NNWarp: Neural Network-based Nonlinear Deformation

Ran Luo, *Student Member, IEEE,* Tianjia Shao, *Member, IEEE,* Huamin Wang, *Member, IEEE,* Weiwei Xu, *Member, IEEE,* Xiang Chen, Kun Zhou, *Fellow, IEEE,* and Yin Yang, *Member, IEEE*

Abstract—NNWarp is a highly re-usable and efficient neural network (NN) based nonlinear deformable simulation framework. Unlike other machine learning applications such as image recognition, where different inputs have a uniform and consistent format (e.g. an array of all the pixels in an image), the input for deformable simulation is quite variable, high-dimensional, and parametrization-unfriendly. Consequently, even though the neural network is known for its rich expressivity of nonlinear functions, directly using an NN to reconstruct the force-displacement relation for general deformable simulation is nearly impossible. NNWarp obviates this difficulty by partially restoring the force-displacement relation via warping the nodal displacement simulated using a simplistic constitutive model – the linear elasticity. In other words, NNWarp yields an incremental displacement fix per mesh node based on a simplified (therefore incorrect) simulation result other than synthesizing the unknown displacement directly. We introduce a compact yet effective feature vector including *geodesic*, *potential* and *digression* to sort training pairs of per-node linear and nonlinear displacement. NNWarp is robust under different model shapes and tessellations. With the assistance of deformation substructuring, one NN training is able to handle a wide range of 3D models of various geometries. Thanks to the linear elasticity and its constant system matrix, the underlying simulator only needs to perform one pre-factorized matrix solve at each time step, which allows NNWarp to simulate large models in real time.

Index Terms—neural network, machine learning, data-driven animation, nonlinear regression, deformable model, physics-based simulation

1 Introduction

Nonlinear shape deformation is ubiquitous in our every day life and simulating deformable objects has long been considered as an important yet challenging task for computer graphics and animation. In the past ten years, the finite element method (FEM) based frameworks [1] become more and more popular due to its versatility of encoding various material behaviors. With the prescribed external force $\mathbf{f}_{\texttt{ext}}$, the dynamic equilibrium is forwarded by solving a high-dimensional nonlinear system at each time step. Most nonlinear solvers start with an initial guess of the unknown displacement $\mathbf{u}$ and iteratively refine the result until the system converges in order to calculate the deformed model shape. While conceptually straightforward, the requirement of repetitive evaluations of the nonlinear internal force $\mathbf{f}_{\texttt{int}}$ or/and its gradient $\partial\mathbf{f}_{\texttt{int}}/\partial\mathbf{u}$ makes the simulation rather computational expensive.

Recently, the rapid development of the computing hardware pushes forward the frontier of machine intelligence to an unprecedented extend, and we have witnessed tremendous successes of utilizing carefully constructed neural networks (NNs) [2] in many classic computing problems like language processing [3], speech recognition [4], [5], object tracking [6], [7] etc. With the support of sufficient ground truth data, an NN serves as a black box mapping its input to the output without the necessity of an explicit mathematical formulation. Since the FEM simulation is able to provide us as many as needed noise-free data, can we also exploit NNs to deal with deformable simulation?

At the first sight of the question, the answer seems to be positive because deformable simulation is essentially the reconstruc-

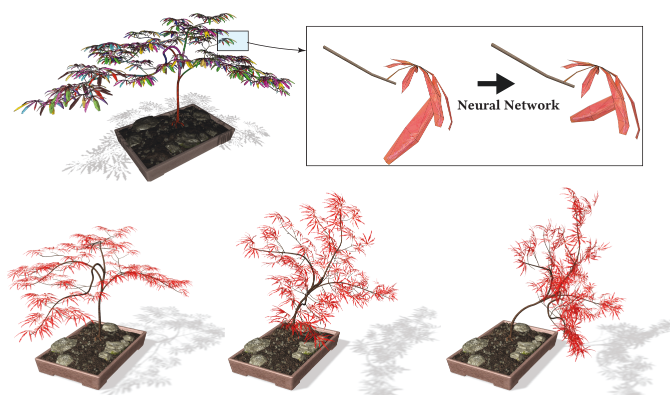


Figure 1: NNWarp is a data-driven neural network based nonlinear deformable simulator. By learning from full FEM simulation poses, it yields more accurate results than existing warping methods. We design three compact contextual features making the network training highly re-usable. In this example, the maple bonsai model consists of $255,552$ elements, and is decomposed into $1,771$ domains. A single net trained using a regular beam handles local dynamics for all the domains. High-quality animations with well-preserved local high-frequency deformations are produced at a near-interactive rate (5 FPS) without using model reduction.

- *Ran Luo and Yin Yang are with Electrical and Computer Engineering Department, University of New Mexico, NM, 87131*
  E-mail: {luoran|yangy}@unm.edu
- *Tianjia Shao is with School of Computing, University of Leeds, UK, LS29JT.*
  E-mail: tianjiashao@gmail.com
- *Huamin Wang is with Department of Computer Science and Engineering, Ohio State University, OH, 43210.*
  E-mail: whmin@cse.ohio-state.edu
- *Weiwei Xu, Xiang Chen and Kun Zhou are with State Key Lab of CAD&CG at Zhejiang University, China.*
  E-mail: weiwei.xu.g@gmail.com;xchen.cs@gmail.com;kunzhou@acm.org

tion of the *force-displacement relation* of an elastic body, and NNs are known good at expressing complex nonlinear relations [8], [9]. However, this problem is challenging in practice because the nonlinear force-displacement relation varies significantly (and intrinsically) under different simulation configurations such as domain geometries, discretizations, boundary conditions, constitutive laws etc. If one chooses to build a network incorporating all the possible input permutations, the network would indubitably be an extremely huge one. Consequently, the corresponding training data would be very big and a single forward pass of the network itself could take a longer time than running a regular FEM simulator due to the complexity of the network.

In this paper, we present a method, named *NNWarp*, to leverage neural networks to tackle intricate force-displacement relations of different nonlinear materials with a simple and light-weight network. As the name implies, our strategy is not to link the standard input ($\mathbf{f}_{\texttt{ext}}$) and output ($\mathbf{u}$) of deformable simulation via a neural network directly. Instead, we map or *warp* a simplified constitutive law $\mathcal{L}_0$ to a more complex and nonlinear one $\mathcal{L}_1$ using NNs. It is expected that, the calculated displacement under $\mathcal{L}_0$ well encapsulates simulation configurations like force magnitude, domain tessellations and boundary conditions so that the remaining warp is local, and can be well fit by a simple net. To this end, we choose to use the linear elasticity for $\mathcal{L}_0$. The linear elasticity has long been used to describe small-scale deformations (i.e. the infinitesimal strain theory). It is based on the Cauchy strain tensor, which is the first-order Taylor approximation of the full Green tensor. Besides, because linear elasticity has a constant stiffness matrix, setting it as $\mathcal{L}_0$ makes NNWarp *polynomial* faster than another other nonlinear constitutive models with the same number of simulation DOFs during the run-time simulation. It is noteworthy that using other existing nonlinear solvers like [10], [11] as $\mathcal{L}_0$ is also legit. However, as those existing nonlinear solvers already yield good results, the effectiveness of using a complex deep network is under utilized.

NNWarp uses a single node-wise NN to correct the nodal linear deformation to the corresponding nonlinear one. In other words, it takes the linear nodal displacement as the input, and outputs a corrective displacement fix to warp the linear result to be a nonlinear one. From this perspective, our method is conceptually similar to stiffness warping [12] and modal warping [13], in which a linear solver is used after rotating the deformed shape back to its undeformed orientation. We augment the input of per-node linear displacement with three novel discriminative features, namely the *geodesic*, *potential* and *digression*. We find that with these three descriptors, NNWarp becomes fairly shape- and tessellation-independent, and the network trained with a simple model can be used to warp deformable bodies of distinctively different geometries making our network training highly re-usable. This important advantage is further enhanced when combined with the substructuring method [14], where we decompose the input model into multiple convex domains, and run NNWarp on each domain separately. For instance, all the experiments reported in the paper (except Fig. 11) are based on the network trained using a simple rectangular beam. NNWarp is fast at both training stage and simulation stage. We utilize the rotation invariant property of local deformation and compress the training set by at least an order. During the simulation, as NNWarp only needs to perform a pre-factorized linear solve at each time step, it is able to handle large-scale models interactively.

2 Related Work

The concept of neural network based learning can be dated back to late 1980s [15] in the machine learning community. Empowered by recent hardware advance, neural networks of various architectures and deeper depths have been harnessed to solve many long-standing computer vision problems such as recognition [16], [17], classification [18]–[21] and segmentation [22]–[24]. Some existing methods are able to match or even beat human's vision perception system e.g. see the report from the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [25]. Given sufficient training data, Deep neural networks (DNNs) provide a general "template" for the user to learn the input-output correspondence, which could be otherwise difficult or even impossible to be analytically formulated.

Learning for animation Indeed, the idea of learning is not new to computer animation, and it is also widely-known as *data driven* methods [26]. For the **cloth animation**, low-resolution simulation can be enriched by using pre-computed high-resolution results with detailed wrinkles [27], [28]. Wang et al. [29] built a piecewise linear stretching and bending model based on measured data to better depict the nonlinear dynamics of different cloth materials. Miguel et al. [30] further enhanced this framework and recorded more deformation behaviors of the cloth simulation. Kim et al. [31] proposed a method to compress a large pre-simulation dataset so that these poses can be used at run time to improve the inertial cloth deformation. Following the similar idea, Xu et al. [32] blended pre-computed cloth shapes to directly synthesize the cloth deformation using the sensitivity analysis. Learning-based methods have also been popular for **motion and control** i.e. the reinforcement learning [33]–[36]. NNs provide a convenient approach for further improving the learning effects [37]. Following this direction, Liu et al. [38] employed the deep Q-network to reorder existing control fragments and created necessary responses to unseen disturbances. Peng et al. [39] used an NN to train a high-level controller and a low-level one, which achieved robust locomotion coordinately. Holden et al. [40] designed a phase-functioned neural network, whose weights are computed using a cyclic function. For **solid modeling**, learning is also a powerful tool, which allows the user to obtain actual physical parameters based on captured point cloud sequences [41]. Xu and Barbič [42] fine-tuned the damping model based on a few example deformations. Kim et al. [43] combined the physics-based simulation and data-driven to produce realistic soft tissue animation. Jones et al. [44] used the similar idea to simulate plastic deformation with a skinning-alike method. An et al. [45] proposed a learning-based numerical procedure named Cubature to efficiently evaluate the internal force and the force gradient during reduced deformable simulation. Deep learning also benefits the **fluid animation**. For instance, Ladicky et al. [46] proposed a random forest based regression method to accelerate fluid simulation by predicting the kinematic configurations of particles based on a large training set. Chu and Thuerey [47] used the convolutional neural networks (CNN) to extract necessary features to augment a coarse simulation and add back high-frequency details.

Nonlinear deformable simulation Physics-based deformable simulation has been an active research topic in graphics and animation since the exemplar work by Terzopoulos et al. [48]. While particle-based methods [49]–[51] or mass-spring systems [52], [53] are also legit, FEM becomes more widely-used [54] for solid simulation. Wang et al. [55] proposed a strain limiting method to increase the numerical stability for stiff deformable bodies. Alternatively, Irving et al. [56] tweaked the principle stress to resolve degenerated elements from extreme deformations. Forming the deformable simulation as a nonlinear optimization procedure, Hecht et al. [57] used an incremental Cholesky factorization scheme to lower the frequency of matrix re-factorization during the simulation. Zhu et al. [58] adopted the multi-grid method to simulate high-resolution deformable volumes. Bouaziz et al. [10] introduced a robust local-global iterative solver named *projective dynamics*. This idea later was generalized as the ADMM solver [59] and synergized with Chebyshev [11], [60], L-BFGS [61] and GPU Gauss-Seidel [62]. Accelerating nonlinear simulation can also be achieved by pre-computed models, for instance using modal analysis [63]–[65] or recent fullspace simulations [66]. Also known as model reduction methods, it is assumed that the deformed shape be a linear combination of those pre-computed poses or *modes* so that the simulation can be projected into the spanned subspace. In an asymptotic sense however, model reduction is not better than regular simulation as the time complexity remains cubic w.r.t. the number of simulation DOFs.

NNWarp and existing warping methods In this paper, we re-investigate this classic animation problem of nonlinear deformable simulation from a data-driven point of view by shaping it as a nonlinear regression using the neural network. Unfortunately, the full spectrum of the force-displacement relation is complex and sensitive to the variance of simulation settings. For instance, modifying the boundary condition (the anchor nodes of an FE mesh) could completely alter the deformed shape even with other simulation parameters unchanged. Besides, the dynamic simulation is essentially 4D – the kinematic status of the deformable body does not only depend on its current external stimuli but also on its historic motion trajectory. To circumvent these two practical obstacles, we forge our regression based on the simulation result obtained using the linear elasticity. This idea is not new in graphics. An epic example would be the stiffness warping [12], which re-used the linear stiffness matrix by un-rotating the external force back to the model's rest shape orientation. Similarly, modal warping [13] and rotation-strain coordinate [67], [68] embedded a local coordinate frame at each node/element to relieve the artifacts of the linear elasticity under rotational deformation. This idea was also used for geometrically constructing nonlinear modes [69]. These geometric warping techniques have been proven effective for animation editing [70], [71], which requires performing high-dimensional space-time optimization.

Solving the linear elasticity encodes many simulation parameters such as boundary condition, tessellation resolution, external force etc. into the resulting linear displacement vector. On the top of this, we train a neural network to further correct the result to be a plausible and nonlinear one without worrying about accommodating all the simulation settings into the net. The trained network using a regular model of few thousand elements generalizes to a wide range of geometrically complex deformable bodies. During the simulation, because the system matrix for the linear elasticity is constant and pre-factorized, we obtain $\mathbf{O}(N^2)$ run-time complexity *in fullspace*, which is polynomially faster than existing nonlinear solvers.

3 Contextual Feature Vector

The underlying mathematical relations between external forces and displacements of elastic bodies could be intrinsically changed under different simulation settings, and it is impossible in practice to encode the entire simulation configuration into a feature vector and feed it to a neural network. Therefore, the primary challenge we are facing is to figure out an *informative* and *compact* feature vector as the input. Informative refers to the discriminability of the feature so that an irrelevant training instance does not interfere. Compact means the feature should also be general so that the built network is small and light-weight. In this section, we start with a short review of the deformable model, pointing out that while the simulation is sophisticated, the linear-nonlinear deformation map of a small local volume is actually smooth. Bearing that in mind, we show that our *heuristic* feature vector augments the extracted kinematic information and produces plausible results.

3.1 Deformable model: a quick review

Given an arbitrary material point x on the deformable body, its deformation gradient $\mathbf{F} \in \mathbb{R}^{3\times 3}$ is computed as $\mathbf{F} = \partial \mathbf{x}/\partial \bar{\mathbf{x}}$, where $\bar{\mathbf{x}}$ and $\mathbf{x}$ denote its rest shape position and the deformed position. Alternatively, we can also express $\mathbf{x}$ using its displacement $\mathbf{u}$ as $\mathbf{x} = \bar{\mathbf{x}} + \mathbf{u}$. Let $\mathbf{G} = \partial \mathbf{u}/\partial \bar{\mathbf{x}}$ and we name this 3 by 3 tensor as *displacement gradient tensor*. It is easy to verify that $\mathbf{F} = \mathbf{G} + \mathbf{I}$. Under the linear elasticity, the deformation is described using the Cauchy strain: $\widetilde{\varepsilon} = \frac{1}{2}(\mathbf{G} + \mathbf{G}^\top)$, and the strain energy density $\widetilde{\Psi}$ is:

$$\widetilde{\Psi} = \mu \widetilde{\varepsilon} : \widetilde{\varepsilon} + \frac{\lambda}{2}\mathtt{tr}^2(\widetilde{\varepsilon}). \tag{1}$$

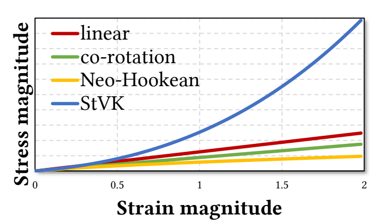
Here μ and λ are the Lamé coefficients. As $\widetilde{\Psi}$ is a quadratic function of $\mathbf{G}$, the corresponding Piola stress becomes a linear function of $\mathbf{G}$:

$$\widetilde{\mathbf{P}} = \mu\left(\mathbf{G} + \mathbf{G}^\top\right) + \lambda \cdot \mathtt{tr}(\mathbf{G})\mathbf{I}. \tag{2}$$

For most other hyperelastic materials, the deformation is actually described with the Green strain: $\varepsilon = \frac{1}{2}(\mathbf{F}\mathbf{F}^\top - \mathbf{I}) = \widetilde{\varepsilon} + \frac{1}{2}\mathbf{G}\mathbf{G}^\top$. One can see that the Cauchy strain used in linear elasticity is simply the linear portion of the Green strain. Take the St. Venant-Kirchhoff (StVK) material as example, whose strain energy density is formulated by replacing $\widetilde{\varepsilon}$ by ε:

$$\Psi_{\mathtt{StVK}} = \mu \varepsilon : \varepsilon + \frac{\lambda}{2}\mathtt{tr}^2(\varepsilon), \tag{3}$$

which is a fourth-order polynomial of the displacement gradient $\mathbf{G}$, and its stress is cubically related to $\mathbf{G}$. With the help of FEM, the differential strain-stress relation is integrated and becomes the macroscopic force-displacement relation that we are interested in.



In reality, the *magnitude* of a deformation, which may be somewhat understood as $|\mathbf{G}|$, is typically small. For instance, doubling the length of an elastic rope by stretching is considered a very large deformation where $|\mathbf{G}| = 1$. In addition, strain-stress curves for various materials are all aligned at the origin (a zero strain yields a zero stress) and within the same monotonically increasing interval (a larger strain yields a larger stress). This implies that the strain-stress curves of the linear elasticity and a nonlinear elasticity do not fundamentally differ from each other in regular deformable simulations. An example is given in the inset figure, where we plot the strain-stress curves of the linear, co-rotation, StVK and Neo-Hookean laws under a rotation-free linear stretch.

Geometric warp In fact, the dominant factor that drives the linear elasticity away from a nonlinear counterpart is not the material nonlinearity, but the geometry nonlinearity. This is because a rigid, deformation-free rotation leads to a non-zero Cauchy strain, which produces unrealistic deformation effects. Under this consideration, the modal warping (MW) technique [13] embeds each node on the mesh local frame. The curl of local displacement field around the i-th node is calculated: $\mathbf{w}_i = \nabla \times \mathbf{u}_i$. If it takes a unit time to displace node i from $\bar{\mathbf{x}}_i$ to $\bar{\mathbf{x}}_i + \mathbf{u}_i$, $\mathbf{u}_i$ also represents its velocity at $t = 1$. $\mathbf{w}_i$ can then be understood as its angular velocity at the same moment. Based on this assumption, MW linearly maps the angular velocity from the rest shape to the current time instance t and calculates a warp transformation as:

$$\mathbf{W}_{\mathtt{MW}} = \frac{1}{t}\int_0^t \mathtt{exp}\left(\frac{\tau}{t}[\mathbf{w}_i]_\times\right)\mathrm{d}\tau, \tag{4}$$

where $[\mathbf{w}_i]_\times$ is the skew symmetric matrix of $\mathbf{w}_i$. Similarly, one can use rotation-strain coordinate by decomposing the $\mathbf{G}_i$ into a skew symmetric part: $[\mathbf{w}_i]_\times = (\mathbf{G}_i - \mathbf{G}_i^\top)/2$ and a symmetric part: $\mathbf{S}_i = (\mathbf{G}_i + \mathbf{G}_i^\top)/2$ [67], [68]. The rotation-strain warp (RSW) transformation can then be computed treating $\mathbf{w}_i$ as an Euler vector:

$$\mathbf{W}_{\mathtt{RSW}} = \mathtt{exp}\left([\mathbf{w}_i]_\times\right)(\mathbf{S}_i + \mathbf{I}) - \mathbf{I}. \tag{5}$$

After a nodal linear displacement $\widetilde{\mathbf{u}}$ is obtained, it is corrected via $\mathbf{u} \leftarrow \mathbf{W}_{\mathtt{MW}}\widetilde{\mathbf{u}}$ or $\mathbf{u} \leftarrow \mathbf{W}_{\mathtt{RSW}}\widetilde{\mathbf{u}}$. For rotation-strain warping, a global Poisson reconstruction is also solved to calculate the final warped displacement. While not physically accurate, these geometric warping methods produce visually pleasing shapes and have been used in many time-critical graphics applications [70], [71].

3.2 Linear-nonlinear correspondence

NNWarp is inspired by the encouraging results from the existing warping methods. However, NNWarp does not explicitly assume a fixed nonlinear regression formula as Eqs. (4) or (5). Instead, we train an NN to obtain a more accurate regression based on full simulations. The key question here is how to determine what is the "right" nonlinear deformation that corresponds to the one calculated using the linear elasticity.

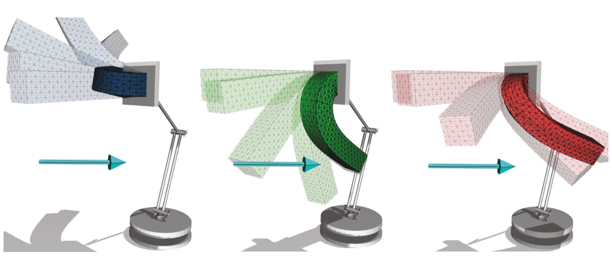


Figure 2: Different motion trajectories lead to different equilibrium shapes even under the same external force.

A naïve thought is to solve the quasi-static equilibrium of $\mathbf{f}_{\mathtt{int}}(\mathbf{u}) = \mathbf{f}_{\mathtt{ext}}$ for a deformable body under the same external force and boundary condition using the linear elasticity and a nonlinear constitutive model. Unfortunately, this method is only valid for small deformations. Under large external forces, depending on the context, i.e. the history of the deformation trajectory, a different minima can be reached (Fig. 2). From a numerical point of view, the solution of $\mathbf{f}_{\mathtt{int}}(\mathbf{u}) = \mathbf{f}_{\mathtt{ext}}$ depends on the initial guess of $\mathbf{u}$ and the strategy of computing $\Delta\mathbf{u}$ during the iteration. The iteration may not converge to the global minimum if the starting guess is far away from it.

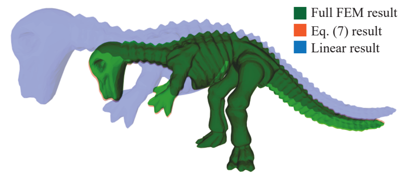


Figure 3: In practice, Eq. (7) yields good approximations of full nonlinear FEM simulation.

Our solution to this problem is to register a linear deformation *sequence* to a nonlinear one starting from the rest shape. Specifically, given an external force $\mathbf{f}_{\mathtt{ext}}$, we compute a series of quasi-static linear deformation by solving the Euler-Lagrange equation with an increased mass damping so that the acceleration at each time step is negligible. Each time step yields a linear displacement vector $\widetilde{\mathbf{u}}$, and we estimate a local rotation for the i-th node as:

$$\mathbf{R}_i = \mathtt{exp}\Big(\Big[\nabla \times \big(\mathbf{P}_i\mathbf{P}_i^\top\big)^{-1}\mathbf{P}_i^\top\mathbf{U}_i\Big]_\times\Big), \tag{6}$$

where columns in $\mathbf{P}_i$ and $\mathbf{U}_i$ are rest shape positions and displacements of neighbor nodes adjacent to i so that $\big(\mathbf{P}_i\mathbf{P}_i^\top\big)^{-1}\mathbf{P}_i^\top\mathbf{U}_i$ gives a least-square evaluation of $\mathbf{G}$ around the i-th node. The linear internal force at the current time step is $\widetilde{\mathbf{f}}_{\mathtt{int}} = \mathbf{K}\widetilde{\mathbf{u}}$. Note that $\widetilde{\mathbf{f}}_{\mathtt{int}} \neq \mathbf{f}_{\mathtt{ext}}$ until the final equilibrium is reached due to the existence of the damping. Afterwards, the corresponding nonlinear deformation $\mathbf{u}$ is obtained by solving:

$$\min_{\mathbf{u}} |\mathbf{f}_{\mathtt{int}}(\mathbf{u}) - \mathcal{R}\mathbf{K}\widetilde{\mathbf{u}}|, \tag{7}$$

where $\mathcal{R}$ is a block-diagonal matrix, and each of its 3 by 3 diagonal block is the estimated nodal rotation computed via Eq. (6). Eq. (7) is clearly an approximate because in practice when NNWarp is being used, we do not know what are the "ground truth" acceleration (which yields the inertia force) and the velocity (which yields the damping force) corresponding the linear pose. Therefore, our best guess is to solve the nonlinear equilibrium of Eq. (7) according to its linear counterpart (i.e. see Fig. 3). We use Newton's method to solve Eq. (7) by setting the initial guess of $\mathbf{u}$ as the solution in the previous time step. In our implementation, we notice that Newton's method occasionally fails during the iteration. Therefore, we impose the Wolfe condition [72] to adjust the step length.

In our network training, we simplify the external force setting by only considering two types of $\mathbf{f}_{\mathtt{ext}}$: directional force field and circular force field. The directional field uses a prescribed force direction, while the force direction in the circular field follows the tangent direction of a set of concentric circles. Such simplification frees us from generating an overwhelmingly large training set due to diverse external force conditions. Its limitation is also obvious: NNWarp loses some local deformation effects induced by high-frequency external forces. Extra training efforts would be needed if one wants to incorporate such deformation effects via NNWarp.

3.3 Discriminative feature

With paired $\langle\widetilde{\mathbf{u}}, \mathbf{u}\rangle$, we can build a node-wise regression machine using a neural network that replaces Eq. (4) or Eq. (5). For the i-th node, in addition to its linear displacement $\widetilde{\mathbf{u}}_i$, the rotation information of its local displacement gradient $\mathbf{G}_i$ is directly pertinent to the warp transformation, and should be passed to the network as the input. To this end, we choose to use the skew symmetric part of

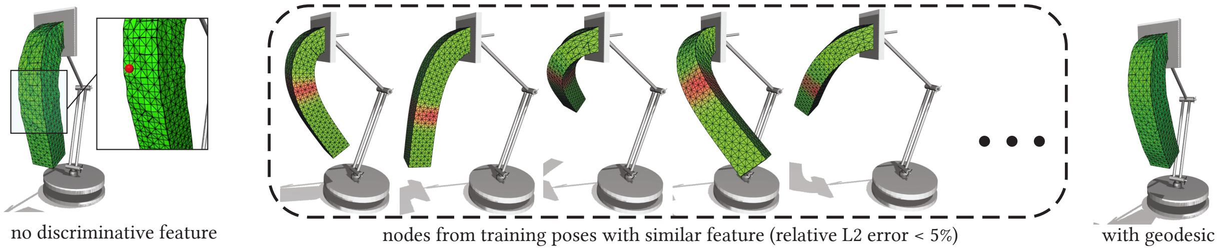


Figure 4: Only using kinematic feature as the input of the network yields noticeable jittery artifacts. A node, because of its kinematic feature is not discriminative, could be influenced by many irrelevant instances in the training data. Large discrepancies among these mismatched nodes induce high-frequency variations of the NNWarp. Incorporating geodesic feature effectively eliminates this artifact.

$\mathbf{G}_i$ and represent it as a 3-vector as in [67]. However, only feeding these two pieces of information to the network is not enough, and the resulting deformation appears jittery and non-smooth as shown in Fig. 4. In this example, we use the Neo-Hookean elasticity, whose strain energy is:

$$\Psi_{\mathrm{NH}} = \frac{\mu}{2}[I_1 - \log(I_3) - 3] + \frac{\lambda}{8}\log^2(I_3). \tag{8}$$

I_1 and I_2 are the invariants of the deformation gradient, defined based on $\mathbf{F}$'s singular values σ_1, σ_2 and σ_3 such that: $I_1 = \sigma_1^2 + \sigma_2^2 + \sigma_3^2$ and $I_3 = \sigma_1^2\sigma_2^2\sigma_3^2$.

This artifact was also noticed and discussed in previous data-driven simulation literature [46], which is because pure node-wise kinematic features do not contain sufficient contextual information, and thus are not discriminative to reach a conclusive per-node linear-nonlinear map. To further illustrate this artifact, we pick a jittery node (marked as a red sphere in the figure) and inversely query for nodes in our training set that have similar features ($< 5\%$ relative L2 error w.r.t. the feature vector from the picked node). We can see from the figure that there are a number of training poses having multiple nodes (on the red-shaded areas) with very similar feature vectors as the input. In other words, the final displacement of the picked node becomes a certain mixture of displacements of many distant and irrelevant nodes. Such ambiguity of pure kinematic feature is the primary reason behind this artifact.

One of our contribution is to design a compact contextual feature to resolve this mismath. While one could follow the method used in [46] to use the integral features of local dynamic parameters around a node, we found that our simple strategy yields satisfying result. We speculate that this is because DOFs in solid simulation are more tightly coupled than in fluid simulation [46]. An important advantage of such compactness is that the network training is also quite fast. Compared with state-of-the-art pre-computed deformable models i.e. [45], we can finish training in a few minutes, and the obtained network can be applied to a wide range of models. Concretely, we will design geodesic, potential and digression features to provide necessary contextual information to avoid ambiguity during the network training. As to be explained shortly, geodesic feature describes "how far" a mesh vertex is from the boundary condition so vertices are not mismatched because they undertake larger external force. Similarly, potential characterizes the distance between vertex and force direction to differentiate the training pairs when the geometric symmetry exists. Digression further enhances the local contextual information so that the network training could be applied to models of different shapes.

Discriminative feature I: geodesic The geodesic of a node i, g_i is the normalized length of the shortest path from node i to its nearest anchor node within the deformable body. For a training model, we first uniformly scale it to fit a unit bounding sphere. Then, we compute the shortest path using the Dijkstra's algorithm for all the un-anchored nodes. Lastly, calculated path lengths are scaled by the maximum geodesic so that all the g values are within the normalized interval of $[0,1]$. It should be noted that if one uniformly scales a mesh by a factor of S, the stiffness $\mathbf{K}$ is scaled by S only, but the mass matrix $\mathbf{M}$ is scaled by S^3. Due to this reason, the final deformation and vibrational frequency of the trajectory after scaling are altered. However, NNWarp treats deformation at each time step a quasi-static pose, and its output only depends on kinematic and contextual features. Therefore, scaling scheme is a viable training pre-processing as in other NN based systems.

Our heuristic of choosing the geodesic feature is based on the observation that if a node is closer to an anchor node, it tends to have less deformation than nodes that are away from it. By inducing the geodesic feature, a node far from anchor nodes does not miss-pair to a node close to anchor nodes only because it undertakes a smaller external force. Consequently, the jittery artifact is effectively removed as shown in the rightmost snapshot in Fig. 4.

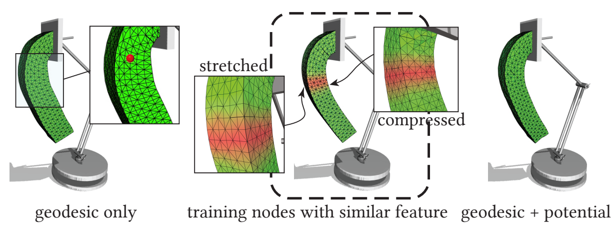
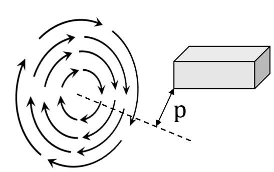


Figure 5: Volume expansion artifact remains even with the geodesic feature added. This is because the nodes with similar geodesic value may have different internal tractions. We use the potential feature to sort the training data to avoid this issue.



Discriminative feature II: potential Including the geodesic feature however, does not avoid the artifact of volume increase and shrinkage. As shown in Fig. 5, bending the beam also increases its volume noticeably, especially at curved areas. In order to dig out the missing contextual information behind this issue, we use the similar approach by picking a node within the problematic area and query the instances from our training set that have a similar feature of the selected one. We can see from the figure that, thanks to the incorporated geodesic feature, now this selected node only pairs with a training pose under a very similar deformation. However, it still matches multiple nodes on this pose.

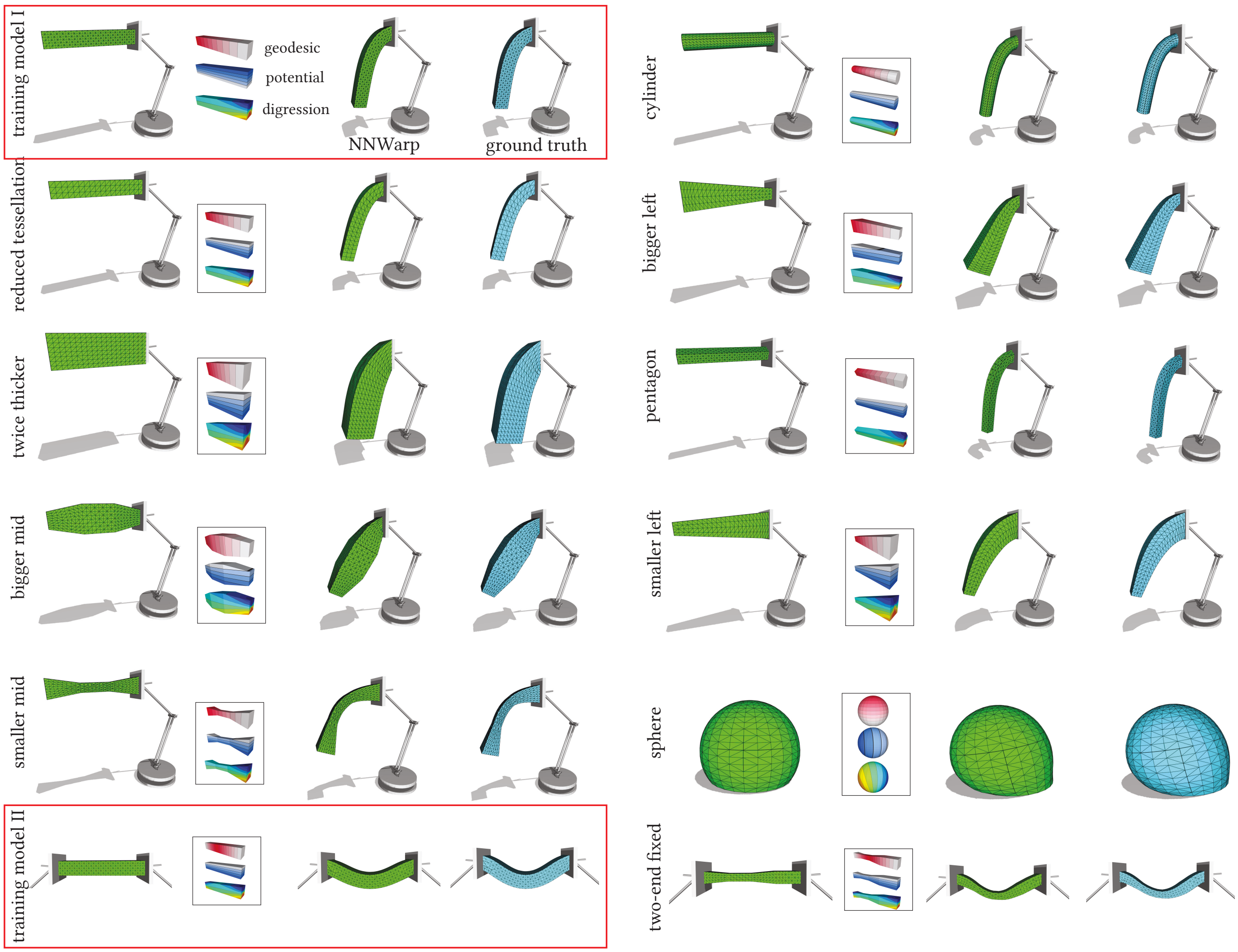


Figure 6: We test the generality of the designed features on a variety of shapes. The training data are generated using the standard rectangular beam (highlighted by a red box). The resulting DNN successfully handles many beam-like models but with distinctively different shapes. The distributions of three features are also plotted.

This is because the beam is a symmetric shape, and a loop of nodes on its surface have similar geodesic values – among which, some are stretched and some are compressed. Without being able to distinguish these contexts, the volume of the warped model is likely to shrink or expand unnaturally.

We notice that whether nodes are being stretched or compressed typically depends on their relative positions in the applied force field. Therefore, we introduce another scaler feature named potential p to resolve this ambiguity. If a directional force field is applied, for each node on the mesh, we project its rest shape position onto the force direction and re-map the resulting projections to the interval of $[0,1]$. On the other hand, if a circular force field is applied, the potential of a node is the distance between its rest shape position and the circular axis, as shown in the inset figure. This value is also scaled to $[0,1]$. As we can see from Fig. 5, the deformation of the beam model is almost identical to the one obtained using the full simulation after we inject the potential feature into the network.

Discriminative feature III: digression So far, we generate a set of registered linear and nonlinear poses of the beam model.

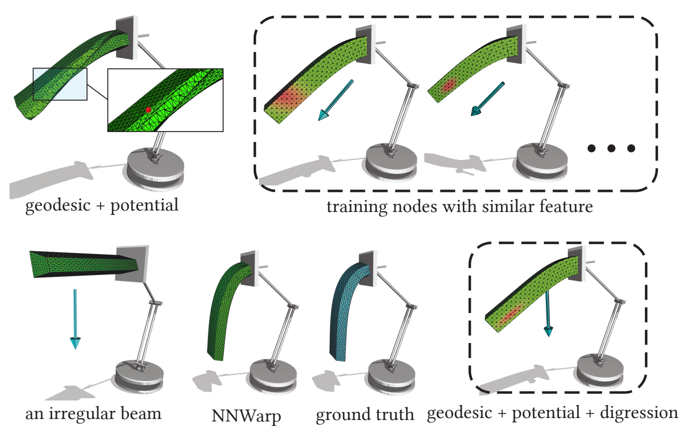


Figure 7: In order to make NNWarp re-usable for various deformable bodies, we use the digression as our third discriminative feature. With this feature included, the neural network is able to handle an irregular beam based on the training set generated using a standard rectangular beam model.

Node-wise linear to nonlinear deviation is learnt by a neural network, which is then used to warp a linear displacement of the same model to obtain its nonlinear shape. While the results are visually plausible, real-world applications will require deformable animations of various 3D models. NNWarp becomes cumbersome and less practical if one needs to re-train a network for each different deformable body.

Unfortunately, if we alter the rest shape geometry of the beam model as shown in Fig. 7, unrealistic jittery deformations are observed again even after incorporating geodesic and potential features. By querying the training set, we see that because the updated shape is irregular and asymmetric, the most similar training poses become the ones under oblique force fields regardless an upright gravity force field is applied in the simulation. To further correct this mismatch, we use the digression feature d to describe the nodal position w.r.t. the direction of the external force. Specifically, the digression for node i is defined as:

$$d_i = \arccos\left(\frac{\bar{\mathbf{x}}_i - \bar{\mathbf{x}}_a}{|\bar{\mathbf{x}}_i - \bar{\mathbf{x}}_a|} \cdot \frac{\mathbf{f}_{\mathtt{ext}}}{|\mathbf{f}_{\mathtt{ext}}|}\right), \tag{9}$$

where $\bar{\mathbf{x}}_a$ is the rest shape position of the anchor node that is closest to node i. Indeed, digression sorts nodes based on their local orientational deviations from the external force direction. The digression feature ranges from 0 to π. If a circular force field is applied, the digression is simply set as -1. As shown in Fig. 7, with geodesic, potential and digression included, the training data generated using a rectangular beam model can also be used to warp the irregular beam, and NNWarp produces high-quality nonlinear deformation.

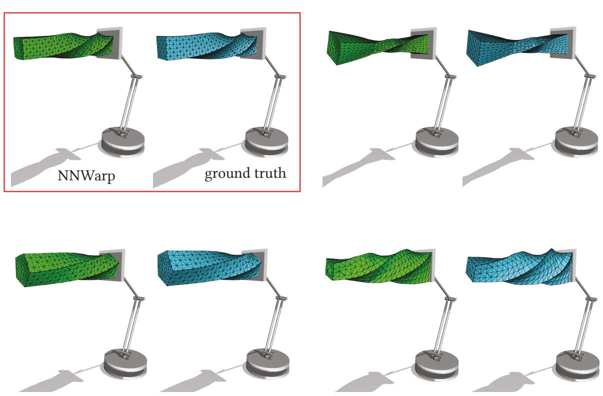


Figure 8: NNWarp works well under circular force field too. In this case, the digression feature is set as -1 for all the nodes on the mesh.

Discussion Our features allow the resulting network properly handles models with various shapes and different tessellations. More results can be found in Figs 6 and 8. From these examples, readers may probably notice that geodesic, potential and digression features actually provide a volumetric *parametrization* of deformable bodies so that models of different geometries and tessellations are somehow registered in a meaningful way, and node-wise neural network can then be applied. In fact, there are many elegant algorithms in graphics and computational geometry that generate the volumetric map between different shapes [73]–[75]. However in the context of NNWarp, this volumetric map depends on the configuration of external force and boundary conditions. While existing methods may also be modified to incorporate these additional conditions or constraints, we found that our simple strategy suffices in most cases. An exception is reported in Fig. 9, and we find that NNWarp using a convex training model often fails when the deformable body gets more concave. Next, we will show how to walk around this limitation without re-training the network for a different target shape.

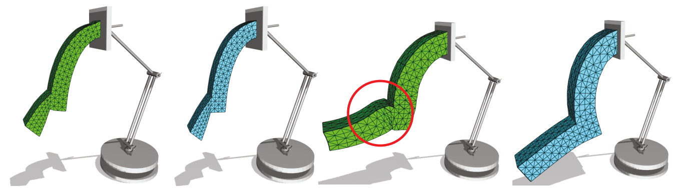


Figure 9: When the shape becomes more concave, the network trained using a rectangular beam produces artifacts.

4 Incorporate Complex Shapes

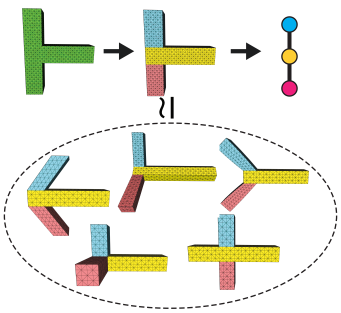


Figure 10: Building domain graph for the domain decomposed model is an easy and effective way to identify shapes with similar concavity.

The exhaustiveness of 3D geometric diversity is endless. Obviously, training set generated with a single rectangular beam cannot cover all the different feature combinations. We find that the network trained using the beam model is able to deal with many convex 3D shapes (i.e. see Fig. 6). However, it often fails when the target deformable body becomes more concave (Fig. 9). A straightforward idea is to train a new network using a model with similar concavity of the target deformable body, but *how to describe the similarity of concavity among 3D shapes?*

We borrow the idea from the graph theory [76], and subdivide a concave model into several convex components or domains. Afterwards, we create a *domain graph* $\mathcal{G}$ by using a graph vertex to represent each of the subdivided domains. An edge connects two vertices if and only if the corresponding two domains are face-connected on the original mesh. We find that if the domain graph $\mathcal{G}$ is isomorphic to the domain graph of the training model $\mathcal{G}_{\mathtt{train}}$ or $\mathcal{G} \simeq \mathcal{G}_{\mathtt{train}}$, NNWarp typically yields satisfying results. An example can be found in Fig. 10. The T-shape beam is decomposed into three domains, each of which is convex and rectangular. Its domain graph is isomorphic to many similar concave shapes like the Y-shape beam, the arrow-shape beam, the crossing beam etc. If we use the T-shape beam as the training model, the resulting neural network is able to handle all of these variations as verified in Fig. 11.

Even utilizing the concept of graph isomorphism, one may still have to re-train the network (and re-generate training poses) for an arbitrary geometrically complex model, which is tedious and time consuming. A more general and powerful solution maximizing the re-usability of the network training is to use the substructuring method [14]. This method wisely leverages the hierarchical propagation of the deformation over a complicated structure and isolates the deformable simulation at each individual domain sequentially. While it loses some physics accuracy (mostly, the frequency of the trajectory due to the mass lumping, which can also be fixed as in [77]), the resulting deformation is natural and realistic. After the domain decomposition is complete so that each domain is a convex

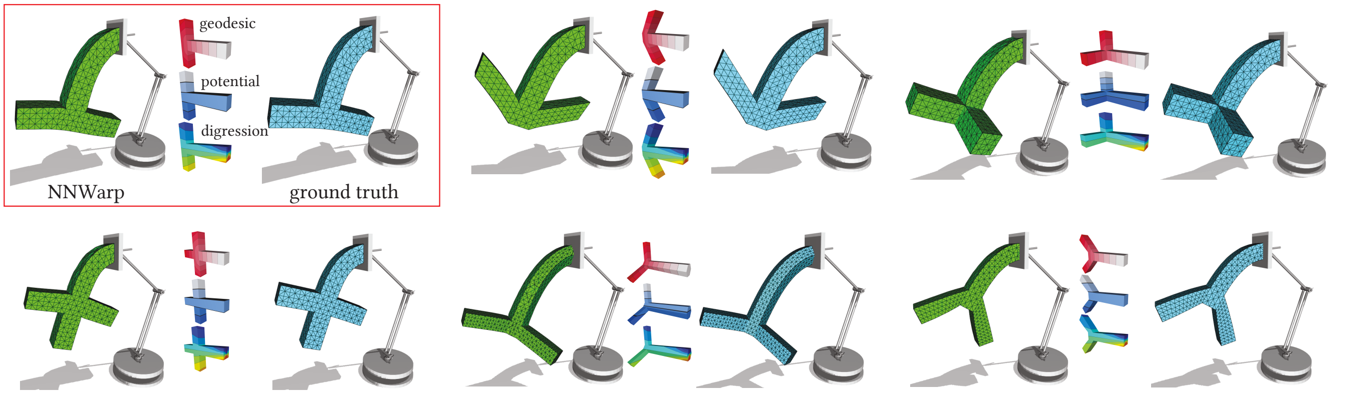


Figure 11: While a simple rectangular beam is not able to handle highly concave shapes, by referring to the domain graph we can train a network using a T-shape beam and the resulting network can be used to warp a wide range of concave beams whose domain graphs are isomorphic to the T-shape beam.

3D shape, we can use one representative convex model to train the network. With the help of the proposed three discriminative features, the resulting network is able to correct local dynamics of all the domains. It should also be noticed that having a loop in the domain hierarchy could be problematic. Using the penalty method (adding an implicit spring force to close the loop) is a commonly used method [14]. Doing so however, requires more training efforts for poses for to the penalty forces.

In our NNWarp version of substructuring, the dynamics of the domain $\mathcal{D}_j$ is updated and corrected by NNWarp. After that, we calculate the best-fitting linear transformation $\mathbf{A}_{j,k}$ for the small patch of the mesh interfacing $\mathcal{D}_j$ and one of its children domain say $\mathcal{D}_k$ as $\mathbf{A}_{j,k} = (\mathbf{P}_{j,k}\mathbf{P}_{j,k}^{\top})^{-1}\mathbf{P}_{j,k}^{\top}\mathbf{Q}_{j,k}$, where $\mathbf{P}_{j,k}$ and $\mathbf{Q}_{j,k}$ store the rest shape and deformed positions of all the nodes on the interface patch. We extract the relative rotation between $\mathcal{D}_j$ and $\mathcal{D}_k$ using the polar decomposition as $\mathbf{A}_{j,k} = \mathbf{R}_{j,k}\mathbf{S}_{j,k}$. Based on this, the angular velocity $\omega_{j,k}$ and the angular acceleration $\dot{\omega}_{j,k}$ can be calculated. Each domain is pinned to a local non-inertial reference frame. Therefore, in addition to the regular external forces, inertial forces originated from the accelerated linear and angular motion of the interface should also be computed as the *system force* and the *interface force*. We refer the reader to the related reference from Barbič and Zhao [14] for the detailed formulation.

An example is given in Fig. 1, where NNWarp is still based on a rectangular beam model. However, because we decompose the maple tree into domains of branches and leaves, the neural network well handles nonlinear dynamics for each domain regardless how complex the original mesh is. Unlike other tree simulation results in the literature [14], [65], [77], [78], the example given in the figure is simulated in the *fullspace* without any reduction of the simulation DOFs. Therefore, local high-frequency details are well preserved. The simulation is close to interactive at 5 FPS – this is roughly 1,000 times faster than running fullspace nonlinear simulation using substructuring.

5 Network Structure and Training

The input of our neural network includes kinematic features of the linear displacement of the i-th node $\widetilde{\mathbf{u}}_i$ and its instantaneous angular velocity $\mathbf{w}_i = \nabla \times \left(\mathbf{P}_i\mathbf{P}_i^{\top}\right)^{-1}\mathbf{P}_i^{\top}\mathbf{U}_i$ as in Eq. (6). As to be discussed shortly, we further compress this pair of vectors into three scalars utilizing the rotation invariance property of the isotropic hyperelastic material. Doing so significantly relieves the effort of generating the training set. Besides, three discriminative features of geodesic, potential and digression are also included. We find that the Young's modulus k behaves more like a linear amplifier. Increasing Young's modulus yields a deformation similar to the one obtained by reducing the magnitude of the external force. Therefore, this material parameter is not explicitly fed to the network. However, Poisson's ratio ν controls the volume change, and its impact on the final deformation is much more nonlinear. This is also reflected in the strain energy formulation of Eqs. (1), (3) and (8). As a result, the Poisson's ratio is also an input feature. Other simulation configurations like the external force, tessellation, boundary conditions are not the input since we believe this information is well encoded during the linear solve. The final input feature is a seven-dimension vector, and the network outputs a 3D vector of $\delta\mathbf{u}_i$ corresponding to a node-wise displacement fix so that $\widetilde{\mathbf{u}}_i + \delta\mathbf{u}_i$ is a well approximated nonlinear nodal displacement for a target material model. We use a different network for a different nonlinear material model instead of building a comprehensive one.

Training data alignment The complexity of a neural network highly depends on its input [9]. For instance, $\mathbf{w}_i$ can be extracted from the displacement gradient tensor $\mathbf{G}_i$. Nevertheless, if we simply put all the nine elements of $\mathbf{G}_i$ into the network, much higher training and testing errors are observed, which could only be improved by spanning the network depth and generating more training data. In order to make the network as compact as possible, we further align vectors $\widetilde{\mathbf{u}}$ and $\mathbf{w}$ based on the fact that *a nodal deformation measure can always be examined within a local coordinate frame which is invariant under rotations*.

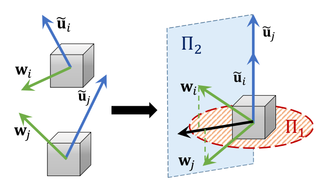


Figure 12: Rotation invariance allows us to further compress the input kinematic feature.

This procedure is illustrated in Fig. 12. Suppose we have two nodes i and j. They are surrounded by two infinitesimal volumes, which are small enough to be considered as symmetric in all the orientations. We first rotate these two volumes so that the linear displacements $\widetilde{\mathbf{u}}_i$ and $\widetilde{\mathbf{u}}_j$ are both in the positive y direction. The follow-up rotation is around the y axis. One can pick an arbitrary direction (the black vector in the figure) within plane Π_1, which is perpendicular to the y axis. In our implementation, we set this direction as the negative x axis. After that, $\mathbf{w}_i$ and $\mathbf{w}_j$ are rotated so that they both reside in plane Π_2 i.e. the xy plane in our implementation. Because the second rotation is around the y axis, $\tilde{\mathbf{u}}_i$ and $\tilde{\mathbf{u}}_j$ remain aligned. By doing so, pairs of $\tilde{\mathbf{u}}$ and $\mathbf{w}$ only differ at the magnitude or the norm of the linear displacement, the magnitude of $\mathbf{w}$ and the angle between them. In other words, the real useful kinematic information hidden behind vectors $\tilde{\mathbf{u}}$ and $\mathbf{w}$ are only three scalars. If one insists on putting the original $\tilde{\mathbf{u}}$ and $\mathbf{w}$ into the network, the net must learn this double-rotation alignment out of the training data first and then fits the linear-nonlinear map. Unfortunately, the neural network is not good at processing such rotation invariant features. For instance, in existing works of using deep learning to perform 3D shape analysis [79], [80], in order to relieve the burden of the analysis of rotation invariant shape features, it is common to use *rotation augmentation* that duplicates a training data by rotating it from multiple angles. The final result is pooled out of all the rotated duplicates. Using data alignment, the size of the training set is reduced by over 10 times, and the training time is also significantly shortened.

Generating training set It is important to make sure that the training set covers the feature space of the simulation, because machine learning is known to have a relatively poor performance for extrapolation. For the direction of the external force field, we evenly scatter samples over a unit semi-hemisphere surface for the rectangular beam model. Specifically, we uniformly sample two variables α and β from the interval of $[0, \pi/2]$, which correspond to the latitudinal and longitudinal spans on the semi-hemisphere. The unit directional vector can be calculated as: $\mathbf{e} = [\sin\beta\cos\alpha, \cos\beta, \sin\beta\sin\alpha]^\top$. The magnitude of the external force determines the magnitude of the linear displacement, which could be an infinitely large vector in theory. However, as we have already normalized our training model into a unit sphere, an excessively large displacement vector is unlike to occur in a real simulation application. Therefore, we stop applying bigger external force if $|\mathbf{u}_i| \geq 2$ during the training data generation. Such assumption could lead to artifacts if sharp and high-frequency external forces occur in the actual simulation. However in many other cases, pre-known external forces are applied, such as the gravity or the wind forces, and NNWarp can then be well leveraged based on compact training set.

The discriminative features g, p and d are essentially for model registration. Therefore, how they are sampled depends on the training model's geometry and tessellation. In general, a moderately fine mesh should suffice for these features. However, if the training model is too coarse i.e. with few hundred elements, one may observe artifacts after warping.

Training specifications In our implementation, the beam model for the network training consists of $2,629$ elements, and we generate $20,829$ training poses including $16,730,893$ training nodal pairs and $167,309$ validation nodal pairs. The test data is $1/7$ of the training data with $2,064,812$ nodal pairs. Training and testing data are stored as binary files in `.npy` format with a total size of 1.42 GB. Unlike [47], we do not need to load these training poses during the simulation. Only the resulting network parameters are needed. The network structure is rather simple – for co-rotation and Neo-Hookean materials there are only two hidden layers, and each of which has 16 neurons. For StVK material, the network has three hidden layers with 16 neurons at each layer.

The network is optimized using the Adam solver [81]. During the training, the neural network was built using `Google Tensowflow` [82] and optimized with `Google Cloud Platform` with 8 virtual CPUs. In practice, we typically stop at 10 epoches. The total training time is less than 10 minutes on `Google Cloud`. It takes similar time if one performs the training on an `i7` PC with a high-end video card. The minibatch size is 1,024 and the learning rate is set as 0.001. Two hyper-parameters β_1 and β_2 control the exponential decay rates of moving averages, which are set as $\beta_1 = 0.9$ and $\beta_2 = 0.999$, and $\varepsilon = 1e-8$. We use the `tanh` defined as $e^x - e^{-x}/e^x + e^{-x}$ as the nonlinear activation function. We found that `tanh` outperforms the widely-used `ReLU` in our experiment. We guess this is because the input-output relation of the net is clearly a smooth nonlinear function in our case, and `ReLU` may excel when the input-output relation contains discontinuity and/or singularity as in many computer vision problems like image recognition. Also, because all the training data generated using FEM simulation are clear and noise-free, and the data coverage is carefully controlled to avoid over- and under-sampling of the input feature space, we do not apply dropout during our training.

Generating all the training set takes about 154 *min* with multi-treading enabled. Together with the training time, the entire NNWarp preparation is 164 *min*. Thanks to our compact features, the network is simple, and applying it over each nodal displacement is also parallelizable. Therefore, NNWarp is orders-of-magnitude faster than the full nonlinear simulation. Take the dinosaur model an example, the standard full simulation runs only at 0.04 *FPS* and 164 *min* (i.e. the total time used for network training) only generates about 400 frames of animation (see Fig. 14). In other words, the NNWarp training easily gains a profit for producing an animation of moderate length.

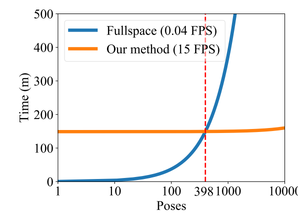


Figure 14: The training effort is paid off for an animation of moderate length.

6 Other Experimental Results

The simulator module was implemented using `MS Visual C++ 2013` on an `Alienware` desktop PC with an `Intel i7 5960` CPU (at 3.0 GHz) and 32 GB memory. It also equips with an `nVidia GTX 970` GPU. We used `Eigen C++` template for most numerical computations. Some of our implementations also used the published `Vega library` [83]. NNWarp utilizes a standard linear simulation running at background. The external force applied at each node needs to be rotated back to its rest-shape orientation as in stiffness warping [12]. This local rotation is computed by converting $\mathbf{w}_i$ into a rotation matrix, which only induces minor extra computing efforts since $\mathbf{w}_i$ itself is also the network's input. The timing statistics of examples shown in the paper are reported in Table 1. The source code (for both neural network and simulator) and executables can be found in the supplementary file. The training data (for the Neo-Hookean material) is also available from an anonymous `dropbox` link provided in the supplementary file.

Comparison with existing geometry warping methods First of all, we compare our method with existing geometry warping methods including modal warping (MW) [13] and rotation-strain warping (RSW) [67]. We stick with using the rectangular beam as our training model, and simulate the bending deformation of a Neo-Hookean toy statue using MW, RSW, NNWarp and fullspace FEM simulation. While all the methods demonstrate plausible

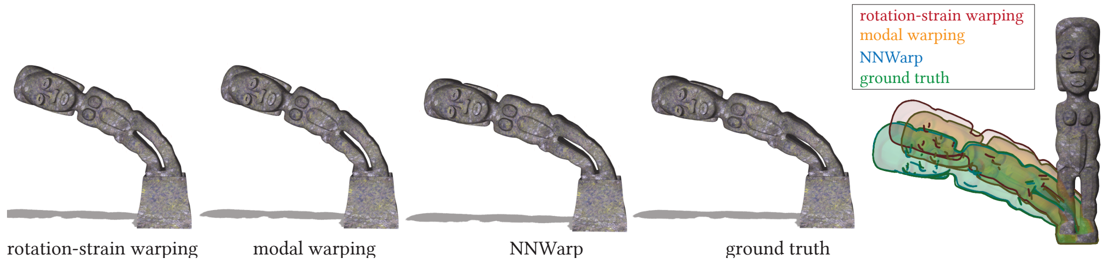


Figure 13: A side-by-side comparison shows a clear advantage of NNWarp over the existing warping techniques. Its data-driven nature makes the result almost identical to the ground truth while the simulation is as fast as the linear elasticity. The training still uses the rectangular beam model.

nonlinear bending effects, when putting together, one can see that MW and RSW are actually quite different from the ground truth result. On the other hand, NNWarp yields a result that is hardly distinguishable from the ground truth. Because MW and RSM use a fixed linear-nonlinear map template (i.e. Eqs (4) and (5)), they show no difference with different hyperelastic materials. However, NNWarp is able to produce high-quality results for various material models due to its data-driven nature.

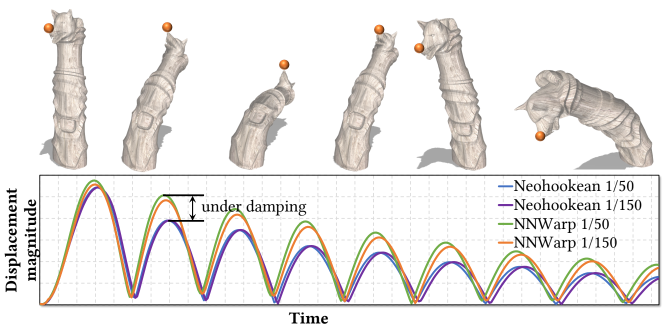


Figure 15: The deformable motion trajectory (at the nose tip of the wolf head) generated using NNWarp well matches the ground truth under different time step sizes. The vibration frequency resembles the ground truth as well. We use the Newmark integrator in this example.

Trajectory comparison Another aspect we would like to investigate is the motion trajectory, and see how far NNWarp deviates from the ground truth along the simulation time. To this end, we apply NNWarp to a wolf totem model and plot the displacement of the node at the nose tip of the wolf head w.r.t. time. Our reference is the fullspace FEM simulation using the Newmark integration and the material model is Neo-Hookean. In both cases, the totem model is scaled into a unit cube. We compare the resulting trajectories with time step size set as $1/50$ *sec* and $1/150$ *sec* respectively. Surprisingly, the trajectory generated using NNWarp is very close to the ground truth in both time step size settings. The vibration frequency is almost identical to the ground truth. This is probably because NNWarp is essentially a fullspace simulator, where the mass inertial is lossless unlike in reduced simulations. On the other hand, we do observe an artificial under-damping issue as we can see from the plotted trajectories. It seems that the linear Rayleigh damping dissipates less energy ($\sim 10\%$ in this example) than the nonlinear one. As NNWarp is a pure geometric-based correction, and it breaks a deformation trajectory into frames, adding new features into the current NNWarp network does not resolve this artifact. Instead, this issue should be fixed by dynamically adjusting the Rayleigh damping coefficients as did in [78].

More examples & GPU implementation With the help of substructuring method [14], training a single model can be utilized to handle geometrically complex deformable bodies. In addition to the example shown in Fig. 1, Fig. 16 shows more results using NNWarp. The Armadillo, dinosaur and dragon models are of StVK, co-rotation and Neo-Hookean materials respectively. The networks used are still based on a single rectangular beam model.

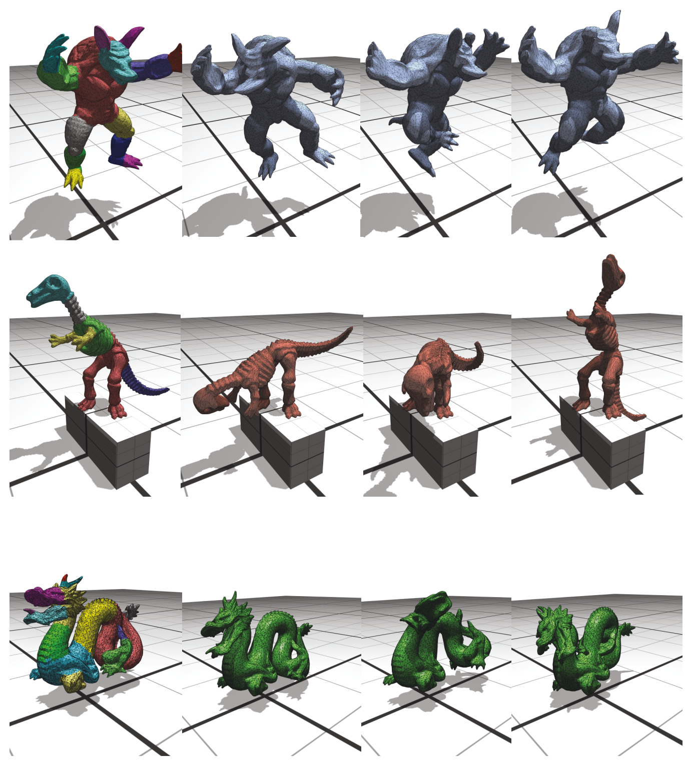


Figure 16: Substructuring allows us to re-use the training data of a regular shape to handle complex deformable bodies. The Armadillo, dinosaur and dragon models use the StVK, co-rotation and Neo-Hookean materials respectively. They use the networks trained with the rectangular beam.

NNWarp is node-wise. Its local correction of each nodal displacement is independent and can be parallelized trivially on GPU. We also implemented a shader version of NNWarp. NNWarp relies an underlying linear solver during the simulation run time. It is known that the asymptotic time complexity of solving a pre-factorized matrix is $\mathbf{O}(N^2)$ while NNWarp correction is just $\mathbf{O}(N)$. In other words, the benefit of the GPU implementation is limited in general. It is easy to see that NNWarp also synergizes well with model reduction. One can use the linear modal analysis

Model	# Tetrahedra	# Domains	Factorization	Solve	NNWarp (CPU)	NNWarp (Shader)	FPS (CPU/GPU)
Beam	$2,629$	1	$6.9ms$	$< 1ms$	$1.5ms$	$< 1ms$	333/666
Dragon	$51,850$	14	$307ms$	$15ms$	$15ms$	$< 1ms$	16/22
Armadillo	$52,278$	15	$403ms$	$15ms$	$18ms$	$< 1ms$	15/21
Dinosaur	$54,796$	14	$334ms$	$18ms$	$15ms$	$< 1ms$	18/24
Bunny	$24,956$	4	$273ms$	$10ms$	$7ms$	$< 1ms$	33/43
Maple bonsai	$255,552$	$1,771$	$1,556ms$	$83ms$	$109ms$	$< 1ms$	5/10

Table 1: Time performance of the examples reported in the paper. **Factorization** is the time needed to pre-factorize the system matrix of the linear elasticity. We use `SimplicialLLT` solver shipped with `Eigen`. During the simulation, we only need to solve the system once. **FPS** reports both CPU and GPU performance.

to construct a r-dimensional linear subspace. Because the model reduction is applied to the linear solver, other more expensive pre-computations like Cubature training [45] are not needed. The network training for NNWarp is much faster than the Cubature training. More importantly, Cubature training is model-dependant, while NNWarp training is more general. With the linear modal reduction, the cost for the diagonalized linear solver is reduced to $\mathbf{O}(r)$, and one should expect more noticeable accelerations by using the GPU. We do not report extra results using model reduced NNWarp since this is a natural extension and not the primary contribution of this work, nevertheless the simulation performance of the maple bonsai model shown in Fig. 1 can easily exceed 100 FPS with modal reduction.

When using the GPU-based NNWarp, some extra cares are needed for the deformation substructuring. This is because all the information regarding the final nonlinear displacement is in the GPU memory, which prevents us to evaluate the system and interface forces for per-domain dynamics at the CPU side. For the interface force, since it is assumed that the number of nodes on the domain's interface is small, we compute a CPU-based NNWarp for all the interface nodes to obtain their corrected displacement. For the system force, we treat an entire domain as a single mass point and estimate a domain-level rotation to warp it to the local non-inertial frame. Doing so compromises the physics accuracy, but avoids expensive data exchange from GPU and CPU.

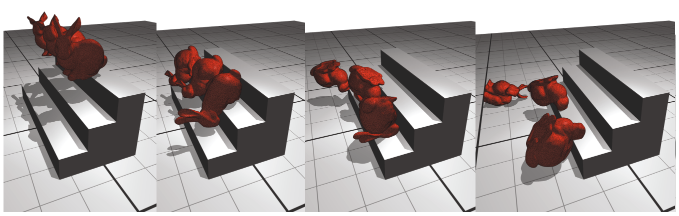


Figure 17: We can simulate free-floating deformable bodies by creating an artificial boundary condition to constrain the element near the mass center.

Free-floating deformable bodies Free-floating objects do not have boundary conditions, and our discriminative features are ill-defined under this situation. As an easy walk-round, we pick a tetrahedron that is closest to the mass center of the deformable body, constrain all of its four nodes and training the network based on it. During the simulation, we couple a rigid body simulator with the deformable simulation as in [84], where NNWarp is applied within the reference frame attached to the rigid body simulator (Fig. 17).

7 Conclusion

NNWarp uses a node-wise light-weight neural network to correct a linear displacement to be a nonlinear one. While it is conceptually similar to existing geometry warping method like stiffness warping, modal warping and rotation-strain warping, NNWarp yields better simulation results in terms of both shape deformations and motion trajectories. Observing that simply feeding kinematic feature into the network leads to serious artifacts, we design three discriminative features: geodesic, potential and digression to provide sufficient contextual information while these features are still quite general so that a network training can be used for deformable bodies of different shapes. Using the substructuring method, NNWarp can simulate large-scale and complex nonlinear deformable objects efficiently without repetitively generating new training poses and training the networks for unseen deformable bodies. The training data alignment also significantly reduces the training effort.

Limitations While it shows some unique advantages over the existing methods like its efficiency, accuracy and re-useable training, the current version of NNWarp also has many limitations. First of all, as a common drawback of learning-based methods, the performance of NNWarp drops rapidly if an extrapolation is needed. Figs 18 and 19 report limitation in detail. In Fig. 18, we keep the external force direction along negative y axis and gradually increase its magnitude. When the force magnitude is within the training coverage, NNWarp yields good results of low errors compared with ground truth. When the force magnitude exceeds the coverage of the training set indicating an extrapolation occurs (i.e. over 100%), we can see artifacts and errors increase accordingly. Similar behavior can also be observed when we keep the force magnitude unchanged, but alter the force field direction as shown in Fig. 19. In this example, our training force fields range from -90° to 0°. In other words, if the training set does not cover the feature vectors that appear in the simulation, NNWarp may produce unrealistic deformations. In our current setting, we only consider isotropic hyperelastic materials. While we believe NNWarp should be able to handle more complicated nonlinear materials like Mooney Rivlin, Yeoh, Ogden etc, doing so may require a re-design of contextual features and more training efforts since we cannot align training pairs within a local frame. Adding a new feature into the network input increases the sampling space by one dimension, which requires significant more training efforts. The corresponding network may also be more complex and deeper. We use directional and rotational force fields as the external forces in our current training data generation, both of which are low-frequency forces. As a result, NNWarp is less accurate when a high-frequency external force is applied i.e. during the collision and contact. One may observe popping artifact when the bunny hits the floor in Fig. 17. A potential solution may be to use the idea

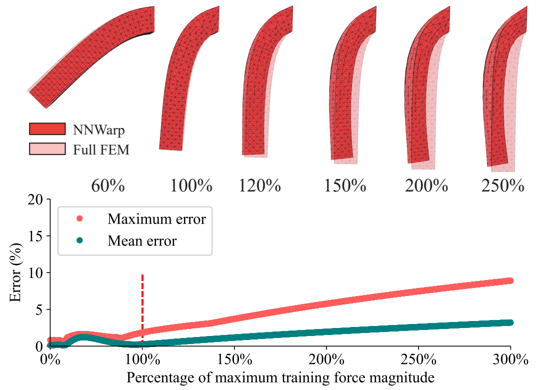


Figure 18: Extrapolation errors when the force magnitude exceeds the training coverage.

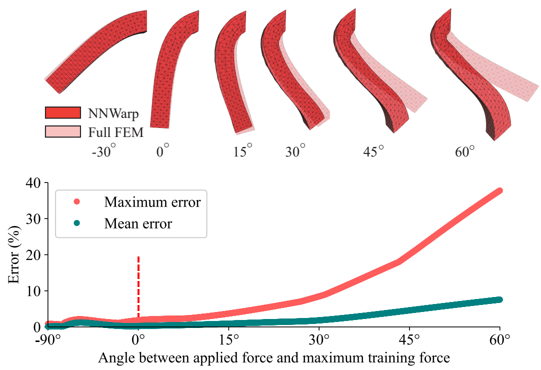


Figure 19: Extrapolation errors when the force direction exceeds the training coverage.

of condensation [85] by splitting the deformable body according to its contact regions and rolling NNWarp back to a regular nonlinear solver to accurately simulate detailed denting effects, or to exhaustively sample the high-frequency external forces during the network training.

8 Acknowledgement

We thank editors and anonymous reviewers for their detailed and constructive comments. Ran Luo and Yin Yang are partially supported by NSF under grant No. CHS-1717972 and AFRL under grant No. FA9453-18-2-0022. Tianjia Shao is partially supported by NSFC (No. 61772462 and No. U1736217) and Microsoft Research Asia. Weiwei Xu is partially supported by NSFC (No. 61732016), Alibaba IDEA Lab and fundamental research fund for the central universities (2017YFB1002600). Huamin Wang is partially supported by NSF CHS-1524992. Xiang Chen is partially supported by NSFC (No. 61772024 and No. 61732016). We also gratefully acknowledge the support of NVIDIA Corporation with the GPU donation. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of funding agencies or others.

References

[1] K.-J. Bathe, *Finite element method*. Wiley Online Library, 2008.

[2] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[3] J.-T. Huang, J. Li, D. Yu, L. Deng, and Y. Gong, "Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 7304–7308.

[4] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Transactions on audio, speech, and language processing*, vol. 20, no. 1, pp. 30–42, 2012.

[5] J. Pan, C. Liu, Z. Wang, Y. Hu, and H. Jiang, "Investigation of deep neural networks (dnn) for large vocabulary continuous speech recognition: Why dnn surpasses gmms in acoustic modeling," in *Chinese Spoken Language Processing (ISCSLP), 2012 8th International Symposium on*. IEEE, 2012, pp. 301–305.

[6] L. Wang, W. Ouyang, X. Wang, and H. Lu, "Visual tracking with fully convolutional networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 3119–3127.

[7] M. Kristan, J. Matas, A. Leonardis, M. Felsberg, L. Cehovin, G. Fernández, T. Vojir, G. Hager, G. Nebehay, and R. Pflugfelder, "The visual object tracking vot2015 challenge results," in *Proceedings of the IEEE international conference on computer vision workshops*, 2015, pp. 1–23.

[8] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[9] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 2, no. 4, pp. 303–314, 1989.

[10] S. Bouaziz, S. Martin, T. Liu, L. Kavan, and M. Pauly, "Projective dynamics: fusing constraint projections for fast simulation," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 154, 2014.

[11] H. Wang and Y. Yang, "Descent methods for elastic body simulation on the gpu," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 6, p. 212, 2016.

[12] M. Müller, J. Dorsey, L. McMillan, R. Jagnow, and B. Cutler, "Stable real-time deformations," in *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*. ACM, 2002, pp. 49–54.

[13] M. G. Choi and H.-S. Ko, "Modal warping: Real-time simulation of large rotational deformation and manipulation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 11, no. 1, pp. 91–101, 2005.

[14] J. Barbič and Y. Zhao, "Real-time large-deformation substructuring," in *ACM transactions on graphics (TOG)*, vol. 30, no. 4. ACM, 2011, p. 91.

[15] R. Dechter, *Learning while searching in constraint-satisfaction problems*. University of California, Computer Science Department, Cognitive Systems Laboratory, 1986.

[16] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2014, pp. 806–813.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[19] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1915–1929, 2013.

[20] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2014, pp. 1725–1732.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[22] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *arXiv preprint arXiv:1606.00915*, 2016.

[23] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1520–1528.

[24] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Region-based convolutional networks for accurate object detection and segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 1, pp. 142–158, 2016.

[25] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[26] M. A. Otaduy, B. Bickel, D. Bradley, and H. Wang, "Data-driven simulation methods in computer graphics: cloth, tissue and faces," in *ACM SIGGRAPH 2012 Courses*. ACM, 2012, p. 12.

[27] H. Wang, F. Hecht, R. Ramamoorthi, and J. F. O'Brien, "Example-based wrinkle synthesis for clothing animation," in *ACM Transactions on Graphics (TOG)*, vol. 29, no. 4. ACM, 2010, p. 107.

[28] L. Kavan, D. Gerszewski, A. W. Bargteil, and P.-P. Sloan, "Physics-inspired upsampling for cloth simulation in games," in *ACM Transactions on Graphics (TOG)*, vol. 30, no. 4. ACM, 2011, p. 93.

[29] H. Wang, J. F. O'Brien, and R. Ramamoorthi, "Data-driven elastic models for cloth: modeling and measurement," in *ACM Transactions on Graphics (TOG)*, vol. 30, no. 4. ACM, 2011, p. 71.

[30] E. Miguel, D. Bradley, B. Thomaszewski, B. Bickel, W. Matusik, M. A. Otaduy, and S. Marschner, "Data-driven estimation of cloth simulation models," in *Computer Graphics Forum*, vol. 31, no. 2pt2. Wiley Online Library, 2012, pp. 519–528.

[31] D. Kim, W. Koh, R. Narain, K. Fatahalian, A. Treuille, and J. F. O'Brien, "Near-exhaustive precomputation of secondary cloth effects," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, p. 87, 2013.

[32] W. Xu, N. Umetani, Q. Chao, J. Mao, X. Jin, and X. Tong, "Sensitivity-optimized rigging for example-based real-time clothing synthesis." *ACM Trans. Graph.*, vol. 33, no. 4, pp. 107–1, 2014.

[33] S. Coros, P. Beaudoin, and M. Van de Panne, "Robust task-based control policies for physics-based characters," in *ACM Transactions on Graphics (TOG)*, vol. 28, no. 5. ACM, 2009, p. 170.

[34] Y. Lee, K. Wampler, G. Bernstein, J. Popović, and Z. Popović, "Motion fields for interactive character locomotion," in *ACM Transactions on Graphics (TOG)*, vol. 29, no. 6. ACM, 2010, p. 138.

[35] X. B. Peng, G. Berseth, and M. Van de Panne, "Dynamic terrain traversal skills using reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, p. 80, 2015.

[36] L. Liu, M. V. D. Panne, and K. Yin, "Guided learning of control graphs for physics-based characters," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 3, p. 29, 2016.

[37] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[38] L. Liu and J. Hodgins, "Learning to schedule control fragments for physics-based characters using deep q-learning," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 3, p. 29, 2017.

[39] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 41, 2017.

[40] D. Holden, T. Komura, and J. Saito, "Phase-functioned neural networks for character control," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 42, 2017.

[41] B. Wang, L. Wu, K. Yin, U. M. Ascher, L. Liu, and H. Huang, "Deformation capture and modeling of soft objects." *ACM Trans. Graph.*, vol. 34, no. 4, pp. 94–1, 2015.

[42] H. Xu and J. Barbič, "Example-based damping design," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 53:1–53:14, Jul. 2017.

[43] M. Kim, G. Pons-Moll, S. Pujades, S. Bang, J. Kim, M. J. Black, and S.-H. Lee, "Data-driven physics for human soft tissue animation," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 54, 2017.

[44] B. Jones, N. Thuerey, T. Shinar, and A. W. Bargteil, "Example-based plastic deformation of rigid bodies," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 34, 2016.

[45] S. S. An, T. Kim, and D. L. James, "Optimizing cubature for efficient integration of subspace deformations," in *ACM transactions on graphics (TOG)*, vol. 27, no. 5, 2008, p. 165.

[46] L. Ladický, S. Jeong, B. Solenthaler, M. Pollefeys, and M. Gross, "Data-driven fluid simulations using regression forests," *ACM Trans. Graph.*, vol. 34, no. 6, pp. 199:1–199:9, Oct. 2015.

[47] M. Chu and N. Thuerey, "Data-driven synthesis of smoke flows with cnn-based feature descriptors," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 69:1–69:14, Jul. 2017.

[48] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer, "Elastically deformable models," in *ACM Siggraph Computer Graphics*, vol. 21, no. 4, 1987, pp. 205–214.

[49] M. Müller, B. Heidelberger, M. Teschner, and M. Gross, "Meshless deformations based on shape matching," *ACM transactions on graphics (TOG)*, vol. 24, no. 3, pp. 471–478, 2005.

[50] M. Pauly, R. Keiser, B. Adams, P. Dutré, M. Gross, and L. J. Guibas, "Meshless animation of fracturing solids," *ACM Transactions on Graphics (TOG)*, vol. 24, no. 3, pp. 957–964, 2005.

[51] S. Martin, P. Kaufmann, M. Botsch, E. Grinspun, and M. Gross, "Unified simulation of elastic rods, shells, and solids," in *ACM Transactions on Graphics (TOG)*, vol. 29, no. 4, 2010, p. 39.

[52] M. Desbrun, P. Schröder, and A. Barr, "Interactive animation of structured deformable objects," in *Graphics Interface*, vol. 99, no. 5, 1999, p. 10.

[53] T. Liu, A. W. Bargteil, J. F. O'Brien, and L. Kavan, "Fast simulation of mass-spring systems," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 6, p. 214, 2013.

[54] E. Sifakis and J. Barbič, "Fem simulation of 3d deformable solids: a practitioner's guide to theory, discretization and model reduction," in *ACM SIGGRAPH 2012 Courses*, 2012, p. 20.

[55] H. Wang, J. O'Brien, and R. Ramamoorthi, "Multi-resolution isotropic strain limiting," in *ACM Transactions on Graphics (TOG)*, vol. 29, no. 6, 2010, p. 156.

[56] G. Irving, J. Teran, and R. Fedkiw, "Invertible finite elements for robust simulation of large deformation," in *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 2004, pp. 131–140.

[57] F. Hecht, Y. J. Lee, J. R. Shewchuk, and J. F. O'Brien, "Updated sparse cholesky factors for corotational elastodynamics," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 5, p. 123, 2012.

[58] Y. Zhu, E. Sifakis, J. Teran, and A. Brandt, "An efficient multigrid method for the simulation of high-resolution elastic solids," *ACM Transactions on Graphics (TOG)*, vol. 29, no. 2, p. 16, 2010.

[59] R. Narain, M. Overby, and G. E. Brown, "Admm $\supseteq$ projective dynamics: Fast simulation of general constitutive models," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '16, 2016, pp. 21–28.

[60] H. Wang, "A chebyshev semi-iterative approach for accelerating projective and position-based dynamics," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 6, p. 246, 2015.

[61] T. Liu, S. Bouaziz, and L. Kavan, "Quasi-newton methods for real-time simulation of hyperelastic materials," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 3, p. 23, 2017.

[62] M. Fratarcangeli, V. Tibaldo, and F. Pellacini, "Vivace: a practical gauss-seidel method for stable soft body dynamics," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 6, p. 214, 2016.

[63] A. Pentland and J. Williams, "Good vibrations: Modal dynamics for graphics and animation," in *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '89, 1989, pp. 215–222.

[64] J. Barbič and D. L. James, "Real-time subspace integration for st. venant-kirchhoff deformable models," in *ACM transactions on graphics (TOG)*, vol. 24, no. 3, 2005, pp. 982–990.

[65] Y. Yang, D. Li, W. Xu, Y. Tian, and C. Zheng, "Expediting precomputation for reduced deformable simulation," *To appear in ACM TOG*, vol. 34, no. 6, 2015.

[66] T. Kim and D. L. James, "Skipping steps in deformable simulation with online model reduction," in *ACM transactions on graphics (TOG)*, vol. 28, no. 5, 2009, p. 123.

[67] J. Huang, Y. Tong, K. Zhou, H. Bao, and M. Desbrun, "Interactive shape interpolation through controllable dynamic deformation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 7, pp. 983–992, 2011.

[68] Z. Pan, H. Bao, and J. Huang, "Subspace dynamic simulation using rotation-strain coordinates," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 6, p. 242, 2015.

[69] C. von Tycowicz, C. Schulz, H.-P. Seidel, and K. Hildebrandt, "An efficient construction of reduced deformable objects," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 6, p. 213, 2013.

[70] J. Barbič, F. Sin, and E. Grinspun, "Interactive editing of deformable simulations," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, p. 70, 2012.

[71] S. Li, J. Huang, F. de Goes, X. Jin, H. Bao, and M. Desbrun, "Space-time editing of elastic motion through material optimization and reduction," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 108, 2014.

[72] P. Wolfe, "Convergence conditions for ascent methods," *SIAM review*, vol. 11, no. 2, pp. 226–235, 1969.

[73] H. Xu, W. Yu, S. Gu, and X. Li, "Biharmonic volumetric mapping using fundamental solutions," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 5, pp. 787–798, 2013.

[74] X.-M. Fu, Y. Liu, and B. Guo, "Computing locally injective mappings by advanced mips," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, p. 71, 2015.

[75] N. Aigerman and Y. Lipman, "Injective and bounded distortion mappings in 3d," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, p. 106, 2013.

[76] B. Andrásfai, *Introductory graph theory*. Pergamon Press, 1977.

[77] Y. Zhao and J. Barbič, "Interactive authoring of simulation-ready plants," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, p. 84, 2013.

[78] B. Wang, Y. Zhao, and J. Barbič, "Botanical materials based on biomechanics," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 135, 2017.

[79] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong, "O-cnn: Octree-based convolutional neural networks for 3d shape analysis," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 72:1–72:11, Jul. 2017.

[80] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1912–1920.

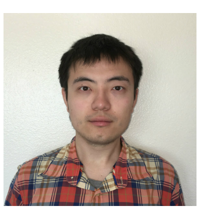
[81] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[82] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[83] F. S. Sin, D. Schroeder, and J. Barbič, "Vega: Non-linear fem deformable object simulator," in *Computer Graphics Forum*, vol. 32, no. 1, 2013, pp. 36–48.

[84] D. Terzopoulos and A. Witkin, "Physically based models with rigid and deformable components," *IEEE Computer Graphics and Applications*, vol. 8, no. 6, pp. 41–51, 1988.

[85] Y. Teng, M. Meyer, T. DeRose, and T. Kim, "Subspace condensation: Full space adaptivity for subspace deformations," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, p. 76, 2015.



Ran Luo received the bachelor degree in Beihang University, previously known as Beijing University of Aeronautics and Astronautics, in 2014. He is currently pursuing the Ph.D. degree in Computer Engineering at the University of New Mexico (UNM), Albuquerque, New Mexico. His research interests include physics-based animation/simulation, virtual reality, machine learning and related topics. He is currently a Research Assistant with the Department of Electrical and Computer Engineering at UNM.

Tianjia Shao is currently a Lecturer in the School of Computing, University of Leeds. He received his PhD in Computer Science from Institute for Advanced Study, and his B.S. from the Department of Automation, both in Tsinghua University. His current research focuses on semantic 3D modeling from consumer hardware (e.g. RGB/RGBD cameras) and common 2D contents (e.g. photographs and sketches), and structure/function aware geometry processing.

Weiwei Xu Weiwei Xu received the BS and master's degrees in computer science from Hohai University in 1996 and 1999, respectively, and the PhD degree in computer graphics from Zhejiang University, Hangzhou. He is a Researcher at Zhejiang university in the Department of Computer Science. Before that, he was a Qianjiang professor at Hangzhou normal university and researcher in Internet Graphics Group, Microsoft Research Asia from October 2005 to June 2012. He has been a postdoc researcher at Ritsmeikan university in Japan around one year from 2004 to 2005. He has published more than 60 papers on international conference and journals, include 16 papers on ACM Transaction on Graphics. His main research interests include digital geometry processing and physics-based simulation techniques. He is now focusing on 3D reconstruction and computational fabrication techniques.

Huamin Wang is an associate professor in the department of Computer Science and Engineering at the Ohio State University. He received his Ph.D. degree in Computer Science from Georgia Institute of Technology, the M.S. degree in Computer Science from Stanford University, and the B.Eng. degree from Zhejiang University. He is the recipient of the College of Engineering's Lumley research award from OSU in 2017. He received NVIDIA fellowship in 2006.

Xiang Chen Xiang Chen is an Associate Professor in the State Key Lab of CAD&CG, Zhejiang University. He received His Ph.D. in Computer Science from Zhejiang University in 2012. His current research interests mainly include fabrication-aware design, image analysis/editing, shape modeling/retrieval and computer-aided design.

Kun Zhou is a Cheung Kong Professor in the Computer Science Department of Zhejiang University, and the Director of the State Key Lab of CAD&CG. Prior to joining Zhejiang University in 2008, Dr. Zhou was a Leader Researcher of the Internet Graphics Group at Microsoft Research Asia. He received his B.S. degree and Ph.D. degree in computer science from Zhejiang University in 1997 and 2002, respectively. His research interests are in visual computing, parallel computing, human computer interaction, and virtual reality. He currently serves on the editorial/advisory boards of ACM Transactions on Graphics and IEEE Spectrum. He is a Fellow of IEEE.

Yin Yang received his Ph.D. degree in computer science from the University of Texas at Dallas in 2013. He is an Assistant Professor in Department of Electrical Computer Engineering at the University of New Mexico, Albuquerque. His research interests include physics-based animation/simulation and related applications, scientific visualization and medical imaging analysis.