A Deep Emulator for Secondary Motion of 3D Characters

Mianlun Zheng[1], Yi Zhou[2], Duygu Ceylan[2], Jernej Barbič[1]
[1]University of Southern California, [2]Adobe Research
{mianlunz, jnb}@usc.edu, {yizho, ceylan}@adobe.com

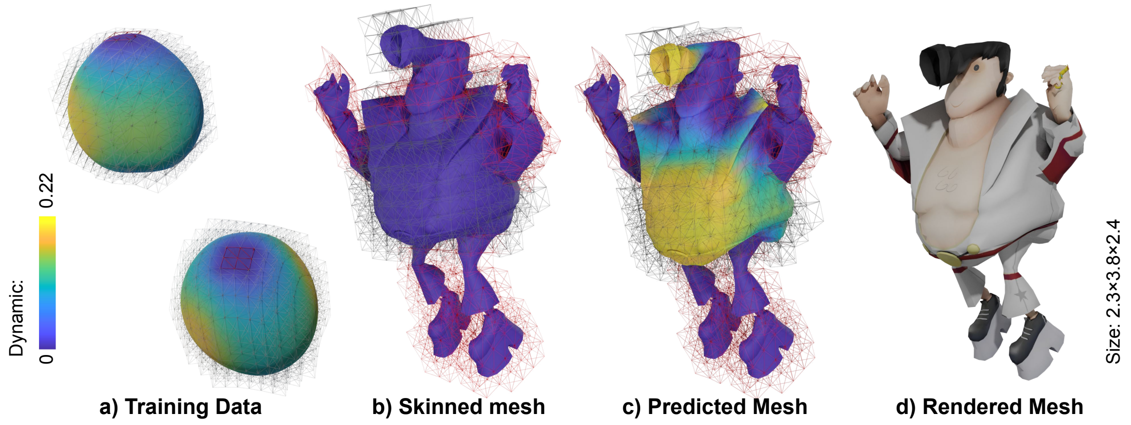


Figure 1: a) Our method is trained on a primitive (sphere) dataset with deformation dynamics but generalizes across topology varying 3D characters. The uniform volumetric mesh surrounding the surface mesh is used for prediction, where the red vertices are set to be constraints. b) At inference time, the input is an artist-specified skinned mesh without dynamics. c) Our neural network predicts the dynamic mesh with secondary motion. d) The surface mesh is rendered with textures.

Abstract

Fast and light-weight methods for animating 3D characters are desirable in various applications such as computer games. We present a learning-based approach to enhance skinning-based animations of 3D characters with vivid secondary motion effects. We design a neural network that encodes each local patch of a character simulation mesh where the edges implicitly encode the internal forces between the neighboring vertices. The network emulates the ordinary differential equations of the character dynamics, predicting new vertex positions from the current accelerations, velocities and positions. Being a local method, our network is independent of the mesh topology and generalizes to arbitrarily shaped 3D character meshes at test time. We further represent per-vertex constraints and material properties such as stiffness, enabling us to easily adjust the dynamics in different parts of the mesh. We evaluate our method on various character meshes and complex motion sequences. Our method can be over 30 times more efficient than ground-truth physically based simulation, and outperforms alternative solutions that provide fast approximations.

1. Introduction

Fast and light-weight methods for animating 3D characters are desirable in various applications including computer games and film visual effects. Traditional skinning-based mesh deformation provides a fast geometric approach but often lacks realistic dynamics. On the other hand, physically-based simulation can add plausible secondary motion to skinned animations, augmenting them with visually realistic and vivid effects, but at the cost of heavy computation.

Recent research has explored deep learning methods to approximate physically-based simulation in a much more time-efficient manner. While some approaches have focused on accelerating specific parts of the simulation [18, 7, 20], others have proposed end-to-end solutions that predict dynamics directly from mesh based features [1, 11, 11, 25]. While demonstrating impressive results, these methods still have some limitations. Most of them assume a fixed mesh topology and thus need to train different networks for different character meshes. Moreover, in order to avoid the computational complexity of training networks on high resolution meshes, some methods operate on reduced subspaces with limited degrees of freedom, leading to low accuracy.

In this paper, we propose a deep learning approach to predict secondary motion, i.e., the deformable dynamics of given skinned animations of 3D characters. Our method addresses the shortcomings of the recent learning-based approaches by designing a network architecture that can reflect the actual underlying physical process. Specifically, our network models the simulation using a volumetric mesh consisting of uniform tetrahedra surrounding the character mesh, where the mesh edges encode the internal forces that depend on the current state (i.e., displacements, velocities, accelerations), material properties (e.g., stiffness), and constraints on the vertices. Mesh vertices encode the inertia. Motivated by the observation that within a short time instance the secondary dynamics of a vertex is mostly affected by its current state, as well as the internal forces due to its neighbors, our network operates on local patches of the volumetric mesh. In addition to avoiding the computational complexity of encoding high resolution character meshes as large graphs, this also enables our method to be applied to any character mesh, independent of its topology. Finally, our network encodes per-vertex material properties and constraints, giving the user the ability to easily prescribe varying properties to different parts of the mesh to control the dynamic behaviour.

As a unique benefit of the generalization capability of our model, we demonstrate that it is not necessary to construct a massive training dataset of complex meshes and motions. Instead, we construct our training data from primitive geometries, such as a volumetric mesh of a sphere. Our network trained on this dataset can generate detailed and visually plausible secondary motions on much more complex 3D characters during testing. By assigning randomized motions to the primitives during training, we are able to let the local patches cover a broad motion space, which improves the network's online predictions in unseen scenarios.

We evaluate our method on various character meshes and complex motion sequences. We demonstrate visually plausible and stable secondary motion while being over 30 times faster than the implicit Euler method commonly used in physically-based simulation. We also provide comparisons to faster methods such as the explicit central differences method and other learning-based approaches that utilize graph convolutional networks. Our method outperforms those approaches both in terms of accuracy and robustness.

2. Related Work

2.1. Physically based simulation methods

Complementing skinning-based animations with secondary motion is a well-studied problem. Traditional approaches resort to using physically-based simulation [31, 30]. However, it is well-known that physically based methods often suffer from computational complexity. Therefore, in the last decade, a series of methods were proposed to accelerate the computation process, including example-based dynamic skinning [26], efficient elasticity calculation [19], formulation of motion equations in the rig subspace [9, 10], and the coupling of the skeleton dynamics and the soft body dynamics [17]. These approaches still have some limitations such as robustness issues due to explicit integration, or unnatural deformation effects due to remeshing, while our method is much more robust in handling various characters and complex motions.

2.2. Learning based methods

Grzeszczuk et al. [8] presented one of the earliest works that demonstrated the possibility of replacing numerical computations with a neural network. Since then research in this area has advanced, especially in the last few years. While some approaches have presented hybrid solutions where a neural network replaces a particular component of the physically based simulation process, others have presented end-to-end solutions.

In the context of hybrid approaches, plug-in deep neural networks were applied in combination with the Finite Elements Method (FEM), to help accelerate the simulation. For example, the node-wise NNWarp [18] was proposed to efficiently map the linear nodal displacements to nonlinear ones. Fulton et al.[7] utilized an autoencoder to project the target mesh to a lower dimensional space to increase the computation speed. Similarly, Tan et al. [28] designed a CNN-based network for dimension reduction to accelerate thin-shell deformable simulations. Romero et al. [22] built a data-driven statistical model to kinematically drive the FEM mechanical simulation. Meister et al. [20] explored the use of neural networks to accelerate the time integration step of the Total Lagrangian Explicit Dynamics (TLED) for complex soft tissue deformation simulation. Finally, Deng et al. [6] modeled the force propagation mechanism in their neural networks. Those approaches improved efficiency but at the cost of accuracy and are not friendly to end users who are not familiar with physical techniques. Ours, instead, allows the user to adjust the animation by simply painting the constraints and stiffness properties.

End-to-end approaches assume the target mesh is provided as input and directly predict the dynamics behaviour. For instance, Bailey et al. [1] enriched the real-time skinning animation by adding the nonlinear deformations learned from film-quality character rigs. The work of Holden et al [11] first trained an autoencoder to reduce the simulation space and then learned to efficiently approximate the dynamics projected to the subspace. Similarly, SoftSMPL [25] modeled the realistic soft-tissue dynamics based on a novel motion descriptor and a neural-network-based recurrent regressor that ran in the nonlinear deformation subspace extracted from an autoencoder. While all these approaches presented impressive results, their main drawback was the assumption of a fixed mesh topology requiring different networks to be trained for different meshes. Our approach, on the other hand, operates at a local patch level and can therefore generalize to different meshes at test time.

Lately, researchers started to utilize the Graph Convolutional Network (GCN) for simulation tasks due to its advantage in handling topology-free graphs. The GCN encodes the vertex positional information and aggregates the latent features to a certain node by using the propagation rule. For particle-based systems, graphs are constructed based on the local adjacency of the particles at each frame and fed into GCNs [16, 29, 23, 5]. Concurrently, Pfaff et al. [21] proposed a GCN for surface mesh-based simulation. While these GCN models interpret the mesh dynamics prediction as a general spatio-temporal problem, we incorporate physics into the design of our network architecture, e.g. inferring latent embedding for inertia and internal forces, which enables us to achieve more stable and accurate results (Section 4.3).

3. Method

Given a 3D character and its primary motion sequence obtained using standard linear blend skinning techniques [13], we first construct a volumetric (tetrahedral) mesh and a set of barycentric weights to linearly embed the vertices of the character's surface mesh into the volumetric mesh [14], as shown in Figure 2. Our network operates on the volumetric mesh and predicts the updated vertex positions with deformable dynamics (also called the secondary motion) at each frame given the primary motion, the constraints and the material properties. The updated volumetric mesh vertex positions then drive the original surface mesh via the barycentric embedding, and the surface mesh is used for rendering; such a setup is very common and standard in computer animation.

We denote the reference tetrahedral mesh and its number of vertices by X and n, respectively. The skinned animation (primary motion) is represented as a set of time-varying positions $\mathbf{x} \in \mathbb{R}^{3n}$. Similarly, we denote the predicted dynamic mesh by U and its positions by $\mathbf{u} \in \mathbb{R}^{3n}$.

Our method additionally encodes mass $\mathbf{m} \in \mathbb{R}^{n}$ and stiffness $\mathbf{k} \in \mathbb{R}^{n}$ properties. The stiffness is represented as Young's modulus. By painting different material properties per vertex over the mesh, users can control the dynamic effects, namely the deformation magnitude.

In contrast to previous works [25, 21] which trained neural networks directly on the surface mesh, we choose to operate on the volumetric mesh for several reasons. First, volumetric meshes provide a more efficient coarse representation and can handle character meshes that consist of multiple disconnected components. For example, in our experiments the "Michelle" character (see Figure 2) consists

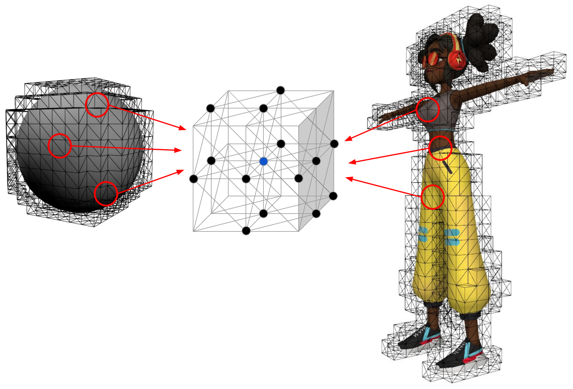


Figure 2: The tetrahedral simulation mesh and the embedded surface mesh. The local patch consists of a center vertex and its neighbors, defined as the vertices of the tetrahedra touching the center vertex.

of $14k$ vertices whereas the corresponding volumetric mesh only has $1k$ vertices. In addition, the "Big Vegas" character mesh (see Figure 1) has eight disconnected components, requiring the artist to build a watertight mesh first if using a method that learns directly on the surface mesh. Furthermore, volumetric meshes not only capture the surface of the character but also the interior, leading to more accurate learning of the internal forces. Finally, we use a uniformly voxelized mesh subdivided into tetrahedra as our volumetric mesh, which enables our method to generalize across character meshes with varying shapes and resolutions.

Next, we will first explain the motion equations in physically-based simulation and then discuss our method in detail, drawing inspiration from the physical process.

3.1. Physically-based Motion Equations

In constraint-based physically-based simulation [2], the equations of motion are

$$\begin{gathered} \mathbf{M}\ddot{\mathbf{u}} + \mathbf{D}\dot{\mathbf{u}} + \mathbf{f}_{int}(\mathbf{u}) = \mathbf{0} \\ \text{subject to } \mathbf{Cu} = \mathbf{Sx}(t), \end{gathered} \tag{1}$$

where $\mathbf{M} \in \mathbb{R}^{3n \times 3n}$ is the diagonal (lumped) mass matrix (as commonly employed in interactive applications), $\mathbf{D}$ is the Rayleigh damping matrix, and $\mathbf{u} \in \mathbb{R}^{3n}$, $\dot{\mathbf{u}} \in \mathbb{R}^{3n}$ and $\ddot{\mathbf{u}} \in \mathbb{R}^{3n}$ represent the positions, velocities and accelerations, respectively. The quantity $\mathbf{f}_{int}(\mathbf{u})$ represents the internal elastic forces. Secondary dynamics occurs because the constraint part of the mesh "drives" the free part of the mesh. Constraints are specified via the constraint matrix $\mathbf{C}$ and the selection matrix $\mathbf{S}$. In order to leave room for secondary dynamics for 3D characters, we typically do not constrain *all* the vertices of the mesh, but only a subset. For example, in the Big Vegas example (see Figure 1), we constrain the legs, the arms and the core inside the torso and head, but do not constrain the belly and hair, so that we can generate secondary dynamics in those unconstrained regions.

One approach to timestep Equation 1 is to use an explicit integrator, such as central differences:

$$\begin{aligned}\dot{\mathbf{u}}(t+1) &= \dot{\mathbf{u}}(t) + \frac{\ddot{\mathbf{u}}(t)+\ddot{\mathbf{u}}(t+1)}{2}\Delta t,\\ \mathbf{u}(t+1) &= \mathbf{u}(t) + \dot{\mathbf{u}}(t)\Delta t + \ddot{\mathbf{u}}(t)\frac{\Delta t^2}{2},\end{aligned} \tag{2}$$

where t and $t+1$ denote the state of the mesh in the current and next frames, respectively, and Δt is the timestep. While the explicit integration is fast, it suffers from stability issues. Hence, the slower but stable implicit backward Euler integrator is often preferred in physically-based simulation [3]:

$$\begin{aligned}\dot{\mathbf{u}}(t+1) &= \dot{\mathbf{u}}(t) + \ddot{\mathbf{u}}(t+1)\Delta t,\\ \mathbf{u}(t+1) &= \mathbf{u}(t) + \dot{\mathbf{u}}(t+1)\Delta t.\end{aligned} \tag{3}$$

We propose to approximate implicit integration as

$$\begin{aligned}\dot{\mathbf{u}}(t+1) = \dot{\mathbf{u}}(t) + &\\ f_\theta\Big(\mathbf{u}(t), \dot{\mathbf{u}}(t), \ddot{\mathbf{u}}(t), \mathbf{x}(t), \dot{\mathbf{x}}(t), \ddot{\mathbf{x}}(t), \mathbf{m}, \mathbf{k}\Big)\Delta t,&\\ \mathbf{u}(t+1) = \mathbf{u}(t) + \dot{\mathbf{u}}(t+1)\Delta t,&\end{aligned} \tag{4}$$

where f is a differentiable function constructed as a neural network with learned parameters θ.

3.2. Network design

As shown in Equation 1, predicting the secondary dynamics entails solving for $3n$ degrees of freedom for a mesh with n vertices. Hence, directly approximating f_θ in Equation 4 to predict all the degrees of freedom at once would lead to a huge and impractical network, which would furthermore not be applicable to input meshes with varying number of vertices and topologies. Inspired by the intuition that within a very short time moment, the motion of a vertex is mostly affected by its own inertia and the internal forces from its neighboring vertices, we design our network to operate on a local patch instead. As illustrated in Figure 3, the 1-ring local patch consists of one center vertex along with its immediate neighbors in the volumetric mesh. Even though two characters might have very different mesh topologies, as shown in Figure 2, their local patches will often be more similar, boosting the generalization ability of our network. The internal forces are caused by the local stress, and the aggregation of the internal forces acts to pull the vertices to their positions in the reference motion, to reduce the elastic energy. Thus, the knowledge of the per-edge deformation and the per-vertex reference motion are needed for secondary motion prediction.

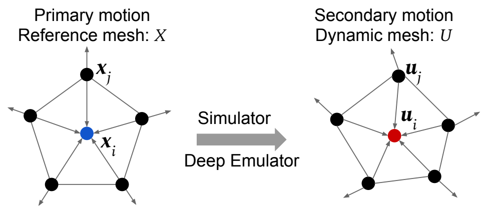


Figure 3: The input reference mesh X and the target dynamic mesh U. We draw the meshes in 2D for convenience.

Hence, we propose to emulate this process as follows:

$$\begin{aligned}\mathbf{z}_i^{\text{inertia}} &= f_\alpha^{\text{inertia}}(\mathbf{u}_i, \mathbf{x}_i, k_i, m_i),\\ \mathbf{z}_{i,j}^{\text{internal\_force}} &= f_\beta^{\text{internal\_force}}(\mathbf{u}_{i,j}, \mathbf{x}_{i,j}, k_i, m_i),\\ \ddot{\mathbf{u}}_i &= g_\gamma\big(\mathbf{z}_i^{\text{inertia}}, \sum_{j\in\mathbb{N}_i} \mathbf{z}_{i,j}^{\text{internal\_force}}\big),\end{aligned} \tag{5}$$

where $f_\alpha^{\text{inertia}}$, $f_\beta^{\text{internal\_force}}$ and g_γ are three different multi-layer perceptrons (MLPs) as shown in Figure 4, $\mathbb{N}_i$ are neighboring vertices of i (excluding i), and the double indices i, j denote the central vertex i and a neighbor j. Quantities $\mathbf{z}_i^{\text{inertia}}$ and $\mathbf{z}_i^{\text{internal\_force}}$ are high dimensional latent vectors that represent an embedding for inertia dynamics and the internal forces from each neighboring vertex, respectively. Perceptron g_γ receives the concatenation of $\mathbf{z}_i^{\text{inertia}}$ and the sum of $\mathbf{z}_i^{\text{internal\_force}}$ to predict the final acceleration of a vertex i. In practice, for simplicity, we train g_γ to directly predict $\dot{\mathbf{u}}(t+1)\Delta t = \mathbf{u}(t+1) - \mathbf{u}(t)$ since we assume a fixed timestep of Δt in our experiments.

We implement all the three MLPs with four hidden fully connected layers activated by the Tanh function, and one output layer. During training, we provide the ground truth positions in the dynamic mesh as input. During testing, we provide the predictions of the network as input in a recurrent manner. Next, we discuss the details of these components.

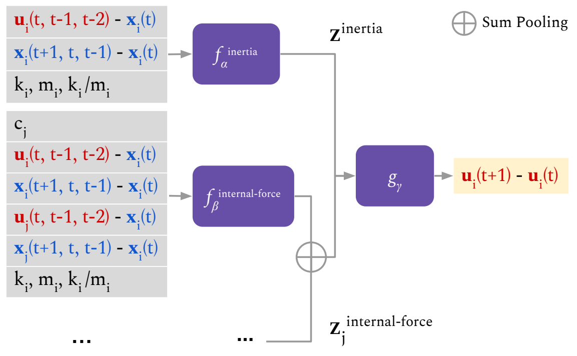

$\mathbf{u}_i$(t, t-1, t-2) - $\mathbf{x}_i$(t)
$\mathbf{x}_i$(t+1, t, t-1) - $\mathbf{x}_i$(t)
k_i, m_i, k_i/m_i
c_j
$\mathbf{u}_i$(t, t-1, t-2) - $\mathbf{x}_i$(t)
$\mathbf{x}_i$(t+1, t, t-1) - $\mathbf{x}_i$(t)
$\mathbf{u}_j$(t, t-1, t-2) - $\mathbf{x}_i$(t)
$\mathbf{x}_j$(t+1, t, t-1) - $\mathbf{x}_i$(t)
k_i, m_i, k_i/m_i
$f_\alpha^{\text{inertia}}$
$f_\beta^{\text{internal-force}}$
$\mathbf{z}^{\text{inertia}}$
$\mathbf{z}_j^{\text{internal-force}}$
g_γ
$\mathbf{u}_i$(t+1) - $\mathbf{u}_i$(t)
Sum Pooling
...
...

Figure 4: Our network architecture.

MLP $f_\alpha^{\text{inertia}}$: This perceptron focuses on the center vertex itself, encoding the "self-inertia" information. That is, the center vertex tends to continue its current motion, driven by both the velocity and acceleration. The input to $f_\alpha^{\text{inertia}}$ is the position of the center vertex in the last three frames both on the dynamic and skinned mesh, $\mathbf{u}(t)$,$\mathbf{u}(t-1)$,$\mathbf{u}(t-2)$ and $\mathbf{x}(t+1)$,$\mathbf{x}(t)$, $\mathbf{x}(t-1)$, as well as its material properties, $k_i, m_i, k_i/m_i$. The positions are represented in local coordinates with respect to $\mathbf{x}(t)$, the current position of the center vertex in the reference motion. The positions in the last three frames implicitly encode the velocity and the acceleration. Since we know that the net force applied on the central vertex is divided by its mass in Equation 4 and it is relatively hard for the network to learn multiplication or division, we also include k_i/m_i explicitly in the input. The hidden layer and output size is 64.

MLP $f_\beta^{\text{internal\_force}}$: For an unconstrained center vertex i, perceptron $f_\beta^{\text{internal\_force}}$ encodes the "internal forces" contributed by its neighbors. The input to the MLP is similar to $f_\alpha^{\text{inertia}}$, except that we provide information both for the center vertex as well as its neighbors. For each neighboring vertex j, we also provide the constraint information c_j ($c_j = 0$ if a free vertex; $c_j = 1$ if constrained). Each $f_\beta^{\text{internal\_force}}$ provides a latent vector for the central vertex. The hidden layer and output size is 128.

MLP g_γ: This module receives the concatenated outputs from $f_\alpha^{\text{inertia}}$ and the aggregation of $f_\beta^{\text{internal\_force}}$, and predicts the final displacement of the central vertex i in the dynamic mesh. The input and hidden layer size is 192. We train the final network with the mean square error loss:

$$l = \frac{1}{n}\sum_i^n ||\dot{\mathbf{u}}_i(t+1) - \dot{\mathbf{u}}_i'(t+1)||_2^2, \tag{6}$$

where $\dot{\mathbf{u}}_i'(t+1)$ is the ground truth. We adopted the Adam optimizer for training, with a learning rate starting from 0.0001 along with a decay factor of 0.96 at each epoch.

3.3. Training Primitives

Because our method operates on local patches, it is not necessary to train it on complex character meshes. In fact, we found that a training dataset constructed by simulating basic primitives, such as a sphere (under various motions and material properties), is sufficient to generalize to various character meshes at test time. Specifically, we generate random motion sequences by prescribing random rigid body motion of a constrained beam-shaped core inside the spherical mesh. The motion of this rigid core excites dynamic deformations in the rest of the sphere volumetric mesh. Each motion sequence starts by applying, to the rigid core, a random acceleration and angular velocity with respect to a random rotation axis. Next, we reverse the acceleration so that the primitive returns back to its starting position, and let the primitive's secondary dynamics oscillate out for a few frames. While the still motions ensure that we cover the cases where local patches are stationary (but there is still residual secondary dynamics from primary motion), the random accelerations help to sample a diverse set of motions of local patches as much as possible. Doing so enhances the networks's prediction stability.

4. Experiments

In this section, we show qualitative and quantitative results of our method, as well as comparisons to other methods. We also run an ablation study to verify why explicitly providing the position information on the reference mesh as input is necessary.

4.1. Dataset and evaluation metrics

For training, we use a uniform tetrahedral mesh of a sphere. We generate 80 random motion sequences at 24 fps, using the Vega FEM simulator [4, 27]. For each motion sequence, we use seven different material settings. Each motion sequence consists of 456 frames resulting in a total of 255k frames in our training set.

We evaluate our method on 3D character animations obtained from Adobe's Mixamo dataset [12]. Neither the character meshes nor the primary motion sequences are seen in our training data. We create test cases for five different character meshes as listed in Table 1 and 15 motions in total. The volumetric meshes for the test characters use the same uniform tetrahedron size as our training data. For all the experiments, we report three types of metrics:

- Single-frame RMSE: We measure the average root-mean-square error (RMSE) between the prediction and the ground truth over all frames, while providing the ground truth positions of the previous frames as input.
- Rollout RMSE: We provide the previous predictions of the network as input to the current frame in a recurrent manner and measure the average RMSE between the prediction and the ground truth over all frames.
- $E_{elastic}[min, stdev, max]$: We use the concept of elastic energy in physically-based simulation to detect abnormalities in the deformation sequence, or any possible mesh explosions. For each frame, we calculate the elastic energy based on the current mesh displacements with respect to its reference state. We list the the min, max as well as the standard deviation ($stdev$) to show the energy distribution across the animation.

4.2. Analysis of Our Method

Performance: In Table 1, we show the speed t_{ours} of our method, as well as that of the ground truth method t_{GT} and a baseline method t_{BL}. For each method, we record the time to calculate the dynamic mesh but exclude other components such as initialization, rendering and mesh interpolation.

We adopted the implicit backward Euler approach (Equation 3) as ground truth and the faster explicit central differences integration (Equation 2) as the baseline. Both our baseline and ground truth were optimized using the deformable object simulation library, Vega FEM [4, 27], and accelerated using multi-cores via Intel Thread Building Blocks (TBB), with 8 cores for assembling the internal forces and 16 cores for solving the linear system. The experiment platform is with 2.90 GHz Intel Xeon(R) CPU E5-2690 (32 GB RAM) which provides for a highly competitive baseline/ground truth implementation. We ran our trained model on a GeForce RTX 2080 graphics card (8 GB RAM). We also tested it on CPU, without any multi-thread acceleration.

Moreover, we also provide performance results for the same character mesh (Big Vegas) with different voxel resolutions. To handle different resolutions of testing meshes, we resize the volumetric mesh to have the local patch similar to the training data (i.e., the shortest edge length is 0.2).

character	# vtx	t_{GT}	t_{BL}	t^{GPU}_{ours}	t^{CPU}_{ours}
Big vegas	1468	0.58	0.056	**0.012**	0.017
Kaya	1417	0.52	0.052	**0.012**	0.015
Michelle	1105	0.33	0.032	**0.011**	0.015
Mousey	2303	0.83	0.084	**0.014**	0.020
Ortiz	1258	0.51	0.049	**0.012**	0.015
Big vegas	6987	2.45	0.32	**0.032**	0.14
Big vegas	10735	4.03	0.53	**0.046**	0.24
Big vegas	18851	8.26	1.06	**0.068**	0.42
Big vegas	39684	24.24	2.96	**0.14**	0.89

Table 1: The running time (s/frame) of a single step (1/24 second) for the ground truth, the baseline, and our method.

Results indicate that when ran on GPU (CPU), our method is around 30 ($\sim$20) times faster than the implicit integrator and 3 ($\sim$2) times faster than the explicit integrator, per frame. Under an increasing number of vertices, our method has an even more competitive performance. Although the explicit method has comparable speed to our method, the simulation explodes after a few frames. In practice, explicit methods require much smaller time steps, which required additional 100 sub-steps in our experiments, to achieve stable quality. We provide a more detailed report on the speed-stability relationship of explicit integration in the supplementary material.

Generalization: We train the network on the sphere dataset and achieve a single frame RMSE of 0.0026 on the testing split of this dataset (the sphere has a radius of 2). As listed in Table 2, when tested on characters, our method achieves a single frame RMSE of 0.0067, showing remarkable generalization capability (we note that the shortest edge length on the volumetric character meshes is 0.2). The mean rollout error increases to 0.064 after running the whole sequences due to error accumulation, but elastic energy statistics are still close to the ground truth. From the visualization of the ground truth and our results in Figure 7, we can see that although the predicted secondary dynamics slightly deviate from the ground truth, they are still visually plausible. We further plot the rollout prediction RMSE and elastic energy of the Big Vegas character in Figure 5. It can be seen that the prediction error remains under 0.07, and the mean elastic energy of our method is always close to the ground truth for the whole sequence, whereas the baseline method explodes quickly. We provide such rollout prediction plots for all characters and the video results[1] in supplemental material.

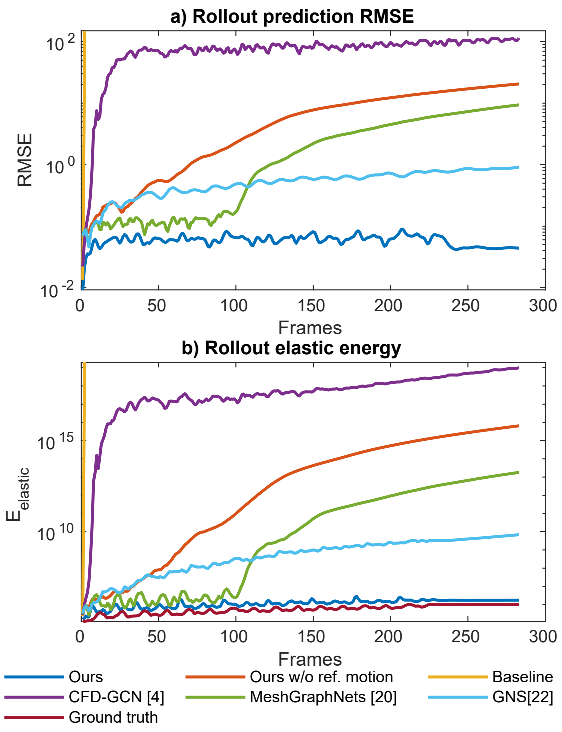


Figure 5: The rollout prediction results of our method and others, tested on the Big Vegas character with 283-frame hip hop dancing motion.

Non-homogeneous Dynamics: Figure 6 shows how to control the dynamics by painting non-homogeneous material properties over the mesh. Varying stiffness values are painted on the hair and the breast region on the volumetric mesh. For better visualization, we render the material settings across the surface mesh in the figure. We display three different material settings, by assigning different stiffness k values. Larger k means stiffer material, hence the corresponding region exhibits less dynamics. In contrast, the regions with smaller k show significant dynamic effects. This result demonstrates that our method correctly models the effect of material properties while providing an interface for the artist to efficiently adjust the desired dynamic effects.

Ablation study: To demonstrate that it is necessary to incorporate the reference mesh motion into the input features of

[1]Videos results are available at `https://zhengmianlun.github.io/publications/deepEmulator.html`.

Methods	single frame	rollout-24	rollout-48	rollout-all	$E_{elastic}$ $[min, stdev, max]$
Ground truth	\	\	\	\	$[3.81E3, 4.06E5, 2.60E6]$
Our method	**0.0067**	**0.059**	**0.062**	**0.064**	$[\mathbf{4.84E3}, \mathbf{6.51E5}, \mathbf{6.32E6}]$
Ours w/o ref. motion	0.050	0.20	0.38	10.09	$[1.62E4, 6.7E16, 4.7E17]$
Baseline	\	$7.26E120$	$9.63E120$	$17.5E120$	$[9.26E0, Nan, 7.22E165]$
CFD-GCN [5]	0.040	41.17	70.55	110.07	$[3.96E4, 1.1E22, 1.6E23]$
GNS [23]	0.049	0.22	0.34	0.54	$[1.09E4, 2.0E11, 2.3E10]$
MeshGraphNets [21]	0.050	0.11	0.43	4.46	$[1.69E4, 1.1E15, 1.1E14]$

Table 2: The single-frame RMSE, rollout-24, rollout-48 and rollout-all of our method and others tested on all five characters with 15 different motions. The shortest edge length in the test meshes is 0.2.

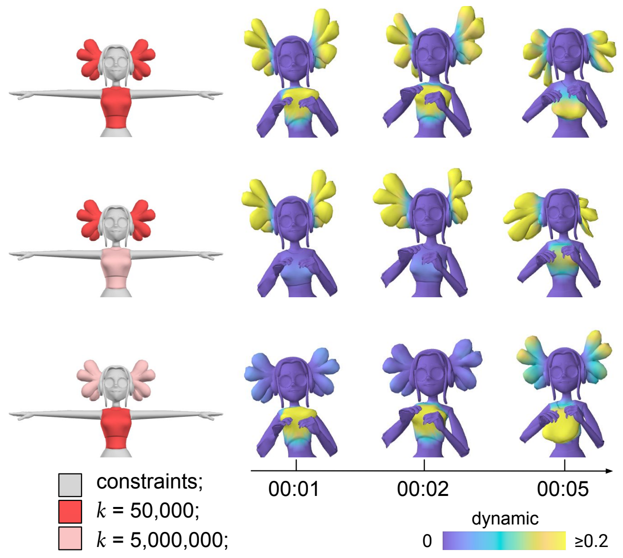


Figure 6: Non-homogeneous dynamics, tested on the Michelle character with 122-frame cross-jumps motion. We only show the upper region (see Figure 2 for the full mesh).

our network, we performed an ablation study. To ensure that the constrained vertices are still driving the dynamic mesh in the absence of the reference information, we update the positions of the constrained vertices based on the reference motion, at the beginning of each iteration. As input to our network architecture, we use the same set of features except the positions on the reference mesh. The results of "Ours w/o ref. motion" in Table 2 and Figure 7 and 5 demonstrate that this version is inferior to our original method, especially when running the network over a long time sequence. This establishes that the reference mesh is indispensable to the quality of the network's approximation.

4.3. Comparison to Previous Work

As discussed in Section 2, several recent particle-based physics and mesh-based deformation systems utilized graph convolutional networks (GCNs). In this section, we train these network models on the same training set as our method and test on our character meshes.

CFD-GCN [5]: We implemented our version of the CFD-GCN architecture, adopting the convolution kernel of [15]. However, we ignored the remeshing part because we assume that the mesh topology remains fixed when predicting secondary motion. As input, we provide the same information as our method, namely the constraint states of the vertices, the displacements and the material properties. We found that the network structure recommended in the paper resulted in a high training error. We then replaced the originally proposed ReLu activation function with the Tanh activation (as used in our method), which significantly improved the training performance. Even so, as shown in Table 2 and Figure 5, the rollout prediction explodes very quickly. We speculate that although the model aggregates the features from the neighbors to a central vertex via an adjacency matrix, it treats the center and the neighboring vertices equally, whereas in reality, their roles in physically-based simulation are distinct.

GNS [23]: The recently proposed GNS [23] architecture is also a graph network designed for particle systems. The model first separately encodes node features and edge features in the graph and then generalizes the GraphNet blocks in [24] to pass messages across the graph. Finally, a decoder is used to extract the prediction target from the GraphNet block output. The original paper embeds the particles in a graph by adding edges between vertices under a given radius threshold. In our implementation, we instead utilized the mesh topology to construct the graph. We used two blocks in the "processor" [23] to achieve a network capacity similar to ours. In contrast to CFD-GCN [5], the GraphNet block can represent the interaction between the nodes and edges more efficiently, resulting in a significant performance improvement in rollout prediction settings. However, we still observe mesh explosions after a few frames, as shown in Figure 7 and in the supplementary video.

MeshGraphNets [21]: In concurrent work to us, MeshGraphNets [21] were presented for physically-based simulation on a mesh, with an architecture similar to GNS [23]. The Lagrangian cloth system presented in their paper is the most closely related approach to our work. Therefore, we

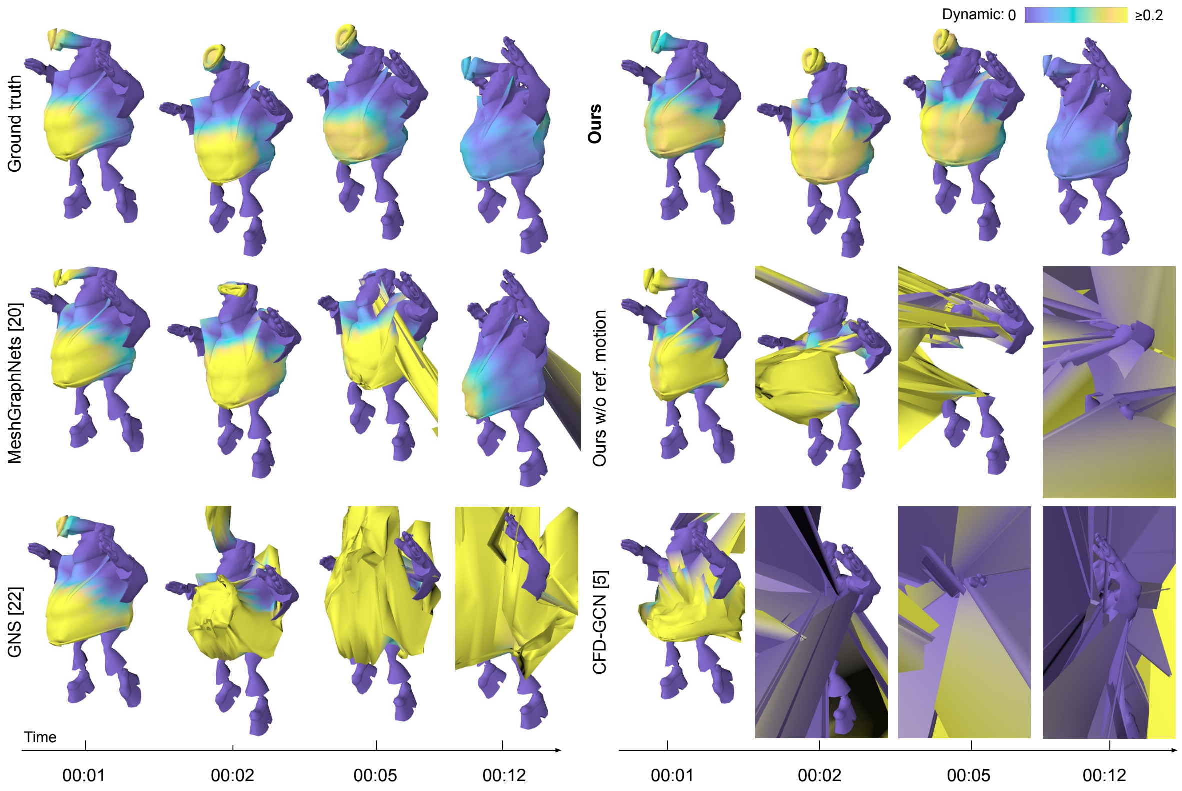


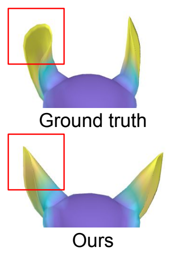
Figure 7: The rollout prediction results of our method and others tested on the Big Vegas character with 283-frame hip hop dancing motion. The baseline cannot be rendered because it explodes.

followed the input formulation of their example, except that we used the reference mesh to represent the undeformed mesh space as the edge feature. In our implementation, we keep the originally proposed encoders ϵ^M and ϵ^V that embed the edge and node features, but exclude the global (world) feature encoder ϵ^W, because it is not applicable to our problem setting. Similarly, we kept the MLPs f^M and f^V, but removed the f^W inside the graph block. We used 15 graph blocks, as suggested by their paper. The network has 10 times more parameters than ours; 2,333,187 parameters compared to our 237,571 parameters. Training lasted for 11 days, whereas our network was trained in less than a day.

We report how MeshGraphNets perform on our test character motions in Table 2. The overall average rollout RMSE of MeshGraphNets is worse than GNS [23]. Nevertheless, we note that out of 15 motions, this approach achieved 5 stable rollout predictions without explosions, while GNS [23] failed on all of them. Our method outperforms each of the compared methods with respect to the investigated metrics.

5. Conclusion

We proposed a *Deep Emulator* for enhancing skinning-based animations of 3D characters with vivid secondary motion. Our method is inspired by the underlying physical simulation. Specifically, we train a neural network that operates on a local patch of a volumetric simulation mesh of the character, and predicts the updated vertex positions from the current acceleration, velocity, and positions. Being a local method, our network generalizes across 3D character meshes of arbitrary topology.



While our method demonstrates plausible secondary dynamics for various 3D characters under complex motions, there are still certain limitations we would like to address in future work. Specifically, we demonstrated that our network trained on a dataset of a volumetric mesh of a sphere can generalize to 3D characters with varying topologies. However, if the local geometric detail of a character is significantly different to those seen during training, e.g., the ears of the mousey character containing many local neighborhood not present in the sphere training data, the quality of our output decreases. One potential avenue for addressing this is to add additional primitive types to training, beyond tetrahedralized spheres. A thorough study on the type of training primitives and motion sequences required to cover the underlying problem domain is an interesting future direction.

References

[1] Stephen W Bailey, Dave Otte, Paul Dilorenzo, and James F O'Brien. Fast and deep deformation approximations. *ACM Transactions on Graphics (TOG)*, 37(4):1–12, 2018. 1, 2

[2] David Baraff and Andrew Witkin. Physically based modeling: Constrained dynamics. *SIGGRAPH 2001 Course Notes: Physically Based Modeling*, 2001. 3

[3] David Baraff and Andrew P. Witkin. Large Steps in Cloth Simulation. In *Proc. of ACM SIGGRAPH 98*, pages 43–54, July 1998. 4

[4] Jernej Barbič, Fun Shing Sin, and Daniel Schroeder. Vega FEM Library, 2012. http://www.jernejbarbic.com/vega. 5, 6

[5] Filipe de Avila Belbute-Peres, Thomas D Economon, and J Zico Kolter. Combining differentiable pde solvers and graph neural networks for fluid flow prediction. In *Proceedings of the 37th International Conference on Machine Learning ICML*, volume 2020, 2020. 3, 7

[6] Congyue Deng, Tai-Jiang Mu, and Shi-Min Hu. Alternating convlstm: Learning force propagation with alternate state updates. *arXiv preprint arXiv:2006.07818*, 2020. 2

[7] Lawson Fulton, Vismay Modi, David Duvenaud, David IW Levin, and Alec Jacobson. Latent-space dynamics for reduced deformable simulation. In *Computer graphics forum*, volume 38, pages 379–391, 2019. 1, 2

[8] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proc. of SIGGRAPH 1998*, pages 9–20, 1998. 2

[9] Fabian Hahn, Sebastian Martin, Bernhard Thomaszewski, Robert Sumner, Stelian Coros, and Markus Gross. Rig-space physics. *ACM Trans. on Graphics (TOG)*, 31(4):1–8, 2012. 2

[10] Fabian Hahn, Bernhard Thomaszewski, Stelian Coros, Robert W Sumner, and Markus Gross. Efficient simulation of secondary motion in rig-space. In *Proc. of Symp. on Computer Animation (SCA)*, pages 165–171, 2013. 2

[11] Daniel Holden, Bang Chi Duong, Sayantan Datta, and Derek Nowrouzezahrai. Subspace neural physics: fast data-driven interactive simulation. In *Proceedings of the 18th annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 1–12, 2019. 1, 2

[12] Adobe Inc. Mixamo. https://www.mixamo.com. 5

[13] Alec Jacobson, Zhigang Deng, Ladislav Kavan, and JP Lewis. Skinning: Real-time shape deformation. In *ACM SIGGRAPH 2014 Courses*, 2014. 3

[14] Doug L. James, Jernej Barbič, and Christopher D. Twigg. Squashing Cubes: Automating Deformable Model Construction for Graphics. In *Proc. of ACM SIGGRAPH Sketches and Applications*, Aug. 2004. 3

[15] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 7

[16] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *ICLR*, 2019. 3

[17] Libin Liu, KangKang Yin, Bin Wang, and Baining Guo. Simulation and control of skeleton-driven soft body characters. *ACM Transactions on Graphics (TOG)*, 32(6):1–8, 2013. 2

[18] Ran Luo, Tianjia Shao, Huamin Wang, Weiwei Xu, Xiang Chen, Kun Zhou, and Yin Yang. Nnwarp: Neural network-based nonlinear deformation. *IEEE Trans. on Visualization and Computer Graphics*, 2018. 1, 2

[19] Aleka McAdams, Yongning Zhu, Andrew Selle, Mark Empey, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. Efficient elasticity for character skinning with contact and collisions. In *ACM SIGGRAPH 2011*, pages 1–12. 2011. 2

[20] Felix Meister, Tiziano Passerini, Viorel Mihalef, Ahmet Tuysuzoglu, Andreas Maier, and Tommaso Mansi. Deep learning acceleration of total lagrangian explicit dynamics for soft tissue mechanics. *Computer Methods in Applied Mechanics and Engineering*, 358:112628, 2020. 1, 2

[21] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020. 3, 7

[22] Cristian Romero, Miguel A. Otaduy, Dan Casas, and Jesus Perez. Modeling and estimation of nonlinear skin mechanics for animated avatars. *Computer Graphics Forum (Proc. of Eurographics)*, 39(2), 2020. 2

[23] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W Battaglia. Learning to simulate complex physics with graph networks. *arXiv preprint arXiv:2002.09405*, 2020. 3, 7, 8

[24] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. volume 80 of *Proceedings of Machine Learning Research*, pages 4470–4479, 2018. 7

[25] Igor Santesteban, Elena Garces, Miguel A. Otaduy, and Dan Casas. SoftSMPL: Data-driven Modeling of Nonlinear Soft-tissue Dynamics for Parametric Humans. *Computer Graphics Forum (Proc. Eurographics)*, 2020. 1, 2, 3

[26] Xiaohan Shi, Kun Zhou, Yiying Tong, Mathieu Desbrun, Hujun Bao, and Baining Guo. Example-based dynamic skinning in real time. *ACM Trans. on Graphics (TOG)*, 27(3):1–8, 2008. 2

[27] Fun Shing Sin, Daniel Schroeder, and Jernej Barbič. Vega: non-linear fem deformable object simulator. In *Computer Graphics Forum*, volume 32, pages 36–48, 2013. 5, 6

[28] Qingyang Tan, Zherong Pan, Lin Gao, and Dinesh Manocha. Realtime simulation of thin-shell deformable materials using cnn-based mesh embedding. *IEEE Robotics and Automation Letters*, 5(2):2325–2332, 2020. 2

[29] Benjamin Ummenhofer, Lukas Prantl, Nils Thuerey, and Vladlen Koltun. Lagrangian fluid simulation with continuous convolutions. In *International Conference on Learning Representations*, 2019. 3

[30] Bohan Wang, Mianlun Zheng, and Jernej Barbič. Adjustable constrained soft-tissue dynamics. *Pacific Graphics 2020 and Computer Graphics Forum*, 39(7), 2020. 2

[31] Jiayi Eris Zhang, Seungbae Bang, David I.W. Levin, and Alec Jacobson. Complementary dynamics. *ACM Trans. on Graphics*, 2020. 2