DOI: 10.1111/cgf.14736

COMPUTER GRAPHICS *forum*
Volume 42 (2023), number 6, e14736

Numerical Coarsening with Neural Shape Functions

Ning Ni, Qingyu Xu, Zhehao Li, Xiao-Ming Fu and Ligang Liu

University of Science and Technology of China, Hefei, China
{nining, liamxu123, zhehaoli, fuxm}@mail.ustc.edu.cn, lgliu@ustc.edu.cn

Abstract
We propose to use nonlinear shape functions represented as neural networks in numerical coarsening to achieve generalization capability as well as good accuracy. To overcome the challenge of generalization to different simulation scenarios, especially nonlinear materials under large deformations, our key idea is to replace the linear mapping between coarse and fine meshes adopted in previous works with a nonlinear one represented by neural networks. However, directly applying an end-to-end neural representation leads to poor performance due to over-huge parameter space as well as failing to capture some intrinsic geometry properties of shape functions. Our solution is to embed geometry constraints as the prior knowledge in learning, which greatly improves training efficiency and inference robustness. With the trained neural shape functions, we can easily adopt numerical coarsening in the simulation of various hyperelastic models without any other preprocessing step required. The experiment results demonstrate the efficiency and generalization capability of our method over previous works.

Keywords: animation, numercial coarsening, physically based animation

CCS Concepts: • Computing methodologies → Physics simulation

1. Introduction

The pursuit of efficiency in the simulation of complex elastic models is an ever-lasting goal in the field of computer graphics. Among all the proposed acceleration techniques, numerical coarsening is one of the promising solutions, in which the computation is applied on the coarse level of resolution with reduced degrees of freedoms (DoFs) rather than on the fine mesh, as the simulation cost usually increases rapidly as the resolution of the model gets finer.

Existing methods [KMOD09, NKJF09, TREO16, KMOD09, CBW*18] consider the coarsening process as an analytical solution or optimization problem with guaranteed accuracy, which can be applied to the simulation of co-rotational linear materials and hyperelastic materials limited in small deformation. One of the recent works [CBW*18] proposes to modify the shape functions of coarse models from scalars to matrices with increased DoFs. However, these methods suffer from a lack of generalization capability. First, they require preprocessing to be conducted in a cumbersome case-by-case way, which fails to generalize to a broad range of simulation scenarios with large deformation while achieving good accuracy. Second, their formulations are still linear, which constrains the expression capability of shape functions. Their methods only behave well for small strains or elastic energy when applied to nonlinear materials.

Inspired by the recent works on learning-based methods [FMD*19, RCPO21], our key idea is to extend the linear mapping between coarse and fine meshes used in [CBW*18] to a nonlinear one by replacing the linear shape functions in numerical coarsening with a nonlinear one to increase the DoFs further. Furthermore, to achieve good generalization capabilities, we adopt a neural representation of the shape function trained from sampled simulation data, enabling our numerical coarsening method to simulate hyperelastic materials with large deformation.

However, directly applying an end-to-end neural representation of shape functions leads to poor performance. The reasons are two folds. First, the space of network parameters is too large to get enough training data with an acceptable number of samples. Second, some intrinsic geometry properties of shape functions, which must be met, may fail to be captured by the plain neural representation, which leads to unstable inference performance in simulation.

To overcome these shortcomings, we propose embedding the geometry constraints as prior knowledge in training the neural shape

© 2023 Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd.

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

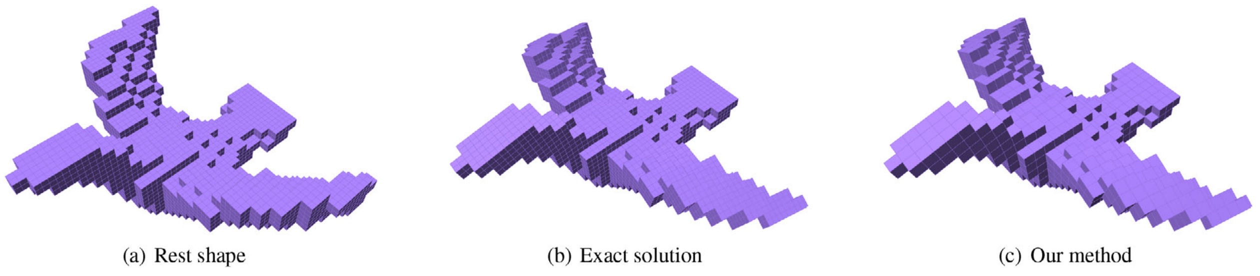


Figure 1: *We compare the simulation on fine grids (b) and coarse grids with our new numerical coarsening method (c). Our method can generalize to different models without any other preprocessing step while achieving good accuracy. For the bird model above, the simulation using our method is 16 times faster than that on fine grids. The position's mean error of our method is* 3.0×10^{-3}.

functions. Specifically, we constrain the learned shape function to be a quasi-linear one, which significantly reduces the dimension of parameter space and improves the robustness of the result in numerical coarsening.

The experiment results (Fig. 1) demonstrate the efficiency and generalization capability of our method. Our trained neural shape functions can be directly applied to different models and materials without requiring other preprocessing. Furthermore, compared with [CBW*18], our method can achieve about twice the accuracy.

Our paper makes the following technical contributions:

- We adopt learning-based shape functions in numerical coarsening to achieve both generalization capability and good accuracy compared with traditional methods.
- We propose to embed specific geometry constraints as prior knowledge in training neural shape functions, which significantly improves the training efficiency and inference robustness.

2. Related Work

2.1. Model reduction

Efficient simulation based on the finite element method (FEM) has been developed for many years. Model reduction is a set of methods that are used widely and attempt to use less DoFs to describe the simulated system, such as the methods described in [HSO03] and [XLCB15]. They approximate the deformation with the linear combination of a set of basis. The basis of the deformation space may be derived from PCA analysis or be chosen by the user. Many other example-based simulations are similar to model reduction methods by also representing the deformation space with a set of finite basis. However, this type of method incurs errors in the simulation of nonlinear material due to the limitation of linear combination. Moreover, the choice of basis greatly influences the simulation results.

2.2. Traditional numerical coarsening

Another set of methods is numerical coarsening. The coarsening method simply embeds the fine grids into the coarse grids in 3D space. But if we use the same material parameters with the fine mesh on the coarse grids, the elastic body usually behaves stiffer than the fine grids. To increase accuracy, many algorithms have been proposed to modify the coarse grids. [KMOD09] attempt to compute the elasticity tensor to capture the physical behaviour, which can be considered as trying to modify the material parameters to fit the ground truth of the simulation. [CBW*18] compute the matrix-valued shape functions for FEM simulation. Such an increase in the DoFs of shape functions helps to capture the physical behaviour without changing material parameters. Global methods have also been implemented to improve the accuracy of dynamic solutions. [CLMK17] and [CLK*19] approximate the global eigenvalues on the coarse mesh. [CBO*19] introduce a hierarchical construction of material-adapted basis functions, which is compromised between global accuracy and local efficiency. However, these methods fail to generalize to different simulation scenarios without case-by-case preprocessing.

2.3. Data-driven methods

A lot of effort has been put into data-driven physics simulation recently. [CLSM15] compute the material parameters of coarse grids to replace the ones of fine grids by data-driven methods. Besides, many approaches tend to implement model reduction via neural networks. [FMD*19] use an autoencoder to construct nonlinear subspaces automatically. Some algorithms have been used to separate the subspace apart and learn only the nonlinear corrections, such as those applied in [RCPO21] and [SYS*21]. Beyond model reduction, NNWarp in [LSW*18] makes use of simple networks to correct linear nodal deformation to nonlinear ones. Some also propose to adopt learning methods in fluid simulations, such as in [KAT*19]. Alternatively, data-driven approaches can be applied to add fine details. Chu et al. [CT17] train a feature descriptor with CNNs to match high-resolution results, while Lahner et al. [LCT18] use GANs to add high-frequency details. However, to the best of our knowledge, few of the applications of the data-driven methods in numerical coarsening have appeared yet. Different from the way to find the best-fit parameters for coarse mesh in the database in [CLSM15], our method to learn an appropriate shape function can act more flexibly and elegantly.

3. Preliminaries

We begin with traditional FEM simulation methods for elastic bodies and introduce some notations. Let $\Omega_0 \subseteq \mathbb{R}^3$ be the initial (rest) configuration of the elastic body in 3D space and $X \in \Omega_0$ be the material point in the initial configuration. Let $\Omega \subseteq \mathbb{R}^3$ be the deformed configuration in 3D space and $x = x(X) \in \Omega$ be the position in the deformed configuration corresponding to the point $X \in \Omega_0$. Similarly, many physical quantities can be considered as a function of X, such as the density of the material $\rho = \rho(X)$ and the displacement $u(X) = x(X) - X$.

3.1. Neo-Hookean constitutive model

We focus on the neo-Hookean model, a hyperelastic material, to compute the elastic force. The neo-Hookean material defines the elastic energy density in a nonlinear way:

$$\varphi(x) = \varphi(F) = \frac{\mu}{2}(J^{-\frac{2}{3}} I_c - 3) + \frac{\kappa}{2}(J-1)^2, \quad (1)$$

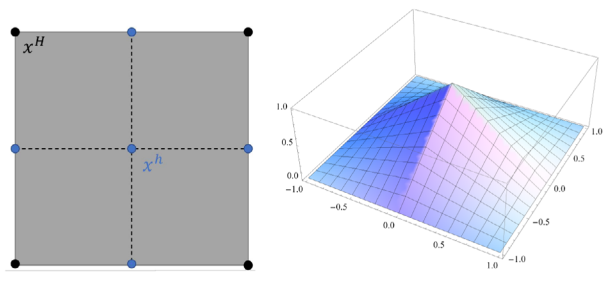
where μ and κ are the coefficients related to the material, $F = \frac{\partial x}{\partial X}$ denotes the deformation gradient and $J = det(F)$, $I_c = tr(F^T F)$. With Equation (1), we can compute the first Piola-Kirchhoff stress $P = \frac{\partial \varphi}{\partial F}$ and the elastic force.

3.2. Shape functions

Shape functions (or basis functions) are used to describe the geometry of the space. To numerically solve a PDE problem, we need to discretize the domain Ω_0 into a set of subdomains Ω_e and nodes x_i. For each element Ω_e and its corner nodes x_i, we define the shape function as $N_i : \Omega^h \longmapsto \mathbb{R}$, which is required to satisfy the Lagrange property: $N_i(x_j) = \delta_{ij}$. Then we can interpolate the displacement field in the domain from u_i, the displacement of the nodes, as

$$u(X) = \sum_i N_i u_i. \quad (2)$$

Specifically, since the trilinear shape functions of hexahedral elements in 3D space are widely used, we only consider voxel grids. In Figure 2b, a bilinear shape function in 2D space is shown, which is similar to the trilinear function in 3D space.



(a) the coarse grid (b) the shape function

Figure 2: *In 2D space, every single coarse element consists of* 2×2 *fine elements. As shown in (a), four corner points are the DoFs of coarse grids and all nine points are the DoFs of fine grids. (b) plots a bilinear function in domain* $[-1, 1]^2$, *which is usually used as the shape function for 2D simulation (similar with 3D simulation).*

3.3. Space discretization

We divide Ω_0 into several subdomains Ω_e (such as hexahedral areas in 3D space) with nodes $x_i \in \Omega_e$. The number of nodes in Ω_0 is denoted as n_e. Then, we can describe the elastic body with $x(t, X) = \Sigma_i N_i(X) x_i(t)$ or $x(t, X) = D(t)N(X)$, where $D(t) = (x_1(t), \ldots, x_{n_e}(t)) \in \mathbb{R}^{3\times n_e}$ and shape function $N(X) = (N_1(X), \ldots, N_{n_e}(X))^T \in \mathbb{R}^{n_e \times 1}$.

The deformation gradient is computed as

$$F(t) = \frac{\partial x}{\partial X} = D(t)\nabla N(X), \quad (3)$$

where $\nabla N(X) = (\nabla N_1(X), \ldots, \nabla N_{n_e}(X))^T \in \mathbb{R}^{n_e \times 3}$. With the deformation gradient, we can quickly compute the elastic energy.

3.4. Force computation

To describe the mechanic system, we define the Lagrangian by $L = T - V$, where T is the total kinetic energy of the system and V is the potential energy of the system. Then, we can get

$$T = \int_\Omega \frac{1}{2}\rho \dot{x}^T \dot{x} \mathrm{d}x = \int_{\Omega_0} \frac{1}{2}\rho_0 \dot{x}^T \dot{x} \mathrm{d}X \quad (4)$$

and

$$V = \int_{\Omega_0} (\rho_0 g^T x + \varphi(x)) \mathrm{d}X. \quad (5)$$

Here $\rho = \rho(x)$ is the density of the material at point x in Ω , $\rho_0 = \rho_0(X)$ is the density of the material at point X in Ω_0, $\dot{x} = \frac{\mathrm{d}x}{\mathrm{d}t}$ is the velocity of point x, and g is the gravitational acceleration.

With the Euler-Lagrange equations $\frac{\mathrm{d}}{\mathrm{d}t}(\frac{\partial L}{\partial \dot{x}_j}) = \frac{\partial L}{\partial x_j}$, we can get the equation:

$$\int_{\Omega_0} \rho_0 (NN^T) \bigotimes I_3 \frac{d^2 \overline{x}}{dt^2} \mathrm{d}X = -\int_{\Omega_0} (\rho_0 N \bigotimes g + \frac{\partial \varphi(x)}{\partial \overline{x}}) \mathrm{d}X, \quad (6)$$

where $\bigotimes$ denotes the Kronecker product, I_3 is the 3×3 identity matrix, and $\overline{x}(t) = vec(D(t))$ (the vectorization of $D(t)$) indicates the list of all DoFs.

3.5. Time integration

Using the backward Euler time integration, we can solve the following optimization problem to get the $(i+1)^{th}$ time step states:

$$\min_x \ g(x) = \frac{1}{2}(x - y)^T M (x - y) + \Delta t^2 V(x), \quad (7)$$

where $M = \int_{\Omega_0} \rho_0 (NN^T) \bigotimes I_3 \mathrm{d}X$, $y = x_i + \Delta t v_i$ is constant, and Δt is the time step. Usually, we solve the problem with Newton-Rapson method with the following iteration scheme:

$$x^{(n+1)} = x^{(n)} - [\nabla^2 g(x^{(n)})]^{-1} \nabla g(x^{(n)}). \quad (8)$$

3.6. Shape function based coarsening

For shape function-based numerical coarsening in [CBW*18], they use matrix-value functions instead of scalar functions as the shape functions for the coarse grids:

$$\begin{aligned} x(X) &= \sum_j N_j^h(X)x_j^h && \text{in fine grids,} \\ x(X) &= \sum_i N_i^H(X)x_i^H && \text{in coarse grids,} \\ N_i^H &= \sum_j n_{ij}N_j^h, \\ x(X) &= \sum_j [N_j^h(\sum_i n_{ij}x_i^H)] && \text{in coarse grids.} \end{aligned} \tag{9}$$

Here $N_j^h \in \mathbb{R}$ and $N_i^H \in \mathbb{R}^{3\times3}$ denote the shape functions of the fine and coarse grids, respectively. $n_{ij} \in \mathbb{R}^{3\times3}$ are the coefficients that need to be computed. The last equation of Equation (9) is derived from the others. By converting N_i^H from scalar to matrix, they can achieve more DoFs and represent larger deformation space. From the first and the last equations in (9), the matrix-value shape functions can be regarded as $h : x^H \in \mathbb{R}^{24} \longmapsto x^h \in \mathbb{R}^{81}$ being the linear mapping from coarse nodes to fine nodes.

4. Method

As introduced in Section 3.6, the shape function works as a bridge to connect the coarse and fine mesh for deformation mapping in numerical coarsening. In [CBW*18], they modify the shape functions of coarse meshes from scalars to matrices with increased DoFs; but, the mapping h is still defined as linear, which restricts their generalization capability to be applied to nonlinear materials with large deformation. Our key idea is to further increase the DoFs of h by adopting a nonlinear mapping to compute x^h from x^H instead of a linear one. Specifically, we use a learning-based neural network as shape functions for its generalization capability.

4.1. Basic neural shape functions

In the beginning, we tried to train an end-to-end neural representation of shape functions $h : x^H \in \mathbb{R}^{24} \longmapsto x^h \in \mathbb{R}^{81}$.

We use a three-layer multilayer perceptron (MLP) to learn the neural representation of the mapping h. The input variables of the neural network include the positions of nodes $x^H \in \mathbb{R}^{24}$ of a coarse element and Young's modulus ($\theta \in \mathbb{R}^8$) of eight fine elements. We fix the input Poisson's ratio as it usually changes only in a small range in simulations. The network output is the positions of fine nodes $x^h \in \mathbb{R}^{81}$. Then we can get the deformation $x(X)$ with the node positions and the shape functions of fine grids using (2) (we use the traditional trilinear function as the shape function of fine grids).

We notice that we can reduce the dimension of network output as we can expect the coarse nodes and their corresponding fine nodes to have the same spatial positions:

$$Kh(x_1^H, \ldots, x_8^H) = x^H. \tag{10}$$

Here K is the selection matrix (i.e., $Kx^h = x^H$). Therefore, we ignore these DoFs when we compute the $h(x^H)$ or just replace the mapping h with $h' = Kh : \mathbb{R}^{24} \longmapsto \mathbb{R}^{81-24=57}$.

4.2. Training data generation

The neural network is trained in a supervised manner. We generate training data by solving FEM simulations to get the data pair $((x^H, \theta), x^h)$. We use the approach of [KMOD09] and solve the following problem:

$$\begin{aligned} \nabla \cdot \sigma(p) &= 0 && \text{in } \Omega_e, \\ \sigma(p) &= \sigma_0 \cdot \mathbf{n}, && \text{on } \partial\Omega_e. \end{aligned} \tag{11}$$

Here σ is the stress tensor, p is the displacement, and $\mathbf{n}$ is the normal vector of $\partial\Omega_e$. We solve the problem above with different σ_0 and get the solution displacements p or position x^h. Then we get $x^H = Kx^h$. In this way, we generate the training data.

We further reduce the amount of required training data by noticing that h is invariant to scaling when we scale some parameters:

$$\begin{aligned} h(x^H, \lambda\theta) &= h(x^H, \theta), \\ h(\lambda x^H, \theta) &= \lambda h(x^H, \theta). \end{aligned} \tag{12}$$

Here $\lambda > 0$ is a scalar. Figure 3 shows that the deformation is similar when we scale some parameters. So we can get all the training data from a unit-size voxel grid and only care about the material parameters' relative ratio of different elements instead of its absolute value.

For training data sampling, we randomly select material parameters θ and different stress tensors (σ_0 in Equation (11)) to generate different deformation samples. Then we randomly select the parameters (the ratio of Young's modulus) and apply the simulation to the elastic object to compute the relationship between x^h and x^H with different stress. Besides the six harmonic displacements (in Figure 4) computed in [KMOD09] with six selected stress tensor σ_0, we randomly sample other stress tensors with a linear combination form of the six stress. When we select different stresses, we may get illegal results (i.e., the inverted hexahedral element), which usually happens when the scale of external force is too large. We will check it by computing the determinant of the deformation gradient and exclude the illegal ones.

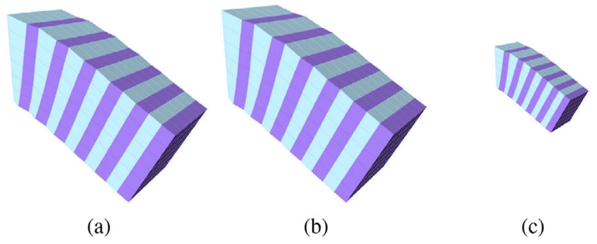


Figure 3: *Our method is scale-invariant: (a) the deformation of a bar under gravity; (b) a deformed bar with two times Young's modulus under two times gravity of (a); (c) a deformed bar with half size under two times of gravity of (a), which has similar deformation with (a).*

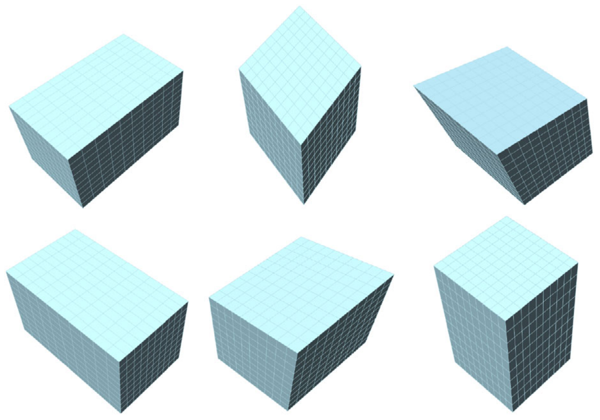


Figure 4: *Our method adopted the six harmonic displacements generated in [KMOD09].*

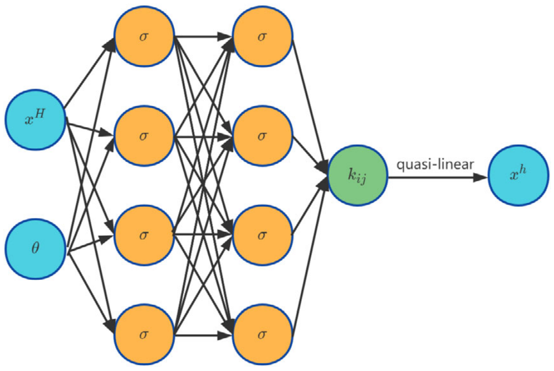


Figure 5: *Instead of an end-to-end neural representation of the relationship between x^H and x^h, we train it with a quasi-linear formulation.*

Table 1: *Experiment results of training an end-to-end neural shape function versus training with geometry constraints embedded as prior knowledge. With the prior knowledge we proposed, the network requires fewer neurons while achieving better performance.*

MLP form	#Neurons (#layers)	Loss	Training time (h)
End-to-end	800(9)	5.5×10^{-2}	5
End-to-end	700(8)	5.9×10^{-2}	5
End-to-end	2500(8)	8.8×10^{-1}	7
End-to-end	500(4)	8.3×10^{-1}	3
Quasi-linear	900(8)	1.0×10^{-3}	7
Quasi-linear	400(4)	1.2×10^{-2}	4
Quasi-linear	500(4)	9.0×10^{-3}	4

4.3. Learning with geometry prior

However, directly adopting the end-to-end network suffers from a couple of serious problems. First, the space of network parameters is too large to get enough training data with an acceptable number of samples. Second, some intrinsic geometry properties of shape functions that must be met may fail to be captured by the plain neural representation, which is introduced below.

4.3.1. *Geometry constraints*

When we translate or rotate an object, its strain or stress will not change. Hence, all the nodes of the object will have the same rigid transform, that is, the same translation or rotation, which means:

$$\begin{aligned} h_j(x_1^H + t, \dots, x_8^H + t) &= h_j(x_1^H, \dots, x_8^H) + t, \\ h_j(Rx_1^H, \dots, Rx_8^H) &= Rh_j(x_1^H, \dots, x_8^H). \end{aligned} \tag{13}$$

Here $t \in \mathbb{R}^3$, $R \in SO(3)$ and $h(x^H) = (h_1^T, \dots, h_{27}^T)^T$.

4.3.2. *Prior knowledge in learning*

To satisfy the above constraints, we adopt the following quasi-linear formulation for the learned mapping, which is similar to the formulations in [VSB17]:

$$h_j(x_1^H, \dots, x_8^H) = \sum_i k_{ij}(\|x_i^H - x_t\|, \theta)(x_i^H - x_t) + x_t. \tag{14}$$

Here $x_t = \frac{1}{8}\sum_i x_i^H$ and θ is the material parameters. We learn the function $k_{ij}(\|x_i^H - x_t\|, \theta)$ which maps $\|x_i^H - x_t\|$ and θ to $x^h \in \mathbb{R}$.

With the required form above, h will automatically be invariant to rigid transforms, that is, translation and rotation. Thus, we take these geometric constraints as the prior knowledge to only learn k_{ij} with neural networks, which means we restrict the form of the learned neural shape functions as a quasi-linear one (Figure 5). The experiment results (Table 1) show that training with this prior knowledge requires fewer network parameters while achieving better performance.

4.4. Force computation and simulation scheme

Once we get the mapping h from the coarse nodes to the fine nodes represented by our neural shape function, we can compute the force on the coarse mesh. Similar to the fine mesh, we compute the Lagrangian with the mapping $\bar{x} = vec(D) = x^h = h(x^H)$:

$$\begin{aligned} T &= \int_{\Omega} \frac{1}{2}\rho \dot{x}^T \dot{x} \mathrm{d}x = \int_{\Omega_0} \frac{1}{2}\rho_0 \dot{x}^T \dot{x} \mathrm{d}X \\ &= \int_{\Omega_0} \rho_0 \nabla^T h(x^H)(NN^T) \bigotimes I_3 \nabla h(x^H) \dot{x}^H \mathrm{d}X \end{aligned} \tag{15}$$

and

$$\begin{aligned} V &= \int_{\Omega_0} (\rho_0 g^T x + \varphi(x)) \mathrm{d}X \\ &= \int_{\Omega_0} (\rho_0 g^T (N^T \bigotimes I_3) h(x^H) + \varphi(x)) \mathrm{d}X. \end{aligned} \tag{16}$$

Algorithm 1. Simulation for one step

```
Require: The point position ($x^H$) of coarse grid in step $i$, material
  parameters ($\theta$)
Output: The point position ($x^H_+$) of coarse grid in step $i+1$
1: while Residual of Equation (17) is larger than a given threshold
     do
2:   for All coarse elements (nodes $x^H_e$) do
3:     1. Compute the corresponding fine grids $x^h$ and the
   gradient $\frac{\partial x^h}{\partial x^H}$ by the learned shape functions.
4:     2. Evaluate gravity and elastic forces with their gradient at
   integral points with $x^h$ and Equation (2).
5:     3. Assemble the matrix and vector in Equation (17).
6:     4. Solve Equation (8) with Newton-Rapson method.
7:   end for
8: end while
```

We get the motion equation from the Euler-Lagrangian equations:

$$\int_{\Omega_0} \rho_0 \nabla^T h(x^H)(NN^T) \bigotimes I_3 \nabla h(x^H) \frac{\mathrm{d}^2 x^H}{\mathrm{d}t^2} \mathrm{d}X$$
$$= -\int_{\Omega_0} \nabla h(x^H)(\rho_0 N \bigotimes g + \frac{\partial \varphi(x^h)}{\partial x^h}) \mathrm{d}X. \tag{17}$$

Here we ignore the second-order derivatives for efficiency. The simulation results show that the accuracy is acceptable with the approximation.

In all, the algorithm of the simulation is shown in Algorithm 1 and the simulation pipeline is shown in Figure 6.

5. Results

We have trained our neural network using Pytorch with an RTX 3060 GPU and implemented the simulation using C++ with an Intel Core i9-10900K 3.7 GHz CPU on a desktop workstation computer. We demonstrate the feasibility and effectiveness of our algorithm with various models and experiments.

5.1. The choice of material parameters

In our experiments, we fix the size of the grids. As described in Section 4.2, we can derive that the deformation of large grids behaves similarly to that of small grids, which only differs by a scalar (the external force needs to be scaled by the same constant). We fixed Poisson's ratio as 0.45 to evaluate the influence of different Young's modules. The deformation only depends on the ratio between the fine elements and the external force. So we set Young's modulus of the stiffest material 50 times that of the softest material.

5.2. Network training configuration

We use a fully connected, three-layer neural network to model k_{ij} in (14), which is a mapping from $\mathbb{R}^n$ to $\mathbb{R}$. We use tanh as the activation function. We generate the training data by solving (11) with different stress tensors. As illustrated in Section 4.2, we randomly select the stress tensor with a uniform distribution and generate 100 thousand displacement samples for about five days.

We use the ℓ_2 norm of the difference between the estimated value and the ground truth as the loss function. We optimize the network using Adam with 2000 to 4000 epochs, a batch size of 128, and a learning rate of 10^{-3}. We trained our neural network for about 8 h and achieved about 10^{-2} ℓ_2 norm error on test data. We also compare the loss on test data of training an end-to-end neural shape function versus training with geometry constraints embedded as prior knowledge (Table 1). From the result, we find that with the prior knowledge we proposed, the network requires less number of neurons while achieving better performance.

5.3. Simulation results

We apply the backward Euler scheme in our simulation algorithm by solving (7) with a time step $\Delta t = 0.03$ s. To compute the space integration, we select 8 Gaussian quadratures in each fine (or coarse) element for fine (or coarse) grids simulation. Thus, the integration points in fine grids are more than in coarse grids. In our experiments, the integration points in fine grids are eight times those in coarse grids.

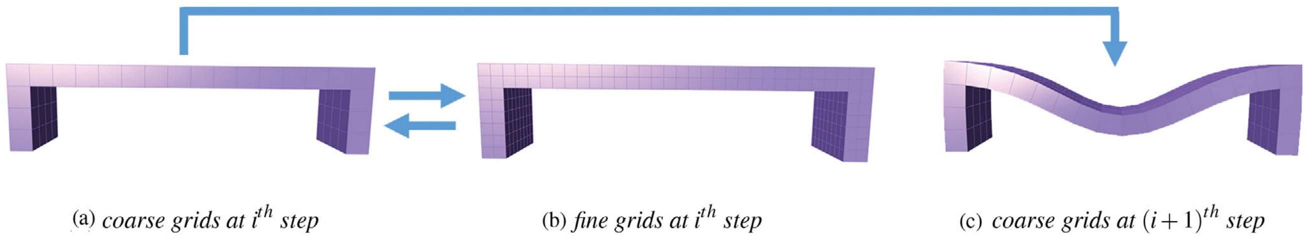

(a) coarse grids at i^{th} step
(b) fine grids at i^{th} step
(c) coarse grids at $(i+1)^{th}$ step

Figure 6: *Our simulation pipeline: first, we get the coarse grid nodes at i^{th} step in (a); then, we map the coarse nodes in (a) to the fine ones via our learned neural shape functions. In (b), we simultaneously compute the internal force of fine grids and its gradient for later simulation. Later, we map the force and its gradient on the fine grids back to that on the coarse grids (a). Finally, we solve the mechanical equation and get the deformed coarse nodes in (c) at $(i+1)^{th}$ step.*

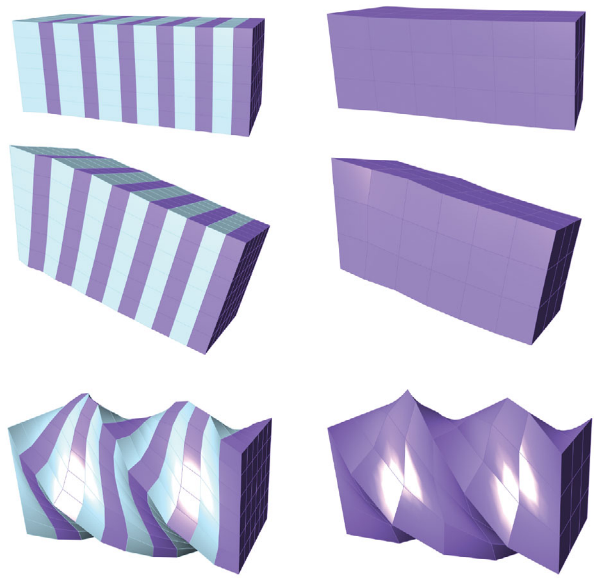

(a) *Exact solution* (b) *Our method*

Figure 7: *We apply our method to simulate the bar to confirm the feasibility with basis deformations (stretch, bending, and twist). The first column is the deformation computed on fine grids and the second column is the same simulation on coarse grids. Our method shows good accuracy when applied to these basic deformation modes.*

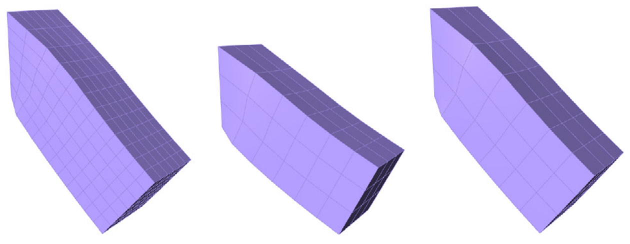

(a) *Exact solution* (b) *Coarse grids* (c) *Our method*

Figure 8: *We simulate the bending of a bar with a single material. It shows a large error when we apply the simulation on the coarse grids with regular trilinear shape functions.*

We use material with different Young's modulus for simulation. In the simulation result figures, deeper colour means a larger Young's modulus. In Figure 7, we test our method on the simplest two-material bar model with basic actions, respectively. In Figure 8, we simulate bar bend on coarse grids with regular trilinear shape functions and compare it with our method. It shows that our method can simulate more accurately. To show the generalization capability of our method, we then test the method on hand and Eiffel tower models whose parameters vary in space (Figure 9). We also examine large models in Figure 9 and Figure 10. The numerical results are shown in Table 3.

Table 2: *Time breakdown of hand (Figure 9) simulation of different resolutions for different solvers. The time in the table is counted for one simulation step.*

		Assembly time (s)		Solving time (s)	
Grid	# DoFs	w/o MLP	with MLP	Eigen	Pardiso
Fine	1.0×10^5	40	-	2409	10
Coarse	1.5×10^4	7.3	10.5	25.7	0.8
Fine	2.9×10^5	81.8	-	2×10^5	42.8
Coarse	4.2×10^4	31.8	40.1	402	2.8

5.4. Timing

We always expect to accelerate the simulation when we use numerical coarsening. We simulate different models and test the time consumed. We solve our simulation with a Newton-Rapson solver using the Cholesky factorization from the Eigen library. The speed-up of numerical coarsening compared to fine meshes is different for different models. For small models, such as the bar with 432 fine elements, the coarse grids perform 10 times faster than fine grids. For large models, such as the hand with 30K fine elements, the coarse grids perform 50 times faster than fine grids. Since the most time is spent solving the linear system, the speed of our method is almost the same as [CBW*18].

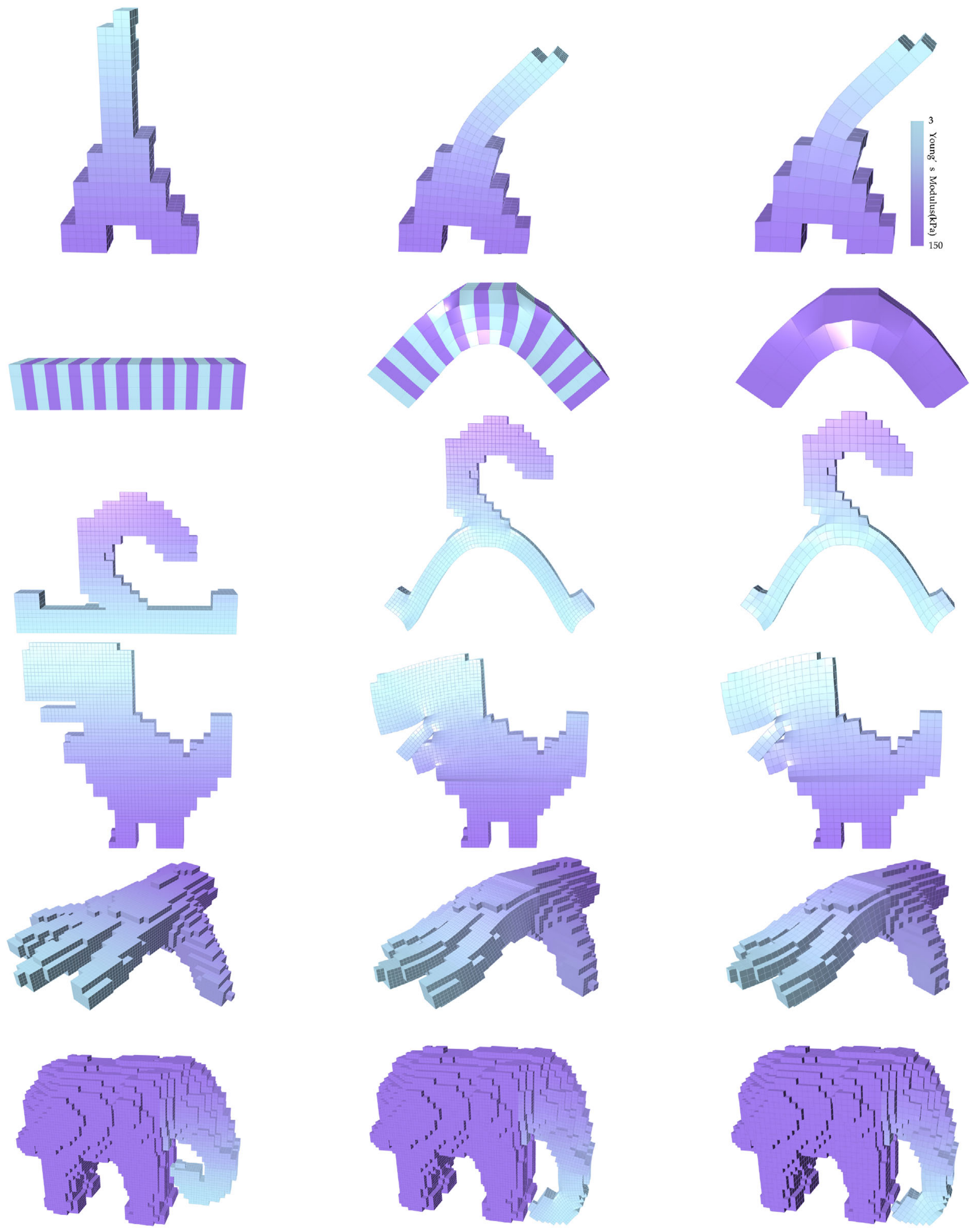
We also test the Pardiso solver. As shown in Table 2, the speed-up mostly comes from reducing the DoFs. Besides, the numerical coarsening reduces the iteration steps. With the MLP, the assembly time will increase, but it is slight. When we use the Pardiso solver instead of the Eigen, the speedup factor decreases as the solving time of the Pardiso is less than the Eigen. The complexity of matrix assembly is $O(N)$, and the complexity of solving linear equations with the Eigen or the Pardiso is larger than $O(N)$. When we simulate a larger model, the time for solving dominates the overall time, and the speedup factor becomes larger.

5.5. Comparison with [CBW*18]

They use linear mapping to compute the fine grid nodes with x^H. Instead, we use a quasi-linear function to represent the mapping, which has more DoFs and can approximate nonlinear deformation better. Besides, we can easily apply numerical coarsening to different models with the learned neural shape functions. For example, when we apply the method to the dynamic simulation of a bar (Figure 11), despite the same error at the first five steps of the simulation, our method outperforms [CBW*18] with a smaller error. More comparisons are shown in Figure 12 and Table 3.

6. Conclusion

We propose using nonlinear shape functions represented as neural networks in numerical coarsening to achieve generalization capability as well as good accuracy. We use MLP to represent the mapping from coarse to fine grid nodes to achieve more DoFs and generalization capability to nonlinear deformation. As the end-to-end representation of the mapping suffers from some limitations, we then



(a) Rest shape (b) Exact solution (c) Our method

Figure 9: *We apply our scheme to the models composed of materials with different Young's modulus (as shown in the colour bar encoding the values of the Young modules). Our method shows good generalization capability to different Young's moduli.*

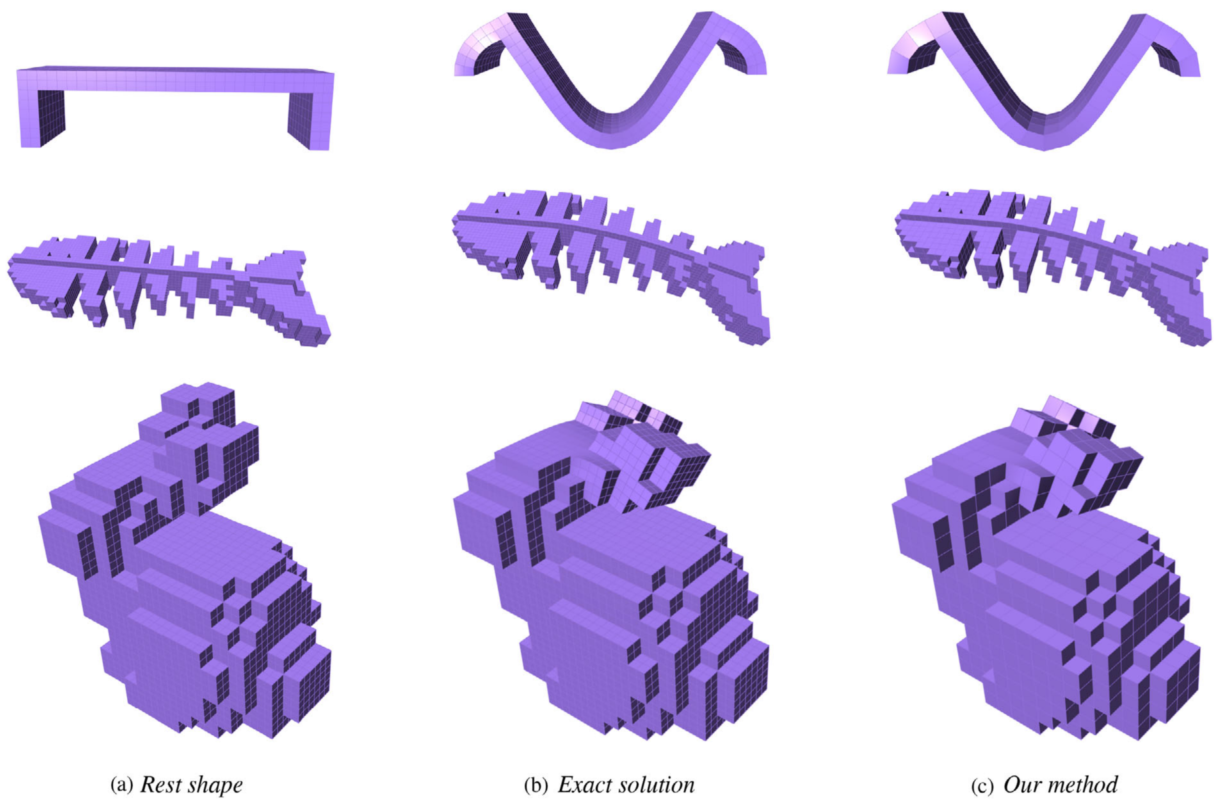


Figure 10: *We also tested our method on bridge, fish, and bunny models. We can see that the results of (c) look much similar to those of (b), demonstrating our method's good accuracy when applied to these complex models without other preprocessing steps.*

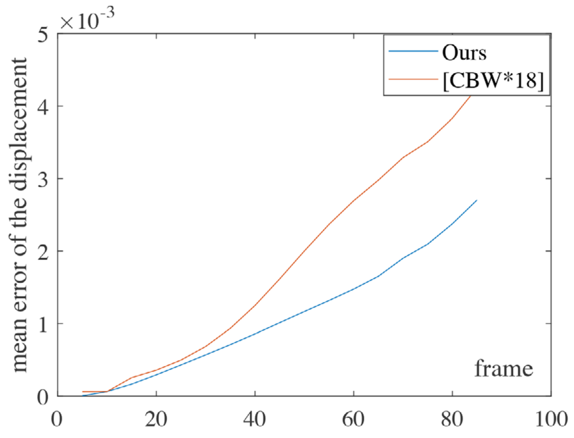


Figure 11: *We simulate the bend of a bar (in Figure 7) and compare our mean error of the displacement with [CBW*18]. Our method performs with less error during the dynamic deformation.*

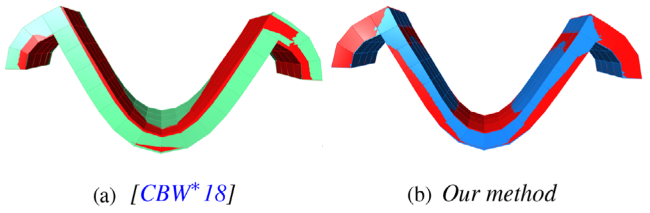


Figure 12: *We simulate the bend of a bridge. The red grid is the exact solution; the green grid is the method of [CBW*18], and the blue grid is our method. Our method shows better accuracy.*

embed the geometry constraints as the prior knowledge in training and formulate the learned neural shape function as a quasi-linear one.

There are some limitations of our work. First, our method is not suitable for extreme simulation configurations, such as where the difference in the material parameters between elements is too large or the deformation is extremely nonlinear. Actually, the ability of the learned mapping to capture different physical behaviours depends on the training data and the complexity of the neural network. Second, we do not apply our method to a multi-level coarsening. If we want to formulate a multi-level coarsening scheme, we have to retrain the network. Finally, we formulate the learned mapping as a quasi-linear function to capture the geometry constraints. However, there are some other intrinsic constraints of shape functions in practice not considered as the prior knowledge in training, for example, some variables of the mapping are actually symmetric, which is an interesting topic for future investigation.

Table 3: *We apply our scheme to several models and compare our results with [CBW*18]. We simulate these models for fixed numbers of frames and list the largest mean error of the deformation during the simulation in the table. Our learning-based coarsening method achieves higher accuracy than [CBW*18] with the same level of speed without the need to do case-by-case preprocessing.*

				Error		Time of one frame simulation(s)			
Model	# Ω^h	#node in Ω^h	#node in Ω^H	ours	[CBW*18]	ours	[CBW*18]	speedup	fine grids
Bird (Figure 1)	7944	10783	1750	3.0×10^{-3}	3.4×10^{-3}	3.9	2.9	16×	62.2
Bar (stretch) (Figure 7)	432	637	112	3.0×10^{-3}	2.1×10^{-2}	0.16	0.11	10×	1.7
Bar (bending) (Figure 7)	432	637	112	9.0×10^{-3}	1.5×10^{-2}				
Bar (twist) (Figure 7)	432	637	112	2.4×10^{-3}	2.5×10^{-2}				
Long bar (Figure 9)	320	525	99	5.2×10^{-3}	6.3×10^{-3}	0.98	0.09	5×	0.48
Eiffel (Figure 9)	1088	1761	324	7.6×10^{-4}	1.5×10^{-3}	0.4	0.3	4×	1.6
Hand (Figure 9)	29072	34685	5104	1.2×10^{-4}	1.8×10^{-4}	27.1	23.6	72×	1952.1
Bridge (Figure 10)	704	1215	230	2.0×10^{-3}	5.8×10^{-3}	0.14	0.1	8×	1.11
Fish (Figure 10)	10184	14389	2404	1.9×10^{-3}	2.1×10^{-3}	4.76	3.49	9×	43.59
Bunny (Figure 10)	8232	10135	1532	1.6×10^{-3}	1.7×10^{-3}	3.55	2.87	38×	135.51
Elephant (Figure 9)	89104	100609	14103	2.9×10^{-3}	3.4×10^{-3}	211	185	100×	2×10^{5}
Dinosaur (Figure 9)	10656	13569	2102	7.5×10^{-2}	9.1×10^{-2}	2.71	2.43	16×	38
Hanger (Figure 9)	3528	5217	901	6.8×10^{-2}	8.7×10^{-2}	0.81	0.75	6×	4.9

Acknowledgements

We would like to thank the anonymous reviewers for their constructive suggestions and comments. This work is supported by the National Natural Science Foundation of China (62025207, 62272429), the Major Project of Science and Technology of Anhui Province (No. 202203a05020050), and the National Key R&D Program of China (No. 2022YFA1005201/2022YFA1005200).

References

[CBO*19] Chen J., Budninskiy M., Owhadi H., Bao H., Huang J., Desbrun M.: Material-adapted refinable basis functions for elasticity simulation. *ACM Transactions on Graphics (TOG) 38*, 6 (2019), 1–15.

[CBW*18] Chen J., Bao H., Wang T., Desbrun M., Huang J.: Numerical coarsening using discontinuous shape functions. *ACM Transactions on Graphics (TOG) 37*, 4 (2018), 1–12.

[CLK*19] Chen Y. J., Levin D. I., Kaufmann D., Ascher U., Pai D. K.: Eigenfit for consistent elastodynamic simulation across mesh resolution. In *Proceedings of the 18th annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2019), pp. 1–13.

[CLMK17] Chen D., Levin D. I., Matusik W., Kaufman D. M.: Dynamics-aware numerical coarsening for fabrication design. *ACM Transactions on Graphics (TOG) 36*, 4 (2017), 1–15.

[CLSM15] Chen D., Levin D. I., Sueda S., Matusik W.: Data-driven finite elements for geometry and material design. *ACM Transactions on Graphics (TOG) 34*, 4 (2015), 1–10.

[CT17] Chu M., Thuerey N.: Data-driven synthesis of smoke flows with cnn-based feature descriptors. *ACM Transactions on Graphics (TOG) 36*, 4 (2017), 1–14.

[FMD*19] Fulton L., Modi V., Duvenaud D., Levin D. I., Jacobson A.: Latent-space dynamics for reduced deformable simulation. *Computer Graphics Forum 38* (2019), 379–391.

[HSO03] Hauser K. K., Shen C., O'Brien J. F.: Interactive deformation using modal analysis with constraints. *Graphics Interface 3* (2003), 16–17.

[KAT*19] Kim B., Azevedo V. C., Thuerey N., Kim T., Gross M., Solenthaler B.: Deep fluids: A generative network for parameterized fluid simulations. *Computer Graphics Forum 38* (2019), 59–70.

[KMOD09] Kharevych L., Mullen P., Owhadi H., Desbrun M.: Numerical coarsening of inhomogeneous elastic materials. *ACM Transactions on graphics (TOG) 28*, 3 (2009), 1–8.

[LCT18] Lahner Z., Cremers D., Tung T.: Deepwrinkles: Accurate and realistic clothing modeling. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 667–684.

[LSW*18] Luo R., Shao T., Wang H., Xu W., Chen X., Zhou K., Yang Y.: Nnwarp: Neural network-based nonlinear deformation. *IEEE Transactions on Visualization and Computer Graphics 26*, 4 (2018), 1745–1759.

[NKJF09] Nesme M., Kry P. G., Jeřábková L., Faure F.: Preserving topology and elasticity for embedded deformable models. In *ACM SIGGRAPH 2009 Papers*. 2009, pp. 1–9.

[RCPO21] Romero C., Casas D., Pérez J., Otaduy M.: Learning contact corrections for handle-based subspace dynamics. *ACM Transactions on Graphics (TOG) 40*, 4 (2021), 1–12.

[SYS*21] Shen S., Yin Y., Shao T., Wang H., Jiang C., Lan L., Zhou K.: High-order differentiable autoencoder for nonlinear model reduction. *arXiv preprint arXiv:2102.11026* (2021).

[TREO16] Torres R., Rodríguez A., Espadero J. M., Otaduy M. A.: High-resolution interaction with corotational coarsening models. *ACM Transactions on Graphics (TOG) 35*, 6 (2016), 1–11.

[VSB17] Vasile C.-I., Schwager M., Belta C.: Translational and rotational invariance in networked dynamical systems. *IEEE Transactions on Control of Network Systems 5*, 3 (2017), 822–832.

[XLCB15] Xu H., Li Y., Chen Y., Barbič J.: Interactive material design using model reduction. *ACM Transactions on Graphics (TOG) 34*, 2 (2015), 1–14.

Supporting Information

Additional supporting information may be found online in the Supporting Information section at the end of the article.

Video S1

© 2023 Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd.