Guaranteed Globally Injective 3D Deformation Processing

YU FANG*, University of California, Los Angeles & University of Pennsylvania & Adobe Research
MINCHEN LI*, University of California, Los Angeles & University of Pennsylvania
CHENFANFU JIANG, University of California, Los Angeles & University of Pennsylvania
DANNY M. KAUFMAN, Adobe Research

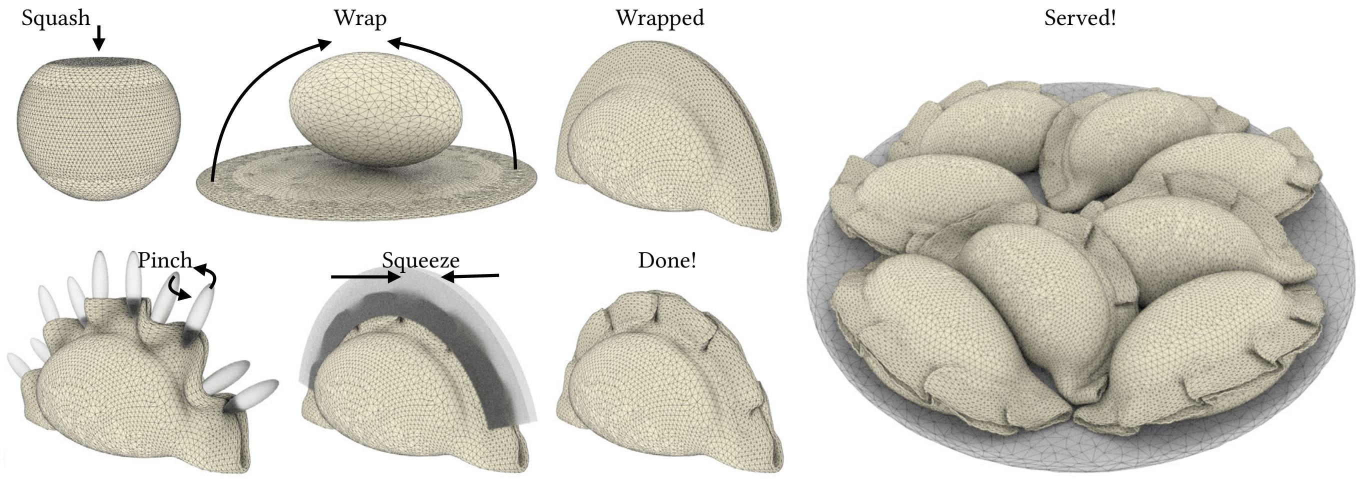

Fig. 1. **Making Dumplings.** Injective Deformation Processing (IDP) enables reliable and efficient deformation with a guarantee of injectivity for a wide range of tasks. Here, applied for modeling and layout with a tetrahedral mesh, we can emulate manual steps with IDP to directly form a complex model from simple primitives in just a few steps. Once complete, we copy-paste the model. Then layout, with non-intersection enforced, provides natural variation for the copies as we nudge the plated dumplings for a final arrangement.

We extend recent advances in the numerical time-integration of contacting elastodynamics [Li et al. 2020] to build a new framework, called Injective Deformation Processing (IDP), for the robust solution of a wide range of mesh deformation problems requiring injectivity. IDP solves challenging 3D (and 2D) geometry processing and animation tasks on meshes, via artificial time integration, with guarantees of both non-inversion and non-overlap. To our knowledge IDP is the first framework for 3D deformation processing that can efficiently guarantee globally injective deformation without geometric locking. We demonstrate its application on a diverse set of problems and show its significant improvement over state-of-the-art for globally injective 3D deformation.

CCS Concepts: • **Computing methodologies** → **Shape modeling**; • **Mathematics of computing** → *Continuous optimization.*

*Joint first authors

Authors' addresses: Yu Fang, University of California, Los Angeles & University of Pennsylvania & Adobe Research, squarefk@gmail.com; Minchen Li, University of California, Los Angeles & University of Pennsylvania, minchernl@gmail.com; Chenfanfu Jiang, University of California, Los Angeles & University of Pennsylvania, chenfanfu.jiang@gmail.com; Danny M. Kaufman, Adobe Research, danny.kaufman@gmail.com.

0730-0301/2021/8-ART75 $15.00
https://doi.org/10.1145/3450626.3459757

Additional Key Words and Phrases: deformation, global injectivity, distortion optimization, modeling, animation, numerical optimization

ACM Reference Format:
Yu Fang, Minchen Li, Chenfanfu Jiang, and Danny M. Kaufman. 2021. Guaranteed Globally Injective 3D Deformation Processing. *ACM Trans. Graph.* 40, 4, Article 75 (August 2021), 13 pages. https://doi.org/10.1145/3450626.3459757

1 INTRODUCTION

A wide array of geometry processing and animation tasks boil down to minimizing a deformation objective while seeking to achieve both targeted boundary conditions *and* injectivity. If we loosen our requirements to satisfying just local injectivity, *or* else to focus solely on 2D problems, then recent advances in distortion optimization offer many promising solutions [Jiang et al. 2017; Kovalsky et al. 2016; Schüller et al. 2013; Smith and Schaefer 2015; Su et al. 2020; Zhu et al. 2018].

However, when it comes to optimizing 3D mesh deformations with global injectivity, very few options are available. At the same time, existing methods come with strong limitations on performance and, perhaps even more restrictive, significant modes of failure.

In summary (we detail the work in Section 3), there is currently an undesirable trade-off between reliability and expressiveness in computing globally injective deformations in 3D. On the one hand, many methods apply iterative contact-processing modules from physics-based animation [Bridson et al. 2002] to resolve intersections in geometry processing. These methods are efficient and easily support large deformations. However, this efficiency and expressiveness come at the cost of reliability: intersections are rarely fully resolved (convergence) while inversions are generally ignored. On the other hand, recent attention has also focused on applying fictitious domain methods from computational mechanics [Pagano and Alart 2008] to reliably provide bijective maps [Jiang et al. 2017]. To enforce bijectivity, negative space is separately discretized by a compatible triangulation. However, without refinement, this triangulation introduces increasingly severe locking from sheared elements as it distorts with the primary mesh. While in 2D this locking can be alleviated by global remeshing, in 3D it cannot (tractability); and here local remeshing [Jiang et al. 2017; Müller et al. 2015] still suffers from severe locking.

Towards providing a robust, reliable, general-purpose solution we follow this well-worn path from physics modeling to geometry processing. We port and extend Incremental Potential Contact (IPC) [Li et al. 2020], a recent method for simulating contacting elastodynamics, to enable globally injective optimization of deforming meshes with guarantees of both non-overlap and non-inversion. This enables us to build a new framework called Injective Deformation Processing (IDP) for the robust solution of a wide range of 3D and 2D optimization problems on meshes requiring injectivity and satisfaction of boundary conditions.

1.1 Contributions

IDP is a general-purpose framework for deformation processing focusing on expressiveness and reliability with guarantees of stability, non-interpenetration and, when desired and appropriate, non-inversion on both triangular and tetrahedral meshes. To our knowledge IDP is the first framework for 3D deformation processing that can efficiently guarantee globally injective deformation without geometric locking. To formulate IDP our contributions include:

- A simple and direct extension of the IPC model from elastodynamics to the artificial time integration solution of a wide range of deformation problems requiring injectivity;
- A comprehensive comparison with the 3D state-of-the-art on a new benchmark set of stress-test examples demonstrating IDP's advances – we show IDP enables results and guarantees not achievable with existing methods; and
- Demonstration and analysis of applications this enables on a range of practical and challenging geometry processing and animation tasks.

2 PROBLEM STATEMENT

We phrase a wide range of geometric tasks on meshes in terms of a simple optimization framework. We begin each task with a reference triangulation T (triangles or tetrahedra depending on application) with n vertex locations in d-dimensional space stored in vector $\bar{x} \in \mathbb{R}^{dn}$ and a corresponding starting geometry $x^0 \in \mathbb{R}^{dn}$.

Our most simple task is to evolve the mesh from x^0 to a local minimizer of a generally nonlinear and nonconvex energy $E(x, \bar{x}) = E_{\bar{x}}(x)$ defined w.r.t. $\bar{x}$. Solutions are x^* satisfying $\|\nabla E_{\bar{x}}(x^*)\| \leq \epsilon$.

Boundary Conditions. Full or partial constraints on the boundary geometry can be imposed. Our task then extends to evolving x^0 to a geometry $x^* = \text{argmin}_x E_{\bar{x}}(x)$ s.t. x satisfying all boundary conditions[1].

Injectivity. When we additionally impose *injectivity* constraints we require the path traversed from x^0 to x^* be non-inverting and/or nonoverlapping. A path that is both non-inverting and non-overlapping gives a globally injective deformation.

Time-varying conditions. Finally, in many tasks we consider the optimization further parameterized with additional terms u, so that we minimize $E_{\bar{x}}(x, u)$. Here parameters u can vary over the optimization process during evolution from x^0 to x^*, parameterized in turn by artificial time variable $t \in \mathbb{R}$. "Time-varying" u enable a rich range of applications and optimization tasks including scripted boundary conditions (e.g. for animation), homotopies of geometry and/or energy, and even online optimization to provide user-in-the-loop interaction.

3 RELATED WORK

While IDP is also an effective tool for globally injective deformation in 2D (see Section 7.4), a range of efficient methods have been recently developed [Jiang et al. 2017; Smith and Schaefer 2015; Su et al. 2020] for this task. Likewise, we note that for applications where a 3D target domain's boundary is *fully* fixed, recently developed methods [Du et al. 2020; Kovalsky et al. 2015] robustly and efficiently obtain bijective maps. Here we target IDP primarily on underserved 3D tasks and so focus here on *3D* globally injective deformation processing efforts and related advances.

Computer graphics has enjoyed a friendly sharing of techniques across borders between physics simulation and geometry processing; see e.g., Bouaziz et al. [2012] and Bouaziz et al. [2014]. This has lent itself to complementary advances and useful borrowings. Most germane to our discussion are a number of intertwined developments in injective mapping and contact mechanics. Broadly, methods seeking globally injective 3D deformation with partial or fully free boundaries have so far followed one of two routes.

In the first route, iterative physics-based collision-response methods [Bridson et al. 2002; Harmon et al. 2008] have been extended for geometry processing tasks [Brochu and Bridson 2009; Harmon et al. 2011; Sacht et al. 2015]. These methods successively step, detect intersection and then correct with iterative sweeps. Modules for these separate components are well-polished (and so readily available and easy to adapt), often parallelized and generally very efficient. However, as touched upon above, these solvers generally do not handle inversion, have no guarantees of convergence and, in practice, generally cannot and do not resolve all interpenetrations [Li et al. 2020].

In turn, even small interpenetrations in meshes lead to unacceptable errors that vary with application, including intersecting animations, unsightly rendering artifacts, downstream cloth simulation failures, and geometries that cannot be 3D-printed. To redress these failures many pipelines in physics have been further augmented with failsafes and correctives [Baraff et al. 2003; Buffet et al. 2019; Volino and Magnenat-Thalmann 2006; Wicke et al. 2006; Wong

[1]Here and throughout argmin and min denote respectively local minimizer and minimum.

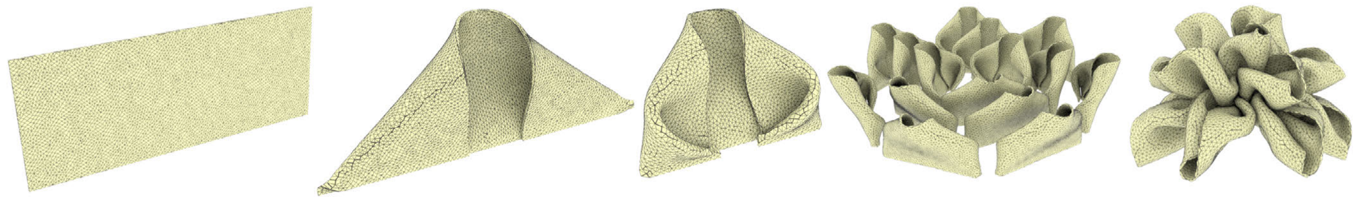
Fig. 2. **Rhododendron.** We model a Rhododendron pattern starting with just a thin box that we fold to form a single petal. Instancing and arranging the folded shape, and then pulling the copies together with contact creates natural variations in the petals and forms our final flower on the right.

et al. 2018; Ye et al. 2017, 2015]. While often helpful, these methods in turn incur cost, complexity and often have unintended consequences. Corrections often distort the geometry with undesirable deformations which also introduce new, subsequent artifacts such as inversion, that require further clean up.

In the second route, fictitious domain methods from computational mechanics, see e.g., Pagano and Alart [2008], have been extended for both physics-based animation [Misztal and Bærentzen 2012; Müller et al. 2015] and for the reliable computation of bijective maps [Jiang et al. 2017]. While effective and efficient in 2D, in 3D, as we cover above in Section 1, they suffer from unavoidable and unacceptable locking. See Section 6 for our analysis of this behavior.

Here we explore a new route, extending the recently developed IPC model [Li et al. 2020] for elastodynamic contact, to 3D injective deformation processing. IPC employs an unsigned-distance, regularized barrier-based approach with continuous collision detection (CCD) applied within nonlinear optimization solves per time steps. We extend IPC to locking-free injective deformation problems by casting optimizations as artificial time stepping of barrier-augmented deformation objectives.

4 METHOD

4.1 Artificial Time Stepping

We include diverse terms in our objective E depending on tasks and needs. To minimize E we apply artificial time integration via the gradient system $M\dot{x} = -\nabla E_{\tilde{x}}(x)$, where M is a mass matrix of the triangulation. For time integration we choose implicit Euler (IE) – selected for its stiff decay [Ascher and Petzold 1998] enabling larger time steps h. Concretely, towards solving $x^* = \text{argmin}_x E_{\tilde{x}}(x)$, starting with x^0 we evolve the system forward per step with[2]

$$x^{t+1} = \underset{x}{\text{argmin}} \tfrac{1}{2}\|x - x^t\|_M^2 + hE\big(x, \tilde{x}, u(t)\big). \tag{1}$$

Resolving each time step then requires the nonlinear minimization of (1). We solve these minimizations via Newton iteration with globalization; see below. Each geometry process is thus divided into time intervals and then further subdivided into subintervals – each defined by a Newton iterate. In order to satisfy injectivity constraints throughout this process we ensure that all steps – that is both between outer time steps and between inner iterations, satisfy non-intersection and/or non-inversion.

[2] $\|y\|_M^2 = y^T My$.

4.2 IPC Barrier Energies

To enforce strict non-intersection guarantees we add the regularized, unsigned distance barrier, $B(x)$, from IPC [Li et al. 2020] to our system energy. The IPC barrier considers unsigned distances $d_i(t) \geq 0$, between all non-incident point-triangle and all non-adjacent edge-edge pairs i in a surface mesh. These are indexed in C. We begin with choice of a target, $\hat{d} > 0$, specifying the distance at which barrier repulsions can begin. Then, the IPC barrier per distance d is

$$b(d, \hat{d}) = \begin{cases} -(d - \hat{d})^2 \ln\left(\frac{d}{\hat{d}}\right), & 0 < d < \hat{d} \\ 0 & d \geq \hat{d} \end{cases} \tag{2}$$

and the corresponding full barrier is then

$$B(x) = \kappa \sum_{i \in C} b(d_i(x), \hat{d}), \tag{3}$$

with $\kappa > 0$ an adaptive conditioning parameter automatically controlling the barrier stiffness [Li et al. 2020]. A few details are worth pointing out here. First, while barriers for *all* necessary surface pair combinations are included in (3), any pair currently at a distance farther than $\hat{d}$ provide no contribution and can safely be removed from current evaluation, while still preserving sufficient continuity for computing barrier Hessians. This enables efficient solutions by minimizing assembly costs and fill-in for linear (Hessian) system solves. Second, the barrier functions b are indeed C^2. However, distance functions evaluated would be C^0 for unavoidable configurations; i.e., parallel edge-edge cases. In turn these discontinuities can hamper convergence, and often even stop it altogether. To this end IPC provides careful mollification and numerical treatment that enables stable and smooth optimization throughout [Li et al. 2020].

Minimizing our objective with the barrier B included extremizes the remaining terms subject to enforcement of one-sided noninterpenetration constraints. While such constraints are usually nonsmooth, here, because the barrier is regularized we can compute both its gradient and Hessian. This allows the barrier to be directly incorporated as an additional potential in our time-stepper and so optimized with second-order information for higher-order convergence. For each step we then apply the Newton-type IPC time-step solver from Li et al. [2020] to ensure non-intersecting descent steps with filtered line-search. See our supplemental document for details of our solver.

4.3 Boundary Conditions

When applied, Dirichlet boundary conditions (BC) specify a subset of positionally constrained vertices $\mathcal{B} \subset [1, n]$. For every vertex $x_k,\ k \in \mathcal{B}$, we have a corresponding target position $\tilde{x}_k$. We first

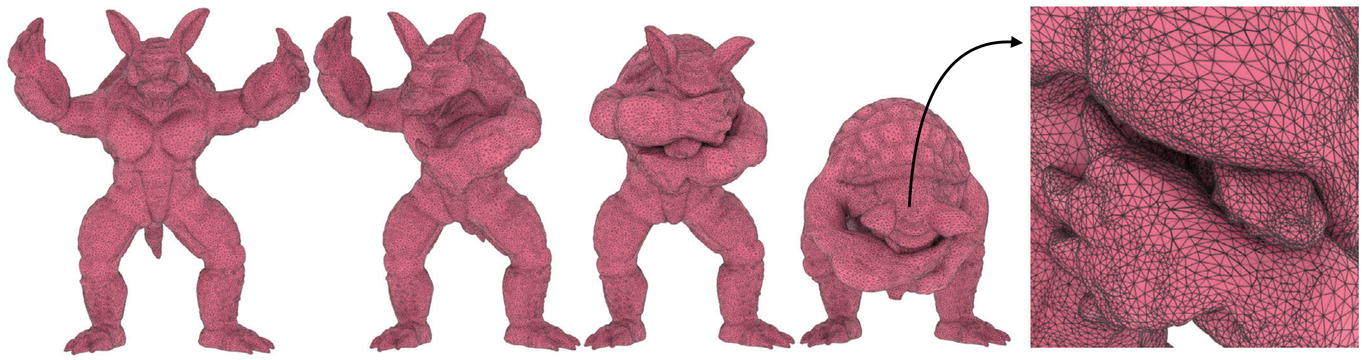

Fig. 3. **Shy Armadillo.** Left to right: moving a sparse set of hand and torso handles (feet fixed) IDP's barriers and deformation enable easy scripting of a complex sequence. We first bend just the armadillo's hands. Contacts then pushes it to bury its head while its nose peeks out without intersection. Next, we rotate the torso – collapsing the hiding armadillo onto it into its legs.

allocate a quota of m steps to reach these targets. Allocation can be determined both by application goals and efficiency. See our discussion below per application in Section 7. We then apply the first m time steps to evolve from x^0 to the geometry at time step m, x^m, satisfying all boundary conditions. From here we can continue to time step in order to further reduce the objective while preserving the achieved boundary conditions.

We subdivide $\tilde{x}_k - x_k^0$ into m subsegments, creating a sequence of intermediary target positions $\tilde{x}_k^t$ for $t \in [1, m]$. In the first Newton iterate of each of time step solve t, we then start by checking the full step taking all bound vertices in $\mathcal{B}$ to their prescribed targets. We find, via CCD and inversion detection (see our supplemental document for details), a largest possible feasible size step towards these targets that causes neither inversion nor intersection, and conservatively apply it to update all bound vertices. We then add a simple, adaptive quadratic penalty to the objective

$$P(x,t) = \frac{\kappa_B}{2} m_k \|x_k - \tilde{x}_k^{t+1}\|^2, \tag{4}$$

where m_k is vertex k's lumped mass. We adaptively increase κ_B by 2× whenever the current Newton iterate is close to convergence (via norm of gradient measure) *and* current targets are not satisfied. Alternately, if the current iterate satisfies the time step's targets, we simply fix the bound vertices to their target positions, discard P from the objective and continue Newton iteration to convergence.

4.4 Energies

For measuring deformation from geometry $\bar{x}$ to x we employ linear finite elements and, where needed, hinge stencils. This gives us corresponding deformation gradients per element $t \in T$ of $F_t(x, \bar{x}) = X_t(\bar{X}_t)^{-1}$ where X_t and $\bar{X}_t$ respectively store deformed and effective rest state frames. E.g, for tetrahedra t with vertices $x1, x2, x3, x4$ we have $X_t = [x4 - x1, x3 - x1, x2 - x1] \in \mathbb{R}^{3\times3}$. Similarly, for 3D shell models we compute the discrete bending/curvature measure from the dihedral angles of each interior edge. To enable elastic deformation behaviors we add standard distortion energies Ψ to our objective E. Giving us contributions to the objective in the form of

$$E_d(x, \bar{x}) = \sum_{t \in T} v_t \Psi\left(F_t\right), \tag{5}$$

where $v_t > 0$ is the area or volume of the rest shape of element t. To enforce non-inversion we apply barrier distortion measures, e.g., neo-Hookean (NH) [Ogden 1997] and symmetric Dirichlet (SD) [Smith and Schaefer 2015], and correspondingly add Smith and Schaefer's [2015] inversion filter to our line-search. Alternately, when we do not require inversion control we can apply a wide range of non-barrier distortion energies [Alexa et al. 2000; Igarashi et al. 2005; Sorkine and Alexa 2007]. Finally, in either case, as needed for shell models we apply discrete shell bending energy [Grinspun et al. 2003].

While geometric distortion measures like SD on their own generally do not provide parameters for changing distortion results, physically motivated measures like NH enable interesting control of deformation shape by changing material moduli, see Section 6. Likewise we enable balancing the relative weighting of multiple terms in the objective E to obtain further fine-grain control of deformation.

4.5 Time-Varying Conditions

Handles. For modeling, sculpting, animation and many other applications the sequence of time-stepped deformations (and not just the final achieved state) provides a useful feedback and often even a final desired output. Binding handles to boundary conditions allows scripted or interactive, time-varying changes to boundary conditions. Here, (rather than precomputing per-time-step targets w.r.t. final destination – as above) we instead provide a sequence or online stream of per-step target locations $\tilde{x}_k^{t+1}$. Our process for resolving them per time step remains otherwise unchanged from Section 4.3 above.

Shape. Once again taking advantage of the discrete time step framework we are also able to script changes in geometry per step and so correspondingly drive the optimization path. Transformations, for example expansions and contractions, of local or global reference shapes, scripted over time, are simply enabled by direct transform of $\bar{X}_t$ at the start of each step. Similarly, in modeling, sculpting and other applications, it is often useful to allow the optimizations to "bake" a current deformation as a new reference (e.g, rest) shape. For example this can be applied at the end of a time step whenever a modeling stage is complete. Finally, changes to reference shapes need not be integrable. For geometric operations plastic deformation is often desirable in order to sculpt and model changes that are not fully resisted by elasticity. To model controllable plasticity effects we apply return mapping from solid mechanics [Dunne and Petrinic 2005; Gao et al. 2017] that apply operators Z to project deformation gradients to limit-surfaces. This allows us to record plastic change by simply updating $\bar{X}_t$. Given current deformation gradients from the last time step, this amounts to just a simple projection, per element, at start of step: $\bar{X}_t^{-1} \leftarrow X_t^{-1} Z(F_t)$.

Mesh. Finally, during extreme deformation regions of large change can be better resolved with re-meshing. Continuous optimization of E with re-meshing would introduce nonsmooth jumps (slowing progress). Here, however, by time stepping with a first order system we can simply and directly remesh our domain at the end of any time step. When remeshing (we use TetGen [Si 2015]) we preserve boundaries, optionally adding Steiner points on the mesh surfaces, and so do not introduce overlaps.

5 IDP FRAMEWORK

Together the above components form our simple and efficient framework, IDP, for globally injective deformation tasks. At each time step IDP minimizes the step energy

$$S^t(x) = \frac{1}{2}\|x - x^t\|_M^2 + hE\big(x, \bar{x}, u(t)\big). \tag{6}$$

For minimization we implement our Newton solver in C++, applying CHOLMOD [Chen et al. 2008], compiled with Intel MKL LAPACK and BLAS for linear solves and Eigen for remaining linear algebra routines [Guennebaud et al. 2010]. To enable future applications, development and testing we will release our implementation of the IDP framework as an open source project and include in our supplemental both input and output meshes from our benchmark testing and demonstration applications below. In the following all our experiments and evaluations are executed on a Macbook Pro with 2.3 GHz 8-Core Intel Core i9, or Ubuntu Desktops with 4.2 GHz 4-core Intel Core i7-7700K, 3.7 GHz 6-core Intel Core i7-8700K, 3.0 GHz 8-core Intel Core i7-9700, or 3.6 GHz 8-core Intel Core i9-9900K as detailed per experiment below.

5.1 Time Step Size

Examining (6) we see that time step size h balances the often strongly nonlinear and nonconvex energies in E against a positive definite quadratic damping term. Larger h advances farther per time step at the cost of more challenging Newton solve and so more iterates per step. While our system offers great flexibility in choosing time step size here, for simplicity, we apply throughout all examples a time step size of $h = (0.04)^2$. We select this size for two reasons. First, we note this step size performs well across examples. Second, in cases where we wish to export a deformation sequence, time steps at this size have the visual effect of stepping a highly damped animation with frame-rate sized time steps of $\sqrt{h}$ (24 fps).

5.2 Setting Distance Accuracy

For differing applications specifying a choice of $\hat{d}$ controls how tightly surfaces can conform. Relatively small $\hat{d}$, e.g. 10^{-3} in our benchmark below, give imperceptible spacing between boundaries. While for other applications, e.g. to create 3D-printable geometries or to enable animation clean-up, we may wish to push to greater size gaps and so allow selection of larger $\hat{d}$ for these tasks.

5.3 Guarantees

By construction, when both our barrier, B, and a non-inverting distortion energy are included in the objective, the IDP time stepper guarantees that every step taken in each inner Newton iterate, and so correspondingly *every* step taken from x^0, is both non-inverting and non-intersecting. The resulting final step we end with is then correspondingly guaranteed to give a globally injective deformation. In turn, as discussed above, it also straightforward to disable all or some of these constraints when desired. To allow interpenetration the user simply does not add the barrier to the objective, while allowing inversion just requires the user to not apply a barrier distortion energy – instead applying non-barrier distortions, e.g. StVK or ARAP, if desired.

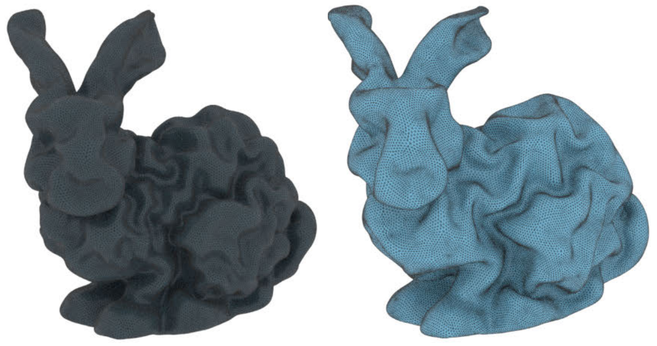

Fig. 4. **Shell bunnies at varying tolerance.** Varying solver tolerance enables a wide range of deformation behaviors (left at 1 and right at 0.1).

5.4 Tolerance

In turn we solve each time step with controllable accuracy w.r.t. the Newton decrement $n(x) = \big(\nabla^2 S^t(x)\big)^{-1} \nabla S^t(x)$, terminating each at a user selected tolerance of $\|n(x)\| \leq \epsilon_s$. With smaller tolerances each solved step better resolves deformation energy (and so for example better improves distortion) near x^t. However, irrespective of the tolerance a user selects, applied injectivity constraints are preserved. For many applications, for example in creating geometry, we observe that large tolerances are more than suitable and often even offer an interesting and useful range of effects as we vary ϵ_s. For example, consider Figure 4 where we get different patterning for the same triangle mesh across differing ϵ_s. On the other hand in other cases, e.g., if we seek to tightly minimize a distortion, smaller tolerances are best. Here IDP exposes the tolerance to the user and converges to requested accuracy per step and, when desired, to convergence of the energy minimized. See below in Section 7 for examples of these applications.

5.5 Restrictions

Finally, we should note specific design choices we have made in building *IDP*. To enforce global injectivity we require a globally injective initial state, x^0. In turn, while *IDP* will then always preserve injectivity, across every step of deformation, this comes with a restriction too. If the steps prescribed by a sequence of imposed BCs, starting from x^0, does not allow an intersection- and inversion-free path (over all variations of injective deformations satisfying the incremental BCs at each step) then *IDP* will not reach the target.

Likewise, an interesting corollary also follows: if x^0 has starting intersections, *IDP* will preserve them throughout its steps. Consider, for example, five initially nested deformable sphere volumes. They start interpenetrating and so their surfaces subsequently cannot intersect. IDP thus preserves the nested arrangement throughout deformation (see inset). In practice this is often useful as starting geometries frequently have (by artist construction) interpenetrations that are a feature, see e.g. the finger nails of the mannequin model in Section 7.3.

5.6 Summary

IDP does not fail for self-intersecting input and also does not disentangle. This is handy in applications like animation cleanup where mesh subcomponents often have "by-design" intersections from artist modeling. More generally IDP ensures that a mesh without self-intersections will remain continuously interpenetration-free throughout its deformation solely via unsigned distances barriers defined on the boundary surface mesh. Vertex penetrations not present in the input can not occur and so are not considered in the barrier. This allows nesting if input meshes are contained in each other and likewise, if any mesh-surface primitive pairs are interpenetrating on input they will remain so throughout the IDP deformation. For 2D triangular meshes and 3D tetrahedral meshes we additionally ensure non-inversion for all elements whenever barrier deformation energies are applied (e.g. NH or SD). Correspondingly, if invertible energies (e.g. ARAP) are applied and/or 3D triangular meshes are deformed we do not. This means that, when desired and appropriate, IDP continuously ensures that deformations are injective throughout the process while only ensuring non-interpenetration otherwise.

6 BENCHMARK

Currently Simplicial Complex Augmentation Framework (SCAF) [Jiang et al. 2017] is the only reported method for achieving guaranteed, globally injective deformation processing in 3D. In Section 7 below we demonstrate a wide range of applications that are enabled by injective mapping in 3D and hope that new methods building off of our framework will open the door to many more. Here we begin, however, by testing with a benchmark inspired by the bunny-in-box stress test demonstrated by Jiang et al. [2017] for the SCAF method.

To start we place a watertight bunny model inside an 10.7× larger enclosing box. In the original SCAF test the bunny's mesh is endowed with ARAP energy on its surface and then "grown" by assigning a 27× scaling of the same mesh as the deformation energy's rest shape. Here the bunny should grow to fill the box (but never intersect it) while likewise ensuring non-intersection. To do so the SCAF method creates a tetrahedral scaffolding (inflated with the SD energy) both inside the model and also outside, conforming to the box boundary. We expect improved behavior (with of course corresponding cost increases) as meshes increase resolution. Here we consider the behavior of methods on this expansion test across triangle meshes of the bunny with respectively 700, 3K, 13.5K and 54K vertices. In a comparable volumetric set-up for IDP we tetrahedralize the same bunny, endow the entire mesh with an NH energy and similarly grow it (without remeshing), by enlarging rest shape, to fill the box. See Figure 5 for set up and respective results for IDP and SCAF.

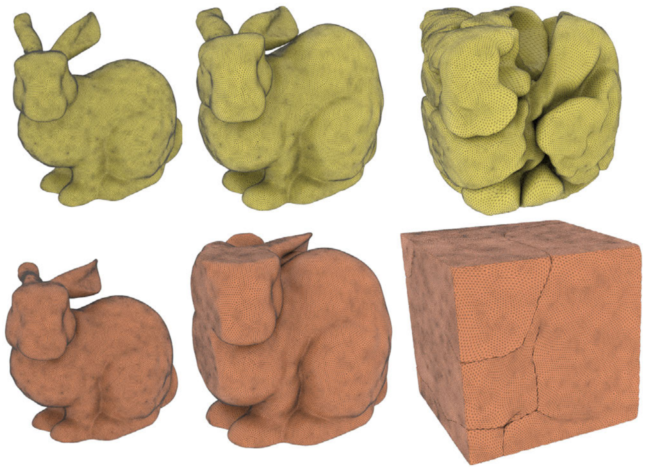

Fig. 5. **SCAF and IDP comparison** on the bunny box test from Jiang et al. [2017]. Here we pick best scaling setting for SCAF at $4K\times$ and compare with IDP with the default 27× scaling, comparing the progression of both deformations (SCAF in yellow, top row, and IDP in orange, bottom row) starting at left and going to the final max achieved deformation for each on the right. Here IDP tightly fills the enclosure while locking prevents SCAF's deformation from progressing.

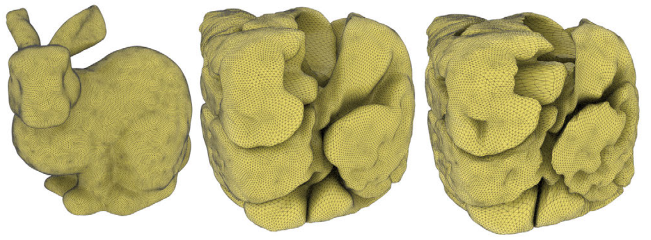

Fig. 6. **SCAF results at different target scalings.** Increasing target scaling (27×, $4K\times$, $729K\times$, final solutions shown from left to right) can improve SCAF's ability to deform in the bunny test. However, as we see here , all suffer from locking.

Across all meshes and over a wide range of stiffnesses and tolerances (and so outputs with controllably varying deformation) IDP rapidly reaches the box boundary without self-intersection nor leaving the box. As expected, as we go to larger mesh resolutions with IDP the bunny's deformation increasingly conforms to both its own boundary and box limits, until by 54K vertices it fills almost the entire box volume with just 0.07% left empty; see Figure 5 bottom right.

In comparison, despite a 27× scaling being much larger than the box, we find that SCAF does not expand to significantly fill voids between the bunny and box even as we scale to the 54K mesh. We note comparable results illustrated in Jiang et al [2017]. In order to push SCAF output to reach full expansion we next disable its fixed, upper iteration limit and instead let it run until stationarity ($\max \delta x < 10^{-5}$). Here we still observe significant gaps at termination. See Figure 6 left, where we illustrate the final shape SCAF's growth achieves for the 27× scaling. Next, in order to increase the effective weight of the ARAP energy's expansion against the weak resistance of its soft SD energy in the scaffold mesh, we experiment with successively increasing the rest shape. In Figure 6 middle and right we see some of the successive results as we increase scaling, still with significant voids separating both between the SCAF bunnies' boundaries as well as between bunnies and box. At a 4096X increase in rest shape we observe best growth for SCAF. Here, for the 54K vertex example, SCAF fills 60.9% of the volume. See Figure 5 top right, for the corresponding gaps in the final SCAF mesh. We note that despite these differences SCAF and IDP have comparable timings throughout the scaling tests. Similarly, replacing the NH energy with SD, we observe comparable behavior for IDP on all tests, albeit with slightly improved timings corresponding to lower numbers of contacts processed. Please see our supplemental for a full summary of statistics for these tests.

Here we see the challenges for SCAF (and more generally 3D fictitious mesh methods – see discussion in Müller et al. [2015]) where an entire remeshing of the domain in 3D is not practical. SCAF (as in Müller et al.) adopts the strategy of local remeshing operations for 3D and so obtains shear-locked configurations of scaffolding elements which, as we see above, resist deformation irrespective of how relatively weak the scaffold energy becomes.

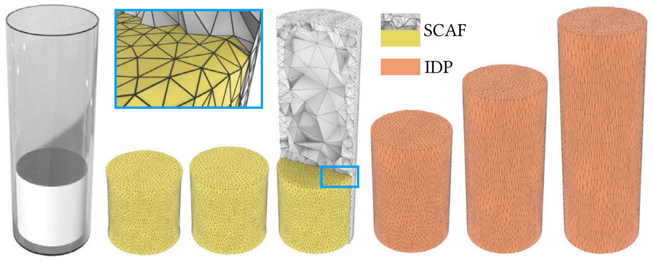

Fig. 7. **Cylinder in cylinder comparison.** We illustrate locking in a simple test in which we attempt to grow a cylinder mesh enclosed in a larger cylinder cage (left). SCAF, despite no obstructions in the cage, quickly locks after 57 iterations (mesh in yellow, scaffolding in grey) with no further progress (middle). IDP (mesh in orange) completely fills the cage after 60 time steps (right).

The bunny-box test, due to its simple boundary, exercises constrained deformation only moderately. Next, we apply two new tests to further push constrained deformation. First, to further investigate locking we propose a simple variation: inserting a 4.4K vertex cylinder (height: 1, radius 0.5) inside a larger, enclosing 5.5K vertex cylinder cage (height: 3.2, radius 0.55), we grow the cylinder by 1024× and 27× respectively for SCAF and IDP, until each method terminates with $\max \delta x < 10^{-5}$. Here in 60 time steps IDP completely fills the cage while SCAF, at 57 iterations, is locked, unable to make further progress with imperceptible growth. See Figure 7 for steps in the respective deformations including final states and scaffold configuration.

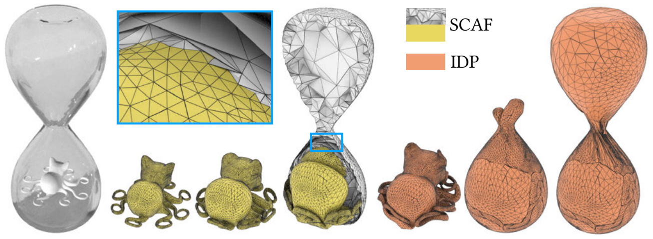

Fig. 8. **Octocat in hourglass comparison.** We expand an octocat model inside an hourglass cage with the goal of enabling deformation to fill the cage and so pass through the tight hourglass constriction (left). Here SCAF grows moderately (mesh in yellow, scaffolding in grey), until locked without filling the bottom lobe (middle). In contrast IDP (mesh in orange) passes through the neck and deforms, without intersection nor inversion to fill the cage volume (right).

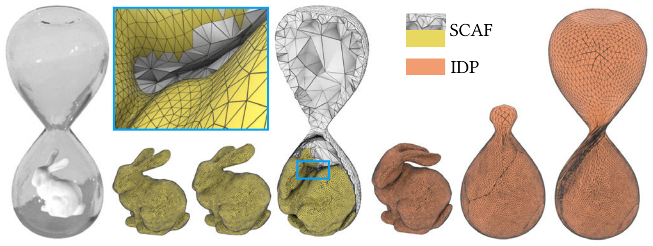

Fig. 9. **Bunny in hourglass comparison.** Following the experiment in Figure 8 we replace the enclosed octocat with a simpler bunny geometry. Here too SCAF is unable to fill the bottom lobe while IDP fills the entire hourglass cage.

Finally, to fully stress IDP we propose an hourglass test where we place respectively bunny (13.5K vertices) and octocat (17.1K) models inside hourglass cages (4.5K); see Figures 8 and 9 for SCAF and IDP results. We apply a growth of 166× and 9261× (best growth factors found) respectively for SCAF bunny and SCAF octocat, with the goal of enabling deformation to fill the cage and so pass through the tight hourglass constriction. Here SCAF grows moderately without filling the either bottom lobe until stopping progress altogether. See Figures 8 and 9 for SCAF steps and the final locked configurations of its scaffoldings. In contrast for both models, each with applied 27× growth, IDP is able to pass through the hourglass neck and deform, without intersection nor inversion, with bunny and octocat filling respectively 98.3% and 97.8% of the hourglass cage volume.

7 APPLICATIONS

Here we demonstrate and analyze a wide range of deformation tasks enabled by the IDP framework.

7.1 Sparse Handle Deformation

Inspired by Zhu et al. [2018] we begin with two sparse handle deformations tasks. We apply scripted sequences of Dirichlet BCs to a bar (5.5K vertices) and armadillo (43K vertices) meshes – each equipped with NH energy.

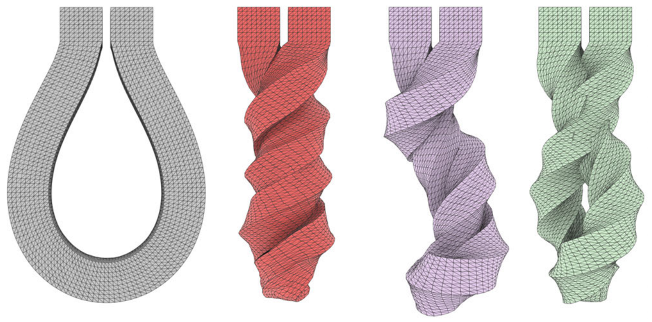

Fig. 10. **Bar bend.** from a straight reference shape (grey). We fix its two ends and then rotate it through 2 rounds of twists. Left to right we demonstrate three variations: an elasto-plastic twist with barrier (red); an elastic twist with barrier (purple); and, by removing the barrier, an elastic twist deformation (green) that is only locally-injective.

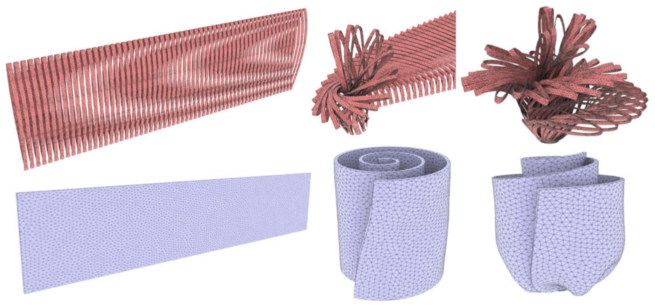

Fig. 11. **Justicia and Rose.** Left to right we illustrate the simple steps for modeling two flowers starting from simple volumetric primitives. Top left: we bend and then repeatedly copy and paste a tetrahedral bar. Both rows: simple winding twists combined with complex contact resolution then form the Justicia and Rose models.

Bending the bar, we fix its two ends and then rotate it through 2 rounds of twists. We demonstrate three variations, left to right in Figure 10: in red we demonstrates a *plastic* twist with barrier – here a more symmetric model is achieved due to the less elastic resistance while preserving global injectivity; in purple we demonstrate a globally injective twist with just the barrier – here elasticity resists the constraints more and so creates a buckled shape; finally, in green, we remove the barrier and solely enforce non-inversion.

For the armadillo model we consider an example animation task and script a small "embarrassed" sequence without plasticity. See Figure 3 and our supplemental video. We apply just a small set of handles on hands and torso – letting the barrier and deformation take care of the rest. We first bend its hands to bury its head inside them. Here the collision between face and arms allows its nose to peek out (without intersection). Next, we rotate the armadillo's torso to hide – collapsing it into its legs. With a model height of $2m$ we exercise the barrier with an imperceptible gap, applying $\hat{d} = 10^{-3}m$.

7.2 Normal Flow

Here we investigate the application of globally injective normal flows. While flows are of course extensively investigated, to our knowledge this is the first demonstration of normal flows that ensure non-intersection, rather than merging or removing degenerate regions; see e.g., Mullen et al. [2007].

IDP applies per time step normal direction, Neumann boundary conditions on mesh surfaces. The objective at time t is then

$$E(x, N^t) = B(x) - \gamma \frac{1}{2} x^T L x - \beta x^T N^t, \tag{7}$$

where N^t is the normal vector field of the surface at start of time step t, β is flow speed (either positive or negative), L is the element volume weighted Laplacian matrix with negative diagonal entries, and γ smoothing intensity. The statistics for all our normal flow examples are summarized in our supplemental document while we detail applications below.

Positive flows. Outward flow with IDP effectively inflates surfaces where contacting surfaces form creases where standard flow would merge. These highlight concavities while smoothing small surface details, giving us cute cats and cartoon-like hands. See Figure 12. Similarly, adding random noise to an ellipsoid obtains brain-like creased shapes after outward flow (Figure 12 (c)).

Negative flows. Inward flow, again without removing degenerate regions, obtains shape skeletons that remain watertight and non-intersecting throughout (Figure 13). Over steps we quickly see exceedingly thin, delicate structures appear as we watch the flow (see our supplemental video). As our Neumann conditions are set uniformly on the surface, in the extreme our inward flow provides a shape quite similar to the medial axes representation of the input, however here we preserve shape topology. For negative flows in addition to N^t we also recompute M and L per time step, obtaining a more pleasing, smoother seqeunce.

Inflation. Inflating curves [Baran and Lehtinen 2009] gives exciting variations to geometries. Here observing the puffy, balloon-like crinkling our outward flows produce we experiment with creating 3D font variations with standard 2D type-faces. Triangulating fonts we fix boundaries and flow the interior to produce inflated fonts with crinkled geometries. For each time step taken we gain ever-expanding versions of fonts enabling animation (see our supplemental video) and a range of 3D type shapes which can be further controlled by modulating smoothing intensity. In Figure 14 we experiment with number of time steps and smoothing intensities on a range of fonts and characters (Figure 14). We first demonstrate two Chinese characters from Ma Shan Zheng font, "Tao" and "Peng" ("the philosophy behind everything" and "contact"), with more and less steps respectively. With more steps Tao's volume grow larger, generating more wrinkles and inter-stroke contacts. We then test two words, "Delicious" and "Seriously", in Linux Libertine O and Arial respectively. With larger smoothing intensity applied to "Seriously", less wrinkles form with odd but interesting bulges at T-junctions, while "Delicious" demonstrates pleasing wrinkles in concavities and bulging where letters collide.

7.3 Animation Repair

Animation sequences often contain unacceptable intersection artifacts. Resulting interpenetrations can be unsightly and obvious (see e.g., Figure 15) or they can instead be visually subtle (and so hard

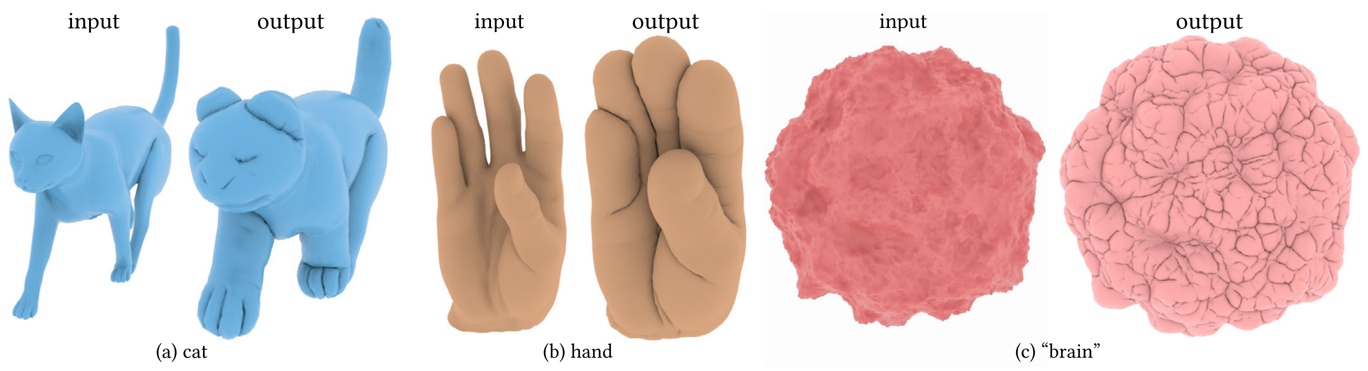

Fig. 12. **Positive (outward) normal flow** with IDP inflates surfaces with contacting regions forming folds in places where standard flows would merge or remove degeneracies. This highlights concavities while smoothing small surface details, giving us (a) cute cats and (b) cartoon-like hands, while (c) flowing a noisy ellipsoid obtains brain-like creases.

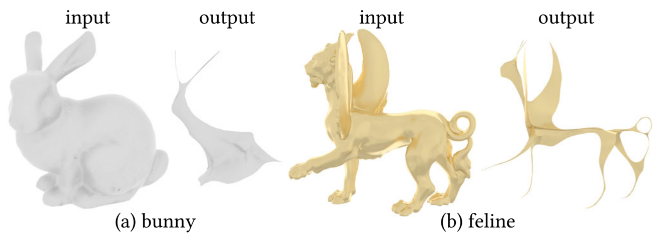

Fig. 13. **Negative (inward) normal flow** with IDP obtains shape skeletons that remain watertight, topology preserving and non-intersecting even for extreme flows.

to notice) but still cause significant failures (see e.g., Figure 17) in downstream tasks like cloth simulation [Baraff et al. 2003; Buffet et al. 2019].

IDP performs animation cleanup starting with an input of m consecutive frames all sharing the same mesh topology and varying vertex positions stored in the sequence $f^1, ..., f^m \in \mathbb{R}^{dn}$. Setting $x^1 = f^1$ and starting from the second frame, f^2 IDP fixes each consecutive frame, f^s, as a single "backward" time step where the target frame, f^s, is applied both as a proxy for the last time step, so we don't stray far, and as reference shape for deformation energies, to minimize difference. Specifically, to obtain an intersection-free fixed frame x^s close to f^s we solve the minimization

$$x^s = \underset{x}{\operatorname{argmin}} \frac{1}{2}||x - f^s||_M^2 + hE(x, f^s), \tag{8}$$

initialized from last fixed frame x^{s-1}. For objective we apply

$$E(x, \bar{x}) = \Psi_{\text{memb}}(x, \bar{x}) + \Psi_{\text{bend}}(x, \bar{x}) + B(x),$$

with an StVK membrane energy (Ψ_{memb}) and discrete shell bending (Ψ_{bend}). After each solve of (8), to process the next frame, f^{s+1}, we again initialize the optimization with fixed frame x^s and repeat through the sequence until we obtain a final, fully cleaned sequence. When the direct, linear path between a cleaned frame x^s and the next input frame f^{s+1} is interpenetration-free, this process leaves the frame unmodified ($x^{s+1} = f^{s+1}$). Otherwise, minimizing the energy for each step will continue to fix frames, starting from the last, with deformation away from the target frame balanced against the barrier. Statistics for IDP animation repair tasks are summarized in our supplemental document.

Mousey. We start by demonstrating our cleanup on an animation sequence created by a commercial rigging tool[3]. Here, meshes bound via skinning to a generic skeleton motions often create large and generally unusable intersections. In Figure ref and our supplemental video we demonstrate both an original "mousey" intersecting animation input sequence as well as the corrected sequence produced by IDP at corresponding frames. For example the original mousey sequence has severe intersections where its hand almost fully disappears into its belly (Figure 15). IDP successfully create an indentation on the belly as pushed by the hand while removing intersection from the frame. Here we set $\hat{d} = 0.001m$ and so obtain tight conforming contact in the animation. We also apply a large stiffness (Young's modulus $10^5 Pa$) to maintain smooth surface curvature.

Mannequin. Next we apply IDP to a set of three animation sequences, again created with the same rigging pipeline, with a mannequin mesh [Li et al. 2018b] (13K vertices) intended for downstream application with cloth simulation. Both before and after IDP's intersection cleanup, the animation output is, on quick visual inspection, nearly identical. However, a more careful look reveals that many frames with smaller but severe intersections are corrected; see Figure 16. Since we minimize both intrinsic and extrinsic surface differences, the fixed regions remain smooth with curvatures close to original geometries where possible. Because these sequences are next intended for cloth simulation we can make this following job easier by not only ensuring non-interpenetration, but also promoting bigger gaps between surfaces with a large $\hat{d} = 0.01m$. In Figure 17 we then compare the effects of IDP fixed vs. starting mesh input for cloth simulation. Here IDP is applied solely to fix intersections in the underlying animated mannequin geometry and is not used to model the garments draped on the mannequin. After IDP repair of the mannequin's geometry in the sequence, a simulator modeling cloth dynamics [Li et al. 2021] drapes a garment on both the repaired and un-repaired mannequin sequences. In the simulated output we

[3]www.mixamo.com

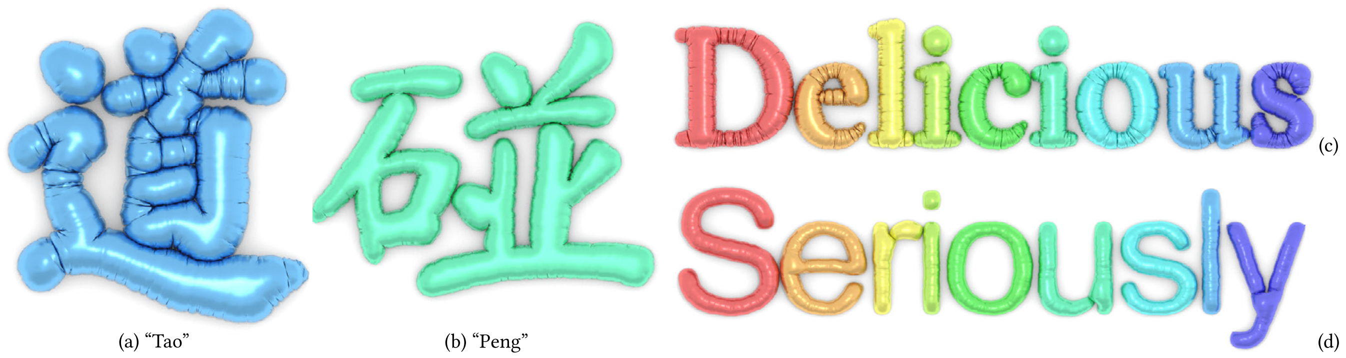

Fig. 14. **Font inflation.** The puffy, balloon-like crinkling of IDP's outward flows creates 3D font variations from standard 2D type-faces. Each time step creates new variations with expanding glyph boundaries reacting to each other in contact.

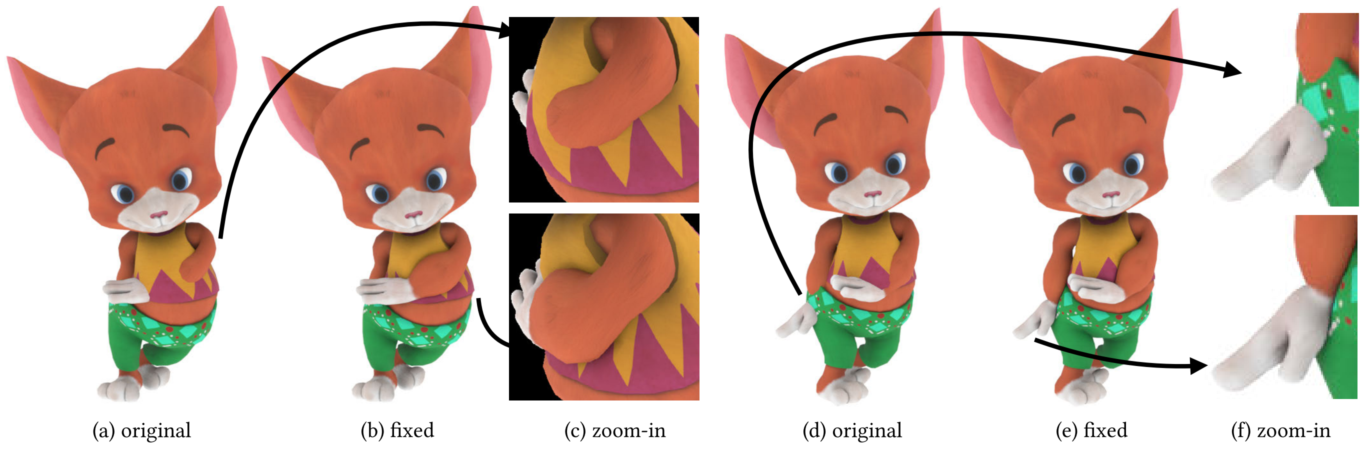

Fig. 15. **Mousey sequence animation repair.** Animation sequences often contain unacceptable and severe intersection artifacts. Here in the Mousey sequence (a and d) arms intersect both the belly and hip. IDP's clean up (b and e) removes all intersection from all frames while creating indentations on the belly and hip as pushed by the hand.

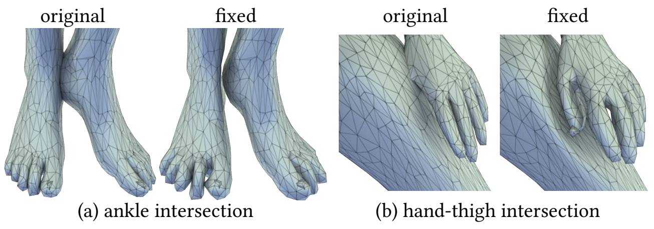

Fig. 16. **Mannequin sequence animation repair.** Applying larger $\hat{d}$ threshold (here $0.01m$) enables larger gaps between the mannequin's surfaces in the repaired sequence and so improves results for downstream cloth simulation pipelines.

see that the unintended bunching, pinching and scratching artifacts generated by simulation with the original input sequence are all avoided when simulation instead uses the fixed sequence.

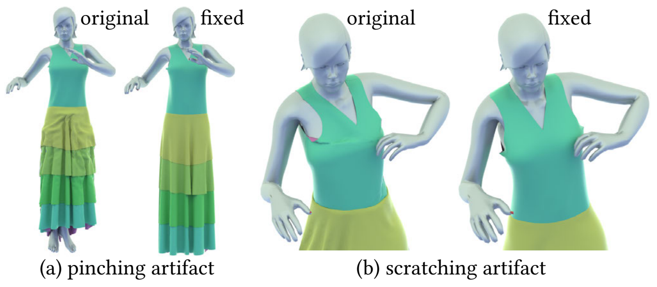

Fig. 17. **Repaired mannequin sequence with simulation output.** Applying IDP's repaired mannequin sequences (see Figure 16) to a downstream cloth simulator [Li et al. 2021] avoids unacceptable pinching and scratching artifacts in resulting garment dynamics.

7.4 Min-Max Distortion Optimization

While primarily focused on 3D deformation processing, IDP also supports 3D⟶ 2D parameterization and 2D deformation tasks. Here we apply IDP to optimize globally injective, min-max distortion problems for 3D and 2D deformation, and parameterization. As demonstrated in Section 7.1, deformation tasks often focus on minimizing energies (5), that discretize distortion measures, Ψ, over triangulations. As we've already seen, this provides an excellent means of driving down average distortion. However, in many cases this can still leave some elements with severe distortion concentrations. In turn, these small concentrated regions can have an outsized negative impact by creating severe artifacts and/or numerical challenges for downstream applications like rendering and simulation.

To improve worst-element distortions we formulate a regularized min-max optimization for IDP by applying a p-norm extension to (5) with with exponential factor $p > 1$,

$$E_p(x,\bar{x}) = r(p) \sum_{t \in T} v_t \big(\Psi(F_t)\big)^p, \;\; r(p) = \frac{1}{p}\big(\Psi(I)\big)^{(1-p)}. \qquad (9)$$

Here $r(p)$ calibrates the modified energy so that the p-norm extension does not change stress derivatives at reference shapes $\bar{x}$. Energy E_p then effectively adds weighting to elements with larger distortion, to minimize them more. In turn, weighting for worst elements increases as p grows large and so optimization approaches the solution of $\min_x \max_t \Psi(F_t)$.

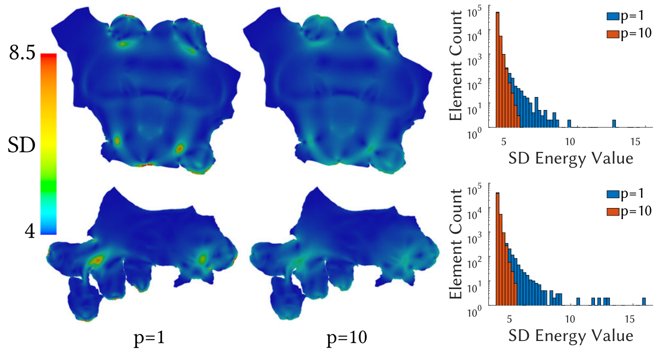

Fig. 18. **Improving worst-case distortion for UV mapping.** Minimizing a p-norm regularized SD energy with $p = 10$, drives down worst-case distortion from a standard SD-optimized input. Starting with distortion above 15 (nearly 3× isotropic stretch/compression) IPD quickly drives distortion to well below 6 (4 is isometric) while preserving injectivity.

3D⟶2D. Applying SD for distortion measure, Ψ, we add the extended energy E_p and barrier B to the IDP objective to *improve* maps. In Figure 18 we take as input two SD-optimized maps produced by OptCuts – a recent seam-cutting method from Li et al. [2018a]. While OptCuts minimizes over both seam placement and distortion energy (average distortion in input is 4.1), worst-case distortion (see Figure 18) in the input still reaches above 15 – nearly 3× isotropic stretch/compression. Starting with these results as input and applying $p = 10$, IDP then drives worst-case distortion below 6 (4 is isometric) for both camel head (28K vertices in 12.9s) and hand (23K vertices in 13.4s) models while preserving bijective maps. The optimization is terminated after it first reaches a time step requiring a single Newton iteration to converge.

2D. Min-max optimization can also be applied to for handle-driven deformation. In Figure 19 we apply the same objective we composed above for map improvement. The task now is to drive four boundary handles of a 64K vertex disk (top, bottom, left and right) inwards with both $p = 10$ and $p = 1$ (for baseline comparison).

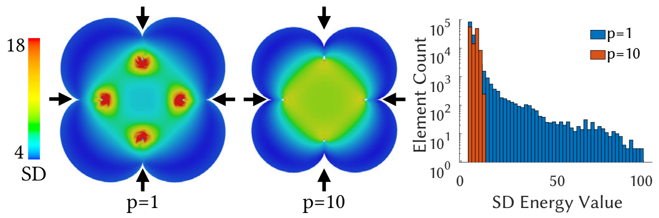

Fig. 19. **2D Min-Max Distortion Optimization.** Minimizing distortion in 2D with a p-norm regularized SD energy we drive four boundary handles of a disk inwards with both $p = 10$ and $p = 1$ (baseline).

Both solutions converge satisfying *global* injectivity and handle targets. Direct optimization of the SD energy ($p = 1$) obtains a worst case distortion of nearly 100, while with $p = 10$ we obtain improved worst-case distortion with all triangles below 14. In terms of performance, optimizing with high p-norm adds no overhead here (at 0.23min per time step) when compared direct SD minimization (at 0.3min per time step).

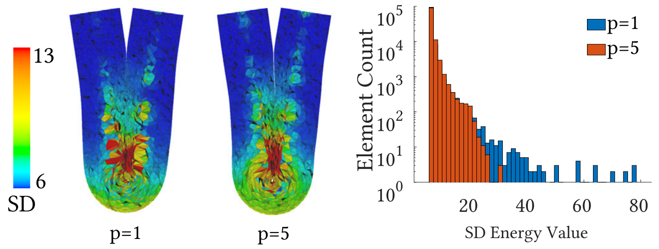

Fig. 20. **3D Min-Max Distortion Optimization.** minimizing distortion in 3D with p-norm regularized SD energy on tetrahedral elements, we fix the center of a tetrahedral mesh cylinder and apply two bone handles (two thin boxes of interior axis vertices) to bend it with both $p = 5$ and $p = 1$ (baseline).

3D. Similarly in 3D, with the same energies in the objective, we fix the center of a 22K vertex tetrahedral mesh cylinder and apply two bone handles (two thin boxes of interior axis vertices) to bend it; see Figure 20. With $p = 5$ worst case distortion drops to 30.29 from the max of 92.78 obtained with $p = 1$. In Figure 20 we can see the subtle geometric differences resulting from the different stress concentrations at the bend. Here, timing increases from 13.7min to 22.2min for the 100 time steps optimized with $p = 1$ and $p = 5$ respectively.

7.5 Modeling and Layout

IDP enables modeling and layout of complex geometries directly from primitive shapes, by emulating real-world manual steps with handles. As demonstrations we follow rough "directions" to make dumplings and a vase of decorative flowers. For our dumpling process please follow along in Figure 1. With plasticity enabled (von-Mises projection [Fang et al. 2019]), we first take a soft sphere volume (NH energy) and squash it down with a translated rigid plane to form a wrapper. Placing an ellipsoid (also NH volume) filling on top, we set handles along the outer two halfs of the wrapper. Folding the handles inwards gives the basic dumpling shape. Next

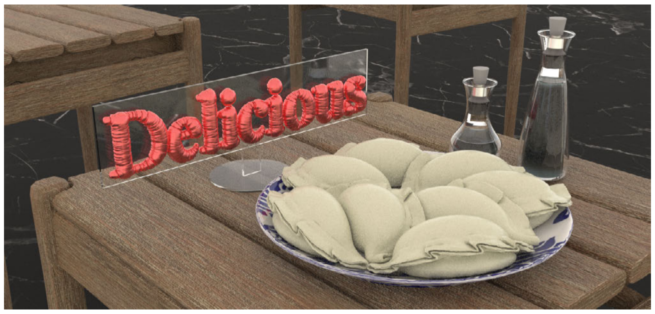

Fig. 21. **A delicious dumpling** layout combines our dumplings (Figure 1) and our inflated 3D font art (Figure 14).

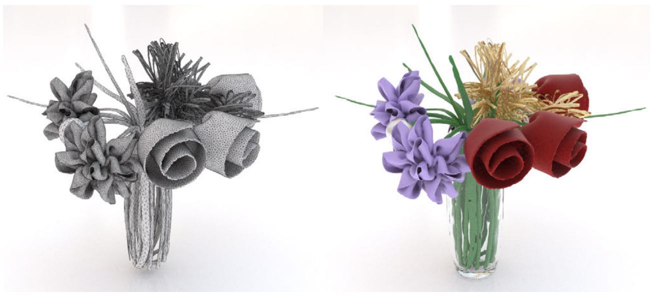

Fig. 22. **A Flower arrangement** layout is created by placing all modeled flowers in a vase. Deformation and contact ensure the models naturally rest against each other and the final vase geometry.

we pinch the dumpling with a simple symmetric motion of 8 rigid ellipsoids. We finished the dumpling by squashing again around the sealing with two more rigid cutouts. With our dumpling geometry completed we can instance it and place it on plate. With deformation still enabled, despite each copied model being initially identical, deformation and nudging give each dumpling a final unique shape in the layout. See our video for our full modeling steps and our supplemental materials for the final dumpling model.

With these illustrative steps as reference, we can next dive a bit more into the details. Given the large shape changes (enabled by the plastic deformation) our tetrahedral meshes can become poorly-shaped after even a single modeling step. At the same time, after each modeling application we have achieved a new shape target that we may wish to keep. As discussed in Section 4 with IDP we enable the option to "bake in" the current shape after any time step (and so after any completed handle sequence). Here this first applies surface-preserving remeshing [Si 2015] followed by a simple reset of the reference mesh to the current modeled shape. Each following stage of modeling then pulls towards the last until the next baking operation is applied (e.g., after the next operation is completed). Likewise, we find it useful to expose deformation stiffness as a controllable parameter. For example here we found it useful to make the "dough" stiffer ($Y = 10^5$) when squashing and then make it softer for finer detailing, for example when pinching the creases ($Y = 10^4$).

With this workflow in place we apply the same set of operations (handle manipulation, baking, instancing, stiffness variation and layout) to follow instructions for creating molded and folded flowers in a vase. Please see Figures 2, 11 and 22 and our video for details of the process, and our supplemental materials for the final modeled geometries.

8 DISCUSSION

Timings and statistics for all applications, as well as for our benchmark tests, are summarized in our supplemental document. In it we observe that where SCAF is able to make some progress, IDP provides both significantly improved deformation and comparable or better timings. However, IDP's key contribution is not in speed but in its ability to provide high-quality, convergent, and interpenetration-free deformations of 3D domains where this was not previously possible with prior methods. As discussed, SCAF is prevented for practical 3D applications by locking while (as discussed in Section 3) prior methods are unable to ensure non-interpenetration due to reliance on iterative collision-response routines. While we look forward to making IDP faster, we note that current timings range from half a second (font inflation) up to a max of 12 minutes (for a large mesh deformation) per frame. Timings vary as expected with application (nonlinearity of terms in objective), severity of deformation (as reflected in number of iterations required), mesh size and constraints (contacts) processed. E.g., Dumpling Fold uses up to 14K nodes, processes 5.7K constraints and requires 35 iterations per frame, while Rose is 8K nodes, processes only 0.6K constraints and requires only 5 iterations per frame.

8.1 A Note on Locking and Feasibility

The term locking is utilized in diverse ways across the literature. Here we specifically apply it to broadly refer to artifacts that unduly restrict the deformation enabled by a computational mesh (i.e. the primary triangulated or tetrahedralized domain), caused by discretization applied to enforce global injectivity. Concretely, in Section 6 we see that the SCAF method suffers from shear locking where tetrahedra in the fictitious domain are severely sheared (even after local remeshing) and so oppose deformation of the primary mesh. This means that non-interpenetration in SCAF comes at the cost of artificially restricting desired deformations and often even (see Section 6) preventing them altogether.

We contrast this with feasibility which we describe as an intersection-free (and, when desired, inversion-free) deformation of the mesh. As discussed in Section 5.5, when a traversed deformation path can not be continued due to infeasibility, IDP will not proceed. In many applications not proceeding into intersection may be a desirable result. While certainly in other applications it may not. For these latter cases, enabling infeasible progress towards a final feasible solution remains a challenging open problem.

9 CONCLUSION

Extending recent results in physical simulation we have developed IDP, the first framework for robustly deforming in 3D with injectivity guarantees. We have performed extensive benchmark testing to confirm IDP's reliable behavior in practice, and have constructed a wide range of applications demonstrating its utility. Here we see opportunity for effectively endless further possibilities – of which our current examples only scratch the surface. Looking ahead we are excited to see what geometries users will be able to create with IDP, and likewise what new applications it will be able to support. To enable such future applications and extensions we will be releasing our code for IDP. In terms of most immediate steps, while in many applications IDP is able to create geometries rapidly and certainly matches 3D SCAF in timings, much more remains to be done in terms of speeding IDP's time step computations. Much of this is low level optimization, but algorithmic improvements in the Newton stepper, for example leveraging efficient preconditioning and aggressive tolerances should also be investigated. Likewise, we have only begun in terms of both energies and operations that could extend injective deformation processing further. Similarly, although we have yet to find a need for it, automatic remeshing, based on local mesh quality should be a useful and easy extension to support. Likewise, topology change to enable cutting and seaming is also another natural operation to investigate in future work. Finally, as discussed, IDP cannot compute injectivity when starting from an input geometry that does not have it. Extending injective deformation processing to find injectivity (with guarantees) for noninjective initializers is an exciting and long-term direction of investigation.

ACKNOWLEDGMENTS

We thank Jacky (Jiecong) Lu for assistance in generating animation sequences, Zhongshi Jiang for the SCAF library, and Etienne Vouga and Tiantian Liu for valuable discussion. This work was supported in part by NSF CAREER IIS-1943199, CCF-1813624, and ECCS-2023780.

REFERENCES

Marc Alexa, Daniel Cohen-Or, and David Levin. 2000. As-rigid-as-possible shape interpolation. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 157–164.

U.M. Ascher and L.R. Petzold. 1998. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM.

David Baraff, Andrew Witkin, and Michael Kass. 2003. Untangling cloth. *ACM Trans. Graph. (TOG)* 22, 3 (2003), 862–870.

Ilya Baran and Jaakko Lehtinen. 2009. Notes on Inflating Curves. (2009).

Sofien Bouaziz, Mario Deuss, Yuliy Schwartzburg, Thibaut Weise, and Mark Pauly. 2012. Shape-up: Shaping discrete geometry with projections. In *Computer Graphics Forum*, Vol. 31. Wiley Online Library, 1657–1667.

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. 2014. Projective dynamics: Fusing constraint projections for fast simulation. *ACM Trans. Graph. (TOG)* 33, 4 (2014), 1–11.

Robert Bridson, Ronald Fedkiw, and John Anderson. 2002. Robust treatment of collisions, contact and friction for cloth animation. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. 594–603.

Tyson Brochu and Robert Bridson. 2009. Robust topological operations for dynamic explicit surfaces. *SIAM Journal on Scientific Computing* 31, 4 (2009), 2472–2493.

Thomas Buffet, Damien Rohmer, Loïc Barthe, Laurence Boissieux, and Marie-Paule Cani. 2019. Implicit untangling: a robust solution for modeling layered clothing. *ACM Trans. Graph. (TOG)* 38, 4 (2019), 1–12.

Yanqing Chen, Timothy A Davis, William W Hager, and Sivasankaran Rajamanickam. 2008. Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. *ACM Trans. on Math. Software (TOMS)* 35, 3 (2008).

Xingyi Du, Noam Aigerman, Qingnan Zhou, Shahar Z. Kovalsky, Yajie Yan, Danny M. Kaufman, and Tao Ju. 2020. Lifting Simplices to Find Injectivity. *ACM Trans. Graph.* (2020).

Fionn Dunne and Nik Petrinic. 2005. *Introduction to computational plasticity*. Oxford University Press on Demand.

Y. Fang, M. Li, M. Gao, and C. Jiang. 2019. Silly rubber: an implicit material point method for simulating non-equilibrated viscoelastic and elastoplastic solids. *ACM Trans. Graph. (TOG)* 38, 4 (2019), 1–13.

M. Gao, A. P. Tampubolon, C. Jiang, and E. Sifakis. 2017. An adaptive generalized interpolation material point method for simulating elastoplastic materials. *ACM Trans. Graph. (TOG)* 36, 6 (2017), 223.

Eitan Grinspun, Anil N Hirani, Mathieu Desbrun, and Peter Schröder. 2003. Discrete shells. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Citeseer, 62–67.

Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3.

David Harmon, Daniele Panozzo, Olga Sorkine, and Denis Zorin. 2011. Interference-aware geometric modeling. *ACM Trans. Graph. (TOG)* 30, 6 (2011), 1–10.

David Harmon, Etienne Vouga, Rasmus Tamstorf, and Eitan Grinspun. 2008. Robust treatment of simultaneous collisions. In *ACM SIGGRAPH 2008 papers*. 1–4.

Takeo Igarashi, Tomer Moscovich, and John F Hughes. 2005. As-rigid-as-possible shape manipulation. *ACM Trans. Graph. (TOG)* 24, 3 (2005), 1134–1141.

Zhongshi Jiang, Scott Schaefer, and Daniele Panozzo. 2017. Simplicial complex augmentation framework for bijective maps. *ACM Trans. Graph.* 36, 6 (2017).

Shahar Z. Kovalsky, Noam Aigerman, Ronen Basri, and Yaron Lipman. 2015. Large-scale bounded distortion mappings. *ACM Trans. Graph. (proceedings of ACM SIGGRAPH Asia)* 34, 6 (2015).

Shahar Z. Kovalsky, Meirav Galun, and Yaron Lipman. 2016. Accelerated Quadratic Proxy for Geometric Optimization. *ACM Trans. Graph. (SIGGRAPH 2016)* (2016).

Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. 2020. Incremental Potential Contact: Intersection- and Inversion-free Large Deformation Dynamics. *ACM Trans. Graph.* 39, 4 (2020).

Minchen Li, Danny M Kaufman, and Chenfanfu Jiang. 2021. Codimensional Incremental Potential Contact. *ACM Trans. Graph. (TOG)* 40, 4 (2021).

Minchen Li, Danny M Kaufman, Vladimir G Kim, Justin Solomon, and Alla Sheffer. 2018a. Optcuts: joint optimization of surface cuts and parameterization. *ACM Trans. Graph. (TOG)* 37, 6 (2018), 1–13.

Minchen Li, Alla Sheffer, Eitan Grinspun, and Nicholas Vining. 2018b. FoldSketch: Enriching Garments with Physically Reproducible Folds. *ACM Trans. Graph. (TOG)* 37, 4 (2018). https://doi.org/10.1145/3197517.3201310

Marek Krzysztof Misztal and Jakob Andreas Bærentzen. 2012. Topology-Adaptive Interface Tracking Using the Deformable Simplicial Complex. *ACM Trans. Graph.* (2012).

Patrick Mullen, Alexander McKenzie, Yiying Tong, and Mathieu Desbrun. 2007. A variational approach to Eulerian geometry processing. *ACM Trans. Graph. (TOG)* 26, 3 (2007), 66–es.

Matthias Müller, Nuttapong Chentanez, Tae-Yong Kim, and Miles Macklin. 2015. Air meshes for robust collision handling. *ACM Trans. Graph. (TOG)* 34, 4 (2015), 1–9.

Raymond W Ogden. 1997. *Non-linear elastic deformations*. Courier Corporation.

Stéphane Pagano and Pierre Alart. 2008. Self-contact and fictitious domain using a difference convex approach. *Int. J. for Numer. Meth. in Eng.* 75 (07 2008).

Leonardo Sacht, Etienne Vouga, and Alec Jacobson. 2015. Nested cages. *ACM Trans. Graph. (TOG)* 34, 6 (2015), 1–14.

Christian Schüller, Ladislav Kavan, Daniele Panozzo, and Olga Sorkine-Hornung. 2013. Locally injective mappings. In *Computer Graphics Forum*, Vol. 32. Wiley Online Library, 125–135.

Hang Si. 2015. TetGen, a Delaunay-based quality tetrahedral mesh generator. *ACM Transactions on Mathematical Software (TOMS)* 41, 2 (2015), 1–36.

Jason Smith and Scott Schaefer. 2015. Bijective parameterization with free boundaries. *ACM Trans. Graph. (TOG)* 34, 4 (2015), 1–9.

Olga Sorkine and Marc Alexa. 2007. As-rigid-as-possible surface modeling. In *Symposium on Geometry processing*, Vol. 4. 109–116.

Jian-Ping Su, Chunyang Ye, Ligang Liu, and Xiao-Ming Fu. 2020. Efficient bijective parameterizations. *ACM Trans. Graph. (TOG)* 39, 4 (2020), 111–1.

Pascal Volino and Nadia Magnenat-Thalmann. 2006. Resolving surface collisions through intersection contour minimization. *ACM Trans. Graph. (TOG)* 25, 3 (2006).

Martin Wicke, Hermes Lanker, and Markus Gross. 2006. Untangling cloth with boundaries. In *Proc. of Vision, Modeling, and Visualization*. 349–356.

Audrey Wong, David Eberle, and Theodore Kim. 2018. Clean cloth inputs: Removing character self-intersections with volume simulation. In *ACM SIGGRAPH 2018 Talks*.

Juntao Ye, Guanghui Ma, Liguo Jiang, Lan Chen, Jituo Li, Gang Xiong, Xiaopeng Zhang, and Min Tang. 2017. A unified cloth untangling framework through discrete collision detection. In *Computer Graphics Forum*, Vol. 36. Wiley Online Library, 217–228.

Juntao Ye, Timo R Nyberg, and Gang Xiong. 2015. Fast discrete intersection detection for cloth penetration resolution. In *2015 IEEE International Conference on Multimedia Big Data*. IEEE, 352–357.

Yufeng Zhu, Robert Bridson, and Danny M. Kaufman. 2018. Blended Cured Quasi-Newton for Distortion Optimization. *ACM Trans. Graph. (SIGGRAPH 2018)* (2018).