Supplementary Material for DrapeNet: Garment Generation and Self-Supervised Draping

Luca De Luigi[*,2] Ren Li[*,1] Benoît Guillard[1] Mathieu Salzmann[1] Pascal Fua[1]

[1]: CVLab, EPFL, {name.surname}@epfl.ch [2]: University of Bologna, luca.deluigi4@unibo.it

In this appendix we first provide more details about our networks and their architectures in Sec. 1.

In Sec. 2 we expand on the choice and formulations of some loss terms we use. Importantly, in Sec. 2.4 we explain the physics-based refinement procedure used in the main paper, and show that *modelling garments as open surfaces is necessary for it.*

Then in Sec. 3 we report additional quantitative and qualitative results of our pipeline and the runtime of its components. Finally, in Sec. 4 we describe how human ratings were collected.

[*] Equal contributions

1. Network Architectures and Training

1.1. Garment Generative Network

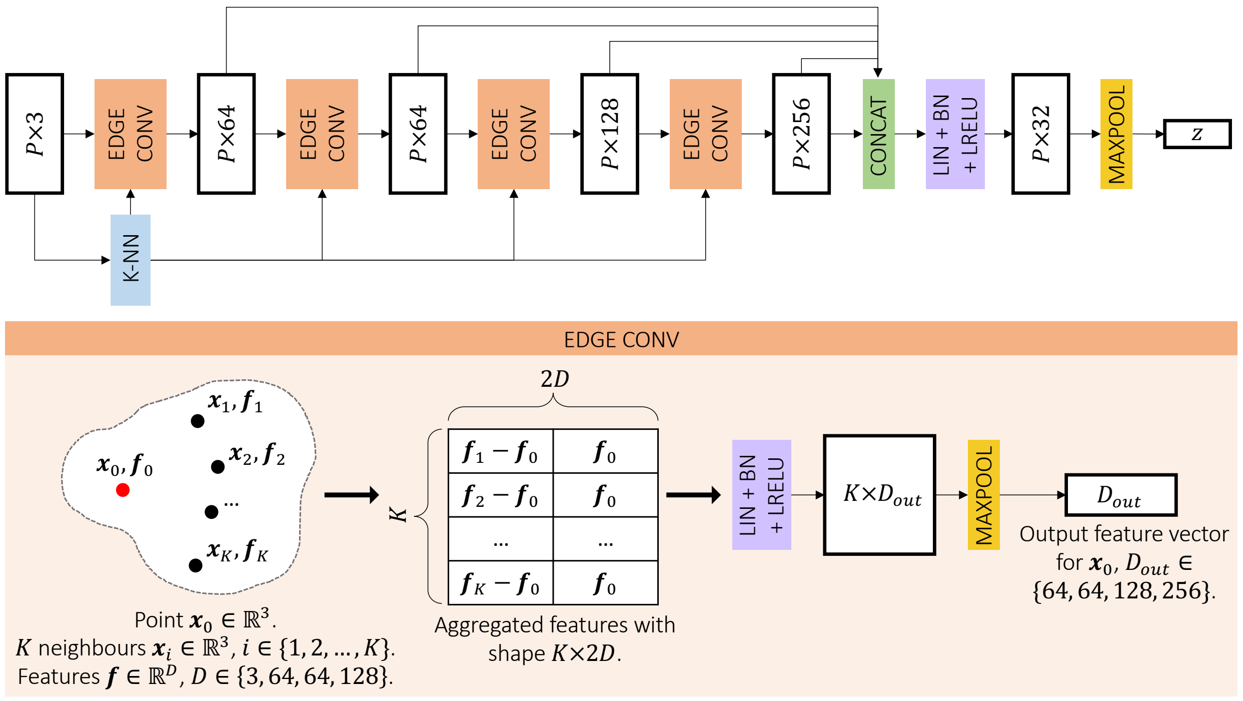
1.1.1 Garment Encoder

To encode a given garment into a compact latent code, we first sample P points from its surface and then we feed them to a DGCNN [23] encoder, detailed in Fig. 11. The input point cloud is processed by four *edge convolution* layers, which project the input 3D points into features with increasing dimensionality – *i.e.*, 64, 64, 128 and finally 256.

Each edge convolution layer works as follows. For each input point, the features from its K neighbours are collected and used to prepare a matrix with K rows. Each row is the concatenation of two vectors: $\mathbf{f}_i - \mathbf{f}_0$ and $\mathbf{f}_0$, $\mathbf{f}_i$ and $\mathbf{f}_0$ being respectively the feature vector of the i-th neighbour and the feature vector of the considered point. Each row of the resulting matrix is then transformed independently to the desired output dimension. The output feature vector for the considered point is finally obtained by applying max pooling along the rows of the produced matrix.

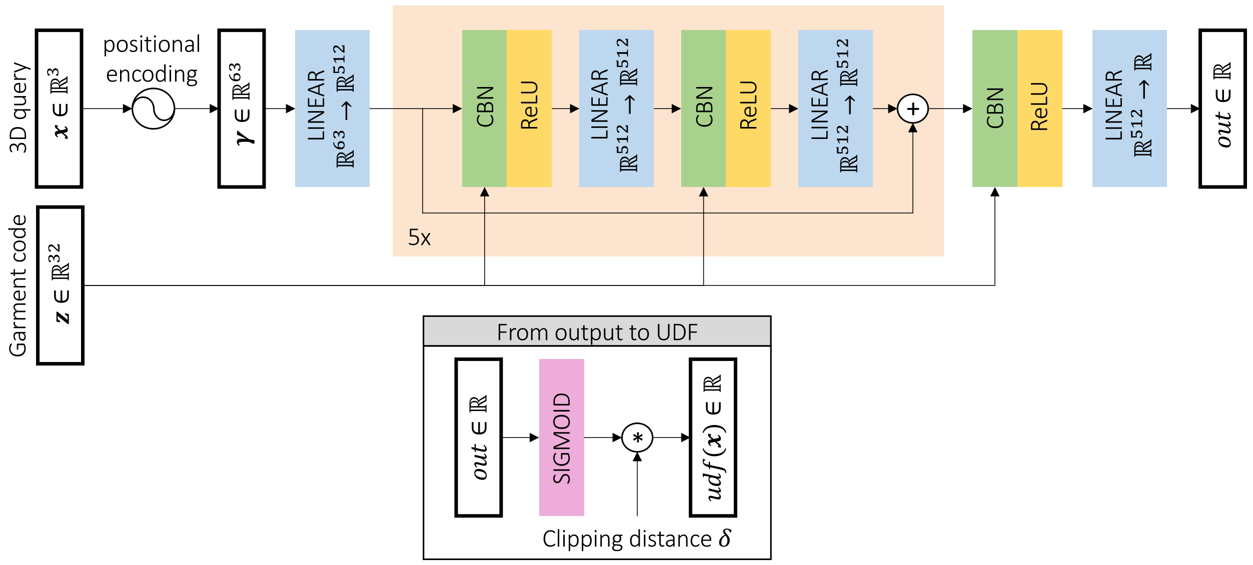
The original DGCNN implementation recomputes the neighborhoods in each edge convolution layer, using the distance between the feature vectors as metric. This can be explained by the original purposes of DGCNN, *i.e.*, point cloud classification and part segmentation. Since we are interested in encoding the geometric details of the input point cloud, we compute neighborhoods only once based on the euclidean distance of the points in the 3D space and reuse this information in every edge convolution layer. We set $K = 16$ in our experiments.

The feature vectors from the four edge convolutions are then concatenated to form a single vector with 512 elements, that is fed to a final linear layer paired with batch normalization and leaky ReLU. Such layer projects the 512 sized vectors into the final desired dimension, which is 32 in our case. The final latent code is obtained by compressing the feature matrix with shape $P \times 32$ along the first dimension with max pooling.



P×3
EDGE CONV
P×64
EDGE CONV
P×64
EDGE CONV
P×128
EDGE CONV
P×256
CONCAT
LIN + BN + LRELU
P×32
MAXPOOL
z
K-NN
EDGE CONV
2D
K
x_1, f_1
x_0, f_0
x_2, f_2
...
x_K, f_K
$f_1 - f_0$
f_0
$f_2 - f_0$
$f_K - f_0$
LIN + BN + LRELU
$K \times D_{out}$
MAXPOOL
D_{out}
Output feature vector for x_0, $D_{out} \in \{64, 64, 128, 256\}$.
Point $x_0 \in \mathbb{R}^3$.
K neighbours $x_i \in \mathbb{R}^3$, $i \in \{1, 2, \ldots, K\}$.
Features $f \in \mathbb{R}^D$, $D \in \{3, 64, 64, 128\}$.
Aggregated features with shape $K \times 2D$.

Figure 11. **DGCNN point cloud encoder.** We adopt DGCNN [23] as the point cloud encoder of our garment generative network. The input cloud is passed through four *edge convolutions*, which gather features of local neighborhoods of points to project them into higher dimensional spaces. The features from all the layers are then concatenated and projected to the final desired dimension. Max pooling is finally used to obtain the latent code $\mathbf{z}$ for the input cloud. *CONCAT* stands for features concatenation, while *LIN + BN + LRELU* represents a linear layer followed by batch normalization and leaky ReLU.



3D query
$x \in \mathbb{R}^3$
positional encoding
$\gamma \in \mathbb{R}^{63}$
LINEAR $\mathbb{R}^{63} \to \mathbb{R}^{512}$
CBN
ReLU
LINEAR $\mathbb{R}^{512} \to \mathbb{R}^{512}$
+
5x
LINEAR $\mathbb{R}^{512} \to \mathbb{R}$
$out \in \mathbb{R}$
Garment code
$z \in \mathbb{R}^{32}$
From output to UDF
SIGMOID
*
$udf(x) \in \mathbb{R}$
Clipping distance δ

Figure 12. **UDF decoder.** Given a 3D query and a garment latent code, the decoder of our garment generative network is trained to predict the UDF of the input query w.r.t. the surface of the garment. The latent code is used to condition the prediction by the means of Conditional Batch Normalization (CBN) [22]. Since we trained the decoder with the binary cross-entropy loss, its outputs need to be converted to UDF values by applying the sigmoid function and then scaling the result with the UDF clipping distance δ.

1.1.2 Garment Decoder

The garment generative network features an implicit decoder that can predict the unsigned distance field of a garment starting from its latent code. More specifically, the decoder is a coordinate-based MLP that takes as inputs the garment latent code and a 3D query. Using the latent code as condition, the decoder predicts the unsigned distance from the query to the garment surface.

Our UDF decoder, shown in Fig. 12, is inspired by [11]. The input 3D query is first mapped to a higher dimensional space ($\mathbb{R}^{63}$) with the positional encoding proposed in [12], which is known to improve the capability of the network to approximate high frequency functions. The encoded query is then mapped with a linear layer to $\mathbb{R}^{512}$ and then goes through 5 residual blocks. The output of each block is computed as $\mathbf{f}_{out} = \mathbf{f}_{in} + \Delta\mathbf{f}$, where $\mathbf{f}_{in}$ is the input vector and $\Delta\mathbf{f}$ is a residual term predicted by two consecutive linear layers starting from $\mathbf{f}_{in}$. The size of the feature vector is 512 across the whole sequence of residual blocks. The output of the last block is mapped to the scalar output $out \in \mathbb{R}$ with a final linear layer.

All the linear layers but the output one are paired with Conditional Batch Normalization (CBN) [22] and ReLU activation function. CBN is used to condition the MLP with the input latent code $\mathbf{z}$. In more details, each CBN module applies standard batch normalization [7] to the input vectors, with the difference that the parameters of the final affine transformation are not learned during the training but are instead predicted at each inference step by dedicated linear layers starting from $\mathbf{z}$.

Finally, recall that our generative network is trained with the binary cross-entropy loss. Thus, the output of the decoder must be converted to the corresponding UDF value by first applying the sigmoid function and then scaling the result with the UDF clipping distance δ, which we set to 0.1 in our experiments. Such procedure is indeed the dual of the one applied on the UDF ground-truth labels during training to normalize them in the range $[0, 1]$.

1.1.3 Surface Sampling

We sample supervision points with a probability inversely proportional to the distance to the surface: 30% of the points are sampled directly on the input surface, 30% are sampled by adding gaussian noise with ϵ variance to surface points, 30% are obtained with gaussian noise with 3ϵ variance, and the remaining ones are gathered by sampling uniformly the bounding box in which the garment is contained. Since in our experiments, the top and bottom garments are normalized respectively into the upper and lower halves of the $[-1, 1]^3$ cube, we set $\epsilon = 0.003$.

1.2. Draping Network

The networks $\mathcal{W}(\mathbf{x}) \in \mathbb{R}^{24}$ and $\Delta x(\mathbf{x}, \beta) \in \mathbb{R}^3$ that predict blending weights and coarse displacements are implemented by a 9-layer multilayer perceptron (MLP) with a skip connection from the input layer to the middle. Each layer has 256 nodes except the middle and the last ones. ReLU is used as the activation function. The body-parameter-embedding module $\mathcal{B}(\beta, \theta) \in \mathbb{R}^{128}$ and the displacement-matrix module $\mathcal{M}(x, \mathbf{z}) \in \mathbb{R}^{128\times 3}$ for Δx_{ref} are implemented by a 5-layer MLP with LeakyReLU activation in-between. Each layer has 512 nodes except the last one. Δx_{IS} uses the same architecture as Δx_{ref}.

1.3. Training Hyperparameters

The generative models (top/bottom ones) are trained on the 600/300 neutral garments for 4000 epochs, using mini-batches of size $B = 4$. Each item of the mini-batch contains an input point cloud with $P = 10,000$ points and $N = 20,000$ random UDF 3D queries. The dimension of the latent codes is set to 32 for both top and bottom garments, and we set $\lambda_g = 0.1$ in

$$\mathcal{L}_{garm} = \mathcal{L}_{dist} + \lambda_g \mathcal{L}_{grad} \,. \tag{14}$$

The draping networks are trained for 250K iterations with mini-batches of size 18, where each item is composed of the vertices of one garment paired with one body shape and pose. We set $\lambda = 0.1$ for $\mathcal{L}_{pin}$ and $\gamma = 0.5$ for $\mathcal{L}_{layer}$.

Both the generative and the draping networks are trained with Adam optimizer [8] and learning rates set to 0.0001 and 0.001 respectively.

2. Loss Terms and Ablation Studies

2.1. $\mathcal{L}_{garm}$ for Garment Reconstruction

We report here an ablation study that we conducted to determine the best formulation for $\mathcal{L}_{garm}$, the loss function presented in Eq. (1) of the main paper, that we use to train our garment generative network.

In particular, we consider three variants for $\mathcal{L}_{dist}$, the term of the supervision signal that guides the network to predict accurate values for the garments UDF. In addition to the binary cross-entropy loss (BCE) presented in Eq. (4) of the main paper, we study the possibility of using more traditional regression losses, such as L1 and L2 losses. Adopting the notation introduced in Sec. 3.1 of the main paper, the L1 loss is defined as $\frac{1}{BN}\sum_{i,j} |min(y_{ij}, \delta) - \widetilde{y}_{ij}|$, while the L2 loss is computed as $\frac{1}{BN}\sum_{i,j}(min(y_{ij}, \delta) - \widetilde{y}_{ij})^2$.

On top of the three variants for $\mathcal{L}_{dist}$, we also consider for each one the possibility of removing the gradients supervision from $\mathcal{L}_{garm}$, *i.e.*, setting $\lambda_g = 0$.

We trained our generative network for 48 hours with the resulting six loss function variants and then compared the

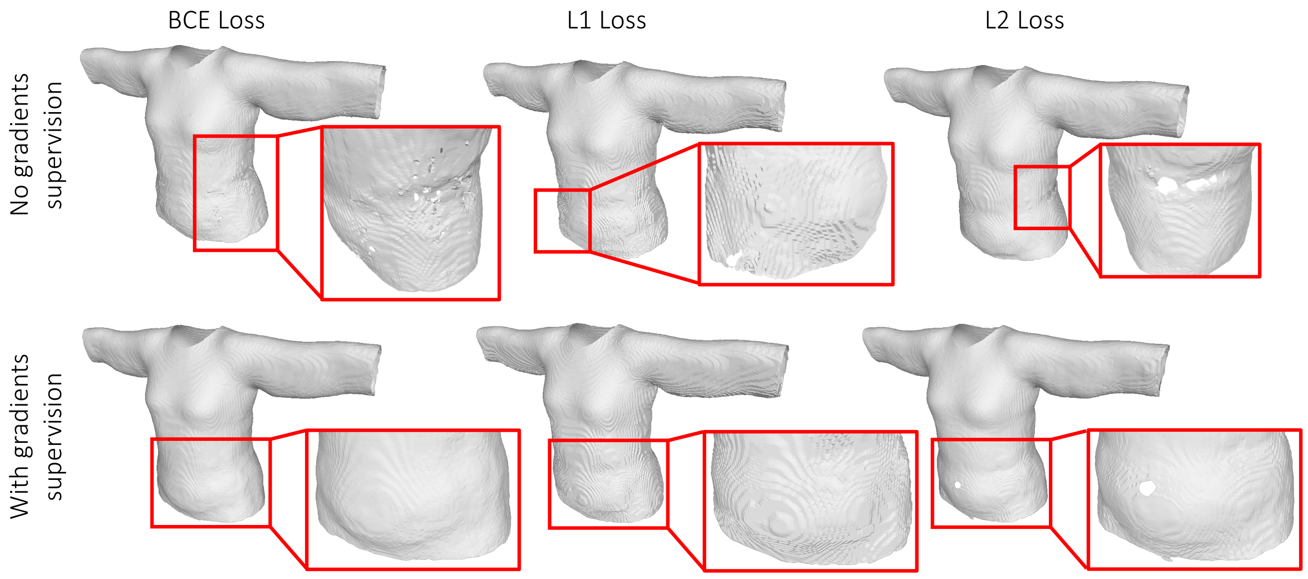


Figure 13. **Comparison between different loss functions for the garment generative network.** We present the same garment reconstructed by our generative network after being trained for 48 hours with six different alternatives of loss functions.

quality of the garments reconstructed with the garment decoder. Fig. 13 presents a significant example of what we observed on the test set. Without gradients supervision (top row of the figure), none of the considered loss functions (BCE, L1 or L2) can guide the network to predict smooth surfaces without artifacts or holes. Adding the gradients supervision (bottom row) induces a strong regularization on the predicted distance fields, helping the network to predict surfaces without holes in most of the cases. However, using the L1 loss leads to rough surfaces, as one can observe in the center column of the bottom row of the figure. The BCE and the L2 losses (first and third columns of the bottom row), instead, produce smooth surfaces that are pleasant to see. We finally opted for the BCE loss over the L2 loss, since the network trained with the latter occasionally predicts surfaces with small holes, as in the example shown in the figure.

2.2. $\mathcal{L}_{pin}$ for Bottom Garments

To determine V, the set of bottom garment vertices that need to be constrained by $\mathcal{L}_{pin}$, we first find the closest body vertex v_B for each bottom garment vertex v. If v_B locates in the body trunk (cyan region as shown in Fig. 14), v is added to V.

In Fig. 15, we show the draping results of bottom garments by using different values for λ in $\mathcal{L}_{pin}$. When λ equals 0 or 1, the deformations along the X and Z axes are not natural because no constraints or too strong constraints are applied, while it is not the case when $\lambda = 0.1$, which is our setting.

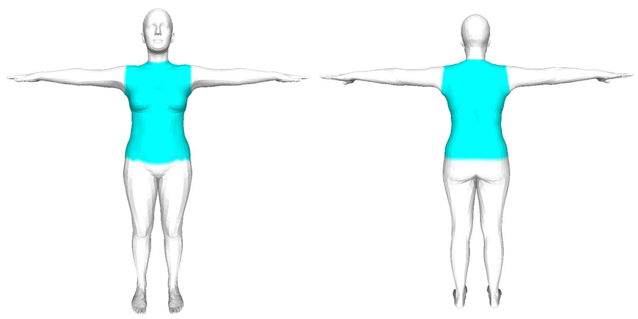


Figure 14. Body region (marked in cyan) used to compute $\mathcal{L}_{pin}$.

2.3. $\mathcal{L}_{layer}$ for Top-bottom Intersection

To determine C, the set of body vertices covered by both the top and bottom garments, we first subdivide the SMPL body mesh for a higher resolution, and then we compute C_{top} the set of closest body vertices for the given top garment, and C_{bottom} the set of closest body vertices for the bottom. C is derived as the intersection of C_{top} and C_{bottom}.

In Fig. 16 we compare the results of models trained without and with $\mathcal{L}_{layer}$. We can observe that without $\mathcal{L}_{layer}$, the top tank can intersect with the bottom trousers, while it is not the case when using $\mathcal{L}_{layer}$. This indicates the efficacy of $\mathcal{L}_{layer}$ to avoid intersections between garments.

2.4. Physics-based Refinement

After recovering the draped garment $\mathbf{G}_D$ from images by the optimization of Eq. (12) of the main paper, we can

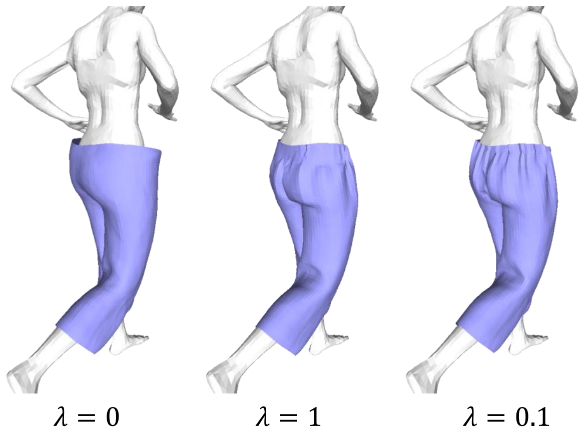


Figure 15. **Comparison between different values for λ of $\mathcal{L}_{pin}$.** To restrict the deformation mainly along the vertical direction (Y axis) and produce natural deformations along other directions, λ has to be a positive value smaller than 1. We use $\lambda = 0.1$ for our training.

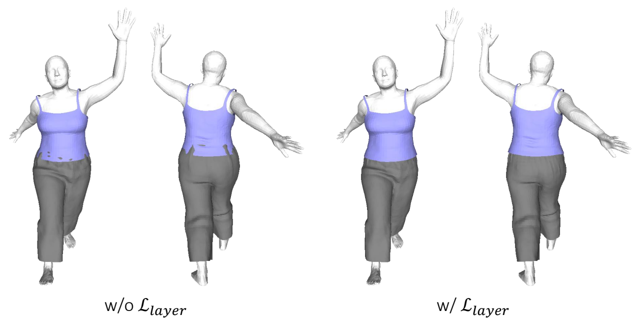


Figure 16. **Comparison: draping without and with $\mathcal{L}_{layer}$.** Without it, the top and bottom garments intersect with each other.

apply the physics-based objectives of Eq. (7) (main paper) to increase its level of realism

$$L(\Delta_{\mathbf{G}}) = \mathcal{L}_{strain}(\mathbf{G}_D + \Delta_{\mathbf{G}}) + \mathcal{L}_{bend}(\mathbf{G}_D + \Delta_{\mathbf{G}}) + \mathcal{L}_{gravity}(\mathbf{G}_D + \Delta_{\mathbf{G}}) + \mathcal{L}_{col}(\mathbf{G}_D + \Delta_{\mathbf{G}}) \ , \tag{15}$$

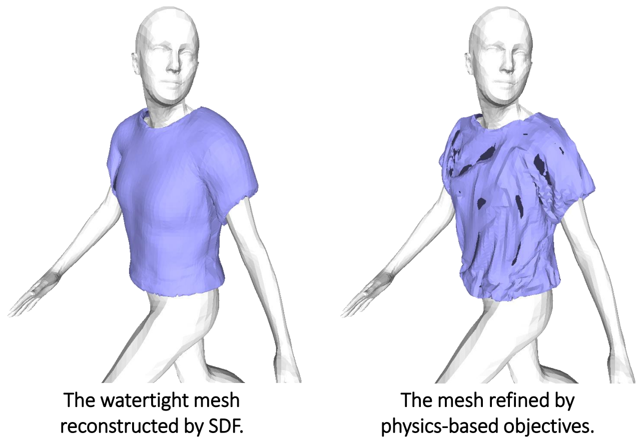
where $\Delta_{\mathbf{G}}$ is the per-vertex-displacement initialized to zero. For the recovery from 3D scans, we apply the following optimization which minimizes both the above physics-based objectives and the Chamfer Distance $d(\cdot)$ to the input scan $\mathbf{S}_{\mathbf{G}}$

$$L(\Delta_{\mathbf{G}}) = \mathcal{L}_{strain}(\mathbf{G}_D + \Delta_{\mathbf{G}}) + \mathcal{L}_{bend}(\mathbf{G}_D + \Delta_{\mathbf{G}}) + \mathcal{L}_{gravity}(\mathbf{G}_D + \Delta_{\mathbf{G}}) + \mathcal{L}_{col}(\mathbf{G}_D + \Delta_{\mathbf{G}}) + d(\mathbf{G}_D + \Delta_{\mathbf{G}}, \ \mathbf{S}_{\mathbf{G}}) \ . \tag{16}$$

This refinement procedure is only applicable to open surface meshes, and our UDF model is thus key to enabling it. Applying Eq. (15) or Eq. (16) to an inflated garment (as recovered by SMPLicit [3], ClothWild [13] and DIG [9]) indeed yields poor results with many self-intersections as illustrated in Fig. 17. This is because inflated garments modelled as SDFs have a non-zero thickness, with distinct inner and outer surfaces whose interactions are not taken into account in this fabric model. The physics model we apply on garment meshes indeed considers collisions of the garment with the body, but not with itself, which is what happens with the inner and outer surfaces in Fig. 17. Adding a physics term to prevent self intersections would not be trivial, and is related to the complex task of untangling layered garments [2, 19]

Note that this is also the case for most garment draping softwares [6, 14–16, 20] to expect single layer garments. Modeling garment with UDFs is thus a key feature of our pipeline for its integration in downstream tasks.

Both the optimizations of Eqs. (12) and (13) of the main paper and Eqs. (15) and (16) are done with Adam [4] but with different learning rates set to 0.01 and 0.001 respectively.



The watertight mesh reconstructed by SDF.
The mesh refined by physics-based objectives.

Figure 17. **Applying post-refinement procedure to watertight mesh**. Left: the watertight mesh reconstructed by DIG [9]. Right: the same mesh after being refined with physics-based objectives (Eq. (15)). Physics-based refinement is not compatible with inflated garment meshes, and leads to many self-intersections.

3. Additional Results

3.1. Garment Encoder/Decoder

3.1.1 Additional Qualitative Results

Fig. 18 and Fig. 19 show the encoding-decoding capabilities of our garment generative network for top and bottom *test* garments, respectively. The ground-truth garments are passed through the garment encoder, which produces a compact latent code for each clothing item. Then, our garment decoder reconstructs the input garments surface from the latent codes. It is possible to notice how the output gar-

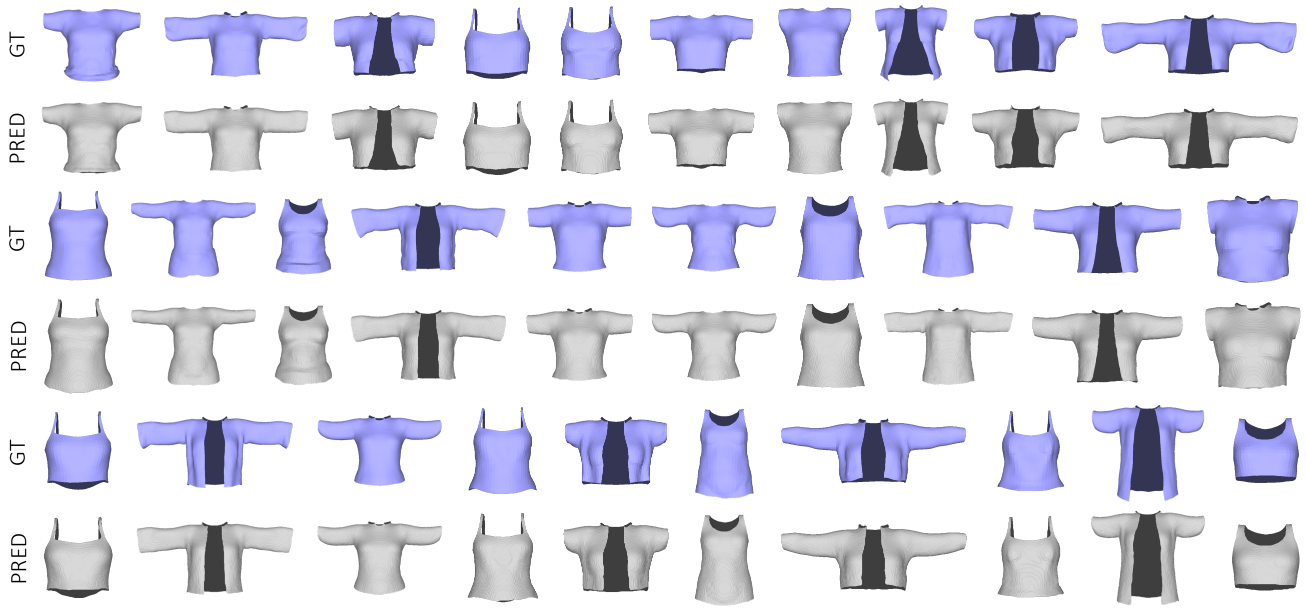


Figure 18. **Generative network: reconstruction of unseen garments in neutral pose/shape (top garments).** Latent codes for unseen garments can be obtained with our garment encoder. These codes are then used by the garment decoder to reconstruct open surface meshes. Input garments are colored in purple, while the reconstructed meshes are colored in gray.

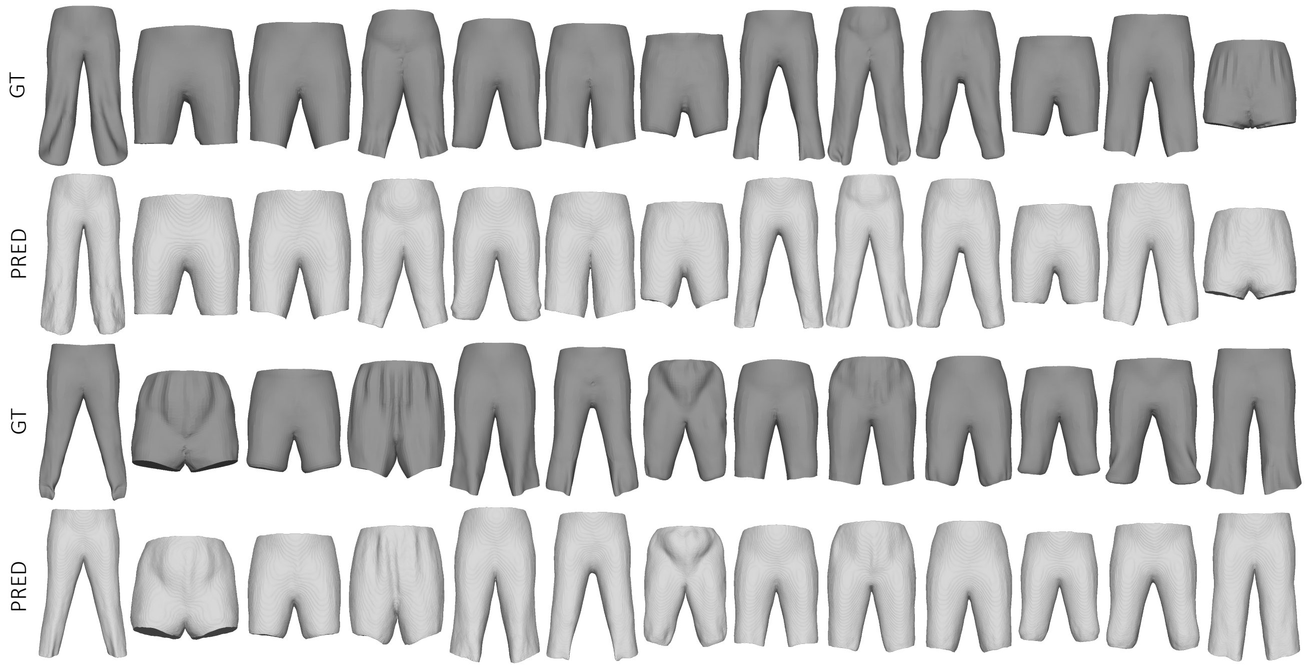


Figure 19. **Generative network: reconstruction of unseen garments in neutral pose/shape (bottom garments).** Latent codes for unseen garments can be obtained with our garment encoder. These codes are then used by the garment decoder to reconstruct open surface meshes. Input garments are colored in dark gray, while the reconstructed meshes are colored in light gray.

ments closely match the input ones, both in terms of geometry and topology.

3.1.2 Latent Space Optimization (LSO).

After training the garment generative network, we obtain a latent space that allows us to sample a garment latent code

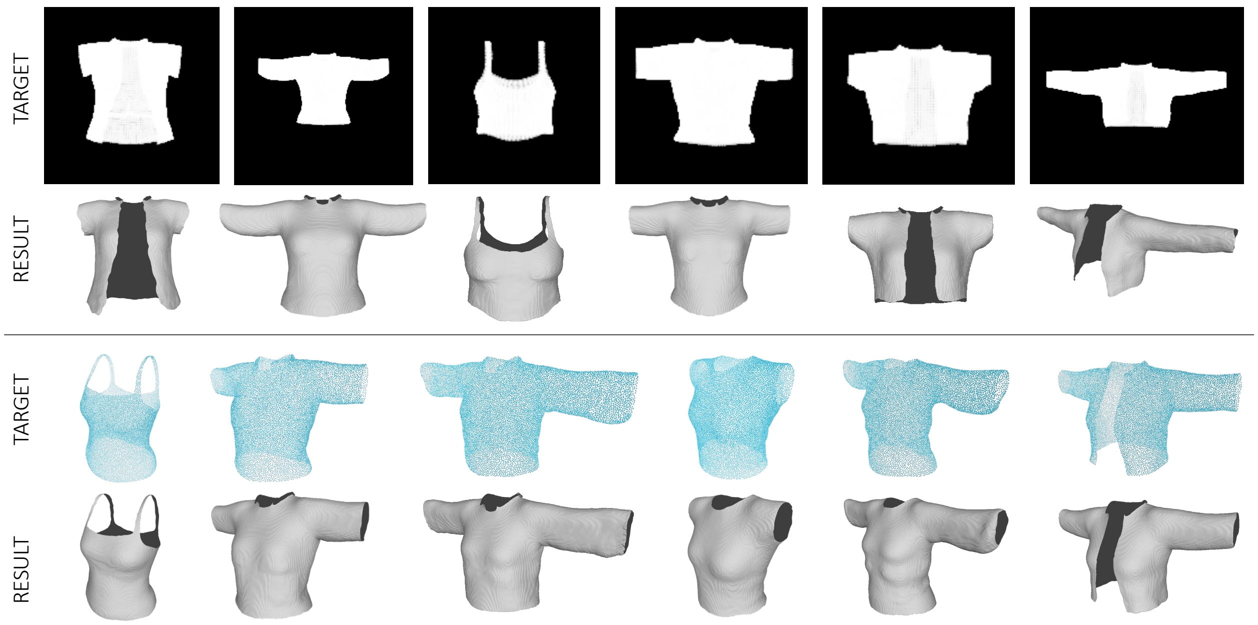


Figure 20. **Generative network: latent space optimization (top garments).** After training, we can explore the latent space learned by the garment generative network with gradient descent, to recover target garments from 2D silhouettes (top) or 3D point clouds (bottom).

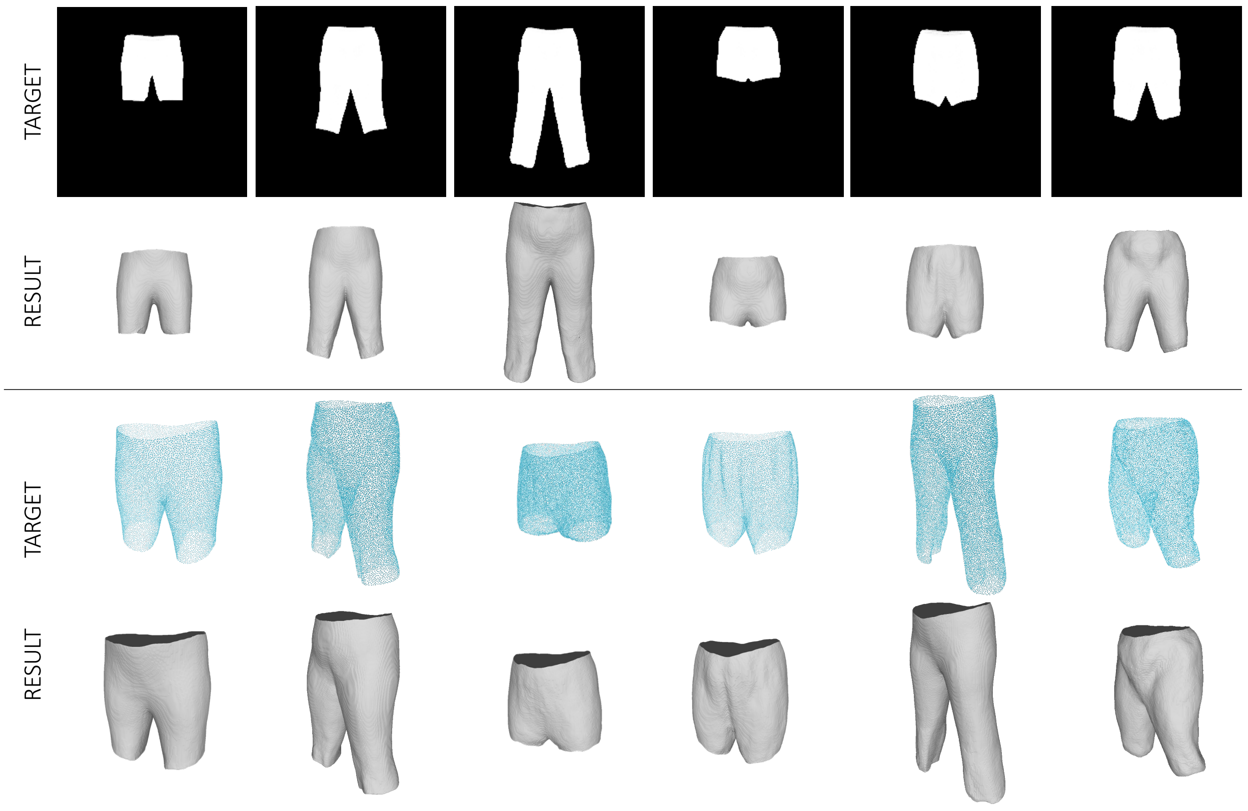


Figure 21. **Generative network: latent space optimization (bottom garments).** After training, we can explore the latent space learned by the garment generative network with gradient descent, to recover garments from 2D silhouettes (top) or 3D point clouds (bottom).

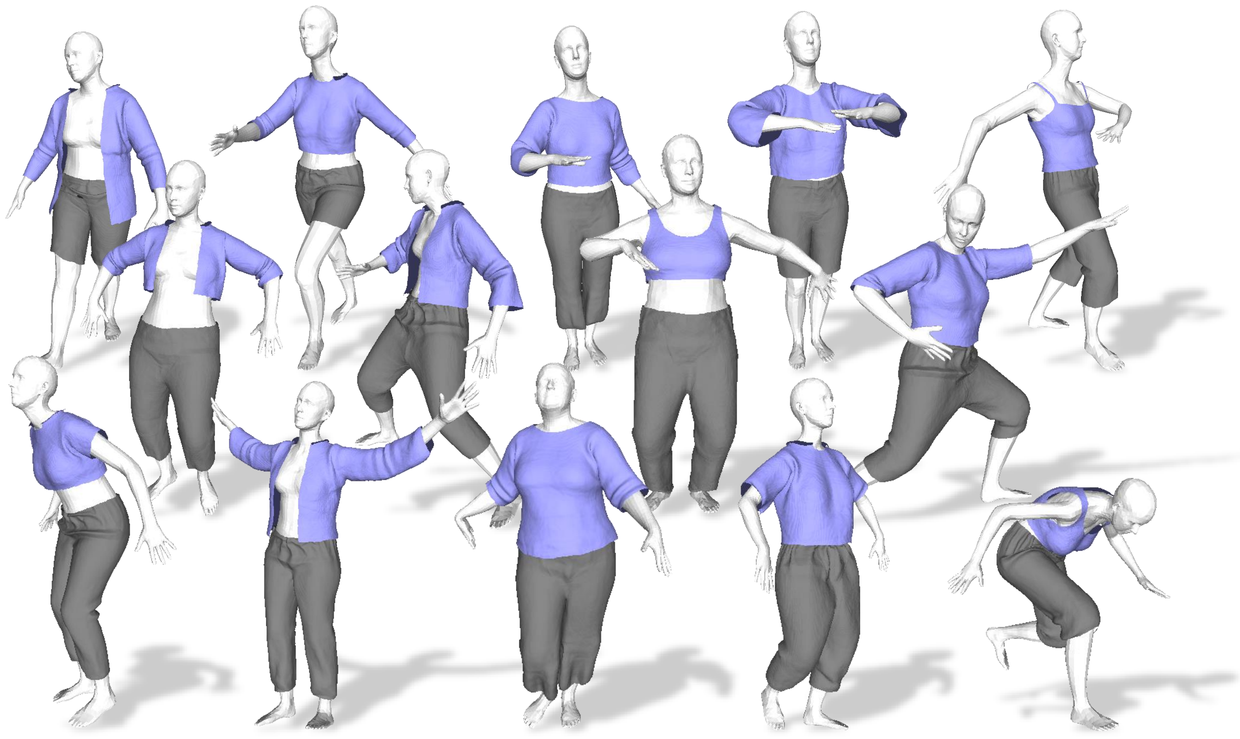


Figure 22. **Additional results:** draping garments of different topologies over bodies in various shapes and poses with our method.

and to feed it to the implicit decoder to reconstruct the explicit surface. We study here the possibility of exploring the garment latent space by the means of LSO. To do that, given a target 2D silhouette or a sparse 3D point cloud of a garment, we optimize with gradient descent a latent code – initialized to the training codes average – so that the frozen decoder conditioned on it can produce a garment which fits the target image or point cloud.

Given the silhouette $\mathcal{S}$ of a target garment, we can retrieve its latent code $\mathbf{z}$ by minimizing

$$\begin{aligned} L(\mathbf{z}) &= L_{\text{IoU}}(R(\mathbf{G}), \mathcal{S}) \ , \\ \mathbf{G} &= \text{MeshUDF}(D_G(\cdot, \mathbf{z})) \ , \end{aligned} \tag{17}$$

where L_{IoU} is the IoU loss [10] in pixel space measuring the difference between 2D silhouettes , $R(\cdot)$ is a differentiable silhouette renderer for meshes [17], and $\mathbf{G}$ is the garment mesh reconstructed with our garment decoder using $\mathbf{z}$.

In the case of a target garment provided as a point cloud $\mathcal{P}$, the garment latent code $\mathbf{z}$ can be obtained by minimizing

$$\begin{aligned} L(\mathbf{z}) &= d(ps(\mathbf{G}), \mathcal{P}) \ , \\ \mathbf{G} &= \text{MeshUDF}(D_G(\cdot, \mathbf{z})) \ , \end{aligned} \tag{18}$$

where $d(a, b)$ is the Chamfer distance [4] between point clouds a and b, and $ps(\cdot)$ represents a differentiable procedure to sample points from a given mesh [17].

In both cases, we run the optimization for 1000 steps, with Adam optimizer [8] and learning rate set to 0.01.

In Fig. 20 and Fig. 21 we present some results of the LSO procedures here described, showing that the latent space learned by the garment generative network can be explored effectively with gradient descent to recover the codes associated with the target garments.

3.2. Draping Network

3.2.1 Additional Qualitative Results

In Fig. 22 we show additional qualitative results of garment draping produced by our method, where the garment meshes are generated by our UDF model. It can be seen that our method can realistically drape garments with different topologies over bodies of various shapes and poses.

3.2.2 Euclidean Distance is not a Good Metric

In Fig. 23, we show an example of bottom garment where our result is more realistic than the competitors DeePSD [1] and DIG [9] despite having the highest Euclidean distance. This demonstrates again that Euclidean distance is not able to measure the draping quality, as discussed in the main paper.

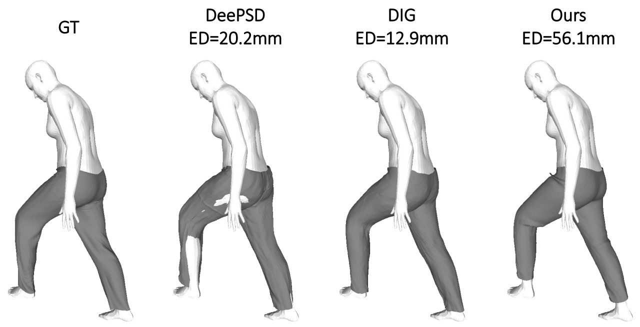


Figure 23. **Comparison between DeePSD, DIG and our method.** Our result is more realistic than the others despite having the highest Euclidean distance (ED) error.

3.2.3 Quantitative Evaluation in Physics-based Energy

In Tab. 2, we report the physics-based energy of *Strain*, *Bending* and *Gravity* as proposed by [18] on test garment meshes when draped by DeePSD, DIG and our method. These energy terms are used as training losses for our garment network (Eqs. (7) and (8) of the main paper). For the gravitational potential energy, we choose the lowest body vertex as the 0 level. Generally, our results have the lowest energies, especially for the *Strain* component. Since DeePSD and DIG do not apply constraints on mesh faces, their results exhibit much higher *Strain* energy. This indicates that our method can produce results that have more realistic physical properties.

3.3. Inference Times

We report inference times for the components of our framework, computed on an NVIDIA Tesla V100 GPU. The garment encoder, which needs to be run only once for each garment, takes ∼25 milliseconds. The decoder takes ∼2 seconds to reconstruct an explicit garment mesh from a given latent code, including the modified Marching Cubes from [5] at resolution 256^3.

The draping network takes ∼5 ms to deform a garment mesh composed of 5K vertices. Since it is formulated in an implicit manner and is queried at each vertex, its inference time increases to ∼8 ms for a mesh with 8K vertices, or ∼53 ms with 100K vertices.

3.4. Fitting SMPLicit [3] to 3D Scans

In Fig. 24 we show results of fitting the concurrent approach SMPLicit [3] to 3D scans of the SIZER dataset [21]. We can observe that they are not as realistic as ours shown in Fig. 10 of the main paper. Since we have no access to their code and not enough information for a re-implementation, we directly extract this figure from [3].

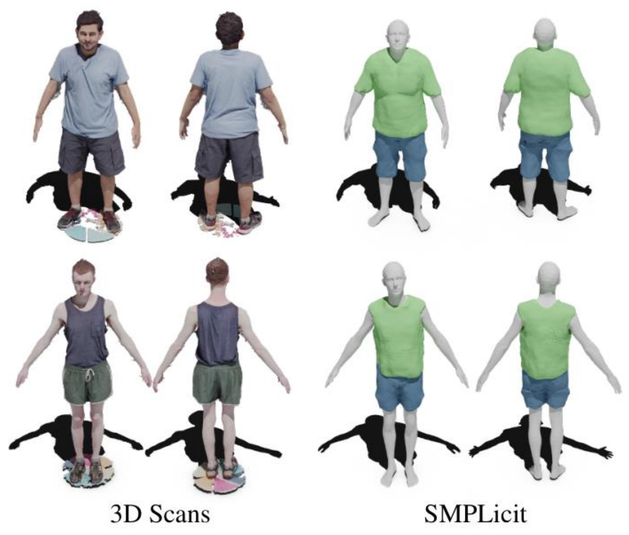
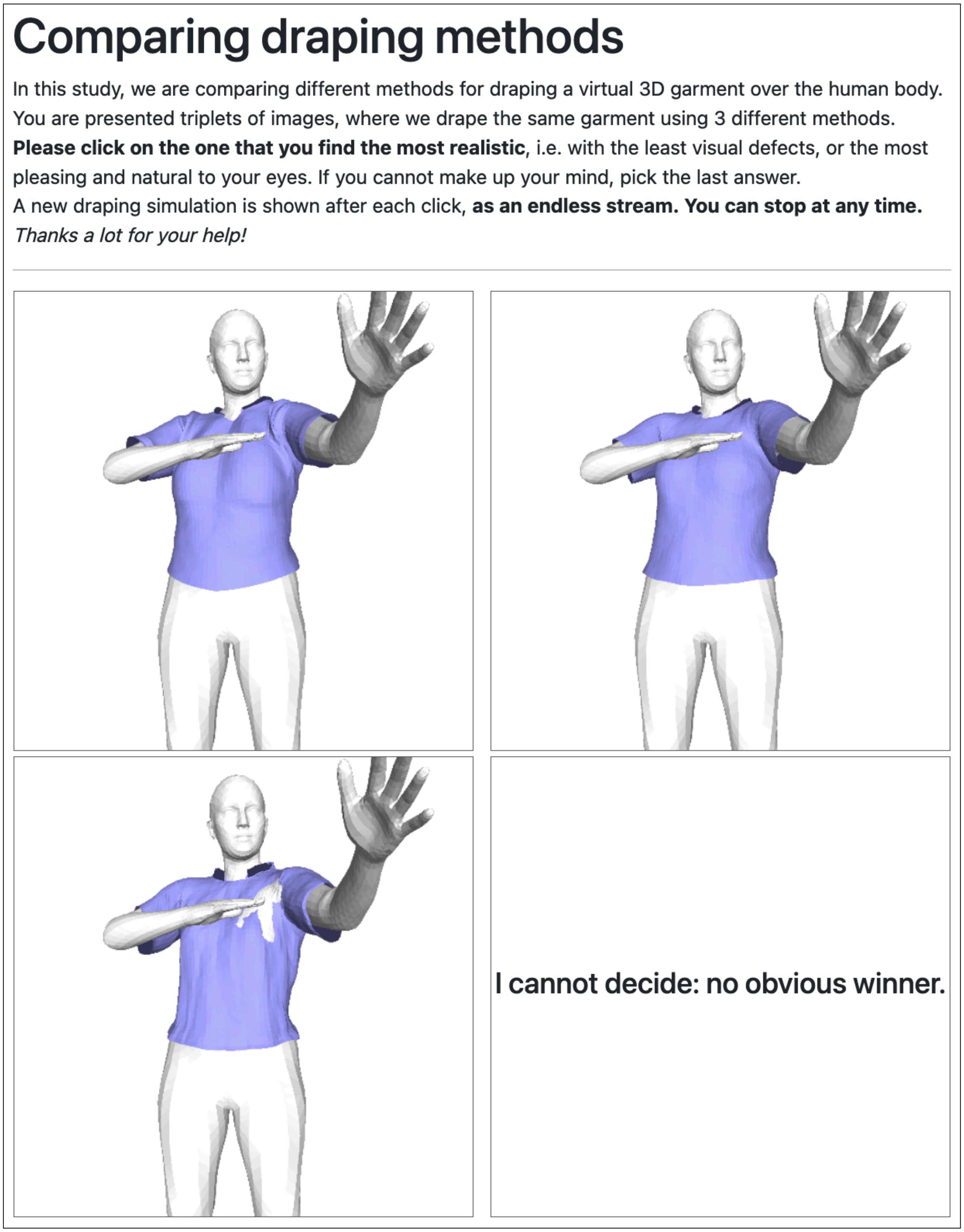


Figure 24. **Recovered garments of SMPLicit from 3D scans**. Figures are extracted from [3].

4. Human Evaluation

In Fig. 25 we show the interface and instructions that were presented to the 187 respondents of our survey. These evaluators were volunteers with various backgrounds from the authors respective social circles, which were purposely not given any further detail or instruction. We collected collected 3738 user opinions in total, each user expressing 20 opinions on average.


Comparing draping methods

In this study, we are comparing different methods for draping a virtual 3D garment over the human body. You are presented triplets of images, where we drape the same garment using 3 different methods. **Please click on the one that you find the most realistic**, i.e. with the least visual defects, or the most pleasing and natural to your eyes. If you cannot make up your mind, pick the last answer.
A new draping simulation is shown after each click, **as an endless stream. You can stop at any time.**
Thanks a lot for your help!

I cannot decide: no obvious winner.

Figure 25. **Interface of our qualitative survey.** The garment is draped with our method, DIG, and DeePSD, in a random order.

Top	Strain ↓	Bending ↓	Gravity ↓	Total ↓
DeePSD	7.22	**0.01**	**0.98**	8.21
DIG	6.32	**0.01**	1.05	7.38
Ours	**0.43**	**0.01**	1.05	**1.81**

Bottom	Strain ↓	Bending ↓	Gravity ↓	Total ↓
DeePSD	8.46	0.02	0.90	9.38
DIG	7.48	**0.01**	0.90	8.39
Ours	**0.41**	**0.01**	**0.86**	**1.28**

Table 2. **Draping unseen garment meshes.** Quantitative comparison in physics-based energy between DeePSD, DIG and our method. “Strain”, “Bending” and “Gravity“ denote the membrane strain energy, the bending energy and the gravitational potential energy, respectively.

References

[1] H. Bertiche, M. Madadi, E. Tylson, and S. Escalera. DeePSD: Automatic Deep Skinning and Pose Space Deformation for 3D Garment Animation. In *International Conference on Computer Vision*, 2021. 8

[2] T. Buffet, D. Rohmer, L. Barthe, L. Boissieux, and M-P. Cani. Implicit untangling: A robust solution for modeling layered clothing. *ACM Transactions on Graphics*, 38(4):1–12, 2019. 5

[3] E. Corona, A. Pumarola, G. Alenya, G. Pons-Moll, and F. Moreno-Noguer. Smplicit: Topology-Aware Generative Model for Clothed People. In *Conference on Computer Vision and Pattern Recognition*, 2021. 1, 5, 9

[4] H. Fan, H. Su, and L. Guibas. A Point Set Generation Network for 3D Object Reconstruction from a Single Image. In *Conference on Computer Vision and Pattern Recognition*, 2017. 5, 8

[5] B. Guillard, F. Stella, and P. Fua. MeshUDF: Fast and Differentiable Meshing of Unsigned Distance Field Networks. In *European Conference on Computer Vision*, 2022. 9

[6] E. Gundogdu, V. Constantin, S. Parashar, A. Seifoddini, M. Dang, M. Salzmann, and P. Fua. Garnet++: Improving Fast and Accurate Static 3D Cloth Draping by Curvature Loss. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):181–195, 2022. 5

[7] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*, 2015. 3

[8] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimisation. In *International Conference on Learning Representations*, 2015. 3, 8

[9] R. Li, B. Guillard, E. Remelli, and P. Fua. DIG: Draping Implicit Garment over the Human Body. In *Asian Conference on Computer Vision*, 2022. 5, 8

[10] R. Li, M. Zheng, S. Karanam, T. Chen, and Z. Wu. Everybody Is Unique: Towards Unbiased Human Mesh Recovery. In *British Machine Vision Conference*, 2021. 8

[11] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. Occupancy Networks: Learning 3D Reconstruction in Function Space. In *Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019. 3

[12] Ben Mildenhall, S. P. P., M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *European Conference on Computer Vision*, 2020. 3

[13] G. Moon, H. Nam, T. Shiratori, and K.M. Lee. 3d clothed human reconstruction in the wild. In *European Conference on Computer Vision*, 2022. 5

[14] R. Narain, T. Pfaff, and J.F. O'Brien. Folding and crumpling adaptive sheets. *ACM Transactions on Graphics*, 2013. 5

[15] R. Narain, A. Samii, and J.F. O'brien. Adaptive anisotropic remeshing for cloth simulation. *ACM Transactions on Graphics*, 2012. 5

[16] T. Pfaff, R. Narain, J.M. De Joya, and J.F. O'Brien. Adaptive tearing and cracking of thin sheets. *ACM Transactions on Graphics*, 33(4):1–9, 2014. 5

[17] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. PyTorch3D. https://github.com/facebookresearch/pytorch3d, 2020. 8

[18] I. Santesteban, M.A. Otaduy, and D. Casas. SNUG: Self-Supervised Neural Dynamic Garments. In *Conference on Computer Vision and Pattern Recognition*, 2022. 9

[19] I. Santesteban, M.A. Otaduy, N. Thuerey, and D. Casas. Ulnef: Untangled layered neural fields for mix-and-match virtual try-on. In *Advances in Neural Information Processing Systems*, 2022. 5

[20] M. Tang, T. Wang, Z. Liu, R. Tong, and D. Manocha. I-Cloth: Incremental Collision Handling for Gpu-Based Interactive Cloth Simulation. In *ACM Transactions on Graphics*, 2018. 5

[21] G. Tiwari, B. L. Bhatnagar, T. Tung, and G. Pons-Moll. Sizer: A Dataset and Model for Parsing 3D Clothing and Learning Size Sensitive 3D Clothing. In *European Conference on Computer Vision*, 2020. 9

[22] H. De Vries, F. Strub, J. Mary, H. Larochelle, O. Pietquin, and A.C. Courville. Modulating Early Visual Processing by Language. In *Advances in Neural Information Processing Systems*, 2017. 2, 3

[23] Y. Wang, Y. Sun, Z. Liu, S. Sarma, M. Bronstein, and J.M. Solomon. Dynamic Graph CNN for Learning on Point Clouds. In *ACM Transactions on Graphics*, 2019. 1, 2