Computational Inverse Design of Surface-based Inflatables Supplementary Material

Julian Panetta, Florin Isvoranu, Tian Chen, Emmanuel Siéfert, Benoît Roman, Mark Pauly

This supplemental document provides more details on the implementation aspects of our numerical solvers (Section 1) as well as all the derivatives needed for the optimization (Section 2). Equation numbers below reference equations in the main paper (except for those prefixed by "A").

1 Numerical Methods and Implementation

1.1 Parametrization strategy

We initialize our flatting by computing a least-squares conformal map Lévy et al. [2002] and then run 1000 iterations of the local-global fitting algorithm minimizing E_{fit} so that the singular values σ_0 and σ_1 are well separated: this is necessary to avoid singularities in our parametrization gradient and Hessian formulas. This process takes only a few seconds on all examples shown, and the resulting singular value histogram gives a good early indication of whether the target surface can feasibly be produced with a single inflatable patch. We then run several rounds of the parametrization algorithm, gradually reducing the regularization weights. A typical weight sequence starts with $(w_\phi, w_\sigma, w_\kappa) = (100, 10, 500)$ and runs 4 additional rounds, dividing w_ϕ and w_σ by 10 and w_κ by 2 after each round. Subsequent runs after the first tend to converge relatively quickly, allowing semi-interactive weight tuning. Since our parametrization energy definition is not mesh dependent and we normalize the input surface dimensions before flattening, similar weights can be used across models.

Our Newton solver's default strategy for coping with indefinite Hessians (like those of our highly nonconvex parametrization energy) is the common approach of adding an increasingly large multiple of the identity matrix until a positive definite matrix is achieved. This trial-and-error process is slow and the modification ends up severely damping the solver's progress. We dramatically accelerate the flattening algorithm by projecting each triangle's contribution to the singular-value-fitting term's Hessian to be positive semi-definite using a 6×6 Eigendecomposition. We use a simple heuristic for enabling and disabling this projection: after 5 consecutive Newton iterations with indefinite Hessians we enable the projection for 15 iterations. We note that while the smoothing and bending regularization terms have a different structure and cannot be handled by this per-triangle projection, they primarily act to increase the Hessian eigenvalues, not decrease them, meaning our simple projection approach is still effective in practice.

1.2 Smoothness of membrane energy density

We note that the relaxed membrane energy density function ψ in (1) derived from tension field theory is only C^1 continuous; it has discontinuous second derivatives, leading to discontinuities in the Hessian of the full potential energy. However, instead of using the smoothed version recommended in Skouras et al. [2014], we elect to work with the original non-smooth energy density and avoid relying on any third or higher derivatives of ψ. This is because the recommended smoothing was performed on ψ's derivatives and does not correspond to a smoothed energy density: the smoothed derivatives are no longer integrable. It is not obvious how to directly smooth the energy density function, and a well-defined energy density function is crucial for formulating the equilibrium problem as an energy minimization (1) instead of a force balance to avoid non-physical unstable equilibria.

1.3 Nullspace of ψ

The tension field theory relaxation of $\psi(F)$ poses two additional challenges for simulation and optimization. First, the Hessian contributions from triangles under compression vanish, which can lead to a singular global Hessian, triggering a Hessian modification that will unnecessarily dampen the Newton steps and slow convergence. We add a small artificial stiffness to these elements, given by the contribution evaluated at the onset of compression ($F = I$) scaled by 10^{-8}. Second, altering the rest shape of elements under compression does not change the equilibrium, meaning the target-fitting objective will not discourage the design optimizer from expanding these rest shapes arbitrarily. To remove this ambiguity and encourage designs with walls in tension (which we hypothesize are likely more stable), we add a further regularization term ($E_{\text{Full}} - E_{\text{TFT}}$) with a low weight ($w_c = 10^{-6}$) to penalize compression. This term measures the difference between the current equilibrium's elastic energy with (E_{Full}) and without (E_{TFT}) the tension field theory relaxation; it vanishes when all elements are in tension and increases as triangles are compressed more and more.

1.4 Design optimization solver

We solve (7) with the L-BFGS-B optimization algorithm Zhu et al. [1997] using the interface provided by `scipy`. This requires computing the gradient $\frac{\partial J}{\partial X_{\text{wall}}}$, which we derive formulas for in Section 2.4.2. We avoid using second-order sensitivity analysis (i.e., computing $\frac{\partial^2 J}{\partial {X_{\text{wall}}}^2}$) since this would require computing third derivatives of the tension field theory energy density ψ, which is only C^1.

We rely on several techniques to accelerate the optimization. First, we cache and reuse the symbolic factorization of the simulation energy's Hessian since the sparsity pattern is fixed throughout the optimization. Second, we use (A4) to construct a first-order prediction of the new equilibrium of the updated design to initialize our Newton solver; this uses the same numeric factorization computed to solve the adjoint equation, (A5). Finally, we avoid wasting time solving for the equilibrium to evaluate J for obviously bad designs by first evaluating the inexpensive collapse barrier and wall smoothness terms; if these terms alone exceed the full objective of the previous design, we bail early, returning a large energy value ($2\times$ the previous iterate).

1.5 Nondimensionalization

The individual terms of our objective in (7) have different units, as do the terms of the simulation energy after $\epsilon T(\mathbf{\Phi})$ is added. This is problematic, since it means weights chosen for one model are likely suboptimal for another with a different scale. However, by dividing by characteristic measurements of the design, we can make each energy term unitless and scale-invariant, greatly reducing the parameter tuning effort. In the following, we define $\hat{A}$ and $\hat{w}$ to be the initial fused region area and median wall width, respectively. We divide the potential energy terms in (1) by $Yh|\hat{\mathcal{S}}_0|$, (the Young's modulus times the initial volume of the sheet material) and divide T by the $\hat{A}\hat{w}^2$; now T measures the mean-squared deviation relative to the wall width. To nondimensionalize the design objective, we additionally divide the barrier term by $|\hat{\mathcal{S}}_0|$ and multiply the smoothing term by $\hat{w}$ (which also makes the smoothing term approximately invariant to the stripe pattern frequency). The full resulting design optimization energy is presented in (A3) below. Finally, we employ a simple change of variables, expressing our optimization and simulation variables in units of $\hat{w}$ so that all gradients are scale-invariant too (which is helpful for L-BFGS-B's initial step size selection).

1.6 Weights

Following the nondimensionalization process, we could use the same settings across nearly all models ($w_{\text{sm}} = 1.0$ for the boundary curve and 0.1 for the internal curves; $w_b = 1.0$). We only made small changes on a few models (setting the smoothing weight for internal curves to 0.05 and 0.01 on the Lilium and Annulus models, respectively, to encourage a tighter fit after a first attempt with the default settings). For most models, we can actually run a single round of optimization with $\epsilon = 0$ (or $\epsilon = 10^{-6}$ to constrain rigid motions for models with free boundaries), but some like the Annulus have nearly-unstable equilibria at early optimization steps that prevent the optimizer from making substantial progress. For these, we run an initial optimization round

with a target-attracting weight of $\epsilon = 10^{-3}$ to stabilize the deformation, before dropping ϵ to the default value.

1.7 Remeshing

For two models we tried (Lilium tower and Annulus), the design optimization pulled a fusing curve sufficiently far from the boundary to badly stretch some of the tube triangles. To ensure an accurate simulation and continue optimizing past this point, we implemented a remeshing process that retriangulates the tube regions of $\mathcal{S}_0$ with high-quality triangles using Triangle Shewchuk [1996] and transfers all optimization states over to the new mesh by sampling. Note that this requires an identical retriangulation of $\hat{\mathcal{S}}_0$, which we obtain by interpolating the original $\hat{X}_{\text{wall}}$ on this new mesh. We also transfer the equilibrium deformation from the old mesh so that the corresponding equilibrium of the remeshed sheet can be computed rapidly.

2 Derivatives

2.1 Derivatives of the SVD

Both the flattening and physical simulation stages of our pipeline require gradient and Hessian formulas for the SVD of a map's Jacobian. For the parametrization problem, the map is from $\mathbb{R}^3$ to $\mathbb{R}^2$, while for the simulation problem it is from $\mathbb{R}^2$ to $\mathbb{R}^3$, so the SVD we must differentiate is of either a 2×3 or 3×2 matrix. We consider here the 2×3 case, and the derivatives for the 3×2 case can be found simply by transposing the formulas.

We denote our mapping Jacobian as F (which corresponds to either ∇f or $\nabla \mathbf{\Phi}$) and write its SVD as:

$$F = U \underbrace{\begin{bmatrix} \sigma_0 & 0 & 0 \\ 0 & \sigma_1 & 0 \end{bmatrix}}_{\Sigma} V^T \qquad U \in O(2), V \in O(3),$$

where U and V contain the left and right singular vectors as columns:

$$U = \begin{bmatrix} \mathbf{u}_0 & \mathbf{v}_0 \end{bmatrix}, \quad V = \begin{bmatrix} \mathbf{v}_0 & \mathbf{v}_1 & \mathbf{v}_2 \end{bmatrix}.$$

We observe that $\mathbf{v}_2$ is the surface normal, while $\mathbf{v}_1$ and $\mathbf{v}_2$ form an orthonormal basis for the tangent plane.

2.1.1 Gradients

We compute first derivatives of the SVD by differentiating both sides of this relationship, considering how the quantities change when F is perturbed with a "velocity" $\dot{F}$:

$$\dot{F} = \dot{U}\Sigma V^T + U\dot{\Sigma}V^T + U\Sigma\dot{V}^T \quad \Longrightarrow \quad U^T\dot{F}V = U^T\dot{U}\Sigma + \dot{\Sigma} + \Sigma\dot{V}^TV \tag{A1}$$

Differentiating the relationships $U^TU = \text{Id}_{2\times 2}$ and $V^TV = \text{Id}_{3\times 3}$ reveals that $U^T\dot{U}$ and $\dot{V}^TV$ are skew symmetric and thus can be written as the infinitesimal rotations:

$$U^T\dot{U} := \begin{bmatrix} 0 & -\alpha \\ \alpha & 0 \end{bmatrix}, \qquad \dot{V}^TV = -V^T\dot{V} := -\begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix},$$

where α and $\boldsymbol{\omega}$ denote angular velocities of the singular vector frame around the current frame axes. Plugging these expressions into (A1) and simplifying, we find:

$$U^T\dot{F}V = \begin{bmatrix} \dot{\sigma}_0 & \sigma_0\omega_z - \sigma_1\alpha & -\sigma_0\omega_y \\ \sigma_0\alpha - \sigma_1\omega_z & \dot{\sigma}_1 & \sigma_1\omega_x \end{bmatrix},$$

which immediately gives us an expression for $\dot{\sigma}_0$, $\dot{\sigma}_1$, ω_x, and ω_y and is straightforward to solve for α and ω_z. We conclude the following derivative formulas:

$$\left(\frac{\partial}{\partial F}\begin{bmatrix}\sigma_0\\ \sigma_1\end{bmatrix}\right) : \dot{F} = \begin{bmatrix}\mathbf{u}_0^T\dot{F}\mathbf{v}_0\\ \mathbf{u}_1^T\dot{F}\mathbf{v}_1\end{bmatrix} = \begin{bmatrix}\mathbf{u}_0\mathbf{v}_0^T : \dot{F}\\ \mathbf{u}_1\mathbf{v}_1^T : \dot{F}\end{bmatrix} \quad \Longrightarrow \quad \begin{matrix}\frac{\partial\sigma_0}{\partial F} = \mathbf{u}_0\mathbf{v}_0^T\\ \frac{\partial\sigma_1}{\partial F} = \mathbf{u}_1\mathbf{v}_1^T\end{matrix}$$

$$\left.\begin{aligned}\omega_x &= -\frac{1}{\sigma_0}\mathbf{u}_0^T\dot{F}\mathbf{v}_2\\ \omega_y &= \frac{1}{\sigma_1}\mathbf{u}_1^T\dot{F}\mathbf{v}_2\\ \omega_z &= \frac{1}{\sigma_0^2-\sigma_1^2}\left(\sigma_1\mathbf{u}_1^T\dot{F}\mathbf{v}_0 + \sigma_0\mathbf{u}_0^T\dot{F}\mathbf{v}_1\right)\end{aligned}\right\} \quad \Longrightarrow \quad \begin{aligned}\frac{\partial\mathbf{v}_0}{\partial F} &= \mathbf{v}_1\otimes W_z - \mathbf{v}_2\otimes W_y\\ \frac{\partial\mathbf{v}_1}{\partial F} &= -\mathbf{v}_0\otimes W_z + \mathbf{v}_2\otimes W_x\\ \frac{\partial\mathbf{v}_2}{\partial F} &= \mathbf{v}_0\otimes W_y - \mathbf{v}_1\otimes W_x\end{aligned}$$

$$\alpha = \frac{1}{\sigma_0^2-\sigma_1^2}\left(\sigma_0\mathbf{u}_1^T\dot{F}\mathbf{v}_0 + \sigma_1\mathbf{u}_0^T\dot{F}\mathbf{v}_1\right) \quad \Longrightarrow \quad \begin{aligned}\frac{\partial\mathbf{u}_0}{\partial F} &= \mathbf{u}_1\otimes A\\ \frac{\partial\mathbf{u}_1}{\partial F} &= -\mathbf{u}_0\otimes A,\end{aligned}$$

where we introduced the following matrices whose double contraction with $\dot{F}$ computes the corresponding angular velocities:

$$\begin{aligned}W_x &:= -\frac{\mathbf{u}_0\mathbf{v}_2^T}{\sigma_0}\\ W_y &:= \frac{\mathbf{u}_1\mathbf{v}_2^T}{\sigma_1}\\ W_z &:= \frac{\sigma_1\mathbf{u}_1\mathbf{v}_0^T + \sigma_0\mathbf{u}_0\mathbf{v}_1^T}{\sigma_0^2-\sigma_1^2}\\ A &:= \frac{\sigma_0\mathbf{u}_1\mathbf{v}_0^T + \sigma_1\mathbf{u}_0\mathbf{v}_1^T}{\sigma_0^2-\sigma_1^2}\end{aligned} \quad \Longrightarrow \quad \begin{aligned}\omega_x &= W_x : \dot{F}\\ \omega_y &= W_y : \dot{F}\\ \omega_z &= W_z : \dot{F}\\ \alpha &= A : \dot{F}.\end{aligned}$$

We note that if the triangle's normal in 3D does not rotate (as is the case for the parametrization problem), both ω_x and ω_y will vanish, and all expressions involving $\mathbf{v}_2$ can be neglected.

2.1.2 Hessians

We compute Hessians with respect to F by differentiating our first derivative expressions. To avoid working with higher order tensors, we evaluate directional derivatives along an arbitrary $\dot{F}$. In fact, our implementation uses only these directional derivative expressions and not the full tensors, e.g. feeding in a FEM shape function's Jacobian as $\dot{F}$ when constructing the energy Hessian matrix. The singular value Hessians are:

$$\begin{aligned}\frac{\partial^2\sigma_0}{\partial F^2} : \dot{F} &= \dot{\mathbf{u}}_0\mathbf{v}_0^T + \mathbf{u}_0\dot{\mathbf{v}}_0^T = \alpha\mathbf{u}_1\mathbf{v}_0^T + \omega_z\mathbf{u}_0\mathbf{v}_1^T - \omega_y\mathbf{u}_0\mathbf{v}_2^T,\\ \frac{\partial^2\sigma_1}{\partial F^2} : \dot{F} &= \dot{\mathbf{u}}_1\mathbf{v}_1^T + \mathbf{u}_1\dot{\mathbf{v}}_1^T = -\alpha\mathbf{u}_0\mathbf{v}_1^T - \omega_z\mathbf{u}_1\mathbf{v}_0^T + \omega_x\mathbf{u}_1\mathbf{v}_2^T.\end{aligned}$$

These singular value expressions are enough to compute Hessians of the elastic sheet's energy density. For the parametrization energy, the Hessians of singular vectors $\mathbf{u}_0$ and $\mathbf{v}_1$ are additionally needed:

$$\begin{aligned}\frac{\partial^2\mathbf{u}_0}{\partial F^2} : \dot{F} &= -\alpha\mathbf{u}_0\otimes A + \mathbf{u}_1\otimes\dot{A}\\ \frac{\partial^2\mathbf{v}_1}{\partial F^2} : \dot{F} &= -\mathbf{v}_1\otimes(\omega_z W_z + \omega_x W_x) + \mathbf{v}_0\otimes(\omega_y W_x - \dot{W}_z) + \mathbf{v}_2\otimes(\omega_y W_z + \dot{W}_x).\end{aligned}$$

These expressions need the directional derivatives $\dot{A}$, $\dot{W}_z$, and $\dot{W}_x$:

$$\dot{A} = 2\frac{\sigma_1\dot{\sigma}_1 - \sigma_0\dot{\sigma}_0}{\sigma_0^2 - \sigma_1^2}A \quad + \frac{\dot{\sigma}_0\mathbf{u}_1\mathbf{v}_0^T + \dot{\sigma}_1\mathbf{u}_0\mathbf{v}_1^T - (\sigma_0\alpha + \sigma_1\omega_z)\mathbf{u}_0\mathbf{v}_0^T + (\sigma_1\alpha + \sigma_0\omega_z)\mathbf{u}_1\mathbf{v}_1^T + (\sigma_1\omega_x\mathbf{u}_0 - \sigma_0\omega_y\mathbf{u}_1)\,\mathbf{v}_2^T}{\sigma_0^2 - \sigma_1^2}$$

$$\dot{W}_z = 2\frac{\sigma_1\dot{\sigma}_1 - \sigma_0\dot{\sigma}_0}{\sigma_0^2 - \sigma_1^2}W_z + \frac{\dot{\sigma}_1\mathbf{u}_1\mathbf{v}_0^T + \dot{\sigma}_0\mathbf{u}_0\mathbf{v}_1^T - (\sigma_1\alpha + \sigma_0\omega_z)\mathbf{u}_0\mathbf{v}_0^T + (\sigma_0\alpha + \sigma_1\omega_z)\mathbf{u}_1\mathbf{v}_1^T + (\sigma_0\omega_x\mathbf{u}_0 - \sigma_1\omega_y\mathbf{u}_1)\,\mathbf{v}_2^T}{\sigma_0^2 - \sigma_1^2}$$

$$\dot{W}_x = \frac{\dot{\sigma}_0\mathbf{u}_0\mathbf{v}_2^T}{\sigma_0^2} - \frac{\alpha\mathbf{u}_1\mathbf{v}_2^T + \mathbf{u}_0(\omega_y\mathbf{v}_0 - \omega_x\mathbf{v}_1)^T}{\sigma_0}.$$

We again remark that, for the parametrization problem where the 3D triangle's normal does not rotate, the terms involving ω_x, ω_y, and $\mathbf{v}_2$ can be dropped. For other applications that might need them, the Hessian of $\mathbf{u}_1$ follows immediately by realizing $\mathbf{u}_1$ is a 90° rotation of $\mathbf{u}_0$, and Hessians for $\mathbf{v}_0$ and $\mathbf{v}_2$ can be computed similarly to $\mathbf{v}_1$.

2.2 Derivatives of the Parametrization Energy

In this section, we compute derivatives of our target surface flattening energy in the discrete setting, using $\mathbf{x}_i$ to denote the parametric position of vertex i of the target surface so that the piecewise linear flattening map f is given by:

$$f = \sum_i \lambda_i \mathbf{x}_i \quad \Longrightarrow \quad \nabla f = \sum_i \mathbf{x}_i \otimes \nabla\lambda_i,$$

where λ_i are the linear finite element shape functions. The flattening's Jacobian ∇f takes the constant value ∇f_t within each triangle t. Our discrete parametrization energy consists of energy terms contributed by each triangle t and each pair of triangles (i, j) connected by edges in the dual intrinsic Delaunay triangulation:

$$E_{\text{flat}}(\mathbf{x}_0, \cdots, \mathbf{x}_{n-1}) = \sum_t E_t(\nabla f_t)A_t + \sum_{ij} E_{\text{edge}}(\nabla f_i, \nabla f_j)l_{ij}.$$

The triangle contributions include the singular value fitting and the bending regularization terms:

$$E_t(\mathcal{F}) := s\big(\sigma_0(\mathcal{F}), \sigma_1(\mathcal{F})\big) + \frac{w_\kappa}{4}\left\|\mathbf{v}_1(\mathcal{F})\right\|_{S_t}^2,$$

and the edge contributions include the direction and stretch smoothness regularizations:

$$E_{\text{edge}}(\mathcal{F}, \mathcal{G}) := \frac{w_\phi}{2}\left\|\mathbf{u}_0(\mathcal{F}) \times \mathbf{u}_0(\mathcal{G})\right\|^2 + \frac{w_\sigma}{2}\left(\sigma_0(\mathcal{F}) - \sigma_0(\mathcal{G})\right)^2.$$

We introduced the following singular value fitting function to simplify notation:

$$s(\sigma_0, \sigma_1) := \frac{1}{|\mathcal{M}|}\left((\sigma_1 - 1)^2 + (\sigma_0 - \texttt{clamp}(\sigma_0, [\sigma_{\min}, \pi/2]))^2\right).$$

2.2.1 Gradients

We now compute the derivative of E_{flat} with respect to component $c \in \{0, 1\}$ of a mesh vertex's parametric position $\mathbf{x}_k$. First, we note that

$$\frac{\partial \nabla f_t}{\partial x_{k,c}} = \mathbf{e}_c \otimes \nabla\lambda_{k,t},$$

where $\mathbf{e}_c$ is the c^{th} standard basis vector for $\mathbb{R}^3$. This term is nonzero only for triangles t in vertex k's ring of adjacent triangles. We now differentiate:

$$\frac{\partial E_{\text{flat}}}{\partial x_{k,c}} = \sum_t A_t\left(\frac{\partial E_t}{\partial \mathcal{F}} : (\mathbf{e}_c \otimes \nabla\lambda_{k,t})\right) + \sum_{ij} l_{ij}\left(\frac{\partial E_{\text{edge}}}{\partial \mathcal{F}} : (\mathbf{e}_c \otimes \nabla\lambda_{k,i}) + \frac{\partial E_{\text{edge}}}{\partial \mathcal{G}} : (\mathbf{e}_c \otimes \nabla\lambda_{k,j})\right).$$

The individual contribution gradients needed in this expression are:

$$\frac{\partial E_t}{\partial \mathcal{F}}(\mathcal{F}) = \frac{\partial s}{\partial \sigma_0}\frac{\partial \sigma_0}{\partial F}(\mathcal{F}) + \frac{\partial s}{\partial \sigma_1}\frac{\partial \sigma_1}{\partial F}(\mathcal{F}) + w_\kappa(\mathbf{v}_1^T S_t \mathbf{v}_1)\left(\mathbf{v}_1^T S_t \frac{\partial \mathbf{v}_1}{\partial F}(\mathcal{F})\right),$$

$$\frac{\partial E_{\text{edge}}}{\partial \mathcal{F}}(\mathcal{F},\mathcal{G}) = w_\phi\big(\mathbf{u}_0(\mathcal{F}) \times \mathbf{u}_0(\mathcal{G})\big) \cdot \left(\frac{\partial \mathbf{u}_0}{\partial F}(\mathcal{F}) \times \mathbf{u}_0(\mathcal{G})\right) + w_\sigma(\sigma_0(\mathcal{F}) - \sigma_0(\mathcal{G}))\frac{\partial \sigma_0}{\partial F}(\mathcal{F}),$$

$$\frac{\partial E_{\text{edge}}}{\partial \mathcal{G}}(\mathcal{F},\mathcal{G}) = w_\phi\big(\mathbf{u}_0(\mathcal{F}) \times \mathbf{u}_0(\mathcal{G})\big) \cdot \left(\mathbf{u}_0(\mathcal{F}) \times \frac{\partial \mathbf{u}_0}{\partial F}(\mathcal{G})\right) - w_\sigma(\sigma_0(\mathcal{F}) - \sigma_0(\mathcal{G}))\frac{\partial \sigma_0}{\partial F}(\mathcal{G}).$$

The function s is straightforward to differentiate, and expressions for the SVD quantities' derivatives are given in Section 2.1.

2.2.2 Hessians

We compute the Hessian with respect to flattened position component c of mesh vertex k and component d of mesh vertex l:

$$\begin{aligned}\frac{\partial^2 E_{\text{flat}}}{\partial x_{k,c}\partial x_{l,d}} = &\sum_t A_t(\mathbf{e}_c \otimes \nabla\lambda_{k,t}) : \frac{\partial^2 E_t}{\partial \mathcal{F}^2} : (\mathbf{e}_d \otimes \nabla\lambda_{l,t}) \\ &+ \sum_{ij} l_{ij}\Big[(\mathbf{e}_c \otimes \nabla\lambda_{k,i}) : \frac{\partial^2 E_{\text{edge}}}{\partial \mathcal{F}^2} : (\mathbf{e}_d \otimes \nabla\lambda_{l,i}) + (\mathbf{e}_c \otimes \nabla\lambda_{k,i}) : \frac{\partial^2 E_{\text{edge}}}{\partial \mathcal{F}\partial \mathcal{G}} : (\mathbf{e}_d \otimes \nabla\lambda_{l,j}) + \\ &\qquad (\mathbf{e}_c \otimes \nabla\lambda_{k,j}) : \frac{\partial^2 E_{\text{edge}}}{\partial \mathcal{G}\partial \mathcal{F}} : (\mathbf{e}_d \otimes \nabla\lambda_{l,i}) + (\mathbf{e}_c \otimes \nabla\lambda_{k,j}) : \frac{\partial^2 E_{\text{edge}}}{\partial \mathcal{G}^2} : (\mathbf{e}_d \otimes \nabla\lambda_{l,j})\Big].\end{aligned}$$

The per-triangle contribution Hessian expression is:

$$\begin{aligned}\delta F_a : \frac{\partial^2 E_t}{\partial \mathcal{F}^2} : \delta F_b = &\sum_{i=0}^{1}\left(\frac{\partial \sigma_i}{\partial F} : \delta F_a\right)\frac{\partial^2 s}{\partial \sigma_i^2}\left(\frac{\partial \sigma_i}{\partial F} : \delta F_b\right) + \frac{\partial s}{\partial \sigma_i}\left(\delta F_b : \frac{\partial^2 \sigma_i}{\partial F^2} : \delta F_b\right) \\ &+ 2w_\kappa\left(\mathbf{v}_1^T S_t\left(\frac{\partial \mathbf{v}_1}{\partial F} : \delta F_a\right)\right)\left(\mathbf{v}_1^T S_t\left(\frac{\partial \mathbf{v}_1}{\partial F} : \delta F_b\right)\right) \\ &+ w_\kappa(\mathbf{v}_1^T S_t \mathbf{v}_1)\left(\left(\frac{\partial \mathbf{v}_1}{\partial F} : \delta F_a\right)^T S_t\left(\frac{\partial \mathbf{v}_1}{\partial F} : \delta F_b\right) + \mathbf{v}_1^T S_t\left(\delta F_a : \frac{\partial^2 \mathbf{v}_1}{\partial F^2} : \delta F_b\right)\right),\end{aligned}$$

where we used the fact that $\frac{\partial^2 s}{\partial \sigma_0 \partial \sigma_1} = 0$. The per-edge contribution Hessian blocks are:

$$\begin{aligned}\delta F_a : \frac{\partial^2 E_{\text{edge}}}{\partial \mathcal{F}^2} : \delta F_b = w_\phi &\Bigg[\left(\left(\frac{\partial \mathbf{u}_0}{\partial F}(\mathcal{F}) : \delta F_a\right) \times \mathbf{u}_0(\mathcal{G})\right) \cdot \left(\left(\frac{\partial \mathbf{u}_0}{\partial F}(\mathcal{F}) : \delta F_b\right) \times \mathbf{u}_0(\mathcal{G})\right) + \\ &\qquad \big(\mathbf{u}_0(\mathcal{F}) \times \mathbf{u}_0(\mathcal{G})\big) \cdot \left(\left(\delta F_a : \frac{\partial^2 \mathbf{u}_0}{\partial F^2}(\mathcal{F}) : \delta F_b\right) \times \mathbf{u}_0(\mathcal{G})\right)\Bigg] \\ &+ w_\sigma\left[\left(\frac{\partial \sigma_0}{\partial F}(\mathcal{F}) : \delta F_a\right)\left(\frac{\partial \sigma_0}{\partial F}(\mathcal{F}) : \delta F_b\right) + (\sigma_0(\mathcal{F}) - \sigma_0(\mathcal{G}))\left(\delta F_a : \frac{\partial^2 \sigma_0}{\partial F^2}(\mathcal{F}) : \delta F_b\right)\right], \\ \delta F_a : \frac{\partial^2 E_{\text{edge}}}{\partial \mathcal{F}\partial \mathcal{G}} : \delta F_b = w_\phi &\Bigg[\left(\left(\frac{\partial \mathbf{u}_0}{\partial F}(\mathcal{F}) : \delta F_a\right) \times \mathbf{u}_0(\mathcal{G})\right) \cdot \left(\mathbf{u}_0(\mathcal{F}) \times \left(\frac{\partial \mathbf{u}_0}{\partial F}(\mathcal{G}) : \delta F_b\right)\right) + \\ &\qquad \big(\mathbf{u}_0(\mathcal{F}) \times \mathbf{u}_0(\mathcal{G})\big) \cdot \left(\left(\frac{\partial \mathbf{u}_0}{\partial F}(\mathcal{F}) : \delta F_a\right) \times \left(\frac{\partial \mathbf{u}_0}{\partial F}(\mathcal{G}) : \delta F_b\right)\right)\Bigg] \\ &- w_\sigma\left[\left(\frac{\partial \sigma_0}{\partial F}(\mathcal{F}) : \delta F_a\right)\left(\frac{\partial \sigma_0}{\partial F}(\mathcal{G}) : \delta F_b\right)\right].\end{aligned}$$

The blocks $\frac{\partial^2 E_{\text{edge}}}{\partial \mathcal{G} \partial \mathcal{F}}$ and $\frac{\partial^2 E_{\text{edge}}}{\partial \mathcal{G}^2}$ can be found by exchanging the $\mathcal{F}$ and $\mathcal{G}$ symbols in the above formulas.

We build the sparse Hessian matrix by looping over the triangles and accumulating the contributions to each influenced variables' Hessian entries; for the edge-based regularization terms, this involves an inner loop over the neighboring triangles in the dual IDT.

2.3 Derivatives of the Simulation Energy

The nondimensionalized potential energy minimized in our simulation consists of a physical energy term and a small fictitious target-attraction term:

$$E_{\text{sim}}[\mathbf{\Phi}; \mathcal{S}_0] := \frac{1}{Yh|\hat{\mathcal{S}}_0|} \underbrace{\int_{\mathcal{S}_0} \psi(\nabla \mathbf{\Phi}) \, \mathrm{d}A - p \, \mathrm{Vol}(\mathcal{S}_{\text{tube}})}_{E_{\text{phys}}[\mathbf{\Phi}; \mathcal{S}_0]} + \frac{\epsilon}{\hat{A}\hat{w}^2} \, T(\mathbf{\Phi}).$$

We discretize this energy for a sheet with undeformed vertex positions $\mathbf{X}_i \in \mathbb{R}^2$ and deformed positions $\mathbf{x}_i \in \mathbb{R}^3$ using linear finite elements. Recall that our inflatable is formed by gluing two identical copies of the triangle sheet mesh together. We assign a single undeformed position $\mathbf{X}_i$ to each vertex in this original mesh, which controls both the top and bottom mesh copies. Likewise, each wall vertex in X_{wall} has a single corresponding deformed position since the top and bottom sheet meshes are fused together at these vertices. However, the vertices inside the tube regions are assigned two distinct deformed positions: one for each copy.

The deformation $\mathbf{\Phi}$ and its Jacobian are given by:

$$\mathbf{\Phi} = \sum_i \mathbf{x}_i \lambda_i \quad \Longrightarrow \quad \nabla \mathbf{\Phi} = \sum_i \mathbf{x}_i \otimes \nabla \lambda_i,$$

where now λ_i represents the sum of all piecewise linear shape functions of vertices sharing deformation variable $\mathbf{x}_i$. Plugging this deformation gradient into the continuous simulation energy and substituting our explicit formula for tube volume, we obtain a discrete energy:

$$\begin{aligned}
E_{\text{sim}}(\mathbf{x}, \mathbf{X}) &:= \frac{1}{Yh|\hat{\mathcal{S}}_0|} E_{\text{phys}}(\mathbf{x}, \mathbf{X}) + T(\mathbf{x}), \\
E_{\text{phys}}(\mathbf{x}, \mathbf{X}) &:= \int_{\mathcal{S}_0(\mathbf{X})} \psi \left(\sum_i \mathbf{x}_i \nabla \lambda_i \right) \mathrm{d}A - \frac{p}{6} \sum_{\mathcal{T}_{ijk} \in \mathcal{S}_{\text{tube}}} \det \left(\begin{bmatrix} \mathbf{x}_i & \mathbf{x}_j & \mathbf{x}_k \end{bmatrix} \right), \\
T(\mathbf{x}) &:= \frac{\epsilon}{\hat{A}\hat{w}^2} \sum_{\mathbf{x}_i} \frac{A_{\mathbf{x}_i}}{2} \left\| \mathbf{x}_i - P_{\mathcal{M}}(\mathbf{x}_i) \right\|^2 \mathbf{1}_i^{\text{wall}}.
\end{aligned}$$

This notation collects all positions $\mathbf{X}_i$ and $\mathbf{x}_i$ into the vectors of undeformed and deformed degrees of freedom $\mathbf{X}$ and $\mathbf{x}$, and we use the wall indicator vector $\mathbf{1}_i^{\text{wall}}$, which is 1 if vertex i is a wall vertex and 0 otherwise. To simulate the inflation of a sheet with specified undeformed positions $\mathbf{X}$ using a Newton-based solver, we need gradients and Hessians of E_{phys} and T with respect to the deformed positions $\mathbf{x}$.

2.3.1 Gradients

We differentiate the terms of the simulation energy with respect to component $c \in \{1, 2, 3\}$ of deformed position $\mathbf{x}_i$, using $\mathbf{e}_c$ to denote the c^{th} standard basis vector for $\mathbb{R}^3$:

$$\begin{aligned}
\frac{\partial E_{\text{phys}}}{\partial x_{i,c}}(\mathbf{x}, \mathbf{X}) &= \int_{\mathcal{S}_0(\mathbf{X})} (\mathbf{e}_c \otimes \nabla \lambda_i) : \frac{\partial \psi}{\partial F} (\nabla \mathbf{\Phi}) \, \mathrm{d}A - \left[\frac{p}{6} \sum_{\mathcal{T}_{ijk} \in \mathcal{S}_{\text{tube}}} \mathbf{x}_j \times \mathbf{x}_k \right]_c, \\
\frac{\partial T}{\partial x_{i,c}}(\mathbf{x}) &= A_{\mathbf{x}_i} \left[\mathbf{x}_i - P_{\mathcal{M}}(\mathbf{x}_i) \right]_c \mathbf{1}_i^{\text{wall}},
\end{aligned} \tag{A2}$$

where we have employed the convention that the specified index i constrains the sum over tube triangles to be a sum over triangles containing vertex $\mathbf{x}_i$. We note that the expression $\frac{\partial P_{\mathcal{M}}(\mathbf{x}_i)}{\partial \mathbf{x}_{i,c}}$ does not arise since it is orthogonal to $\mathbf{x}_i - P_{\mathcal{M}}(\mathbf{x}_i)$.

The derivative of the tension field theory elastic energy density, $\frac{\partial \psi}{\partial F}$, is straightforward to compute from its piecewise definition in the main paper and the SVD derivative formulas of Section 2.1.

While it is not immediately obvious from this gradient expression, the gradient of the pressure term with respect to $\mathbf{x}_i$ actually equals the i^{th} vertex's inward-pointing normal scaled by the pressure and the vertex area (one-third the incident triangle area), using the area-weighted average of incident triangle normals as the vertex normal definition. This lends a natural interpretation of the pressure term: it induces an outward-pointing normal force of the appropriate magnitude. This equivalence can be seen by rigidly translating the mesh so that vertex $\mathbf{x}_i$ is at the origin.

2.3.2 Hessians

We now differentiate a second time with respect to component $d \in \{1, 2, 3\}$ of deformed position $\mathbf{x}_j$:

$$\frac{\partial^2 E_{\text{phys}}}{\partial x_{i,c} \partial x_{j,d}}(\mathbf{x}, \mathbf{X}) = \int_{\mathcal{S}_0(\mathbf{X})} (\mathbf{e}_c \otimes \nabla \lambda_i) : \frac{\partial^2 \psi}{\partial F^2} : (\mathbf{e}_d \otimes \nabla \lambda_j) \, \mathrm{d}A + \left[\frac{p}{6} \sum_{\mathcal{T}_{ijk} \in \mathcal{S}_{\text{tube}}} [\mathbf{x}_k]_\times \right]_{c,d},$$

$$\frac{\partial^2 T}{\partial x_{i,c} \partial x_{j,d}}(\mathbf{x}) = A_{\mathbf{x}_i} \delta_{ij} \left[\mathbf{e}_d - \frac{\partial P_{\mathcal{M}}(\mathbf{x}_i)}{\partial x_{i,d}} \right]_c \mathbf{1}_i^{\text{wall}}.$$

We again used the convention that the specified indices i, j constrain the sum over tube triangles to be a sum over triangles containing vertex $\mathbf{x}_i$ and $\mathbf{x}_j$. The symbol δ_{ij} is the Kronecker delta, and $[\mathbf{x}_k]_\times$ is the matrix such that $[\mathbf{x}_k]_\times \mathbf{v} = \mathbf{x}_k \times \mathbf{v}$ for all $\mathbf{v}$. $\frac{\partial^2 \psi}{\partial F^2}$, is also straightforward to compute from its piecewise definition in the main paper and the SVD derivative formulas of Section 2.1.

2.4 Derivatives of the Design Optimization

Our sheet design optimization minimizes the discretized objective function, $J(X_{\text{wall}}) = \tilde{J}(\mathbf{X}(X_{\text{wall}}))$:

$$\tilde{J}(\mathbf{X}) := \frac{1}{\hat{A}\hat{w}^2} T\left(\mathbf{x}^*(\mathbf{X})\right) + \frac{w_b}{|\hat{\mathcal{S}}_0|} \int_{\hat{\mathcal{S}}_0} b(\nabla \varphi) \, \mathrm{d}A + w_{\text{sm}} \mathcal{R}_{\text{sm}}(\mathbf{X}) + w_{\text{c}} \mathcal{R}_{\text{c}}(\mathbf{x}, \mathbf{X}) \tag{A3}$$

where $\mathcal{R}_{\text{sm}}$ and $\mathcal{R}_{\text{c}}$ are the nondimensionalized discrete smoothness and compression penalty terms (see Section 1.3), $\varphi : \hat{\mathcal{S}}_0 \to \mathcal{S}_0(\mathbf{X})$ is the piecewise linear mapping of the initial sheet mesh $\hat{S}_0$ (with vertices $\hat{\mathbf{X}}_i$) induced by the design alteration prescribed by X_{wall}, and

$$\mathbf{x}^*(\mathbf{X}) := \operatorname*{argmin}_{\mathbf{x}} E_{\text{sim}}(\mathbf{x}, \mathbf{X})$$

are the deformed vertex positions of the inflated equilibrium.

We discretize the smoothness term as a sum over *non-endpoint* vertices of curves in γ, which we collect in index set $\mathcal{V}_\gamma$. We introduce the current and initial Voronoi lengths l_i and $\hat{l}_i$ associated with vertex $i \in \mathcal{V}_\gamma$ and define $\boldsymbol{e}^i$ and $\boldsymbol{e}^{i-1}$ (and their hatted counterparts in the initial mesh) to be the edge vectors of the two incident edges. Then our discrete smoothness term is:

$$\mathcal{R}_{\text{sm}}(\mathbf{X}) := \frac{\hat{w}}{2} \sum_{i \in \mathcal{V}_\gamma} \left(\frac{1}{l_i} \angle(\boldsymbol{e}^{i-1}, \boldsymbol{e}^i) - \frac{1}{\hat{l}_i} \angle(\hat{\boldsymbol{e}}^{i-1}, \hat{\boldsymbol{e}}^i) \right)_+^2 \hat{l}_i + \left(\frac{\|\boldsymbol{e}^{i-1}\|}{\|\hat{\boldsymbol{e}}^{i-1}\|} - \frac{\|\boldsymbol{e}^i\|}{\|\hat{\boldsymbol{e}}^i\|} \right)^2 \frac{1}{\hat{l}_i},$$

where $\angle(\mathbf{a}, \mathbf{b})$ measures the unsigned angle between two vectors.

The compression penalty term measures the difference between the equilibrium elastic energies evaluated with and without tension field theory relaxation:

$$\mathcal{R}_{\mathrm{c}}(\mathbf{x}, \mathbf{X}) := \frac{1}{Yh|\hat{\mathcal{S}}_0|} \left(E_{\mathrm{Full}}(\mathbf{x}, \mathbf{X}) - E_{\mathrm{TFT}}(\mathbf{x}, \mathbf{X}) \right).$$

We will for now differentiate $\tilde{J}$, treating every undeformed position $\mathbf{X}_i$ as an independent variable of the design optimization. Later in Section 2.4.2, we will show how to compute derivatives when $\mathbf{X}_i$ are set by harmonically interpolating the wall positions. Thus, φ can be written explicitly as:

$$\varphi = \sum_i \mathbf{X}_i \hat{\lambda}_i, \qquad \nabla\varphi = \sum_i \mathbf{X}_i \otimes \nabla\hat{\lambda}_i,$$

where $\hat{\lambda}$ are the piecewise linear FEM shape functions for $\hat{\mathcal{S}}_0$.

2.4.1 Derivatives With Respect to All Mesh Vertices

Differentiating with respect to the undeformed vertex position $\mathbf{X}_i$:

$$\frac{\partial \tilde{J}}{\partial \mathbf{X}_i} = \left(\frac{1}{\hat{A}\hat{w}^2} \frac{\partial T}{\partial \mathbf{x}} + w_{\mathrm{c}} \frac{\partial \mathcal{R}_{\mathrm{c}}}{\partial \mathbf{x}} \right) \frac{\partial \mathbf{x}^*}{\partial \mathbf{X}_i} + \int_{\hat{\mathcal{S}}_0} b'(\nabla\varphi) \nabla\hat{\lambda}_i \, \mathrm{d}A + w_{\mathrm{c}} \frac{\partial \mathcal{R}_{\mathrm{c}}}{\partial \mathbf{X}} + w_{\mathrm{sm}} \frac{\partial \mathcal{R}_{\mathrm{sm}}}{\partial \mathbf{X}}$$

The derivative of the barrier function b' is straightforward to compute using the singular value derivative formulas from Section 2.1 (either by conceptually padding the 2×2 matrix $\nabla\varphi$ to 2×3 or by truncating the derivative formulas). The derivatives $\frac{\partial \mathcal{R}_{\mathrm{c}}}{\partial \mathbf{x}}$ and $\frac{\partial T}{\partial \mathbf{x}}$ can be adapted from (A2) (ignoring the pressure term and enabling/disabling the tension field theory relaxation). Likewise, $\frac{\partial \mathcal{R}_{\mathrm{c}}}{\partial \mathbf{X}}$ can be obtained with a simpler version of the calculation (A7) below. The derivative $\frac{\partial \mathcal{R}_{\mathrm{sm}}}{\partial \mathbf{X}}$ can be calculated with expressions for the derivative of a vector's length and angle, which can be obtained with simple geometric arguments. For example $\frac{\partial \angle(\mathbf{a},\mathbf{b})}{\partial \mathbf{a}} = \frac{\mathbf{a}^\perp}{\|\mathbf{a}\|^2}$, where $\perp$ rotates $\mathbf{a}$ away from $\mathbf{b}$ by $\frac{\pi}{2}$. The only challenging part is $\frac{\partial \mathbf{x}^*}{\partial \mathbf{X}_i}$.

Away from the discontinuities discussed in the *Navigating local minima* paragraph of the main paper, we can determine how the equilibrium deformation positions evolve as undeformed positions $\mathbf{X}$ change by differentiating both sides of the equation characterizing the equilibrium:

$$\frac{\partial E_{\mathrm{sim}}}{\partial \mathbf{x}}\big(\mathbf{x}^*(\mathbf{X}), \mathbf{X}\big) = 0 \quad \Longrightarrow \quad \frac{\partial^2 E_{\mathrm{sim}}}{\partial \mathbf{x}^2} \frac{\partial \mathbf{x}^*}{\partial \mathbf{X}} + \frac{\partial^2 E_{\mathrm{sim}}}{\partial \mathbf{x} \partial \mathbf{X}} = 0$$

$$\Longrightarrow \quad \frac{\partial \mathbf{x}^*}{\partial \mathbf{X}} = -\left[\frac{\partial^2 E_{\mathrm{sim}}}{\partial \mathbf{x}^2} \right]^{-1} \frac{\partial^2 E_{\mathrm{sim}}}{\partial \mathbf{x} \partial \mathbf{X}}. \tag{A4}$$

Since the mesh consists of thousands of vertices (over $250K$ for the inflatable face mask example), we must employ the adjoint method to make the gradient calculation tractable: we solve for the adjoint state,

$$\mathbf{p} := \left[\frac{\partial^2 E_{\mathrm{sim}}}{\partial \mathbf{x}^2} \right]^{-1} \left(\frac{1}{\hat{A}\hat{w}^2} \frac{\partial T}{\partial \mathbf{x}} + w_{\mathrm{c}} \frac{\partial \mathcal{R}_{\mathrm{c}}}{\partial \mathbf{x}} \right), \tag{A5}$$

and then compute the full gradient as:

$$\boxed{\frac{\partial \tilde{J}}{\partial \mathbf{X}_i} = -\mathbf{p}^T \frac{\partial^2 E_{\mathrm{sim}}}{\partial \mathbf{x} \partial \mathbf{X}_i} + \int_{\hat{\mathcal{S}}_0} b'(\nabla\varphi) \nabla\hat{\lambda}_i \, \mathrm{d}A + + w_{\mathrm{c}} \frac{\partial \mathcal{R}_{\mathrm{c}}}{\partial \mathbf{X}} + w_{\mathrm{sm}} \frac{\partial \mathcal{R}_{\mathrm{sm}}}{\partial \mathbf{X}}.} \tag{A6}$$

Finally, we compute the term $\frac{\partial^2 E_{\mathrm{sim}}}{\partial \mathbf{x} \partial \mathbf{X}}$ by changing variables to express the simulation energy gradient $\frac{\partial E_{\mathrm{sim}}}{\partial \mathbf{x}}$ as an integral over the fixed initial sheet $\hat{\mathcal{S}}_0$ that can be directly differentiated: this change of variables

involves transferring quantities from $\mathbf{X}_i$ of $\mathcal{S}_0$ to the corresponding vertex $\hat{\mathbf{X}}_i$ of $\hat{\mathcal{S}}_0$. Only the elastic energy term of E_{sim} depends on $\mathbf{X}$, so we have only one integral to differentiate:

$$\frac{\partial^2 E_{\text{sim}}}{\partial x_{i,c} \partial X_{j,d}} = \frac{\partial}{\partial X_{j,d}} \int_{\hat{\mathcal{S}}_0} \left((\mathbf{e}_c \otimes \nabla \hat{\lambda}_i) \nabla \varphi^{-1} \right) : \frac{\partial \psi}{\partial F} \left(\hat{\nabla} \mathbf{\Phi} \nabla \varphi^{-1} \right) \det(\nabla \varphi) \, \mathrm{d}A, \tag{A7}$$

where $\hat{\nabla} \mathbf{\Phi}$ is shorthand for $\sum_k \mathbf{x}_k \otimes \nabla \hat{\lambda}_k$, and we used the gradient transformation formula $\nabla \lambda_k = \nabla \varphi^{-T} \nabla \hat{\lambda}_k$. Now that the integration domain is fixed, we can differentiate the integral using the following formulas:

$$\begin{aligned} \frac{\partial \nabla \varphi}{\partial X_{j,d}} = \mathbf{e}_d \otimes \nabla \hat{\lambda}_j \quad \Longrightarrow \quad & \frac{\partial \nabla \varphi^{-1}}{\partial X_{j,d}} = -\nabla \varphi^{-1} \left(\mathbf{e}_d \otimes \nabla \hat{\lambda}_j \right) \nabla \varphi^{-1}, \\ & \frac{\partial \det(\nabla \varphi)}{\partial X_{j,d}} = \det(\nabla \varphi) \left[\nabla \varphi^{-T} : \left(\mathbf{e}_d \otimes \nabla \hat{\lambda}_j \right) \right] = \det(\nabla \varphi) \left(\mathbf{e}_d \cdot \nabla \varphi^{-T} \nabla \hat{\lambda}_j \right). \end{aligned}$$

$$\begin{aligned} \frac{\partial^2 E_{\text{sim}}}{\partial x_{i,c} \partial X_{j,d}} = & - \int_{\hat{\mathcal{S}}_0} \left((\mathbf{e}_c \otimes \nabla \hat{\lambda}_i) \nabla \varphi^{-1} (\mathbf{e}_d \otimes \nabla \hat{\lambda}_j) \nabla \varphi^{-1} \right) : \frac{\partial \psi}{\partial F} \left(\hat{\nabla} \mathbf{\Phi} \nabla \varphi^{-1} \right) \det(\nabla \varphi) \, \mathrm{d}A \\ & - \int_{\hat{\mathcal{S}}_0} \left((\mathbf{e}_c \otimes \nabla \hat{\lambda}_i) \nabla \varphi^{-1} \right) : \frac{\partial^2 \psi}{\partial F^2} \left(\hat{\nabla} \mathbf{\Phi} \nabla \varphi^{-1} \right) : \left(\hat{\nabla} \mathbf{\Phi} \nabla \varphi^{-1} (\mathbf{e}_d \otimes \nabla \hat{\lambda}_j) \nabla \varphi^{-1} \right) \det(\nabla \varphi) \, \mathrm{d}A \\ & + \int_{\hat{\mathcal{S}}_0} \left((\mathbf{e}_c \otimes \nabla \hat{\lambda}_i) \nabla \varphi^{-1} \right) : \frac{\partial \psi}{\partial F} \left(\hat{\nabla} \mathbf{\Phi} \nabla \varphi^{-1} \right) \det(\nabla \varphi) \left(\mathbf{e}_d \cdot \nabla \varphi^{-T} \nabla \hat{\lambda}_j \right) \, \mathrm{d}A. \end{aligned}$$

Changing variables back to an integral over the current sheet, we obtain our final expression:

$$\begin{aligned} \frac{\partial^2 E_{\text{sim}}}{\partial x_{i,c} \partial X_{j,d}} = & - \int_{\mathcal{S}_0} \left((\mathbf{e}_c \otimes \nabla \lambda_i)(\mathbf{e}_d \otimes \nabla \lambda_j) \right) : \frac{\partial \psi}{\partial F} - (\mathbf{e}_c \otimes \nabla \lambda_i) : \frac{\partial^2 \psi}{\partial F^2} : \left((\nabla \mathbf{\Phi})(\mathbf{e}_d \otimes \nabla \lambda_j) \right) \\ & + (\mathbf{e}_c \otimes \nabla \lambda_i) : \frac{\partial \psi}{\partial F} \, \mathbf{e}_d \cdot \nabla \lambda_j \, \mathrm{d}A. \end{aligned} \tag{A8}$$

where we suppressed the energy density's argument $(\nabla \mathbf{\Phi})$ for brevity. We also can obtain a simplified expression for the term of $\tilde{J}$'s gradient in which this expression appears:

$$-\mathbf{p}^T \frac{\partial^2 E_{\text{sim}}}{\partial \mathbf{x} \partial X_{j,d}} = \int_{\mathcal{S}_0} \left(\nabla \mathbf{p} (\mathbf{e}_d \otimes \nabla \lambda_j) \right) : \frac{\partial \psi}{\partial F} + \nabla \mathbf{p} : \frac{\partial^2 \psi}{\partial F^2} : \left((\nabla \mathbf{\Phi})(\mathbf{e}_d \otimes \nabla \lambda_j) \right) - \nabla \mathbf{p} : \frac{\partial \psi}{\partial F} \, \mathbf{e}_d \cdot \nabla \lambda_j \, \mathrm{d}A.$$

(An abuse of notation was made to simplify the right-hand side of this equation, using the shorthand $\nabla \mathbf{p}$ for $\sum_k \mathbf{p}_k \otimes \nabla \lambda_k$, i.e., not distinguishing between the FEM coefficient vector $\mathbf{p}$ and its corresponding piecewise linear vector field.)

We note that computing higher-order sensitivity information $\frac{\partial^2 \tilde{J}}{\partial \mathbf{X}^2}$ would require taking a third derivative of ψ, which is in fact only C^1 (with non-smooth transitions between compression and tension). We opt instead to use a BFGS Hessian approximation.

2.4.2 Derivatives With Respect to the Wall Vertices

The gradient formula (A6) is for differentiating the design objective with respect to every mesh vertex. However, our design optimization only optimizes X_{wall}, the undeformed positions of the wall vertices, and positions the rest of $\mathbf{X}$ using a harmonic interpolation:

$$\mathbf{X}(X_{\text{wall}}) = \begin{bmatrix} X_{\text{interior}} \\ X_{\text{wall}} \end{bmatrix} = - \begin{bmatrix} \hat{L}_{ii}^{-1} \hat{L}_{iw} \\ I \end{bmatrix} X_{\text{wall}}.$$

We obtain the necessary gradient formula with the chain rule:

$$\frac{\partial J}{\partial X_{\text{wall}}} = \frac{\partial \tilde{J}(\mathbf{X}(X_{\text{wall}}))}{\partial X_{\text{wall}}} = \frac{\partial \tilde{J}}{\partial \mathbf{X}} \frac{\partial \mathbf{X}(X_{\text{wall}})}{\partial X_{\text{wall}}} = -\frac{\partial \tilde{J}}{\partial \mathbf{X}} \begin{bmatrix} \hat{L}_{ii}^{-1} \hat{L}_{iw} \\ I \end{bmatrix}, \tag{A9}$$

which we can evaluate with a single backsolve of the sparse, constant matrix $\hat{L}_{ii}$ and a sparse matrix-vector product.

References

Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. 2002. Least Squares Conformal Maps for Automatic Texture Atlas Generation. *ACM Trans. Graph.* 21, 3 (July 2002), 362–371. https://doi.org/10.1145/566654.566590

Jonathan Richard Shewchuk. 1996. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, Ming C. Lin and Dinesh Manocha (Eds.). Lecture Notes in Computer Science, Vol. 1148. Springer-Verlag, 203–222.

Mélina Skouras, Bernhard Thomaszewski, Peter Kaufmann, Akash Garg, Bernd Bickel, Eitan Grinspun, and Markus Gross. 2014. Designing Inflatable Structures. *ACM Trans. Graph.* 33, 4, Article 63 (July 2014), 10 pages. https://doi.org/10.1145/2601097.2601166

Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. 1997. Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound-Constrained Optimization. *ACM Trans. Math. Softw.* 23, 4 (Dec. 1997), 550–560. https://doi.org/10.1145/279232.279236