Coercing Machine Learning to Output Physically Accurate Results

Zhenglin Geng[**], Daniel Johnson[*], Ronald Fedkiw[*]

Stanford University, 353 Jane Stanford Way, Gates Computer Science, Stanford, CA, 94305, United States

Abstract

Many machine/deep learning artificial neural networks are trained to simply be interpolation functions that map input variables to output values interpolated from the training data in a linear/nonlinear fashion. Even when the input/output pairs of the training data are physically accurate (e.g. the results of an experiment or numerical simulation), interpolated quantities can deviate quite far from being physically accurate. Although one could project the output of a network into a physically feasible region, such a postprocess is not captured by the energy function minimized when training the network; thus, the final projected result could incorrectly deviate quite far from the training data. We propose folding any such projection or postprocess directly into the network so that the final result is correctly compared to the training data by the energy function. Although we propose a general approach, we illustrate its efficacy on a specific convolutional neural network that takes in human pose parameters (joint rotations) and outputs a prediction of vertex positions representing a triangulated cloth mesh. While the original network outputs vertex positions with erroneously high stretching and compression energies, the new network trained with our physics prior remedies these issues producing highly improved results.

Keywords: Machine Learning, Physical Simulation

1. Introduction

Many aspects of physical problems are not well understood, and various modeling approximations have helped to make progress; however, many problems remain difficult, whether it be turbulence in fluid flows, surface tension and two-phase flows, coupling between adhesion and cohesion for contact angles, parameters for solid constitutive modeling, the conditions under which materials fracture, etc. In all of these aforementioned examples, although physical experiments facilitate data generation, it is often unclear how to utilize this data in order to obtain models and parameters for use in numerical simulations. Notably, the recent attention given to machine/deep learning stems from the ability to simply annotate data in various ways and subsequently train networks to interpolate from this data without requiring a full understanding or explicit modeling of the underlying system. Of course, ignoring the physics may lead to wildly physically inaccurate results, even though those results might otherwise naively seem like valid interpolations. In addition, such errors are often exacerbated by the use of sparse data representing a physically valid low-dimensional manifold in an otherwise high-dimensional space. Since a number of authors have begun to consider the use of machine/deep learning for problems in traditional computational physics, see e.g. [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11], we are motivated to consider methodologies that constrain the interpolatory results of a network to be contained within a physically admissible region. Quite recently, [12] proposed adding physical constraints to generative adversarial networks (GANs) also considering projection as we do, while stressing the interplay between scientific computing and machine learning; we refer the interested reader to their work for even more motivation for such approaches.

Generally speaking, networks can be used to interpolate a function f from known training pairs/examples (x_T, y_T) with $y_T = f(x_T)$. The network approximation $\hat{f}(w, x)$ depends on parameters w that specify the network so that $y_T \approx \hat{f}(w, x_T)$ for all (x_T, y_T). Suitable parameters w are typically found by minimizing

---

*{zhenglin, dansj, rfedkiw}@stanford.edu

**Corresponding author

an energy function of the form $\sum_T \|y_T - \hat{f}(w, x_T)\|$ with respect to w. The network architecture, i.e. the form of $\hat{f}$, and the subsequent energy minimization are both extremely important for obtaining desirable results. For example, if $\hat{f}$ lacks the expressiveness to capture variability in the training data, i.e. underfitting, there will be large errors in $\hat{f}(w, x_T)$ when compared to y_T. On the other hand, although one could create a network with many degrees of freedom in order to capture $y_T = \hat{f}(w, x_T)$ as accurately as desired, even exactly, $\hat{f}(w, x)$ could oscillate wildly and inaccurately when x is not equal to x_T, i.e. overfitting. See, e.g. [13, 14, 15, 16]. One needs to take great care when designing the network architecture in order to avoid underfitting while still allowing for enough regularization to also avoid overfitting. Likewise, the form of the energy function and nature of the numerical optimization techniques also need careful consideration. Some of the most popular methods include variants of BFGS [17, 18, 19] and a number of methods based on gradient descent [20, 21] (see also [22] and the references therein) or interpreting gradient descent as a numerical approximation to an ordinary differential equation to be solved via various approaches motivated by order of accuracy [23, 24] and adaptive time-stepping [25, 26, 27, 28, 29].

Devising a network architecture with enough representative capability to alleviate underfitting while still being amenable to the regularization required to avoid overfitting, and subsequently applying numerical optimization techniques to an adequately designed energy in order to find reasonable parameters w is a quite difficult and mostly experimental endeavour. Thus, much of the progress made by the community emanates from the laborious creation of data sets that many researchers can consider in order to design network architectures and find suitable parameters w, see e.g. [30]. This is typically driven by a community (rather than an individual or group) effort, and state-of-the-art results are often obtained incrementally by leveraging the works of others. Following this methodology, we choose an existing network and add a postprocess that projects the network's output to be physically admissible/feasible/accurate (see Sections 3 and 5). Importantly, any such postprocess needs to be robust enough to handle the potentially wildly physically inaccurate output of an interpolatory network. Additionally, we incorporate this postprocess into the network itself by modifying the energy to be minimized to use the results of the postprocess instead of the network output (see Sections 4 and 6). This approach requires that any such postprocess be differentiable enough to be embedded into the numerical optimization. We demonstrate the efficacy of our approach on the convolutional neural network from [31] that predicts a cloth shape from joint angles, showing that our procedure not only produces a more physically accurate result but also matches the ground truth as well as the original network.

Section 2 presents the details of the convolutional neural network from [31], and Section 3 describes how we postprocess the output of that network to make it more physically accurate. Those more physically accurate results may deviate quite far from both the training data and the ground truth. So, in Section 4, we incorporate the postprocess into the network itself as a so-called prior, obtaining results that are not only physically accurate but also well match both the training data and the ground truth. Sections 3 and 4 only consider over-stretched material and do not capture buckling phenomena, thus in sections 5 and 6 we duplicate the considerations of Section 3 and 4 for a more complex postprocess that also considers buckling.

Regarding related works, our proposal to embed a second order cone program (Section 4) and a quasistaic physics simulation (Section 6) into a trained network appears to be novel to the best of our knowledge. [32] did add a convex optimization layer, but they only considered a quadratic program with linear constraints, which is a subset of second order cone programming [33]. Furthermore, they implemented a dense solver using a primal-dual method without using a faster solver enabled by second order cone programming. They only demonstrated examples with hundreds of variables whereas we considered around 10,000. Recently, [34] proposed a method that differentiates through a cone program with generality; however, they assume invertibility whereas we show in Appendix A that our case has a null space. In addition, they obtain a non-symmetric system whereas we make our system symmetric in order to use a fast solver. This is quite important because even with a fast solver, the second order cone program approach in Section 3 and 4 is 40 to 50 times slower than the quasistatic approach in Section 5 and 6. Regarding the quasistatic case, the derivatives are more straight-forward, although care must be taken for poor conditioning and inversion; however, this has all been previously addressed some time ago, see [35, 36, 37] which require only minor modifications (as discussed in [38, 39]) for our purposes.
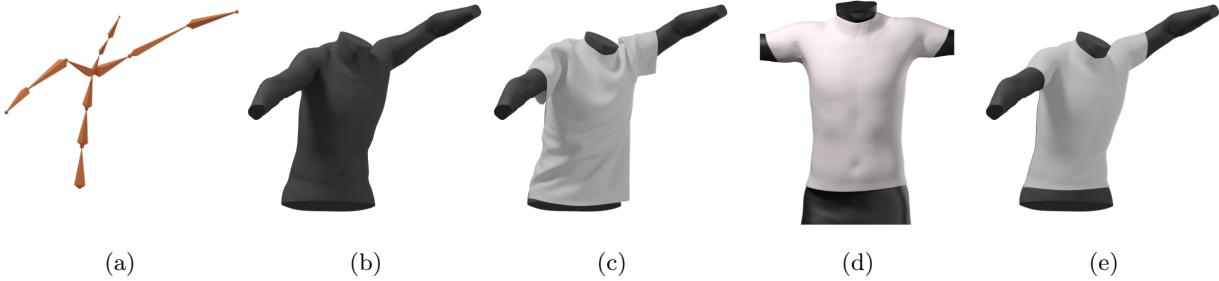
Figure 1: (a) A body pose specified by joint angles. (b) A triangulated surface representing the body exterior for the pose shown in (a). (c) A cloth simulation detecting and processing collisions with the body geometry shown in (b). (d) Shrink-wrapped cloth on the neutral pose. (e) Shrink-wrapped cloth vertices follow their parent triangles as the body surface $\mathcal{B}$ deforms.

2. Data-Driven Network for Cloth

An articulated rigid body skeleton is posed by specifying a set of joint angles θ, where it is assumed that the root node (e.g. the pelvis) is fixed. See Figure 1a. Then, a procedural skinning algorithm is used to form a triangulated surface $\mathcal{B}$ representing the exterior surface of the body in a manner consistent with the pose θ. See Figure 1b. There are a wide variety of approaches for skinning a body triangulated surface $\mathcal{B}$ from joint angles θ, ranging from error-prone estimates to those with more bio-mechanical accuracy, see e.g. [40, 35, 41, 42, 43, 44, 45, 46]. The body triangulated surface $\mathcal{B}$ is converted to a level set representation, see e.g. [47], which is used to detect and process collisions with the triangulated surface $\mathcal{C}$ representing the cloth, see e.g. [48]. See Figure 1c. Although any method could be used to simulate the cloth triangulated surface $\mathcal{C}$ with elasticity and bending, the authors of [31] used methods derived from [49, 50]. As is typical, they carried out a number of numerical simulations to create a large data set of corresponding pairs $(\theta, \mathcal{C})$; a representative subset of those pairs, $(\theta_T, \mathcal{C}_T)$, was chosen as the training set for the network, and the remaining pairs were used to ascertain the predictive capabilities of any such network. Notably, after finding suitable parameters w for the network, the entire set of data (including the training pairs) may be discarded.

Given θ, which may or may not be in the training set, the network needs to predict a cloth triangulated surface $\mathcal{C}$; however, since we keep the topology fixed, it only needs to predict vertex positions r. Designing an appropriate network can be quite difficult because of the non-linear joint rotations. Thus, [31] proposed a preprocess that utilizes a procedural skinning algorithm to obtain vertex positions that include a significant portion of the non-linear rotations. They accomplished this by shrink-wrapping the cloth vertices to the body surface $\mathcal{B}$ in the neutral pose (see Figure 1d), and subsequently barycentrically embedding those cloth vertices to follow their parent triangles $b \in \mathcal{B}$ as the body mesh deforms (see Figure 1e). We write the simulated cloth vertex positions as $r = \tilde{r} + d$ where the $\tilde{r}$ one-to-one correspond to r but are each embedded to follow a body triangle b, and d is the remaining displacement/offset from $\tilde{r}$ to r. Thus for any θ, $\tilde{r}(\theta)$ is well-determined by the chosen skinning algorithm, and the network only needs to predict $d(\theta)$ from training
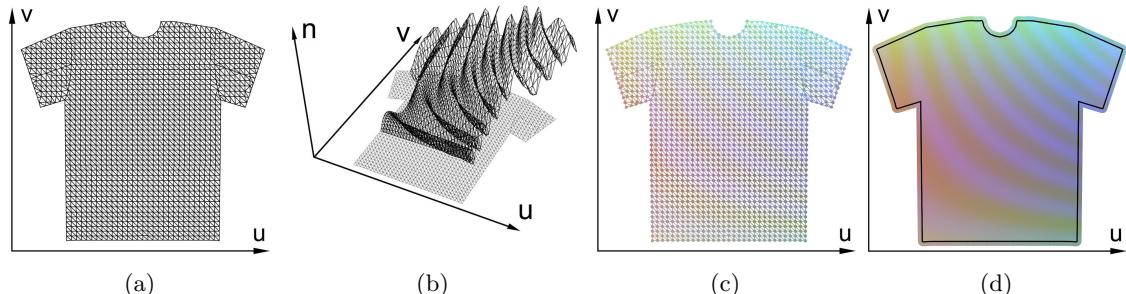

Figure 2: (a) A subset of the cloth mesh depicted in a two dimensional pattern space. (b) Depiction of the three dimensional displacement $d = r - \tilde{r}$ in the pattern space. (c) Displacements from (b) converted to RGB colors for each vertex. (d) Rasterization of the vertex colors from (c) onto a background grid of pixels to create an image.

3

pairs (θ_T, d_T) which do not possess as much non-linear variation as (θ_T, r_T) do. Moreover, since $\tilde{r}(\theta)$ is independent of the network weights w, it does not need to be differentiated during numerical optimization. In summary, this can be seen as a preprocess that decomposes a function f into $f_1 + f_2$ where f_1 has less variation and is thus easier to approximate while f_2 has more variation but is a known function of the parameters.

Although one could train a network to interpolate from the pairs (θ_T, d_T), [31] noted that the nature of the problem lends itself to an image based convolutional neural network (CNN) approach, and accomplished this by laying out subsets of cloth vertices in two-dimensional pattern spaces (motivated by actual garment construction from textiles). See Figure 2a. The three dimensional displacement d can be transformed into this pattern space and displayed as a displacement of the form $(\Delta u, \Delta v, \Delta n)$ for each vertex. If the cloth were actually skin-tight, d would be identically zero; but otherwise, d can be depicted as a new triangulated surface as shown Figure 2b. Notably $(\Delta u, \Delta v, \Delta n)$ can be converted to RGB colors stored at each vertex (see Figure 2c), and those vertex colors can be rasterized to an underlying image of pixels to obtain a cloth image I (see Figure 2d). Then, a convolutional neural network can be trained to interpolate from training pairs (θ_T, I_T). Afterwards, given a pose θ, the network predicts an image I, and the RGB colors of each cloth vertex are interpolated from this image, converted to a displacement d, and added to $\tilde{r}$ to obtain $\hat{r}$.

Given training pairs (θ_T, I_T), [31] trained their network using Adam [28] to find network weights w that minimized an energy of the form $\|I(\theta_T, w) - I_T\|$. The actual goal was to make the vertex positions match, i.e. to minimize $\|\hat{r}(\theta_T, w) - r_T\| = \|\tilde{r}(\theta_T) + d(\theta_T, w) - (\tilde{r}_T + d_T)\| = \|d(\theta_T, w) - d_T\|$, and so terms of this form may be introduced as well. Moreover, to encourage visual similarity, [31] also used terms that penalized differences between actual and predicted normal vectors.

3. Inextensibility Postprocess

Even though the network designed in [31] predicts vertex positions as a function of joint angles quite well on average, there are a number of over-stretched/compressed elements. Thus, the cloth triangulated surface $\mathcal{C}$ is generally of poor physical quality and would likely behave poorly when subsequently simulated, assuming the numerical simulation would work at all. As is discussed in [51], compression can serve as a proxy for bending/buckling that is under-resolved by the mesh; however, triangle edges and faces should not stretch beyond some elastic threshold regardless of the discretization. This led [51] to use different stiffnesses for compression versus extension. Furthermore, they showed that strengthening per-element resistance to in-plane elastic deformation causes spurious locking (see e.g. [52]) that incorrectly removes bending degrees of freedom. To remedy this, they proposed a two-phase approach to elastic stretching where a reasonable strength elasticity model was used for small deformations and a constraint-based approach was used to limit larger deformations (e.g. when warp and weft threads align, see Figure 3).

Following the spirit of [51], we replace their numerical simulation with the network prediction from [31], while still using the constraints proposed in [51] to project the network output to have edge length stretching limited by a specified tolerance. Let $\mathcal{E}$ be the set of all edges e that connect two vertices in the
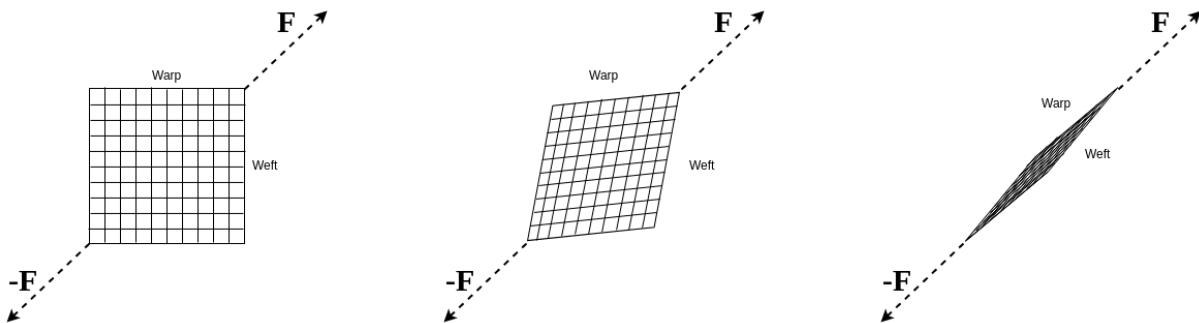

Figure 3: A square piece of cloth consists of generally perpendicular warp and weft threads. (a) Forces are applied to diagonal corners of a cloth patch. (b) Initially, warp and weft threads slide past each other, subject to inter-thread frictional forces. (c) As the threads align, the resistance to deformation increases dramatically since it is difficult to stretch individual threads along their axial direction.
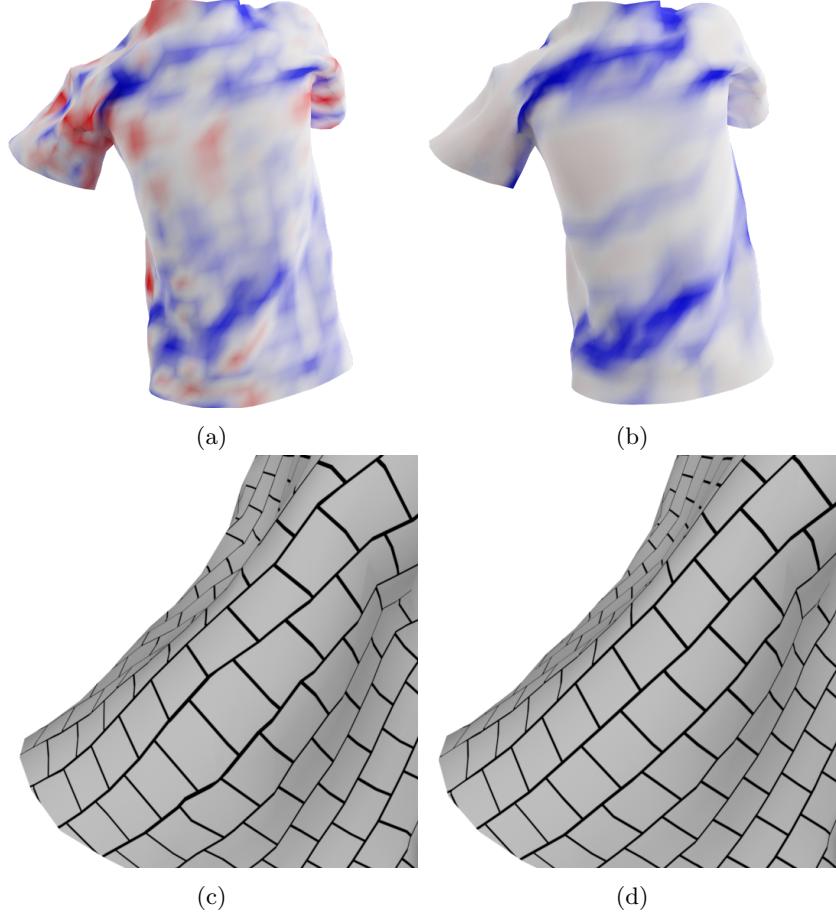
Figure 4: (a) Network output from [31] depicting over-stretching in red and over-compression in blue (pure red indicates 1.25 times stretching, pure blue indicates 0.75 times compression, and white indicates no distortion). (b) Results obtained after applying the inextensibility postprocess to (a). (c) Zoomed-in textured view of (a). (d) Zoomed-in textured view of (b).

cloth triangulated surface $\mathcal{C}$. Then, given a set of vertex positions $\hat{r}$ predicted by the network, they are projected to a new set of vertex positions r such that each edge $\vec{l}_e(r)$ has its length $l_e(r) \leq l_e^{\max}$ where $l_e^{\max} = (1 + \varepsilon)l_e^{\text{rest}}$ with l_e^{rest} the rest length and ε a small number. Since the solution to this problem is not unique, we make the further assumption that r deviate from $\hat{r}$ as little as possible, which is justified since the network was trained to match $\hat{r}$ to the numerical simulations. In particular, we use the sum of the square of the distances between each r and corresponding $\hat{r}$, which is a spring potential energy.

Each vertex r_i is connected to its corresponding vertex $\hat{r}_i$ via a zero-length spring with elastic force $k_i^z(\hat{r}_i - r_i)$ and potential energy $\frac{1}{2}k_i^z\|\hat{r}_i - r_i\|_2^2$ where k_i^z is the spring constant. Since the triangulated cloth surface $\mathcal{C}$ is spatially adaptive with a varying mass per vertex, we set k_i^z proportional to the vertex mass so that accelerations (force over mass) are more uniform. Any overall global scaling of these forces is unimportant since it does not change the configuration where the minimum is achieved. Thus, the problem is formulated as follows:

$$\min_r \quad \sum_i \frac{1}{2}k_i^z\|\hat{r}_i - r_i\|_2^2$$
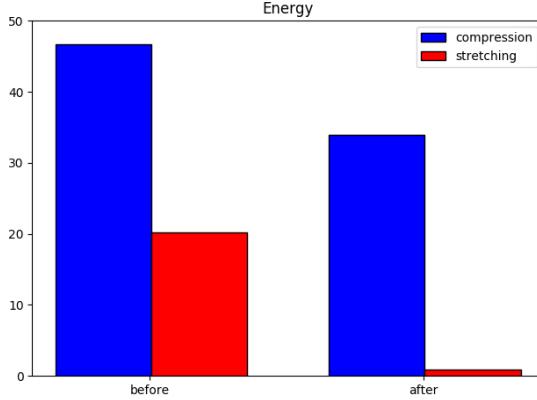$$\text{subject to} \quad l_e(r) \leq l_e^{\max}, \quad \forall e \in \mathcal{E}. \tag{1}$$

Figure 5: Comparison of the energies before/after the inextensibility postprocess (averaged over all the examples in the test set).

This is a convex minimization problem with a unique solution, and has KKT conditions [33]:

$$\lambda_e \geq 0, \quad l_e(r) - l_e^{\max} \leq 0, \quad \lambda_e(l_e(r) - l_e^{\max}) = 0, \quad \forall e \in \mathcal{E}$$
$$k_i^z(\hat{r}_i - r_i) + \sum_{e \in \mathcal{E}_i} \lambda_e \hat{l}_e^{\,i}(r) = 0, \quad \forall i \tag{2}$$

where λ_e are Lagrange multipliers which may only be non-zero when $l_e(r) = l_e^{\max}$. $\mathcal{E}_i$ consists of all the edges from $\mathcal{E}$ that include vertex i, reoriented (if necessary) so that vertex i appears first, and $\hat{l}_e^{\,i}(r)$ is the unit vector that points from r_i to the location of the other vertex on the edge $e \in \mathcal{E}_i$. When non-zero, the equal and opposite per-edge constraint forces act as strong edge springs that balance the zero length springs into quasi-static equilibrium while enforcing the constraints.

3.1. Examples

Equation 1 can be recast into a second-order cone program (see Section 4) and solved efficiently with the method from [53]. Figure 4a shows a typical result output from the network proposed in [31], depicting over-stretched edges in red and over-compressed edges in blue. Figure 4b shows the result obtained by solving Equation 1 as a postprocess to the network result shown in Figure 4a. Figure 4c shows a zoomed-in textured view of the right sleeve in Figure 4a. Figure 4d shows a zoomed-in textured view of the same area of Figure 4b, highlighting how the postprocess alleviates in-plane distortions. Figure 5 shows a comparison of the spring energies before and after the inextensibility postprocess (averaged over all examples in the test set) illustrating a drastic removal of over-stretching.

4. Inextensibility Prior

Assuming r^* is the unique solution to Equation 1, embedding the postprocess of Section 3 into the neural network is accomplished by changing the energy to be minimized replacing terms of the form $\|\hat{r} - r_T\|_2^2$ with $\|r^* - r_T\|_2^2$. Then, the minimization process requires the derivatives of this new energy with respect to the network weights w. Note that $\frac{\partial \hat{r}}{\partial w}$ is readily accessible since it was used to train the network in [31]. In addition, the derivatives of the energy with respect to r^* are typically straight-forward. Thus, we only need discuss $\frac{\partial r^*}{\partial \hat{r}}$; however, $\frac{\partial r^*}{\partial \hat{r}}$ is not altogether clear from Equations 1 and 2. In order to better elucidate $\frac{\partial r^*}{\partial \hat{r}}$, we consider the solution process proposed in [53].

Equation 1 is rewritten in conic form as follows. Let Q^m be the space of second order cones with dimension m, i.e. $[q_0, q_1^T]^T \in Q^m$ if and only if $\|q_1\|_2 \leq q_0$ where $q_0 \in \mathbb{R}$, $q_1 \in \mathbb{R}^{m-1}$. Then defining $s_0 = [t, \sqrt{k_i^z}(r_i - \hat{r}_i)^T, \ldots]^T$ allows the objective function in Equation 1 to be recast as minimizing the slack variable t while maintaining $s_0 \in Q^{3n+1}$. Similarly, defining $s_e = [l_e^{\max}, \vec{l}_e(r)^T]^T$ allows the constraints to be represented by simply stating that $s_e \in Q^4$ for all $e \in \mathcal{E}$. Alternatively, one can specify $s_e = [\alpha_e, \vec{l}_e(r)^T]^T$

where the α_e are additional variables constrained via $\alpha_e = l_e^{\max}$. Concatenating all the α_e into a single vector α, we obtain

$$
\begin{aligned}
\min_{r,t,\alpha} \quad & t \\
\text{subject to} \quad & s_0 \in Q^{3n+1} \\
& s_e \in Q^4, \quad \forall e \in \mathcal{E} \\
& \alpha_e = l_e^{\max}, \quad \forall e \in \mathcal{E}.
\end{aligned}
\tag{3}
$$

To simplify, let $x = [r^T, t, \alpha^T]^T$. Similarly, concatenate s_0 and all the s_e into a single vector s, defining $\mathcal{K}$ as the space of all s that have sub-vector $s_0 \in Q^{3n+1}$ and all sub-vectors $s_e \in Q^4$. With this notation, Equation 3 can be written more formally as a second order cone program (SOCP):

$$
\begin{aligned}
\min_{x,s} \quad & c^T x \\
\text{subject to} \quad & Ax = b \\
& s \in \mathcal{K} \\
& s = h - Gx.
\end{aligned}
\tag{4}
$$

Here, c merely selects t from x, and thus does not depend on $\hat{r}$. A selects all α_e from x and sets them equal to the corresponding $l_e^{\max}$ in b, and so neither A nor b depends on $\hat{r}$. The constraints $s_0 \in Q^{3n+1}$ and $s_e \in Q^4$ for $\forall e \in \mathcal{E}$ are folded into $s \in \mathcal{K}$. Finally, $s = h - Gx$ ties s and x together, defining s_0 and all s_e. Although s_e is defined by G alone, s_0 is defined by both G and h with $-\sqrt{k_i^z}\hat{r}_i$ terms in h making h the only term in Equation 4 that depends on $\hat{r}$. Thus, we may write $\frac{\partial r^*}{\partial \hat{r}}$ as $\frac{\partial r^*}{\partial h}\frac{\partial h}{\partial \hat{r}}$, where $\frac{\partial h}{\partial \hat{r}}$ has non-zero terms of the form $-\sqrt{k_z^i}$. Equation 4 is the primal form of the second order cone program, and the dual form is

$$
\begin{aligned}
\max_{y,z} \quad & -b^T y - h^T z \\
\text{subject to} \quad & G^T z + A^T y + c = 0 \\
& z \in \mathcal{K}.
\end{aligned}
\tag{5}
$$

As discussed in [53], the primal and dual problems are optimal at the same point. All the constraints from Equations 4 and 5 can be collected to write

$$
\begin{bmatrix} 0 & A^T & G^T & 0 \\ A & 0 & 0 & 0 \\ G & 0 & 0 & I \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ s \end{bmatrix} = \begin{bmatrix} -c \\ b \\ h \end{bmatrix}, \quad s, z \in \mathcal{K}.
\tag{6}
$$

Next, given $q = [q_0, q_1^T]^T$ and $\hat{q} = [\hat{q}_0, \hat{q}_1^T]^T$, the conic product is defined as $q \circ \hat{q} = \begin{bmatrix} q^T \hat{q} \\ q_0 \hat{q}_1 + \hat{q}_0 q_1 \end{bmatrix}$. For the concatenated cone space $\mathcal{K}$, the conic product $s \circ z$ is defined via separate conic products between each pair of component cones. As discussed in [53], the joint solution to the primal and dual problems satisfies the constraints in Equation 6 along with $s \circ z = 0$.

Taking differentials of the linear system in Equation 6 as well as $s \circ z = 0$ yields

$$
\begin{bmatrix} 0 & A^T & G^T & 0 \\ A & 0 & 0 & 0 \\ G & 0 & 0 & I \\ 0 & 0 & S & Z \end{bmatrix} \begin{bmatrix} dx \\ dy \\ dz \\ ds \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ dh \\ 0 \end{bmatrix}
\tag{7}
$$

where S and Z are the matrices defined via $Sdz = s \circ dz$ and $Zds = z \circ ds$ ($= ds \circ z$) respectively, so that the last equation is $Sdz + Zds = 0$.

The interior point solution method of [53] provides a point which is nearly optimal, but not exactly so due to numerical errors, the stopping conditions/tolerance on iterations, and the need to project the

7

solution/iterates into the domain interior. Thus, their solution will approximately satisfy the optimality conditions, and the differentials evaluated at their solution will approximately satisfy Equation 7. Although Equation 7 is generally not full rank, we discuss its null space in Appendix A so that one may still find suitable values for the differentials in a minimal norm sense. Notably, the interior point method of [53] returns a solution where the coefficient matrix in Equation 7 is full rank.

To symmetrize Equation 7, we column scale defining a new variable $d\psi$ such that $ds = Sd\psi$ to obtain

$$\begin{bmatrix} 0 & A^T & G^T & 0 \\ A & 0 & 0 & 0 \\ G & 0 & 0 & S \\ 0 & 0 & S & ZS \end{bmatrix} \begin{bmatrix} dx \\ dy \\ dz \\ d\psi \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ dh \\ 0 \end{bmatrix}. \tag{8}$$

Both Z, S are symmetric, and their product ZS *should* be symmetric as well. Since Z, S and ZS are all block diagonal, each block can be considered independently. Let q and $\hat{q}$ be sub-cones, and define M_q so that $M_q \hat{q} = q \circ \hat{q}$, i.e.

$$M_q = \begin{bmatrix} q_0 & q_1^T \\ q_1 & q_0 I \end{bmatrix}. \tag{9}$$

Then, a diagonal block of ZS has the form:

$$M_q M_{\hat{q}} = \begin{bmatrix} q_0 \hat{q}_0 + q_1^T \hat{q}_1 & q_0 \hat{q}_1^T + \hat{q}_0 q_1^T \\ q_0 \hat{q}_1 + \hat{q}_0 q_1 & q_1 \hat{q}_1^T + q_0 \hat{q}_0 I \end{bmatrix} \tag{10}$$

where the upper left hand corner is a scalar, and the off diagonal terms $q_0 \hat{q}_1 + \hat{q}_0 q_1$ and its transpose should both be identically 0 under the optimality constraints. The lower right hand term has a symmetric component $q_0 \hat{q}_0 I$ and an outer product component $q_1 \hat{q}_1^T$. If q_0 or $\hat{q}_0$ is 0, then all of q_1 or $\hat{q}_1$ respectively is also zero, making the entire term on the lower right hand block of $M_q M_{\hat{q}}$ equal to zero, so the block is trivially symmetric. In practice, the interior solver [53] returns a point on the interior of $\mathcal{K}$, so both q_0 and $\hat{q}_0$ are non-zero. Using the optimality condition $q_0 \hat{q}_1 + \hat{q}_0 q_1 = 0$, one can rewrite $q_1 \hat{q}_1^T$ as either $-\frac{\hat{q}_0}{q_0} q_1 q_1^T$ or $-\frac{q_0}{\hat{q}_0} \hat{q}_1 \hat{q}_1^T$ depending on whether q_0 or $\hat{q}_0$ is larger (for robustness). In summary, we symmetrize each diagonal block in this manner, projecting away numerical errors from the interior solver of [53] that lead to small asymmetries.

Denoting the symmetrized version of the coefficient matrix in Equation 8 as K and the unknown vector as $d\xi$, Equation 8 should be solved separately for each variable in h. This is accomplished by dividing both sides by dh_j to obtain unknown $\frac{\partial \xi}{\partial h_j}$ with basis vector $[0, 0, e_j^T, 0]^T$ on the right hand side. This can be conveniently notationally stacked into the expression $K \frac{\partial \xi}{\partial h} = I_h$ where $\frac{\partial \xi}{\partial h}$ consists of columns of the form $\frac{\partial \xi}{\partial h_j}$ and I_h is a block column matrix of zeros except for an identity matrix corresponding to the location of h. Since r^* is a sub-component of x^*, we may write $\frac{\partial r^*}{\partial h} = I_r \frac{\partial \xi}{\partial h}$ where I_r is a block row matrix of zeros with an identity matrix corresponding to the location of r^*. In summary,

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial r^*} \frac{\partial r^*}{\partial \xi} \frac{\partial \xi}{\partial h} \frac{\partial h}{\partial \hat{r}} \frac{\partial \hat{r}}{\partial w} = \frac{\partial E}{\partial r^*} I_r K^{-1} I_h \frac{\partial h}{\partial \hat{r}} \frac{\partial \hat{r}}{\partial w} = \left(\frac{\partial E}{\partial r^*} I_r K^{-1} \right) I_h \frac{\partial h}{\partial \hat{r}} \frac{\partial \hat{r}}{\partial w} \tag{11}$$

where $\frac{\partial E}{\partial r^*}$ is straight-forward to calculate based on the form of the energy function, $\frac{\partial h}{\partial \hat{r}}$ is based on the definition of h in Equation 4, and $\frac{\partial \hat{r}}{\partial w}$ is available from the network in [31]. For efficiency, note that $\frac{\partial E}{\partial r^*}$ and $\frac{\partial E}{\partial r^*} I_r$ are row vectors, and so we may compute the row vector $\eta = \frac{\partial E}{\partial r^*} I_r K^{-1}$ by solving $K \eta^T = I_r^T \left(\frac{\partial E}{\partial r^*} \right)^T$ once for each training example; in fact, we use sparse LDL similar to [53].

4.1. Examples

We use the cloth data set from [31] to obtain both the training examples and ground truth consistent with our approach. The cloth consists of 2969 vertices with 8787 inequality constraints. We minimize an energy of the form $\|r^* - r_T\|$, whereas [31] minimized an energy of the form $\|\hat{I} - I_T\|$; thus, we retrain the same network from [31] using $\|\hat{r} - r_T\|$. The derivatives of the total energy with respect to the weights w can be computed separately for the terms corresponding to each training example using Equation 11. For each training example, solving Equation 1 takes 3-4 seconds and computing $\frac{\partial E}{\partial \hat{r}}$ via η takes about 0.5 seconds.

8

Method	Training Set		Validation Set		Test Set	
	SqrtMSE	MaxDist	SqrtMSE	MaxDist	SqrtMSE	MaxDist
[31]	2.5370	10.554	7.6060	29.799	7.6840	29.637
Postprocess Only	2.0781	10.042	7.2026	28.880	7.4326	29.012
Trained Postprocess	2.0767	9.9807	7.1903	29.181	7.1603	29.006

Table 1: Errors on a data set with 1000 training examples in the energy function. Numbers are in millimeters.

Method	Training Set		Validation Set		Test Set	
	SqrtMSE	MaxDist	SqrtMSE	MaxDist	SqrtMSE	MaxDist
[31]	4.1824	17.351	5.9018	25.506	5.9405	25.597
Postprocess Only	3.9257	16.853	5.6153	25.502	5.6582	25.215
Trained Postprocess	3.8096	16.790	5.5875	25.030	5.6511	25.267

Table 2: Errors on a data set with 8000 training examples in the energy function. Numbers are in millimeters.

We train our network on a data set with 1000 training examples and report the errors in Table 1. We use two error metrics: square root of mean square error (SqrtMSE) and maximum vertex distance (MaxDist). We compare errors for three approaches: results obtained using the network from [31] (but with $\|\hat{r} - r_T\|$), applying the inextensibility postprocess only to the results of [31], and training with the inextensibility postprocess in the network. The training set is the set of examples used in the energy function. At training time, we periodically evaluate the network on a validation set and save the network weights w that have the smallest SqrtMSE. The test set of data is not seen during training and is used as a proxy for measuring generalization error. As can be seen in the table, the inextensibility postprocess reduces the errors in all cases, ranging from small improvements up to around 25%; however, including the postprocess in training generally yields only minor improvements. The results obtained by incorporating the postprocess in training are similar enough to those obtained using the postprocess only that the comments in Section 3.1 and Figures 4 and 5 are representative of both. Table 2 shows results one would expect when increasing the number of training examples from 1000 to 8000: it is harder to reduce the energy, so the training set errors increase; however, having more training examples reduces the errors on the validation set and test set.

5. Buckling and Inextensibility Postprocess

Although the inextensibility postprocess discussed in Sections 3 and 4 adequately prevents overstretching, it still allows for compression as motivated by [51]; however, there are often times when it is desirable to limit element compression as well. Thus, we introduce a quasistatic simulation postprocess to better capture buckling phenomena in addition to inextensibility. Although there are a variety of material models one might employ, the numerical method should be robust enough to handle the poorly posed initial conditions output by the network including poorly conditioned and inverted elements/configurations.
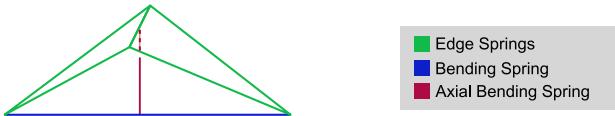

Figure 6: A pair of triangles in the cloth mesh (green). Besides using springs on the edges of the original cloth mesh, bending springs (blue) are created for fictitious edges connecting the non-shared vertices for every pair of triangles. In order to prevent ill-conditioned oscillations near a flat rest state, a zero-length spring (red) connects the fictitious bending edge with the edge shared by the pair of triangles. See [54].

Figure 6 illustrates the material model we utilize. For notational clarity, we rename the edges of the triangle mesh from $\mathcal{E}$ to $\mathcal{E}_L$ going forward, and denote the fictional bending edges as $\mathcal{E}_B$ and zero-length axial bending springs as $\mathcal{E}_A$. Then the total spring energy is a combination of $E_{edge}(r) = \sum_{\mathcal{E}_L} \frac{1}{2} k_e^L \left(l_e^L(r) - \bar{l}_e^L \right)^2$, $E_{bending}(r) = \sum_{\mathcal{E}_B} \frac{1}{2} k_e^B \left(l_e^B(r) - \bar{l}_e^B \right)^2$, and $E_{axial}(r) = \sum_{\mathcal{E}_A} \frac{1}{2} k_e^A \left((1-u)r_e^a + ur_e^b - (1-v)r_e^c - vr_e^d \right)^2$. This total energy is approximated by a truncated Taylor expansion $E(r + dr) \approx E(r) + J^T(r)dr + \frac{1}{2} dr^T H(r)dr$,

9

(a) output from [31]   (b) $\mu = 10^3$ (c) $\mu = 10^2$ (d) $\mu = 10^1$ (e) $\mu = 10^0$

(f) ground truth   (g) $\mu = 10^{-1}$ (h) $\mu = 10^{-2}$ (i) $\mu = 10^{-3}$ (j) $\mu = 10^{-4}$
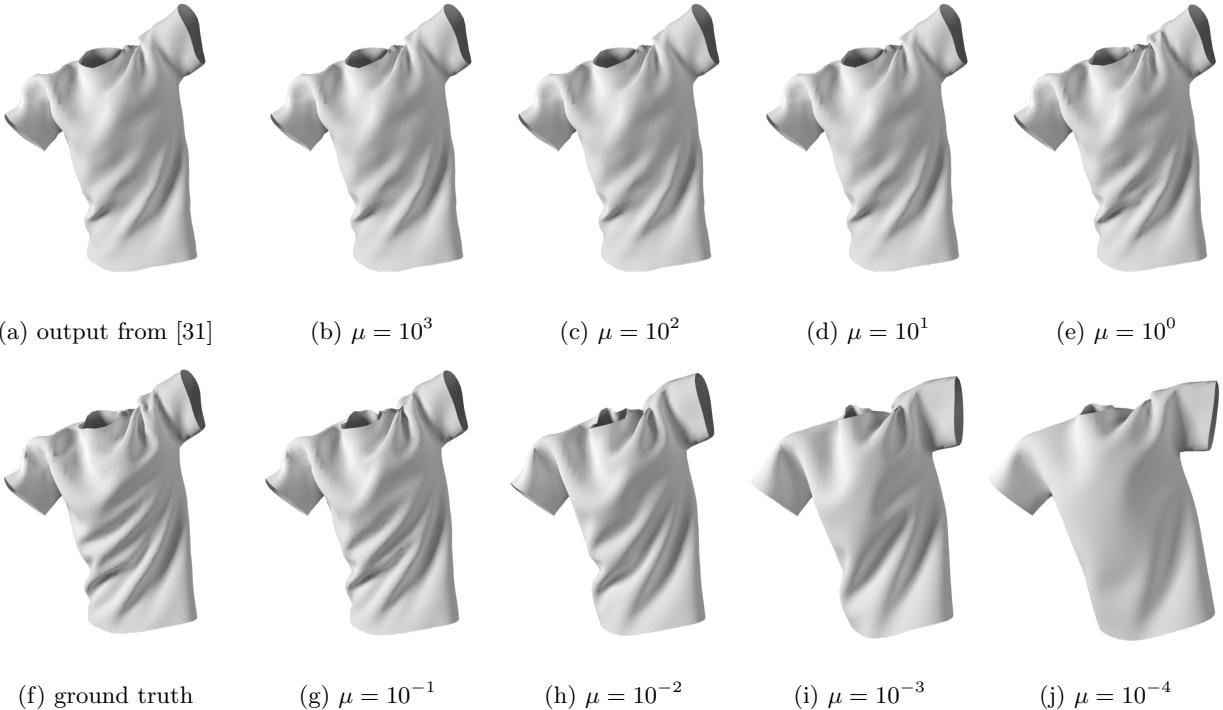
Figure 7: Comparison of the ground truth (f), the network result (a), and a number of quasistatic postprocessed solutions. Note how the stiffest zero length springs in (b) produce a result close to [31], and how the weaker zero-length springs allow the cloth to drift too far from the ground truth. Also note how well the quasistatic solution shown in figure (g) matches the ground truth shown in (f).

which is minimized by $H(r)dr = -J(r)$. At each iteration, following a Newton-Raphson approach, we invert $H(r)$ to find a direction dr for a line search aiming to minimize the energy. Since zero-length springs only stretch and do not compress, H_{axial} is symmetric positive semi-definite (SPSD); however, both H_{edge} and $H_{bending}$ suffer from indefiniteness that we remedy via [55] so that a fast conjugate gradient solver may be used. As discussed in [55], the fix for definiteness does not adversely impact the final solution to the minimization. Note that Figure 6 depicts a tetrahedron which may invert towards a spurious inside out steady state, see e.g. [36]. Although this too can be addressed in our formulation, we omit the discussion since this is not an issue for the flat rest states considered herein.

Similar to Section 3, we add additional zero-length springs connecting r to $\hat{r}$, with $E_Z = \sum_i \frac{1}{2} k_i^z \|\hat{r}_i - r_i\|_2^2$. Then E_Z is added to the total energy minimized, noting that the hessian of E_Z is SPSD. Unlike in Section 3 where only the relative not the overall scaling of k_i^z mattered, here the zero-length springs compete with the cloth material model to achieve an equilibrium; thus, stronger k_i^z adhere r to $\hat{r}$ and weaker k_i^z allow the cloth to equilibrate independent of the network prediction. Figure 7 shows the postprocessed results for various global scaling factors μ. Surprisingly, an adequate value of k_i^z matches the ground truth quite well, even though the entirety of the effects from collisions and gravity are modeled only by the network. This is a significant optimization as compared to a pure numerical simulation because one does not need costly geometric representations of the collision body, e.g. level sets [47], or algorithmic acceleration structures to make processing collisions more feasible.

5.1. Examples

Figure 8 shows how the quasistatic postprocess greatly improves the results removing non-physical in-plane distortions as compared to both the network results as well as the results obtained using the inextensibility postprocess. In addition, Figure 9 shows that similar improvements are obtained when considering the energies.
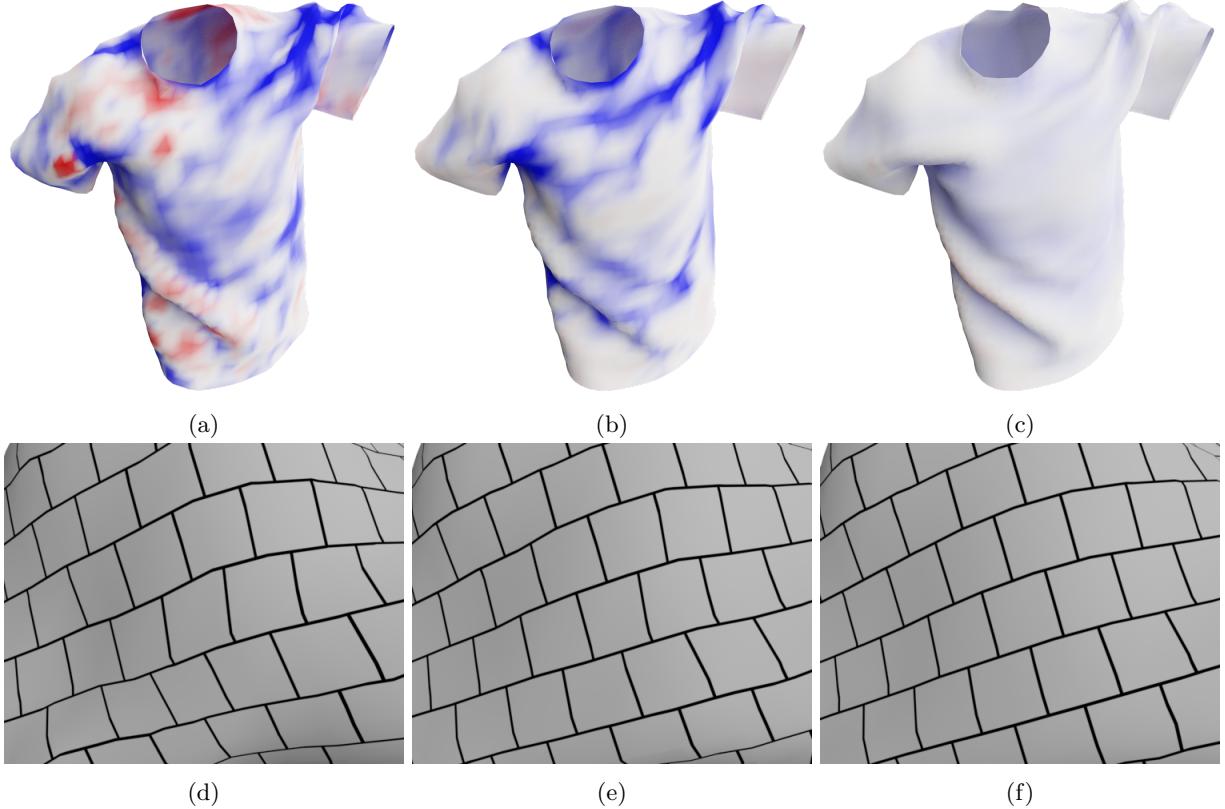
10

Figure 8: (a) Network output depicting over-stretching in red and over-compression in blue (pure red indicates 1.25 times stretching, pure blue indicates 0.75 times compression, and white indicates no distortion). (b) Results obtained after applying the inextensibility postprocess from Section 3 to (a). (c) Results obtained after applying the quasistatic postprocess to (a). Subfigures (d), (e), and (f) show zoomed-in textured views of (a), (b), and (c) respectively, illustrating the removal of non-physical in-plane distortion.
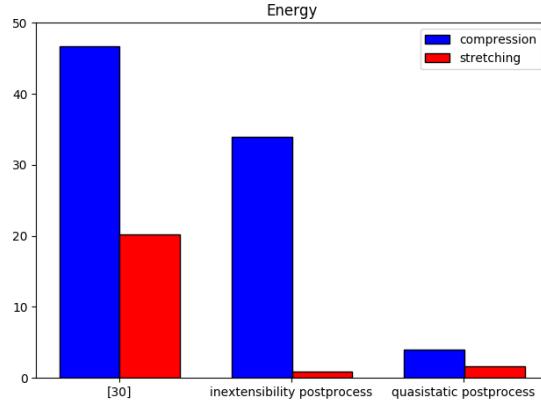

Figure 9: The quasistatic postprocess removes spuriously high stretching energy as well as the inextensibility postprocess did, but is also able to remove spurious compression energy, unlike the inextensibility postprocess. The energy is averaged over all the examples in the test set.

6. Buckling and Inextensibility Prior

Similar to Section 4 we minimize an energy containing terms of the form $\|r^* - r_T\|_2^2$. Again, the minimization process requires the derivatives of this energy with respect to the network weights w, and since the derivatives with respect to r^* are readily computed and $\frac{\partial \hat{r}}{\partial w}$ is accessible from [31], we only need consider $\frac{\partial r^*}{\partial \hat{r}}$.

11

Conceptually speaking, the goal is to find a set of network parameters w that determine cloth geometry via the network in [31] such that the zero-length springs attached to that network driven cloth geometry drive the quasistatic simulation mesh to well match the training data. [39] addressed a similar problem where they solved for a set of parameters that determined blendshape muscle geometry such that attached zero-length springs drove their quasistatic simulation mesh to well match a ground truth target. They utilized the approach in [37] to evaluate search directions for the optimization (noting that [37] solved for muscle activations directly whereas [39] obtained activations indirectly using zero-length springs attached to kinematically driven geometry as proposed in [38]).

Although the material model forces only depend on r^* unlike the zero length springs that depend on both r^* and $\hat{r}$, r^* depends on $\hat{r}$ and so the dependencies may be written as $f_Z(r^*(\hat{r}), \hat{r})$ and $f_M(r^*(\hat{r}))$. The total derivative of the forces $f = f_Z + f_M$ with respect to the network output $\hat{r}$ is

$$\frac{\partial f_Z}{\partial r^*}\frac{\partial r^*}{\partial \hat{r}} + \frac{\partial f_Z}{\partial \hat{r}} + \frac{\partial f_M}{\partial r^*}\frac{\partial r^*}{\partial \hat{r}} = \frac{\partial f_Z + f_M}{\partial r^*}\frac{\partial r^*}{\partial \hat{r}} + \frac{\partial f_Z}{\partial \hat{r}} = \frac{\partial f}{\partial r^*}\frac{\partial r^*}{\partial \hat{r}} + \frac{\partial f_Z}{\partial \hat{r}} \tag{12}$$

implying that the quasistatic net force equal to zero solution may be obtained by solving

$$\frac{\partial f}{\partial r^*}\frac{\partial r^*}{\partial \hat{r}} = -\frac{\partial f_Z}{\partial \hat{r}} \tag{13}$$

to find $\frac{\partial r^*}{\partial \hat{r}}$. In addition, noting that force is the negative derivative of potential energy with respect to position, $\frac{\partial f}{\partial r^*}$ is the negative Hessian from Section 5. Again using associativity for efficient computation:

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial r^*}\frac{\partial r^*}{\partial \hat{r}}\frac{\partial \hat{r}}{\partial w} = \frac{\partial E}{\partial r^*}H^{-1}\frac{\partial f_Z}{\partial \hat{r}}\frac{\partial \hat{r}}{\partial w} = \left(\frac{\partial E}{\partial r^*}H^{-1}\right)\frac{\partial f_Z}{\partial \hat{r}}\frac{\partial \hat{r}}{\partial w} \tag{14}$$

where $\frac{\partial E}{\partial r^*}$ is again a row vector. Define the row vector $\eta = \frac{\partial E}{\partial r^*}H^{-1}$; then, we may compute η by solving $H\eta^T = \frac{\partial E}{\partial r^*}^T$ once for each training example. This is the same linear system solved for quasistatics in Section 5. Finally, note that $\frac{\partial f_Z}{\partial \hat{r}}$ is a sparse matrix and can be computed efficiently.

6.1. Examples

Our cloth consists of 2969 vertices, each with a zero-length spring. The material model has 8787 edge springs, 8661 bending springs, and 8661 axial bending springs. The derivatives of the total energy with respect to the weights w can be computed separately for the terms corresponding to each training example using Equation 14. For each training example, solving the quasistatic problem for r^* takes about 0.07 seconds, and computing $\frac{\partial E}{\partial \hat{r}}$ via η takes about 0.01 seconds.

Similar to Section 4.1, we train our network on a data set with 1000 training examples and report the errors in Table 3. Unlike the inextensibility postprocess, the overall scaling of the zero-length spring stiffness matters for the quasistatic postprocess. Thus, we compare errors for three different global scalings: $\mu = 1$, 0.1, and 0.01. When $\mu = 1$, the quasistatic postprocess reduces SqrtMSE; however, including the postprocess in training generally yields only minor improvements. This is because the zero-length springs are too strong relative to the material model, and thus the material model is unable to provide a significant correction to the output of the network. Table 4 substantiates this, showing relatively high errors in compression/stretching energies for $\mu = 1$. Next, $\mu = 0.1$ can be viewed as nearly optimal based on Figure 7, and in this case, including the postprocess in training yields up to around 25% improvement. Table 4 shows that the errors in energy for $\mu = 0.1$ are quite small. When the scaling is 0.01 the springs are too weak, and the postprocess by itself does a poor job matching the ground truth because the material model dominates the network prediction pulling the final results quite far from the ground truth. In this case, including the postprocess in training makes significant improvements.

6.2. Optimizing for Zero-Length Spring Stiffness

Instead of experimentally determining the best global scaling μ for the zero-length springs, one could augment the network to choose this parameter itself. Replacing $\hat{r}$ with μ in Equation 12 and the related discussion results in solving

$$\frac{\partial f}{\partial r^*}\frac{\partial r^*}{\partial \mu} = -\frac{\partial f_Z}{\partial \mu} \tag{15}$$

12

Scaling	Method	Training set		Validation set		Test set	
		SqrtMSE	MaxDist	SqrtMSE	MaxDist	SqrtMSE	MaxDist
N/A	[31]	2.5364	10.896	7.5385	29.322	7.5131	29.308
$\mu = 1$	Postprocess Only	2.2014	10.333	7.3195	29.371	7.2918	29.604
	Trained Postprocess	2.1754	9.7121	7.2804	29.716	7.2794	31.191
$\mu = 0.1$	Postprocess Only	2.2268	12.881	7.2168	30.371	7.1879	30.599
	Trained Postprocess	1.7223	10.864	7.1435	30.246	7.1465	31.191
$\mu = 0.01$	Postprocess Only	4.3621	22.045	8.8966	34.594	8.9365	35.022
	Trained Postprocess	2.3120	16.303	7.3002	31.718	7.3596	32.494
variable μ_i	Trained Postprocess	1.6213	10.969	6.8900	31.027	6.9375	31.816

Table 3: Errors on a data set with 1000 training examples in the energy function. Numbers are in millimeters.

Scaling	Training Set		Validation Set		Test Set	
	Compression	Stretching	Compression	Stretching	Compression	Stretching
$\mu = 1$	4.5191	1.0058	14.612	4.1526	14.897	4.0208
$\mu = 0.1$	0.4444	0.4786	1.7589	0.4661	1.7435	0.4415
$\mu = 0.01$	1.5587	0.3319	1.6390	0.6551	1.7534	0.6847
variable μ_i	0.3183	0.4580	0.5990	0.4289	0.6433	0.4717

Table 4: Compression/stretching energy errors for different scalings of the zero-length spring stiffness.

to find $\frac{\partial r^*}{\partial \mu}$, similar to Equation 13 except with a different right hand side. Then, similar to Equation 14, we obtain:

$$\frac{\partial E}{\partial \mu} = \frac{\partial E}{\partial r^*} \frac{\partial r^*}{\partial \mu} = \frac{\partial E}{\partial r^*} H^{-1} \frac{\partial f_Z}{\partial \mu} = \left(\frac{\partial E}{\partial r^*} H^{-1} \right) \frac{\partial f_Z}{\partial \mu} \tag{16}$$

where $\eta = \frac{\partial E}{\partial r^*} H^{-1}$ was already computed for Equation 14. Starting with an initial guess of $\mu = 0.1$, we obtain $\mu = 0.0984$ confirming the optimality of $\mu \approx 0.1$. Furthermore, starting with initial guesses of $\mu = 1$ and $\mu = 0.01$, the network converged to $\mu = 0.1003$ and 0.0903 respectively. Instead of choosing μ globally, one might ansatz that different regions of the cloth would benefit from different scalings. As an experiment, we trained the network to learn μ_i on a per-vertex basis using the appropriate analog of Equations 15 and 16. Figure 10 illustrates that stronger scalings are learned near the upper neck and back as opposed to the midsection of the shirt. As shown in Tables 3 and 4, allowing the network to determine μ_i yields quite good results.

7. Conclusion

Although networks are trained to match ground truth data and regularized to interpolate reasonably well away from the training data, this is typically in an averaged sense which may allow for poor local behavior and a lack of physical constraints. In fact, the resulting mesh could be ill-conditioned or could contain inverted elements, prohibiting any simulation whatsoever. Even when the results may be simulated, the initial conditions may contain large non-physical energies that lead to spurious behaviors including fracture, leading to the failure of the simulator. Thus, we proposed the notion of a postprocess that projects the output of a network into a feasible state ready for simulation. Notably, this postprocess must be robust enough to accept network output as an initial condition. We also require the postprocess to be differentiable so that we may include it in the network training such that the final result not only has good physical properties but also well matches the training data.

We proposed two such postprocesses in this paper and applied both to the network from [31] that predicts cloth vertex positions from joint angles. The first postprocess simply constrains mesh edges to not overstretch and was reformulated into a second order cone program and shown to greatly reduce the non-physical stretching energy in the network output. The second postprocess was based on a robust quasistaic simulation model that deforms the output of the network into a state with highly improved stretching/compression energies. In both cases, zero length spring constraints were used to tie the network
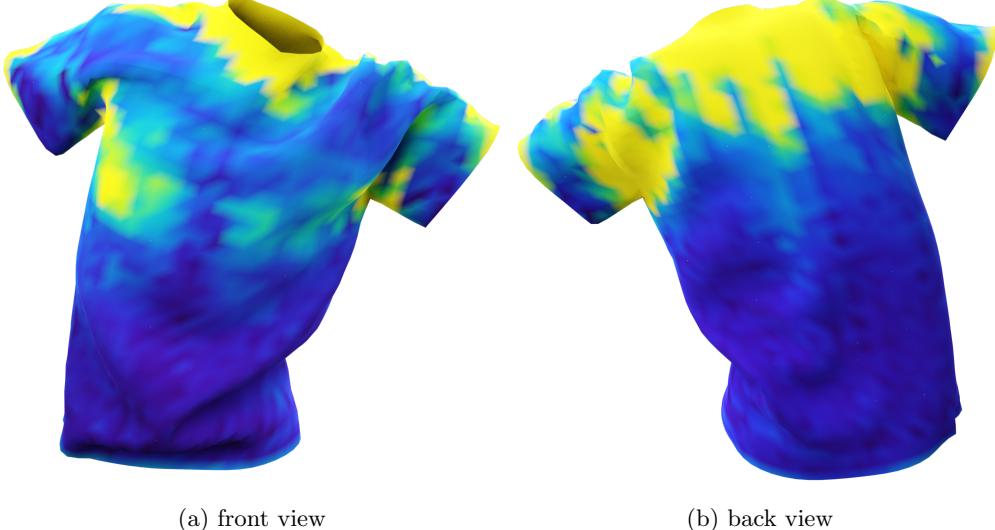
(a) front view   (b) back view

Figure 10: Spatially varying scaling. Yellow indicates $\mu_i = 0.2$, and purple indicates $\mu_i = 0.01$.

predictions to the final result. In the case of the second order cone program, the final results are independent of the overall scaling/strength of the zero-length spring constraints. In the quasistatics case, the results are sensitive to this scaling, and we showed that the network could be trained to learn the scaling; in fact, it learned a value similar to that hypothesized to be optimal experimentally. Moreover, we also showed that the network could learn a spatially varying scaling that produced quite good results. We hypothesize that even better results could probably be obtained by enforcing various smoothness or regional constraints on the local scaling.

Acknowledgement

Appendix A

Here, we provide a column rank analysis for the coefficient matrix in Equation 7. For the sake of exposition, we write

$$
G_0 = \begin{bmatrix} -\sqrt{k_i^z} I & \\ & \ddots \end{bmatrix},
$$

$$
G_e = \begin{bmatrix} 0\cdots & I & 0\cdots & -I & 0\cdots \end{bmatrix}, \quad I_e = \begin{bmatrix} 0\cdots & -1 & 0\cdots \end{bmatrix},
$$

$$
s_0^0 = t, \quad s_0^1 = \begin{bmatrix} \sqrt{k_i^z}(r_i - \hat{r}_i) \\ \vdots \end{bmatrix}, \quad s_e^0 = l_e^{\max}, \quad s_e^1 = \vec{l}_e(r) \tag{17}
$$

so that Equation 7 can be written in more detail as follows:

$$
\begin{bmatrix}
& & I & -1 & G_0^T & G_e^T & \cdots & & \\
& I & & & I_e^T & & & & \\
\hline
& I & & & & & & & \\
-1 & & & & & & & & \\
G_0 & & & & & & & & \\
G_e & I_e & & & & & & I & \\
\vdots & & & & & & & & \\
\hline
& & & s_0^0 \ s_0^{1T} & & z_0^0 \ z_0^{1T} & & & \\
& & & s_0^1 \ s_0^0 I & & z_0^1 \ z_0^0 I & & & \\
& & & & s_e^0 \ s_e^{1T} & & z_e^0 \ z_e^{1T} & & \\
& & & & s_e^1 \ s_e^0 I & & z_e^1 \ z_e^0 I & & \\
& & & & & \ddots & & \ddots &
\end{bmatrix}
\begin{bmatrix}
dr \\ dt \\ d\alpha \\ \hline dy \\ dz_0^0 \\ dz_0^1 \\ dz_e^0 \\ dz_e^1 \\ \vdots \\ ds_0^0 \\ ds_0^1 \\ ds_e^0 \\ ds_e^1 \\ \vdots
\end{bmatrix}
= \begin{bmatrix} \\ \hline \bar{dh} \\ \hline \end{bmatrix}. \tag{18}
$$

The second row partition is $I d\alpha = 0$, and thus setting $d\alpha = 0$ removes the second row partition as well as the third block column. The third block row is $I dy + \sum_e I_e^T dz_e^0 = 0$, and so dy can be computed from dz_e^0 independent of the coupled system; thus, the third block row and the fourth block column can be removed. The resulting system has the following condition on any potential null space vector:

$$
\begin{bmatrix}
& & G_0^T & G_e^T & \cdots & & \\
& -1 & & & & & \\
\hline
-1 & & & & & & \\
G_0 & & & & & & \\
G_e & & & & & I & \\
\vdots & & & & & & \\
\hline
& s_0^0 \ s_0^{1T} & & z_0^0 \ z_0^{1T} & & & \\
& s_0^1 \ s_0^0 I & & z_0^1 \ z_0^0 I & & & \\
& & s_e^0 \ s_e^{1T} & & z_e^0 \ z_e^{1T} & & \\
& & s_e^1 \ s_e^0 I & & z_e^1 \ z_e^0 I & & \\
& & & \ddots & & \ddots &
\end{bmatrix}
\begin{bmatrix}
b_r \\ b_t \\ \hline c_0^0 \\ c_0^1 \\ c_e^0 \\ c_e^1 \\ \vdots \\ d_0^0 \\ d_0^1 \\ d_e^0 \\ d_e^1 \\ \vdots
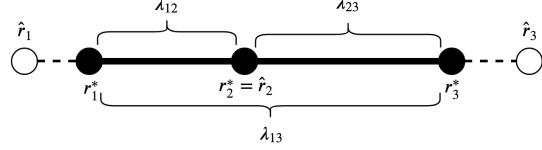\end{bmatrix}
= 0. \tag{19}
$$

Figure 11: Here, the optimal r_1^*, r_2^*, r_3^* are in a linear configuration, targeting the network predicted $\hat{r}_1, \hat{r}_2, \hat{r}_3$ also in a linear configuration. All three edges of this triangle need to be maximally stretched in order for it to be a null space candidate. The constraint forces on λ_{12} and λ_{23} need to balance each other so that r_2^* remains coincident with $\hat{r}_2$. The total constraint force of $\lambda_{12} + \lambda_{13}$ on r_1^* needs to remain fixed as well as $\lambda_{23} + \lambda_{13}$ on r_3^*. Then, one can freely increase/decrease the force λ_{12} and λ_{23} applied to r_1^* and r_3^* (keeping the forces on r_2^* balanced), while decreasing/increasing the force on λ_{13} so that the net forces on r_1^* and r_3^* do not change.

377   From Equation 9 , the determinant of M_q is $(q_0^2 - q_1^T q_1)q_0^{m-2}$. When $q_0 = 0$, then $q_0 \geq \|q_1\|_2$ forces

378    $q_1 = 0$ and M_q is rank 0. When $q_0 > 0$, M_q has either full rank, or has a single null space vector $\begin{bmatrix} q_0 \\ -q_1 \end{bmatrix}$.

379   Firstly, consider only the first column partition of Equation 19. Since $\begin{bmatrix} -1 \\ G_0 \end{bmatrix}$ has full rank, the first

380   column partition has full rank amongst itself.

381   Secondly, expand to consider the first two column partitions of Equation 19. The first two column

382   partitions are block orthogonal, so we only need consider the second column partition. If $s_0^0 = 0$ (i.e. $t=0$),

383   then $s_0^1 = 0$ (i.e. $r = \hat{r}$, meaning that the input $\hat{r}$ is already optimal, with no over-stretched edges, alleviating

384   the need for any perturbations that would stretch zero-length springs). In this case, with $r = \hat{r}$, the null

385   space vector is unimportant since we simply have $\frac{\partial r}{\partial \hat{r}} = I$. So in the following, we only discuss the case when

386    $s_0^0 > 0$. When $s_0^0 > 0$, (i.e. $t > 0$), we have $t^2 = \sum_i k_i^z \|r_i - \hat{r}_i\|_2^2$ i.e. $(s_0^0)^2 - s_0^{1T} s_0^1 = 0$. The second block

387   row containing only a (-1) sets $c_0^0 = 0$, and then the row with $s_0^0 I$ forces $c_0^1 = 0$. If $(s_e^0)^2 - s_e^{1T} s_e^1 > 0$, then

388    $\begin{bmatrix} s_e^0 & s_e^{1T} \\ s_e^1 & s_e^0 I \end{bmatrix}$ is invertible making $\begin{bmatrix} c_e^0 \\ c_e^1 \end{bmatrix} = 0$. So, only consider the set of all edges where $(s_e^0)^2 - s_e^{1T} s_e^1 = 0$

389   (i.e. maximally stretched). First, note that $\begin{bmatrix} s_e^0 & s_e^{1T} \\ s_e^1 & s_e^0 I \end{bmatrix} \begin{bmatrix} c_e^0 \\ c_e^1 \end{bmatrix} = 0$ implies $\begin{bmatrix} c_e^0 \\ c_e^1 \end{bmatrix} = k_e \begin{bmatrix} s_e^0 \\ -s_e^1 \end{bmatrix}$ for arbitrary k_e.

390   Second, consider $\sum_e G_e^T c_e^1 = 0$. The rows for vertex i have the form $\sum_{e \in N_i} \pm k_e s_e^1 = 0$ where N_i is the set of

391   maximally stretched edges that contain vertex i. A simple example of the null space is shown in Figure 11. In

392   general, a null space exists only when degenerate configurations exist within the set of maximally stretched

393   edges where varying constraint forces all produce the same net force on the zero length springs.

394   Thirdly, expand to consider the full matrix. We will show that all of the d components of the the null

395   space vector in Equation 19 must be zero which means that the null space analysis of the first two column

396   partitions covers the entire null space.

397   The optimality conditions (Equation 6 and $s \circ z = 0$) can be written in more detail as follows:

$$
\begin{bmatrix}
& & & \begin{pmatrix} & G_0^T \\ -1 & \\ & I_e^T \end{pmatrix} & \begin{pmatrix} & G_e^T \\ & \end{pmatrix} & \cdots \\
\hline
& I & & & & \\
\begin{pmatrix} -1 & \\ G_0 & \\ & I_e \\ G_e & \end{pmatrix} & & & & & & I \\
\vdots & & & & &
\end{bmatrix}
\begin{bmatrix}
r \\ t \\ \alpha \\ \hline y \\ z_0^0 \\ z_0^1 \\ z_e^0 \\ z_e^1 \\ \vdots \\ s_0^0 \\ s_0^1 \\ s_e^0 \\ s_e^1 \\ \vdots
\end{bmatrix}
=
\begin{bmatrix}
-1 \\ \hline b \\ G_0 \hat{r} \\ \vdots
\end{bmatrix}
$$

$$(20a)$$

$$
\begin{bmatrix} s_0^0 & s_0^{1T} \\ s_0^1 & s_0^0 I \end{bmatrix}
\begin{bmatrix} z_0^0 \\ z_0^1 \end{bmatrix} = 0, \quad
\begin{bmatrix} s_e^0 & s_e^{1T} \\ s_e^1 & s_e^0 I \end{bmatrix}
\begin{bmatrix} z_e^0 \\ z_e^1 \end{bmatrix} = 0. \tag{20b}
$$

The second block row of Equation 20a is $(-1)z_0^0 = -1$, implying $z_0^0 = 1 > 0$. Switching the first two block columns in Equation 19 to make the first column partition strictly diagonal, followed by eliminating the third column partition using the identity matrix results in:

$$
\begin{bmatrix}
& \begin{pmatrix} -1 & \\ & G_0^T \end{pmatrix} & \begin{pmatrix} & G_e^T \end{pmatrix} & \cdots \\
\hline
\begin{pmatrix} z_0^0 & -z_0^{1T}G_0 \\ z_0^1 & -z_0^0 G_0 \\ & -z_e^{1T}G_e \\ & -z_e^0 G_e \\ & \vdots \end{pmatrix} & \begin{pmatrix} s_0^0 & s_0^{1T} \\ s_0^1 & s_0^0 I \end{pmatrix} & \begin{pmatrix} s_e^0 & s_e^{1T} \\ s_e^1 & s_e^0 I \end{pmatrix} & \ddots
\end{bmatrix}
\begin{bmatrix}
b_t \\ \hline b_r \\ c_0^0 \\ c_0^1 \\ c_e^0 \\ c_e^1 \\ \vdots
\end{bmatrix} = 0. \tag{21}
$$

As a side note, scaling the second column partition of Equation 21 by Z is an alternative symmetrization to that we proposed in Equation 8 (i.e. column scaling by S). When $(s_e^0)^2 - s_e^{1T}s_e^1 > 0$, Equation 20b implies $\begin{bmatrix} z_e^0 \\ z_e^1 \end{bmatrix} = 0$, which in turn zeros out the corresponding rows in the first column partition of Equation 21, resulting in $\begin{bmatrix} c_e^0 \\ c_e^1 \end{bmatrix} = 0$ as well. Thus, we only need to consider a sub-system with $(s_e^0)^2 - s_e^{1T}s_e^1 = 0$ for all e hereafter.

From $s_e^0 = l_e^{\max} > 0$ and Equation 20b, we have $\begin{bmatrix} z_e^0 \\ z_e^1 \end{bmatrix} = k_e \begin{bmatrix} s_e^0 \\ -s_e^1 \end{bmatrix}$ where $k_e \geq 0$ because $z_e^0 \geq 0$. Substituting this into Equation 21 modifies the corresponding block rows to be

$$
\begin{bmatrix} k_e s_e^{1T} G_e & s_e^0 & s_e^{1T} \\ -k_e s_e^0 G_e & s_e^1 & s_e^0 I \end{bmatrix}
\begin{bmatrix} b_r \\ c_e^0 \\ c_e^1 \end{bmatrix} = 0. \tag{22}
$$

Premultiplying with $\begin{bmatrix} s_e^0 & -s_e^{1T} \end{bmatrix}$ zeros out the last two columns obtaining $2k_e s_e^0 s_e^{1T} G_e b_r = 0$ or $k_e s_e^{1T} G_e b_r = 0$ or $z_e^{1T} G_e b_r = 0$. The first block row of Equation 21 implies $c_0^0 = 0$. From $s_0^0 > 0$, $(s_0^0)^2 - s_0^{1T}s_0^1 = 0$, and Equation 20b, we have $\begin{bmatrix} z_0^0 \\ z_0^1 \end{bmatrix} = k_0 \begin{bmatrix} s_0^0 \\ -s_0^1 \end{bmatrix}$ where $k_0 > 0$ because $z_0^0 > 0$. Substituting this into Equation 21

17

modifies the corresponding block rows to be

$$\begin{bmatrix} k_0 s_0^0 & k_0 s_0^{1T} G_0 & s_0^{1T} \\ -k_0 s_0^1 & -k_0 s_0^0 G_0 & s_0^0 I \end{bmatrix} \begin{bmatrix} b_t \\ b_r \\ c_0^1 \end{bmatrix} = 0. \tag{23}$$

Premultiplying with $\begin{bmatrix} s_0^0 & s_0^{1T} \end{bmatrix}$ zeros out the first two columns obtaining $2s_0^0 s_0^{1T} c_0^1 = 0$ or $s_0^{1T} c_0^1 = 0$. Then the first row of Equation 23 is $k_0 s_0^0 b_t + k_0 s_0^{1T} G_0 b_r = 0$ or $z_0^0 b_t - z_0^{1T} G_0 b_r = 0$, which can be written as $z_0^0 b_t + \sum_e z_e^{1T} G_e b_r = 0$ using the first row of Equation 20a. Since we have shown that $z_e^1 = 0$ when $(s_e^0)^2 - s_e^{1T} s_e^1 > 0$ and $z_e^{1T} G_e b_r = 0$ when $(s_e^0)^2 - s_e^{1T} s_e^1 = 0$, $\sum_e z_e^{1T} G_e b_r$ is identically 0 making $b_t = 0$ because $z_0^0 > 0$.

The reduced subsystem becomes:

$$\begin{bmatrix} & & G_0^T & (& G_e^T &) & \cdots \\ \hline -z_0^{1T} G_0 & & s_0^{1T} & & & & \\ -z_0^0 G_0 & & s_0^0 I & & & & \\ -z_e^{1T} G_e & & & & s_e^0 & s_e^{1T} & \\ -z_e^0 G_e & & & & s_e^1 & s_e^0 I & \\ \vdots & & & & & & \ddots \end{bmatrix} \begin{bmatrix} b_r \\ \hline c_0^1 \\ c_e^0 \\ c_e^1 \\ \vdots \end{bmatrix} = 0. \tag{24}$$

The third block row gives $c_0^1 = \frac{z_0^0}{s_0^0} G_0 b_r$, and the fifth, seventh, etc. block rows give $c_e^1 = \frac{z_e^0}{s_e^0} G_e b_r - \frac{c_e^0}{s_e^0} s_e^1$.
Substituting these into the first block row gives $M b_r - \sum_e \frac{c_e^0}{s_e^0} G_e^T s_e^1 = 0$ where $M = \left(\frac{z_0^0}{s_0^0} G_0^T G_0 + \sum_e \frac{z_e^0}{s_e^0} G_e^T G_e \right)$
is symmetric positive definite. Thus, Equation 24 reduces to:

$$\begin{bmatrix} M & -\frac{1}{s_e^0} G_e^T s_e^1 \cdots \\ \hline k_e s_e^{1T} \bar{G}_e & \\ \vdots & \end{bmatrix} \begin{bmatrix} b_r \\ \hline c_e^0 \\ \vdots \end{bmatrix} = 0 \tag{25}$$

where the second the the subsequent rows come from premultiplying Equation 22 with $\begin{bmatrix} s_e^0 & -s_e^{1T} \end{bmatrix}$. Assuming $k_e \neq 0$, we can row scale by $-\frac{1}{s_e^0 k_e}$ to obtain $\begin{bmatrix} M & N \\ N^T & \end{bmatrix} \begin{bmatrix} b_r \\ c \end{bmatrix} = 0$ where $N = \begin{bmatrix} -\frac{1}{s_e^0} G_e^T s_e^1 \cdots \end{bmatrix}, c = \begin{bmatrix} c_e^0 \cdots \end{bmatrix}^T$.
The first equation gives $b_r = -M^{-1} N c$ and the second equation gives $N^T b_r = -N^T M^{-1} N c = 0$. Therefore, $c^T N^T M^{-1} N c = 0$ and thus $N c = 0$ and $b_r = 0$. Finally setting b_r and b_t to 0 eliminates the first column partition in Equation 19 so that the identity matrix in the third column partition sets all of the d components of the null space vector to 0.

Finally, if $k_e = 0$ for some edge, then $\begin{bmatrix} z_e^0 \\ z_e^1 \end{bmatrix} = 0$, satisfying Equation 20b trivially and also removing the

columns corresponding to $\begin{bmatrix} z_e^0 \\ z_e^1 \end{bmatrix}$ in Equation 20a. Then $\begin{bmatrix} s_e^0 \\ s_e^1 \end{bmatrix}$ decouples from the system. That is, such edges need not appear in the coupled optimality conditions, and thus our null space discussion may assume $k_e \neq 0$ with completeness.

References

[1] J. Ling, R. Jones, J. Templeton, Machine learning strategies for systems with invariance properties, Journal of Computational Physics 318 (2016) 22–35.

[2] E. J. Parish, K. Duraisamy, A paradigm for data-driven predictive modeling using field inversion and machine learning, Journal of Computational Physics 305 (2016) 758–774.

[3] M. Raissi, P. Perdikaris, G. E. Karniadakis, Machine learning of linear differential equations using gaussian processes, Journal of Computational Physics 348 (2017) 683–693.

[4] M. Raissi, G. E. Karniadakis, Hidden physics models: Machine learning of nonlinear partial differential equations, Journal of Computational Physics 357 (2018) 125–141.

[5] R. K. Tripathy, I. Bilionis, Deep uq: Learning deep neural network surrogate models for high dimensional uncertainty quantification, Journal of Computational Physics 375 (2018) 565–588.

[6] J. Sirignano, K. Spiliopoulos, Dgm: A deep learning algorithm for solving partial differential equations, Journal of Computational Physics 375 (2018) 1339–1364.

[7] Y. Qi, J. Lu, R. Scardovelli, S. Zaleski, G. Tryggvason, Computing curvature for volume of fluid methods using machine learning, Journal of Computational Physics 377 (2019) 155–161.

[8] H. Chang, D. Zhang, Identification of physical processes via combined data-driven and data-assimilation methods, Journal of Computational Physics 393 (2019) 337–350.

[9] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational Physics 378 (2019) 686–707.

[10] K. Yeo, I. Melnyk, Deep learning algorithm for data-driven simulation of noisy dynamical system, Journal of Computational Physics 376 (2019) 1212–1231.

[11] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, P. Perdikaris, Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data, Journal of Computational Physics 394 (2019) 56–81.

[12] P. Stinis, T. Hagge, A. M. Tartakovsky, E. Yeung, Enforcing constraints for interpolation and extrapolation in generative adversarial networks, Journal of Computational Physics 397 (2019) 108844.

[13] I. Goodfellow, Y. Bengio, A. Courville, Deep learning, MIT press, 2016.

[14] T. Hastie, R. Tibshirani, J. Friedman, J. Franklin, The elements of statistical learning: data mining, inference and prediction, The Mathematical Intelligencer 27 (2005) 83–85.

[15] A. Ng, Cs229 lecture notes on learning theory, 2019. URL: http://cs229.stanford.edu/summer2019/cs229-notes4.pdf.

[16] F.-F. Li, J. Johnson, S. Yeung, Cs231n lecture notes on training neural networks, 2019. URL: http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture08.pdf.

[17] C. G. Broyden, The convergence of a class of double-rank minimization algorithms 1. general considerations, IMA Journal of Applied Mathematics 6 (1970) 76–90.

[18] D. C. Liu, J. Nocedal, On the limited memory bfgs method for large scale optimization, Mathematical programming 45 (1989) 503–528.

[19] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, A. Y. Ng, Large scale distributed deep networks, in: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, NIPS'12, Curran Associates Inc., USA, 2012, pp. 1223–1231.

[20] H. Robbins, S. Monro, A stochastic approximation method, The Annals of Mathematical Statistics (1951) 400–407.

[21] S. J. Wright, Coordinate descent algorithms, Mathematical Programming 151 (2015) 3–34.

[22] M. Bao, X. Hua, R. Fedkiw, Improved search strategies for determining facial expression, arXiv preprint arXiv:1812.02897 (2018).

[23] B. T. Polyak, Some methods of speeding up the convergence of iteration methods, USSR Computational Mathematics and Mathematical Physics 4 (1964) 1–17.

[24] Y. E. Nesterov, A method for solving the convex programming problem with convergence rate $o(1/k^2)$, in: Doklady Akademii nauk SSSR, volume 269, 1983, pp. 543–547.

[25] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, Journal of Machine Learning Research 12 (2011) 2121–2159.

[26] M. D. Zeiler, Adadelta: an adaptive learning rate method, arXiv preprint arXiv:1212.5701 (2012).

[27] T. Tieleman, G. Hinton, Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude, COURSERA: Neural networks for machine learning 4 (2012) 26–31.

[28] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).

[29] T. Dozat, Incorporating nesterov momentum into adam (2016).

[30] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, Li Fei-Fei, Imagenet: A large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 248–255.

[31] N. Jin, Y. Zhu, Z. Geng, R. Fedkiw, A pixel-based framework for data-driven clothing, arXiv preprint arXiv:1812.01677 (2018).

[32] B. Amos, J. Z. Kolter, Optnet: Differentiable optimization as a layer in neural networks, in: Proceedings of the 34th International Conference on Machine Learning-Volume 70, JMLR. org, 2017, pp. 136–145.

[33] S. Boyd, L. Vandenberghe, Convex optimization, Cambridge university press, 2004.

[34] A. Agrawal, S. Barratt, S. Boyd, E. Busseti, W. Moursi, Differentiating through a cone program, Journal of Applied and Numerical Optimization 1 (2019) 107–115.

[35] J. Teran, S. Blemker, V. Hing, R. Fedkiw, Finite volume methods for the simulation of skeletal muscle, in: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, Eurographics Association, 2003, pp. 68–74.

[36] G. Irving, J. Teran, R. Fedkiw, Invertible finite elements for robust simulation of large deformation, in: Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '04, Eurographics Association, Goslar Germany, Germany, 2004, pp. 131–140.

[37] E. Sifakis, I. Neverov, R. Fedkiw, Automatic determination of facial muscle activations from sparse motion capture marker data, ACM Trans. Graph. 24 (2005) 417–425.

[38] M. Cong, K. S. Bhat, R. Fedkiw, Art-directed muscle simulation for high-end facial animation, in: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '16, Eurographics Association, Goslar Germany, Germany, 2016, pp. 119–127.

[39] M. Bao, M. Cong, S. Grabli, R. Fedkiw, High-quality face capture using anatomical muscles, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 10802–10811.

[40] N. Magnenat-Thalmann, R. Laperrière, D. Thalmann, Joint-dependent local deformations for hand animation and object grasping, in: Proceedings on Graphics Interface '88, Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 1988, pp. 26–33.

[41] S. S. Blemker, S. L. Delp, Three-dimensional representation of complex muscle architectures and geometries, Annals of Biomedical Engineering 33 (2005) 661–673.

[42] L. Kavan, S. Collins, J. Žára, C. O'Sullivan, Skinning with dual quaternions, in: Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games, I3D '07, ACM, New York, NY, USA, 2007, pp. 39–46.

[43] I. Stavness, C. A. Sánchez, J. Lloyd, A. Ho, J. Wang, S. Fels, D. Huang, Unified skinning of rigid and deformable models for anatomical simulations, in: SIGGRAPH Asia 2014 Technical Briefs, SA '14, ACM, New York, NY, USA, 2014, pp. 9:1–9:4.

[44] B. H. Le, J. K. Hodgins, Real-time skeletal skinning with optimized centers of rotation, ACM Trans. Graph. 35 (2016) 37:1–37:10.

[45] S. Lee, M. Park, K. Lee, J. Lee, Scalable muscle-actuated human simulation and control, ACM Trans. Graph. 38 (2019) 73:1–73:13.

[46] B. H. Le, J. P. Lewis, Direct delta mush skinning and variants, ACM Trans. Graph. 38 (2019) 113:1–113:13.

[47] S. Osher, R. Fedkiw, Level set methods and dynamic implicit surfaces, volume 153, Springer Science & Business Media, 2006.

[48] R. Bridson, S. Marino, R. Fedkiw, Simulation of clothing with folds and wrinkles, in: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '03, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2003, pp. 28–36.

[49] R. Bridson, R. Fedkiw, J. Anderson, Robust treatment of collisions, contact and friction for cloth animation, in: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '02, ACM, New York, NY, USA, 2002, pp. 594–603.

[50] A. Selle, J. Su, G. Irving, R. Fedkiw, Robust high-resolution cloth using parallelism, history-based collisions, and accurate friction, IEEE Transactions on Visualization and Computer Graphics 15 (2009) 339–350.

[51] N. Jin, W. Lu, Z. Geng, R. Fedkiw, Inequality cloth, in: Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation, SCA '17, ACM, New York, NY, USA, 2017, pp. 16:1–16:10.

[52] T. Hughes, R. Taylor, W. Kanoknukulchai, A simple and efficient finite element for plate bending, International Journal for Numerical Methods in Engineering 11 (1977) 1529–1543.

[53] A. Domahidi, E. Chu, S. Boyd, ECOS: An SOCP solver for embedded systems, in: 2013 European Control Conference (ECC), IEEE, 2013, pp. 3071–3076.

[54] A. Selle, M. Lentine, R. Fedkiw, A mass spring model for hair simulation, ACM Trans. Graph. 27 (2008) 64:1–64:11.

[55] J. Teran, E. Sifakis, G. Irving, R. Fedkiw, Robust quasistatic finite elements and flesh simulation, in: Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '05, ACM, New York, NY, USA, 2005, pp. 181–190.