DOI: 10.1111/cgf.14361
Eurographics Symposium on Geometry Processing 2021
K. Crane and J. Digne
(Guest Editors)

Volume 40 (2021), Number 5

Globally Injective Geometry Optimization with Non-Injective Steps

Matthew Overby[1], Danny Kaufman[2], and Rahul Narain[3]

[1]University of Minnesota, USA
[2]Adobe Research, USA
[3]Indian Institute of Technology Delhi, India

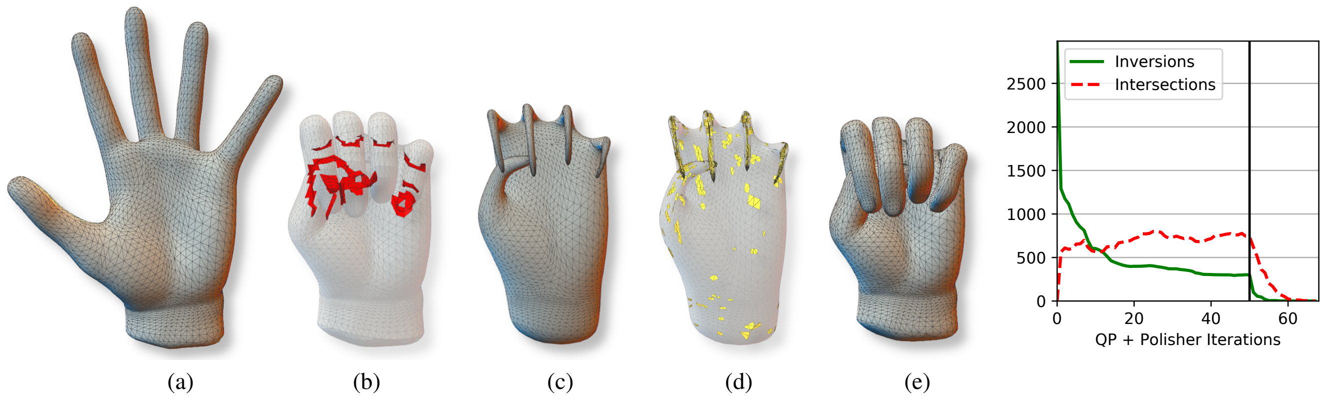


Figure 1: *Globally injective shape deformation. A tetrahedral hand mesh (a) is mapped to a target surface that has self-intersections (b). Flowing the surface with cMCF [KSBC12] (c) resolves the penetrations but creates many element inversions in the resulting mesh (d). From this starting point, we apply our algorithm (GINI) to resolve all inversions and preserve non-penetration, producing a final mesh (e) that is globally injective. Our method combines a fast approximate quadratic programming (QP) solver with a constraint polisher to simultaneously minimize deformation energy (here, neo-Hookean) and resolve hard constraints. At right, we visualize progress in resolving injectivity violations over first GINI iteration (50 QP iterations, 16 polisher iterations), with a vertical line indicating the start of the constraint polisher.*

Abstract

We present a method to minimize distortion and compute globally injective mappings from non-injective initialization. Many approaches for distortion minimization subject to injectivity constraints require an injective initialization and feasible intermediate states. However, it is often the case that injective initializers are not readily available, and many distortion energies of interest have barrier terms that stall global progress. The alternating direction method of multipliers (ADMM) has recently gained traction in graphics due to its efficiency and generality. In this work we explore how to endow ADMM with global injectivity while retaining the ability to traverse non-injective iterates. We develop an iterated coupled-solver approach that evolves two solution states in tandem. Our primary solver rapidly drives down energy to a nearly injective state using a dynamic set of efficiently enforceable inversion and overlap constraints. Then, a secondary solver corrects the state, herding the solution closer to feasibility. The resulting method not only compares well to previous work, but can also resolve overlap with free boundaries.

CCS Concepts
• ***Computing methodologies*** $\rightarrow$ ***Shape modeling;***

1. Introduction

Many important tasks in computer graphics require computing a low-distortion mapping of a shape subject to two constraints: no inverted elements and no interpenetration. Minimizing mapping distortion subject to these constraints, which together ensure global injectivity, is a challenging task. First, many deformation energies of interest are designed to diverge to infinity as elements approach degeneracy and so pose severe difficulties for numerical solvers.

© 2021 The Author(s)
Computer Graphics Forum © 2021 The Eurographics Association and John Wiley & Sons Ltd. Published by John Wiley & Sons Ltd.

delivered by EUROGRAPHICS DIGITAL LIBRARY
www.eg.org diglib.eg.org

Such barrier-type energies are often essential for modeling elastic deformation as well as for high-quality mesh parameterization and geometry processing. Second, non-inversion and non-overlap give rise to challenging nonlinear constraints that are not easy to enforce globally, particularly for high-resolution meshes. However, these constraints are necessary for many applications in geometry processing and penetration-free shape manipulation.

Most prior work in geometry optimization, following the seminal work of Smith and Schaefer [SS15], focuses on iteratively approaching a minimum while ensuring each intermediate step remains globally injective. These conservative approaches, called "maintenance-based" methods by some [SFL19], comes with natural limitations. First, ensuring injectivity at each iteration necessitates a filtered line search step. The presence of even a small number of highly distorted elements can generate "line search blocking" [ZBK18] which may stall global progress. Because of this, maintenance-based methods often perform better when the starting point has lower distortion. Second, the solver must be initialized with a globally injective state in order to begin the optimization. Creating a globally injective initializer itself is a nontrivial task for many problems, such as 3D deformation with user-specified constraints, and 3D shape mapping, where typical initializers such as harmonic mapping [EDD*95] or Tutte embedding [Tut63] do not guarantee an injective solution.

For these problems, maintenance-based solvers must be preceded by an initial phase that repairs the input mesh until it is feasible. This task is addressed by a much smaller class of methods, referred to as "map fixers" [NZZ20]. For large and complex domains, this map-fixing phase can have a substantial additional cost. More importantly, there are no map fixers that can also maintain non-penetration in the presence of free boundaries, necessitating that the boundary be artificially held fixed to ensure a globally injective output. Therefore, a general solution for nonlinear infeasible-start geometry optimization with global injectivity constraints remains a challenging problem.

Our Approach In this work, we investigate a different, less conservative approach to globally injective distortion minimization that permits the use of non-injective initialization and intermediate iterates.

Building on the ADMM-PD algorithm's [OBLN17] ability to take large optimization steps even when initialized with many inverted elements, we introduce an ADMM-based solver that quickly reduces distortion while approximately respecting injectivity constraints. This is interleaved with a secondary "polishing" solver that rapidly drives the mesh towards global injectivity. These solvers are coupled through a dynamically updated set of constraint functions to enforce per-element volumes and non-penetration that activate when they are near violation. These three novel components together define our algorithm for **g**lobally **i**njective geometry optimization with **n**on-**i**njective steps (GINI). Compared to existing techniques for this task, GINI offers the following advantages:

1. It is highly efficient at computing a low-distortion globally injective mapping on large-scale problems. In our experiments, we find that it reaches injectivity much faster, and often more reliably, than existing map fixers on problems with fixed boundaries (Sections 8.1) and free boundaries (Section 8.2). As such, it can be used in place of a map fixer to compute an initialization for a maintenance-based solver.
2. It efficiently supports overlap constraints, which pose a challenge for existing techniques, especially in 3D. When starting from a state that does not have boundary overlap, it can be used as a standalone solver to perform interpenetration-free shape deformation (Section 8.3).

We show two examples in Fig. 2 of both a fixed boundary mapping problem and energy minimization. In the left image, a high-resolution armadillo mesh has 219,355 vertices and 928,030 tetrahedra. Our algorithm reaches injectivity in just 9.9 seconds with one QP iteration and ten polisher iterations. In the right image, a wrench of 14,798 vertices and 50,122 tetrahedra is sheared and rotated inducing 4,726 inversions. A globally injective state is reached in just under a second, with further iterations continuing to minimize energy.

2. Related Work

General deformation tasks and quasi-static simulation aim to achieve realistic behavior in the form of hyperelastic models found in continuum mechanics literature. These derive from empirical analysis and include neo-Hookean [BW97], St. Venant-Kirchhoff, Mooney-Rivlin [Riv97, Moo40], Fung [Fun13], and others. These models resist extreme compression, typically in the form of a barrier term which goes to infinity as the volume approaches zero. Numerical problems related to these energies have motivated techniques to improve robustness under inversion [ITF04, SHST12].

Texture parameterization is another task that necessitates a low-distortion, one-to-one mapping, but from a 3D surface to a 2D plane. Tutte's theorem [Tut63] guarantees such a mapping exists when the boundary is convex, though the result has high distortion. Harmonic maps [EDD*95] often produce lower distortion, but do not guarantee injectivity on general meshes. Tutte and harmonic embeddings are often used as initializers for other methods, including volumetric mappings, that aim to further minimize distortion. However, Tutte's guarantee of injectivity does not extend to 3D. We discuss methods for injective geometry optimization and creating injective initializers below.

Geometry Optimization Early methods for distortion minimization with local injectivity relied on barrier functions [SKPSH13, FLG15] which have been improved in terms of speed and scalability [RPPSH17]. Barrier energies have also been used for global injectivity by [SS15] to prevent overlap in free boundary parameterization and were one of the earliest methods to prevent interpenetration in computer graphics simulation [TPBF87]. To improve efficiency, [SS15] introduce flip-avoiding line search to compute the maximum allowable step size until an element inverts, thus reducing the amount of energy evaluations. Another approach called barrier-aware filtering has been used to mitigate line search blocking [ZBK18]. Interest has also grown in fictitious domain methods to prevent overlap, which recast the problem to the embedding domain by creating so-called scaffolding or an air-mesh [ZMT05, JSP17, SYLF20, MCKM15]. While [JSP17] and [MCKM15] are capable of volumetric simulation, the approach

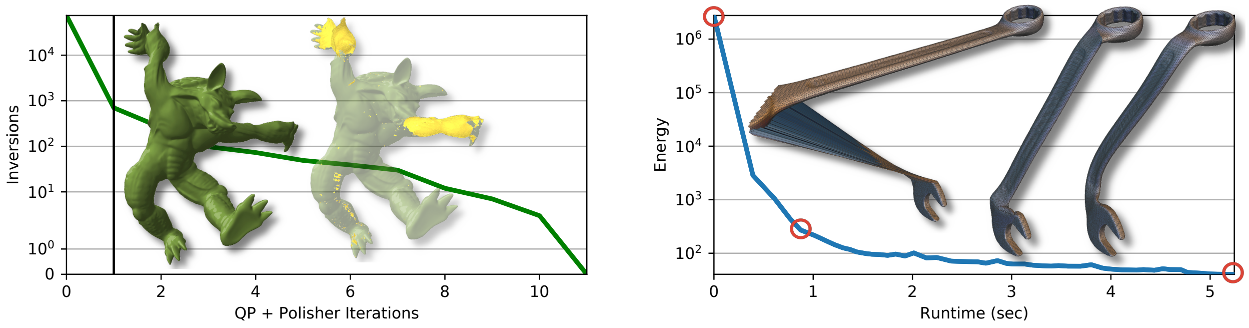


Figure 2: *Two example applications of the GINI solver. Left: GINI resolves 74k initial inversions (yellow) of a 928k element fixed boundary mesh in 9.9 seconds, with a vertical bar indicated the start of the polisher. Right: ARAP energy is minimized on a 50k element mesh, showing intermediate configurations at initialization (left), the 3rd iteration in which all inversions have been resolved (center), and 50th iteration (right).*

does not scale well to 3D due to the challenges imposed by boundary conforming tetrahedralization.

If the distortion or deformation model is nonlinear, it requires some iterative technique to minimize. Descent-based approaches are most common and require the computation of the gradient and potentially the Hessian. However, commonly used energies may result in a Hessian that is not symmetric positive definite (SPD) which is needed for stable descent. Methods that retain second-order accuracy focus on projecting the energy Hessians to SPD [TSIF05, GSC18, CW17, SGK19] or apply majorization [SPSH*17]. In geometric optimization it is also common to approximate or replace the Hessian with the mesh Laplacian [KGL16, RPPSH17], or use it as a preconditioner [CBSS17].

Infeasible Initialization Robust approaches exist for generating injective initializers for 2D problems [SJZP19, WZ14]. For this reason we focus on 3D initialization. A common approach is to project a mapping into the bounded distortion space [AL13, KABL14, KABL15]. Similarly, [FL16] project individual simplices into a distortion-bounded space and then reassemble connectivity. [SFL19] perform bounded projections with a local-global solver. One of the few methods that guarantees injective volumetric mappings is [CSZ16] but does so through mesh refinement. Most recently, [DAZ*20] introduce an energy model with injective minima, called total lifted content (TLC), that can be optimized with standard descent methods. They report a much greater success rate than previous works. We compare to TLC in Section 8.1 and show that our method is as robust for most 3D problems, but much faster. One of the few methods that does not require fixed boundaries is ABCD [NZZ20] which introduces a modified distortion energy to repair injective elements. We also compare to ABCD in Section 8.2 and show our method scales better with increased complexity. Apart from being faster, our method has another benefit over previous methods: it can attain global injectivity in problems with free boundaries.

ADMM Typically, the nature of barrier energies prohibit infeasible initialization for general nonlinear solvers. Proximal algorithms [PB14] provide a way around this restriction, in which energy or constraint evaluations are broken down into smaller, easier to solve proximal operators that can be evaluated even at infeasible points. The alternating direction method of multipliers (ADMM) is a proximal algorithm that has gained popularity in the computer graphics community [BOFN18, ZPOD19, LJ19, FLGJ19, OPY*20]. ADMM-PD [OBLN17] in particular has several attractive features, including support for arbitrary nonlinear deformation energies with or without infinite energy barriers, and fast iterations using a prefactored linear solve. Most importantly, even in the presence of barrier-type energies, it supports initial states that have inverted elements. Despite this, ADMM's ability to take non-injective steps while maintaining generality to nonlinear energies has been overlooked in the context of infeasible-start mesh optimization, perhaps because it is not guaranteed to fully satisfy constraints except at convergence. Our work alleviates that limitation and exploits the efficiency and generality offered by ADMM.

3. Method Overview

We aim to minimize energy on an input triangular or tetrahedral mesh subject to positional constraints and global injectivity. The mesh has n vertices in d-dimensional space assembled in the state variable $\mathbf{x} \in \mathbb{R}^{nd}$. A matrix $\mathbf{D}$ is used to map $\mathbf{x}$ to some energy-specific reduction (typically, the deformation gradients of all elements). The energy is expressed as a sum of per-element energies $\sum E_i(\mathbf{Dx})$ which may be elastic constitutive models (e.g., neo-Hookean) or a measure of distortion between mappings (e.g., symmetric Dirichlet). Hard positional constraints can be supported by simply removing the fixed vertices as optimization variables, while soft constraints $\mathbf{Px} = \mathbf{q}$ with user-specified stiffness κ are represented as a quadratic penalty $\frac{\kappa}{2}||\mathbf{Px}-\mathbf{q}||^2$. Then, the full problem we seek to solve is:

$$\min_{\mathbf{x}} \frac{\kappa}{2}||\mathbf{Px}-\mathbf{q}||^2 + \sum E_i(\mathbf{Dx}), \quad \text{s.t. } \mathbf{x} \text{ is globally injective.} \tag{1}$$

As discussed in Section 1, we solve this problem with a combination of a primary ADMM solver, a secondary polisher, and a

© 2021 The Author(s)
Computer Graphics Forum © 2021 The Eurographics Association and John Wiley & Sons Ltd.

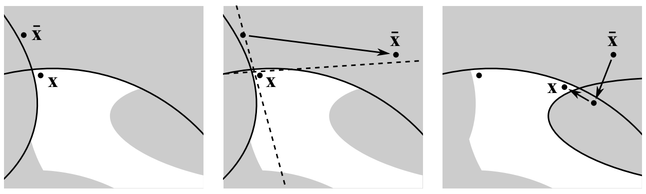


Figure 3: *An illustrative diagram of the different components of our algorithm. Left: We approximate the space of globally injective states (unshaded) using a set of constraint functions $\mathscr{C}$ (black curves). Middle: The primary ADMM solver updates $\bar{\mathbf{x}}$ using a linearization of the constraints in $\mathscr{C}$. Right: The polisher updates $\mathbf{x}$ to an intersection-free state near $\bar{\mathbf{x}}$. Meanwhile, constraints that were not active in the ADMM step are removed, and new constraints encountered by the polisher are added to $\mathscr{C}$.*

dynamically updated set of constraint functions, $\mathscr{C}$, which together act as a model of a global injectivity constraint.

Each constraint in $\mathscr{C}$ characterizes a specific potential violation of injectivity, namely inversion of a particular element or intersection between a pair of surface elements, encoded as a simple-to-evaluate constraint function $c(\mathbf{x}) \geq 0$. We maintain the invariant that if $\mathbf{x}$ is not globally injective, there is at least one constraint $c \in \mathscr{C}$ such that $c(\mathbf{x}) < 0$, which we can use to push the state towards injectivity. Thus, if all constraints in $\mathscr{C}$ have $c(\mathbf{x}) \geq 0$, the state is globally injective. The constraint set is further described in Section 6.

A unique component of our algorithm is that we maintain two separate states of the mesh, $\bar{\mathbf{x}}$ and $\mathbf{x}$. To fully exploit ADMM's strengths as a proximal method, we allow the primary solver to take large infeasible steps in $\bar{\mathbf{x}}$, only requiring it to approximately satisfy a linearization of the constraints in $\mathscr{C}$. Thus, there may exist both self-intersection and inverted elements in $\bar{\mathbf{x}}$. Then, the polisher takes $\bar{\mathbf{x}}$ and finds a nearby corrected configuration $\mathbf{x}$ that is fully intersection-free and has fewer inversions. Consequently, $\mathbf{x}$ is a state where non-penetration is always maintained. The resulting output, $\mathbf{x}$, is the output variable of the overall algorithm: the globally injective, energy minimized state. We describe the constraint polisher in Section 5.

The algorithm alternates between ADMM and polishing while simultaneously refining $\mathscr{C}$ according to the inversion- and interpenetration-prone elements in $\bar{\mathbf{x}}$. This can be done efficiently because the primary solver, being constraint-aware, gives an $\bar{\mathbf{x}}$ that is already close to injective. The rationale for two states is subtle but important: we do not want to modify ADMM's path and possibly harm optimization progress. The path from $\mathbf{x}$ to $\bar{\mathbf{x}}$ also gives us a set of linear trajectories from which to perform continuous collision detection (CCD). The constraints necessary to take $\bar{\mathbf{x}}$ to injectivity are added to $\mathscr{C}$ and used by the primary solver in subsequent iterations, while unnecessary constraints are removed. The different steps of our algorithm are illustrated in Fig. 3.

4. Approximately Injective ADMM

The primary solver that minimizes energy is based on ADMM-PD [OBLN17], which alternates between a local step that minimizes per-element energies in parallel, and a global step which solves a prefactored linear system. We extend this approach to incorporate injectivity constraints by linearizing the constraint set $\mathscr{C}$ and including them in the global step. The efficiency of ADMM-PD is retained using a fast approximate quadratic programming (QP) solver which still leverages the prefactored global matrix.

We linearize the constraint functions $c(\bar{\mathbf{x}})$ about the current ADMM state $\bar{\mathbf{x}}_{\text{curr}}$ via first order Taylor series expansion,

$$\nabla c_i^T \bar{\mathbf{x}} \geq -c_i(\bar{\mathbf{x}}_{\text{curr}}) + \nabla c_i^T \bar{\mathbf{x}}_{\text{curr}} + \eta_i, \tag{2}$$

which we collect into the form $\mathbf{C}\bar{\mathbf{x}} \geq \mathbf{d}$, with the vector inequality interpreted element-wise. We associate a constraint offset, η_i, for each constraint to represent the desired gap for collision constraints, or minimum volume for inversion constraints. This is a user-specified parameter, but if set too large it may result in problems without a feasible solution, discussed in Section 8. The goal of the primary solver, then, is to solve

$$\begin{aligned} \min_{\bar{\mathbf{x}}} \ & \frac{\kappa}{2}\|\mathbf{P}\mathbf{x}-\mathbf{q}\|^2 + \sum E_i(\mathbf{D}\bar{\mathbf{x}}), \\ \text{s.t. } & \mathbf{C}\bar{\mathbf{x}} \geq \mathbf{d}. \end{aligned} \tag{3}$$

Following [OBLN17], we introduce a consensus variable $\mathbf{z}$ and obtain the following splitting:

$$\min_{\bar{\mathbf{x}},\mathbf{z}} \ \frac{\kappa}{2}\|\mathbf{P}\bar{\mathbf{x}}-\mathbf{q}\|^2 + \sum E_i(\mathbf{z}) \tag{4a}$$

$$\text{s.t. } \mathbf{W}(\mathbf{D}\bar{\mathbf{x}}-\mathbf{z}) = \mathbf{0}, \tag{4b}$$

$$\mathbf{C}\bar{\mathbf{x}} \geq \mathbf{d}. \tag{4c}$$

The consensus constraint Eq. 4b ensures that $\mathbf{z} = \mathbf{D}\bar{\mathbf{x}}$ at the solution, where $\mathbf{W}$ is a diagonal weighting matrix chosen to improve convergence speed. This produces the following ADMM update rules:

$$\mathbf{z} \leftarrow \arg\min_{\mathbf{z}} \left(\sum E_i(\mathbf{z}) + \frac{1}{2}\|\mathbf{W}(\mathbf{D}\bar{\mathbf{x}}-\mathbf{z}+\mathbf{u})\|^2 \right), \tag{5a}$$

$$\mathbf{u} \leftarrow \mathbf{u} + \mathbf{D}\bar{\mathbf{x}} - \mathbf{z}, \tag{5b}$$

$$\bar{\mathbf{x}} \leftarrow \arg\min_{\mathbf{C}\bar{\mathbf{x}}\geq\mathbf{d}} \left(\frac{\kappa}{2}\|\mathbf{P}\bar{\mathbf{x}}-\mathbf{q}\|^2 + \frac{1}{2}\|\mathbf{W}(\mathbf{D}\bar{\mathbf{x}}-\mathbf{z}+\mathbf{u})\|^2 \right). \tag{5c}$$

The $\mathbf{z}$- and $\mathbf{u}$-updates forming the local step and the initial choice of $\mathbf{W}$ are unchanged from [OBLN17]; in particular, the local step can be carried out independently on all elements in parallel.

The global step, Eq. 5c, is equivalent to

$$\min_{\bar{\mathbf{x}}} \ \frac{1}{2}\bar{\mathbf{x}}^T\mathbf{A}\bar{\mathbf{x}} - \mathbf{b}^T\bar{\mathbf{x}} \tag{6a}$$

$$\text{s.t. } \mathbf{C}\bar{\mathbf{x}} \geq \mathbf{d}, \tag{6b}$$

where

$$\mathbf{A} = \mathbf{D}^T\mathbf{W}^T\mathbf{W}\mathbf{D} + \kappa\mathbf{P}^T\mathbf{P}, \tag{7a}$$

$$\mathbf{b} = \mathbf{D}^T\mathbf{W}^T\mathbf{W}(\mathbf{z}^{k+1}-\mathbf{u}^{k+1}) + \kappa\mathbf{P}^T\mathbf{q}. \tag{7b}$$

The Laplacian matrix (plus positional penalties), $\mathbf{A}$, is constant with unchanged mesh topology and pin indices. Thus we only need to perform factorization once during initialization. Importantly, $\mathbf{A}$ is dimensionally separable. This means that solving $\mathbf{A}\mathbf{x} = \mathbf{b}$ is equivalent to solving a much smaller $n \times n$ system $\hat{\mathbf{A}}\mathbf{X} = \mathbf{B}$, where $\mathbf{X}, \mathbf{B} \in \mathbb{R}^{n \times d}$ are reshaped copies of $\mathbf{x}, \mathbf{b}$. In large domains, this can reduce the run time of both the initial factorization and linear solves considerably. To simplify the exposition, we continue to write $\mathbf{A}$ as if it were $\mathbb{R}^{nd \times nd}$.

Eq. 6 is a standard convex QP problem: a quadratic objective with a symmetric positive definite (and notably, sparse) Hessian subject to linear inequality constraints. While QP solvers have seen recent advancements, e.g., OSQP [SBG*20] and NASOQ [CKKD20], these are expensive options requiring factorization of KKT-type matrices. We have tried both these solvers for Eq. 6, but even with careful tuning and warm starting they remain too costly for our purposes in large domains. Moreover, solving the QP to high accuracy is wasted effort for our iterations. This is because even when the linearized constraints are resolved, the original nonlinear constraints may still be violated. Additionally, the global step will almost always introduce new inversions or collisions that must be dealt with. This motivates the need for an approximate solution to Eq. 6, obtained as fast as possible, taking advantage of $\mathbf{A}$'s constant structure.

Noting that the number of constraints in Eq. 6 is invariably far smaller than the number of vertices, we work with the dual QP [NW06, p. 349], shown below for clarity. Consider the Lagrangian of Eq. 6

$$L(\bar{\mathbf{x}}, \lambda) = \frac{1}{2}\bar{\mathbf{x}}^T \mathbf{A}\bar{\mathbf{x}} - \mathbf{b}^T \bar{\mathbf{x}} - \lambda^T (\mathbf{C}\bar{\mathbf{x}} - \mathbf{d}), \tag{8}$$

with dual function $g(\lambda) = \inf_{\bar{\mathbf{x}}} L(\bar{\mathbf{x}}, \lambda)$. For a fixed λ the infimum is attained with

$$\bar{\mathbf{x}}(\lambda) = \mathbf{A}^{-1}(\mathbf{b} + \mathbf{C}^T \lambda). \tag{9}$$

Substituting this into the dual problem and minimizing the negative dual Lagrangian results in the problem

$$\min_{\lambda \geq 0} \frac{1}{2}(\mathbf{b} + \mathbf{C}^T\lambda)^T \mathbf{A}^{-1}(\mathbf{b} + \mathbf{C}^T\lambda) - \mathbf{d}^T\lambda. \tag{10}$$

Now the QP has been simplified to a box-constrained minimization on dual variable λ, with $\bar{\mathbf{x}}$ recoverable via Eq. 9.

This formulation has two significant benefits. First, the most computationally expensive component of Eq. 10 is the application of $\mathbf{A}^{-1}$, in which the matrix is already factorized and does not change with new $\mathbf{C}$ or $\mathbf{d}$. Second, to obtain fast convergence we can apply off-the-shelf L-BFGS-B implementations [ZBLN97], which require just the gradient of the dual function, $\nabla g = \mathbf{C}\bar{\mathbf{x}}(\lambda) - \mathbf{d}$. We also warm start L-BFGS-B using the value of λ cached from the previous iteration. As a result, solving the QP is efficient, robust, and easy to implement.

5. Nonlinear Constraint Polishing

After an ADMM step, $\bar{\mathbf{x}}$ mostly satisfies the *linearized* injectivity constraints present in $\mathbf{C}$, but may violate the original nonlinear constraints or ones not included in the constraint set $\mathscr{C}$. The role of the constraint polisher, similar to post-stabilization approaches [CP03] used in rigid body simulation, is to compute a nearby state $\mathbf{x}$ which

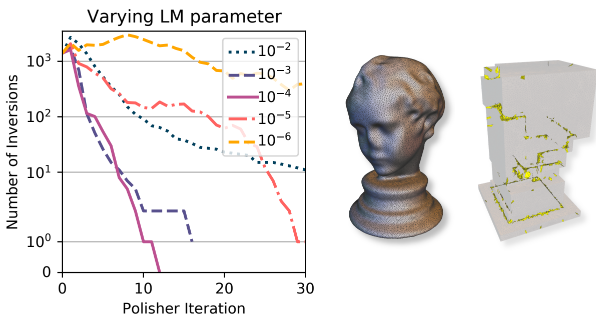


Figure 4: *A sculpture is mapped to a polycube with many initial inversions. The lower bound of γ is varied and the number of inversions is plotted over 30 polisher iterations. Smaller values of γ result in aggressive steps that may introduce more inversions, while larger values are overly damped.*

better satisfies the nonlinear injectivity constraints. Specifically, we require that $\mathbf{x}$ has no self-intersections, and all inversions have been brought closer to resolution (Section 5.1).

The constraint polisher performs its operations on a temporary variable $\hat{\mathbf{x}}$ that initialized with $\hat{\mathbf{x}} = \bar{\mathbf{x}}$. At the start of every polisher iteration, the constraint set $\mathscr{C}$ is updated and CCD queries are performed from $\mathbf{x}$ to $\hat{\mathbf{x}}$ (see Section 6.1). Viewing polishing as a feasibility problem, we define a function $h(\hat{\mathbf{x}})$ which penalizes constraint violations, then perform descent iterations to minimize it. We define a vector $\mathbf{e}(\hat{\mathbf{x}})$ of constraint evaluations with $e_i(\hat{\mathbf{x}}) = \min(0, c_i(\hat{\mathbf{x}}) - \eta_i)$ for each $c_i \in \mathscr{C}$, and set

$$h(\hat{\mathbf{x}}) = \frac{1}{2}\sum \min(0, c_i(\hat{\mathbf{x}}) - \eta_i)^2 = \frac{1}{2}\|\mathbf{e}(\hat{\mathbf{x}})\|^2, \tag{11}$$

which is zero only when all $c_i(\hat{\mathbf{x}}) \geq \eta_i$. Adopting a Gauss-Newton approach, we approximate $\mathbf{e}(\hat{\mathbf{x}} + \mathbf{p}) \approx \mathbf{e}(\hat{\mathbf{x}}) + \mathbf{J}\mathbf{p}$ for small perturbations $\mathbf{p}$, where $\mathbf{J} = \nabla\mathbf{e}(\hat{\mathbf{x}})$ is the Jacobian. The optimal $\mathbf{p}$ then satisfies $\mathbf{J}^T\mathbf{J}\mathbf{p} = -\mathbf{J}^T\mathbf{e}(\hat{\mathbf{x}})$. However, this is an underdetermined problem since there are invariably far fewer constraints than vertices. To remain as close as possible to the original $\hat{\mathbf{x}}$, we take the minimum-norm solution for $\mathbf{p}$, given by $\mathbf{p} = -\mathbf{J}^T(\mathbf{J}\mathbf{J}^T)^{-1}\mathbf{e}(\hat{\mathbf{x}})$.

As the matrix $\mathbf{J}\mathbf{J}^T$ may be ill-conditioned, we follow the Levenberg-Marquardt (LM) algorithm and regularize the linear system by adding a diagonal matrix $\mathbf{U}$. We choose a simple heuristic: a weighting that is the product of an adaptive damping parameter γ and the maximum coefficient of the diagonal of $\mathbf{J}\mathbf{J}^T$, denoted $j_{\max}$. Then, our weighting matrix is $\mathbf{U} = \mathbf{I}\gamma j_{\max}$. The constraint polisher proceeds by taking iterative steps of the following update:

$$\mathbf{p} = -\mathbf{J}^T(\mathbf{J}\mathbf{J}^T + \mathbf{U})^{-1}\mathbf{e}(\hat{\mathbf{x}}), \tag{12a}$$

$$\alpha = \text{linesearch}(\hat{\mathbf{x}}, \mathbf{p}), \tag{12b}$$

$$\hat{\mathbf{x}} = \hat{\mathbf{x}} + \alpha\mathbf{p}, \tag{12c}$$

in which Eq. 12b performs backtracking to find the largest $\alpha = (0, 1]$ such that $h(\hat{\mathbf{x}} + \alpha\mathbf{p}) < h(\hat{\mathbf{x}})$. We initialize the LM damping parameter γ to 10^{-3}, and update it after each line search using:

$$\gamma \leftarrow \begin{cases} \gamma/3 & \text{if } \alpha > 0.75, \\ 2\gamma & \text{if } \alpha < 0.25, \\ \gamma & \text{otherwise,} \end{cases} \tag{13a}$$

$$\gamma \leftarrow \text{clamp}(\gamma, [10^{-4}, 10^{-1}]). \tag{13b}$$

In practice, γ quickly reduces to 10^{-4}. We show an example of a fixed boundary mapping with different γ lower bounds in Fig. 4. As expected, larger values result in damped but stable iterations. Smaller values may result in faster resolution of the constraints, but if set too low the iterates are less stable. For our constraints, we find 10^{-4} to be a reasonable lower bound.

The Hessian approximate $\mathbf{J}\mathbf{J}^T + \mathbf{U}$ is a square matrix of the same dimension as the number of constraints. So as the constraint set grows, so does this matrix. Fortunately, the descent direction Eq. 12a depends on only a subset of the vertices in $\hat{\mathbf{x}}$. Additionally, the violated constraints are often scattered in many small regions of the mesh. This allows us to split the constraint set $\mathscr{C}$ into independent "constraint zones" and solve Eq. 12 for each zone in parallel. This is done by merging all constraints that share vertices into a single zone. For each zone, $\mathbf{J}\mathbf{J}^T + \mathbf{U}$ remains small relative to the domain and so can be practically factorized using Cholesky decomposition.

After performing a step of Eq. 12, $\hat{\mathbf{x}}$ has changed and there may be new penetrations or inversions. Immediately following the polisher iteration, previous constraints are updated, new constraints are added into new zones, and then all zones (including previous ones) are merged if they share a vertex. For any two zones that are merged, the lower γ of either zone is kept. Solving, updating, and merging zones is repeated until $\hat{\mathbf{x}}$ meets the termination criteria described in the following section. This process is similar to impact zones [Pro97, BFA02, HVTG08] popular in cloth simulation.

5.1. Constraint Polisher Termination

Each iteration of constraint polish may carry sizeable computational cost, since it involves at least one constraint set update. For 3D tasks involving overlap constraints, the broad phase of CCD can be expensive. Furthermore, for a high-resolution mesh, a loop over all elements to find inversion constraints has a non-negligible cost. Thus, it is desirable to terminate the constraint polisher as early as possible.

Note that it is unnecessary to require full global injectivity in $\mathbf{x}$ at every GINI iteration, and only a penetration-free state is needed for updating the collision constraints. However, we still want the polisher to make significant progress on resolving inversions. We find a reasonable balance and stop only once the state is penetration free and $c(\hat{\mathbf{x}}) > c(\mathbf{x})$ for all violated inversion constraints $c(\hat{\mathbf{x}}) < 0$. When this termination criterion is met, we update $\mathbf{x} = \hat{\mathbf{x}}$.

If the constraint polisher reaches some maximum number of iterations before the state is free of penetration, we leave $\mathbf{x}$ at its previous state. However, all is not lost: the polisher has spent time refining the constraint set, so the ADMM step will find a state closer to injectivity in the next GINI iteration.

6. The Constraint Set

We have shown how the primary solver minimizes energy in Section 4 and the constraint polisher finds injectivity in Section 5. Both solvers rely on the characterization of global injectivity provided by the constraint set $\mathscr{C}$. In this section, we describe in more detail how $\mathscr{C}$ represents injectivity through constraint functions $c(\mathbf{x}) \geq 0$, and how it is updated as the algorithm proceeds.

An inversion constraint is defined on each element that is close to inversion. The natural choice of constraint function $c(\mathbf{x})$ is the signed volume of the element. Overlap constraints are defined on pairs of surface primitives (vertex-face and edge-edge) that are likely to intersect, which we find through CCD as described below. For vertex-face pairs, we use the predicted intersection points $\mathbf{x}_i, \mathbf{x}_j$ and face normal $\mathbf{n}(\mathbf{x})$ to define the gap constraint $c(\mathbf{x}) = \mathbf{n}(\mathbf{x})^T(\mathbf{x}_i - \mathbf{x}_j)$. Edge-edge pairs, conversely, do not have a continuous well-defined normal. In particular, the commonly used cross product vanishes when edges are parallel, which occurs often in practice. Instead we find it effective to treat edge-edge pairs as volumetric constraints, in which an "air element" similar to [SMT08] is inserted in place of a gap constraint and its signed volume is used as $c(\mathbf{x})$. However, our approach differs from [SMT08] in that the winding order is set so that the volume is positive immediately before time of collision, and not in the collision-free state. Additionally, these "air-tets" stay within the constraint set and are updated throughout the solve.

The scaling of the constraint functions affects the convergence of both the QP solver in the ADMM step and the LM iterations in the polisher. If the gradient of one constraint is much larger in magnitude than others, it will dominate the descent direction. To mitigate this, gradients of inversion constraints are scaled by $(s/2)^{-1}$ where s is the boundary measure (perimeter or surface area) of the reference element. This ensures that the magnitudes of constraint gradients are close to unity, even if the element volumes differ by orders of magnitude. Gap constraints for vertex-face pairs are automatically well-scaled. Volumetric constraints for edge-edge pairs are scaled by surface area in the same manner as inversion constraints.

6.1. Constraint Set Update

We perform a constraint set update before each iteration of the polisher, to account for new injectivity violations introduced by the previous ADMM or polisher step. We also disable interpenetration constraints that are no longer relevant, to prevent locking artifacts.

Inversion constraints are added for all elements whose signed volume is less than its associated η (See Table 1). New collision constraints are found with standard CCD using a bounding volume hierarchy. Due to ADMM's large step sizes, the full swept volume can be quite large for a single primitive, resulting in many points of contact. To minimize the inclusion of unnecessary constraints, we find a good heuristic is to only add the first colliding pair for any given vertex, face, or edge that does not already appear in $\mathscr{C}$.

The question remains of what to do with collision constraints in $\mathscr{C}$ that are no longer colliding. We do not want to remove them because they may help prevent penetration, especially with large displacements and sliding contact. But naively considering all col-

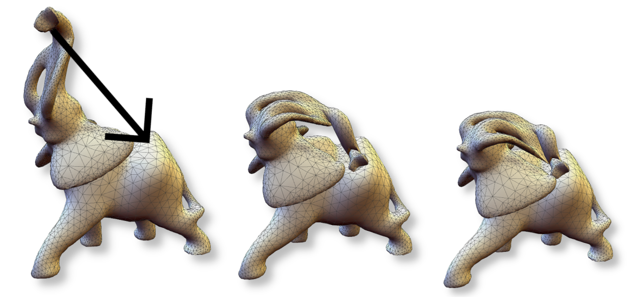
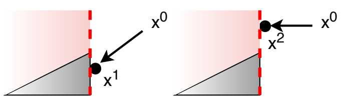


Figure 5: *An elephant deforms as its trunk is pulled downward toward its body, undergoing collisions with the tusks and ear. Intermediate iterations of the simulation are shown with and without constraint removal (center and right, respectively). When constraints are not removed, visible sticking occurs between the trunk and tusk, motivating the need for a dynamic constraint set.*

lisions constraints in Eq. 11 results in *locking* artifacts, in which the polisher iteration may become blocked from making progress.



Consider the illustration to the left. Here, a particle collides with a triangle on the first iteration ($\mathbf{x}^1$) and a constraint is created showing the restricted space in red. In the next iteration ($\mathbf{x}^2$), the particle attempts to move past the triangle but is blocked unnecessarily by the constraint.

To reduce locking artifacts, all constraints in $\mathscr{C}$ that are no longer colliding are flagged as *proximal* constraints. If any barycentric coordinate of the projection onto the splitting plane is negative for proximal constraints, it is temporarily skipped, but not removed, during the calculation of the Jacobian. Any such skipped constraint is then ignored in that iteration when calculating Eq. 11 in line search.

6.2. Constraint Removal

It is not appropriate to only add constraints, we need to remove unnecessary ones as well. These may include elements that are no longer prone to inversion, and collision constraints between two primitives separated by a third (layered contact). We perform constraint removal once per GINI iteration, immediately following the ADMM global step. The Lagrange multipliers λ calculated in Eq. 10 are valuable in finding such constraints. If $\lambda_i = 0$, the constraint is not active at $\bar{\mathbf{x}}$, and the corresponding constraint is removed from $\mathscr{C}$. We also remove collision constraints flagged as proximal if the distance between two non-colliding primitives is much greater (100η) at $\bar{\mathbf{x}}$ than $\mathbf{x}$, since this indicates the primitives are moving apart. Finally, air-tets used by edge-edge collisions may be incorrectly active, e.g., if they are counter-rotating against each other. For this reason we find it effective to remove any edge-edge constraint that was flagged as proximal from $\mathscr{C}$. Otherwise, the solver will encounter sticking artifacts as seen in Fig. 5 between the trunk and tusk in the rightmost image of the elephant.

Algorithm 1: GINI

```
Initialize $\mathbf{x}, \bar{\mathbf{x}}$
Factorize $\mathbf{A}$
Update $\mathscr{C}$ at $\mathbf{x}$ (Section 6.1)
while not injective do
    ADMM local step: update $\mathbf{z}, \mathbf{u}$ (Eqs. 5a, 5b)
    ADMM global step: update $\bar{\mathbf{x}}$ (Eq. 10)
    Remove inactive constraints from $\mathscr{C}$ (Section 6.1)
    $\hat{\mathbf{x}} = \bar{\mathbf{x}}$
    for i to max iterations do
        Update $\mathscr{C}$ from $\mathbf{x}$ to $\hat{\mathbf{x}}$ (Section 6.1)
        if criteria met (Section 5.1) then
            $\mathbf{x} = \hat{\mathbf{x}}$
            break
        end
        Form constraint zones (Section 5)
        for each zone in parallel do
            Polisher step, update $\hat{\mathbf{x}}$ (Eq. 12)
        end
    end
end
```

7. Summary

We summarize the entire algorithm in Alg. 1. Each iteration of GINI begins with one iteration of ADMM to reduce the energy subject to linearized constraints. This is followed by removal of unnecessary constraints. Finally, we perform polisher iterations until all the intersections are removed and inversions are improved. Before each polisher iteration, the constraint set is updated so that any new violations of injectivity are accounted for. This process continues until the mesh is globally injective. Alternatively, the algorithm can be run until the energy is sufficiently minimized according to user preference.

We show a table of parameters and their default values in Table 1. To the best of our ability we attempted to use the same parameters throughout all the experiments, discussed in the next section.

Parameter	Value	Exceptions
κ	$\mathbf{A}_{ii}^{max}$	Section 8.2, $10^{-5}\mathbf{A}_{ii}^{max}$
η (VF collision)	10^{-4}	-
η (volume)	$a/10$	FF-Data, see Section 8.1
max iter QP	50	-
max iter polish	30	-
min γ	10^{-4}	-

Table 1: *Parameters used in our experiments, with $\mathbf{A}_{ii}^{\max}$ = max diagonal coefficient of $\mathbf{A}$, and a = rest volume. Some mappings of the FF-Data testbed needed parameter adjustment, which we describe in Section 8.1.*

8. Evaluation and Discussion

We evaluate our approach, GINI, on many challenging problems across different scales and inputs. All our examples were computed on a desktop computer using Ubuntu 18.04LTS with a 3.5GHz Intel Xeon processor. CPU parallelization was done with OpenMP. We use Eigen [GJ*10] for matrix/vector calculations, MKL PARDISO for linear solves, and libigl [JP*18] for geometry-processing related tasks and rendering. In all examples (unless otherwise noted), the GINI solver minimizes ARAP energy and stops when the state is globally injective. As a result, the output state is feasible but may be arbitrarily distorted. If a low-distortion solution is desired, it is recommended to subsequently use a higher order optimization technique (e.g., [SPSH*17, TSIF05]) starting from GINI's output. We show an example of this in Section 8.2.

8.1. Local Injectivity with Fixed Boundaries

We first consider a set of fixed-boundary, volume-mapping problems. Here we seek to map an input tetrahedral mesh to a target shape with a fixed, prescribed boundary. The boundary points are mapped as hard constraints while methods seek inversion-free tetrahedra. We compare GINI to total lifted content (TLC) [DAZ*20] using the authors' quasi-Newton reference implementation, which is the current state of the art. Both methods stop iterating when an injective state is reached. We denote these as local injectivity tests because overlap is not explicitly resolved. Obtaining a non-inverting mesh then gives a globally injective solution.

GINI is compared to TLC on two data sets for a total of 950 mappings:

1. **FF-Data:** A reference set of mappings from [SFL19] that contains 40 polycube mappings, 20 sphere mappings, and 40 free-surface mappings (see Fig. 6). There are three different initializers for each mesh: vertices mapped to a point, vertices randomized, and harmonic initialization.
2. **Dillo-Anim:** Two data sets of a twisting armadillo at different resolutions, 23,982 and 121,190 elements, all with harmonic initialization. These were created by exporting the frame-by-frame mesh of a simulation generated using the IPC integrator [LFS*20]. The lower resolution animation contains 450 frames provided by [DAZ*20]. The higher resolution animation was generated by us and has 200 frames.

[DAZ*20] have shown that many previous methods [SFL19, KABL15, FL16] excluding their own fail to find an injective solution for some frames of the lower resolution armadillo in Dillo-Anim. Our method finds local injectivity for all of them, and indeed, for all 950 target surface mappings and initializers in our data sets. Interestingly, TLC was not able to reach injectivity on 63 of the one-point initializers in FF-Data. However, we note that one-point initialization is not practical since a harmonic initializer is always available. We still consider them because they are useful for showing sensitivity to input and for inducing many initial inversions to show how the methods scale with complexity.

Run time results are shown in Fig. 6 and Fig. 7, with data points colored based on the number of initial inversions. We see that GINI has a relatively flat scaling with problem difficulty, particularly pronounced in the FF-Data testbed. Most of the cost associated with

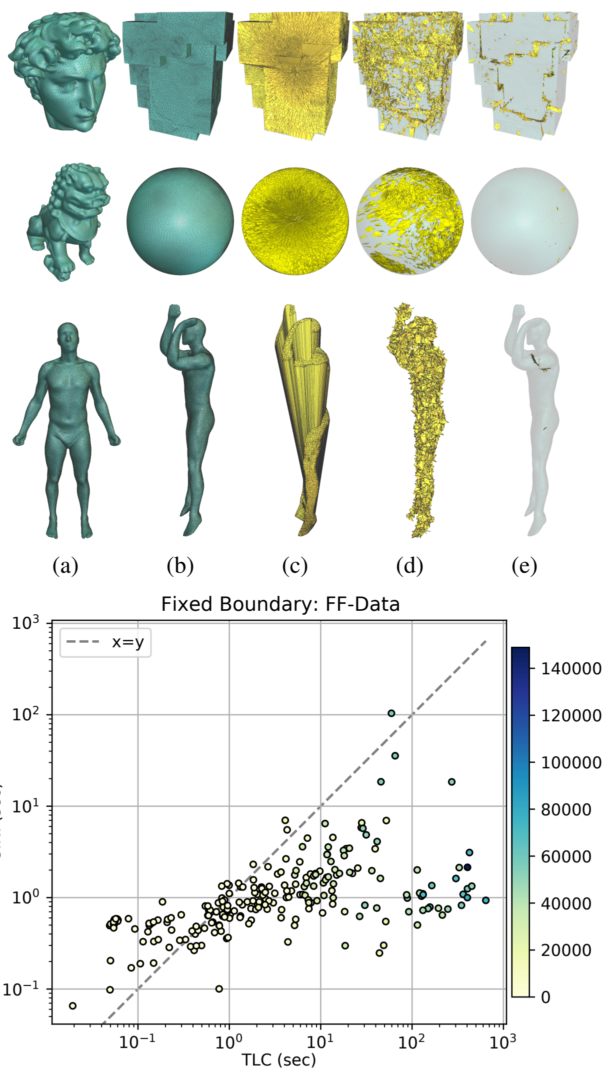


Figure 6: *The FF-Data testbed [SFL19]. Top: Example meshes with inverted elements in yellow. Each tetrahedral mesh (a) is mapped to a target surface (b) with three different initializers: one-point (c), random (d), and harmonic (e). Bottom: Comparison of GINI to TLC [DAZ*20]. Data points are colored by the number of initial inversions.*

the solve is the initial factorization. We found the factorization takes an average of 32% of the run time for FF-Data and 34% for Dillo-Anim. In contrast, TLC requires more time as the difficulty of the problem grows. For this reason, GINI tends to outperform TLC on fixed boundary volumetric mappings, sometimes by several orders of magnitude.

Caveats GINI attempts to push each element to at least its minimum volume (by default, $\eta = a/10$). A larger η allows the constraint polisher to complete sooner. However, if the target volume is not achievable, i.e., the problem itself has no feasible solution,

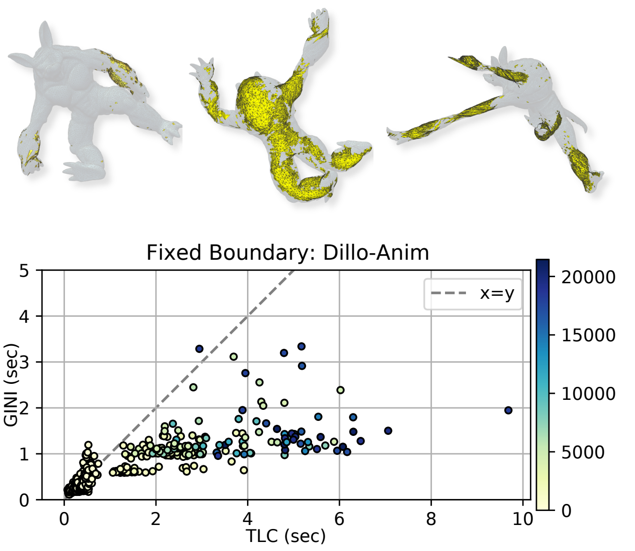


Figure 7: *Comparisons of GINI and TLC [DAZ*20] on the Dillo-Anim testbed. An armadillo at different resolutions is flung around using the IPC [LFS*20] time integrator and frames are exported to generate a database of surface mappings (top). Points are colored by initial number of inversions.*

	TLC	$a/10$	$a/100$	$a/1000$
1	2.634	1.592	0.631	0.622
2	12.999	–	1.489	1.000
3	30.882	–	–	2.239
4	56.035	–	–	6.048
5	73.203	–	–	11.872
6	83.788	–	–	–

Table 2: *A test designed to make GINI fail. A wheel is progressively smoothed resulting in near-zero volumes at the cleats, and a mapping is computed from the rest shape to the smoothed surface. The time to injectivity for TLC and GINI with varying η is listed in seconds if it succeeded. Shown is the mesh at 0, 2, and 6 smoothing iterations.*

the constraint polisher may not be able to resolve all inversions. This was the case for the sphere mappings (a total of 60 samples), in which we set $\eta = a/100$ and increased the volume of the mapped surface. Similarly, the David mesh (Fig. 6, top) required uncharacteristically specific parameters to succeed. We had to change η to 0.15 for harmonic and one-point initialization, and 0.075 for random initialization. The David mesh was the only mapping in the entire set of 950 mappings that required this level of fine tuning to succeed.

To investigate the limitation further, we devise a test specifically to stress GINI. A spikey wheel is progressively smoothed in Table 2, so that the spikes collapse to nearly zero-volume tetrahedra. TLC performs significantly slower but succeeds at more smoothing iterations. A better chosen η allows GINI to succeed up to a certain point, highlighting the need for adaptive parameters as future work.

8.2. Non-Inversion with Free Boundaries

Next, we consider free-boundary problems where we seek a non-inverting solution from an inverting initializer. The most recent work in this domain is ABCD [NZZ20], which we compare GINI to in this section. Table 3 shows the time to a non-inverting solution using the data set provided by [NZZ20] which contains both 3D (wrench, bar12k, bar30k) and 2D examples. We show one such example in Fig. 9, where an elephant mesh is initialized with randomized vertex positions. Our timings differ considerably from those reported in the original paper, possibly because we were only able to run ABCD with Eigen but not with PARDISO — nevertheless, our method outperforms most of the reported timings as well. Note that in order to perform a fair comparison, we modify GINI to also use Eigen solvers for all tests versus ABCD. In terms of time to injectivity, GINI outperforms ABCD on all examples, with a speedup of over 50x for some 3D meshes.

In our experiments we found the meshes produced by ABCD are often with lower distortion due to the increased number of iterations taken. To test how the methods perform in a practical setting, i.e., using GINI's output as an initializer for another method to optimize, we conduct the following experiment. For the higher resolution armadillo of the Dillo-Anim testbed (121,190 elements), we pin the hands in place and run GINI until the mesh is injective. Then, we switch to ABCD's core projected Newton solver and optimize until the convergence criteria is met, which is the combination of the characteristic gradient [ZBK18] and a displacement norm (see [NZZ20] for details). With ABCD, we give it the same initializer as GINI and let the method run until the same convergence criterion is met.

The results of both the time to injectivity and time to convergence (which includes time to injectivity) are show in Fig. 8. GINI greatly outperforms ABCD at reaching injectivity by several orders of magnitude. We stopped ABCD if the total time ever reaches more than 5 minutes, which occurs on 131 of the 200 examples. If the final iteration started before the 5-minute mark, we kept the solution. As seen on the left of Fig. 8, GINI's output was sufficiently low distortion so that the minimization always finished sooner than ABCD. Each projected Newton iteration taken by the optimization was approximately 30 seconds, so there is a visible clustering along the horizontal based on the number of iterations.

8.3. Globally Injective Mappings

Next, we aim to minimize elastic energies subject to positional constraints while seeking global injectivity. Here the solver is given an

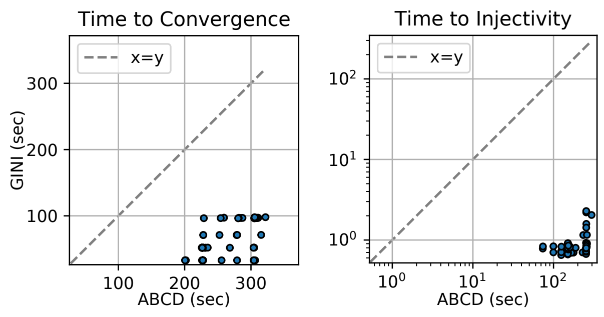


Figure 8: *Comparison of GINI to ABCD [NZZ20] on the higher resolution armadillo mesh of Dillo-Anim. Convergence is determined by ABCD's characteristic gradient and displacement norm.*

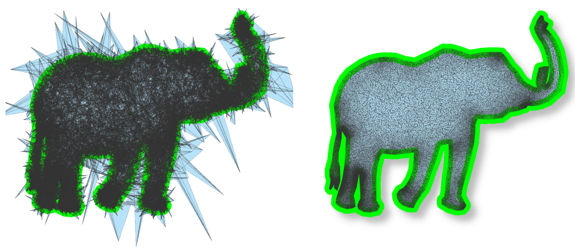


Figure 9: *An elephant mesh from [NZZ20] is initialized with random vertex positions (left) and mapped to an injective state (right) with a fixed boundary (green). The mesh has over 54k initial inverted elements and an injective solution is found in less than one second with our method.*

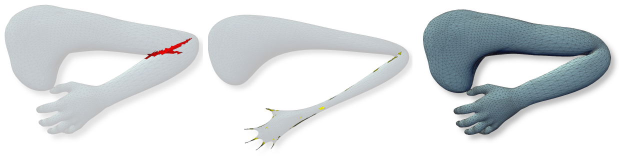


Figure 10: *An intersecting mesh is repaired with GINI. Left: The initial intersected mapping. Center: The mesh is flowed to remove intersections, but now has 3,374 inversions. Right: Stable neo-Hookean energy is minimized alongside resolving collisions and inversions as the mesh is pulled back to the target shape.*

input volumetric mesh and a target boundary surface, and should output a final mesh, free of inversion and self-penetration. We add soft constraints pulling the boundary vertices towards the corresponding points on the target surface, with a stiffness of $\kappa = 10^{-5}\mathbf{A}_{ii}^{\max}$. In this problem we also allow the constraint polisher to move these soft-constrained vertices to resolve intersections.

We follow the same initial process as [SJP*13] to start the target mapping in an intersection-free state, shown in Fig. 1. The mapped surface mesh is flowed using conformalized mean curvature flow (cMCF) [KSBC12] until it is free of intersection. Then, we compute a harmonic initializer in the flowed state, preserving the tetrahedralization of the input volumetric mesh. After that, we start the optimization. GINI uses quadratic constraints to pull the surface close to its mapped shape, while minimizing elastic energy and satisfying global injectivity.

The first example is a neo-Hookean material hand model, Fig. 1, shaped to form a fist. The mesh has 31,197 tetrahedra, 7,256 vertices, and 9,556 faces. GINI was able to achieve global injectivity on the second iteration, taking a total of 19.0 seconds, with only one inversion remaining after the first iteration. We plot the number of inversions and triangle-triangle intersections over the first iteration in Fig. 1. Many initial constraints are resolved by the QP solver while the energy is minimized. The deformation introduces new collisions which can be seen over the first 50 QP solver iterations. Once the constraint polisher starts, indicated by a vertical line, both the number of active inversions and collisions rapidly decrease. The most costly component of the simulation was collision detection that occurred during constraint polish, taking a total of 17.2 seconds. Next was the global step, which took 1.07 seconds.

We show another example with a stable neo-Hookean material [SGK18] arm bending at the elbow in Fig. 10. The mesh has 39,292 tetrahedra, 10,750 vertices, and 16,618 faces. Bending at the elbow causes intersections between the upper and lower arm, which are removed by flowing the surface. After one GINI iteration taking 47.7 seconds, the arm is free of inversions and self-intersection. Like the hand mesh, collision detection takes the most time with a total of 47.2 seconds. This is largely due to the number of broad phase candidates encountered at the fingers which were collapsed to thin strands.

To further test our algorithm on resolving challenging collision scenarios, we reuse the higher resolution examples in Dillo-Anim. For each of the 200 meshes, we pin the hands in place and run GINI until a globally injective state is reached using the stable neo-

Mesh	#V	#E	#I	ABCD	GINI
elephant-low	1105	1796	786	0.102	**0.051**
elephant-med	15,193	28,736	14,043	2.223	**0.173**
elephant-high	59,121	114,944	54,906	25.155	**0.533**
octopus-low	3,924	5,986	23	0.330	**0.106**
octopus-med	13,833	23,944	95	1.771	**0.715**
octopus-high	51,609	95,776	1,007	19.329	**3.067**
d1-01121	49,274	95,880	3	0.653	**0.275**
d1-02392	47,535	91,912	25	5.063	**0.291**
d1-00478	25,293	48,964	22	0.685	**0.175**
gorilla	420,408	839,092	20	126.025	**7.522**
wrench	14,798	50,122	322	58.242	**0.669**
bar12k	2,541	12,000	4,726	15.871	**0.179**
bar30k	6,171	30,000	104	26.988	**0.444**

Table 3: *Examples from the ABCD-Data testbed [NZZ20] with partially fixed boundaries. Reported is number of vertices #V, number of elements #E, number of initial inverted elements #I, and time to injectivity (seconds) for ABCD and GINI. In problems with a large number of initial inversions, the speedup is often over* 10× *(green).*

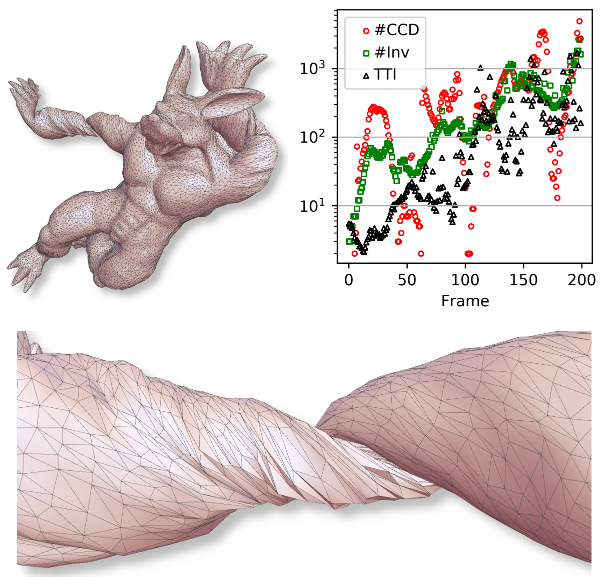


Figure 11: *The Dillo-Anim testbed contains challenging initializers. When only the hands of the armadillo are fixed, minimizing elastic energy results in many collisions, especially near the twisted arms (bottom). The number of collisions and inversions resolved by our algorithm in each example are plotted with the time to injectivity (TTI) in seconds.*

Hookean energy. The number of collisions and inversions encountered by the QP solver are shown in Fig. 11, along with the time to injectivity (TTI) measured in seconds. GINI reaches injectivity for all examples; here, in several cases, there were over 3,000 collision constraints being resolved. The bottom image of Fig. 11 shows the twisted arm after the mesh has been made globally injective by our algorithm.

8.4. Stress Test

To stress test our algorithm further we compress a bunny mesh (32,126 tetrahedra, 8,112 vertices, 9,790 faces) with an ever-shrinking cube using the ARAP energy. Every time the polisher succeeds at finding a globally injective state, the cube is then further compressed. Here this benchmark test will eventually exit once the polisher fails three times consecutively. We show three states of the optimization in Fig. 12. Our algorithm was able to compress the bunny to 19.6% of its original volume after 126 seconds of run time. Collision detection was the most expensive part of the solve, which took 66.1 seconds of the simulation. The next most expensive component was the QP solve, which took 22.9 seconds. In this test case the constraint set grew to 14,298 active constraints in the last successful iteration. Eventually, the solver was unable to make progress because some faces were compressed to zero area and so constraints could not be evaluated.

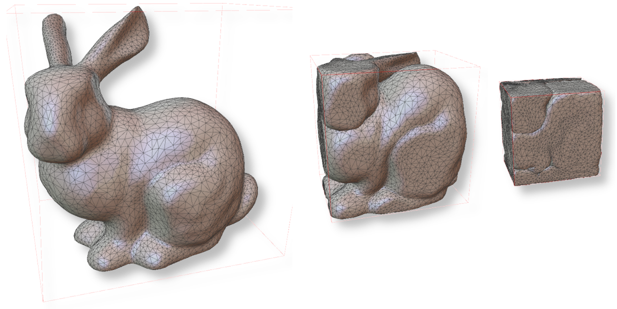


Figure 12: *A tetrahedral mesh is progressively compressed in a box to stress test our algorithm. GINI resolves inversions and collisions down to 19.6% of its original volume.*

8.5. Limitations and Future Work

Our focus in this work is not solely on robustness. We attempt a balance between generality with respect to energy models and scalability for higher resolution meshes. Indeed, like most existing map fixers, GINI is not guaranteed to find an injective state (see Table 2). Additionally, we tested the rotating cube cutout example from the concurrent work of [GKK*21] (Fig. 6 of their paper) and GINI was only able to solve the 45-degree rotation. As evidenced by Fig. 4 and Table 2, parameter adjustment can improve the success rate. In future work, we expect investigation of an automatic approach to adjusting parameters will make our solver more robust.

As shown in Section 8.2, collision detection is often the most expensive component of our solves. For meshes with surfaces that are initialized with enough distance (e.g., Fig. 10) coarse bounding meshes such as those employed by Jiang et al. [JSZP20] can be used to speed up the collision processing. However, a thorough analysis of different methods for collision detection is outside the scope of this work. Additionally, we are unsure if coarse boundings can be applicable to all our benchmarks, specifically the armadillo, which has exceedingly small initial separation distances. Examining faster methods to process the overlap constraint would likewise be a productive research direction.

Finally, we would like to apply our algorithm to texture mapping. As GINI can start from infeasible configurations, there is more flexibility for low-distortion initialization compared to maintenance-based methods. We have tried this and found that GINI works well even for high-resolution meshes. For example, Fig. 13 shows our method starting from a non-injective harmonic initialization and produces a globally injective result after the first iteration. Unfortunately, certain meshes exhibit "pinching" issues, where some boundary triangles are compressed to near-degeneracy. From our experiments we find that dynamically increasing or decreasing ADMM weights can help resolve pinching, but then correspondingly require a numerical refactorization of the global matrix. We plan to investigate improvements for bijective texture mapping in the future.

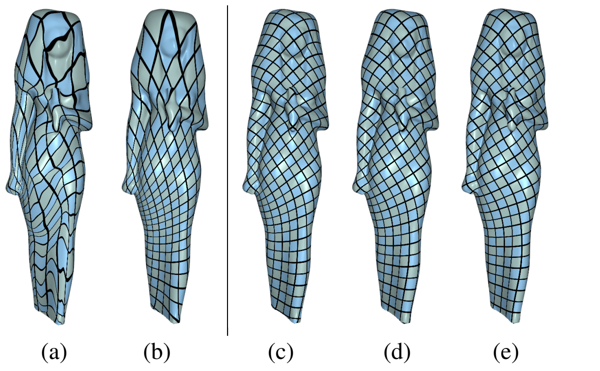


Figure 13: *Texture parameterization with symmetric Dirichlet. Since GINI can take non-injective steps, it does not have to start from a Tutte embedding (a) and can instead start from a harmonic one (b) that has 25 inversions but lower distortion. An injective map is found immediately on the first iteration (c). The third (d) and seventh (e) iteration are also shown. Each iteration takes an average of 8.4 seconds for a mesh with 1,357,857 vertices and 2,689,024 triangles.*

9. Conclusion

Geometric optimization subject to global injectivity is a challenging problem and is important to computer graphics. A standing limitation of many existing solvers is that they are restricted to feasible iterates and starting points, which can be difficult to produce and impede solver progress. Map fixers can be used to generate injective initializers, but are often slow or may fail to produce valid results. Additionally, no previous map fixer is capable of maintaining non-penetration with free boundaries. Methods such as ADMM are not limited to feasible iterates and may take large, inexpensive steps toward a constrained objective, but have difficulty fully satisfying constraints. We show how global injectivity can be approximated using locally defined, incrementally constructed inequality constraints for inversion and overlap. We build on ADMM to resolve these constraints using an approximate QP solver that benefits from a constant system matrix, while still maintaining the benefits of fast, non-injective intermediate steps. This is coupled with a secondary solver that rapidly attains global injectivity and updates the set of constraints used in subsequent iterations. The resulting algorithm outperforms the state of the art on a large testbed of complex meshes, in some cases by multiple orders of magnitude.

Acknowledgments This work was funded in part by Adobe. We would like to thank the reviewers for their useful feedback, George Brown for the helpful discussions, and François Protais for supplying the rotated cube test case. The spiked wheel by Thingiverse user Priit Kallas is licensed under CC BY-NC 4.0.

References

[AL13] Aigerman N., Lipman Y.: Injective and bounded distortion mappings in 3d. *ACM Trans. Graph. 32*, 4 (July 2013). 3

[BFA02] Bridson R., Fedkiw R., Anderson J.: Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph. 21*, 3 (July 2002), 594–603. 6

[BOFN18] Brown G. E., Overby M., Forootaninia Z., Narain R.: Accurate dissipative forces in optimization integrators. *ACM Trans. Graph. 37*, 6 (Dec. 2018). 3

[BW97] Bonet J., Wood R. D.: *Nonlinear continuum mechanics for finite element analysis*. Cambridge university press, 1997. 2

[CBSS17] Claici S., Bessmeltsev M., Schaefer S., Solomon J.: Isometry-aware preconditioning for mesh parameterization. *Comput. Graph. Forum 36*, 5 (Aug. 2017), 37–47. 3

[CKKD20] Cheshmi K., Kaufman D. M., Kamil S., Dehnavi M. M.: Nasoq: Numerically accurate sparsity-oriented qp solver. *ACM Trans. Graph. 39*, 4 (July 2020). 5

[CP03] Cline M. B., Pai D. K.: Post-stabilization for rigid body simulation with contact and constraints. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)* (2003), vol. 3, pp. 3744–3751 vol.3. 5

[CSZ16] Campen M., Silva C. T., Zorin D.: Bijective maps from simplicial foliations. *ACM Trans. Graph. 35*, 4 (July 2016). 3

[CW17] Chen R., Weber O.: Gpu-accelerated locally injective shape deformation. *ACM Trans. Graph. 36*, 6 (Nov. 2017). 3

[DAZ*20] Du X., Aigerman N., Zhou Q., Kovalsky S. Z., Yan Y., Kaufman D. M., Ju T.: Lifting simplices to find injectivity. *ACM Trans. Graph. 39*, 4 (July 2020). 3, 8, 9

[EDD*95] Eck M., DeRose T., Duchamp T., Hoppe H., Lounsbery M., Stuetzle W.: Multiresolution analysis of arbitrary meshes. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1995), SIGGRAPH '95, Association for Computing Machinery, p. 173–182. 2

[FL16] Fu X.-M., Liu Y.: Computing inversion-free mappings by simplex assembly. *ACM Trans. Graph. 35*, 6 (Nov. 2016). 3, 8

[FLG15] Fu X.-M., Liu Y., Guo B.: Computing locally injective mappings by advanced mips. *ACM Trans. Graph. 34*, 4 (July 2015). 2

[FLGJ19] Fang Y., Li M., Gao M., Jiang C.: Silly rubber: An implicit material point method for simulating non-equilibrated viscoelastic and elastoplastic solids. *ACM Trans. Graph. 38*, 4 (July 2019). 3

[Fun13] Fung Y.-c.: *Biomechanics: mechanical properties of living tissues*. Springer Science & Business Media, 2013. 2

[GJ*10] Guennebaud G., Jacob B., et al.: Eigen v3. http://eigen.tuxfamily.org, 2010. 8

[GKK*21] Garanzha V., Kaporin I., Kudryavtseva L., Protais F., Ray N., Sokolov D.: Foldover-free maps in 50 lines of code, 2021. 11

[GSC18] Golla B., Seidel H.-P., Chen R.: Piecewise linear mapping optimization based on the complex view. *Computer Graphics Forum 37*, 7 (2018), 233–243. 3

[HVTG08] Harmon D., Vouga E., Tamstorf R., Grinspun E.: Robust treatment of simultaneous collisions. In *ACM SIGGRAPH 2008 Papers* (New York, NY, USA, 2008), SIGGRAPH '08, Association for Computing Machinery. 6

[ITF04] Irving G., Teran J., Fedkiw R.: Invertible finite elements for robust simulation of large deformation. In *Proc. ACM SIGGRAPH/Eurographics SCA* (2004), SCA, Eurographics Association, pp. 131–140. 2

[JP*18] Jacobson A., Panozzo D., et al.: libigl: A simple C++ geometry processing library, 2018. https://libigl.github.io/. 8

[JSP17] Jiang Z., Schaefer S., Panozzo D.: Simplicial complex augmentation framework for bijective maps. *ACM Trans. Graph. 36*, 6 (Nov. 2017), 186:1–186:9. 2

[JSZP20] Jiang Z., Schneider T., Zorin D., Panozzo D.: Bijective projection in a shell. *ACM Trans. Graph. 39*, 6 (Nov. 2020). 11

[KABL14] Kovalsky S. Z., Aigerman N., Basri R., Lipman Y.: Controlling singular values with semidefinite programming. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH) 33*, 4 (2014). 3

[KABL15] KOVALSKY S. Z., AIGERMAN N., BASRI R., LIPMAN Y.: Large-scale bounded distortion mappings. *ACM Trans. Graph. 34*, 6 (Oct. 2015). 3, 8

[KGL16] KOVALSKY S. Z., GALUN M., LIPMAN Y.: Accelerated quadratic proxy for geometric optimization. *ACM Trans. Graph. 35*, 4 (July 2016). 3

[KSBC12] KAZHDAN M., SOLOMON J., BEN-CHEN M.: Can mean-curvature flow be modified to be non-singular? *Comput. Graph. Forum 31*, 5 (Aug. 2012), 1745–1754. 1, 10

[LFS*20] LI M., FERGUSON Z., SCHNEIDER T., LANGLOIS T., ZORIN D., PANOZZO D., JIANG C., KAUFMAN D. M.: Incremental potential contact: Intersection- and inversion-free large deformation dynamics. *ACM Transactions on Graphics 39*, 4 (2020). 8, 9

[LJ19] LIU H.-T. D., JACOBSON A.: Cubic stylization. *ACM Trans. Graph. 38*, 6 (Nov. 2019), 197:1–197:10. 3

[MCKM15] MÜLLER M., CHENTANEZ N., KIM T.-Y., MACKLIN M.: Air meshes for robust collision handling. *ACM Trans. Graph. 34*, 4 (July 2015). 2

[Moo40] MOONEY M.: A theory of large elastic deformation. *Journal of applied physics 11*, 9 (1940), 582–592. 2

[NW06] NOCEDAL J., WRIGHT S. J.: *Numerical Optimization*, 2 ed. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006. 5

[NZZ20] NAITSAT A., ZHU Y., ZEEVI Y. Y.: Adaptive block coordinate descent for distortion optimization. *Computer Graphics Forum 39*, 6 (2020), 360–376. 2, 3, 9, 10

[OBLN17] OVERBY M., BROWN G. E., LI J., NARAIN R.: Admm $\supseteq$ projective dynamics: Fast simulation of hyperelastic models with dynamic constraints. *IEEE Transactions on Visualization and Computer Graphics 23*, 10 (Oct 2017), 2222–2234. 2, 3, 4

[OPY*20] OUYANG W., PENG Y., YAO Y., ZHANG J., DENG B.: Anderson acceleration for nonconvex admm based on douglas-rachford splitting. *Computer Graphics Forum 39*, 5 (2020), 221–239. 3

[PB14] PARIKH N., BOYD S.: Proximal algorithms. *Found. Trends Optim. 1*, 3 (Jan. 2014), 127–239. 3

[Pro97] PROVOT X.: Collision and self-collision handling in cloth model dedicated to design garments. In *Computer Animation and Simulation*. Springer, 1997, pp. 177–189. 6

[Riv97] RIVLIN R. S.: Some applications of elasticity theory to rubber engineering. In *Collected papers of RS Rivlin*. Springer, 1997, pp. 9–16. 2

[RPPSH17] RABINOVICH M., PORANNE R., PANOZZO D., SORKINE-HORNUNG O.: Scalable locally injective mappings. *ACM Trans. Graph. 36*, 2 (Apr. 2017). 2, 3

[SBG*20] STELLATO B., BANJAC G., GOULART P., BEMPORAD A., BOYD S.: OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation 12*, 4 (2020), 637–672. 5

[SFL19] SU J.-P., FU X.-M., LIU L.: Practical foldover-free volumetric mapping construction. *Computer Graphics Forum 38*, 7 (2019), 287–297. 2, 3, 8

[SGK18] SMITH B., GOES F. D., KIM T.: Stable neo-hookean flesh simulation. *ACM Trans. Graph. 37*, 2 (Mar. 2018). 10

[SGK19] SMITH B., GOES F. D., KIM T.: Analytic eigensystems for isotropic distortion energies. *ACM Trans. Graph. 38*, 1 (Feb. 2019). 3

[SHST12] STOMAKHIN A., HOWES R., SCHROEDER C., TERAN J. M.: Energetically consistent invertible elasticity. In *Proc. ACM SIGGRAPH/Eurographics SCA* (2012), SCA, Eurographics Association, pp. 25–32. 2

[SJP*13] SACHT L., JACOBSON A., PANOZZO D., SCHÜLLER C., SORKINE-HORNUNG O.: Consistent volumetric discretizations inside self-intersecting surfaces. *Computer Graphics Forum (proceedings of EUROGRAPHICS/ACM SIGGRAPH Symposium on Geometry Processing) 32*, 5 (2013), 147–156. 10

[SJZP19] SHEN H., JIANG Z., ZORIN D., PANOZZO D.: Progressive embedding. *ACM Trans. Graph. 38*, 4 (July 2019). 3

[SKPSH13] SCHÜLLER C., KAVAN L., PANOZZO D., SORKINE-HORNUNG O.: Locally injective mappings. In *Proceedings of the Eleventh Eurographics/ACMSIGGRAPH Symposium on Geometry Processing* (Goslar, DEU, 2013), SGP '13, Eurographics Association, p. 125–135. 2

[SMT08] SIFAKIS E., MARINO S., TERAN J.: Globally coupled collision handling using volume preserving impulses. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Goslar, DEU, 2008), SCA '08, Eurographics Association, p. 147–153. 6

[SPSH*17] SHTENGEL A., PORANNE R., SORKINE-HORNUNG O., KOVALSKY S. Z., LIPMAN Y.: Geometric optimization via composite majorization. *ACM Trans. Graph. 36*, 4 (July 2017). 3, 8

[SS15] SMITH J., SCHAEFER S.: Bijective parameterization with free boundaries. *ACM Trans. Graph. 34*, 4 (July 2015), 70:1–70:9. 2

[SYLF20] SU J.-P., YE C., LIU L., FU X.-M.: Efficient bijective parameterizations. *ACM Trans. Graph. 39*, 4 (July 2020). 2

[TPBF87] TERZOPOULOS D., PLATT J., BARR A., FLEISCHER K.: Elastically deformable models. In *Proc. Conf. on Comp. Graph. and Interactive Tech.* (1987), SIGGRAPH, ACM, pp. 205–214. 2

[TSIF05] TERAN J., SIFAKIS E., IRVING G., FEDKIW R.: Robust quasistatic finite elements and flesh simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2005), SCA '05, Association for Computing Machinery, p. 181–190. 3, 8

[Tut63] TUTTE W. T.: How to draw a graph. *Proceedings of the London Mathematical Society 3*, 1 (1963), 743–767. 2

[WZ14] WEBER O., ZORIN D.: Locally injective parametrization with arbitrary fixed boundaries. *ACM Trans. Graph. 33*, 4 (July 2014). 3

[ZBK18] ZHU Y., BRIDSON R., KAUFMAN D. M.: Blended cured quasi-newton for distortion optimization. *to appear ACM Trans. on Graphics (SIGGRAPH 2018)* (2018). 2, 9

[ZBLN97] ZHU C., BYRD R. H., LU P., NOCEDAL J.: Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw. 23*, 4 (Dec. 1997), 550–560. 5

[ZMT05] ZHANG E., MISCHAIKOW K., TURK G.: Feature-based surface parameterization and texture mapping. *ACM Trans. Graph. 24*, 1 (Jan. 2005), 1–27. 2

[ZPOD19] ZHANG J., PENG Y., OUYANG W., DENG B.: Accelerating admm for efficient simulation and optimization. *ACM Trans. Graph. 38*, 6 (Nov. 2019). 3