Realtime Simulation of Thin-Shell Deformable Materials using CNN-Based Mesh Embedding

Qingyang Tan[1], Zherong Pan[2], Lin Gao[3], and Dinesh Manocha[1]

Video Link: `https://youtu.be/zuXoQYJeAfc`

Abstract— **We address the problem of accelerating thin-shell deformable object simulations by dimension reduction. We present a new algorithm to embed a high-dimensional configuration space of deformable objects in a low-dimensional feature space, where the configurations of objects and feature points have approximate one-to-one mapping. Our key technique is a graph-based convolutional neural network (CNN) defined on meshes with arbitrary topologies and a new mesh embedding approach based on physics-inspired loss term. We have applied our approach to accelerate high-resolution thin shell simulations corresponding to cloth-like materials, where the configuration space has tens of thousands of degrees of freedom. We show that our physics-inspired embedding approach leads to higher accuracy compared with prior mesh embedding methods. Finally, we show that the temporal evolution of the mesh in the feature space can also be learned using a recurrent neural network (RNN) leading to fully learnable physics simulators. After training our learned simulator runs $500-10000\times$ faster and the accuracy is high enough for robot manipulation tasks.**

I. Introduction

A key component in robot manipulation tasks is a dynamic model of target objects to be manipulated. Typical applications include cloth manipulation [27], [32], liquid manipulation [43], and in-hand rigid object manipulation [45]. Of these objects, cloth is unique in that it is modeled as a thin-shell, i.e., a 2D deformable object embedded in a 3D workspace. To model the dynamic behaviors of thin-shell deformable objects, people typically use high-resolution meshes (e.g. with thousands of vertices) to represent the deformable objects. Many techniques have been developed to derive a dynamic model under a mesh-based representation, including the finite-element method [31], the mass-spring system [6], [11], the thin-shell model [21], etc. However, the complexity of these techniques can vary from $\mathcal{O}(n^{1.5})$ to $\mathcal{O}(n^3)$ [20], where n is the number of DOFs, which makes them very computationally cost on high-resolution meshes. For example, [39] reported an average computational time of over 1 minute for predicting a single future state of a thin-shell mesh with around 5000 vertices. This simulation overhead is a major cost in various cloth manipulation algorithms including [27], [32], [30].

In order to reduce the computational cost, one recent trend is to develop machine learning methods to compute low-dimensional embeddings of these meshes. Low-dimensional embeddings were original developed for applications such as image compression [29] and dimension reduction [56]. The key idea is to find a low-dimensional feature space with approximate one-to-one mapping between a low-dimensional feature point and a high-dimensional mesh shape. So that the low-dimensional feature point can be treated as an efficient, surrogate representation of the original mesh.

However, computing low-dimensional embeddings for general meshes poses new challenges because, unlike 2D images, meshes are represented by a set of unstructured vertices connected by edges and these vertices can undergo large distortions when cloth deforms. As a result, a central problem in representing mesh deformation data is to find an effective parameterization of the feature space that can handle arbitrary mesh topologies and large, nonlinear deformations. Several methods for low-dimensional mesh embeddings are based on PCA [2], localized PCA [40], and Gaussian Process [55]. However, these methods are based on vertex-position features and cannot handle large deformations.

Main Results: We present a novel approach that uses physics-based constraints to improve the accuracy of low-dimensional embedding of arbitrary meshes for deformable simulation. We further present a fully learnable physics simulator of clothes in the feature space. The novel components of our algorithm include:

- A mesh embedding approach that takes into account the inertial and internal potential forces used by a physical simulator, which is achieved by introducing a physics-inspired loss function term, i.e., vertex-level physics-based loss term (PB-loss). This also preserves the material properties of the mesh.
- A stateful, recurrent feature-space physics simulator that predicts the temporal changes of meshes in the feature space, which are modeled by introducing accurate enough for learning cloth features and training cloth manipulation controllers (see Figure 5).

To test the accuracy of our method, we construct multiple datasets by running cloth simulations using a high-resolution mesh under different material models, material parameters, and mesh topologies. We show that our embedding approach leads to better accuracy in terms of physics rule preservation than prior method [47] that uses only a data term, with up to 70% improvement. We have also observed up to 19% and 18% improvements in mesh embedding accuracy on commonly used metrics such as $\mathcal{M}_{rms}$ and $\mathcal{M}_{STED}$. Finally, we show that our feature space physics simulator can robustly predict dynamic behaviors of clothes undergoing unseen robot manipulations, while achieving $500-10000\times$ speedup over simulators running in the high-dimensional configuration space.

The paper is organized as follows. We first review related work in Section II. We define our problem and introduce the basic method of low-dimensional mesh embedding in Section III. We introduce our novel PB-loss and the learnable simulation architecture in Section IV. Finally, we describe

[1]Qingyang Tan and Dinesh Manocha are with Department of Computer Science and Electrical & Computer Engineering, University of Maryland at College Park. {qytan,dm@cs.umd.edu} [2]Zherong Pan is with Department of Computer Science, University of North Carolina at Chapel Hill. {zherong@cs.unc.edu} [3]Lin Gao is with Institute of Computing Technology, Chinese Academy of Sciences. {gaolin@ict.ac.cn}

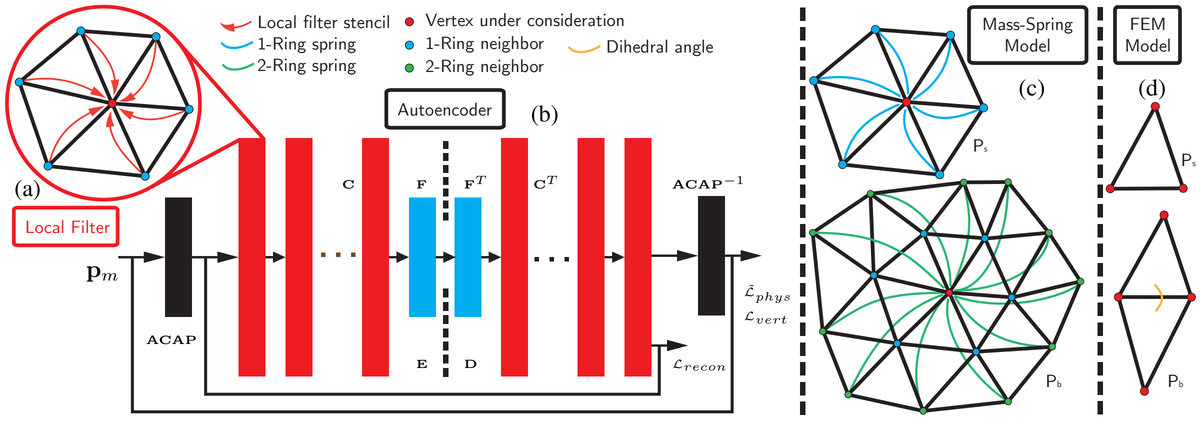

Fig. 1: Overview of our method: Each generated mesh $(\mathbf{p}_m)$ is represented as vertices connected by edges. (a): We use a graph-based CNN where each convolutional layer is a local filter and the filter stencil is the 1-ring neighbor (red arrow). (b): We build an autoencoder using the filter-based convolutional layers. The decoder $\mathbf{D}$ mirrors the encoder $\mathbf{E}$ and both $\mathbf{D}, \mathbf{E}$ use L convolutional layers and one fully connected layer. The input of $\mathbf{E}$ and the output of $\mathbf{D}$ are defined in the ACAP feature space, in which we define the reconstruction loss, $\mathcal{L}_{recon}$. We recover the vertex-level features, $\mathbf{p}_m$, using the function $\mathbf{ACAP}^{-1}$, on which we define our PB-loss, $\mathcal{L}_{phys}$, and vertex-level regularization, $\mathcal{L}_{vert}$. The PB-loss can be formulated using two methods. (c): In the mass-spring model, the stretch resistance term is modeled as springs between each vertex and its 1-ring neighbors (blue) and the bend resistance term is modeled as springs between each vertex and its 2-ring neighbors (green). (d): FEM models the stretch resistance term as the linear elastic energy on each triangle and the bend resistance term as a quadratic penalty on the dihedral angle between each pair of neighboring triangles (yellow).

the applications in Section V and highlight the results in Section VI.

II. Related Work and Background

We summarize related work in mesh deformations and representations, deformable object simulations, and machine learning methods for mesh deformations.

Deformable Simulation for Robotics are frequently encountered in service robots applications such as laundry cleaning [8], [30] and automatic cloth dressing [12]. Studying these objects can also benefit the design of soft robots [38], [16]. While these soft robots are usually 3D volumetric deformable objects, we focus on 2D shell-like deformable objects or clothes. In some applications such as visual servoing [26] and tracking [10], deformable objects are represented using point clouds. In other applications including model-based control [42] and reconstruction [49], the deformable objects are represented using meshes and their dynamics are modeled by discretizing the governing equations using the finite element method (FEM). Solving the discretized governing equation is a major bottleneck in training a cloth manipulation controller, e.g., [5] reported up to 5 hours of CPU time spend on thin-shell simulation which is 4-5 times more costly than the control algorithm.

Deformable Object Simulations is a key component in various model-based control algorithms such as virtual surgery [3], [4], [33] and soft robot controllers [42], [14], [27]. However, physics simulators based on the finite element method [31], the boundary-element method [9], or simplified models such as the mass-spring system [11] have a super-linear complexity. An analysis is given in [20], resulting in $\mathcal{O}(n^{1.5})$ complexity, where n is the number of DOFs. In a high-resolution simulation, n can be in the tens of thousands. As a result, learning-based methods have recently been used to accelerate physics simulations. This can be done by simulating under a low-resolution using FEM and then upsampling [52] or by learning the dynamics behaviors of clothes [41] and fluids [51]. However, these methods are either not based on meshes [51] or not able to handle arbitrary topologies [41].

Machine Learning Methods for Mesh Deformations has been in use for over two decades, of which most methods are essentially low-dimensional embedding techniques. Early work are based on principle component analysis (PCA) [2], [56], [40] that can only represent small, local deformations or Gaussian processes [50], [55] that are computationally costly to train and do not scale to large datasets. Recently, deep neural networks have been used to embed high-dimensional nonlinear functions [29], [44]. However, these methods rely on regular data structures such as 2D images. To handle meshes with arbitrary topologies, earlier methods [37] represent a mesh as a 3D voxelized grid or reconstruct 3D shapes from 2D images [53] using a projection layer. Recently, methods have been proposed to define CNN directly on mesh surfaces, such as CNN on parametrized texture space [36], and CNN based on spatial filtering [15]. The later has been used in [47] to embed large-scale deformations of general meshes. Our contribution is orthogonal to these techniques and can be used to improve the embedding accuracy for any one of these methods.

III. Low-Dimensional Mesh Embedding

In this section, we provide an overview of low-dimensional embedding of thin shell like meshes such as clothes. Our goal is to represent a set of N deformed meshes, S_m, with each mesh represented using a set of K vertices, denoted as $\mathbf{p}_m \in \mathbf{R}^{3K}$. We denote the ith vertex as $\mathbf{p}_{m,i} \in \mathbf{R}^3$. Here $m = 1, \cdots, N$ and $i = 1, \cdots, K$. These vertices are

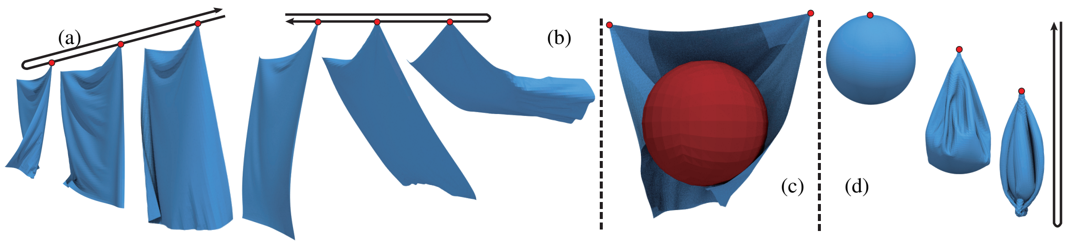

Fig. 2: A visualization of our two datasets.The SHEET dataset contains 4 simulation sequences, each with $N = 2400$ frames. (a,b): We generate the dataset by grasping two corners of the cloth (red dot) and moving the grasping points back and forth along the $\pm X/Y$ axes. (c): In two sequences of the SHEET dataset, we add a spherical obstacle to interact with the cloth. (d): The BALL dataset contains 6 simulation sequences, each with $N = 500$ frames. We generate the dataset by grasping the topmost vertex of the cloth ball (red dot) and moving the grasping point back and forth along the $\pm Z$ axes.

connected by edges, so we can define the 1-ring neighbor set, $\mathcal{N}_i^1$, and the 2-ring neighbor set, $\mathcal{N}_i^2$, for each $\mathbf{p}_i$, as shown in Figure 1 (c). Our goal is to find a map $\mathbf{z} \to \mathbf{p}$, where $\mathbf{z}$ is a low-dimensional feature and $\mathbf{p} \in \mathbf{R}^{3K}$ such that, for each m, there exists a $\mathbf{z}_m$ where $\mathbf{z}_m$ is mapped to a mesh close to $\mathbf{p}_m$. To define such a function, we use graph-based CNN and ACAP features [18] to represent large-scale deformations.

A. ACAP Feature

For each S_m, an ACAP feature is computed by first finding the deformation gradient $\mathbf{T}_{m,i}$ on each vertex:

$$\mathbf{T}_{m,i} \triangleq \underset{\mathbf{T}}{\mathbf{argmin}} \sum_{j\in\mathcal{N}_i^1} c_{ij}\|(\mathbf{p}_{m,i}-\mathbf{p}_{m,j})-\mathbf{T}(\mathbf{p}_{1,i}-\mathbf{p}_{1,j})\|^2, \quad (1)$$

where c_{ij} are cotangent weights [13]. Here, we use S_1 as a reference shape. Next, we perform polar decomposition to compute $\mathbf{T}_{m,i} = \mathbf{R}_{m,i}\mathbf{S}_{m,i}$ where $\mathbf{R}_{m,i}$ is orthogonal and $\mathbf{S}_{m,i}$ is symmetric. Finally, $\mathbf{R}_{m,i}$ is transformed into log-space in an as-consistent-as-possible manner using mixed-integer programming. The final ACAP feature is defined as $\mathbf{ACAP}_{m,i} \triangleq \{\mathbf{log}(\mathbf{R}_{m,i}), \mathbf{S}_{m,i}\} \in \mathbf{R}^9$ due to the symmetry of $\mathbf{S}_{m,i}$. We denote the ACAP feature transform as: $\mathbf{ACAP}(\mathbf{p}_m) \in \mathbf{R}^{9K}$. It is suggested, e.g., in [24], that mapping $\mathbf{z}_m$ to the ACAP feature space leads to better effectiveness in representing large-scale deformations. Therefore, we define our mapping function to be $\mathbf{D}(\mathbf{z}) : \mathbf{z} \to \mathbf{ACAP}(\mathbf{p})$ and then recover $\mathbf{p}$ via the inverse feature transform: $\mathbf{ACAP}^{-1}$.

B. Graph-Based CNN for Feature Embedding

The key idea in handling arbitrary mesh topologies is to define $\mathbf{D}$ as a graph-based CNN using local filters [15]:

$$\mathbf{D} \triangleq \mathbf{C}_L^T \cdots \mathbf{C}_2^T \circ \mathbf{C}_1^T \circ \mathbf{F}^T,$$

where L is the number of convolutional layers and $\mathbf{C}^T$ is the transpose of a graph-based convolutional operator. Finally, $\mathbf{F}$ is a fully connected layer. Each layer is appended by a leaky ReLU activation layer. A graph-based convolutional layer is a linear operator defined as:

$$\mathbf{C}(\mathbf{ACAP}(\mathbf{p}_m))_{m,i} \triangleq \mathbf{W} \times \mathbf{ACAP}_{m,i} + \mathbf{W}_{\mathcal{N}} \sum_{j\in\mathcal{N}_i^1} \mathbf{ACAP}_{m,j}/|\mathcal{N}_i^1| + \mathbf{b},$$

where $\mathbf{W}, \mathbf{W}_{\mathcal{N}}, \mathbf{b}$ are optimizable weights and biases, respectively. All the weights in the CNN are trained in a self-supervised manner using an autoencoder and the reconstruction loss:

$$\mathcal{L}_{recon} = \sum_{m=1}^{N} \|\mathbf{D} \circ \mathbf{E} \circ \mathbf{ACAP}(\mathbf{p}_m) - \mathbf{ACAP}(\mathbf{p}_m)\|^2/N,$$

where $\mathbf{E}$ is a mirrored encoder of $\mathbf{D}$ with a weight-tied architecture defined as:

$$\mathbf{E} \triangleq \mathbf{F} \circ \mathbf{C}_1 \cdots \mathbf{C}_{L-1} \circ \mathbf{C}_L,$$

which means that each layer in $\mathbf{E}$ is a transpose of the corresponding layer in $\mathbf{D}$ with shared weights. The construction of this CNN is illustrated in Figure 1 (a). In the next section, we extend this framework to make it aware of physics rules.

IV. Physics-Based Loss Term

We present a novel physics-inspired loss term that improves the accuracy of low-dimensional mesh embedding. Our goal is to combine physics-based constraints with graph-based CNNs, where our physics-based constraints take a general form and can be used with any material models such as FEM [39] and mass-spring system [11]. We assumes that S_m is generated using a physics simulator that solves a continuous-time PDE of the form:

$$\mathbf{M}\frac{\partial \mathbf{p}}{\partial t} = -\mathbf{force}(\mathbf{p}, \mathbf{q}), \quad (2)$$

where $\mathbf{M}$ is the mass matrix and t is the time. This form of governing equation is the basis for state-of-the-art thin shell simulators including [11], [39]. $\mathbf{force}(\mathbf{p}, \mathbf{q})$ models internal and external forces affecting the current mesh $\mathbf{p}$. The force is also a function of the current control parameters $\mathbf{q}$, which are the positions of the grasping points on the mesh (red dots of Figure 2). This continuous time PDE Equation 2 can be discretized into N timesteps such that S_m is the position of S at time instance $i\Delta t$, where Δt is the timestep size. A discrete physics simulator can determine all $\mathbf{p}_m$ given the initial condition $\mathbf{p}_1, \mathbf{p}_2$ and the sequence of control parameters $\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_N$ by the recurrent function:

$$\mathbf{p}_m \triangleq f(\mathbf{p}_{m-2}, \mathbf{p}_{m-1}, \mathbf{q}_m), \quad (3)$$

where f is a discretization of Equation 2. To define this discretization, we use a derivation of [34] that reformulates f as the following optimization:

$$\mathbf{p}_m \triangleq \underset{\mathbf{p}}{\mathbf{argmin}}\, \mathcal{L}_{phys}(\mathbf{p}_{m-2}, \mathbf{p}_{m-1}, \mathbf{p}, \mathbf{q}_m) \quad (4)$$

$$\mathcal{L}_{phys} \triangleq \|\mathbf{p} - 2\mathbf{p}_{m-1} + \mathbf{p}_{m-2}\|_{\mathbf{M}}^2/(2\Delta t^2) + \mathcal{P}(\mathbf{p}, \mathbf{q}_m).$$

Note that Equation 4 is just one possible implementation of Equation 3. Here the first term models the kinematic energy, which requires each vertex to move in its own velocity as much as possible if no external forces are exerted. The second term models forces caused by various potential energies at configuration $\mathbf{p}$. In this work, we consider three kinds of potential energy:

- Gravitational energy $\mathcal{P}_g(\mathbf{p}) \triangleq -\sum_{i=1}^{K} \mathbf{g}^T\mathbf{M}\mathbf{p}$, where $\mathbf{g}$ is the gravitational acceleration vector.
- Stretch resistance energy, $\mathcal{P}_s$, models the potential force induced by stretching the material.
- Bending resistance energy, $\mathcal{P}_b$, models the potential force induced by bending the material.

There are many ways to discretize $\mathcal{P}_s, \mathcal{P}_b$, such as the finite element method used in [39] or the mass-spring model used in [34], [11]. Both formulations are evaluated in this work.

- [11] models the stretch resistance term, $\mathcal{P}_s$, as a set of Hooke's springs between each vertex and vertices in its 1-ring neighbors. In addition, the bend resistance term, $\mathcal{P}_b$, is defined as another set of Hooke's springs between each vertex and vertices in its 2-ring neighbors. (Figure 1 (c))
- [39] models the stretch resistance term, $\mathcal{P}_s$, as a linear elastic energy resisting the in-plane deformations of each mesh triangle. In addition, the bend resistance term, $\mathcal{P}_b$, is defined as a quadratic penalty term resisting the change of the dihedral angle between any pair of two neighboring triangles. (Figure 1 (d))

Our approach uses Equation 4 as an additional loss function for training $\mathbf{D}, \mathbf{E}$. Since Equation 4 is used for data generation, using it for mesh deformation embedding should improve the accuracy of the embedded shapes. However, there are two inherent difficulties in using $\mathcal{P}$ as an loss function. First, $\mathcal{P}$ is defined on the vertex level as a function of $\mathbf{p}_m$, not on the feature level as a function of $\mathbf{ACAP}(\mathbf{p}_m)$. To address this issue, we use the inverse function $\mathbf{ACAP}^{-1}$ to reconstruct $\mathbf{p}_m$ from $\mathbf{ACAP}(\mathbf{p}_m)$. The implementation of $\mathbf{ACAP}^{-1}$ is introduced in Section IV-A. By combining $\mathbf{ACAP}^{-1}$ with $\mathcal{L}_{phys}$, we can train the mesh deformation embedding network using the following loss:

$$\begin{aligned}\tilde{\mathcal{L}}_{phys} &\triangleq \mathcal{L}_{phys}(\mathbf{ACAP}^{-1} \circ \mathbf{D}(\mathbf{z}_{m,m-1,m-2}), \mathbf{q}_m)\\ \mathcal{L}_{ephys} &= \tilde{\mathcal{L}}_{phys}(\mathbf{E} \circ \mathbf{ACAP}(\mathbf{p}_{m,m-1,m-2}), \mathbf{q}_m).\end{aligned} \quad (5)$$

Our second difficulty is that the embedding network is stateless and does not account for temporal information. In other words, function $\mathbf{E}$ only takes $\mathbf{p}_m$ as input, while Equation 4 requires $\mathbf{p}_m, \mathbf{p}_{m-1}, \mathbf{p}_{m-2}$. To address this issue, we use a small, fully connected, recurrent network to represent the physics simulation procedure in the feature space. The training of this stateful network is introduced in Section IV-B. Finally, in addition to the PB-loss, we also add an autoencoder reconstruction loss on the vertex level as a regularization:

$$\mathcal{L}_{vert} = \sum_{m=1}^{N} \|\mathbf{ACAP}^{-1} \circ \mathbf{D} \circ \mathbf{E} \circ \mathbf{ACAP}(\mathbf{p}_m) - \mathbf{p}_m\|^2/N.$$

This vertex level loss can be removed from our loss function without significantly affect the quality of results. However, $\mathcal{L}_{vert}$ provides complementary information to $\mathcal{L}_{recon}$. Since ACAP is feature in the gradient domain, using only $\mathcal{L}_{recon}$ will reconstruct accurate local geometric features, but can lead to large error in vertices' positions. Therefore, we combine $\mathcal{L}_{vert}$ and $\mathcal{L}_{recon}$ to reduce the gradient domain errors and absolute vertex position errors.

A. The Inverse of the ACAP Feature Extractor

The inverse of the $\mathbf{ACAP}$ function (black block in Figure 1) involves three steps. Fortunately, each step can be easily implemented in a modern neural network toolbox such as TensorFlow [1]. The first step computes $\mathbf{R}_{m,i}$ from $\mathbf{Log}(\mathbf{R}_{m,i})$ using the Rodrigues' rotation formula, which involves only basic mathematical functions such as dot-product, cross-product, and the cosine function. The second step computes $\mathbf{T}_{m,i}$ from $\mathbf{R}_{m,i}, \mathbf{S}_{m,i}$, which is a matrix-matrix product. The final step computes $\mathbf{p}_{m,i}$ from $\mathbf{T}_{m,i}$. According to Equation 1, this amounts to pre-multiplying the inverse of a fixed sparse matrix, $\mathbf{L}$, representing the Poisson reconstruction. However, this $\mathbf{L}$ is rank-3 deficient because it is invariant to rigid translation. Therefore, we choose to define a pseudo-inverse by fixing the position of the grasping points $\mathbf{q}$:

$$\mathbf{L}^{\dagger}\mathbf{p} \triangleq \begin{pmatrix} \mathbf{I} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{L} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{p} \\ \mathbf{q} \end{pmatrix}, \quad (6)$$

which can be pre-factorized. Here $\mathbf{A}^{3\times 3K}$ is a matrix selecting the grasping points.

B. Stateful Recurrent Neural Network

A physics simulation procedure is Markovian, i.e. current configuration $\begin{pmatrix} \mathbf{p}_{m-1} & \mathbf{p}_m \end{pmatrix}$ only depends on previous configuration $\begin{pmatrix} \mathbf{p}_{m-2} & \mathbf{p}_{m-1} \end{pmatrix}$ of the mesh. As a result, $\mathcal{L}_{phys}$ is a function of both $\mathbf{p}_{m-2}$, $\mathbf{p}_{m-1}$, and $\mathbf{p}_m$, which measures the violation of physical rules. However, our embedding network is stateless and only models $\mathbf{p}_m$. In order to learn the entire dynamic behavior, we augment the embedding network with a stateful, recurrent network represented as a multilayer perceptron (MLP). This MLP represents a physically correct simulation trajectory in the feature space and is also Markovian, denoted as:

$$\mathbf{MLP}(\mathbf{z}_{m-2}, \mathbf{z}_{m-1}, \mathbf{q}_m) = \mathbf{z}_m. \quad (7)$$

Here the additional control parameters $\mathbf{q}$ are given to $\mathbf{MLP}$ as additional information. We can build a simple reconstruction loss below to optimize $\mathbf{MLP}$:

$$\mathcal{L}_{sim} = \sum_{m=3}^{N} \|\mathbf{MLP}(\mathbf{z}_{m-2}, \mathbf{z}_{m-1}, \mathbf{q}_m) - \mathbf{z}_m\|^2/(N-2).$$

In addition, we can also add PB-loss to train this MLP, for which we define $\mathcal{L}_{mphys}$ on a sequence of N meshes by unrolling the recurrent network:

$$\begin{aligned}&\mathcal{L}_{mphys} = \frac{1}{N-2} * \quad (8)\\ &(\tilde{\mathcal{L}}_{phys}(\mathbf{z}_1, \mathbf{z}_2, \mathbf{MLP}(\mathbf{z}_1, \mathbf{z}_2, \mathbf{q}_3), \mathbf{q}_3) +\\ &\tilde{\mathcal{L}}_{phys}(\mathbf{z}_2, \mathbf{z}_3, \mathbf{MLP}(\mathbf{z}_2, \mathbf{z}_3, \mathbf{q}_4), \mathbf{q}_4) + \cdots +\\ &\tilde{\mathcal{L}}_{phys}(\mathbf{z}_{N-2}, \mathbf{z}_{N-1}, \mathbf{MLP}(\mathbf{z}_{N-2}, \mathbf{z}_{N-1}, \mathbf{q}_N), \mathbf{q}_N)).\end{aligned}$$

However, we argue that Equation 8 will lead to a physically incorrect result and cannot be directly used for training. To see this, we note that Equation 4 is the variational form of

Equation 2. So that $\mathbf{p}_m$ is physically correct when $\mathcal{L}_{phys}$ is at its local minima, i.e. the following partial derivative vanishes:

$$\frac{\partial \mathcal{L}_{phys}(\mathbf{p}_{m-2}, \mathbf{p}_{m-1}, \mathbf{p}_m, \mathbf{q}_m)}{\partial \mathbf{p}_m} = 0 \quad \forall m. \tag{9}$$

However, if we sum up $\mathcal{L}_{phys}$ over a sequence of N meshes and require the summed-up loss to be at a local minimum, as is done in Equation 8, then we are essentially requiring the following derivatives to vanish:

$$\frac{\partial \mathcal{L}_{phys}(\mathbf{p}_{m-2}, \mathbf{p}_{m-1}, \mathbf{p}_m, \mathbf{q}_m)}{\partial \mathbf{p}_m} + \frac{\partial \mathcal{L}_{phys}(\mathbf{p}_{m-1}, \mathbf{p}_m, \mathbf{p}_{m+1}, \mathbf{q}_m)}{\partial \mathbf{p}_m} + \frac{\partial \mathcal{L}_{phys}(\mathbf{p}_m, \mathbf{p}_{m+1}, \mathbf{p}_{m+2}, \mathbf{q}_m)}{\partial \mathbf{p}_m} = 0 \quad \forall m. \tag{10}$$

The difference between Equation 9 and Equation 10 is the reason that Equation 8 gives an incorrect result. To resolve the problem, we slightly modify the back propagation procedure of our training process by setting the partial derivatives of $\mathcal{L}_{phys}$ with respect to its first two parameters to zero:

$$\frac{\partial \mathcal{L}_{phys}(\mathbf{p}_{m-2}, \mathbf{p}_{m-1}, \mathbf{p}_m, \mathbf{q}_m)}{\partial [\mathbf{p}_{m-1}, \mathbf{p}_{m-2}]} = 0 \quad \forall m,$$

which, combined with Equation 10, leads to Equation 9. (We add similar gradient constraints when optimizing over Equation 5.) This procedure is equivalent to an alternating optimization procedure, where we first compute a sequence of feature space coordinates, $\mathbf{z}_m$, using the recurrent network (Equation 7) and then fix the first two parameters $\mathbf{z}_{m-2}, \mathbf{z}_{m-1}$ and optimize $\mathcal{L}_{mphys}$ with respect to its third parameter $\mathbf{z}_m$.

V. Applications

The two novel components in our method, the $\mathbf{ACAP}^{-1}$ operator and the stateful PB-loss, enable a row of new applications, including realtime cloth inverse kinematics and feature space physics simulations.

A. Cloth Inverse Kinematics

Our first application allows a robot to grasp several points of a piece of cloth and then infer the full kinematic configuration of the cloth. Such inverse kinematics can be achieved by minimizing a high-dimensional nonlinear potential energy, such as ARAP energy [46], which is computationally costly. Using the inverse of the ACAP feature extractor, our method allows vertex-level constraints. Therefore, we can perform solve for the cloth configuration by a fast, low-dimensional minimization in the feature space as:

$$\mathbf{z}^* = \underset{\mathbf{z}}{\mathbf{argmin}}\, \mathcal{L} \circ \mathbf{ACAP}^{-1} \circ \mathbf{D}(\mathbf{z}),$$

where we treat all the grasped vertices as control parameters $\mathbf{q}$ used in Equation 6. This application is stateless and the user controls a single feature of a mesh, $\mathbf{z}$, so that we drop the kinetic term in $\mathcal{L}_{phys}$ and only retain the potential term $\mathcal{P}_s + \mathcal{P}_b$. Some inverse kinematic examples generated using this formulation are shown in Figure 3. Note that detailed wrinkles and cloth-like deformations are synthesized in unconstrained parts of the meshes.

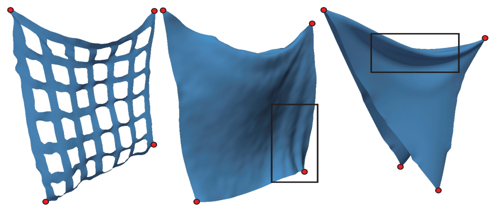
Fig. 3: Three examples of cloth inverse kinematics with fixed vertices marked in red. Note that our method can synthesize detailed wrinkles and cloth-like deformations in unconstrained parts of the meshes (black box).

B. Feature Space Physics Simulation

For our second application, we approximate an entire cloth simulation sequence (Equation 3) in the 128-dimensional feature space. Starting from $\mathbf{z}_1, \mathbf{z}_2$, we can generate an entire sequence of N frames by using the recurrent relationship in Equation 7 and can recover the meshes via the function $\mathbf{ACAP}^{-1} \circ \mathbf{D}$. Such a latent space physics model has been previously proposed in [51] for voxelized grids, while our model works on surface meshes. We show two synthesized simulation sequences in Figure 4.

C. Accuracy of Learned Simulator for Robotic Cloth Manipulation

We show three benchmarks (Figure 5) from robot cloth manipulation tasks defined in prior work [27]. In these benchmarks, the robot is collaborating with human to maintain a target shape of a piece of cloth. To design such a collaborating robot controller, we use imitation learning by teaching the robot to recognized cloth shapes under various, uncertain human movements. Our learnable simulator can be used to efficiently generate these cloth shapes for training the controller. To this end, we train our neural-network using the original dataset from [27] obtained by running the FEM-based simulator [39], which takes 3 hours. During test time, we perturb the human hands' grasp points along randomly directions. Our learned physical model can faithfully predict the dynamic movements of the cloth.

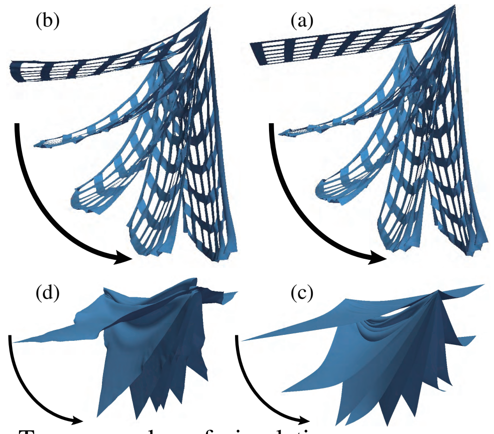

Fig. 4: Two examples of simulation sequence generation in our feature space. (a): 5 frames in the simulation of a cloth swinging down. (b): Synthesized simulation sequence. (c): Another example where two diagonal points are grasped. (d): Synthesized simulation sequence.

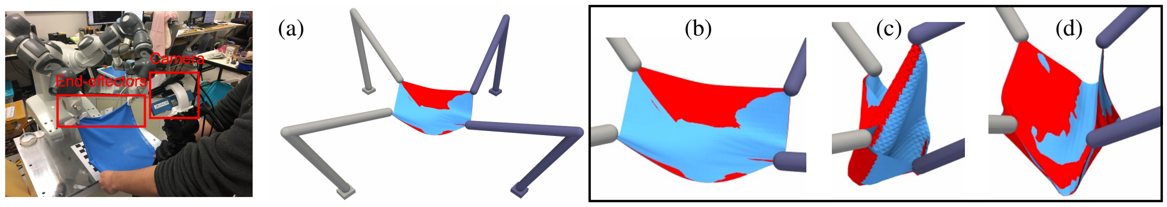

Fig. 5: We reproduce benchmarks from [27] where the robot is collaborating with human to manipulate a piece of cloth (a). We randomly perturb two grasp points on the left (gray arms) and the robot is controlling the other two grasp points (purple arms) using a visual-serving method to maintain the cloth at a target state, e.g., keeping the cloth flat (b), twisted (c), or bent (d). The red cloth is the groundtruth acquired by running the accurate FEM-based cloth simulator [39], which takes 3 hours. The difference between our result (blue) and the groundtruth is indistinguishable.

VI. Results

To evaluate our method, we create two datasets of cloth simulations using Equation 4. Our first dataset is called SHEET, which contains animations of a square-shaped cloth sheet swinging down under different conditions, as shown in Figure 2 (a). This dataset involves 6 simulation sequences, each with $N = 2400$ frames. Among these 6 sequences, the first sequence uses the mass-spring model [11] to discretize Equation 2 and the cloth mesh has no holes (denoted as SHEET+[11]). The second and the third simulation sequences in the dataset use different material parameter, by multiplying the stretch/bend resistance term by 0.1 and thereby making the material softer and less resilient when stretched or bent. These two sequences are denoted as SHEET+[11]+$0.1\mathcal{P}_s$ and SHEET+[11]+$0.1\mathcal{P}_b$, respectively. The forth sequence uses the mass spring model and the cloth mesh has holes, as shown in Figure 4 (a,b), which is denoted as (SHEET+[11]+holes). The fifth sequence uses FEM [39] to discretize Equation 2 and the cloth mesh has no holes (denoted as SHEET+[39]). The sixth sequence uses FEM to discretize Equation 2 and the cloth interacts with an obstacle, as shown in Figure 2 (c) (denoted as SHEET+[39]+obstacle). In the SHEET dataset, the cloth mesh without holes has $K = 4225$ vertices and the cloth mesh with holes has $K = 4165$ vertices. Our second dataset is called BALL, which contains animations of a cloth ball being dragged up and down under different conditions, as shown in Figure 2 (d). This dataset also involves 4 simulation sequences, each with $N = 500$ frames. Using the same notation as the SHEET dataset, the 4 sequences in the BALL dataset are (BALL+[11], BALL+[39], BALL+[39]+$0.1\mathcal{P}_s$, BALL+[39]+$0.1\mathcal{P}_b$). In the BALL dataset, the cloth ball mesh has $K = 1538$ vertices. During comparison, for each dataset, we select first 12 frames in every 17 frames to form the training set. The other frames are used as the test set.

A. Implementation

We implement our method using Tensorflow [1] and we implement the PB-loss as a special network layer. When there is an obstacle interacting with the cloth, we model the collision between the cloth and the obstacle using a special potential term proposed in [19]. For better conditioning and a more robust initial guess, our training procedure is broken into three stages. During the first stage, we use the loss:

$$\mathcal{L}_1 = \sum \lambda_i \mathcal{L}_i \quad i \in \{recon, vert, ephys\}$$

to optimize $\mathbf{E}, \mathbf{D}$. During the second stage, we use the loss:

$$\mathcal{L}_2 = \sum \lambda_i \mathcal{L}_i \quad i \in \{sim, mphys\}$$

to optimize $\mathbf{MLP}$. Finally, we add a fine-tuning step and use the loss:

$$\mathcal{L}_3 = \sum \lambda'_i \mathcal{L}_i \quad i \in \{recon, vert, ephys, sim\}$$

to optimize both $\mathbf{E}, \mathbf{D}$ and $\mathbf{MLP}$. Notice that, in order to train the mesh embedding network and $\mathbf{MLP}$ at the same time, we feed:

$$\mathbf{z}' = 0.5 * \mathbf{z}_m + 0.5 * \mathbf{MLP}(\mathbf{z}_{m-2}, \mathbf{z}_{m-1}, \mathbf{q}_m)$$

to $\mathbf{D}$ for better stability during the third stage.

B. Physics Correctness of Low-Dimensional Embedding

We first compare the quality of mesh deformation embeddings using two different methods. The quality of embedding is measured using three metrics. The first metric is the root mean square error, $\mathcal{M}_{rms}$ [28], which measures the averaged vertex-level error over all N shapes and K vertices. Our second metric is the STED metric, $\mathcal{M}_{STED}$ [48]. This metric linearly combines several aspects of errors crucial to visual quality, including relative edge length changes and temporal smoothness. However, since $\mathcal{M}_{STED}$ is only meaningful for consecutive frames, we compute $\mathcal{M}_{STED}$ for the 5 consecutive frames in every 17 frames, which is the test set. Finally, we introduce a third metric, physics correctness, which measures how well the physics rule is preserved. Inspired by Equation 9, physics correctness is measured by the norm of partial derivatives of $\mathcal{L}_{phys}$: $\mathcal{M}_{phys} = \|\partial \mathcal{L}_{phys}/\partial \mathbf{p}_m\|^2$. Note that the absolute value of $\mathcal{M}_{phys}$ can vary case by case. For example, $\mathcal{M}_{phys}$ using the FEM method can be orders of magnitude larger than that using the mass-spring system in our dataset. So that only the relative value of $\mathcal{M}_{phys}$ indicates improvement in physics correctness.

Our first experiment compares the accuracy of mesh embedding with or without PB-loss. The version without PB-loss is our baseline, which is equivalent to adding vertex level loss to [47]. In addition, we remove the sparsity regularization from [47] to make it consistent with our formulation. We denote this baseline as [47]+$\mathcal{L}_{vert}$. A complete summary of our experimental results is given in Table I. The benefit of three-stage training is given in Table IV. From Table I, we can see that including PB-loss significantly and consistently improves $\mathcal{M}_{phys}$. This improvement is large, up to 70% on the SHEET+[39] dataset. In addition, by adding

Dataset	Method	$\mathcal{M}_{rms}$	$\mathcal{M}_{STED}$	$\mathcal{M}_{phys}$
SHEET+[11]	ours	**9.724**	**0.01556**	**14581.46**
	[47]+$\mathcal{L}_{vert}$	10.351	0.01688	15446.70
	[47]	11.841	0.01760	16198.02
SHEET+[11] +0.1$\mathcal{P}_s$	ours	**10.477**	**0.01882**	**1996.01**
	[47]+$\mathcal{L}_{vert}$	11.341	0.01957	2128.05
SHEET+[11] +0.1$\mathcal{P}_b$	ours	**9.580**	**0.01675**	**10189.24**
	[47]+$\mathcal{L}_{vert}$	11.319	0.01773	10260.39
SHEET+[11] +holes	ours	**12.456**	**0.02443**	**19548.58**
	[47]+$\mathcal{L}_{vert}$	12.928	0.02480	20632.13
SHEET+[39]	ours	**10.671**	0.01398	**29109.03**
	[47]+$\mathcal{L}_{vert}$	10.675	**0.01195**	103878.20

Dataset	Method	$\mathcal{M}_{rms}$	$\mathcal{M}_{STED}$	$\mathcal{M}_{phys}$
SHEET+[39] +obstacle	ours	**8.075**	**0.01784**	160545.91
	[47]+$\mathcal{L}_{vert}$	8.425	0.01885	**155117.28**
BALL+[11]	ours	**17.008**	**0.02570**	**338465.20**
	[47]+$\mathcal{L}_{vert}$	20.326	0.03018	459564.89
	[47]	22.798	0.03622	476210.46
BALL+[39]	ours	**11.663**	**0.01501**	**49392520.44**
	[47]+$\mathcal{L}_{vert}$	12.200	0.01662	61048220.56
BALL+[39] +0.1$\mathcal{P}_s$	ours	**16.853**	0.03394	**80389.68**
	[47]+$\mathcal{L}_{vert}$	17.706	**0.03357**	84508.12
BALL+[39] +0.1$\mathcal{P}_b$	ours	**23.766**	**0.04320**	**674115.57**
	[47]+$\mathcal{L}_{vert}$	27.390	0.05320	974481.48

TABLE I: We compare the embedding quality using our method ($\lambda_{recon} = 1, \lambda_{vert} = 1, \lambda_{ephys} = 0.1\ to\ 1$, where λ_{ephys} for our method is tuned for different datasets.) and [47]+$\mathcal{L}_{vert}$ ($\lambda_{recon} = \lambda_{vert} = 1, \lambda_{ephys} = 0$). We also compare results trained using dataset with different cloth material properties ($0.1\mathcal{P}_s$ means that the stretch resilience has $1/10$ of the original material value and $0.1\mathcal{P}_b$ means ten times less bending resilience). From left to right: name of dataset, method used, $\mathcal{M}_{rms}$, $\mathcal{M}_{STED}$, and $\mathcal{M}_{phys}$.

$\mathcal{L}_{ephys}$, our method also better recognizes the relationship between each model and embeds them, thus improves $\mathcal{M}_{rms}$ in all the cases. However, our method sometimes sacrifice $\mathcal{M}_{STED}$ as temporal smoothness is not modeled explicitly in our method. Finally, we have added two rows to Table I, comparing our method with and without $\mathcal{L}_{vert}$, which shows that $\mathcal{L}_{vert}$ effectively reduces the error in terms of absolute vertex positions.

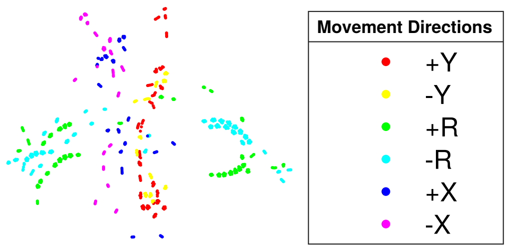

Fig. 6: A feature space visualization for SHEET+[11] using t-SNE.

C. Discriminability of Feature Space

In our second experiment, we evaluate the discriminability of mesh embedding by classifying the meshes using their feature space coordinates. Note that our datasets (Figure 2) are generated by moving the grasping points back and forth. We use these movement directions as the labels for classification. For the SHEET dataset, we have 6 labels: $\pm X/Y, \pm R$, where $\pm R$ means rotating the grasping points around $\pm Z$ axes. For the BALL dataset, we have 2 labels: $\pm Z$. Note that it is trivial to classify the meshes if we know the velocity of the grasping points. However, this information is missing in our feature space coordinates because ACAP features are invariant to global rigid translation, which makes the classification challenging. We visualize the feature space using t-SNE [35] compressed to 2 dimensions in Figure 6. We report retrieval performance in the KNN neighborhoods across different K's, using method suggested by [54]. The normalized discounted cumulative gain (DCG) on the test set for SHEET+[11] is 0.8045 and for BALL+[11] is 0.9128.

Method	(1, 1, 0.1)	(1, 1, 0.5)	(1, 1, 1)	[47]+$\mathcal{L}_{vert}$	[40]	[25]
$\mathcal{M}_{STED}$	0.01712	**0.01635**	**0.01556**	0.01688	0.03728	0.02381
$\mathcal{M}_{phys}$	**15164.34**	**14230.44**	**14581.46**	15446.70	36038.51	30732.07

TABLE II: We compare the performance of our method with several previous ones in terms of $\mathcal{M}_{STED}$ and $\mathcal{M}_{phys}$ under different weights (λ_{recon}, λ_{vert}, λ_{ephys}) of $\mathcal{L}$. The experiment is done on the dataset SHEET+[11]. Our method outperforms [47]+$\mathcal{L}_{vert}$ over a wide range of parameters. Previous methods, including [40] and [25], generate even worse results, which supports our choice of using convolutional neural network and ACAP feature for mesh deformation embedding.

D. Sensitivity to Training Parameters

In our third experiment, we evaluate the sensitivity of our method with respect to the weights of loss terms, as summarized in Table II. Our method outperforms [47]+$\mathcal{L}_{vert}$ under a range of different parameters. We have also compared our method with other baselines such as [40] and [25]. As shown in the last two columns of Table II, they generate even worse result, which indicates that [47]+$\mathcal{L}_{vert}$ is the best baseline.

E. Robustness to Mesh Resolutions

In this experiment, we highlight the robustness of our method to different mesh resolutions by lowering the resolution of our dataset. For SHEET+[11], we create a mid-resolution counterpart with $K = 1089$ vertices and a low-resolution counterpart with $K = 289$ vertices. On these two new datasets, we compare the accuracy of mesh embedding with or without PB-loss. The results are given in Table III. Including PB-loss consistently improves $\mathcal{M}_{phys}$ and overall embedding quality, no matter the resolution used.

F. PB-Loss with Alternative Neural Network Architectures

Our PB-loss is orthogonal to the architecture of neural networks. Therefore, we have conducted an additional set of experiments to highlight the performance improvement using a full-connected underlying neural network like [17]. The results are shown in Table V.

G. Difficulty in Contact Handling

One exception appears in the SHEET+[39]+obstacle (blue row in Table I), where our method deteriorates physics

Dataset	#Vertices	Method	$\mathcal{M}_{rms}$	$\mathcal{M}_{STED}$	$\mathcal{M}_{phys}$
SHEET+[11]	4225	ours	$\mathbf{9.724}$	$\mathbf{0.01556}$	$\mathbf{14581.46}$
	4225	[47]+$\mathcal{L}_{vert}$	10.351	0.01688	15446.70
	1089	ours	$\mathbf{11.744}$	$\mathbf{0.01511}$	$\mathbf{15712.89}$
	1089	[47]+$\mathcal{L}_{vert}$	11.842	0.1648	16076.44
	289	ours	$\mathbf{11.757}$	$\mathbf{0.01273}$	$\mathbf{10565.27}$
	289	[47]+$\mathcal{L}_{vert}$	12.510	0.01543	15498.58

TABLE III: We profile the improvement in various metrics under different mesh resolution ($\lambda_{recon} = 1, \lambda_{vert} = 1, \lambda_{ephys} = 0.5\ to\ 1$), compared with [47]+$\mathcal{L}_{vert}$. From left to right: name of dataset, number of vertices, method used, $\mathcal{M}_{rms}$, $\mathcal{M}_{STED}$, and $\mathcal{M}_{phys}$. Our method consistently outperforms [47]+$\mathcal{L}_{vert}$.

Dataset	Method	$\mathcal{M}_{rms}$	$\mathcal{M}_{STED}$	$\mathcal{M}_{phys}$
SHEET+[11]	baseline	15.184	0.01765	21019.28
	2nd stage	14.910	0.01705	18103.66
	3rd stage	$\mathbf{14.000}$	$\mathbf{0.01672}$	$\mathbf{17819.85}$
SHEET+[11]+holes	baseline	18.843	0.02703	29673.30
	2nd stage	17.909	0.02667	28526.89
	3rd stage	$\mathbf{17.412}$	$\mathbf{0.02633}$	$\mathbf{28393.81}$

TABLE IV: We compare the physical simulation performance of **MLP** after training with $\lambda_{mphys} = 0$ (baseline), training with $\lambda_{mphys} = 0.5\ to\ 1$ (2nd stage), and fine-tuning (3rd stage). For the 5 consecutive meshes in every 17 frames (the test set), we give **MLP** the first 2 frames and predict the remaining 3 frames to generate this table.

Dataset	Method	$\mathcal{M}_{rms}$	$\mathcal{M}_{STED}$	$\mathcal{M}_{phys}$
SHEET+[11]	FC+$\mathcal{L}_{vert}$+$\mathcal{L}_{ephys}$	$\mathbf{22.700}$	$\mathbf{0.01742}$	$\mathbf{16565.55}$
	FC+$\mathcal{L}_{vert}$	24.020	0.02043	19628.23
SHEET+[11] +holes	FC+$\mathcal{L}_{vert}$+$\mathcal{L}_{ephys}$	$\mathbf{18.513}$	$\mathbf{0.02504}$	$\mathbf{21852.67}$
	FC+$\mathcal{L}_{vert}$	19.253	0.02684	22748.91

TABLE V: We use a fully connected underlying neural network like [17] and train with or without our PB-loss. The profiled results show that our method can improve performance in terms of $\mathcal{M}_{rms}$, $\mathcal{M}_{STED}$, $\mathcal{M}_{phys}$, which is independent of the type of neural network architectures.

Dataset	#Vertices	NN(s)	FEM(s)
SHEET+[11]	4225	3.259	9094.906
SHEET+[11]+holes	4166	3.432	10336.001
Figure 5 (b)	1089	1.697	582.635
Figure 5 (c)	1089	1.701	701.971
Figure 5 (d)	1089	1.883	893.197

TABLE VI: We compare the computational time to generate a sequence of cloth data using our stateful neural network and the conventional FEM-based method.

correctness. This is the only dataset where the mesh is interacting with an obstacle. The deterioration is due to the additional loss term penalizing the penetration between the mesh and the obstacle. This term is non-smooth and has very high value and gradient when the mesh is in penetration, making the training procedure unstable. This means that direct learning a feature mapping for meshes with contacts and collisions can become unstable. However, we can solve this problem using a two-stage method, where we first learn a feature mapping for meshes without contacts and collisions, and then handle contacts and collisions at runtime using conventional method [22], as is done in [7].

H. Speedup over FEM-Based Cloth Simulators

In Table VI, we have used our neural network as a physics simulator and compared its performance with a conventional FEM-based method. Our method achieves $500 - 10000\times$ speedup over the FEM-based method [39] on average.

VII. Conclusion & Limitations

In this paper, we present a new method that bridges the gap between mesh embedding and and physical simulation for efficient dynamic models of clothes. We achieve low-dimensional mesh embedding using a stateless, graph-based CNN that can handle arbitrary mesh topologies. To make the method aware of physics rules, we augment the embedding network with a stateful feature space simulator represented as a MLP. The learnable simulator is trained to minimize a physics-inspired loss term (PB-loss). This loss term is formulated on the vertex level and the transformation from the ACAP feature level to the vertex level is achieved using the inverse of the ACAP feature extractor.

Our method can be used for several applications, including fast inverse kinematics of clothes and realtime feature space physics simulation. We have evaluated the accuracy and robustness of our method on two datasets of physics simulations with different material properties, mesh topologies, and collision configurations. Compared with previous models for embedding, our method achieves consistently better accuracy in terms of physics correctness and the mesh change smoothness metric ([48]).

A future research direction is to apply our method to other kinds of deformable objects, i.e., volumetric objects [23]. Each and every step of our method can be trivially extended to handle volumetric objects by replacing the triangle surface mesh with a tetrahedral volume mesh. A minor limitation of the current method is that the stateful MLP and the stateless mesh embedding cannot be trained in a fully end-to-end fashion. We would like to explore new optimization methods to train the two networks in an end-to-end fashion while achieving good convergence behavior. Finally, our approach is limited to a single setting of thin-shell simulation and needs to be re-trained whenever there are changes in the resolution of the mesh, the material parameters, or the obstacles in the environment.

VIII. Acknowledgement

This work was supported by ARO grants (W911NF1810313 and W911NF1910315) and Intel. Lin Gao was partially supported by the National Natural Science Foundation of China (No. 61872440) and Beijing Municipal Natural Science Foundation (No. L182016).

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *USENIX OSDI*, 2016, pp. 265–283.
[2] M. Alexa and W. Müller, "Representing animations by principal components," *Comp. Graph. Forum*, vol. 19, no. 3, pp. 411–418, 2000.
[3] R. Alterovitz, M. Branicky, and K. Goldberg, "Motion planning under uncertainty for image-guided medical needle steering," *The International journal of robotics research*, vol. 27, no. 11-12, pp. 1361–1374, 2008.
[4] R. Alterovitz, K. Y. Goldberg, J. Pouliot, and I.-C. Hsu, "Sensorless motion planning for medical needle insertion in deformable tissues," *IEEE Transactions on Information Technology in Biomedicine*, vol. 13, no. 2, pp. 217–225, 2009.

[5] Y. Bai, W. Yu, and C. K. Liu, “Dexterous manipulation of cloth,” in *Comp. Graph. Forum*, vol. 35, no. 2. Wiley Online Library, 2016, pp. 523–532.
[6] D. Baraff and A. Witkin, “Large steps in cloth simulation,” in *SIGGRAPH*. ACM, 1998, pp. 43–54.
[7] J. Barbič and D. L. James, “Subspace self-collision culling,” *ACM Trans. Graph.*, vol. 29, no. 4, pp. 81:1–81:9, 2010.
[8] C. Bersch, B. Pitzer, and S. Kammel, “Bimanual robotic cloth manipulation for laundry folding,” in *IEEE/RSJ IROS*, Sep. 2011, pp. 1413–1419.
[9] C. A. Brebbia and M. H. Aliabadi, Eds., *Industrial Applications of the Boundary Element Method*. Billerica, MA, USA: Computational Mechanics, Inc., 1993.
[10] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla, “Segmentation and recognition using structure from motion point clouds,” in *ECCV*, 2008, pp. 44–57.
[11] K.-J. Choi and H.-S. Ko, “Stable but responsive cloth,” in *SIGGRAPH ’02*. ACM, 2002, pp. 604–611.
[12] A. Clegg, W. Yu, J. Tan, C. K. Liu, and G. Turk, “Learning to dress: Synthesizing human dressing motion via deep reinforcement learning,” *ACM Trans. Graph.*, vol. 37, no. 6, pp. 179:1–179:10, Dec. 2018.
[13] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr, “Implicit fairing of irregular meshes using diffusion and curvature flow,” in *SIGGRAPH ’99*. ACM, 1999, pp. 317–324.
[14] C. Duriez, “Control of Elastic Soft Robots based on Real-Time Finite Element Method,” in *IEEE ICRA*, Karlsruhe, France, 2013.
[15] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, “Convolutional networks on graphs for learning molecular fingerprints,” in *NIPS*, 2015, pp. 2224–2232.
[16] J. Fras, Y. Noh, M. Macias, H. Wurdemann, and K. Althoefer, “Bio-inspired octopus robot based on novel soft fluidic actuator,” in *IEEE ICRA*, May 2018, pp. 1583–1588.
[17] L. Fulton, V. Modi, D. Duvenaud, D. I. W. Levin, and A. Jacobson, “Latent-space dynamics for reduced deformable simulation,” *Computer Graphics Forum*, 2019.
[18] L. Gao, Y.-K. Lai, J. Yang, L.-X. Zhang, L. Kobbelt, and S. Xia, “Sparse Data Driven Mesh Deformation,” *arXiv:1709.01250*, 2017.
[19] T. F. Gast, C. Schroeder, A. Stomakhin, C. Jiang, and J. M. Teran, “Optimization integrator for large time steps,” *IEEE transactions on visualization and computer graphics*, vol. 21, no. 10, pp. 1103–1115, 2015.
[20] A. George and E. Ng, “On the complexity of sparse qr and lu factorization of finite-element matrices,” *SIAM Journal on Scientific and Statistical Computing*, vol. 9, no. 5, pp. 849–861, 1988.
[21] E. Grinspun, A. N. Hirani, M. Desbrun, and P. Schröder, “Discrete shells,” in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA ’03. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2003, pp. 62–67.
[22] G. Hirota, S. Fisher, and M. Lin, “Simulation of non-penetrating elastic bodies using distance fields,” 2000.
[23] Z. Hu, T. Han, P. Sun, J. Pan, and D. Manocha, “3-d deformable object manipulation using deep neural networks,” *IEEE RA-L*, vol. PP, pp. 1–1, 07 2019.
[24] 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, July 2011.
[25] Z. Huang, J. Yao, Z. Zhong, Y. Liu, and X. Guo, “Sparse localized decomposition of deformation gradients,” *Comp. Graph. Forum*, vol. 33, no. 7, pp. 239–248, 2014.
[26] B. Jia, Z. Pan, Z. Hu, J. Pan, and D. Manocha, “Cloth manipulation using random forest-based controller parametrization,” *CoRR*, vol. abs/1802.09661, 2018.
[27] B. Jia, Z. Pan, and D. Manocha, “Fast motion planning for high-dof robot systems using hierarchical system identification,” 2018.
[28] L. Kavan, P.-P. Sloan, and C. O’Sullivan, “Fast and efficient skinning of animated meshes,” *Comp. Graph. Forum*, vol. 29, no. 2, pp. 327–336, 2010.
[29] D. P. Kingma and M. Welling, “Auto-encoding variational bayes.” *arXiv:1312.6114*, 2013.
[30] K. Lakshmanan, A. Sachdev, Z. Xie, D. Berenson, K. Goldberg, and P. Abbeel, “A constraint-aware motion planning algorithm for robotic folding of clothes,” in *Experimental Robotics*. Springer, 2013, pp. 547–562.
[31] M. G. Larson and F. Bengzon, *The Finite Element Method: Theory, Implementation, and Applications*. Springer Publishing Company, Incorporated, 2013.
[32] Y. Li, Y. Yue, D. Xu, E. Grinspun, and P. K. Allen, “Folding deformable objects using predictive simulation and trajectory optimization,” in *IEEE/RSJ IROS*. IEEE, pp. 6000–6006.
[33] Y.-J. Lim and S. De, “Real time simulation of nonlinear tissue response in virtual surgery using the point collocation-based method of finite spheres,” *Computer Methods in Applied Mechanics and Engineering*, vol. 196, no. 31-32, pp. 3011–3024, 2007.
[34] T. Liu, A. W. Bargteil, J. F. O’Brien, and L. Kavan, “Fast simulation of mass-spring systems,” *ACM Trans. Graph.*, vol. 32, no. 6, pp. 209:1–7, Nov. 2013.
[35] L. v. d. Maaten and G. Hinton, “Visualizing data using t-sne,” *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.
[36] H. Maron, M. Galun, N. Aigerman, M. Trope, N. Dym, E. Yumer, V. G. Kim, and Y. Lipman, “Convolutional neural networks on surfaces via seamless toric covers,” *ACM Trans. Graph.*, vol. 36, no. 4, pp. 71:1–71:10, July 2017.
[37] D. Maturana and S. Scherer, “Voxnet: A 3d convolutional neural network for real-time object recognition,” in *IEEE/RSJ IROS*, Sept 2015, pp. 922–928.
[38] K. Nakajima, “Muscular-hydrostat computers: Physical reservoir computing for octopus-inspired soft robots,” in *Brain Evolution by Design*. Springer, 2017, pp. 403–414.
[39] R. Narain, A. Samii, and J. F. O’Brien, “Adaptive anisotropic remeshing for cloth simulation,” *ACM Trans. Graph.*, vol. 31, no. 6, pp. 147:1–10, Nov. 2012.
[40] T. Neumann, K. Varanasi, S. Wenger, M. Wacker, M. Magnor, and C. Theobalt, “Sparse localized deformation components,” *ACM Trans. Graph.*, vol. 32, no. 6, pp. 179:1–179:10, Nov. 2013.
[41] Y. J. Oh, T. M. Lee, and I.-K. Lee, “Hierarchical cloth simulation using deep neural networks,” *arXiv:1802.03168*, 2018.
[42] Z. Pan and D. Manocha, “Realtime planning for high-dof deformable bodies using two-stage learning,” in *IEEE ICRA*, May 2018, pp. 1–8.
[43] Z. Pan, C. Park, and D. Manocha, “Robot motion planning for pouring liquids,” in *Twenty-Sixth International Conference on Automated Planning and Scheduling*, 2016.
[44] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” *arXiv:1511.06434*, 2015.
[45] A. Rajeswaran*, V. Kumar*, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, “Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations,” in *Proceedings of Robotics: Science and Systems (RSS)*, 2018.
[46] O. Sorkine and M. Alexa, “As-rigid-as-possible surface modeling,” in *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, ser. SGP ’07. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2007, pp. 109–116.
[47] Q. Tan, L. Gao, Y. Lai, J. Yang, and S. Xia, “Mesh-based autoencoders for localized deformation component analysis,” in *AAAI*, 2018.
[48] L. Vasa and V. Skala, “A perception correlated comparison method for dynamic meshes,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 2, pp. 220–230, Feb. 2011.
[49] B. Wang, L. Wu, K. Yin, L. Liu, and H. Huang, “Deformation capture and modeling of soft objects,” *ACM Trans. Graph.*, vol. 34, no. 4, pp. 94:1–94:12, 2015.
[50] J. M. Wang, D. J. Fleet, and A. Hertzmann, “Gaussian process dynamical models for human motion,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 283–298, Feb 2008.
[51] S. Wiewel, M. Becher, and N. Thuerey, “Latent-space physics: Towards learning the temporal evolution of fluid flow,” *arXiv:1802.10123*, 2018.
[52] Y. Xie, E. Franz, M. Chu, and N. Thuerey, “tempogan: A temporally coherent, volumetric gan for super-resolution fluid flow,” *ACM Trans. Graph.*, vol. 37, no. 4, p. 95, 2018.
[53] X. Yan, J. Yang, E. Yumer, Y. Guo, and H. Lee, “Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision,” *arXiv:1612.00814*, 2016.
[54] Z. Yang, J. Peltonen, and S. Kaski, “Optimization equivalence of divergences improves neighbor embedding,” in *ICML*, 2014, pp. 460–468.
[55] J. Zhu, S. C. H. Hoi, and M. R. Lyu, “Nonrigid shape recovery by gaussian process regression,” in *IEEE CVPR*, June 2009, pp. 1319–1326.
[56] H. Zou, T. Hastie, and R. Tibshirani, “Sparse principal component analysis,” *J. Comp. Graph. Statistics*, vol. 15, p. 2006, 2004.