Real-Time Neural Cloth Deformation using a Compact Latent Space and a Latent Vector Predictor

Chanhaeng Lee

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Computer Science (Computer Science) at

Concordia University

Montréal, Québec, Canada

September 2024

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Chanhaeng Lee**

Entitled: **Real-Time Neural Cloth Deformation using a Compact Latent Space and a Latent Vector Predictor**

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Sudhir Mudur

_____ Examiner
Dr. Sudhir Mudur

_____ Examiner
Dr. Marta Kersten-Oertel

_____ Supervisor
Dr. Tiberiu Popa

Approved by _____
Joey Paquet, Chair
Department of Computer Science and Software Engineering

_____ 2024

_____
Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Real-Time Neural Cloth Deformation using a Compact Latent Space and a Latent Vector Predictor

Chanhaeng Lee

We propose a method for real-time cloth deformation using neural networks, especially for draping a garment on a human body. The computational overhead of most of the existing learning methods for cloth deformation often limits their use in interactive applications. Employing a two-stage training process, our method predicts garment deformations in real-time. In the first stage, a graph neural network extracts cloth vertex features which are compressed into a latent vector with a mesh convolution network. We then decode the latent vector to blend shape weights, which are fed to a trainable blend shape module. In the second stage, we freeze the latent extraction and train a latent predictor network. The predictor uses a subset of the inputs from the first stage, ensuring that inputs are restricted to those which are readily available in a typical game engine. Then, during inference, the latent predictor predicts the compacted latent which is processed by the decoder and blend shape networks from the first stage. Our experiments demonstrate that our method effectively balances computational efficiency and realistic cloth deformation, making it suitable for real-time use in applications such as games.

Acknowledgments

I am deeply grateful to my supervisor, Dr. Tiberiu Popa, whose guidance and support throughout my master's degree have been crucial to my growth as a researcher in this field. I would like to extend my thanks to my academic advisors, Dr. Sudhir Mudur and Dr. Eric Paquette, as well as my industry advisor, Dr. Saeed Ghorbani, for their insightful feedback and constant encouragement. Additionally, I would like to acknowledge my coworker, Maksym Perepichka, for his invaluable assistance in navigating my research journey and for our conversations toward better research results. I would like to reiterate my appreciation for all his support throughout my master's degree. I am also grateful to everyone at Ubisoft La Forge for providing me with the opportunity to interact with exceptional coworkers and for the resources that contributed to the completion of my research.

I would like to express my deepest gratitude to my family and friends. I could not have completed my research without their support and encouragement. I am deeply thankful to my parents for their constant belief in me and their solid support. I am also grateful to my relatives and friends for their constant companionship, which has brought joy and laughter into my life. Finally, I want to express my indescribable gratitude to my soon-to-be wife, Trieu. Thank you for standing by my side through all the ups and downs.

Contents

List of Figures

Chapter 1

Introduction

Modeling, synthesizing and rendering clothing on virtual characters is a critical task in many applications such as games, special effects, telepresence, and VR environments. Physical simulation has been typically used with excellent results [2, 3], but with significant limitations related especially to performance, stability, and controllability [3].

Modern video games contain many complex components related to graphics and animation working together with a limited computational budget and with a high number of assets (i.e. characters, garment types, accessories, motion types). Therefore, garment synthesis methods aimed at games require, in addition to the high quality of the results, high performance as well as a high degree of generalization to adapt to the large number of assets that typically appear in a game.

Most used methods in games predict displacement on a template garment in the canonical pose and subsequently drive cloth deformations using a combination of blend shapes and Linear Blend Skinning (LBS) [4, 5]. However, as noted by Grigorev et al. [6], this pose-driven deformation has difficulties both in correctly representing loose garments and with the dynamic behaviour of clothing. This is because pose-driven methods are not supervised by any real physics simulation. The HOOD method [6] tries to address these issues by learning complex cloth deformations using direct physics supervision at a vertex level and, as such, is agnostic to the underlying body. One major concern with HOOD though is that it is far too slow to be used in video games.

In this work, we propose a garment synthesis method specifically designed with game requirements in mind. Our method uses the blend shapes and LBS methods while producing good realistic

cloth deformation in real time. Our method uses two key ideas. First, we distil the per-vertex feature vector obtained from a powerful but slower method such as HOOD [6] that encodes the complex information about the garment deformation to a per-garment compact latent space that can be efficiently decoded. Then we compute the posed garment from the compact latent space. This addresses only half of the problem: even if the decoding is efficient, the encoding based on the graph embedding and message passing in HOOD is quite slow. To address this issue, our second key idea is to create a fast latent predictor that computes the latent code of the next frame of the deformation. We demonstrate that this approach produces high-quality results at very fast speeds. Moreover, it generalizes over different body shapes thus allowing only one latent space for a variety of character shapes and sizes. We further demonstrate its suitability for game-like applications by showing a real-time demo where the body poses are generated ad-hoc and in real-time using motion matching [7, 8], a common method used in games for character pose synthesis. Our main contributions can be summarized as:

- a network architecture for effective distillation of per-vertex features to a compact latent space.
- an efficient latent predictor for real-time purposes relying on a compact set of inputs readily available in game engines.
- a two-stage training strategy to achieve quality and speed in computing cloth dynamics.

Chapter 2

Background

This chapter provides the necessary background for the method presented in this thesis. It includes techniques for animating meshes using linear blend skinning, cloth simulation, Graph Network-based Simulator, and unsupervised learning method for garment synthesis and simulation.

2.1 Animating meshes

Garments are draped on top of animated human bodies, both of which are represented as meshes composed of vertices and faces. Vertices are three-dimensional coordinates in the Cartesian coordinate system, while faces are indices referencing these vertices, with each face typically forming a triangle of three indices. Faces are primarily used for rendering purposes.

To animate a mesh, we need a data representation for the animation sequence. This representation can vary depending on the algorithm used. In this work, we focus on the representation for Skeletal-Subspace Deformation, especially linear blend skinning. Additionally, we explain the 6D rotation representation used for our neural networks, introduced by Zhou et al. [9]. We also discuss the Skinned Multi-Person Linear (SMPL) model, which is employed to represent different shapes of human bodies. This model is particularly useful for representing various body shapes with a small set of parameters. We also cover blend shapes and pose space deformation, which are central to both SMPL and our method.

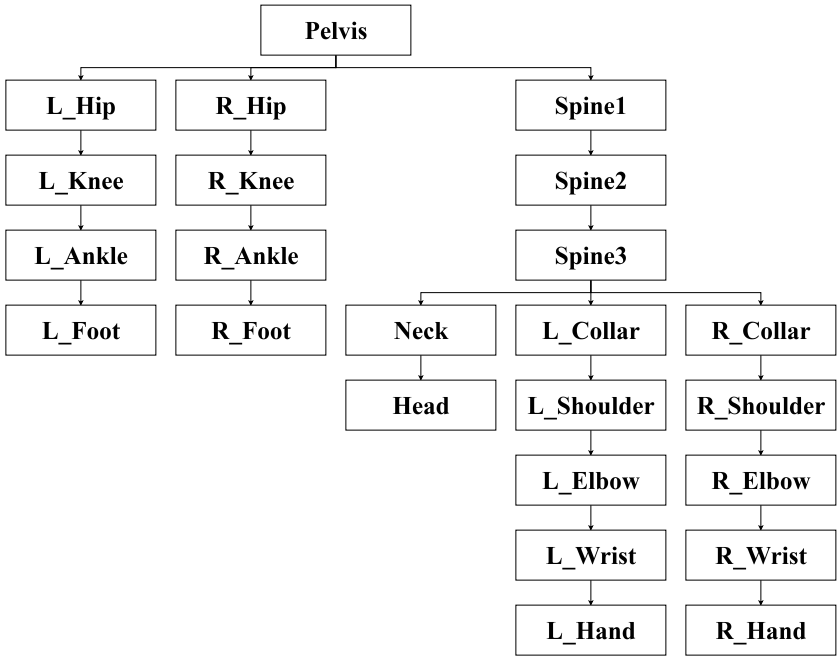


Figure 1: Joint Hierarchy for SMPL. L and R mean left and right, respectively. The parent joints point to their children with the arrows.

2.1.1 Linear Blend Skinning And Representations for Animation Sequence

In character animation, Skeleton-Subspace Deformation (SSD) method, also known as skinning or enveloping, is employed to animate a mesh [5]. Derived from SSD, Linear Blending Skinning (LBS) is defined as $LBS(\mathbf{T}, \mathbf{J}, \boldsymbol{\theta}, \mathcal{W})$ with vertices in the canonical pose $\mathbf{T} \in \mathbb{R}^{N\times 3}$, joint locations $\mathbf{J} \in \mathbb{R}^{K\times 3}$, joint rotations $\boldsymbol{\theta} \in \mathbb{R}^{K\times 3\times 3}$ for a frame of an animation sequence, and skinning weight matrix $\mathcal{W} \in \mathbb{R}^{N\times K}$ with N vertices and K joints. The LBS equation outputs the posed vertices $\mathbf{v} \in \mathbb{R}^{N\times 3}$. The posed vertices $\mathbf{v}$ is further translated with the root joint translation from the animation sequence. Therefore, an animation sequence for LBS over F frames is represented as joint rotation matrices $\mathbf{JR} \in \mathbb{R}^{F\times K\times 3\times 3}$ and root joint translations $\mathbf{RT} \in \mathbb{R}^{F\times 3}$. Thus, the final animated mesh for the frame f is evaluated by $LBS(\mathbf{T}, \mathbf{J}, \mathbf{JR}_f, \mathcal{W}) + \mathbf{RT}_f$.

LBS requires a skeleton with joints in a hierarchy for a mesh. One example for the joint hierarchy from SMPL, detailed in Sec. 2.1.2, is illustrated in Fig. 1. Each vertex in the mesh is linearly transformed from the canonical space to the posed space by joint transformation matrices with weights through LBS. These weights, called skinning weights, influence vertices linearly.

Accordingly, each transformed vertex $\mathbf{v}_i$ from the LBS is defined as:

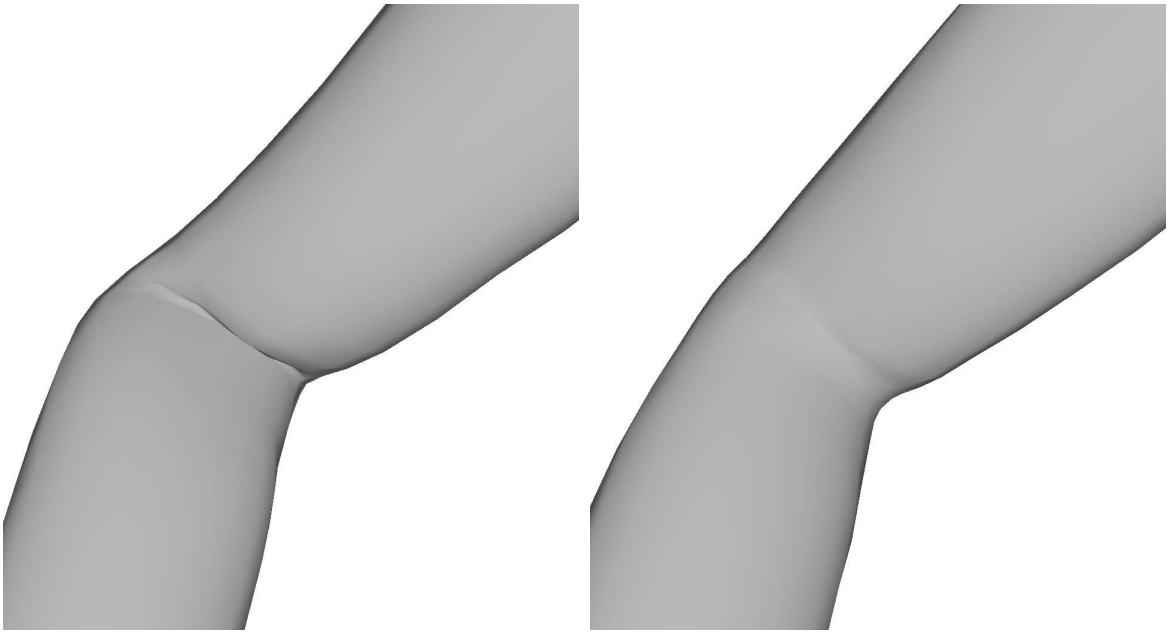
$$\mathbf{v}_i = \sum_{k}^{K} w_{i,k} L_k \mathbf{p}_i, \tag{1}$$

where $w_{i,k}$ is the kth joint (skinning) weight for the ith vertex from the skinning weight matrix $\mathcal{W}$, $L_k \in \mathbb{R}^{4\times4}$ is the kth joint global transformation matrix for the ith vertex, and $\mathbf{p}_i$ is the homogeneous coordinates for the ith vertex of the mesh in the canonical pose from the template vertices $\mathbf{T}_i$. The joint global transformation matrices are evaluated by forward kinematics, traversing the joint hierarchy from the root joint to children joints in the hierarchy. The forward kinematics enables the transformation from the joint space to the world space. Thus, $\boldsymbol{\theta}$ and $\mathbf{J}$ are joint local rotations and locations relative to the reference frame of their parent joint. One transformation matrix for each joint, usually called the inverse bind matrix, is further required to transform vertices of the mesh from its canonical space to the joint space. The inverse bind matrix is constructed by joint local rotations in the canonical pose. The final equation for each joint global transformation matrix is defined as:

$$L_k(\boldsymbol{\theta}, \mathbf{J}) = \prod_{j\in A(k)} \left[\begin{array}{c|c} \boldsymbol{\theta}_j & \mathbf{J}_j \\ \hline \mathbf{0} & 1 \end{array}\right] \left(\prod_{j\in A(k)} \left[\begin{array}{c|c} \boldsymbol{\theta}^*_j & \mathbf{J}_j \\ \hline \mathbf{0} & 1 \end{array}\right] \right)^{-1}, \tag{2}$$

where $\boldsymbol{\theta}_j \in \mathbb{R}^{3\times3}$ is a joint local rotation of a frame in an animation sequence, $\mathbf{J}_j \in \mathbb{R}^3$ is a joint local location, $\boldsymbol{\theta}^*_j \in \mathbb{R}^{3\times3}$ is a joint local rotation in the canonical pose, and $A(k)$ is the ordered list of joint ancestors of the joint k.

Since rotations can be represented in various ways, in addition to choosing an animation representation for LBS, selecting an effective animation representation for training neural networks has become important. Zhou et al. [9] investigated the continuity of many representations for 3D rotations. There are several representations for 3D rotations, such as Euler angles, quaternions, and axis-angle representations. However, these representations are discontinuous according to Zhou et al. [9]. A possible alternative for the continuous representation of 3D rotation is the 6D representation. Specifically, the mapping function from 3D rotations to the 6D representation is defined



(a) LBS **(b)** SMPL

Figure 2: (**a**) Failure case of LBS where the correct deformation does not belong to the subspace formed by joint transformations and skinning weights. (**b**) The SMPL model [1] addresses this limitation using blend shapes based on Pose Space Deformation (PSD).

as:

$$g\left(\begin{bmatrix} | & | & | \\ a_1 & a_2 & a_3 \\ | & | & | \end{bmatrix}\right) = \begin{bmatrix} | & | \\ a_1 & a_2 \\ | & | \end{bmatrix}, \tag{3}$$

where a_1, a_2, and a_3 are the column vectors for a 3D rotation matrix and the function g is the continuous representation of 3D rotations by taking the first two columns out of three columns from the 3D rotation matrix. This function is validated to have continuous representation for 3D rotations, since the output of the function g can be remapped to the original space for the 3D rotation matrix by the Gram-Schmidt process and the cross product. The experiments by Zhou et al. [9] showed that this representation is more effective than any other representations for training neural networks.

2.1.2 Blend Shapes, Pose Space Deformation, and SMPL

In addition to LBS, blend shapes are a popular technique for further animating mesh vertices. Blend shapes, also known as shape interpolation or morphing, are used in various ways to achieve

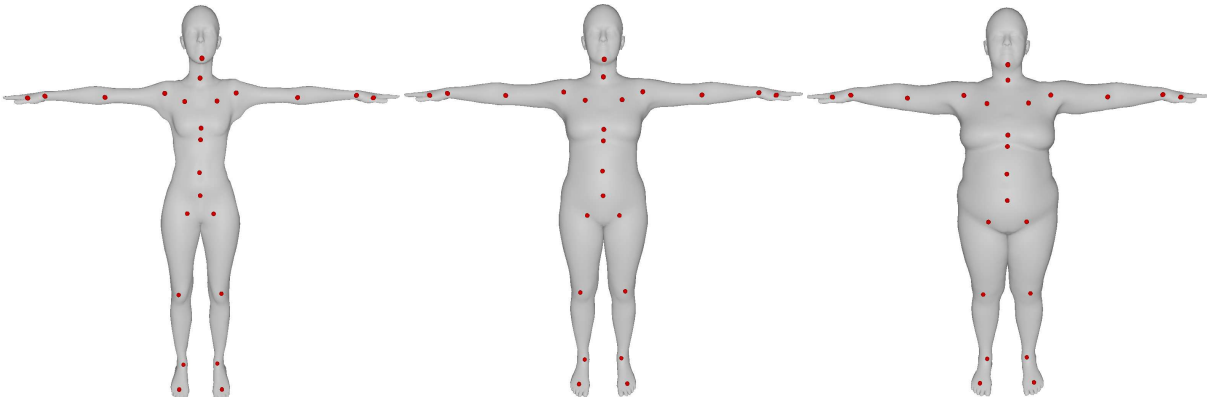
desired animation results. The blend shapes are defined as

$$\sum_{b}^{B} \beta_b \mathbf{S}_b \tag{4}$$

where B is the number of shapes and the scalar β_b weights a blend shape $\mathbf{S}_b \in \mathbb{R}^{N\times 3}$. This linear combination deforms the template vertices $\mathbf{T}$ to achieve desired animation results, such as facial animation, when combined with LBS.

One significant limitation of LBS is invalid deformation in some parts of a mesh, where the correct deformation does not belong to the subspace constructed by joint transformations and skinning weights [5], as illustrated in Fig. 2a. Since the desired animated mesh cannot always be achieved through LBS with blend shapes, the Pose Space Deformation (PSD) method supplements LBS. The PSD model takes poses as an input and computes vertex displacement relative to the mesh vertices in the canonical pose using data interpolation methods such as radial basis functions [5]. Then, LBS is applied to the deformed vertices to achieve desired results, such as better deformations. The PSD function is modeled with sufficient examples of animated meshes and poses to find desired vertex displacements.

Capitalizing on these two techniques, the Skinned Multi-Person Linear (SMPL) model extends one template body to different shapes and addresses the limitation of LBS through its learning method [1]. The core of SMPL is additive and corrective blend shapes for its template body. Its shape blend shapes $\mathbf{B}_S(\boldsymbol{\beta})$ represent different body shapes with 10 scalar parameters of $\boldsymbol{\beta}$, and its pose blend shapes $\mathbf{B}_P(\boldsymbol{\theta})$ correct artifacts from LBS based on the pose $\boldsymbol{\theta}$, as shown in Fig. 2b. The SMPL model also regresses joint locations for different body shapes with its joint regressor $\mathcal{J}(\boldsymbol{\beta})$. A visual representation of different body shapes with joint locations regressed by the joint regressor is shown in Fig. 3. The pose blend shapes can be considered one of the PSD methods since they take a pose as input and return vertex displacement for the template vertices. However, the pose blend shapes differ from conventional PSD methods in that the pose blend shapes are a linear model without using a data interpolation method. Recent literature in deep learning often refers to methods as PSD if they take a pose as input and output vertex displacement for template vertices. In conclusion, the SMPL model with LBS is defined as $LBS(\mathbf{T} + \mathbf{B}_S(\boldsymbol{\beta}) + \mathbf{B}_P(\boldsymbol{\theta}), \mathcal{J}(\boldsymbol{\beta}), \boldsymbol{\theta}, \mathcal{W})$.



(a) Thin **(b)** Template **(c)** Heavy

Figure 3: Different body shapes of the SMPL model with joint locations in red, including the thin body **(a)**, the template body **(b)**, and the heavy body **(c)**.

2.2 Cloth Simulation

While LBS enables cloth synthesis on human bodies by animating meshes with fast runtime performance, physically-based cloth simulation offers more realistic deformation of cloth at a slower runtime performance compared to LBS. Pioneering work by Baraff and Witkin [2] introduces an implicit time integration method for cloth simulation, also known as the backward Euler method. This approach is fundamental in achieving stable and realistic physics-based simulation, even with large time steps. The discretized backward Euler equation is defined as

$$\frac{d}{dt}\begin{pmatrix}\mathbf{x}\\ \mathbf{v}\end{pmatrix} = \begin{pmatrix}\mathbf{v}\\ \mathbf{M}^{-1}\mathbf{f}(\mathbf{x},\mathbf{v})\end{pmatrix}, \tag{5}$$

$$\begin{pmatrix}\mathbf{x}_{n+1}-\mathbf{x}_n\\ \mathbf{v}_{n+1}-\mathbf{v}_n\end{pmatrix} = \begin{pmatrix}\Delta\mathbf{x}\\ \Delta\mathbf{v}\end{pmatrix} = \begin{pmatrix}h(\mathbf{v}_n+\Delta\mathbf{v})\\ h(\mathbf{M}^{-1}\mathbf{f}(\mathbf{x}_n+\Delta\mathbf{x},\mathbf{v}_n+\Delta\mathbf{v}))\end{pmatrix}, \tag{6}$$

where Eq. (5) represents the equations of motion with positions $\mathbf{x}$, velocities $\mathbf{v}$, a diagonal mass matrix $\mathbf{M}$, and forces $\mathbf{f}$ for a triangular mesh of particles. The backward Euler method in Eq. (6) defines the changes in positions and velocities in terms of quantities from the next time step with the time step h. Solving the nonlinear equation from Eq. (6) is required to advance cloth simulation.

Many research works on physics-based simulation have adopted the implicit Euler method for simulating various objects, including cloth, and have developed ways to evaluate the solution to

Eq. (6). Baraff and Witkin [2] rather converted the nonlinear equation into a linear equation applying a Taylor series expansion to the forces $\mathbf{f}$, and solved the linear equation with their modified conjugate gradient method. They also mentioned using a direct method such as Gaussian elimination to solve small linear systems. Goldenthal et al. [10] demonstrated that solving the nonlinear equation of the implicit Euler method is equivalent to an optimization problem. Martin et al. [11] also used numerical optimization to solve the nonlinear equation for their example based simulation, calling it the variational Euler implicit function. Using the optimization method, Liu et al. [12] showed cloth simulation based on mass-spring systems. Gast et al. [13] validated that recasting the solution of the nonlinear equation as an optimization problem improves the robustness of results using optimization techniques. Recasting Eq. (6) as an optimization problem has become the crux of physics-based simulation.

To recast the backward Euler nonlinear equation as an optimization problem, $\mathbf{v}_{n+1}$ is eliminated using $\frac{\mathbf{x}_{n+1}-\mathbf{x}_n}{h} = \mathbf{v}_{n+1} = \mathbf{v}_n + \Delta\mathbf{v}_n$:

$$\frac{\mathbf{x}_{n+1} - \mathbf{x}_n - h\mathbf{v}_n}{h} = h(\mathbf{M}^{-1}\mathbf{f}(\mathbf{x}_{n+1}, \frac{\mathbf{x}_{n+1} - \mathbf{x}_n}{h})). \tag{7}$$

Rearranging Eq. (7), we obtain:

$$\mathbf{M}\frac{\mathbf{x}_{n+1} - \mathbf{x}_n - h\mathbf{v}_n}{h^2} = \mathbf{f}(\mathbf{x}_{n+1}, \frac{\mathbf{x}_{n+1} - \mathbf{x}_n}{h}). \tag{8}$$

Newton's method typically evaluates the solution $\mathbf{x}_{n+1}$ with:

$$\mathbf{h}(\mathbf{x}_{n+1}) = \mathbf{M}\frac{\mathbf{x}_{n+1} - \mathbf{x}_n - h\mathbf{v}_n}{h^2} - \mathbf{f}(\mathbf{x}_{n+1}, \frac{\mathbf{x}_{n+1} - \mathbf{x}_n}{h}), \tag{9}$$

where $\mathbf{h}(\mathbf{x}_{n+1}) = \mathbf{0}$ at the root. From Eq. (9), we rewrite the equation $\mathbf{h}(\mathbf{x}_{n+1})$ to represent it as a minimization problem:

$$\mathbf{h}(\mathbf{x}_{n+1}) = \mathbf{M}\frac{\mathbf{x}_{n+1} - \mathbf{x}_n - h\mathbf{v}_n}{h^2} + \frac{\partial\Phi}{\partial\mathbf{x}}, \tag{10}$$

under the assumption that the (conservative) forces $\mathbf{f}$ can be represented as derivatives of potential

energies Φ [13]. Integrating Eq. (10), we obtain the scalar objective function to minimize:

$$E(\mathbf{x}_{n+1}) = \frac{1}{2h^2}(\mathbf{x}_{n+1} - \mathbf{x}_n - h\mathbf{v}_n)^T\mathbf{M}(\mathbf{x}_{n+1} - \mathbf{x}_n - h\mathbf{v}_n) + \Phi, \tag{11}$$

where the derivative of $E(\mathbf{x}_{n+1})$ at a minimizer $\mathbf{x}^*_{n+1}$ is zero when evaluating a local minimum for $E(\mathbf{x}_{n+1})$ [13]:

$$\frac{\partial E(\mathbf{x}_{n+1})}{\partial \mathbf{x}} = \mathbf{h}(\mathbf{x}_{n+1}) = \mathbf{0}. \tag{12}$$

Therefore, finding a local minimum for $E(\mathbf{x}_{n+1})$ is equivalent to solving $\mathbf{h}(\mathbf{x}_{n+1})$ according to the first-order optimality condition [13, 14].

This optimization-based approach has not only advanced physics-based simulation, but also impacted deep learning for cloth simulation and synthesis. Since a deep learning framework focuses on minimizing losses, the minimization for $E(\mathbf{x}_{n+1})$ can be used similarly to enforce physical behaviors by neural networks, even without ground-truth training data [15]. Further details on this application are provided in Sec. 2.4.

2.3 Graph Network-based Simulator

Recent efforts in learning physics simulation through deep learning frameworks have shown promising results. Sanchez-Gonzalez et al. [16] introduced Graph Network-based Simulators (GNS), $d_\theta : \mathcal{X} \to \mathcal{Y}$, which predict dynamics information $\mathcal{Y}$, such as accelerations, from the state of a world $\mathcal{X}$ using parameters θ. At the core of GNS is the use of graph neural networks with message passing techniques to model interactions between nodes in a graph. GNS performs semi-implicit Euler integration to calculate the next state of particles using the predicted accelerations:

$$\dot{\mathbf{p}}_{n+1} = \dot{\mathbf{p}}_n + h\ddot{\mathbf{p}}_n \tag{13}$$

$$\mathbf{p}_{n+1} = \mathbf{p}_n + h\dot{\mathbf{p}}_{n+1}, \tag{14}$$

where $\mathbf{p}$ and $\dot{\mathbf{p}}$ are the positions and velocities of particles, respectively, and $\ddot{\mathbf{p}}$ are the accelerations predicted by GNS. For simplicity, the time step h is set to 1. GNS is trained with an L_2 loss between

the predicted positions $\mathbf{p}_{n+1}$ and the corresponding ground truth positions $\bar{\mathbf{p}}_{n+1}$.

Pfaff et al. [17] extended the above particle-based GNS to their mesh-based GNS. Both particle-based and mesh-based GNS share an Encode-Process-Decode architecture, where an encoder encodes input states into a graph with latent features, a processor processes interactions of nodes with edges from the graph through message-passing, and a decoder decodes dynamics information from node features to predict future states. The mesh-based GNS uses a mesh $\mathcal{M}$ as an input to its GNS instead of the states of particles $\mathcal{X}$.

Inputs and **Outputs**: Since GNS utilizes a graph representation in its processor, it requires input features for both nodes and edges. The input node features can include positions $\mathbf{x}$, velocities $\dot{\mathbf{x}}$, and node types (e.g. object/obstacle/boundary nodes), depending on the implementations. The input edge features include relative positional displacement $\mathbf{x}_i - \mathbf{x}_j$ and its norm $||\mathbf{x}_i - \mathbf{x}_j||$. Edges are defined by sender indices i and receiver indices j, with each edge connecting two nodes. In particle-based GNS, edges are constucted between particles within a connectivity radius since particles do not have explicit connectivities. In contrast, mesh-based GNS uses the inherent connectivity of the mesh while additionally constructing edges in world space based on spatial proximity. The primary outputs from GNS are mostly node accelerations, but GNS can also predict changes in momentum, density, and pressure, depending on the simulation type.

Encode: The encoder in GNS converts input features into latent features, which correspond to nodes and edges of an input graph to the processor. The encoder typically consists of one Multi-Layer Perceptron (MLP) for node features and another MLP for edge features. In mesh-based GNS, an additional MLP is used for edge features in world space. Specifically, input node features are transformed into latent node features $\mathbf{v}$ using the node MLP, while input edge features from different edgesets (including mesh space edge features $\mathbf{e}_m$ and world space edge features $\mathbf{e}_w$ from the total edgesets $\mathbf{E}$) are separately transformed into mesh space latent edge features $\mathbf{e}_m$ and world space latent edge features $\mathbf{e}_w$ by their respective MLPs. Particle-based GNS handles only a single edgeset. These latent features are then passed to the processor.

Process: The processor in GNS comprises a stack of M graph neural networks with message passing steps, particularly a stack of M Graph Networks (GNs) proposed by Sanchez-Gonzalez et al.

[18]. These GNs are an extension of the interaction network by Battaglia et al. [19]. The processing steps involve first updating edge features and then updating node features, as outlined in Algorithm 1. Specifically, each GN in the stack takes an input graph $G_n = (\mathbf{v}, \mathbf{E})$ and outputs an updated graph $G_{n+1} = (\mathbf{v}', \mathbf{E}')$. Each edgeset has a specific MLP that updates the edge features within the edgeset. Since mesh-based GNS includes an additional edgeset (world space edges), it utilizes two MLPs for updating edge features from both the mesh space and world space. These edge features from edgesets are updated with latent node features gathered via sender and receiver node indices. These updated edge features are then used to update node features by aggregating edge features based on receiver node indices. For mesh-based GNS, the aggregated edge features from different edgesets are concatenated and passed through an MLP to update node features. The number of GNs in the stack (M) influences the accuracy and computational efficiency of the predictions. Pfaff et al. [17] determined that using 15 GNs strikes a balance between computational effciency and predictive accuracy accross various simulations, making it a practical choice for achieving optimal results in most scenarios.

Algorithm 1 Graph Network (GN) for mesh-based Graph Network-based Simulator (GNS)

Input: Graph, $G_n = (\mathbf{v}, \mathbf{E} = \{\mathbf{e}_m, \mathbf{e}_w\})$
Output: Graph, $G_{n+1} = (\mathbf{v}', \mathbf{E}' = \{\mathbf{e}'_m, \mathbf{e}'_w\})$
for each edgeset features $\mathbf{e}$ with a MLP f from $\mathbf{E}$ **do**
    Gather sender node features $\mathbf{v}_s$ and receiver node features $\mathbf{v}_r$ from $\mathbf{v}$ with edge indices
    Update edgeset features $\mathbf{e}' = f_{\mathbf{e}}(\mathbf{e}, \mathbf{v}_s, \mathbf{v}_r)$
end for
for Each node feature $\mathbf{v}_i$ **do**
    $\hat{\mathbf{e}} \leftarrow \emptyset$
    for each edgeset features $\mathbf{e}'$ from $\mathbf{E}$ **do**
        Aggregate edge features with receiver node indices r_j and concatenate
        $\hat{\mathbf{e}} = \hat{\mathbf{e}} \cup \sum_{r_j} \mathbf{e}'_{r_j}$
    end for
    Update node features $\mathbf{v}'_i = f_n(v_i, \hat{\mathbf{e}})$
end for

Decode: Using an MLP, the decoder in GNS predicts dynamics information, such as accelerations, directly from the node features of the final output graph produced by the processor. At this stage, edge features are not utilized, as the focus is on deriving node-specific outputs. The final positions of the nodes are then computed using the predicted accelerations through semi-implicit Euler

integration, as shown in Eq. (13) and Eq. (14).

Particle-based GNS has been successfully applied to a variety of physical domains including fluid dynamics, rigid body simulations, and deformable materials. The extended mesh-based GNS further advanced the field by enabling realistic physics simulation for complex scenarios like cloth behavior, structural mechanics, aerodynamics. The versatility and accuracy of the GNS framework have significantly influenced subsequent research in learning physics-based simulation, including garment synthesis method.

2.4 Unsupervised Learning for Garment Synthesis and Simulation

While supervised learning methods for physics simulation, including cloth simulation, have shown promising results, they depend on high-quality ground-truth simulation data, which can takes hours or even days to generate. In constrast, Bertiche et al. [20] pioneered the use of unsupervised learning for garment draping on human bodies, exploring the potential of learning cloth simulation without relying on ground-truth data. Specifically, their method, PBNS, is a PSD-based cloth synthesis technique that uses LBS and trains networks by applying cloth simulation constraints as losses. While this work yielded promising results, it was limited to static deformation and did not account for dynamic deformation. SNUG by Santesteban et al. [15] advanced this line of research by introducing an inertia term in the loss function, inspired by the reformulation of the implicit Euler method as an optimization problem. This addition enabled SNUG to better capture dynamic behavior in cloth synthesis. However, Bertiche et al. [21] critiqued SNUG for being limited to only three frames of dynamics and proposed an improved method capable of modeling more extensive dynamic behavior.

These PSD-based methods share similar characteristics in their approach to learning static and dynamic garment deformations. One key characteristic is the nature of the inputs and outputs of the neural networks. As these methods are based on PSD, the inputs to the neural networks are primarily sequences of poses, supplemented by additional features that enhance the results, while the outputs are vertex displacements on the garment vertices in the rest pose. Thus, a garment model

for PSD-based unsupervised cloth synthesis is defined with LBS as:

$$\mathbf{x} = LBS(\mathbf{T} + f(\boldsymbol{\sigma}), \mathbf{J}, \boldsymbol{\theta}, \mathcal{W}), \tag{15}$$

where $\mathbf{T}$ represents garment vertices in the rest pose, $f(\boldsymbol{\sigma})$ is the vertex displacement predicted by a neural network f given network inputs $\boldsymbol{\sigma}$, $\mathbf{J}$ is joint locations, $\boldsymbol{\theta}$ is the pose, and $\mathcal{W}$ is the skinning weights. The neural network in this approach can predict vertex displacement directly using an MLP [15]. Alternatively, it can predict blend shape weights, which are then fed into a learnable blend shape module [20, 21]. Li et al. [22] further extended this approach by adding neural network predictions for skinning weights and blend shapes on top of predicting vertex displacement.

Another common characteristic is the enforcement of cloth behavior using constraints for cloth simulation as training losses, similar to reformulating the backward Euler method as an optimization problem. In this context, Eq. (11) is used directly as a total loss for training neural networks in an unsupervised manner. From this equation, the inertia loss is defined as:

$$\mathcal{L}_{\text{inertia}} = \frac{1}{2h^2}(\mathbf{x}_{n+1} - \mathbf{x}_n - h\mathbf{v}_n)^T\mathbf{M}(\mathbf{x}_{n+1} - \mathbf{x}_n - h\mathbf{v}_n), \tag{16}$$

where $\mathbf{x}_{n+1}$ represents the predicted positions in the next frame from Eq. (15), $\mathbf{x}_n$ represents the positions in the current frame, $\mathbf{M}$ is a diagonal mass matrix, $\mathbf{v}_n$ is the velocity in the current frame, and h is the timestep. The potential energies Φ for cloth simulation in Eq. (11) typically include stretching, bending, gravity, and collision constraints. The potential energy for the stretching constraint can be implemented using the mass-spring model, the continuum formulation of Baraff and Witkin [2], or the Saint Venant Kirchhoff (StVK) elastic material model. We refer the reader to SNUG [15] for the implementation of the other loss functions. This training technique without ground-truth data, referred to as physics supervision, is very significant not only in unsupervised cloth synthesis, but also in unsupervised cloth simulation.

In addition to developments in unsupervised cloth synthesis, unsupervised cloth simulation has been explored. Grigorev et al. [6] proposed HOOD that trains their network without ground-truth data, enforcing cloth behavior with the physics supervision to drape a garment on a human body. As HOOD is built on GNS, it predicts accelerations and performs time integration to evaluate the next

frame positions. According to the results from HOOD, this time integration method produces better cloth dynamics with wrinkles compared to unsupervised cloth synthesis methods. However, GNS-based methods, which utilize graph neural networks with message passing, require more intensive computation compared to MLP-based cloth synthesis, limiting their applicability in real-time scenarios.

Given the superior expressiveness of GNS-based methods over PSD-based methods, leveraging the representation from GNS-based methods is a promising approach. However, the high computational cost of graph neural networks makes them less suitable for interactive applications. Therefore, we propose addressing this issue by developing a two-stage learning process to make use of the expressiveness from GNS-based methods and fast computation from PSD-based methods.

Chapter 3

Related Works

In the last decade, an enormous amount of work has been levied toward applying deep learning-based methods to various academic fields including computer graphics and computer animation. Character garment animation has recently received increasing attention for said research. Garment synthesis using neural networks can be largely classified into pose-driven methods where the prediction is conditioned on the body poses and the outputs of the neural networks are displacement for a template garment [23, 15, 24, 25] and/or deformations encoded as blend shapes [26, 20, 22]. These approaches are appealing as the computation of linear blend skinning and blend shapes can be done extremely efficient on GPU.

An early example of displacement based methods is TailorNet [23], which is a supervised method for predicting character clothing using pose, body shape, and garment style as input. The key to the methodology involves explicitly separating low-frequency and high-frequency garment deformations in-order to avoid the common problem of overly smooth output that neural networks suffer from. Low and high-frequency displacement are generated as functions of body shape, pose, and style using a Multi-Layered Perceptron (MLP). The high-frequency displacement is further refined using the mixture weights predicted by the garment style and the body shape. Self-Supervised Neural Dynamic Garments (SNUG) [15] formulates garment physical-based constraints as loss terms which are minimized during training, allowing for the model to learn displacement for dynamic garment deformations in a self-supervised manner removing the need of simulated data to

train. Swish [26] is a quasi-static garment deformer, taking as input the character pose and outputting PCA weights which are used to reconstruct displacement with PCA vectors, similar to blend shapes. This is one of the only two methods that are suited for deployment in games and in fact was used in Electronic Arts Madden NFL 21 to deform football player jerseys. However, this method is limited to tight garments and it does not take into account the dynamics of the cloth. The largest limitation of the LBS-based methods is their failure on loose garments. Zhang et al. [24] address this by learning a generative space of plausible garment geometries. Then, their method learns a mapping to this space to capture the motion-dependent dynamic deformations, conditioned on the previous state of the garment as well as its relative position with respect to the underlying body. SMPLicit [27] is another generative model capable of representing body pose, shape, and clothing geometry. It can represent multiple garment topologies with the same model. This is achieved via a learned implicit function.

Another way to address the failure cases for loose garments is to model the deformed garment using blend shapes [28, 22, 20, 21]. In the training stage, a set of blend shapes is created to span the deformation space of the garment thus improving over the LBS-only methods. Physically-Based Neural Solver (PBNS) [20] uses a self-supervised learning approach for learning garment deformations similar to SNUG [15]. Neural cloth simulation (NCS) [21] utilizes an encoder-decoder neural network architecture which explicitly disentangles static and dynamic cloth deformations.

HOOD [6] departs from the pose-based generation framework to get the initially posed garment and uses physics supervision at the vertex level to drive garment deformation. While the results are impressive, due to its usage of Graph Neural Networks coupled with the necessity of computing per-vertex features at every time step, this approach is ill suited to real-time in-game cloth deformation.

Chapter 4

Method

4.1 Overview

Our method, depicted in Figure 4, predicts displacement on a template garment in the canonical space and uses LBS to perform garment synthesis for computational efficiency and ease of application. Designed for real-time applications, such as games, our method utilizes a two-stage training process to model dynamic cloth behavior. Our key idea is to obtain a compact latent representation for cloth deformation that can be predicted given sparse inputs and converted into displacement on a template garment.

While a method like HOOD [6] can simulate high-quality garment deformation with dynamics using time integration by predicting acceleration, it results in a high-dimensional latent representation and demands significant computational resources at inference time, making it unsuitable for game engines and real-time applications. Specifically, its encoder generates high-dimensional latent features, its processor updates these features with graph neural networks and message passing, and then its decoder predicts acceleration with the processed features. Utilizing the architecture of the encoder and the processor, which are effective at capturing garment deformation and dynamics, our compact latent learning stage aims to train a network that predicts displacement by compressing the high-dimensional latent features from the encoder and the processor into a compact latent vector using Mesh Convolution [29]. Leveraging the compact latent vector, we predict the displacement with our decoder and learnable blend shapes. The decoder predicts blend shape weights from the

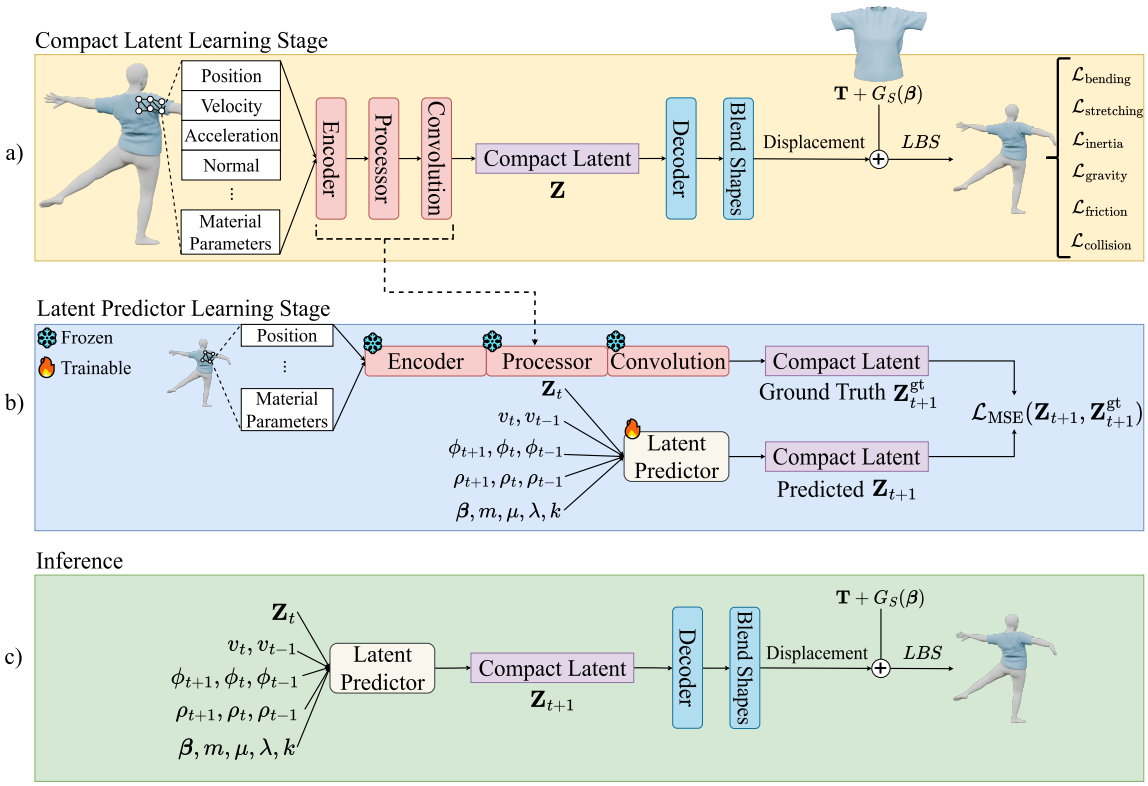


Figure 4: Overview of our method. We develop a compact latent space for cloth deformation in the first stage (a), and then in the second stage (b) we train a latent predictor to efficiently predict a vector in the compact latent space. During inference (c), we utilize lightweight MLPs (the latent predictor and the decoder) and blend shapes for fast inference.

compact latent vector, and then the displacement is computed as a weighted sum of the learnable blend shapes with the blend shape weights. Finally, we perform LBS on the deformed garment with the displacement. Training these networks in an end-to-end manner, as illustrated in Fig. 4 (a), enables us to learn a compact latent space that effectively represents garment deformation and dynamics.

Our latent predictor learning stage trains a lightweight MLP to get rid of the encoder, the processor, and the convolutional neural network of the compact latent learning stage, ensuring fast inference, as shown in Fig. 4 (b). This block is trained to predict the latent vector at the next time step $\mathbf{Z}_{t+1}$ using the current latent vector $\mathbf{Z}_t$, root joint velocities of two frames $\{v_t, v_{t-1}\}$, joint rotations of three frames $\{\phi_{t+1}, \phi_t, \phi_{t-1}\}$, joint positions of three frames $\{\rho_{t+1}, \rho_t, \rho_{t-1}\}$, the body shape parameter $\boldsymbol{\beta}$, and material parameters $\{m, \mu, \lambda, k\}$, which are detailed in Sec. 4.4. These inputs are necessary and sufficient to predict the next step latent vector within interactive frame rates.

During inference time, using this simple MLP with the decoder and the blend shapes enables our method to be applied in real-time applications, as presented in Fig. 4 (c).

4.2 Garment Model

Our garment model $G(\boldsymbol{\beta}, \boldsymbol{\theta}, \mathbf{X})$ is defined as:

$$G(\boldsymbol{\beta}, \boldsymbol{\theta}, \mathbf{X}) = LBS(T(\boldsymbol{\beta}, \boldsymbol{\theta}, \mathbf{X}), J(\boldsymbol{\beta}), \boldsymbol{\theta}, \tilde{\mathcal{W}}) \tag{17}$$

where $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$ are shape and pose parameters used by the SMPL body model [1], $\mathbf{X}$ is a feature vector defined in sections 4.3 and 4.4, and LBS is the linear blend skinning function. This function transforms the deformed garment $T(\boldsymbol{\beta}, \boldsymbol{\theta}, \mathbf{X})$ from the canonical space to the posed space with joint locations $J(\boldsymbol{\beta})$ of the body in shape $\boldsymbol{\beta}$, pose $\boldsymbol{\theta}$, and the diffused skinning weights $\tilde{\mathcal{W}}$.

The core of our garment model is the deformed garment $T(\boldsymbol{\beta}, \boldsymbol{\theta}, \mathbf{X})$. This involves deforming the template garment $\mathbf{T}$ with diffused shape blend-shapes $G_S(\boldsymbol{\beta})$ based on the body shape $\boldsymbol{\beta}$ and further deforming the shaped garment $(\mathbf{T} + G_S(\boldsymbol{\beta}))$ using displacements predicted by network inputs $\mathbf{X}$. The diffusion method for skinning weights $\tilde{\mathcal{W}}$ and shape blend shapes $G_S(\boldsymbol{\beta})$, introduced by Grigorev et al. [6], is necessary due to the challenge posed by the diverse SMPL body shapes. This method allows garments, which are initially designed to fit the template SMPL body, to be aligned with different shapes of the template body, ensuring appropriate deformation.

4.3 Compact Latent Learning Stage

The goal of this learning stage is to train our network Φ and obtain a low-dimensional latent representation for garment deformation and dynamics. Our network Φ follows an Encode-Process-Convolve-Decode architecture, which integrates the Encode-Process architecture [17] of HOOD with a mesh convolution network and our decoder. The network Φ is trained in an unsupervised manner to predict displacement on the shaped garment for the next time step. This is different from HOOD which predicts acceleration for the next time step. With no ground-truth simulation data requirement, we optimize physics-based losses on the final garment vertices $G(\boldsymbol{\beta}, \boldsymbol{\theta}, \mathbf{X}_{H+})$ using

the deformed garment $T(\boldsymbol{\beta}, \boldsymbol{\theta}, \mathbf{X}_{H+})$. The deformed garment in this stage is defined as:

$$T(\boldsymbol{\beta}, \boldsymbol{\theta}, \mathbf{X}_{\text{H+}}) = \mathbf{T} + G_S(\boldsymbol{\beta}) + \Phi(\mathbf{X}_{\text{H+}}), \tag{18}$$

where $\Phi(\mathbf{X}_{\text{H+}})$ is the displacement predicted by our network for the given inputs $\mathbf{X}_{\text{H+}}$.

Inputs $\mathbf{X}_{\text{H+}}$: The network Φ takes inputs $\mathbf{X}_{\text{H+}}$, similar to those in Grigorev et al. [6]. The inputs $\mathbf{X}_{\text{H+}}$ consist of per-vertex and per-edge feature vectors from the garment mesh and the body mesh at time t. The feature vectors for each vertex include velocity, normals, material parameters, vertex type, and vertex level. Material parameters for each vertex consist of mass m, Lamé parameters μ and λ, and the bending coefficient k. The vertex type indicates whether a vertex is pinned to prevent a garment from falling down from a body, and the vertex level represents the coarse level of a vertex (HOOD uses coarsened graphs for its multi-level message passing). Additionally, we incorporate positions and accelerations for garment and body vertices to enhance garment behavior modelling. The feature vectors for each edge of the garment include the relative position of connected vertices at the current time t and in the rest pose, with norms of these relative positions, along with material parameters and delta time. The same per-edge feature vectors are included from coarsened graphs of the garment. For edges connecting body and garment vertices by their proximity, the edge feature vectors include relative positions at the current time t and the next time $t + 1$, with norms and delta time.

Encode and **Process**: The encoder comprises MLPs in the same way as HOOD. It transforms the input feature vectors $\mathbf{X}_{\text{H+}}$ into latent vertex and edge features, which are then processed by a series of message passing blocks in the processor. This processor updates the features to capture garment deformation and dynamics, following the same approach and inputs of latent vertex and edge features as HOOD. Since HOOD passes only the processed vertex features to its decoder after the processor, indicating that the vertex features contain the necessary information for garment deformation and dynamics, we also pass only the processed vertex features to the mesh convolution network without the edge features.

Convolve: We compress the processed vertex features into a compact latent vector $\mathbf{Z}$ with the mesh convolution network by Zhou et al. [29]. This network excels at constructing localized latent

features, making it ideal for our compression task. We utilize its convolution and residual layers to reduce the dimensionality of the processed vertex features and flatten the compressed vertex features into a latent vector. Since the compressed latent vector $\mathbf{Z} \in \mathbb{R}^L$ with the dimension L is used at inference time, the dimension L of the latent vector has to be small enough to be evaluated in interactive frame rates, and large enough to encode garment deformation and dynamics. This latent vector is then passed to the decoder.

Decode: When predicting garment deformations, learnable blend shapes have proven effective in modeling non-linear garment behavior [30, 20, 21]. Following this approach, we build our decoder $f_{\text{dec}} : \mathbb{R}^L \rightarrow \mathbb{R}^D$ with an MLP, which takes the latent vector $\mathbf{Z} \in \mathbb{R}^L$ and predicts D blend shape weights for our learnable blend shapes. The learnable blend shapes consist of D blend shape matrices. Thus, the final displacement for each vertex is defined as:

$$\sum_{j}^{D} \mathbf{D}_{j,i} f_{\text{dec}}(\mathbf{Z})_j, \tag{19}$$

where $\mathbf{D}$ is the array of the learnable blend shape matrices and $\mathbf{D}_{j,i}$ indicates the blend shape basis for the ith garment vertex of the jth blend shape matrix in the array. A single blend shape matrix $\mathbf{D}_j \in \mathbb{R}^{N\times 3}$ consists of blend shape bases for N garment vertices. The combination of the MLP with the blend shapes is fast enough to enable real-time inference.

Loss Functions: We train our network with the same losses as described by Grigorev et al. [6]. We employ the bending loss $\mathcal{L}_{\text{bending}}$ that introduces smoothness by penalizing sharp bends, measured through the dihedral angles between adjacent triangles [2]. The stretching loss $\mathcal{L}_{\text{stretching}}$, based on the Saint Venant-Kirchhoff (StVK) model, enforces hyperelastic material behavior. The inertia loss $\mathcal{L}_{\text{inertia}}$ is used to generate realistic dynamic motion by resisting drastic changes in velocity. The gravity loss $\mathcal{L}_{\text{gravity}}$ applies a constant downward force on the vertices of the garment mesh, creating realistic drapes and falls. The friction loss $\mathcal{L}_{\text{friction}}$ prevents the sliding motion of the garment, enhancing its stability and realism. Since all the above losses can cause interpenetration between the garment and the body, the collision loss $\mathcal{L}_{\text{collision}}$ is used to move the garment vertices away from

the body vertices. Therefore, the total loss $\mathcal{L}$ is defined as a weighted sum of these individual losses:

$$\begin{aligned}\mathcal{L} =& w_b \mathcal{L}_{\text{bending}} + w_s \mathcal{L}_{\text{stretching}} + \\ & w_i \mathcal{L}_{\text{inertia}} + w_g \mathcal{L}_{\text{gravity}} + \\ & w_f \mathcal{L}_{\text{friction}} + w_c \mathcal{L}_{\text{collision}},\end{aligned} \tag{20}$$

where w_b, w_s, w_i, w_g, w_f, and w_c are scalar weights that control the contributions of each loss term. By adjusting these weights, we fine-tune the balance between different aspects of cloth behavior. Following unsupervised training methods with physics-based losses [20, 15, 21, 6], optimizing our network using these losses allows for realistic cloth deformation, similar to solving equations of motion for cloth simulation through energy optimization.

4.4 Latent Predictor Learning Stage

The goal of this stage is to train a latent predictor f_{LP}, which will replace the computationally intensive Encode-Process-Convolve components of the network Φ in the compact latent learning stage.

To ensure that the latent predictor is lightweight and suitable for real-time use, it is constructed solely with an MLP. The latent predictor takes a set of simpler inputs $\mathbf{X}_c$ compared to the inputs $\mathbf{X}_{\text{H+}}$. The inputs $\mathbf{X}_c$ are optimized for instantaneous preparation in each frame while being robust enough to accurately predict the compact latent vector $\mathbf{Z}$. These inputs are defined as $\mathbf{X}_c = \{\mathbf{Z}_t, \boldsymbol{v}, \boldsymbol{\phi}, \boldsymbol{\rho}, \boldsymbol{\beta}, m, \mu, \lambda, k\}$. The latent vector at the current time step $\mathbf{Z}_t$ provides temporal context, aiding in the prediction of the next latent vector. This input latent vector is set to zero in the first frame. The inputs include joint velocities $\boldsymbol{v}$, local joint rotations $\boldsymbol{\phi}$, and global joint positions $\boldsymbol{\rho}$. Specifically, the joint velocities comprise root joint velocities over two frames $\{v_t, v_{t-1}\}$, and the pose consists of local joint rotations over three frames $\{\phi_{t+1}, \phi_t, \phi_{t-1}\}$ in 6D representation [9] and global joint positions over three frames $\{\rho_{t+1}, \rho_t, \rho_{t-1}\}$. We include the global joint positions which are evaluated by forward kinematics with animation sequences, in order to enable the latent predictor to find a better mapping function from the inputs to a latent vector. To prevent the latent predictor from overfitting to the global joint positions of animation sequences, we subtract global

joint positions with the root joint translation and remove the root joint translation from the inputs. We also exclude joints such as wrists and hands from the inputs, due to their minimal impact on garment deformation. We additionally incorporate the body shape $\boldsymbol{\beta}$ and garment material parameters. The material parameters are mass m, Lamé parameters μ and λ, and the bending coefficient k, which are also used for the compact latent learning stage. Through these inputs, we can reduce the number of variables in $\mathbf{X}_{\text{H+}}$ from hundreds of thousands of parameters down to thousands in $\mathbf{X}_c$. For example, we reduce 886,165 floating-point variables in $\mathbf{X}_{\text{H+}}$ to 2,591 floating-point variables in $\mathbf{X}_c$ for a t-shirt garment in our training dataset.

Utilizing these inputs, the latent predictor predicts the next latent vector $\mathbf{Z}_{t+1} = f_{\text{LP}}(\mathbf{X}_c)$. To optimize the latent predictor, we compute the ground truth latent vector $\mathbf{Z}^{\text{gt}}_{t+1}$ using the Encode-Process-Convolve components with the inputs $\mathbf{X}_{\text{H+}}$, while keeping the weights of the components frozen. The optimization is performed by minimizing the mean squared error loss between the predicted latent vector and the ground truth:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{L}\sum_{i}^{L}\left(\mathbf{Z}^{\text{gt}}_{t+1,i} - \mathbf{Z}_{t+1,i}\right)^2 \tag{21}$$

During inference, our garment model $G(\boldsymbol{\beta}, \boldsymbol{\theta}, \mathbf{X}_c)$ gets the final transformed vertices using the deformed garment $T(\boldsymbol{\beta}, \boldsymbol{\theta}, \mathbf{X}_c)$. The deformed garment at inference time is defined as:

$$T(\boldsymbol{\beta}, \boldsymbol{\theta}, \mathbf{X}_c) = \mathbf{T} + G_S(\boldsymbol{\beta}) + \Psi(\mathbf{X}_c), \tag{22}$$

where the function Ψ predicts the displacement on the shaped garment using the latent predictor, the decoder, and the learnable blend shapes with the given inputs $\mathbf{X}_c$. The displacement for each vertex is computed as $\sum_j^D \mathbf{D}_{j,i} f_{\text{dec}}(f_{\text{LP}}(\mathbf{X}_c))_j$.

4.5 Implementations

The network Φ in our compact latent learning stage consists of the encoder, the processor, the mesh convolution network, and the decoder with the learnable blend shapes.

The encoder and processor are constructed similarly to those in HOOD [6]. The encoder, which

has six MLPs, encodes input vertex and edge features into latent features. These input features include garment and body vertex feature vectors, garment edge feature vectors, edge feature vectors from three coarsened graphs of the garment, and features for edges connecting body and garment vertices by their proximity. The processor consists of 15 message-passing steps, each of which includes an MLP for latent vertex features and MLPs for latent edge features. The processor includes two down sampling blocks and two up sampling blocks for its hierarchical message passing. Each MLP in the encoder and processor comprises three linear layers and one normalization layer at the end. The linear layers convert the size of input features into 128, with ReLU activation applied to the outputs of the first two linear layers.

Our mesh convolution network consists of four blocks. Each block contains a convolution layer and a residual layer, denoted as a combination of vcDownConv and vdDownRes, according to Zhou et al. [29]. These blocks require graph sampling information to perform down-samplings with the convolution and residual methods. We selected remaining vertices for the graph sampling information of each garment in the training dataset, using a stride of two and a vertex ring size of two for each down-sampling operation. The number of remaining vertices after the four blocks for each garment normally ranges from 80 to 300. The first three blocks convert input features of 128 dimension into features of 256, 512, and 1024 dimensions, respectively. The last block converts the feature dimension of 1024 into a final dimension size where the product of the number of remaining vertices and the final feature dimension is around 2048. Therefore, the dimension L of a latent vector $\mathbf{Z}$ is around 2048, depending on the number of the remaining vertices. We used 32 weight bases for each block. Unlike the ELU activation function adopted by Zhou et al. [29], we use the ReLU activation function in our mesh convolution network.

The decoder has an MLP with 3 linear layers, converting the dimension of an input latent vector into 1024, 512, and 512, with ReLU activation applied after each layer. The number D of elements in the array of the learnable blend shapes matrices $\mathbf{D}$ is accordingly 512.

Our latent predictor has an MLP with three linear layers, converting the input features into dimensions of 1024, 1024, and the latent vector size L, respectively. We use ReLU activation for the outputs of the first two layers.

Chapter 5

Results and Discussion

In this section, we evaluate the effectiveness and performance of our networks. We first detail our training setup. Next, we qualitatively compare our networks with a state-of-the-art method, NCS [21]. Additionally, we measure and compare the computational performance of our method against NCS. Furthermore, we validate the effectiveness of our two-stage training process by an ablation study, and we demonstrate the efficiency and applicability of our method by implementing it in a real-time application demo.

5.1 Training Setup

Training Dataset: For our experiments, we use pose sequences from the AMASS dataset [31]. We adopt the sequence list from the VTO dataset [32] for training purposes. The VTO dataset takes sequences from the AMASS dataset, but we replace some unavailable sequences of the list (`104_17`, `104_04`, `104_53`, `104_54`, `144_30`, `26_11`, `104_11`) with others in similar pose categories of the AMASS dataset (`105_36`, `111_23`, `127_04`, `127_20`, `144_32`, `26_10`, `105_11`). This sequence list contains 56 sequences with a total of 19,145 frames, comprising 13 walking, 12 running, 10 jumping, 10 arm movements, 6 torso movements, 4 dancing, and 1 avoidance movement sequence. For testing, we use other sequences from the AMASS dataset, excluding those used for training. The garments for our experiments include a dress, a long-sleeve top, pants, a tank top, and a t-shirt, provided by Grigorev et al. [6] for training their HOOD network. These

garments are aligned with the template female body of SMPL. Accordingly, we used the SMPL female body for our experiments.

Training Details: We trained five networks for compact latent learning and five latent predictors with the five garments in the training set. We implemented our method and trained the networks on a PC, equipped with an Intel Xeon W-2135 CPU, an NVIDIA RTX A4000 GPU, and 64GB of RAM. Training for the compact latent learning stage took approximately 40 hours per garment, with 120,000 iterations each. We used the Adam optimizer for training the networks in the compact latent learning stage, with a learning rate 5×10^{-5}, and applied gradient clipping with a max norm of 1.0. Since the training method from HOOD does not support batch training, we used a batch size of one. For each training pose, We randomly sampled the shape $\boldsymbol{\beta}$ from the uniform distribution $\mathcal{U}(-3, 3)$. We also randomly sampled material parameters for the inputs by sampling a value from the uniform distribution $\mathcal{U}(0, 1)$ and scaling it with its minimum and maximum values. The mass for each vertex of a garment m ranged from 4.34×10^{-2} to 7×10^{-1}, the range of Lamé's second parameter μ from 15909 to 63636, the range of Lamé's first parameter λ from 3535.41 to 93333.73, and the range of the bending coefficient k from 6.37×10^{-8} to 1.31×10^{-3}. To advance our garment deformation over time, we apply the autoregressive training from HOOD, predicting one next step at the beginning and increasing the number of prediction steps every 5000 iterations to 5. We further set the inertia loss weight w_i from 3.0 to 5.0 and the gravity loss weight w_g from 2.0 to 3.0 to make our garment deformation more dynamic, with the other weights w_b, w_s, w_f set to 1.0. The collision loss w_c starts at 5×10^3 at the beginning of training and increases linearly to 8×10^6 from 50,000 iterations to 100,000 iterations. Each latent predictor was trained for 5 million iterations, which took approximately 12 hours. We trained latent predictors for each garment using an Adam optimizer with a learning rate of 1×10^{-4} and a batch size of 512.

Normalization: When training a network in the compact latent learning stage, we adopted the inputs and outputs normalization from HOOD. Unlike the normalization method for the outputs from HOOD, which gathers statistics for acceleration from linearly-skinned garments, we collect statistics for displacement from our prediction and evaluate the final displacement by denormalizing the predicted outputs. When training a latent predictor, we also normalize the inputs $\mathbf{X}_c$ and the outputs $\mathbf{Z}$ by mean and standard deviations for each input and each output except for the material

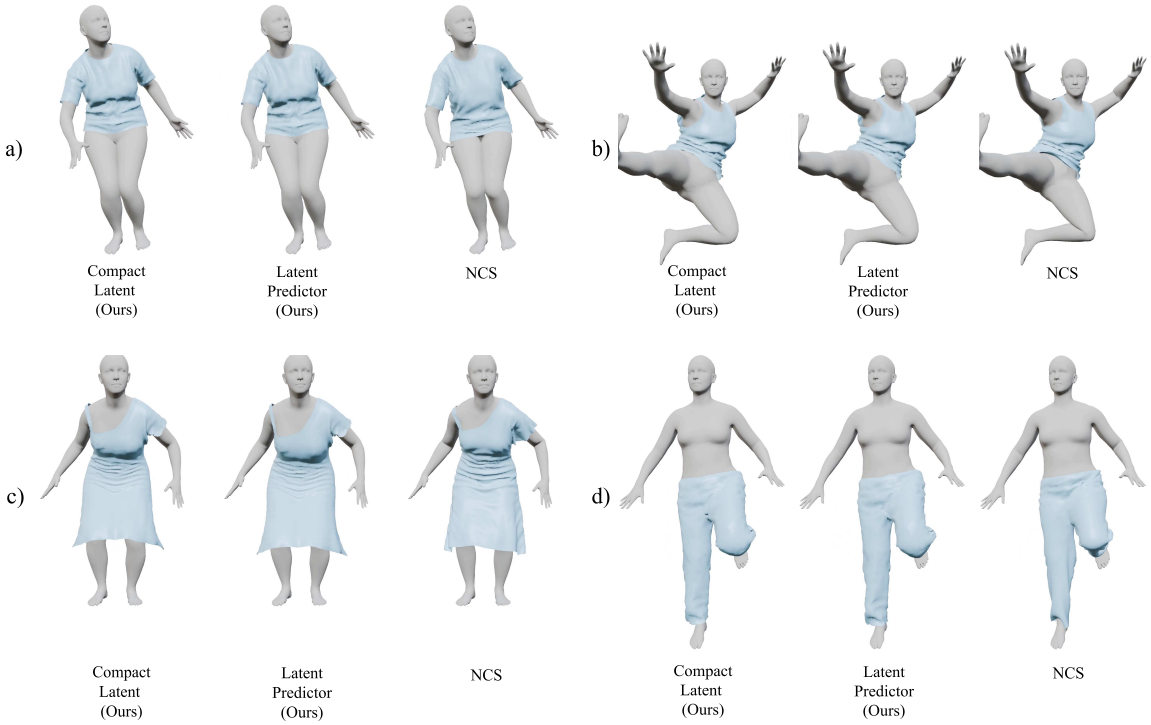


Figure 5: Qualitative comparison with NCS. For each garment, the first column shows the results from networks Φ in the compact latent learning stage, the second column shows results from networks Ψ using latent predictors f_{LP}, and the third column shows results from NCS networks.

parameters. The material parameters in the inputs are normalized to fall between 0.0 and 1.0 based on their value ranges.

5.2 Comparison with State-of-the-art

For this comparison, we trained five NCS networks for each garment in the training set with the template SMPL female body, as NCS supports only a fixed-shaped body with a garment. We used the public code released by Bertiche et al. [21] and a batch size of 512. We chose the cloth model of Baraff et al. [2], since it is more optimal than the StVK model for convergence according to Bertiche et al. [21]. We further applied the heuristic method of Li et al. [22], which gradually increases the inertia loss weight from 0.1 to 1.0. This heuristic allows us to get the best cloth dynamics.

Unsupervised methods lack ground-truth data for quantitative analysis. Generating ground-truth data with a cloth simulator and evaluating the distance to this ground truth would not be a fair comparison. This is because there are various types of cloth simulators, each with different

Figure 6: The generalization capacity to different body shapes from networks Ψ using the latent predictor, the decoder, and the blend shapes. **(a)** and **(d)** are the thin bodies, **(b)** and **(e)** are the template bodies, and **(c)** and **(f)** are the heavy bodies.

ranges of parameters that can influence the distance between our method's results and the ground truth-data. Furthermore, the variety of network architectures, loss function implementations, and parameter ranges complicates a fair quantitative comparison. Moreover, the sole comparison of loss values does not work. For instance, a lower inertia loss value does not guarantee superior quality or more dynamic deformation, as demonstrated by Bertiche et al. [21]. Consequently, our comparison is solely qualitative.

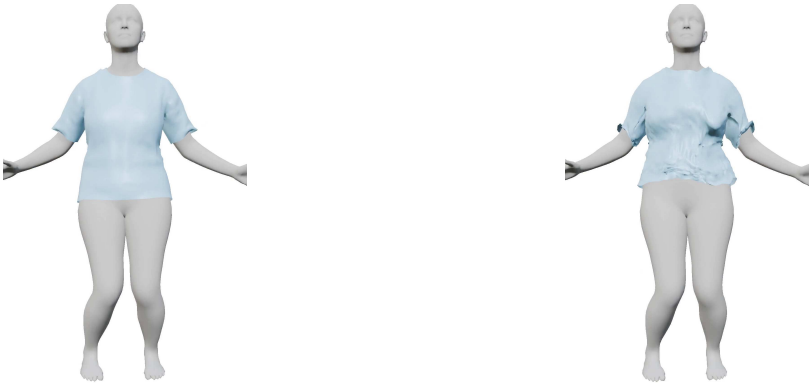
We do not compare our method directly with HOOD, even if our method incorporates high-dimensional features from the encoder and processor of HOOD. Based on our observations, HOOD has a superior capacity for capturing cloth dynamics compared to other pose-based cloth synthesis methods. We attribute this to HOOD's ability to utilize a large number of features, which allows it to construct its broader cloth deformation space than the cloth deformation space from any other pose-based garment synthesis methods. While these high-dimensional features enable a better deformation space, they also restrict HOOD's usage to offline garment simulation due to its computational intensity from high-dimensional features. In contrast, pose-based neural cloth synthesis methods can be employed in real-time applications using lower-dimensional features which are less expressive than the features from HOOD. This distinction suggests that comparing methods designed for real-time applications with those intended for offline use, based on feature dimensions alone, is not a fair comparison, since the deformation space from the number of features inherently would be a trade-off between expressive features and fast inference time. Therefore, we limit our qualitative

comparison to NCS, the state-of-the-art method for pose-based neural cloth synthesis.

We present our qualitative results with NCS in Fig. 5. The first column shows the inference from the compact latent learning stage network, the second column shows the inference using the latent predictor with the decoder and blend shapes, and the third column shows the inference from NCS. As seen in Fig. 5, our method produces wrinkles and dynamics of similar quality to NCS. Unlike NCS, our method supports generalization to different body shapes with the same trained network. This generalization is shown in Fig. 6, which includes a thin body, the template body, and a heavy body. Our latent predictor can handle different body shape parameters β and predicts a compact latent vector accordingly.

For computational performance comparisons, a wide range of neural models for cloth simulation exist. Graph neural network models such as HOOD [6] and SwinGar [22] do not claim to achieve engine-ready real-time performance, as their architecture limits their performance to orders of magnitude slower. Only pose-driven models such as PBNS [20], SNUG [15], and NCS [21] achieve the desired performance criteria. Out of these models, PBNS is purely a static model, not being capable of achieving garment dynamics, and SNUG has dynamics limited to three frames, as thoroughly discussed in Bertiche et al. [21]. As such, we limit our comparison to the closest competitor, NCS [21].

Performance Evaluation: We evaluated the per-frame inference times on GPU for the "full model" (including both model inference and blend shape resolution). We also separately evaluated the runtimes for solely the neural network inference without the blend shape resolution step. Our method takes 523 microseconds per frame for the full model and 238 microseconds for the neural network inference. In contrast, NCS takes 1266 microseconds per frame for the full model and 1030 microseconds for the neural network inference. This result shows that our method outperforms NCS on GPU in the computational performance. We believe this behavior is explained by architecture differences between our model and NCS: our model utilizes more neurons but a simpler structure than NCS, as we do not utilize a GRU layer. As such, when computing on GPU, our model benefits more from GPU parallelism than NCS, while when computing on CPU, the lack of parallelism hurts our performance. The evaluation was done using models exported to ONNX Runtime on a t-shirt garment with 4424 vertices. The hardware utilized was an Intel Xeon W-2255 CPU @ 3.7GHz



(a) Full method **(b)** Training only the inference block (Fig. 4 (c))

Figure 7: (a) Our method shows accurate garment deformation and realistic dynamics (results from network Ψ using the latent predictor, the decoder, and the blend shapes). **(b)** The ablation study, trained with only the inference block (Fig. 4 (c)), results in poor and invalid deformations, highlighting the necessity of the compact latent learning stage.

CPU, and an NVIDIA RTX 2070 Super GPU.

5.3 Ablation Study

To validate the effectiveness of our two-stage training process in learning garment deformation and dynamics, we performed an ablation study. We directly trained only the components used during inference (Fig. 4 (c)), specifically the latent predictor, the decoder, and the learnable blend shapes. We used the same training setup with the t-shirt garment as in the compact latent learning stage, and additionally supported batch training with a batch size of 32. The results, presented in Fig. 7 and the supplementary video, demonstrate that training with only the MLPs and blend shapes does not achieve proper garment deformation and dynamics. The results from the ablation study exhibit less realistic dynamics and invalid deformations in parts of the garment. This outcome validates that our compact latent learning stage is essential for effectively learning garment deformation and dynamics.

5.4 Application on Real-Time Demo

We implemented our inference model with the trained latent predictor, decoder, and blend shapes using C++ and OpenGL. In the supplementary video and Fig. 8, we show a SMPL body

controlled using motion matching [7, 8]. This shows the ease with which our method can be applied in real time, thanks to our inference model consisting of only two MLPs with blend shapes.

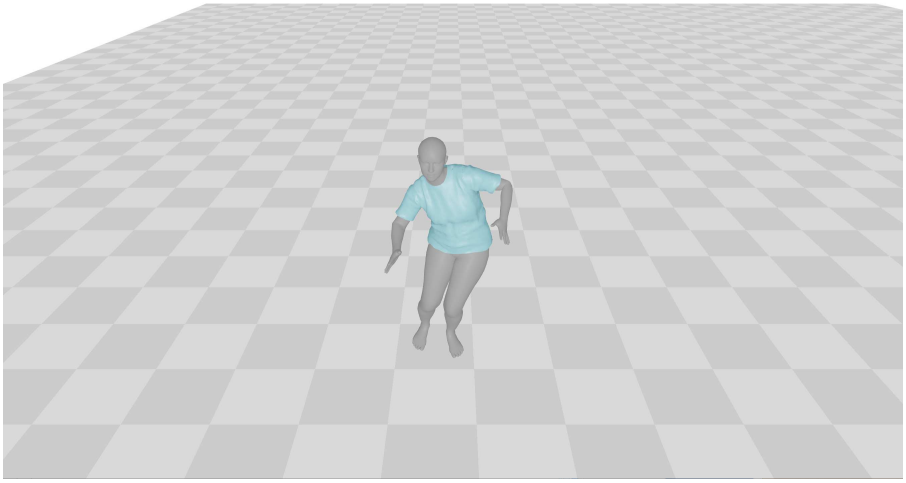


Figure 8: The screenshot of our real-time demo. The template SMPL body is controlled using motion matching, which outputs joint translations and rotations. The pose from the motion matching with the other inputs is used to predict garment deformation by our networks.

Chapter 6

Conclusion

Realistic cloth deformation in game applications is complex due to varying character body shapes, poses, and movements, and many garments with different geometry, topology and materials. On top of these requirements, real-time cloth deformation is essential for an enjoyable game experience. Procedural physics-based techniques are being replaced by neural network approaches. The latter seems more robust and yields high-quality results. However, the representation models required to capture realistic cloth deformation behavior are of very high dimension and correspondingly difficult to incorporate in game engines with limited resources and also far too slow for use in games. The two-stage training strategy presented in this work balances computational resource constraints and realistic cloth deformation effectively for application in games. The first stage learns a compact representation from the high-dimensional feature representation of a complex trained network, and the second stage learns a latent predictor with inputs in a format best suited for input to game engines. At inference time, the latent predictor predicts the compacted latent which is processed by the decoder and blend shape networks from the first stage, enabling framewise inference in real-time. This two-stage strategy is general and could be used in other neural network methods, wherein high-quality results mandate very high dimensional feature representations, and correspondingly high computational resources, limiting their application in real-time environments. Our experiments demonstrate the effectiveness and applicability of our method compared to NCS.

Our method has some limitations. Similar to other learning-based methods for garment synthesis, there are instances in test sequences where interpenetration between the garment and the body is

Figure 9: After our method fails to resolve collisions between the body and the garment, this failure can cause invalid garment deformations. Enhancing collision handling by an edge-based collision loss can be a potential solution for this problem.

not fully resolved. Since we utilize the previous compact latent in the latent predictor, these failures can lead to invalid garment deformation. This issue is especially problematic on loose garments, as shown in Fig. 9. Future work could focus on enhancing collision handling by implementing an edge-based collision loss, which may provide more robust results compared to the vertex-based collision loss currently used [33]. Another direction to address our limitations is to explore better network components for constructing the garment deformation space. Currently, our deformation space struggles to accommodate extreme poses, making it crucial to identify components that can effectively handle such scenarios. While increasing the dimensionality of some components might seem like a feasible and straightforward solution, it could compromise fast inference. Therefore, balancing the need for a more robust deformation space with the requirement for fast computation remains a key challenge in this direction.

Bibliography

[1] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. SMPL: A skinned multi-person linear model. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, 34(6):248:1–248:16, October 2015.

[2] David Baraff and Andrew Witkin. Large steps in cloth simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, page 43–54, New York, NY, USA, 1998. Association for Computing Machinery. ISBN 0897919998.

[3] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman, and Mark Carlson. Physically based deformable models in computer graphics. In *Computer graphics forum*, volume 25, pages 809–836. Wiley Online Library, 2006.

[4] Nadia Magnenat-Thalmann, Richard Laperrire, and Daniel Thalmann. Joint-dependent local deformations for hand animation and object grasping. In *In Proceedings on Graphics interface'88*. Citeseer, 1988.

[5] John P Lewis, Matt Cordner, and Nickson Fong. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 165–172, 2000.

[6] Artur Grigorev, Bernhard Thomaszewski, Michael J. Black, and Otmar Hilliges. HOOD: Hierarchical graphs for generalized modelling of clothing dynamics. In *Computer Vision and Pattern Recognition (CVPR)*, 2023.

[7] Simon Clavet. Motion matching and the road to next-gen animation. In *GDC 2016*, 2016.

[8] Daniel Holden, Oussama Kanoun, Maksym Perepichka, and Tiberiu Popa. Learned motion matching. *ACM Trans. Graph.*, 39(4), aug 2020. ISSN 0730-0301.

[9] Yi Zhou, Connelly Barnes, Lu Jingwan, Yang Jimei, and Li Hao. On the continuity of rotation representations in neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[10] Rony Goldenthal, David Harmon, Raanan Fattal, Michel Bercovier, and Eitan Grinspun. Efficient simulation of inextensible cloth. *ACM Trans. Graph.*, 26(3):49–es, jul 2007. ISSN 0730-0301.

[11] Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus Gross. Example-based elastic materials. In *ACM SIGGRAPH 2011 Papers*, SIGGRAPH '11, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450309431.

[12] Tiantian Liu, Adam W. Bargteil, James F. O'Brien, and Ladislav Kavan. Fast simulation of mass-spring systems. *ACM Trans. Graph.*, 32(6), nov 2013. ISSN 0730-0301.

[13] T. F. Gast and C. Schroeder. Optimization integrator for large time steps. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '14, page 31–40, Goslar, DEU, 2015. Eurographics Association.

[14] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, 2e edition, 2006.

[15] Igor Santesteban, Miguel A Otaduy, and Dan Casas. SNUG: Self-Supervised Neural Dynamic Garments. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[16] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to simulate complex physics with graph networks. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

[17] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021.

[18] 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. In *International conference on machine learning*, pages 4470–4479. PMLR, 2018.

[19] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 4509–4517, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

[20] Hugo Bertiche, Meysam Madadi, and Sergio Escalera. Pbns: Physically based neural simulation for unsupervised garment pose space deformation. *ACM Trans. Graph.*, 40(6), dec 2021. ISSN 0730-0301.

[21] Hugo Bertiche, Meysam Madadi, and Sergio Escalera. Neural cloth simulation. *ACM Trans. Graph.*, 41(6), nov 2022. ISSN 0730-0301.

[22] Tianxing Li, Rui Shi, Qing Zhu, and Takashi Kanai. Swingar: Spectrum-inspired neural dynamic deformation for free-swinging garments. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–16, 2023.

[23] Chaitanya Patel, Zhouyingcheng Liao, and Gerard Pons-Moll. Tailornet: Predicting clothing in 3d as a function of human pose, shape and garment style. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7365–7375, 2020.

[24] Meng Zhang, Duygu Ceylan, and Niloy J Mitra. Motion guided deep dynamic 3d garments. *ACM Transactions on Graphics (TOG)*, 41(6):1–12, 2022.

[25] Honghu Chen, Yuxin Yao, and Juyong Zhang. Neural-abc: Neural parametric models for articulated body with clothes. *IEEE Transactions on Visualization and Computer Graphics*, 2024.

[26] Christopher Lewin. Swish: Neural network cloth simulation on madden nfl 21. In *ACM SIGGRAPH 2021 Talks*, pages 1–2. 2021.

[27] Enric Corona, Albert Pumarola, Guillem Alenya, Gerard Pons-Moll, and Francesc Moreno-Noguer. Smplicit: Topology-aware generative model for clothed people. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11875–11885, 2021.

[28] Zorah Lahner, Daniel Cremers, and Tony Tung. Deepwrinkles: Accurate and realistic clothing modeling. In *Proceedings of the European conference on computer vision (ECCV)*, pages 667–684, 2018.

[29] Yi Zhou, Chenglei Wu, Zimo Li, Chen Cao, Yuting Ye, Jason Saragih, Hao Li, and Yaser Sheikh. Fully convolutional mesh autoencoder using efficient spatially varying kernels. *Advances in neural information processing systems*, 33:9251–9262, 2020.

[30] Hugo Bertiche, Meysam Madadi, Emilio Tylson, and Sergio Escalera. Deepsd: Automatic deep skinning and pose space deformation for 3d garment animation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5471–5480, 2021.

[31] Naureen Mahmood, Nima Ghorbani, Nikolaus F. Troje, Gerard Pons-Moll, and Michael J. Black. AMASS: Archive of motion capture as surface shapes. In *International Conference on Computer Vision*, pages 5442–5451, October 2019.

[32] Igor Santesteban, Nils Thuerey, Miguel A Otaduy, and Dan Casas. Self-Supervised Collision Handling via Generative 3D Garment Models for Virtual Try-On. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[33] Theodore Kim and David Eberle. Dynamic deformables: implementation and production practicalities (now with code!). In *ACM SIGGRAPH 2022 Courses*, SIGGRAPH '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393621.