Srinivasan et al. 2021] add quasistatic simulation as the final layer of a neural network in order to constrain output to physically attainable manifolds.

Transition-based methods. Here we discuss prior works that use a temporal history of states, typically for resolving dynamic/inertia related behaviors. In one of the earliest works (before the deep learning era) [Grzeszczuk et al. 1998] uses a neural network to learn temporal transitions and leverage back propagation to optimize control parameters. [De Aguiar et al. 2010] incorporates an approximation to the quasistatic equilibrium that serves as a control for a dynamics layer. [Guan et al. 2012] predicts a cloth mesh from body poses and previous frames, solving a linear system to fix penetrations. [Hahn et al. 2014] uses dynamic subspace simulation on an adaptive selected basis generated from the current body pose. [Holden et al. 2019] computes a linear subspace of configurations with principal component analysis (PCA) and learns subspace simulations from previous frames with a fully connected network. [Fulton et al. 2019; Tan et al. 2018, 2020] obtain nonlinear subspaces with autoencoder networks. Similar methods are commonly used to animate fluids using regression forests [Ladický et al. 2015] or recurrent neural networks (RNNs) [Wiewel et al. 2019]. [Pfaff et al. 2020] and [Sanchez-Gonzalez et al. 2020] use graph networks to learn simulations with both fixed and changing topology. [Chentanez et al. 2020] proposes a transition-based model with position and linear/angular velocity of the body as network input (in addition to a state-based model). [Meister et al. 2020] uses a fully connected network to predict node-wise acceleration for total Lagrangian explicit dynamics. [Deng et al. 2020] proposes a convolutional long
short-term memory (LSTM) layer to capture elastic force propagation. [Zhang et al. 2021] uses an image based approach to enhance detail in low resolution dynamic simulations.

**Secondary dynamics for characters.** Numerical methods that resolve the dynamic effects of inertia-driven deformation have a long history in computer graphics skin and flesh animation. We refer interested readers to only a few recent papers and a plethora of references therein (e.g. [Sheen et al. 2021; Wang et al. 2020; Zhang et al. 2020]). We note that any of these techniques could be used to generate training data for learning-based methods. Secondary dynamics for characters have also been added using data-driven methods: [Pons-Moll et al. 2015] provides a motion capture dataset with dynamic surface meshes, and proposes a linear auto-regressive model to capture dynamic displacements compressed by PCA. [Loper et al. 2015] extends this method to the SMPL human model. See also [Casas and Otaduy 2018; Santesteban et al. 2020; Seo et al. 2021].

3 QUASISTATIC NEURAL NETWORK

We use the (freely available) MetaHuman [Epic Games 2021] which (e.g. linear blend skinning [Lewis et al. 2000; Magnenat-Thalmann paper]). Both approaches worked rather well in our experiences. Section 3.1) and 4D scanning (which will be discussed in a future two different approaches: quasistatic simulation (as discussed in x as accurately as possible. We obtain ground truth for \(x^{s\text{kin}}(\theta)\) via two different approaches: quasistatic simulation (as discussed in Section 3.1) and 4D scanning (which will be discussed in a future paper). Both approaches worked rather well in our experiences.

**Proportional-derivative control.** Our analytic zero-restlength spring targeting method resembles proportional-derivative (PD) control algorithms used in both computer graphics and robotics. We refer interested readers to several papers leveraging PD control and control parameter optimization for various usages [Allen et al. 2011; De Luca et al. 2005; Hodgins et al. 1995; Wang et al. 2012; Weinstein et al. 2007].

3.1 Quasistatic Simulation

First, we use Tetgen [Si 2015] (alternatively, [Hu et al. 2018],[Shewchuk 1998] could be used) to create a volumetric tetrahedron mesh whose boundary corresponds to the Metahuman surface mesh in a reference A-pose. Next, we interpolate skinning weights from the Metahuman surface vertices to the tetrahedron mesh boundary vertices, and subsequently solve a Poisson equation on the tetrahedron mesh to propagate the skinning weights to interior vertices [Cong et al. 2015]. Then, we use a geometric approximation to a skeleton in order to specify which interior vertices of the tetrahedron mesh should follow their skinned positions with either Dirichlet boundary conditions or zero-restlength spring penalty forces.

Our training dataset includes about 5000 poses generated randomly, from motion capture data, and manually specified animations. Given any target pose, specified by a set of joint angles \(\theta\), we solve for the equilibrium configuration of the volumetric tetrahedron mesh using the method from [Teran et al. 2005] in order to avoid issues with indefiniteness and the method from [Marquez et al. 2022] to enforce contact boundary conditions on the surface of the tetrahedron mesh. Although simulation can be time-consuming, quasistatic simulation is much faster than dynamic simulation. Furthermore, the amount of simulation required is significantly smaller than that which would be needed to obtain similar efficacy for a network aiming to capture temporal information, since such a network would require far more parameters to prevent underfitting.

3.2 QNN

Instead of inferring the positions of the surface vertices directly, we augment the skinning result \(x^{s\text{kin}}(\theta)\) with per-vertex displacements \(d(\theta)\) so that the non-linearities from joint rotations \(\theta\) are mostly captured by the skinning. This reduces the demands on the neural network allowing for a smaller model and thus requiring less training data. Given ground truth displacements \(d(\theta)\), we train our quasistatic neural network (QNN) to minimize the loss between \(d(\theta)\) and the network inferred result \(d^{\text{net}}(\theta)\). We follow an approach similar to [Jin et al. 2020] rasterizing the per-vertex displacements into a displacement map image so that a convolutional neural network (CNN) can be used. Of course, one could alternatively use PCA with a fully connected network; however, GPUs are more amenable to the image-based frameworks used by CNNs (see e.g. [Wang 2021]), which discusses the benefit of using data structures that resemble
images on GPUs). Our QNN can fix skinning artifacts like interpenetration and volume loss (see Figure 2), thus providing a simpler dynamics layer for analytic zero-restlength springs to capture (see Section 7 for discussions).

4 KINEMATICS

The skeletal animation will be queried at a user-specified time scale (likely proportional to the frame rate). While these samples are inherently discrete, our approach utilizes the analytic solution of temporal ODEs: therefore, we extend these discrete samples to the continuous time domain. Specifically, given a sequence of skeletal joint angles \( \theta^1, \theta^2, \ldots \), we construct a target function of surface vertex positions \( \hat{x}(t) \) defined for all \( t \geq 0 \). Options include e.g. Heaviside (discontinuous), piecewise linear (C^0), or cubic (C^1) interpolation. We utilize cubic interpolation given its relative simplicity and favorable continuity. Between sample \( n \) at time \( t^n \) and sample \( n+1 \) at time \( t^{n+1} + \Delta t \), we define

\[
\hat{x}(t^n + \Delta t) = \hat{x}^n + \Delta t \dot{\hat{x}} + \frac{\Delta t^2}{2} \ddot{\hat{x}} + \frac{\Delta t^3}{6} \dddot{\hat{x}}
\]

where \( \epsilon \in [0, 1] \) and Equation 2 absorbs the powers of \( \Delta t \) into the non-hatted variables for simplicity of exposition. Enforcing C^1 continuity at times \( t^n \) and \( t^{n+1} \) requires the following position and derivative constraints

\[
\begin{bmatrix}
0 & 0 & 1 & q^n \\
0 & 1 & 0 & a^n \\
1 & 1 & 1 & b^n \\
2 & 3 & 2 & c^n
\end{bmatrix} = \begin{bmatrix}
\frac{1}{2} (x_{net}(\theta^n)) - x_{net}(\theta^{n-1}) \\
\frac{1}{2} (x_{net}(\theta^{n+1}) - x_{net}(\theta^n)) \\
\frac{1}{2} (x_{net}(\theta^{n+2}) - x_{net}(\theta^{n+1})) \\
\frac{1}{2} (x_{net}(\theta^{n+3}) - x_{net}(\theta^{n+2}))
\end{bmatrix},
\]

which can readily be solved to determine \( q^n, a^n, b^n, c^n \). Here, \( x_{net}(\theta^n) = x^{kin} + d_{net}(\theta^n) \) are QNN-inferred surface vertex positions at time \( t^n \). Note, in the first interval, \( \frac{1}{2} (x_{net}(\theta^{n+1}) - x_{net}(\theta^{n-1})) \) is replaced by the one-sided difference \( x_{net}(\theta^{n+1}) - x_{net}(\theta^n) \).

5 DYNAMICS

We connect a particle (with mass \( m \)) to each kinematic vertex \( \hat{x}(t^n + \Delta t) \) using a zero-restlength spring (although other analytically integratable dynamic models could be used). The position of each simulated particle obeys Hooke’s law,

\[
\ddot{x}(t) = k_s (\hat{x}(t) - x(t)) + k_d (\ddot{x}(t) - \ddot{x}(t)),
\]

where \( k_s \) and \( k_d \) are the spring stiffness and damping (both divided by the mass \( m \)) respectively. This equation can be analytically integrated (separately for each particle) to determine a closed form solution, which varies per interval because \( q^n, a^n, b^n, c^n \) vary. Consider one interval \( [t^n, t^{n+1}] \) with initial conditions

\[
x^n = x(t^n)
\]

\[
\dot{x}^n = \dot{x}(t^n)
\]

determined from the previous interval; then, the closed form solution in this interval can be written as

\[
x(t^n + \Delta t) = e^{-\frac{k_d}{2} \Delta t} g(t^n + \Delta t) + p(t^n + \Delta t)
\]

where \( s \in [0, 1] \). Here \( p(t^n + \Delta t) \) is the particular solution associated with the inhomogeneous terms arising from the targets \( \dot{x}(t^n + \Delta t) \)

\[
p(t^n + \Delta t) = \dot{x}(t^n + \Delta t) - \frac{6\eta q^n + 2x^n}{k_s \Delta t^2} + \frac{6k_d q^n}{k_s^2 \Delta t^2}
\]

The spring is overdamped when \( k_d^2 - 4k_s > 0 \), underdamped when \( k_d^2 - 4k_s < 0 \), and critically damped when \( k_d^2 - 4k_s = 0 \). Defining a (unitless) \( \epsilon \) for both the overdamped case, \( \epsilon = \frac{\Delta t}{\Delta t^2} \sqrt{k_s^2 - 4k_s} \), and the underdamped case, \( \epsilon = \frac{\Delta t}{2 \Delta t^2} \sqrt{k_d^2 - k_s^2} \), allows us to write

\[
\begin{align*}
g_o(t^n + \Delta t) &= \frac{y^n_1 \epsilon + \epsilon - \epsilon}{2} + \frac{\epsilon \Delta t}{2} \\
g_o(t^n + \Delta t) &= \frac{y^n_2 \epsilon + \epsilon - \epsilon}{2}
\end{align*}
\]

6 LEARNING THE CONSTITUTIVE PARAMETERS

Given one or more temporal sequences \( \{ \theta^1, \theta^2, \ldots, \theta^N \} \) and corresponding dynamic simulation or motion capture results \( \{ x_1^N, x_2^N, \ldots, x_N^N \} \), we automatically learn constitutive parameters \( k_s \) and \( k_d \) for each spring. For each such temporal sequence, we create a loss function of the form

\[
L = \sum_{n=1}^N \| x(t^n) - x_N^n \|_2^2
\]

where \( x(t^n) \) is determined as described in Section 5. When there is more than one temporal sequence, the loss function can simply be added together. Notably, the loss can be minimized separately for each particle in a highly parallel and efficient manner. We use gradient descent, where initial guesses are obtained from a few iterations of a genetic algorithm [Holland 1992].

The gradient of \( L \) with respect to the parameters \( k_s \) and \( k_d \) requires the gradient of \( x(t^n) \) with respect to \( k_s \) and \( k_d \), i.e. \( \frac{\partial x}{\partial k_s} \) and \( \frac{\partial x}{\partial k_d} \). From Equation 7, one can readily see that the chain rule takes the form

\[
\begin{align*}
\frac{\partial x}{\partial k_s} &= -\epsilon \Delta t^2 - \frac{k_d}{2} \Delta t s \\
\frac{\partial x}{\partial k_d} &= -\epsilon \Delta t^2 - \frac{k_s}{2} \Delta t \dot{s}
\end{align*}
\]

where \( \frac{\partial s}{\partial k_s}, \frac{\partial s}{\partial k_d} \), and \( g \) all vary based on \( \epsilon \), i.e. based on whether \( k_s \) and \( k_d \) admit overdamping, underdamping, or critically damping. As we have seen (see Equation 9, 10, 11 and the discussion thereafter), \( g \) is continuous in the 2-dimensional \( k_s,k_d \) phase space; however, one needs to carefully implement \( \frac{\partial s}{\partial k_s} \) and \( \frac{\partial s}{\partial k_d} \) to replace potentially spurious floating point divisions by the asymptotic result when \( \epsilon \)
is small. One can similarly show that \( \frac{dg}{dk_s} \) and \( \frac{dg}{dk_d} \) are continuous, and thus \( \frac{dk}{de} \) and \( \frac{dk}{de} \) are continuous.

To see that \( \frac{dg}{dk_s} \) and \( \frac{dg}{dk_d} \) are continuous, we expand them via the chain rule

\[
\frac{dg}{dk_s} = \frac{dg}{dy^n} \frac{dy^n}{dk_s} + \frac{dg}{dy^n} \frac{dy^n}{dk_s} + \left( \frac{dg}{de} \right) \left( \frac{de}{dk_s} \right)
\]

and note that \( \frac{dg}{dy^n} \) and \( \frac{dg}{dy^n} \) are continuous for the same reasons that \( g \) is. As can be seen in Equations 12 and 13:

\[
\frac{dk}{de} = \frac{dy^n}{dk_s} + \frac{dy^n}{dk_s} + \left( \frac{de}{de} \right) \frac{de}{dk_s} \quad \text{(17)}
\]

\[
\frac{dk}{de} = \frac{dy^n}{dk_s} + \frac{dy^n}{dk_s} + \left( \frac{de}{de} \right) \frac{de}{dk_s} \quad \text{(18)}
\]

and \( \frac{dk}{de} \) recursively depend on the prior interval via \( x^t \) and \( x^n \) (and eventually the initial conditions) but add no new discontinuities of their own. We inserted \( 1 \) and \( \epsilon \) into the last term in both Equations 17 and 18 so that \( \frac{dk}{de} = \mp \Delta t \) and \( \frac{dk}{de} = \pm \Delta t \) are robust to compute (the \( \mp \) and \( \pm \) signs represent overdamping/underdamping respectively). Then, we write

\[
\frac{dg}{de} = y^n \frac{e^{-e^{-k}}} {2e} + y^n \frac{\Delta t \{ (e-1)e^{-1} + (e-1)e^{-1}\}} {2e^3}
\]

(19)

\[
\frac{dg}{de} = -\left( y^n \frac{\sin e} {e} + y^n \frac{\sin e} {e} \right)
\]

(20)

to identify two more functions that must be carefully implemented (as \( e \rightarrow 0 \), \( \frac{e-1}{e} \rightarrow \frac{1}{3} \) and \( \frac{\sin e}{e} \rightarrow \frac{1}{2} \)). The sign difference between Equation 19 and 20 matches that in \( \frac{dk}{de} \) and \( \frac{dk}{de} \) showing that both \( \frac{dg}{dk_s} \) and \( \frac{dg}{dk_d} \) are continuous.

Finally, it is worth noting that a 2-dimensional gradient cannot be computed on the codimension-1 curve associated with critically damping; however, taking the dot product of the continuous (between overdamping and underdamping) gradient with the tangent to the codimension-1 curve (and adjusting for either \( k_s \) or \( k_d \) parameterization) matches the derivative along the curve as expected.

7 RESULTS AND DISCUSSION

Figure 3 quantitatively illustrates how our approach alleviates the demand on the neural network for a particular dynamic simulation example ("calisthenics"). Figure 3a shows the \( L_2 \) norm of the vertex positions (red curve) measured relative to a coordinate system whose origin is placed on the pelvis coordinate system.

\[
\text{Fig. 3. Red curve: } L_2 \text{ norm of vertex positions in the pelvis coordinate system.}
\]

\[
\text{Blue curve: } L_2 \text{ norm of displacements from skinned to dynamics. Green curve: } L_2 \text{ norm of displacements from QNN to dynamics.}
\]

\[
\text{Orange curve: } L_2 \text{ norm of displacements from QNN to zero-restlength springs.}
\]

shape, but with smaller magnitude, due to regularization. However, even with regularization, our method still outputs quite compelling dynamics (as can be seen in the supplementary video).

As mentioned in Section 6, we learn our spring constitutive parameters using a (surprisingly) small amount of ground truth simulation data. We obtain the dynamic simulation results \( \{x_i^1, x_i^2, \ldots, x_i^N\} \) via backward Euler simulation. Figure 4 shows examples of two dynamic simulation sequences ("jumping jacks" and "calisthenics") we use to learn zero-restlength spring constitutive parameters.

\[
\text{Fig. 4. Dynamic simulation sequences used to learn zero-restlength spring constitutive parameters.}
\]
that any reasonable animation sequence with dynamics can be used, even motion capture data (see e.g. [Pons-Moll et al. 2015]). Although we use a dataset with 5000 data samples in order to train a robust QNN (see Section 3), only a few dynamic simulation examples are required in order to learn zero-restlength spring constitutive parameters that generalize well to unseen animations. This also means that we only need to engineer the network architectures and hyperparameters for the configuration-only QNN, which is much easier than engineering a network that captures configuration transitions (see Section 1 for the discussion about underfitting and overfitting of transition-based methods).

### 7.1 Examples

Our analytic zero-restlength spring model generalizes very well to unseen animations and does not face severe underfitting or overfitting, which is common in machine learning methods if the network architecture is not carefully designed and trained on a plethora of data. Figure 5 qualitatively shows two example frames comparing a skinning-only result with our analytic zero-restlength springs added on top of our QNN. The frame on the left (“jumping jacks”) is taken from an animation sequence used in training while the frame on the right (“shadow boxing”) is taken from an animation sequence not used in training. In both examples, our method successfully recovers ballistic motion (e.g. in the belly). Our method runs in real-time (30–90 fps, or even faster pending optimizations) and emulates the effects of accurate, but costly, dynamic backward Euler simulation remarkably well. We refer readers to our supplementary video for a compelling demonstration, particularly of the secondary inertial motion.

Full dynamic simulation is costly and prone to instabilities. Often this results in a few simulated frames with visible errors. To avoid such artifacts, we modify our training procedure to avoid overfitting to poorly converged frames (that would lead to poor generalization). See Figure 6. We note that similar approaches are common in the computer vision community (see e.g. random sample consensus [Fischler and Bolles 1981]).

Figure 7 shows a heatmap visualization of learned $k_s$, $k_d$ and the overdamping/underdamping indicator $k_d^2 - 4k_s$, respectively. Note how symmetric our optimization result is, even if we optimize each particle. In regions where rigid motion dominates (e.g. hands, feet, head, etc.), the optimization results in overdamped springs with large stiffness. The code can be accelerated by replacing the constitutive parameters of all such springs with a single set of constitutive parameters. In regions where soft-tissue dynamics dominates (e.g. belly, thigh, etc.), the optimization results in underdamped springs with small stiffness. Since our optimization is per particle decoupled, it is easy to troubleshoot (if necessary).

As a final note, one could obviously add our zero-restlength springs on top of the skinning result directly; however, we obtained better results using our QNN to fix skinning artifacts due to volume loss and collision.

### 8 CONCLUSION AND FUTURE WORK

We present an analytically integratable physics model that can recover dynamic modes in real-time. The main takeaway is that the problem can be separated into a configuration-only quasistatic layer and a transition-dependent dynamics layer, where the dynamics layer can be well approximated by a simple physics model. The constitutive parameters of the physics model can be robustly learned from only a few backward Euler simulation examples. In particular, determining $k_s$ and $k_d$ requires a gradient that can erroneously overflow/underflow near the critical damping manifold in $k_s$-$k_d$ phase space. We quite robustly addressed this by isolating non-dimensionalized functions that were trivially carefully implemented to obtain the correct asymptotic result in all cases. For more discussions on both numerical and analytical issues with gradients, we refer the interested readers to [Johnson and Fedkiw 2022; Metz et al. 2021].

**REFERENCES**

- Brian F Allen, Michael Neff, and Petros Faloutsos. 2011. Analytic proportional-derivative control for precise and compliant motion. In 2011 IEEE International Conference on Robotics and Automation. IEEE, 6039–6044.
- Stephen W Bailey, Dalton Omens, Paul Dilorenzo, and James F O’Brien. 2020. Fast and deep facial deformations. ACM Transactions on Graphics (TOG) 39, 4 (2020), 94–1.
- Stephen W Bailey, Dave Otte, Paul Dilorenzo, and James F O’Brien. 2018. Fast and deep deformation approximations. ACM Transactions on Graphics (TOG) 37, 4 (2018), 1–12.
- Hugo Bertiche, Meysmad Madadi, and Sergio Escalera. 2020. PBNS: Physically Based Neural Simulator for Unsupervised Garment Pose Space Deformation. arXiv preprint arXiv:2012.11510 (2020).
- Dan Casas and Miguel A Otaduy. 2018. Learning nonlinear soft-tissue dynamics for interactive avatars. Proceedings of the ACM on Computer Graphics and Interactive Techniques 1, 1 (2018), 1–15.
- Lan Chen, Juntao Ye, Liping Jiang, Chengcheng Ma, Zhanglin Cheng, and Xiaopeng Zhang. 2018. Synthesizing cloth wrinkles by CNN-based geometry image super-resolution. Computer Animation and Virtual Worlds 29, 3-4 (2018), e1810.
- Nuttapong Chentanez, Miles Macklin, Matthias Müller, Stefan Jeschke, and Tae-Yong Kim. 2020. Cloth and skin deformation with a triangle mesh based convolutional neural network. In Computer Graphics Forum. Vol. 39. Wiley Online Library, 123–134.
- Matthew Conn, Michael Tao, Jane E. Kian S Rhat, and Ronald Fedkiw. 2015. Fully automatic generation of anatomical face simulation models. In Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation. 175–183.
- Suvarn Dee, Dhannanjay Deo, Ganesh Sankaranarayanan, and Venkata S Arikatla. 2011. A physics-driven neural networks-based simulation system (phynness) for multimodal interactive virtual environments involving nonlinear deformable objects. Presence 20, 4 (2011), 289–308.
- Edisson De Aguilar, Leonid Sigal, Adrien Treuille, and Jessica K Hodgins. 2010. Stable spaces for real-time clothing. ACM Transactions on Graphics (TOG) 29, 4 (2010), 1–9.
- Alessandro De Luca, Bruno Siciliano, and Loredana Zollo. 2005. PD control with on-line gravity compensation for robots with elastic joints: Theory and experiments. automatica 41, 10 (2005), 1809–1819.
Fig. 6. Robust training in the presence of simulation errors. Subfigures in columns (a)-(c) are per-axis trajectories of an example vertex in the jumping jack sequence. The backward Euler trajectory is shown in blue and our analytic zero-rest-length spring trajectory is shown in orange. The high-frequencies in Frames 31–34 are caused by poorly converged dynamics in the presence of collisions. Subfigures in column (d) show the $\ell_2$ loss between the zero-rest-length springs and backward Euler. The first row is the initial training result and the second row is the re-trained result with the 10% highest-loss frames ignored. The second row more closely follows the backward Euler trajectory for the frames that don’t have simulation errors.

Fig. 7. Heatmap visualization (logarithm scale) of stiffness $k_s$, damping $k_d$, and $k_d^2 - 4k_s$ which determines overdamping/underdamping, respectively. In heavily constrained regions the springs are stiffer and more overdamped, while in flexibly regions the springs are softer and more underdamped. Note that more constrained regions occur based on proximity to the bones used in the dynamic simulation training data (e.g. chest, forearms, shins, etc.).
