WRAPD: Weighted Rotation-aware ADMM for Parameterization and Deformation

GEORGE E. BROWN, University of Minnesota
RAHUL NARAIN, Indian Institute of Technology Delhi

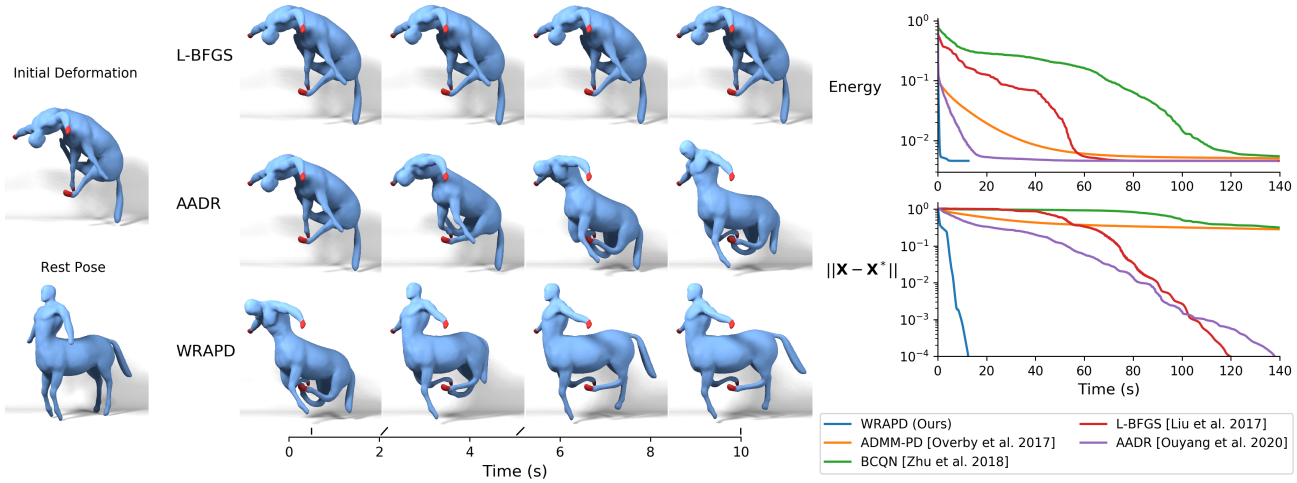

Fig. 1. Quasi-static simulation of a neo-Hookean centaur with 98K tetrahedra and 26K vertices, subject to pin constraints. *Left*: Initial and rest shapes. *Middle*: Selected frames from the first ten seconds of simulation comparing our algorithm to competitors. *Top right*: Objective value vs. time. *Bottom right*: Norm of position error relative to optimal state vs. time.

Local-global solvers such as ADMM for elastic simulation and geometry optimization struggle to resolve large rotations such as bending and twisting modes, and large distortions in the presence of barrier energies. We propose two improvements to address these challenges. First, we introduce a novel local-global splitting based on the polar decomposition that separates the geometric nonlinearity of rotations from the material nonlinearity of the deformation energy. The resulting ADMM-based algorithm is a combination of an L-BFGS solve in the global step and proximal updates of element stretches in the local step. We also introduce a novel method for dynamic reweighting that is used to adjust element weights at runtime for improved convergence. With both improved rotation handling and element weighting, our algorithm is considerably faster than state-of-the-art approaches for quasi-static simulations. It is also much faster at making early progress in parameterization problems, making it valuable as an initializer to jump-start second-order algorithms.

CCS Concepts: • **Computing methodologies** → **Physical simulation**.

Additional Key Words and Phrases: quasi-statics, parameterization, geometric nonlinearity, rotation-aware, reweighting, optimization, ADMM

Authors' addresses: George E. Brown, University of Minnesota, brow2327@umn.edu; Rahul Narain, Indian Institute of Technology Delhi, narain@cse.iitd.ac.in.

0730-0301/2021/8-ART82 $15.00
https://doi.org/10.1145/3450626.3459942

ACM Reference Format:
George E. Brown and Rahul Narain. 2021. WRAPD: Weighted Rotation-aware ADMM for Parameterization and Deformation. *ACM Trans. Graph.* 40, 4, Article 82 (August 2021), 14 pages. https://doi.org/10.1145/3450626.3459942

1 INTRODUCTION

From quasi-static and dynamic simulation of elastic bodies in physics-based animation to interactive shape manipulation and mesh parameterization in geometry processing, recent work has converged on minimization of deformation energies as a central task in computer graphics. The deformation energy model takes many different forms in different applications, from physically-based hyperelastic models such as the neo-Hookean model, to geomerically motivated distortion metrics such as the symmetric Dirichlet energy. These energies can be highly nonlinear with respect to stretching deformations, but have the key property that they are invariant to rigid transformations.

Many popular techniques for minimizing such energies use a local-global approach [Sorkine and Alexa 2007; Bouaziz et al. 2012; Overby et al. 2017; Liu et al. 2017; Peng et al. 2018; Zhang et al. 2019; Ouyang et al. 2020], in which chosen per-element quantities are introduced as additional optimization variables. Optimization is then performed in an alternating fashion, with a local step acting on the per-element local variables and a global step updating the vertex positions. We focus on ADMM-PD [Overby et al. 2017], a local-global algorithm that has been shown to be effective for nonlinear energies

including those with infinite barrier terms. In contrast to Newton-type methods that use Hessian information, local-global techniques such as ADMM-PD have computationally cheap iterations and often make rapid progress early on. However, beyond the early iterations such solvers typically suffer from slow convergence. In this work, we identify and address two major reasons for the slow convergence of local-global solvers like ADMM-PD.

First, in all previous work, the local variable for each element is chosen to be the Jacobian of the deformation map, i.e. the deformation gradient, or a similar quantity that is invariant to translation. This choice factors out the translation invariance of the deformation energy from the local step, allowing the algorithm to take large steps in resolving translations in the global step. However, when the problem requires large rotations from the initial guess to the optimal solution, such approaches converge slowly since the rotation of each element is updated gradually over multiple global and local iterations. We argue that to fully exploit the geometrical properties of deformation energies, the local variables should be chosen to be invariant to both translations and rotations. In particular, we use the polar decomposition to decompose the deformation gradient and extract a rotation-invariant stretch tensor. Using this choice, our first contribution is an ADMM-based deformation solver that treats only the per-element stretch tensors as the local variables, and solves for vertex positions as global variables. Thus, the local step deals only with the *material nonlinearity* of the deformation energy with respect to principal strains in reference space. The task of the global step is to recover vertex positions in order to best match the estimated element stretches, accounting only for the *geometric nonlinearity* of rotations.

Second, some local-global methods like ADMM-PD and Liu et al. [2017] require the specification of per-element weights. These weights have a dramatic impact on convergence and stability, but good choices of their values depend on the curvature of the objective at the optimal configuration and are therefore difficult to select in advance. When weights are too large, the rate of progress is greatly inhibited, but when too small ADMM-PD can oscillate and fail to converge to the optimum. As our second contribution, we introduce a novel strategy for locally updating element weights. Our reweighting method seamlessly updates element weights as the optimization proceeds, using per-element tangent stiffness matrices to account for local curvature at the current configuration.

Combining our rotation-awareness and local reweighting contributions leads to our new algorithm called WRAPD. We evaluate our proposed algorithm on a large variety of examples from geometry optimization and parameterization. With reweighting, WRAPD requires no weight tuning and dynamically adjusts weights in response to changing element stiffnesses. With rotation awareness, WRAPD rapidly minimizes distortion in problems dominated by rotational geometric nonlinearities. Combined, these features dramatically improve the convergence of ADMM-based optimization. We demonstrate that our method is considerably faster than state-of-the-art algorithms for solving quasi-static deformation problems. As a standalone algorithm WRAPD is also competitive with existing approaches for parameterization. However, even faster convergence is obtained when WRAPD is used as an initializer for other parameterization algorithms. This hybrid approach outperforms existing techniques for obtaining quick, early solutions to parameterization problems.

2 RELATED WORK

Recent work in both physics-based animation and geometry processing has been converging towards optimization formulations, in which one seeks a configuration with minimum deformation energy subject to other influences such as inertia or user constraints.

In the field of geometry optimization, early work focused on simple energies that do not exhibit material nonlinearity. The ARAP energy model has been widely used since its introduction to the community by Sorkine and Alexa [2007]. It is a simple but effective model for responding to global deformations while seeking to preserve localized shape. In subsequent work Liu et al. [2008] used ARAP in their local-global solver to compute local transformations for triangles during mesh parameterization. Bouaziz et al. [2012] extended and formalized these ideas in their unified framework for geometric optimization based on shape constraints.

Much recent work has focused on optimizing nonlinear deformation energies. Kovalsky et al. [2016] proposed the accelerated quadratic proxy (AQP) algorithm, which uses local quadratic proxies instead of the exact energy Hessian. The cost of their method scales better with mesh resolution than alternative quasi-Newton approaches. Rabinovich et al. [2017] also use simpler proxy energies with an iterative reweighting scheme in their SLIM method. Claici et al. [2017] account for the invariance of deformation energy with respect to rigid motions via an isometry-aware preconditioner for steepest descent. Another technique is composite majorization (CM) by Shtengel et al. [2017], which solves strictly 2D convex-concave decomposed problems using a convex majorizer solver. More recently, Zhu et al. [2018] advocated for their second-order convergent blended cured quasi-Newton (BCQN) algorithm that forms quadratic energy proxies by adaptively blending Sobolev gradient and L-BFGS descent. Their method is further improved by a barrier-aware line-search and improved termination criteria.

In physics-based animation, Martin et al. [2011] pointed out that backward Euler time integration can be formulated as an energy optimization problem. Based on this formulation, Gast et al. [2015] presented an optimization algorithm that enabled for faster and more robust simulation of stiff systems at larger time steps than is possible using traditional Newton-based methods. Bouaziz et al. [2014] introduced projective dynamics, a fast and efficient local-global algorithm for solving problems with linear constitutive models and soft constraints. Subsequent work has applied projective dynamics to fluid simulation [Weiler et al. 2016] and real-time skinning [Komaritzan and Botsch 2018, 2019], and has combined it with model reduction [Brandt et al. 2018]. Liu et al. [2017] showed that projective dynamics can be interpreted as a quasi-Newton method, and designed a method using L-BFGS for real-time simulation of hyperelastic materials. Overby et al. [2017] reinterpreted projective dynamics in terms of ADMM optimization algorithm, permitting the use of nonlinear energies and dynamic hard constraints.

Peng et al. [2018] introduced Anderson acceleration to the graphics community and used it to accelerate the convergence of many

deformation optimization algorithms, including projective dynamics. Zhang et al. [2019] used Anderson acceleration to improve the convergence of ADMM applied to both simulation and geometry processing tasks. Recently, Fu et al. [2020] observed that Douglas-Rachford splitting is equivalent to ADMM, and used this fact to formulate an efficient Anderson-accelerated Douglas-Rachford (AADR) method for convex problems. Ouyang et al. [2020] expanded the scope of AADR to handle nonconvex problems too, and they demonstrated that it accelerates more reliably compared to Anderson-accelerated ADMM.

3 BACKGROUND

We consider a generic geometry optimization problem on a discretized model with n nodes and m elements (springs, triangles, and/or tetrahedra) in $\mathbb{R}^d$ where $d = 2, 3$. The positions of the nodes are collected into a matrix $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^T \in \mathbb{R}^{n \times d}$. For the ith element, its deformation gradient (the Jacobian of the deformation map) is a linear function of the node positions. Consequently, it can be expressed in the form $\mathbf{F}_i = (\mathbf{D}_i\mathbf{X})^T$ for some matrix $\mathbf{D}_i \in \mathbb{R}^{k \times n}$, where $k = 1, 2, 3$ for springs, triangles, and tetrahedra respectively. Finally, we are given a deformation energy function Ψ that determines the total energy of the model as $E = \sum_{i=1}^{m} V_i \Psi(\mathbf{F}_i)$ where V_i is the measure (area or volume) of the ith element. Our goal is to find the node positions $\mathbf{X}$ to minimize E. User-specified controls may also be added to the problem, in the form of linear constraints and/or quadratic penalties; we omit these for simplicity in our exposition.

Since the deformation energy of each element depends only on some low-dimensional local coordinates describing its deformation, local-global solvers such as projective dynamics and ADMM-PD exploit this property by treating these local coordinates as new optimization variables (denoted $\mathbf{P}_i$ or $\mathbf{Z}_i$). In the "local step" we minimize the deformation energy in the space of these local coordinates. In the "global step" we reconstruct node positions such that the element deformations best match the optimized local coordinates.

The choice of local coordinates has a significant impact on the convergence of local-global optimization [Bouaziz et al. 2012, Fig. 10]. In existing work, the local coordinates are chosen to be linear and translation-invariant in $\mathbf{X}$, typically the deformation gradient. Overby et al. [2017] use this choice for local coordinates. We will next summarize their main results and solver using our notation.

Using the shorthand $\{\mathbf{Z}_i\}$ to refer to the collection of all local variables $\mathbf{Z}_1, \ldots, \mathbf{Z}_m$, the problem is expressed as

$$\min_{\mathbf{X}, \{\mathbf{Z}_i\}} E(\{\mathbf{Z}_i\}) \tag{1a}$$

$$\text{s.t. } \mathbf{C}_i(\mathbf{X}, \mathbf{Z}_i) = \mathbf{0} \quad \forall i = 1, \ldots, m, \tag{1b}$$

where the linear coupling constraint $\mathbf{C}_i$ for the ith element is

$$\mathbf{C}_i(\mathbf{X}, \mathbf{Z}_i) := (\mathbf{D}_i\mathbf{X})^T - \mathbf{Z}_i = \mathbf{0}. \tag{2}$$

To find a solution to the problem in (1) we begin by constructing an associated augmented Lagrangian function. For this we must introduce an additional set of m dual variables $\{\mathbf{U}_i\}$. Like Overby and colleagues, in our construction we will hold fixed the ADMM variable $\rho = 1$ and choose per-element weights w_i to improve convergence. With these choices the augmented Lagrangian for our problem is

$$L(\mathbf{X}, \{\mathbf{Z}_i\}; \{\mathbf{U}_i\}) = E(\{\mathbf{Z}_i\}) + \sum_{i=1}^{m} \frac{w_i^2}{2} \left\| \mathbf{C}_i(\mathbf{X}, \mathbf{Z}_i) + \mathbf{U}_i \right\|_F^2, \tag{3}$$

where w_i is the scalar weight for the ith element. Following the ADMM algorithm this yields the iteration

$$\mathbf{X} \leftarrow \arg\min_{\mathbf{X}} L(\mathbf{X}, \{\mathbf{Z}_i\}; \{\mathbf{U}_i\}), \tag{4a}$$

$$\mathbf{Z}_i \leftarrow \arg\min_{\mathbf{Z}_i} L(\mathbf{X}, \mathbf{Z}_i; \mathbf{U}_i) \quad \forall i = 1, \ldots, m, \tag{4b}$$

$$\mathbf{U}_i \leftarrow \mathbf{U}_i + \mathbf{C}_i(\mathbf{X}, \mathbf{Z}_i) \quad \forall i = 1, \ldots, m. \tag{4c}$$

Element translations are determined by the global step, which acts on $\mathbf{X}$ by solving (4a). For this problem formulation, with static weights and linear coupling constraints, the global step is tantamount to a single linear solve $\mathbf{AX} = \mathbf{B}$ with a constant global matrix $\mathbf{A}$ that can be prefactored [Overby et al. 2017]. The matrix $\mathbf{A}$ and right-hand side $\mathbf{B}$ can be computed by summing contributions from all elements according to

$$\mathbf{A} = \sum_{i=1}^{m} w_i^2 \mathbf{D}_i^T \mathbf{D}_i, \tag{5a}$$

$$\mathbf{B} = \sum_{i=1}^{m} w_i^2 \mathbf{D}_i^T (\mathbf{Z}_i - \mathbf{U}_i). \tag{5b}$$

Then the local step solves (4b) in parallel over all elements. Specifically, each element's $\mathbf{Z}_i$ is updated by solving

$$\arg\min_{\mathbf{Z}_i} \Psi(\mathbf{Z}_i) + \frac{w_i^2}{2} \left\| (\mathbf{D}_i\mathbf{X})^T - \mathbf{Z}_i + \mathbf{U}_i \right\|_F^2. \tag{6}$$

As pointed out by Overby et al. [2017], this is a proximal operator of Ψ at $(\mathbf{D}_i\mathbf{X})^T + \mathbf{U}_i$, and for rotation-invariant Ψ it can be performed on just the singular values of $\mathbf{Z}_i$. Specifically, we compute the SVD of $(\mathbf{D}_i\mathbf{X})^T + \mathbf{U}_i$ and perform the above minimization on only the singular values, while keeping the same singular vectors. The minimization is performed using Newton's method with a flip-preventing line search. Finally, the dual variables $\mathbf{U}_i$ are updated according to (4c).

This formulation has two major limitations. First, the deformation gradient is a rotation-dependent measure. This means local coordinates based on $\mathbf{F}_i$ change whenever an element is re-oriented in space, even when it is not at all stretched within its own local basis. Consequently, algorithms that solve (1) can be prohibitively slow to make progress whenever the solution requires a large number of element rotations. In Section 7 we provide many practical examples where convergence is slowed when the local step is tasked with handling even mild amounts of element rotations. Second, ADMM algorithms designed to solve (1) require a prescription for per-element weights. These are notoriously difficult to define, and if poorly chosen the algorithm may be slow to make progress or even fail to converge. We address both of these problems and present solutions in Sections 4 and 5, respectively.

4 ROTATION-AWARE ADMM

Our key idea here is to choose local coordinates that are invariant to rotations as well as translations. In particular, we factorize the

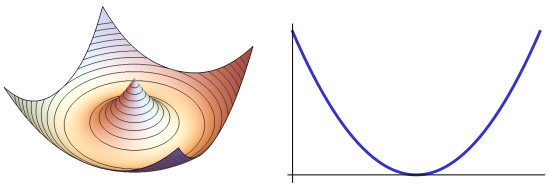

Fig. 2. An illustration of the benefits of rotation-invariant local coordinates. For a unit length spring between vertices a and b, the analogues of $\mathbf{F}_i$ and $\mathbf{S}_i$ are the edge vector $\mathbf{e} = \mathbf{x}_a - \mathbf{x}_b$ and its magnitude $\|\mathbf{e}\|$ respectively. The spring energy as a function of $\mathbf{e}$ (left) is nonconvex and has a codimension-1 manifold of minima, while as a function of $\|\mathbf{e}\|$ (right) it is strictly convex.

deformation gradient of each element using the polar decomposition $\mathbf{F}_i = \mathbf{R}_i\mathbf{S}_i$, then we use only the symmetric factor $\mathbf{S}_i$ as the local variable. As a function of $\mathbf{S}_i$, Ψ is much more well-behaved (Fig. 2). Further, this formulation cleanly separates the geometric nonlinearity of rotations from the material nonlinearity of the deformation energy. The price of this modification is that the objective in the global step becomes nonlinear. Nevertheless, it remains a smoothly defined function with none of the infinite barrier terms that make energies like neo-Hookean and symmetric Dirichlet challenging to minimize. As we shall see, it can be efficiently solved by a quasi-Newton algorithm like L-BFGS while still taking advantage the prefactored matrix in (5a).

For brevity we begin by introducing notation that will be used in the description of our algorithm. We will use $\mathrm{sym}(\mathbf{A})$ to denote a function that takes as input a matrix $\mathbf{A}$ and returns its symmetric factor $\mathbf{S}$ computed using the polar decomposition $\mathbf{A} = \mathbf{RS}$. (Note that in general $\mathrm{sym}(\mathbf{A}) \neq \frac{1}{2}(\mathbf{A} + \mathbf{A}^T)$.) We will also continue to use $\mathbf{Z}_i$ to refer to the local coordinates of the ith element, and $\{\mathbf{Z}_i\}$ for the set of all local coordinates in the system. This allows us to concretely discuss the stages of our algorithm in terms of its effect on just two state variables: $\mathbf{X}$ and $\{\mathbf{Z}_i\}$. Now we will reformulate (1) in terms of this alternate set of optimization variables. Our new problem formulation is

$$\min_{\mathbf{X},\{\mathbf{Z}_i\}} E(\{\mathbf{Z}_i\}) \tag{7a}$$

$$\text{s.t. } \{\mathbf{C}_i^{\mathrm{sym}}(\mathbf{X}, \mathbf{Z}_i)\} = \mathbf{0}, \tag{7b}$$

where the symmetric coupling constraint for the ith element is

$$\mathbf{C}_i^{\mathrm{sym}}(\mathbf{X}, \mathbf{Z}_i) := \mathrm{sym}\left((\mathbf{D}_i\mathbf{X})^T\right) - \mathbf{Z}_i = \mathbf{0}. \tag{8}$$

This formulation is in fact equivalent to (1). To see this, recall that for rotationally-invariant deformation energies $\Psi(\mathbf{F}_i) = \Psi(\mathrm{sym}(\mathbf{F}_i))$. The additional constraint (7b) ensures that the optimization variables $\mathbf{Z}_i$ always correspond to the symmetric part of the deformation gradient.

We solve (7) using ADMM with the variables $\mathbf{X}$ and $\{\mathbf{Z}_i\}$. The problem deviates from the traditional ADMM framework, in which the coupling between the split variables is expected to be a linear constraint. In our case, the coupling constraint in (8) is nonlinear. Nevertheless, ADMM has been applied successfully with nonlinear coupling constraints in existing work [Benning et al. 2016], so we proceed with an ADMM approach as well.

The augmented Lagrangian for this problem is

$$L(\mathbf{X}, \{\mathbf{Z}_i\}; \{\mathbf{U}_i\}) = E(\{\mathbf{Z}_i\}) + \sum_{i=1}^{m} \frac{w_i^2}{2} \left\|\mathbf{C}_i^{\mathrm{sym}}(\mathbf{X}, \mathbf{Z}_i) + \mathbf{U}_i\right\|_F^2, \tag{9}$$

where $\mathbf{U}_i$ is the Lagrange multiplier estimate for $\mathbf{C}_i^{\mathrm{sym}}$. Following the ADMM algorithm, we have the iteration

$$\mathbf{X} \leftarrow \arg\min_{\mathbf{X}} L(\mathbf{X}, \{\mathbf{Z}_i\}; \{\mathbf{U}_i\}), \tag{10a}$$

$$\mathbf{Z}_i \leftarrow \arg\min_{\mathbf{Z}_i} L(\mathbf{X}, \mathbf{Z}_i; \mathbf{U}_i) \quad \forall i = 1, \ldots, m, \tag{10b}$$

$$\mathbf{U}_i \leftarrow \mathbf{U}_i + \mathbf{C}_i^{\mathrm{sym}}(\mathbf{X}, \mathbf{Z}_i) \quad \forall i = 1, \ldots, m. \tag{10c}$$

The local step is almost the same as in the standard ADMM formulation from the previous section. The only difference is the proximal operator of Ψ is instead evaluated at $\mathrm{sym}((\mathbf{D}_i\mathbf{X})^T) + \mathbf{U}_i$. Again, in practice the proximal solve can be performed on just the singular values of $\mathbf{Z}_i$.

4.1 The global step with rotation awareness

The first stage of the iteration is the global step (10a). We update $\mathbf{X}$ by solving

$$\arg\min_{\mathbf{X}} \sum_{i=1}^{m} \frac{w_i^2}{2} \left\|\mathrm{sym}\left((\mathbf{D}_i\mathbf{X})^T\right) - \mathbf{P}_i\right\|_F^2, \tag{11}$$

where $\mathbf{P}_i = \mathbf{Z}_i - \mathbf{U}_i$, which is introduced for simplification. We solve this minimization problem using L-BFGS. This requires an expression for the gradient of the objective function with respect to $\mathbf{X}$. Here we give the main result; a derivation is provided in the supplementary document. This derivative can be found to be

$$\frac{dL}{d\mathbf{X}} = \sum_{i=1}^{m} w_i^2 \mathbf{D}_i^T \mathbf{M}_i^T, \tag{12}$$

where each $\mathbf{M}_i$ is computed as follows (we omit the element index i on intermediate variables for clarity):

$$\mathbf{F} = (\mathbf{D}_i\mathbf{X})^T, \tag{13}$$

$$(\mathbf{U}, \Sigma, \mathbf{V}) = \mathrm{svd}(\mathbf{F}), \tag{14}$$

$$\mathbf{H} = \left[\frac{\sigma_i}{\sigma_i + \sigma_j}(\mathbf{v}_i^T\mathbf{P}\mathbf{v}_j + \mathbf{v}_j^T\mathbf{P}\mathbf{v}_i)\right]_{ij} \tag{15}$$

$$\mathbf{M}_i = \mathbf{F} - \mathbf{U}\mathbf{H}\mathbf{V}^T. \tag{16}$$

L-BFGS does not require the exact Hessian of L; it progressively updates an approximate Hessian each iteration by using a history of past iterates. However, it does require an initial Hessian approximation as input. There are various options for initializers, but we have had good success following the approach proposed by Liu et al. [2017]. Specifically, we use the constant prefactored weighted Laplacian from the ADMM-PD formulation in (5a). One important benefit of this choice is that our method behaves no worse than ADMM-PD when L-BFGS has not built up a history.

The L-BFGS history window determines how many past iterates are used to approximate the Hessian. If set too small then there may be insufficient information for a good approximation, but if too large then the per-iteration cost increases, and distant past iterates may negatively bias the approximation. We use a window length of $l = 6$ in our method. Further, to enable each global step to immediately

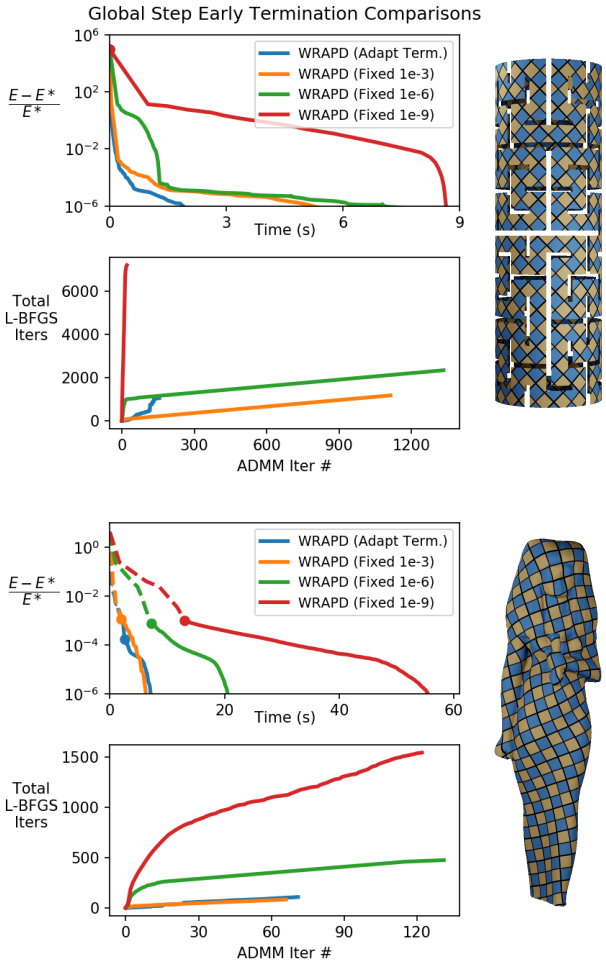

Fig. 3. Comparing global step termination methods using two symmetric Dirichlet parameterization examples. Our adaptive criterion is compared to three fixed tolerances. Top two plots: Energy error vs. time and total global step L-BFGS iterations vs. ADMM iteration, respectively. A dashed line indicates when an intermediate solution has triangle flips. Bottom two plots: Same as above, but using a different parameterization example. To the right of each pair of plots the corresponding mesh is textured with its solution.

benefit from previous iterates, we maintain the L-BFGS history across ADMM iterations whenever possible.

L-BFGS termination criterion. One important tradeoff is the balance between the time spent on solving the global step vs. the local steps. Since L-BFGS is only used in the global step, running it for too many iterations will not necessarily improve the overall convergence of ADMM. It is important to allow the local step to run often, especially when a problem is dominated by non-rotational material deformations. The relative importance of each the local and global steps is not only problem dependent, it may also change during minimization. For these reasons we favor an adaptive termination criterion that gives more time to whichever step is making more progress in minimizing the problem. In particular, we keep running L-BFGS iterations as long as each iteration is decreasing the augmented Lagrangian L by more than the previous local step did. This leads to the termination criterion

$$\Delta \mathbf{X} : \frac{dL}{d\mathbf{X}} \leq \kappa \left| \Delta L \right|, \tag{17}$$

where $\Delta \mathbf{X}$ is the step to be taken by the next L-BFGS iteration, and $|\Delta L|$ is the change in the augmented Lagrangian from the previous local step (before the $\mathbf{U}$-update). The coefficient κ allows us to tune the relative importance of each step. We discuss empirically good choices for κ in Section 6.

In Figure 3 we compare our adaptive termination criterion to fixed tolerances using two different parameterization examples. Our adaptive criterion outperforms fixed 10^{-6} and 10^{-9} thresholds, and is competitive with the much less precise 10^{-3} tolerance. This shows that the L-BFGS solves do not need to be solved very precisely. Our adaptive criterion takes advantage of this and only performs additional iterations when they are most beneficial for convergence. While we allow L-BFGS to run for a maximum of 100 iterations, in practice the global step almost always terminates in much fewer iterations — typically only 1 or 2.

Rotation-aware ADMM for ARAP. It is instructive to consider how rotation-aware ADMM behaves on the ARAP energy $\Psi(\mathbf{F}) = \| \operatorname{sym}(\mathbf{F}) - \mathbf{I} \|_F^2$. In this case, the ADMM local step has a closed-form update rule,

$$\mathbf{Z}_i \leftarrow \frac{1}{2}(\mathbf{I} + \operatorname{sym}((\mathbf{D}_i\mathbf{X})^T) + \mathbf{U}_i), \tag{18}$$

$$\mathbf{U}_i \leftarrow \mathbf{U}_i + \operatorname{sym}((\mathbf{D}_i\mathbf{X})^T) - \mathbf{Z}_i. \tag{19}$$

Therefore $\mathbf{P}_i = \mathbf{Z}_i - \mathbf{U}_i = \mathbf{I}$, and the global objective becomes exactly the total ARAP energy over the entire mesh. That is, the method reduces to solving the ARAP problem entirely in the global step using L-BFGS. In contrast, the original ADMM-PD algorithm behaves similarly to the classical local-global strategy, as observed by [Overby et al. 2017].

Practical benefits. In the first plot of Figure 4 we compare four variants of the ADMM method (with/without rotation-awareness, with/without weight updates) run on a practical quasi-static example. The rotation-aware algorithm (R-ADMM) dramatically outperforms ADMM-PD. For this comparison both methods use carefully tuned static weights. In Figure 5 we provide a similar comparison, but this time run on a parameterization example. Here rotation awareness is still highly beneficial, but to a lesser extent. Instead, for this parameterization problem weight updates are more important for rapid early progress. In the next section we will discuss our dynamic reweighting strategy for performing weight updates that can be used for often faster and more reliable convergence.

5 DYNAMIC REWEIGHTING

One notable limitation of ADMM-PD is that convergence greatly depends on the choice of weights. Lower weights can allow each iterate to make bigger updates, but if set too low the algorithm may not converge at all. In contrast, higher weights improve stability at the expense of performance. Existing methods use simple heuristics for defining the weights from approximate curvature information.

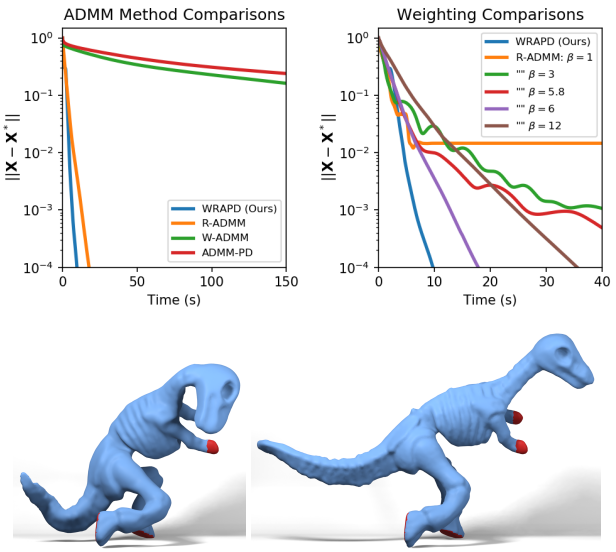

Fig. 4. Normalized position error vs. time comparisons using a quasi-static Neo-Hookean example. Top left: Comparing ADMM-PD [Overby et al. 2017], W-ADMM (dynamic reweighting), R-ADMM (rotation-awareness), and WRAPD (has both). Top right: Comparing WRAPD to R-ADMM with different weight multipliers β. Bottom left: Deformed initial mesh. Constrained regions are colored red. Bottom right: Quasi-static solution subject to constraints.

Overby et al. show that the energy Hessian evaluated in the rest pose is typically a good choice for the (squared) weights. Liu et al. [2017] instead use the slope of the best linear approximation to the stress function, integrated over an interval of expected deformations. Both of these methods may yield poor approximations to the Hessian when a mesh is highly deformed. To address this Overby and colleagues globally scale the weights by a user-specified factor. This is not a robust solution, since it is impossible to predict a good multiplier. Also, a few problematic elements may necessitate a very high multiplier for the whole mesh, and that will greatly impact performance. An example of these problems and limitations is shown in the right plot of Figure 4. This example demonstrates how sensitive convergence is to the choice of weight multiplier.

In the following subsections we discuss our method for dynamically updating weights during optimization. Later in Section 6 we demonstrate the effectiveness of this approach in both quasi-static and parameterization applications.

5.1 Weighting heuristic and Reweighting Step

Our basic idea is to derive dynamic weights from the maximum eigenvalue of the current Hessian. That is, we would like to derive weights from an effective stiffness given by

$$K_e(\boldsymbol{\sigma}) = \lambda_{\max}(\mathbf{K}(\boldsymbol{\sigma})), \tag{20}$$

where σ are the singular values in the current state and $\mathbf{K} = \frac{d^2\Psi}{d\boldsymbol{\sigma}^2}$. This is similar in concept to the heuristic Overby and colleagues use, except that theirs is evaluated at the reference state. Before

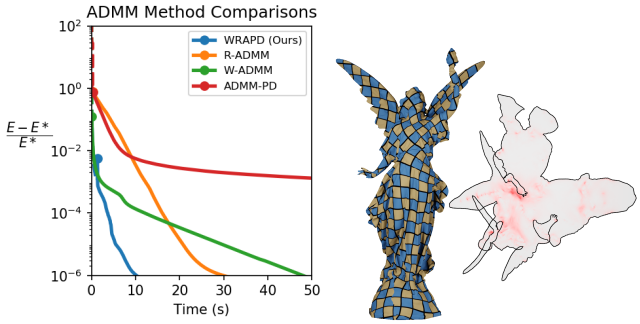

Fig. 5. Energy error vs. time comparisons using a symmetric Dirichlet parameterization example. *Left*: Comparing ADMM-PD [Overby et al. 2017], W-ADMM (dynamic reweighting), R-ADMM (rotation-awareness), and WRAPD (has both). *Middle*: Original mesh textured with solution. *Right*: Distortion-mapped UV coordinates; redder means higher distortion.

applying global multipliers, they set weights according to

$$w_0^2 = V_0 K_e(\boldsymbol{\sigma}_0) = V_0 \lambda_{\max}(\mathbf{K}_0), \tag{21}$$

where the subscript 0 is used to indicate that a term is evaluated in the zero-energy state, and V_0 is the reference measure (area or volume).

It is tempting to use (20) and set weights to $w^2 = V_0 K_e$ every ADMM iteration. However, there are a number of problems with this approach. At initialization a highly distorted element would call for a very high weight. This would impede early progress, often unnecessarily, since usually the optimized solution is much less distorted and can be attained faster with smaller weights. Another problem with constantly changing weights is the effect it has on the global step. One of the benefits of ADMM is its ability to use an unchanging prefactored matrix. However, each time weights are changed the $\mathbf{A}$ must be refactorized. Since $\mathbf{A}$ is the initial Hessian approximation for L-BFGS, its history must be cleared whenever $\mathbf{A}$ is changed. These operations incur extra overhead and hurt rotation-aware acceleration.

With consideration for the aforementioned issues, our method for computing and updating weights is as follows. First, we set weights to their reference values at initialization using (21). Then, at the end of each local step we compute the singular values of the updated $\{\mathbf{Z}_i\}$. We use local variables as opposed to $\mathbf{X}$ since the latter is not guaranteed to be free from inversions. We then use (20) to obtain the effective stiffness K_e in the current state, and use it to compute candidate weights, w_{cand}. These are clamped to within an acceptable range, specifically,

$$w_{\text{cand}}^2 = \text{clamp}(V_0 K_e, \beta_{\min} w_0^2, \beta_{\max} w_0^2), \tag{22}$$

where $\beta_{\min}$ and $\beta_{\max}$ are user-specified values for clamping.

To prevent small changes in weights from triggering global reweighting too frequently, we only reweight when the candidate weights differ significantly from the current weights. After the global step, we compare each element's current weight w to its latest candidate weight, w_{cand}. If the ratio between them exceeds some threshold, then that is an indication that at least one element's weight needs updating. Specifically, we flag the entire mesh for reweighting if

ALGORITHM 1: WRAPD

forall *elements i in parallel* **do**
  Initialize $\mathbf{Z}_i \leftarrow \mathrm{sym}\left((\mathbf{D}_i\mathbf{X})^T\right)$
  Initialize $\mathbf{U}_i \leftarrow \mathbf{0}$
  Initialize weight to reference value using (21)
end
Initialize $\mathbf{A}$ using (5a) and compute Cholesky factorization
while *not yet terminated* **do**
  Compute augmented Lagrangian L_0 using (9)
  forall *elements i in parallel* **do**
    Update $\mathbf{Z}_i$ using (10b)
    Update candidate weight using (22)
  end
  Compute augmented Lagrangian L_1 using (9)
  $|\Delta L| \leftarrow |L_1 - L_0|$
  forall *elements i in parallel* **do**
    Update $\mathbf{U}_i$ using (10c)
  end
  do
    Update $\mathbf{X}$ by performing an L-BFGS iteration
    Check for global step termination using (17)
  while *global step not yet terminated*
  if *any element needs reweighting using* (23) **then**
    forall *elements i in parallel* **do**
      $\mathbf{U}_i \leftarrow \left(\frac{w_i^2}{w_{\mathrm{cand},i}^2}\right)\mathbf{U}_i$
      $w_i \leftarrow w_{\mathrm{cand},i}$
    end
    Update $\mathbf{A}$ using (5a) and refactorize
    Clear the L-BFGS history
  end
  Check for ADMM termination
end

any element satisfies

$$\max\left(\frac{w_{\mathrm{cand}}^2}{w^2}, \frac{w^2}{w_{\mathrm{cand}}^2}\right) \geq \gamma, \tag{23}$$

where γ is a controllable threshold parameter. We empirically found that a value of $\gamma = 1.5$ works well, so we use this for all applications. In our supplementary document we include experimental results that show how performance varies as a function of γ in practical examples.

If the mesh is flagged for reweighting, then we perform the following operations. First, the ADMM $\{\mathbf{U}_i\}$ variables are rescaled by a factor of $\frac{w^2}{w_{\mathrm{cand}}^2}$ to maintain consistency between the optimization variables. The particular factor is chosen because the $\{\mathbf{U}_i\}$ are inversely proportional to the squared weights [Overby et al. 2017]. After rescaling, *all* element weights are set to their candidate values and the $\mathbf{A}$ matrix used in the global step is updated and factorized. Finally, the L-BFGS history is cleared.

Our proposed method WRAPD uses the rotation-aware algorithm introduced in the previous section coupled with dynamic reweighting. A summary of WRAPD is given in Algorithm 1.

6 APPLICATIONS

In this section we present applications of our algorithm to two types of geometric optimization problems: quasi-static deformation and parameterization. Although WRAPD is used for both, each has unique characteristics that benefit from specialized treatment. We will discuss specific parameter and heuristic choices that we have empirically found to yield good performance.

6.1 Quasi-static deformation

During the early phase of optimization we use static reference weights, so dynamic reweighting is not used. We make this choice because during the early iterations we are not concerned with late-phase convergence quality. Rather, the goal is for the mesh to reduce initial distortion as quickly as possible. We have found that only once an intermediate solution has been obtained is it necessary to consider adequate late-convergence weighting. Accordingly, we set $\beta_{\min} = \beta_{\max} = 1$ for the first k_{rw} iterations, then $\beta_{\min} = 0.1$ and $\beta_{\max} = 10$ afterwards. We empirically settled on a value of $k_{\mathrm{rw}} = 50$ for all of our quasi-static demos. Beyond iteration k_{rw} we follow the reweighting method outlined in Section 5.

For the L-BFGS early termination coefficient, we use $\kappa = 1$. This value was empirically obtained through experiments, as it was found to yield much better convergence in several demos.

6.2 Parameterization

Mesh parameterization is a challenging computational problem tasked with obtaining a mapping between vertex coordinates in $\mathbb{R}^3$ and a parameter domain surface in $\mathbb{R}^2$. Ideally this mapping should be foldover-free and have low distortion. The main task is to then reduce distortion from an initial embedding while still maintaining a foldover-free parameterization. This is a highly nonlinear problem that we solve using WRAPD.

As a first step we rescale the input mesh to unit area. This allows us to define solver parameters that work well for inputs of arbitrary scale. We then attempt to obtain an initial state by applying a harmonic embedding using cotangent weights. If this yields any flipped triangles we fall back to Tutte's embedding. For fair comparison this procedure is used for all methods in this paper.

We observe any nearly-degenerate triangle from the initial embedding has a large effective stiffness according to (20). Thus, for the first ADMM iteration we use static reference weights to help massively reduce initial distortion. Some problems do require specific elements to maintain high distortion in order to minimize global distortion. So, we would like for reweighting to be fairly insensitive to early distortion, but consider it more strongly in later iterations. To achieve this we devised a geometrically-increasing maximum weight clamp $\beta_{\max}$. Specifically, as functions of the ADMM iteration number $k = 0, 1, 2, \ldots$, we define

$$\beta_{\min}(k) = 1, \tag{24a}$$

$$\beta_{\max}(k) = \min(10(1.5)^k, 10^9), \tag{24b}$$

We fix $\beta_{\min} = 1$ because for parameterization we have found it is best if candidate weights are never allowed to be smaller than reference values. A hard cap of 10^9 ensures proper numerical conditioning during the global step linear solves. Difficult problems

may call for larger caps, but we find this to be more than sufficient in the vast majority of cases. In our supplementary document, we experimentally validate the need for increasing $\beta_{\max}$ gradually.

Finally, we have chosen an L-BFGS early termination coefficient of $\kappa = 2$, where the termination condition is given by (17). This is twice as large as what we use for quasi-statics. This allows the global step to terminate early with fewer iterations, which is beneficial for our goal to prioritize local steps and reweighting over additional global step iterations.

6.2.1 WRAPD as an Initializer. Although WRAPD makes good progress in both the early and late phases of convergence, it is not a second-order method. One algorithm with this attractive property is CM [Shtengel et al. 2017]. Theirs, like other Newton methods, generally starts slower but rapidly accelerates when nearing the solution. These opposing strengths motivate us to consider a hybrid approach, in which we use WRAPD as an initializer for CM. In the paper we will refer to this as WRAPD+CM.

Others have advocated for switching to second-order methods on challenging problems as well [Rabinovich et al. 2017, Fig. 19], [Jiang et al. 2017, Fig. 4 and Fig. 11]. Liu et al. [2018] use SLIM in their early iterations to quickly reduce initial distortion before switching to CM. However, we find WRAPD to be much more effective for this task, and to our knowledge ours is the first to use an ADMM-based solver as an initializer for parameterization.

In the WRAPD+CM method we run WRAPD for a minimum of one iteration. After each iteration we check to see if there are any element flips in the mesh. If there are, further iterations of WRAPD are performed, because CM requires a flip-free initialization. Once a valid state is attained CM is used thereafter and run to convergence.

7 RESULTS

The machine used to generate all our results and performance measurements contained a ten-core Intel i9-10900K CPU @ 3.70 GHz, and 128 GB of RAM.

7.1 Quasi-static Deformation

Our algorithm for solving quasi-static deformation problems was validated using a variety of different meshes and initial conditions. In all of the following tests we used Neo-Hookean elasticity with a Poisson's ratio of 0.45, but any other rotation-invariant energy model could be used too. We include comparisons to several existing techniques, namely ADMM-PD [Overby et al. 2017], BCQN [Zhu et al. 2018], L-BFGS [Liu et al. 2017], and AADR [Ouyang et al. 2020]. In agreement with the observations of Ouyang and colleagues, we found that in all our experiments AADR outperformed the previous AA-ADMM method [Zhang et al. 2019]. For this reason we have omitted comparisons to AA-ADMM.

In our experiments, BCQN has relatively poor performance compared to other techniques, in contrast to the results reported by Zhu et al. [2018]. We attribute this to the fact that their implementation of BCQN, which we use, performs linear solves using CHOLMOD [Chen et al. 2008], while all other methods tested use PARDISO [Alappat et al. 2020; Bollhöfer et al. 2020, 2019]. Considering their observations that PARDISO is "usually 1.4 to 3 times faster" than CHOLMOD, we expect that using the former would make their results more competitive with other methods, but still not faster than WRAPD.

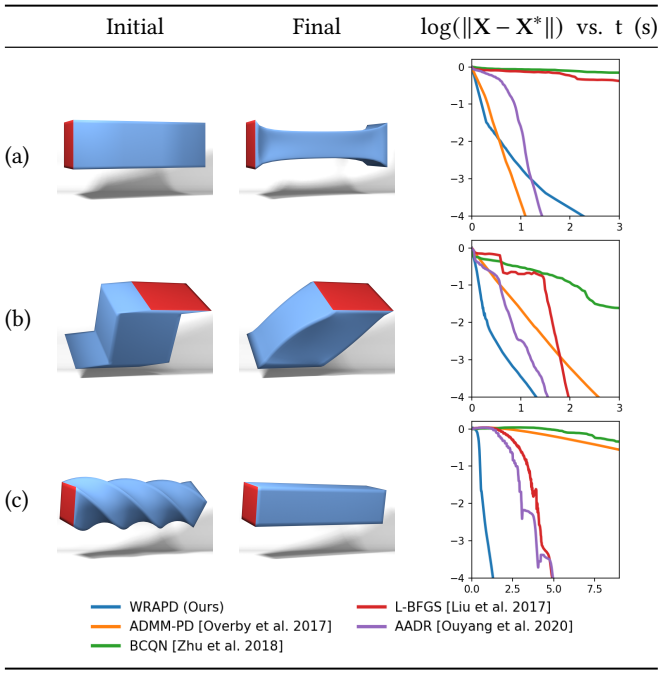

Fig. 6. Quasi-static neo-Hookean examples. *Left*: The initial deformed meshes. Constrained regions are colored red. *Middle*: Minimum energy solutions, subject to constraints. *Right*: Plots of normalized position error vs. time.

7.1.1 Performance Metric. Compared to stretching deformations, rotational modes such as bending and twisting often contribute very weakly to the problem objective. Thus, although an algorithm may make significant early progress to reduce distortion, it can still be far from the optimal solution $\mathbf{X}^*$. An example of this is illustrated in Figure 1, where we compare plots of energy E and position error $\|\mathbf{X} - \mathbf{X}^*\|$ vs. computation time. In this example the limbs and tail of the mesh model undergo bulk rotations in the late phase of convergence. Though this has a minor effect on the energy, it is visually significant for quasi-static deformation applications. We observed similar behavior in most of our tests, and for this reason we prefer to plot position error vs. computation time when comparing methods for quasi-statics.

For each problem the optimal solution $\mathbf{X}^*$ is obtained by running R-ADMM with static weights for a large number of iterations. For this task we use a large weight multiplier to ensure convergence. We confirmed that given enough iterations all algorithms get arbitrarily close to this solution, which validates that it is the correct configuration.

7.1.2 Experiments. We begin with some illustrative examples intended to evaluate the performance of different methods on stretching-dominated and rotation-dominated problems. All of our performance numbers are reported in Table 1.

Table 1. Quasi-static Neo-Hookean examples data. V and T denote the number of vertices and tetrahedra in each mesh, respectively. The terms N_c and t_c denote the iterations and time required to bring the normalized position error down to 10^{-4}. The fastest times are bolded. A dash is used for methods which were not included in the corresponding test.

Fig.	V/T	WRAPD (Ours)		ADMM-PD		BCQN		L-BFGS		AADR	
	($\times 10^3$)	N_c	t_c (s)	N_c	t_c (s)	N_c	t_c (s)	N_c	t_c (s)	N_c	t_c (s)
1	26/98	239	**12.5**	196604	3693	16167	13520	904	120	1786	138
6(a)	5/20	305	2.3	217	**1.1**	498	30.2	123	7.1	83	1.4
6(b)	5/20	158	**1.3**	548	2.6	166	10.7	50	2.0	94	1.6
6(c)	5/20	134	**1.3**	9757	42.2	282	34.2	249	4.9	249	5.0
7(a)	16/55	163	**4.1**	60883	559	29578	12301	417	16.3	590	23.3
7(b)	16/57	134	**3.5**	4104	45.6	1352	604	179	9.4	284	14.7
7(c)	16/58	429	**11.4**	122831	1270	6981	3107	336	15.4	860	34.3
7(d)	17/61	91	**2.3**	3900	47.4	1010	446	175	8.7	228	10.2
7(e)	25/94	224	**10.5**	168722	3078	10735	8459	650	77.9	1255	86.7
7(f)	25/93	239	**9.7**	123683	2107	6770	5372	471	52.2	1196	84.9
7(g)	26/102	117	**5.3**	6973	144	1237	771	210	26.3	308	24.9

In Figure 6(a) two faces of a cube are pulled outwards and pinned in place. The solution to this problem primarily involves element stretching and compression across the mesh. Consequently, our rotation-aware algorithm isn't clearly beneficial for this problem. However, this problem demonstrates that even in problems not dominated by rotations, our algorithm has minimal additional overhead so it remains competitive.

In Figure 6(b) an elastic cube is sheared by translating and pinning the top and bottom faces. Although this problem still doesn't significantly exercise rotational modes of deformation, it does so enough for WRAPD to outperform the ADMM-PD algorithm. In Figure 6(c) an elastic beam is initialized in a twisted configuration with one face pinned. The solution for this problem is the zero-energy untwisted state. Unlike the stretched beam, this problem almost exclusively consists of rotational deformations and WRAPD greatly outperforms all existing algorithms.

Next we consider less academic and more interesting examples involving mild deformations in practical meshes. The meshes in Figures 7(a)-7(d) were obtained from Zhu et al. [2018]. In almost all these problems WRAPD converges much faster than the other methods. The fastest among those we compared to is the L-BFGS method of Liu et al. [2018], which nearly ties WRAPD in the example of Figure 7(c).

Finally we ran our algorithm on four harder problems with significant, though still artistically plausible deformations (Figure 1 and Figures 7(e)-7(g)). The deformed and rest configuration meshes were obtained from Su et al. [2019], and constrained regions were chosen manually. In all these examples WRAPD converges at least 5x faster than all previous methods.

Our algorithm can also be used to solve problems with higher co-dimensions, like deformable thin sheets in $\mathbb{R}^3$. As an example, in Figure 8 a square of cloth is quasi-statically relaxed from a deformed configuration, subject to constrained corners and the influence of gravity. The cloth is modeled with ARAP elasticity, hard strain limiting, and quadratic bending resistance [Bergou et al. 2006]. During minimization large wrinkles propagate across the cloth, so the problem is dominated by element rotations. With rotation awareness WRAPD easily handles this and quickly converges to a low energy solution.

7.2 Parameterization

We performed a series of experiments to validate our parameterization algorithm and compare it to state-of-the-art methods. Before discussing results from these experiments, we will first briefly summarize our performance metrics and early termination criteria.

7.2.1 Performance Metrics and Early Termination. Similar to Liu et al. [2018], each parameterization algorithm is terminated early when either the relative change between two successive iterations gets sufficiently small, or the norm of the gradient is sufficiently small. Specifically, at the end of iteration k we terminate early if

$$\frac{|E^k - E^{k-1}|}{E^k} \leq \epsilon \quad \text{or} \quad \|\nabla E^k\| \leq \epsilon. \tag{25}$$

We chose $\epsilon = 10^{-12}$ for our standard benchmarks and scaling tests so we could accurately track the objective error down to within 10^{-6} of the solution. For these experiments the iterations and runtime cost required to reach the 10^{-6} error are denoted as N_c and t_c, respectively. For practical reasons and for consistency with Liu and colleagues, we used a bigger termination threshold of $\epsilon = 10^{-6}$ for large dataset experiments. For those N_c and t_c are instead defined as the iterations and time required to reach early termination. We also tracked the number of iterations η and runtime cost τ required to bring the objective down to within one percent of the converged energy solution, E_c. This metric is useful for quantifying how quickly each algorithm makes progress in the early iterations.

7.2.2 Experiments. We provide comparisons to multiple existing methods, including SLIM [Rabinovich et al. 2017], AKVF [Claici et al. 2017], CM [Shtengel et al. 2017], and BCQN [Zhu et al. 2018]. We evaluate the effectiveness of our parameterization algorithm on a variety of examples (Figure 11 and Table 2). In the graphs we use a dashed line to indicate when there may be flipped triangles in the

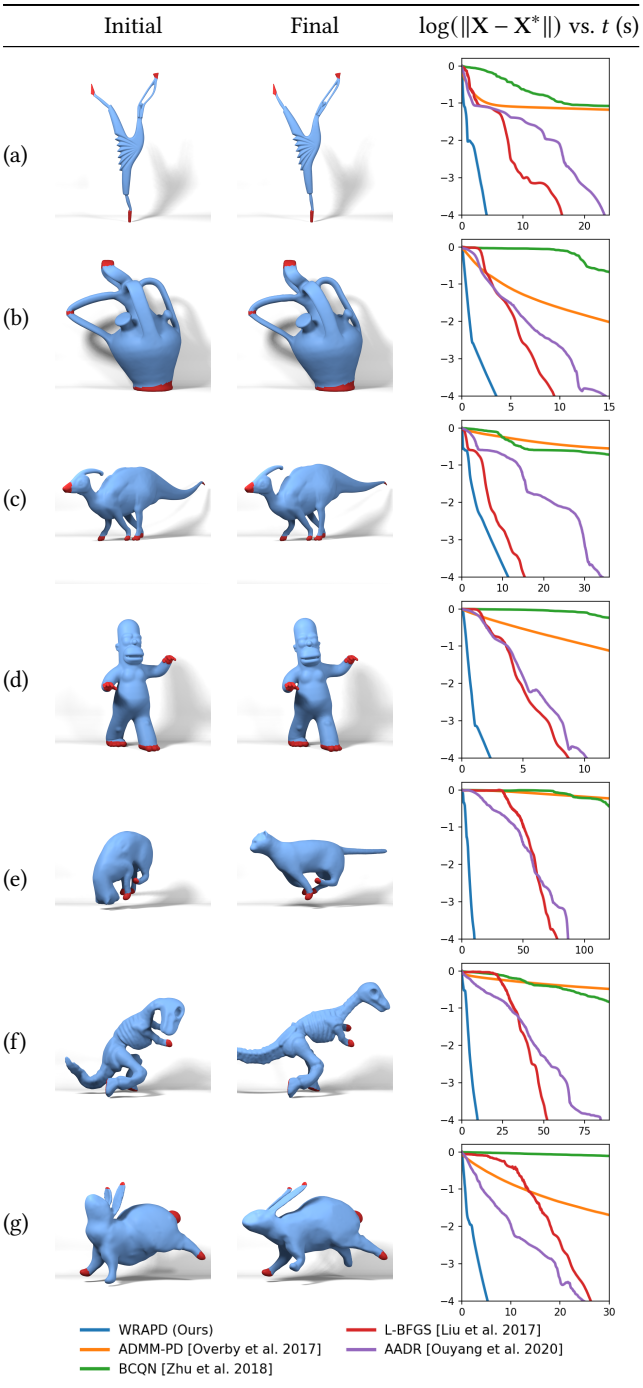

Fig. 7. Quasi-static neo-Hookean examples. *Left*: The initial deformed meshes. Constrained regions are colored red. *Middle*: Minimum energy solutions, subject to constraints. *Right*: Plots of normalized position error vs. time.

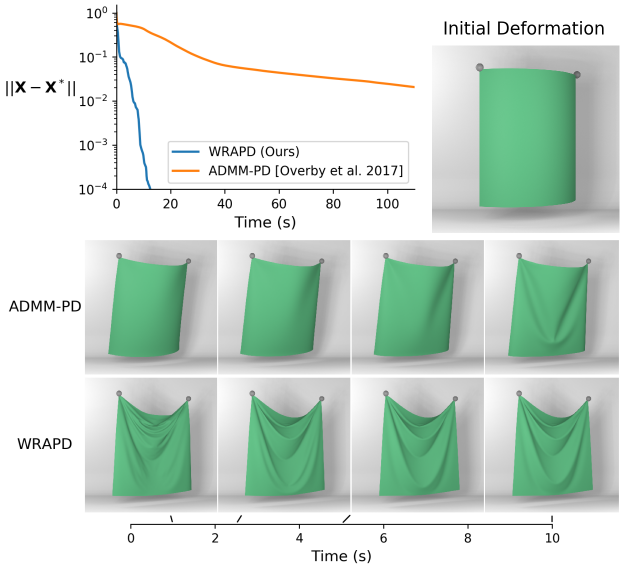

Fig. 8. Quasi-static cloth example. The cloth is modeled with ARAP stretching resistance, hard strain limiting, and quadratic bending resistance [Bergou et al. 2006]. *Top Left*: Plot of normalized position error vs. time. *Top Right*: Initial deformation. *Bottom*: Comparison between ADMM-PD and WRAPD intermediate solutions at select times.

intermediate solution. When such flips are present, we use our dual variables for computing the objective, since the energy function in terms of these coordinates is always well defined. In the plots we place a marker on the iteration immediately succeeding the last iteration that had triangle flips. Beyond this marker our method is plotted with a solid line, and it is during this solid-line region that a flip-free parameterization is maintained.

In each of these cases WRAPD makes strong progress in the early phases and dramatically reduces distortion in only a handful of iterations. This is in great contrast to the other methods that each take much longer to achieve similar quality approximate solutions. The other algorithms' slower early progress is primarily a consequence of line search filtering. Specifically, each existing method uses a flip-preventing line search to ensure that the mesh maintains local injectivity. Each of those algorithms require this invariant to be maintained every iteration. Since elements are most distorted near the initialization, their line searches yield smaller step sizes, which explains their much slower early progress. Our method does not have this limitation, due to the way variables are split using ADMM. Only our dual variables must remain injective, and this is trivially maintained in our local step. Consequently our method is robust to element inversions in the primal $\mathbf{X}$ variables. Moreover, while our method does not strictly prevent triangles from flipping, in practice we find that flips usually only occur in the beginning.

Since the standalone WRAPD algorithm is an enhanced first-order method, it is generally outperformed by CM (a second-order method) in the late phase of convergence. This limitation is overcome when WRAPD is used as an initializer for CM. The hybrid WRAPD+CM method is often faster than standalone WRAPD in the early phase, and it is competitive with CM in the late phase.

Table 2. Symmetric Dirichlet parameterization examples data. The mesh resolutions are specified in number of vertices V and triangles F. The terms η (N_c) and τ (t_c) denote the iterations and time required to reach the early (late) phases of convergence, respectively. The fastest times are bolded.

Fig.	V/F	WRAPD (Ours)		WRAPD+CM		CM		SLIM		AKVF		BCQN	
	($\times 10^3$)	η/N_c	τ/t_c (s)	η/N_c	τ/t_c (s)	η/N_c	τ/t_c (s)	η/N_c	τ/t_c (s)	η/N_c	τ/t_c (s)	η/N_c	τ/t_c (s)
11(a)	18/35	7/114	0.10/0.94	4/14	**0.06**/**0.26**	10/17	0.18/0.32	12/739	0.19/11.9	6/44	0.16/1.34	35/264	1.57/17.5
11(b)	40/78	9/209	0.27/4.22	8/27	**0.26**/**1.27**	18/38	0.95/2.07	19/1343	0.81/57.2	19/134	1.35/10.3	114/326	8.92/38.3
11(c)	47/92	8/76	**0.25**/2.02	5/19	0.26/**1.06**	16/29	0.92/1.73	12/2864	0.54/130	14/148	1.07/14.1	69/737	6.76/134
11(d)	25/48	5/208	**0.08**/1.91	5/19	0.08/0.37	9/15	0.21/**0.35**	22/818	0.48/16.1	35/82	1.05/2.68	48/237	2.44/19.7
11(e)	44/86	8/77	0.20/1.47	4/17	**0.16**/**0.72**	11/20	0.44/0.82	15/2048	0.52/72.1	11/610	0.62/51.1	76/1154	6.75/191
11(f)	62/121	7/73	**0.29**/**2.25**	5/37	0.36/2.87	12/36	0.92/2.88	10/3920	0.66/255	19/1015	1.99/151	130/681	17.2/167
11(g)	18/32	23/314	0.17/3.77	6/89	**0.05**/0.90	17/74	0.13/**0.69**	46/235726	0.33/1455	9/4387	0.13/57.1	56/751	2.48/21.6
11(h)	21/40	6/60	0.08/0.66	3/21	**0.05**/**0.42**	12/31	0.21/0.56	12/897	0.18/13.6	9/36	0.24/0.99	36/229	1.58/14.4
11(i)	188/374	7/72	0.92/7.74	4/11	**0.79**/**3.52**	6/13	2.00/4.35	17/1065	5.29/333	19/514	7.40/212	61/351	25.8/195

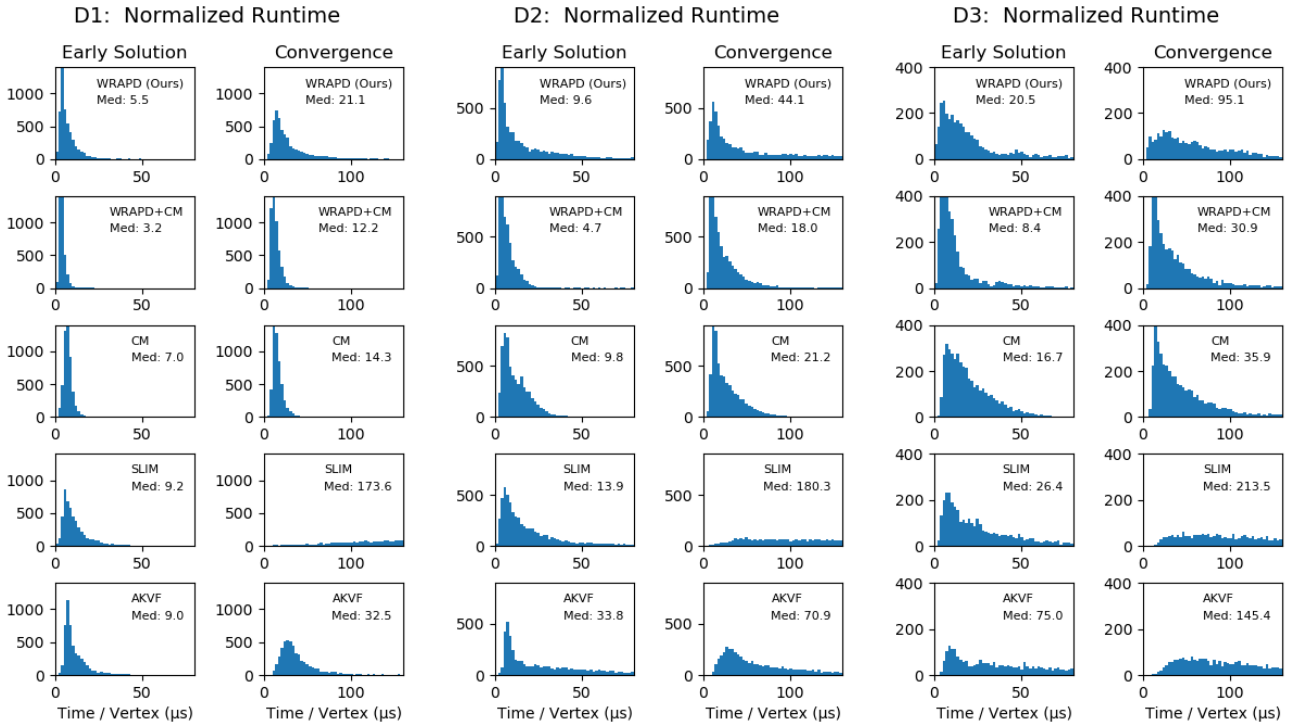

Fig. 9. Results from running symmetric Dirichlet parameterization methods on the large datasets from Liu et al. [2018]. The performance numbers for each run are normalized by the number of mesh vertices. For each dataset we show separate distributions for reaching the early and late phases of convergence.

For a more systematic comparison we ran our algorithm on the three large datasets from Liu et al. [2018]. Runtime statistics from these comparisons are shown in histograms in Figure 9. Since the parameterization problems all have different resolutions, like Liu and colleagues we chose to plot effective per-vertex runtime costs in the histograms. Our standalone WRAPD algorithm outperforms SLIM and AKVF in both the early and late phases. It is competitive with CM in the early phase but CM is faster in the late phase due to its second-order nature. However, WRAPD+CM is fastest among all algorithms; it finishes the early phase 2x faster than CM, and is slightly faster in the late phase, too.

To investigate the scalability of our algorithm we used the scalability benchmark provided by Su et al. [2020]. In this benchmark seven different resolutions of the Lucy mesh are parameterized, and the results are shown in Figure 10 and Table 3. Both WRAPD and WRAPD+CM scale well with increasing problem size in the early phase. Since WRAPD+CM switches to CM by the late phase, it exhibits excellent late phase scaling in close alignment with CM.

Table 3. Data from running a symmetric Dirichlet parameterization scaling test from Su et al. [2020]. The mesh resolutions are specified in number of vertices V and triangles F. The terms η (N_c) and τ (t_c) denote the iterations and time required to reach the early (late) phases of convergence, respectively. The fastest times are bolded. A dash is used for methods which did not reach the late phase within 10^4 seconds.

V/F	WRAPD (Ours)		WRAPD+CM		CM		SLIM		AKVF	
($\times 10^3$)	η/N_c	τ/t_c (s)	η/N_c	τ/t_c (s)	η/N_c	τ/t_c (s)	η/N_c	τ/t_c (s)	η/N_c	τ/t_c (s)
51/100	11/580	0.39/12.1	6/69	**0.26**/3.53	13/65	0.62/**3.28**	24/11569	1.04/518	19/294	1.43/28.2
105/206	9/2001	0.68/73.1	5/124	**0.54**/**18.1**	16/138	2.22/19.8	26/10197	3.23/1337	37/601	6.93/152
205/405	9/472	1.44/42.0	5/67	**1.23**/**24.2**	16/101	5.45/34.9	36/12086	11.1/3891	27/1636	11.9/989
403/800	9/517	3.04/97.3	9/78	**3.01**/**60.5**	18/76	14.4/63.2	29/—	22.1/—	30/3426	33.2/4790
813/1616	9/2271	**7.15**/755	6/76	7.88/**174**	25/83	57.0/195	31/—	68.6/—	41/—	112/—
1618/3224	9/4607	**16.6**/3183	6/118	21.2/629	23/85	117/**448**	36/—	182/—	71/—	455/—
3198/6376	9/7806	**41.3**/—	6/172	60.5/2704	29/152	445/**2421**	36/—	560/—	243/—	4217/—

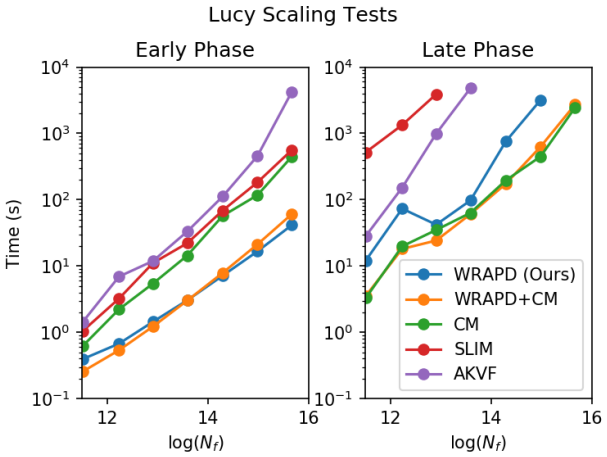

Fig. 10. Symmetric Dirichlet parameterization scaling test from Su et al. [2020]. Methods were run on seven different resolutions of the Lucy model. The time required to reach the early (late) phase of convergence is plotted on the left (right). The x and y axes increase logarithmicly in mesh resolution (# triangles) and time, respectively.

8 CONCLUSION

Our work in this paper arose from two key insights: first, that local-global solvers for geometry optimization should use rotation-invariant local coordinates in order to fully exploit the problem structure, and second, that element weights should be adapted to the local curvature of the objective for optimal convergence. We have shown that both goals can be achieved without adding substantial computational overhead to the ADMM framework, and pay off in dramatically improved convergence and stability. The resulting WRAPD algorithm converges much faster than existing approaches on many practical problems. It makes especially fast progress in the early iterations, making it valuable both as a general-purpose geometry optimization algorithm and as an initializer for more expensive second-order methods such as CM.

8.1 Limitations and future work

In this work, we only minimize the sum of per-element distortion energies, without considering any nonlocal constraints such as non-interpenetration between different parts of the mesh. As a result, our computed solutions are inversion-free but may not be globally injective. Indeed, overlaps can be observed in some of our parameterization results. Adding non-interpenetration constraints to ADMM-based geometry optimization is ongoing work that we hope to build on in the future.

As with the previous ADMM-PD algorithm, we cannot offer any theoretical guarantees on convergence, since classical convergence results for ADMM only apply to convex optimization problems [Boyd et al. 2011]. While ADMM-PD can indeed fail to converge on geometry optimization problems, particularly when the weights are too low, WRAPD usually avoids this pitfall thanks to our reweighting technique. Nevertheless, even with reweighting enabled, we have observed that on the Liu et al. [2018] parameterization dataset, WRAPD failed to reach the termination threshold of $\epsilon = 10^{-6}$ within 5000 iterations on 585 of the 15k meshes (3.8%). It did reach injectivity on many of these meshes, allowing WRAPD+CM to converge on all but 201 meshes (1.3%). No convergence failures occurred on any of our other examples.

The ADMM optimization framework has only first-order convergence, and despite our improvements to convergence speed, our method inherits this asymptotic convergence behavior. As a result, while we observe rapid progress in the early iterations, convergence can slow down and lag behind second-order methods like CM in the late phase. For applications where it is desired to reach very close to the minimum, we therefore recommend using our algorithm as an initializer to a second-order method, as in our WRAPD+CM variant for parameterization.

Many implementations of the ADMM algorithm [Narain et al. 2016; Overby et al. 2017; Zhang et al. 2019], including ours, define per-element weights to be scalar multiples of the identity matrix. Typically the scalar multiple is derived from some characteristic stiffness of the material. Using non-scalar weights could potentially improve convergence, especially when some stretching modes are stiffer than others. In the ADMM-PD algorithm the coupling of geometric and material nonlinearity makes it difficult to define good rotationally invariant weights. However since the local step of our method only contains material nonlinearity, it may be possible to derive reliable non-scalar weights directly from physical quantities such as the material stiffness matrix.

Our hybrid parameterization method has some features similar to progressive parameterizations (PP) [Liu et al. 2018]. Like PP, our hybrid method uses two different algorithms as subprocesses. For strong early progress we use WRAPD, while PP uses SLIM. Both our method and PP use CM in later iterations. However, a key distinction between our method and PP is how reference triangles are handled. We always use reference triangles with the same shape as the triangles in the original 3D surface mesh. In contrast, the main contribution of PP is their reference triangle updating scheme. In their method reference triangles are regularly modified so as to minimize element distortions and in turn reduce the complexity of their sub-problems, solved with either SLIM or CM. We believe that many algorithms, including ours, could benefit from this progressive framework. Because of this observation, we have omitted a direct comparison to PP. For future work we would like to explore using the progressive framework in conjunction with our method.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable comments. We also appreciate the insightful discussions and feedback received from Matthew Overby while working on this project. This work was supported in part by the National Science Foundation under grant IIS-1657089.

REFERENCES

Christie Alappat, Achim Basermann, Alan R. Bishop, Holger Fehske, Georg Hager, Olaf Schenk, Jonas Thies, and Gerhard Wellein. 2020. A Recursive Algebraic Coloring Technique for Hardware-Efficient Symmetric Sparse Matrix-Vector Multiplication. *ACM Trans. Parallel Comput.* 7, 3, Article 19 (June 2020), 37 pages. https://doi.org/10.1145/3399732

Martin Benning, Florian Knoll, Carola-Bibiane Schönlieb, and Tuomo Valkonen. 2016. Preconditioned ADMM with Nonlinear Operator Constraint. In *System Modeling and Optimization*, Lorena Bociu, Jean-Antoine Désidéri, and Abderrahmane Habbal (Eds.). Springer International Publishing, Cham, 117–126.

Miklos Bergou, Max Wardetzky, David Harmon, Denis Zorin, and Eitan Grinspun. 2006. A Quadratic Bending Model for Inextensible Surfaces. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing (SGP '06)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 227–230. http://dl.acm.org/citation.cfm?id=1281957.1281987

Matthias Bollhöfer, Aryan Eftekhari, Simon Scheidegger, and Olaf Schenk. 2019. Large-scale Sparse Inverse Covariance Matrix Estimation. *SIAM Journal on Scientific Computing* 41, 1 (2019), A380–A401. https://doi.org/10.1137/17M1147615 arXiv:https://doi.org/10.1137/17M1147615

Matthias Bollhöfer, Olaf Schenk, Radim Janalik, Steve Hamm, and Kiran Gullapalli. 2020. State-of-the-Art Sparse Direct Solvers. (2020), 3–33. https://doi.org/10.1007/978-3-030-43736-7_1

Sofien Bouaziz, Mario Deuss, Yuliy Schwartzburg, Thibaut Weise, and Mark Pauly. 2012. Shape-Up: Shaping Discrete Geometry with Projections. *Comput. Graph. Forum* 31, 5 (Aug. 2012), 1657–1667. https://doi.org/10.1111/j.1467-8659.2012.03171.x

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. 2014. Projective Dynamics: Fusing Constraint Projections for Fast Simulation. *ACM Trans. Graph.* 33, 4, Article 154 (July 2014), 11 pages. https://doi.org/10.1145/2601097.2601116

S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. 2011. . https://doi.org/10.1561/2200000016

Christopher Brandt, Elmar Eisemann, and Klaus Hildebrandt. 2018. Hyper-Reduced Projective Dynamics. *ACM Trans. Graph.* 37, 4, Article Article 80 (July 2018), 13 pages. https://doi.org/10.1145/3197517.3201387

Yanqing Chen, Timothy A. Davis, William W. Hager, and Sivasankaran Rajamanickam. 2008. Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate. *ACM Trans. Math. Softw.* 35, 3, Article 22 (Oct. 2008), 14 pages. https://doi.org/10.1145/1391989.1391995

S. Claici, M. Bessmeltsev, S. Schaefer, and J. Solomon. 2017. Isometry-Aware Preconditioning for Mesh Parameterization. *Comput. Graph. Forum* 36, 5 (Aug. 2017), 37–47. https://doi.org/10.1111/cgf.13243

Anqi Fu, Junzi Zhang, and Stephen Boyd. 2020. Anderson Accelerated Douglas–Rachford Splitting. *SIAM Journal on Scientific Computing* 42, 6 (2020), A3560–A3583.

Theodore F. Gast, Craig Schroeder, Alexey Stomakhin, Chenfanfu Jiang, and Joseph M. Teran. 2015. Optimization Integrator for Large Time Steps. *IEEE Transactions on Visualization and Computer Graphics* 21, 10 (Oct. 2015), 1103–1115. https://doi.org/10.1109/TVCG.2015.2459687

Zhongshi Jiang, Scott Schaefer, and Daniele Panozzo. 2017. Simplicial Complex Augmentation Framework for Bijective Maps. *ACM Trans. Graph.* 36, 6, Article 186 (Nov. 2017), 9 pages. https://doi.org/10.1145/3130800.3130895

Martin Komaritzan and Mario Botsch. 2018. Projective Skinning. In *Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*.

Martin Komaritzan and Mario Botsch. 2019. Fast Projective Skinning. In *Motion, Interaction and Games (MIG '19)*. Association for Computing Machinery, New York, NY, USA, Article 22, 10 pages. https://doi.org/10.1145/3359566.3360073

Shahar Z. Kovalsky, Meirav Galun, and Yaron Lipman. 2016. Accelerated Quadratic Proxy for Geometric Optimization. *ACM Trans. Graph.* 35, 4, Article 134 (July 2016), 11 pages. https://doi.org/10.1145/2897824.2925920

Ligang Liu, Chunyang Ye, Ruiqi Ni, and Xiao-Ming Fu. 2018. Progressive Parameterizations. *ACM Transactions on Graphics(SIGGRAPH)* 37, 4 (2018).

Ligang Liu, Lei Zhang, Yin Xu, Craig Gotsman, and Steven J. Gortler. 2008. A Local/Global Approach to Mesh Parameterization. In *Proceedings of the Symposium on Geometry Processing (SGP '08)*. Eurographics Association, Goslar, DEU, 1495–1504.

Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. 2017. Quasi-Newton Methods for Real-Time Simulation of Hyperelastic Materials. *ACM Trans. Graph.* 36, 4, Article 116a (May 2017), 16 pages. https://doi.org/10.1145/3072959.2990496

Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus Gross. 2011. Example-based Elastic Materials. *ACM Trans. Graph.* 30, 4, Article 72 (July 2011), 8 pages.

Rahul Narain, Matthew Overby, and George E. Brown. 2016. ADMM ⊇ Projective Dynamics: Fast Simulation of General Constitutive Models. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '16)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 21–28. http://dl.acm.org/citation.cfm?id=2982818.2982822

Wenqing Ouyang, Yue Peng, Yuxin Yao, Juyong Zhang, and Bailin Deng. 2020. Anderson Acceleration for Nonconvex ADMM Based on Douglas-Rachford Splitting. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 221–239.

Matthew Overby, George E. Brown, Jie Li, and Rahul Narain. 2017. ADMM ⊇ Projective Dynamics: Fast Simulation of Hyperelastic Models with Dynamic Constraints. *IEEE Transactions on Visualization and Computer Graphics* 23, 10 (Oct 2017), 2222–2234. https://doi.org/10.1109/TVCG.2017.2730875

Yue Peng, Bailin Deng, Juyong Zhang, Fanyu Geng, Wenjie Qin, and Ligang Liu. 2018. Anderson Acceleration for Geometry Optimization and Physics Simulation. *ACM Trans. Graph.* 37, 4, Article 42 (July 2018), 14 pages. https://doi.org/10.1145/3197517.3201290

Michael Rabinovich, Roi Poranne, Daniele Panozzo, and Olga Sorkine-Hornung. 2017. Scalable Locally Injective Mappings. *ACM Trans. Graph.* 36, 2, Article 37a (April 2017). https://doi.org/10.1145/2983621

Anna Shtengel, Roi Poranne, Olga Sorkine-Hornung, Shahar Z. Kovalsky, and Yaron Lipman. 2017. Geometric Optimization via Composite Majorization. *ACM Trans. Graph.* 36, 4, Article 38 (July 2017), 11 pages. https://doi.org/10.1145/3072959.3073618

Olga Sorkine and Marc Alexa. 2007. As-Rigid-as-Possible Surface Modeling. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing (SGP '07)*. Eurographics Association, Goslar, DEU, 109–116.

Jian-Ping Su, Xiao-Ming Fu, and Ligang Liu. 2019. Practical Foldover-Free Volumetric Mapping Construction. *Computer Graphics Forum* 38, 7 (2019), 287–297. https://doi.org/10.1111/cgf.13837

Jian-Ping Su, Chunyang Ye, Ligang Liu, and Xiao-Ming Fu. 2020. Efficient Bijective Parameterizations. *ACM Trans. Graph.* 39, 4, Article 111 (July 2020), 8 pages. https://doi.org/10.1145/3386569.3392435

Marcel Weiler, Dan Koschier, and Jan Bender. 2016. Projective Fluids. In *Proceedings of the 9th International Conference on Motion in Games (MIG '16)*. ACM, New York, NY, USA, 79–84. https://doi.org/10.1145/2994258.2994282

Juyong Zhang, Yue Peng, Wenqing Ouyang, and Bailin Deng. 2019. Accelerating ADMM for Efficient Simulation and Optimization. *ACM Trans. Graph.* 38, 6, Article Article 163 (Nov. 2019), 21 pages. https://doi.org/10.1145/3355089.3356491

Yufeng Zhu, Robert Bridson, and Danny M. Kaufman. 2018. Blended Cured Quasi-newton for Distortion Optimization. *ACM Trans. Graph.* 37, 4, Article 40 (July 2018), 14 pages. https://doi.org/10.1145/3197517.3201359

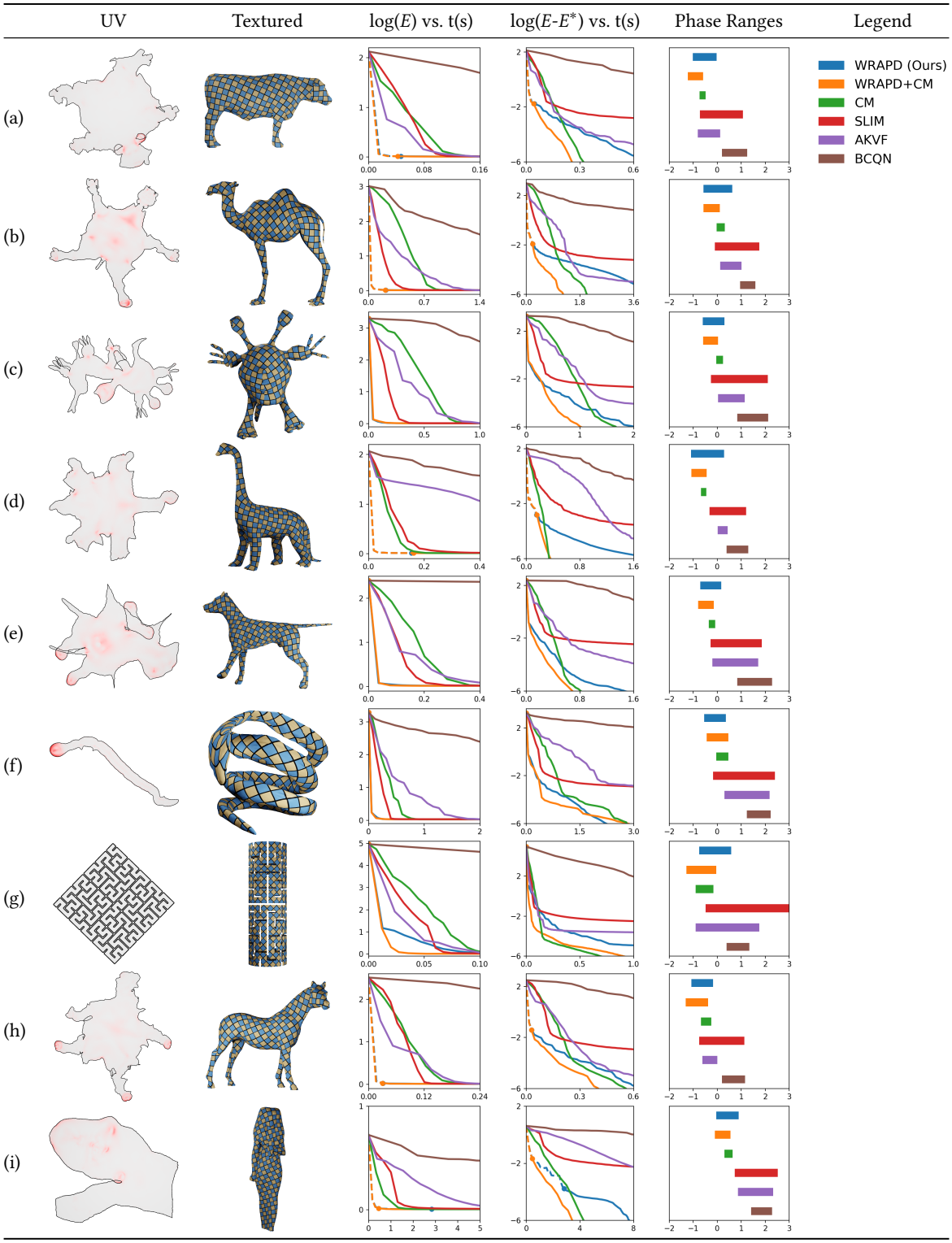

Fig. 11. Symmetric Dirichlet parameterization examples. Columns from left to right: (1) Distortion-mapped UV coordinates; redder means higher distortion. (2) Original mesh textured with solution (3) log(Energy) vs. Time (s). A dashed line indicates when an intermediate solution has triangle flips. (4) log(Energy error) vs. Time (s). Error is measured with respect to converged solution. (5) Phase ranges. Horizontal bars span from the end of the early phase to the start of the late phase of convergence. The x-axis is the log of the time in seconds.