CLOTH3D: Clothed 3D Humans
Supplementary material

Hugo Bertiche[1,2][0000−0002−6632−1902], Meysam Madadi[1,2][0000−0002−7384−5712], and Sergio Escalera[1,2][0000−0003−0617−8873]

[1] Universitat de Barcelona, Spain
[2] Computer Vision Center, Spain
hugo_bertiche@hotmail.com

This document provides additional details about CLOTH3D dataset and ablation analyses. First, Sec.1.1 provides further descriptions on the data generation process. Secondly, Sec.1.2 describes the variability and size of the dataset. Then, Sec.1.3 gives specific details on data format. Along with this document a video is submitted. The content of the supplementary video is described in Sec.2. Later, in Sec.3 we provide more details about the implemented Graph CNN (Sec. 3.1 and 3.2), further discussions about wrinkle factorization (Sec. 3.3), qualitative generated results from the learnt latent space (Sec. 3.4), and analysis on the DVAE error distribution (Sec. 3.5). Finally, Sec. 4 describes theoretical potential applications of CLOTH3D dataset.

1 CLOTH3D

1.1 Dataset generation details

In this section we explain in more detail relevant aspects of the dataset generation process. First, we introduce a summary of the generation steps. Then, we start on the human sequence generation (Sec.3.1 on main paper), with the specifics on self-collision solving. Later, we move to garment generator (Sec.3.2 on main paper) shaping step, where a formal description is given, plus a clarification on how to keep tightness semantically meaningful w.r.t. SMPL shape displacements. Finally, we describe an important pre-requisite for simulation (Sec.3.3 on main paper) that allows the simulation of any sequence from rest pose.

Generation algorithm. Here we briefly summarize the steps performed by our generator:

1. HUMAN
   1.1. Pick SMPL parameters
   1.2. Compute SMPL body sequence
   1.3. Solve self-collisions
2. OUTFIT
   2.1. Pick template garments
   2.2. Shape sleeves/legs/skirt
   2.3. Cut
   2.4. Resize
   2.5. Sew garments into jumpsuit/dress (optional)

3. SIMULATION
   3.1. Fabric settings
   3.2. Body shape transition (from $\beta + \gamma$ to β)
   3.3. Pose transition
   3.4. Simulate sequence

Self-collision. As explained on Sec.3.1 of the main paper, SMPL can present self-collisions, specially on samples with challenging shapes. This makes cloth simulation difficult for many combinations of shape and pose. In order to overcome this and drastically increase the amount of valid samples, we propose a simple, fast self-collision solving step. Through visual inspection, we identified problematic body regions (armpit, crotch, ...) on which to detect and solve collisions. Using SMPL segmentation, we separate vertices belonging to different body parts. Collisions appear as intersection of pairs of segments. For each of these pairs, we test edges of a segment vs faces of the other, and viceversa. Since we identified problematic regions, the number of edge-vs-face tests is significantly reduced. This yields a set of intersection points to which we approximate a plane. Then, each collided body vertex is moved to the corresponding side of this plane based on segment index. We leave a separation of 4mm as for simulation, a 2mm margin is used for cloth-body collisions.

Resizing. In Sec.3.2 of the main paper, we explained how we resized garments with SMPL shape parameters plus an offset to represent garment tightness. While first and second shape parameters represent overall size and fatness respectively, the sign of the first parameter has opposite meaning for male and female. To take this into account and so that tightness remains semantically consistent, the sign of the first offset is $(-1)^{g+1}$, where g is gender (0 = female and 1 = male). This means a positive tightness shall produce smaller garments.

Body transition. Outfit generation process yields garments on rest pose resized to SMPL shape plus tightness. We need a body transition from this state to the state of the initial frame of the sequence for a correct simulation. To ensure no body-to-cloth penetration is present due to resizing to a different shape, we generate a few frames of transition where the body shape changes from $\beta + \gamma$ (shape+tightness) to β. Finally, more frames are devoted to a transition from rest pose to the initial pose of the sequence split. Pose transition is computed based on quaternions for a smooth posing.

1.2 Variability and Size

Each template garment is shaped with linear deformations, similar to SMPL shaping, where coefficients are uniformly sampled to yield a balanced distribution. Garments are cut at uniformly sampled lengths along limbs and waist/torso. With this we potentially obtain all possible combinations of sleeve length and t-shirt length, or leg length and waist height. Furthermore, upper-body garments have specifically designed cuts that change shape and topology, also uniformly sampled. Finally, on resizing stage, a tightness two-dimensional factor is uniformly sampled from $[-1.5, 0.5]$ (biased to loose garments for more complex dynamics and wrinkles), further increasing garment variability. All these randomly generated properties, once combined, almost guarantee that

an outfit will never appear twice in the dataset. Afterwards, different simulation properties will also ensure cloth shall behave differently. Each pose sequence from source data is downgraded from 60fps to 30fps and split into 300 frames subsequences, each of them simulated once with a different outfit, thus totalling around 2M frames.

1.3 Data format

Each sample is a 600-frames split of the original sequence on source data, downgraded from 60fps to 30fps, totalling 300 frames for each split. Note that frames refer to time instants, not images. The name of the sample contains the name of the original sequence and the number of the split (e.g.: '01_01_s0', sequence is '01_01' and split is 's0'). As the number of frames of original sequences is not a multiple of 300, some splits will have smaller length. Each sample has static and dynamic garment information. **Static** information is the outfit fitted to rest pose for the corresponding body shape. Static garments are represented by OBJ files, which include vertices on rest position and topology data. The **dynamic** information contains garment animation data. 3D animation data is usually stored as PC2 (Point Cache 2) files. We propose the PC16 format, a PC2 conversion from 32-bit floats to 16-bit, halving dataset space requirements. Precision loss on this conversion is none to minimal within the range $[-1, 1]$ ($\leq 2^{-11}$) and insignificant in $[-2, 2]$ ($\leq 2^{-10}$). By storing garment vertex positions relative to SMPL body root joint, we ensure values will always be in range $[-2, 2]$. Sample metadata contains SMPL parameters, garment names and their fabrics. To ease the fitting process, rest pose is redefined such that legs are slightly open, due to the high geometric complexity of this region. Average number of vertices per outfit is around 20K, which implies an average size of $35 - 40$MB per sequence.

Fig. 1 shows random static samples. Fig. 2 shows random frames of sequences. Finally, Fig. 3 shows random samples of different sequences with different representation modalities that can be obtained from CLOTH3D data: depth maps, surface normals, 3D velocities and segmentation masks.

2 Video

We provide a video supplementary material which visualizes a few samples of CLOTH3D. We further show different data representations provided at the dataset for those samples. The top-left image of the video is a 3D visualization as RGB data. We chose high quality lighting and shaders to make more evident the geometric details of the 3D models. On top-right part of the image, we show a rendering of depth information w.r.t. camera. Depth maps are normalized per frame. Then, the bottom-left image shows an encoding of 3D surface normals as RGB colors. Since normals are already within range $[0, 1]$, only a mapping to RGB is performed. This means each color channel represents a 3D coordinate of the normal vectors. Finally, the bottom-right image renders 3D velocities. As for normals, they are encoded as RGB data, and, as depth, it is normalized per frame. Due to normalization, frames with low motion appear noisy.

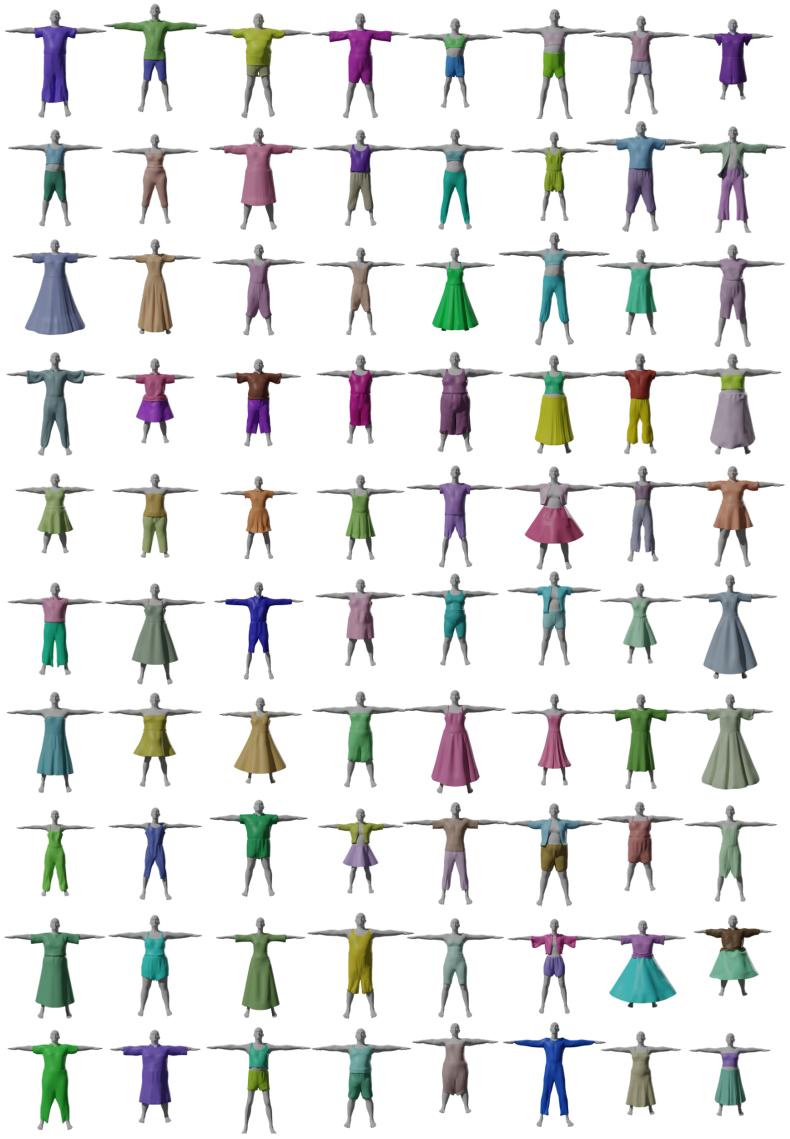


Fig. 1: CLOTH3D static samples.

Fig. 2: CLOTH3D dynamic samples.

Fig. 3: CLOTH3D data representations. From left to right: RGB, depth, normals, velocities, and segmentation masks.

3 Dressed human generation

As we explained in Sec.4 in the main paper, we apply a variational autoencoder with graph convolutional neural (GCN) blocks to learn some latent codes. Later in test time we sample from the latent space to generate dressed humans. We condition the model on the garment type, tightness, body shape and pose. Conditioning on the garment type can be done by a one-hot encoding of the garment category (e.g. T-shirt, dress, etc.). Although we have fixed garment categories in the dataset, their variability in size, shape or type allows us to encode garments into a continuous space, which is more meaningful than fixed categories to be conditioned on. Therefore we train our model on static garments (in the rest pose) to learn the garment type code. We briefly remind the notations from the main paper for the easiness of reading. Here, we provide additional details of the implemented GCN, Sec.3.1. Then, Sec.3.3 explains our wrinkles factorization procedure. Finally Sec.3.4 shows some generated results.

We train SVAE on static garments and DVAE on dynamic garments and refer to learnt latent codes as garment code ($z_s \in \mathbb{R}^{128}$) and wrinkle code ($z_d \in \mathbb{R}^{128}$), respectively. Here, we provide DVAE details. DVAE has three main modules: encoder $\{cvar^z, z\} = \Psi(\bar{X}, \bar{T}; \tau^\Psi)$, conditioning $\{cvar, cvar^z\} = \Gamma(cvar; \tau^\Gamma)$ and decoder $\{\bar{X}, M\} = \Phi(z, cvar^z; \tau^\Phi)$. $X \in \mathbb{R}^{V_T \times 3}$ are the offsets computed on dynamic samples, $\bar{X}$ is a normalization of X, $T \in \mathbb{R}^{V_T \times 3}$ and $\bar{T}$ are body vertices and its normalization, $cvar$ is the stacking of conditioning variables (body shape β, garment tightness γ, body pose θ and garment code z_s), $cvar^z$ is the middle layer features of autoencoder network Γ, $M \in \mathbb{R}^{V_T \times 1}$ is the garment mask and $\tau^\cdot$ are networks weights. Finally, $C \in \mathbb{R}^{V_T \times 3}$ are the garment vertices after registration on top of T.

3.1 Graph convolutions

Sec.4.2 of the main paper describes the model architecture. Here we focus on the formulation for graph convolutions we implemented. Our network input data is defined by V_T and topology. This allows to use graph convolutional filters to learn features.

Following the definition of spectral graph convolution, filtering is computed as:

$$y = g_\omega(\mathbf{L})x = \sum_{i=0}^{K} \omega_i \mathbf{T}_i(\hat{\mathbf{L}})x, \tag{1}$$

where ω_i are the learnable filters, $\mathbf{L}$ is the normalized Laplacian matrix, $\hat{\mathbf{L}} = 2\mathbf{L}/\lambda_{max} - \mathbf{I}$, and $\mathbf{T}_i(\hat{\mathbf{L}})$ is the i-th Chebyshev polynomial order. Given an input graph with F_{in} features for each node, the described convolution will return that same graph with a different set of features F_{out}. Chebysev polynomial order defines the size of the receptive field K, meaning feature filtering aggregates the K-ring neighbourhood for each node. In our case, we keep an small neighborhood with $K = 1$. Therefore to have a high receptive field we build a deep network. Each convolutional and pooling layer further combines node features with higher K-ring neighbours. We also include skip connections throughout the whole network. This leads to an effective information passing helping to learn fine-grain garment details.

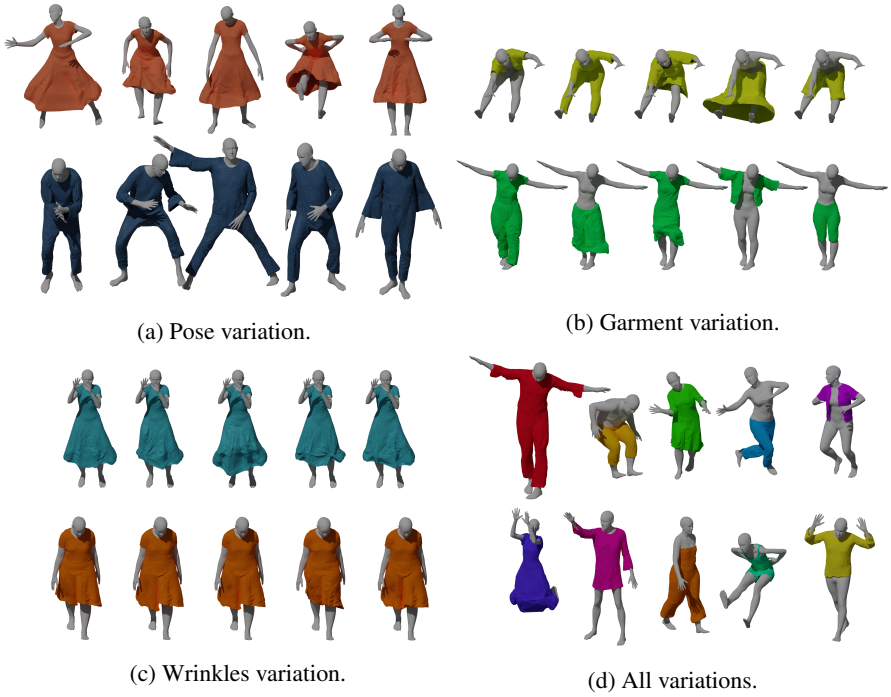
3.2 Architecture details and justification

As a standard operation in CVAE, conditional variables should be fed into the encoder. However, these variables ($cvar$) are not balanced with offsets (X) in terms of size and scale. $cvar$ can be partially decoded to body vertices $\bar{T}$ and offsets $\bar{X}$. Therefore, we concatenate $\bar{T}$ to $\bar{X}$ and feed it to the network Ψ. To better factor out $cvar$ from latent code z we include an additional MLP branch at the end of the encoder and before sampling layer. The goal of this branch is to regress $cvar$ during training. It also helps to have a more stable training. Regularizing the encoder by regressing $cvar$ has a limitation when the dimensionality of $cvar$ is high (e.g. $cvar_d$), that is, optimization can stick to local minima. Therefore, we regress $cvar^z$ instead of $cvar$. Finally, decoder generates normalized offsets $\bar{X}$ and garment mask M in two branches at the last layer. Note that DVAE does not have branch M in the decoder. Note that we train our GCN with dual topology to handle skirt-like garments vs. the rest which has not been done before in 3D garment reconstruction.

3.3 Wrinkle factorization vs. the rest conditional variables

We introduce two versions of DVAE. In one version, we computed offsets X as $C - T$ and concatenated with T as input to the network (as explained in the main paper). In this version, offsets are not invariant to pose. To better factorize z_d to code garment wrinkles, we also implemented another version of DVAE where offsets are partially invariant to pose and garment type. Let $\mathcal{K}(A, \theta, J)$ be forward kinematic function that receives articulated object $A \in \mathbb{R}^{V_A \times 3}$ with joints $J \in \mathbb{R}^{V_J \times 3}$ and transforms it w.r.t. the rotations (or pose) θ. Note that A and J are in rest pose in this definition. Here, A can be replaced by C or T. Since we register garments on top of body, garments can be transformed by body joints. Given this definition, $\mathcal{K}^{-1}$ can be defined as the inverse of the kinematic function which transforms back the object to its rest pose.

We define new offsets as $X^{new} = \mathcal{K}^{-1}(C_d, \theta, J) - C_s$ where C_d is the dynamic dressed garment in pose θ and C_s is the static garment in rest pose. We then concatenate



(a) Pose variation.

(b) Garment variation.

(c) Wrinkles variation.

(d) All variations.

Fig. 4: Generated samples from learnt latent codes, conditioning on different variables for dynamic garments (DVAE network).

X^{new} with C_s and T, normalize it and feed the network. Finally, at the output of the network, garments are reconstructed by the reverse process. Since we train SVAE to reconstruct C_s, we can rely on it at test time to generate dressed garments. We train this network with the same loss functions as before. By doing this factorization on the offsets, we are able to feed the network with relevant information of the wrinkles and factorize z_d in a more meaningful way.

3.4 Wrinkles factorization results

In Fig. 4 we show some qualitative results of the learnt latent space and conditioning on different variables, specifically pose (θ), garment code (z_s) and wrinkle code (z_d). One can see the network can learn a meaningful and consistent space. Regarding the learnt wrinkle code (Fig. 4c), a rest pose (upper row) shows less wrinkle variability than a complex action category (lower row). This is while by conditioning on pose (Fig. 4a) or garment code (Fig. 4b), we can accurately retarget fixed wrinkle codes to new scenarios.

3.5 Dynamic Garment Generation error.

As explained in the main paper, this work proposes encoding garments into SMPL body (subdivided for higher resolution) as offsets. This creates and implicit registration er-

ror, we represent its effects on Fig.5a-c. Additionally, we analyze the error distribution through the body. Per vertex reconstruction error is shown in Fig. 5 for static and dynamic samples. It can be seen the error is higher near the feet. This is because of the dynamics of skirts and dresses, which have the highest error (see Tab.4 on the main paper). Interestingly, trousers have the lowest error among others.

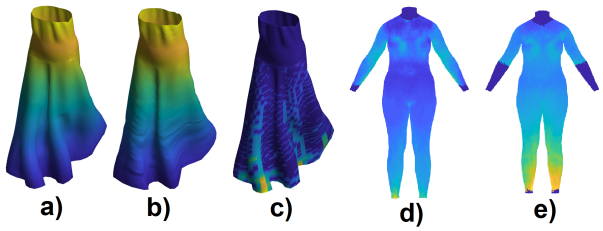


Fig. 5: Implicit registration and S/DVAE per vertex error. a) original garment. b) reconstruction from offsets and body vertices. c) original vs. reconstruction error heatmap (scale in cm.). d-e) SVAE vs. DVAE garment generation error heatmaps. Dark blue=0, yellow$>$10cm.

3.6 Post-processing

Since garments are encoded as SMPL offsets, their meshes follow the same connectivity of vertices as the human body model, except for dresses and skirts which follow the topology proposed at Sec. 4.1 on the main paper. Then, garment vertices are selected according to the predicted mask. In practice, this yields noisy boundaries. Fortunately, this is easily solvable through a simple post-processing. Garment's boundary vertices are identified by checking their neighbour's mask prediction. A normalized Laplacian matrix $\mathbf{L} \in \mathbb{R}^{14475\times14475}$ is constructed such that it represents a new connectivity where only boundary vertices are linked by edges. Then, smoothing boundaries can be achieved by iteratively multiplying the garment vertices $\mathbf{X} \in \mathbb{R}^{14475\times3}$ as:

$$\mathbf{X}_{i+1} = \mathbf{L}\mathbf{X}_i, \tag{2}$$

where i is the iteration count. Empirically, 10 iterations are more than enough to produce visually appealing results, while increasing iterations might destroy high-frequency details. This smoothing can be efficiently computed as sparse matrix multiplication. In Fig.6 are depicted a few samples before and after applying the described post-processing. We observe how the proposed method can improve noisy boundaries and generate more natural-looking garments.

4 Applications of the Dataset

CLOTH3D could be used not just for 3D garment generation but in other application scenarios, such as human pose and action recognition in depth images, garment motion

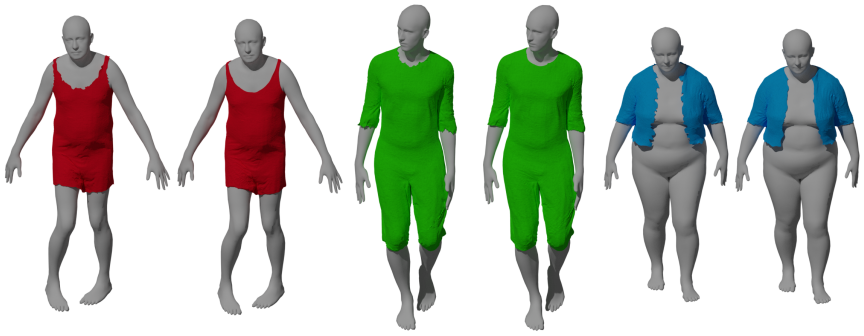


Fig. 6: Boundaries are noisy as mask prediction is noisy at boundaries as well. With the proposed post-processing we can mitigate most of this noise, yielding visually appealing garments. Left: raw prediction. Right: post-processing results.

analysis, filling missing vertices of scanned bodies with additional meta data (e.g. garment segments), support designers and animators tasks, or estimating 3D garment from RGB images, just to mention a few. We ran some proof-of-concept applications using our CLOTH3D data, shown in Fig. 7: a rendered depth image, garment motion velocities, and RGB-to-3D cloth estimation. For the later, given our layered garment structure and SMPL segmentation, we rendered 10K samples of t-shirts and trousers with different poses of Human3.6M [**?**] dataset. These data contains images of body segments and garment silhouette. We then trained ResNet50 to regress available static garment codes. In test time, we assume body shape, pose and garment silhouette are available. This information can be extracted by state-of-the-art SMPL based pose estimation and cloth parsing methods. In our case, we manually segmented garments of two frames of this dataset (shown in Fig. 7(c)) and used them to estimate garment code. We then copied the wrinkles from the nearest sample in CLOTH3D. Finally 3D garments were reconstructed and rendered as shown.

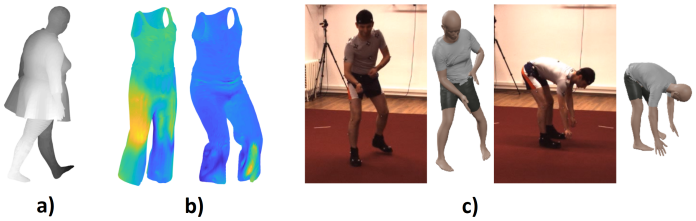


Fig. 7: Applications of CLOTH3D. a): A rendered depth image, b): vertex velocities (Dark blue=0, yellow>0.8.), c): Automatic RGB to 3D cloth estimation.