Data-Driven Clothing for Interactive Applications

Kangrui Xue*
Stanford University
kangruix@stanford.edu

Jane Wu
Stanford University
janehwu@stanford.edu

1 Introduction

Traditional cloth simulation methods work by treating cloth deformation as a partial differential equation with mass-spring (or other finite element) models. Though capable of producing highly detailed, realistic clothing, the delicate time-stepping schemes and sheer number of collisions leave them notoriously slow [3]. In applications like animation (where such procedural methods have become ubiquitous), artists can afford to wait longer for higher quality renderings. However, for interactive applications that demand real-time clothing without compromising fidelity (i.e. e-commerce, gaming, virtual reality), an alternate approach is needed.

Recently, data-driven cloth has emerged as a promising solution. Motivated by the success of neural networks in predicting T-shirt shape via offsets from a skinned T-shirt mesh [1, 2], we expand the scope of previous work by generalizing to full-body, loose clothing. Whereas tight clothing (i.e. T-shirts) exhibits consistent behavior for all poses, loose clothing may behave differently depending on whether the body is stationary or moving. In particular, we adopt a two-stage pipeline:

1. Large-scale, offline procedural simulation to generate a dataset (outputs) for a coat.
2. Training a fully-connected neural network to predict simulated coat outputs, given input pose parameters and skinned cloth meshes.

By leveraging the detailed realism of procedural cloth simulation methods and augmenting them with data-driven approaches, we achieve a close approximation of physics-based cloth – albeit now in real-time. In this paper, we illustrate our data-driven cloth pipeline on an assortment of poses commonly found in interactive applications. We analyze, both quantitatively and qualitatively, the performance of different neural network architectures on real-time coat shape prediction and, with these insights in mind, conclude with suggestions for future research.

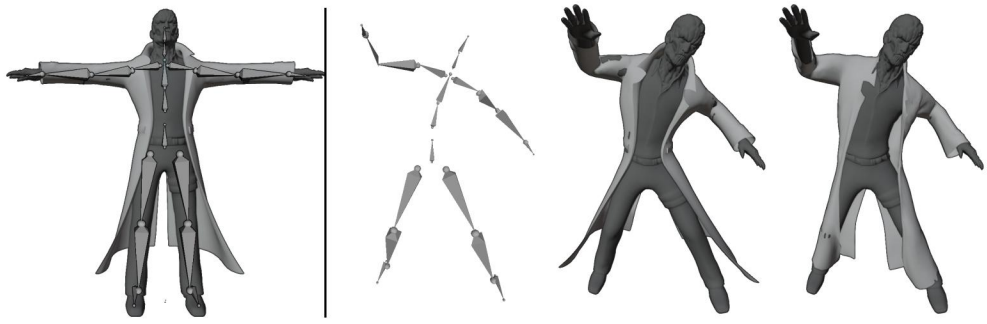
2 Dataset

Our dataset consists of skinned and simulated coat meshes corresponding to 10,000 randomly sampled poses. Each example is generated according to the three-stage pipeline outline in Figure 1. For input features, we use input joint rotation quaternions (14 joints $*$ 4 (w,x,y,z) = 56 dimensions), since they are easily accessible and more physically meaningful than alternative features like skinned coat mesh vertices. Our output consists of vertex offsets (2,170 vertices per mesh * 3 (x,y,z) = 6,510 dimensions) from each skinned coat mesh to a predicted coat mesh.

As a pre-processing step, we compute the squared L_2 norm (distance) of these (x,y,z) offsets averaged over all vertices. We then remove all examples above a distance threshold of 0.0350, since inspecting them revealed faulty simulations or unreasonable behavior (i.e. sliding off the body). Of the remaining 9,813 examples, we perform a 6,000–2,000–1,813 Training–Validation–Testing split. The overall distribution of distances and the two examples with the lowest and highest distance are visualized in Figure 2. Qualitatively, poses where the legs lean forward exhibit greater distance, since the coattails closely follow the skeleton during skinning but fall to a more neutral state after simulation.

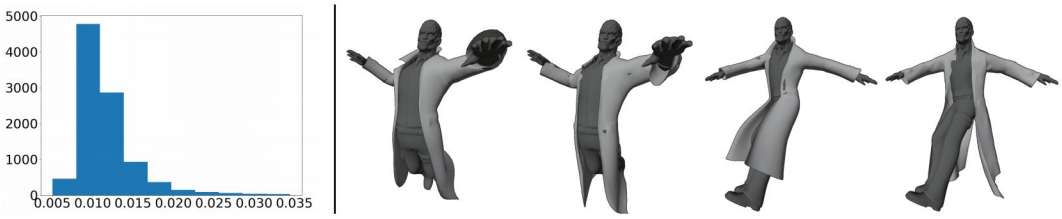
*CS 229 student

Preprint. Under review.



(a) (b) (c) (d)

Figure 1: Our three-stage data generation pipeline: We start with T-pose (a). First, joint rotations are randomly sampled from the range of human motion to construct pose (b). The body and coat meshes (c) are then skinned (in Blender) to match this pose. Thirdly, the output coat mesh (d) is simulated with gravity, elastic and damping forces, and friction [3] until the arms reach static equilibrium.



(a) (b) (c) (d) (e)

Figure 2: (a) Histogram of number of examples per squared distance bin (average squared distance = 0.0117). Lowest squared distance = 0.0050: (b) skinned coat vs. (c) simulated coat. Highest squared distance = 0.0345: (d) skinned coat vs. (e) simulated coat.

3 Methods

3.1 Principal Component Analysis

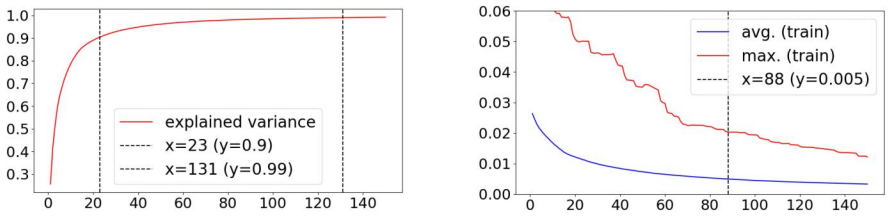
Previous work [4] in data-driven clothing leverages principal component analysis (PCA), a classical method for dimensionality reduction, to drive cloth dynamics in real-time. In practice, PCA is carried out via eigendecomposition of the dataset's covariance matrix, with the principal eigenvector representing the basis that preserves the most variance (and so on). For example, if we want to reduce our data x_i down to k dimensions, we should keep the first k eigenvectors u_l ($l = 1...k$) and reconstruct an approximation of $\vec{x}$ according to:

$$x_i = \left(\sum_{l=1}^{k} c_l u_l \right) + \mu$$

where μ is the sample mean, c_l is the l-th PCA coefficient given by $proj_{u_l} x$. With this formulation, we perform PCA on our vertex offsets to reduce output dimension during the supervised learning stage. This in turn lowers computational cost and helps prevent overfitting.

3.2 Fully-Connected Neural Networks

For the primary task of predicting output offsets, we implement a fully-connected neural network in PyTorch. The network takes in joint rotation quaternions (56 dimensions) and predicts the PCA coefficients c_l from the PCA reconstruction equation above. The number of hidden layers, hidden layer dimensions, activation functions, as well as the number of PCA components used (or whether to bypass PCA and predict vertex offsets directly) are all parametrizable.



(a) (b)

Figure 3: Statistics of running PCA on our output offsets. (a) Plot of cumulative explained variance over the first k principal components. (b) Plot of vertex error from reconstructing the simulated coat using the first k principal components

Table 1: Network Architecture Ablation Study

ID	Hidden Layers	Output (dim.)	Training Cost	Testing Cost
0	None (linear regression)	Vertex offsets (6,510)	3.855e-4	3.832e-4
1	relu_128_256	Vertex offsets (6,510)	1.002e-4	1.592e-4
2	relu_128_256	256 PCA coeffs (256)	1.109e-4	1.527e-4
3	relu_128_256	128 PCA coeffs (128)	**0.989e-4**	**1.498e-4**
4	relu_128_256	64 PCA coeffs (64)	1.051e-4	1.568e-4

We use MSE loss as our performance metric. Explicitly, given observed offsets x_i and predicted offsets $\hat{x_i}$, our loss function is defined as:

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x_i})^2 = \frac{1}{n}\sum_{i=1}^{n}[x_i - (\sum_{l=1}^{k}\hat{c_l}u_l + \mu)]^2$$

When the network is configured to predict PCA coefficients, we append a "PC layer" that reconstructs vertex offsets and whose weights and biases (principal components u_l, sample mean μ) are fixed. Finally, the network is trained using the Adam optimizer with a batch size of 32 for 1000 epochs.

3.3 Linear Regression Baseline

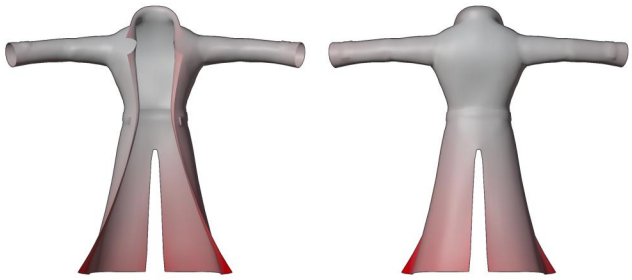
Though neural networks are the focus of our experiments, we use a linear regression model as a baseline. In fact, linear regression is equivalent to a neural network with zero hidden layers (which is how we implement it). Our model takes in joint rotation quaternions (56 dimensions) and applies a linear transformation to obtain vertex offsets (6,510 dimensions).

Because linear blend skinning (LBS) – the technique we use to compute our skinned coat meshes – also depends on a linear combination of joint rotations, we liken our linear regression model to an idealized, physics-aware skinning model of sorts. As such, our baseline performance can be conceptualized as an estimate of the optimal achievable performance from skinning alone.

4 Results

4.1 Quantitative Benchmarks

As an initial test, we examine the trade-off between number of PCA coefficient outputs and network predictive power. The results are shown in Table 1. For consistency, we use hidden layer dimensions of 128 and 256, ReLU activation functions, and a learning rate of $1e-4$ for all architectures – though experimentally, these hyperparameters were close to optimal across the board. Overall, an output of 128 PCA coefficients (neural network 3) achieved both the lowest training error and lowest testing error. Outputting more PCA coefficients tended to overfit the training set, and outputting less PCA coefficients failed to capture enough variance in the original data.



(a) (b)

Figure 4: Squared per-vertex error, averaged over all test examples. We take the square root of the errors to reduce variance and map these results to values of red from 0.0 to 1.0: (a) front-view of coat, (b) back-view of coat.

Table 2: Cloth Simulation Speed

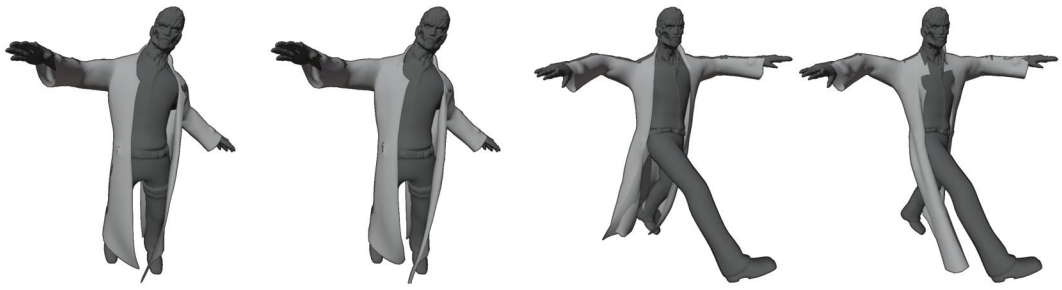
Method	Considerations	Avg. Runtime
Procedural (from [3])	C++ (PhysBAM), CPU	$\sim$ 180 s
Linear regression	Python (PyTorch), CUDA	0.991e-4 s
Neural network 3	Python (PyTorch), CUDA	1.069e-4 s

Having validated neural network 3 as the optimal architecture, we then run inference on the test dataset to generate 1,813 predicted coat meshes. We compute the squared per-vertex error, as visualized in Figure 4. As expected, looser parts of the cloth (i.e. coattails) exhibit larger error.

Finally, because interactive applications demand real-time cloth simulation, we compute average simulation times for high-performance procedural cloth simulation code (from [3]), our linear regression, and our neural network 3. The results are shown in Table 2. Because the procedural cloth simulation is implemented in C++ (PhysBAM), the comparison to PyTorch models is merely intended to provide an approximate sense of scale.

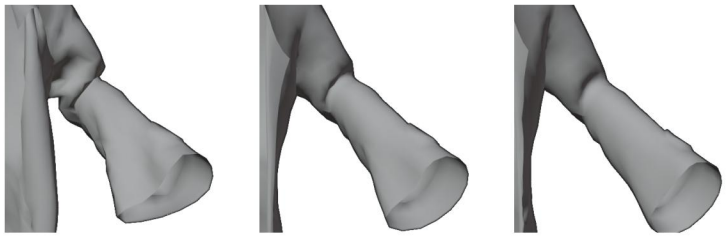
4.2 Qualitative Performance

Of the predicted coat meshes, the two examples with the lowest and highest error are visualized in Figure 5. For examples with low error, the predicted coat matches the ground truth in both overall shape and finer cloth details (wrinkles, folds, etc.). Examples with high error can be explained by differing coattail orientations between the predicted coat and the ground truth. However, because our problem formulation lacks temporal coherence (i.e. is the body stationary or moving?), no orientation is necessarily correct. Although the overall shape may differ, the neural network consistently predicts physically reasonable coats and even preserves finer cloth details.



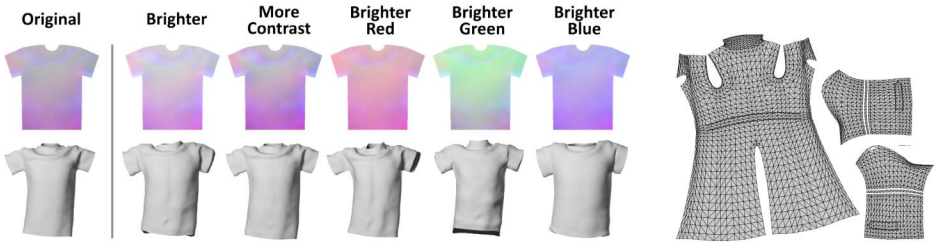
(a) (b) (c) (d)

Figure 5: Lowest squared vertex error = 0.00005: (a) predicted coat vs. (b) ground truth coat. Highest squared vertex error = 0.02295: (c) predicted coat vs. (d) ground truth coat.



(a) (b) (c)

Figure 6: Close-ups of left sleeve: (a) ground truth vs. (b) predicted, neural network 3 vs. (c) predicted, linear regression.



(a) (b)

Figure 7: Pixel-based framework from [1]. By mapping texture-space (u,v,n) offsets to an (r,g,b) image, CNNs can be leveraged for simulated cloth prediction: (a) Image editing operations applied to a T-shirt image (top row) and their corresponding modified cloth shapes (bottom row), taken from [1]. (b) texture map for the coat mesh used in this paper.

One byproduct of incorporating PCA into the network is the automatic smoothing of high variance details. Close-ups of coat meshes are shown in Figure 6. Indeed, procedural cloth simulation may introduce excessive twisting which – although detailed – may be visually undesirable in interactive applications. On the other extreme, our linear regression baseline predictions are too smooth, losing many of the characteristic wrinkles. Depending on the user's preferred cloth smoothness, the model and number of principal components can be tuned accordingly.

5 Conclusion

In conclusion, we have demonstrated that neural networks can indeed predict full-body, loose clothing in real-time. Most notably, compared to high-performance procedural cloth simulation code [3], our data-driven approach achieves over an order-of-magnitude 100,000x speedup while maintaining a squared per-vertex error below 4% (on average). Furthermore, our approach is able to smooth out excessive twisting while still preserving finer cloth details. All in all, these properties make it ideal for applications like e-commerce, gaming, and virtual reality.

However, one limitation is the lack of temporal coherence in our problem formulation. We simulate the coat quasi-statically using a fixed pose when we should be simulating the coat dynamically using an animation – so as to capture physically meaningful coattail orientations. Additionally, state-of-the-art performance in data-driven clothing is currently achieved using convolutional neural networks, as previewed in Figure 7. Because our fully connected neural network runtimes are well within the threshold for real-time use, next steps also include implementing the more complicated pixel-based framework from [1], as well as exploring data augmentation [5] and feature selection [6] along the way.

Contributions and Acknowledgments

As the only CS 229 (Spring 2021) student involved in this project, I (Kangrui Xue) was responsible for all implementation aspects, including dataset generation and exploration, model training and benchmarking, as well as the write-up. Nonetheless, this project would not have been possible without the technical guidance and research mentorship of Jane Wu. Additional thanks go to Prof. Ron Fedkiw for providing me access to the PhysBAM cloth simulation codebase.

The data pre-processing scripts (pose sampling, skinning, cloth simulation) were setup prior to this quarter, but generating and exploring the dataset itself took place entirely in this quarter.

References

[1] Ning Jin, Yilin Zhu, Zhenglin Geng, and Ronald Fedkiw. A pixel-based framework for data-driven clothing. CoRR, abs/1812.01677, 2018.

[2] Zhenglin Geng, Daniel Johnson, and Ronald Fedkiw. Coercing machine learning to output physically accurate results. Journal of Computational Physics, 406:109099, 2020.

[3] Andrew Selle, Jonathan Su, Geoffrey Irving, and Ronald Fedkiw. Robust high-resolution cloth using parallelism, history-based collisions, and accurate friction. IEEE Transactions on Visualization and Computer Graphics, 15(2):339–350, 2009.

[4] Edilson De Aguiar, Leonid Sigal, Adrien Treuille, and Jessica K Hodgins. Stable spaces for real-time clothing. ACM Transactions on Graphics (TOG), 29(4):1–9, 2010.

[5] L. Kavan, D. Gerszewski, A. W. Bargteil, and P.-P. Sloan. Physics-inspired upsampling for cloth simulation in games. In ACM SIGGRAPH 2011 Papers, SIGGRAPH '11, pages 93:1–93:10, New York, NY, USA, 2011. ACM.

[6] Tancik, Matthew, et al. "Fourier features let networks learn high frequency functions in low dimensional domains." arXiv preprint arXiv:2006.10739 (2020).