HOOD: Hierarchical Graphs for Generalized Modelling of Clothing Dynamics

Artur Grigorev[1,2] Bernhard Thomaszewski[1] Michael J. Black[2]

Otmar Hilliges[1]

[1] ETH Zurich, Department of Computer Science
[2] Max Planck Institute for Intelligent Systems, Tubingen

1. Implementation details

1.1. Garments preprocessing

Hierarchical graph construction We tackle the problem of limited propagation radius in the message-passing networks by constructing several levels of coarsened garment graphs.

To generate coarse edges from the initial garment graph, we first find a centre node of a graph v_c, that is a node, whose eccentricity is equal to the graphs' radius. Usually, garment graphs have more than one node with this property, so we randomly choose one of them. Then, for coarsened graph, we keep only those nodes, whose distance (i.e., the number of edges in the shortest path) to the centre is an even number and replace other nodes with edges between pairs of the kept nodes. See Algorithm 1 for details.

input : fine graph $G_f(V_f, E_f)$
output: coarse graph $G_c(V_c, E_c)$

$v_{center} \leftarrow$ center of G_f ;
for v_i *in* V_f **do**
  $d_i \leftarrow distance(v_{center}, v_i)$;
$V_c \leftarrow \{v_i \in V_f |\ d_i \mod 2 = 0\}$;
$E_c \leftarrow \{\}$;
$V^{interm} \leftarrow \{v_i \in V_f |\ d_i \mod 2 = 1\}$;
for v_i *in* V^{interm} **do**
  $V^{from} \leftarrow$
   $\{v_j \in V_f |\ e_{ij} \in E_f \ \&\ d_j = d_i - 1\}$;
  $V^{to} \leftarrow \{v_j \in V_f |\ e_{ij} \in E_f \ \&\ d_j = d_i + 1\}$;
  for v_j *in* V^{from} **do**
    for v_k *in* V^{to} **do**
      $push(e_{jk}, E_c)$

Algorithm 1: Building a coarse graph G_c from a given input graph G_f.

Resting pose geometry As we use resting pose geometries to compute the physical objective function (see Sec. 1.3), for some garments, we need to make a preprocessing step to generate these geometries.

Specifically, we find that canonical 3D geometries of the garments used in [4] are overly smooth and tight. At the same time [4] uses 2D triangle geometries in the UV space as canonical for computing the $\mathcal{L}_{stretching}$ term. However. we compute $\mathcal{L}_{stretching}$ using 3D resting pose triangles projected into 2D.

To better match the garment sizes used in [4] we generate the resting pose geometries with a preprocessing step. To do that, we run an LBFGS optimization using the exact same physical objectives used in [4] with the same 2D triangle geometries to get a relaxed version of the garments for the canonical SMPL pose.

For other garments, for example, those generated with a FoldSketch method [3] or the one created manually in Blender, we don't run this preprocessing step and use their original 3D geometries as resting geometries.

Skinning weights Our model autoregressively predicts nodal accelerations for the garment mesh given it's geometries from two previous time steps (this gives us nodal positions and velocities for the previous step). As we want to initialize the garment geometry from any point in the training sequences, during training we need a way to approximate the garment geometries for any given body pose.

To do that we follow [4] and, for each garment node, borrow skinning weights and blend-shapes from the closest SMPL vertex in canonical pose.

Although it works fine for tight-fitting garments, we find that when applied to loose garments (e.g. dress), this results in overly stretched triangles. Hence, it is difficult for the model to start from a garment with severely sub-optimal potential energy. To overcome this issue, we employ diffused body model formulation from [5] to compute diffused

skinning weights $\tilde{\mathcal{W}}$ and pose and shape blend-shapes $\tilde{\mathcal{B}}_{p,s}$.

$$\tilde{\mathcal{W}}(x) = \frac{1}{N} \sum_{q_n \sim N(x,d)} \mathcal{W}(\phi(q)) \quad (1)$$

$$\tilde{\mathcal{B}}_{p,s}(x) = \frac{1}{N} \sum_{q_n \sim N(x,d)} \mathcal{B}_{p,s}(\phi(q)), \quad (2)$$

where x is a position of a garment node in resting pose, d is the distance from this node to the closest SMPL vertex and $\phi(\cdot)$ is a function that for the given 3D point returns the closest SMPL vertex. However, instead of training an MLP, as in [5], for each garment node in the canonical pose, we directly sample M 3D points from normal distribution $N(x_i, d)$. We find $M = 10000$ enough to get adequate initialization.

Pinned vertices For some of the garments we need to specify a set of "pinned" vertices, i.e. vertices whose positions are rigidly connected to the body mesh (e.d. pants belt). We generate positions of these pinned vertices using our linear blend-skinning formulation and give them as input to the network while also giving these vertices a separate type label. Since there are only a few such vertices in the garment, using linear blend skinning (or any other rigid transformation w.r.t. the body mesh) does not affect the realism of the predicted dynamics.

1.2. Forward pass

Input feature vectors For each time step, we first endow each node and edge in the input graph with a designated feature vector that describes the state of the node or edge on the previous step along with the local material parameters. Here we list the contents of the input vectors for the nodes of the graph, garment edges and body edges.

The input vectors for the **nodes** of the graph consist of:

- velocity $\vec{v}$ of the not in the previous time step
- normal vector $\vec{n}$ of the node
- nodal $mass$
- local material parameters: μ_{Lame}, λ_{Lame} and $k_{bending}$
- the weight of the inertial loss α (see Section 3.4 of the paper)
- one-hot encoding of the node type (garment node, pinned garment node or body node)
- one-hot encoding of the deepest coarse level the node is present in (separate code for body nodes)

for body nodes we set $mass$, μ_{Lame}, λ_{Lame} and $k_{bending}$ to -1

For each **garment edge** (including fine and coarse ones) connecting nodes i and j, each input vector is the concatenation of:

- a vector describing the relative position of the connected nodes in previous time step $(x_i^{t-1} - x_j^{t-1})$ and its' norm $\|(x_i^{t-1} - x_j^{t-1})\|$
- a vector describing the relative position of the connected nodes in resting pose $(x_i^{rest} - x_j^{rest})$ and its' norm $\|(x_i^{rest} - x_j^{rest})\|$
- local material parameters: μ_{Lame}, λ_{Lame} and $k_{bending}$ averaged across the connected nodes
- the weight of the inertial loss α (see Section 3.4 of the paper)

For each **body node** connecting garment node i and body node j, the input vector is the concatenation of:

- a vector describing the relative position of the garment node and the body node in the previous time step $(x_i^{t-1} - x_j^{t-1})$ and its' norm $\|(x_i^{t-1} - x_j^{t-1})\|$
- a vector describing the relative position of the garment node in the previous time step and the body node in the current time step $(x_i^{t-1} - x_j^{t})$ and its' norm $\|(x_i^{rest} - x_j^{rest})\|$
- the weight of the inertial loss α (see Section 3.4 of the paper)

Network architecture Our model consists of 3 parts: the encoder, N message-passing steps and the decoder. The encoder converts the input feature vectors into latent vectors with h dimensions. Message-passing steps update the latent vectors for each node and edge of the graph. The decoder decodes the nodal latent vectors into scalar accelerations. We use the latent dimensionality $h = 128$ in our experiments.

Here, we describe the architectures for each these parts.

The **encoder** comprises $M + 2$ multi-layer perceptrons (MLPs), where M is the total number of levels in the network. Each of the M MLPs encode feature vectors of the garment edges on the specific level. The other two MLPs encode the nodal feature vectors and the feature vectors for the body edges.

Each of the **message-passing steps** consists of $L + 2$ MLPs. Here L is the number of levels processed by this specific step. The other two MLPs, again, process the nodal features and the body edge features.

The **decoder** is a single MLP that decodes the features of each garment node into the predicted accelerations.

Each MLP in the architecture has 2 hidden layers with ReLU activations and layer normalizations.

1.3. Physical supervision

We directly borrow the physical objective terms introduced in SNUG [4] to train our model, with the exceptions of the collision term $\mathcal{L}_{collision}$ and stretching term $\mathcal{L}_{stretching}$, with we slightly modify for our needs and the novel term $\mathcal{L}_{friction}$.

Here we describe these terms in more detail.

Collision term We modify $\mathcal{L}_{collision}$ to compute the penetration of each garment node with respect to the closest *face* of the body mesh rather than to the closest *vertex*.

Apart from that, while we compute penetrations happening in the current time step, we use the body faces that were closest to the garment vertices in the previous time step. So the process for the computation of $\mathcal{L}_{collision}$ in time step t is as follows:

1. for each garment node v_i find the body face f_j, whose center f_j^{center} is closest to v_i in the time step $t-1$,
2. compute the size of the collision of vertex v_i with respect to the face f_j

The collision size is computed similarly to SNUG:

$$\mathcal{L}_{collision}(v_i) = max(\epsilon - d(v_i, f_j)) \tag{3}$$

$$d(v_i, f_j) = (v_i - f_j^{center}) \cdot \vec{n_j} \tag{4}$$

where $\vec{n_j}$ is the normal vector of the face f_j.

In that way, the body-garment correspondences align with those used to build world edges. We find this little modification of the collision term crucial for the convergence of the model. Note, that for the metrics reported in Table 1 we use the regular collision term, with correspondences for the current time step.

Friction We introduce a novel physical objective $\mathcal{L}_{friction}$, which penalizes the sliding friction between the garment and the body. The friction energy is computed separately for every vertex. Following [1, 2], we penalize the movement of a node relative to the body for those vertices that fall into the r of the closest body face in both current and previous time frames.

Here we describe the process of computing the fiction term for a specific vertex v step-by-step.

First, we find indices m and k of two body faces, closest to v in the current and the previous time steps (see Figure 1).

$$m = \arg\min_p \|f_p^t - v^t\| \tag{5}$$

$$k = \arg\min_p \|f_p^{t+1} - v^{t+1}\| \tag{6}$$

Then we compute the node's projected position $\hat{v}^{t+1}$ in time step $t+1$, assuming it moves the same way as f_m

$$\hat{v}^{t+1} = v^t + (f_m^t - f_v^{t+1}), \tag{7}$$

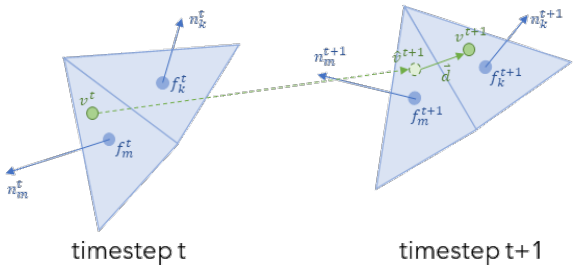

Figure 1. We model friction by penalizing the motion of garment nodes relative to the body.

and find a vector from $\hat{v}^{t+1}$ to v^{t+1} and project it onto a plane with slope θ which is an average between slopes of f_m^t and f_k^{t+1}. Using the norm d of this projected vector, we compute the friction term with the following formula.

$$\mathcal{L}_{friction} = \mu_k \times g \times mass \times \cos\theta \times d \tag{8}$$

$$\theta = (\theta_{f_m}^t + \theta_{f_k}^{t+1})/2 \tag{9}$$

$$d = \|proj(v^{t+1} - \hat{v}^{t+1}, \theta)\|_2, \tag{10}$$

where μ_k is friction ratio, g is gravitational acceleration, $mass$ is a mass of the node, and $proj(\cdot, \theta)$ is an operation of projecting a vector onto a plane with the slope θ_i

While it is a very rough approximation of the friction energy, we find that it allows for more realistic modelling of body-garment interactions. We demonstrate the effect of the friction term in the supplementary video *additional_videos/friction.mp4*.

Vertex mass and canonical geometries Another slight difference to the physical objective formulation from SNUG is how we compute the mass of each garment vertex. As it SNUG, to compute the vertex mass, we need to know the material density and the areas of the adjacent triangle faces. The difference is that, while SNUG uses original 3D garment geometries to compute triangle areas, we use the relaxed ones, acquired using the preprocessing step described in Section 1.1.

Apart from that, SNUG uses original 2D triangles from UV space to compute $\mathcal{L}_{stretching}$, while we compute $\mathcal{L}_{stretching}$ using 3D resting pose triangles projected into 2D.

2. Experiment details

We trained our final model for 150000 training iterations which took around 26 hours on NVIDIA Quadro RTX 6000 GPU.

Below we provide some additional information concerning the experiments from the main paper.

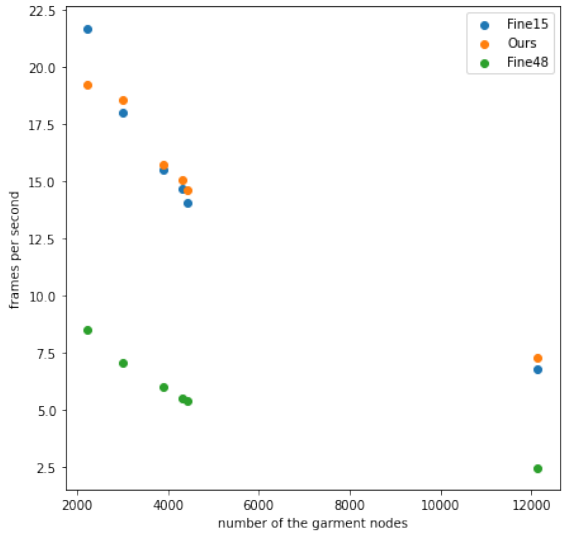

Figure 2. Inference speed in frames per second for our final model and two baseline architectures. Each point corresponds to one of the training garments. The speed was measured using NVIDIA GeForce RTX 3060 GPU

2.1. Comparison to state-of-the-art

Since we do not learn for the physically simulated data as [5] does and use slightly different objective function form SNUG [4] (See Sec. 1.3), it is difficult to quantitatively our method to them. However, we can compute the size and the number of body-garment penetrations by our method and the baselines. We provide these values in Table 1

	garments	$\mathcal{L}_{collision}$	% penetrating vertices
SNUG [4]	t-shirt, long sleeve top,	1.81e-8	1.26e-5
Ours	tank top, pants, shorts	**2.69e-10**	**2.55e-6**
SSCH [5]	t-shirt, dress	5.3e-8	1.23e-5
Ours		**1.22e-9**	**2.38e-7**

Table 1. Our method generates fewer garment-body penetrations compared to state-of-the-art methods in terms of both average penetration size and percentage of garment vertices penetrating the body. The numbers were averaged over the whole validation set.

2.2. Architecture analysis

In addition to Table 1 in the main paper, we provide detailed metrics averaged across the whole validation set for our final model *Ours* and two baseline architectures *Fine15* and $Fine48$ in Table 2.

Different architectures with different sets of fine and coarse message-passing steps may result in models with different inference speeds, propagation radii and levels of fine details. We demonstrate the qualitative differences between different architectures in the supplementary video *additional_videos/architectures.mp4*.

References

[1] George E. Brown, Matthew Overby, Zahra Forootaninia, and Rahul Narain. Accurate dissipative forces in optimization integrators. *ACM Trans. Graph.*, 37(6), dec 2018. 3

[2] Moritz Geilinger, David Hahn, Jonas Zehnder, Moritz Bächer, Bernhard Thomaszewski, and Stelian Coros. Add: Analytically differentiable dynamics for multi-body systems with frictional contact. *ACM Trans. Graph.*, 39(6), nov 2020. 3

[3] Minchen Li, Alla Sheffer, Eitan Grinspun, and Nicholas Vining. Foldsketch: Enriching garments with physically reproducible folds. *ACM Transaction on Graphics*, 37(4), 2018. 1

[4] Igor Santesteban, Miguel A Otaduy, and Dan Casas. Snug: Self-supervised neural dynamic garments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8140–8150, 2022. 1, 3, 4

[5] Igor Santesteban, Nils Thuerey, Miguel A. Otaduy, and Dan Casas. Self-supervised collision handling via generative 3d garment models for virtual try-on. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11763–11773, June 2021. 1, 2, 4

	garments	average speed, fps	$\mathcal{L}_{stretching}$	$\mathcal{L}_{bending}$	$\mathcal{L}_{inertia}$	$\mathcal{L}_{gravity}$	$\mathcal{L}_{collision}$	$\mathcal{L}_{friction}$
Fine15	only dress	6.65	3.73e-3	3.28e-5	1.26e-5	**2.24e-3**	5.74e-9	5.68e-6
Fine48		2.45	**3.95e-4**	**2.71e-5**	9.28e-6	2.3e-3	1.96e-9	5.44e-6
Ours_full		**7.27**	5.22e-4	3.32e-5	**8.48e-6**	2.3e-3	**1.7e-9**	**4.83e-6**
Fine15	all	13.1	2.2e-3	1.51e-5	8.06e-6	**1.41e-3**	1.5e-9	2.92e-6
Fine48		4.99	**2.68e-4**	**1.36e-5**	7.2e-6	1.42e-3	5.54e-10	2.77e-6
Ours_full		**13.6**	3.59e-4	1.53e-5	**6.76e-6**	1.43e-3	**4.48e-10**	**2.55e-6**

Table 2. Comparison of our final model to two ablations in terms of physical objectives and inference speed. We provide the metrics for the largest garment in the training set (*dress*, 12K vertices) separately.