Eurographics Symposium on Geometry Processing 2013
Yaron Lipman and Richard Hao Zhang
(Guest Editors)

Volume 32 (2013), Number 5

Consistent Volumetric Discretizations Inside Self-Intersecting Surfaces

Leonardo Sacht[1,2] Alec Jacobson[1] Daniele Panozzo[1] Christian Schüller[1] Olga Sorkine-Hornung[1]

[1]ETH Zurich, Switzerland [2]IMPA, Brazil

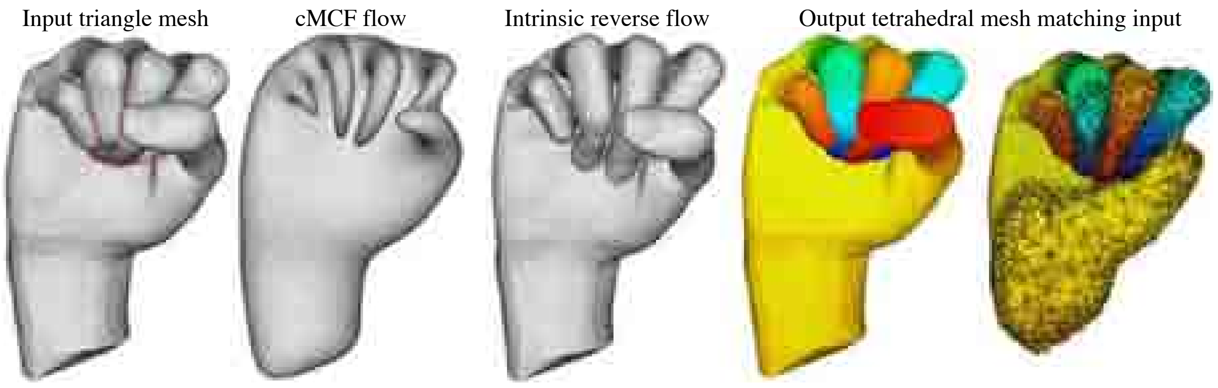

Figure 1: *The triangle mesh of the* Hand *forms a closed surface, but contains nearly 2000 intersecting triangle pairs. Our method flows the surface according to conformalized mean-curvature flow (cMCF) until all self-intersections are removed. Then we* reverse *the flow so that shape intrinsics are restored but self-intersections are avoided. Finally we can tet-mesh inside this surface and map the mesh so that it matches the original surface. We may then solve PDEs, such as this biharmonic function.*

Abstract

Decades of research have culminated in a robust geometry processing pipeline for surfaces. Most steps in this pipeline, like deformation, smoothing, subdivision and decimation, may create self-intersections. Volumetric processing of solid shapes then becomes difficult, because obtaining a correct volumetric discretization is impossible: existing tet-meshing methods require watertight input. We propose an algorithm that produces a tetrahedral mesh that overlaps itself consistently with the self-intersections in the input surface. This enables volumetric processing on self-intersecting models. We leverage conformalized mean-curvature flow, which removes self-intersections, and define an intrinsically similar reverse flow, which prevents them. We tetrahedralize the resulting surface and map the mesh inside the original surface. We demonstrate the effectiveness of our method with applications to automatic skinning weight computation, physically based simulation and geodesic distance computation.

1 Introduction

Recent years have shown leaping advancements in *surface-based* shape processing, in particular polygonal mesh processing [BKP*10]. Volumetric shape processing, on the other hand, lags behind. One significant obstacle is the inability to convert boundary representations of solid shapes into explicit volumetric representations at any given stage of the geometry processing pipeline. Robust tools for creating tetrahedral meshes from watertight input surfaces do exist, e.g. [Si03]. However, the presence of self-intersections in the surface mesh invalidates the otherwise clean (closed, orientable) input to volume meshing algorithms. Unfortunately, most—if not nearly all—steps in the *surface-based* geome-

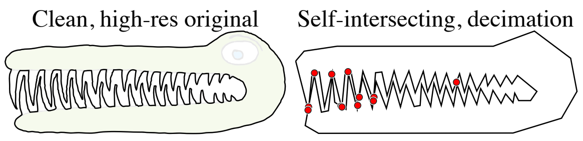

Figure 2: *Even simple geometric operations like boundary decimation can introduce self-intersections (red dots).*

try processing pipeline (such as decimation, smoothing, subdivision, remeshing, surface-based deformations) may invalidate watertightness by creating self-intersections [HPSZ11] (see Figure 2). As a consequence, attempts at further volumetric processing reveal artifacts resulting from ignoring or deleting self-intersecting regions (see Figure 10), and therefore geometry processing remains limited to the surface.

Volumetric processing has a lot of valuable advantages. While surface meshes are appropriate representations for some shapes, such as thin shells (e.g. an automobile fender), many interesting shapes are solids. For a solid shape, like a deformable human character, we may sometimes get away with a surface-only representation thanks to its intrinsic, though indirect, relationship to the underlying volume and because we often will only render the surface. However, many processing tasks perform drastically differently when treating a solid shape as a surface rather than a volume, e.g. the bending of scanned clay statuette (see Figure 10 in [BPWG07]), rendering shapes made of translucent material, like an amber jewel [LSR*12], registering two poses of a dancing human [LBB12], or even simple shape smoothing (see Figure 3). Notably, volumetric representations facilitate volume preservation and internal geodesic distance computation. Ubiquitous techniques like finite element analysis and solving PDEs typically require an *explicit* representation of a shape's volume: most commonly, a tetrahedral mesh.

We propose a method to construct a tetrahedral mesh for self-intersecting input. Instead of gluing overlapping regions together, our output volume mesh overlaps itself consistently with the self-intersections in the input surface (see Figure 1). This enables correct geodesic information necessary for shape-aware volumetric processing at any stage in the geometry processing pipeline.

We begin with a key observation: For sphere-topology surfaces, conformalized mean-curvature flow (cMCF) converges to the unit sphere [KSBC12] and removes all intersections. Given an input surface, we follow this flow until all self-intersections are removed; typically long before reaching the sphere (see Figure 5). Meshing techniques like constrained Delaunay tessellation (CDT) should in theory work on this resulting surface. We could try to mesh its interior and then try to map this mesh back inside the original surface as an exercise in volumetric parameterization. However, exponential scaling of the triangles during cMCF introduces numerical issues for existing CDT software. This large distortion also makes the subsequent volumetric parameterization difficult or impossible, even with state-of-the-art methods [SKPSH13].

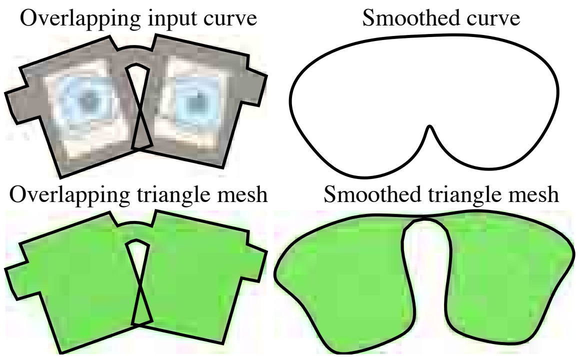

Figure 3: *Geometric operations perform drastically differently on boundary versus region representations. Compare the results of Laplacian smoothing for these eyeglasses as a curve and as a region.*

We solve this problem in two steps. We first reverse the flow in an intrinsic way, while maintaining absence of self-intersections as an invariant. For each step backward in the flow, we minimize a surface-distortion energy subject to safe contact constraints. Because cMCF is smooth (even conformal in the limit), this may be interpreted as an intersection-free, surface simulation *regularized* by the flow. When we have returned to time zero, our surface is similar to the original input surface intrinsically, but self-intersection free (see Figure 6). Now, we may safely apply existing CDT methods. As a final step, we map the surface of this volume to the original, self-intersecting input surface and propagate the map to the interior. We show the success of our method for applications including solving PDEs, volumetric elastic deformation and simulation, automatic skinning weight definition and geodesic distance computation.

2 Problem context.

Volumetric discretizations of *watertight* shapes have greatly improved in the past years [She12]. State-of-the-art methods provide guarantees on element quality and have rich feature sets like the ability to specify spatially-varying density fields or exactly conforming to a given piecewise-linear surface mesh [CGAL, Si03, LS07, GR09]. However, they all assume that the input is a representation (implicit function, triangle mesh) of a watertight surface[1]. We bootstrap these methods in order to discretize volumes of *self-intersecting* solids, which of course have self-intersecting surfaces. We heavily

[1] Formally, a watertight surface is "a 2-[manifold] embedded in $\mathbb{R}^3$ whose underlying space is same [sic] as the boundary of the closure of a 3-manifold in $\mathbb{R}^3$" [DG03].

employ the CDT and mesh refinement routines of the award-winning TetGen software [Si03]. By leveraging locally injective volumetric parameterization, we prevent inverted elements in our output, so we may further post-process our output by other mesh refinement techniques to achieve gradations or even higher quality [She12].

Some surface repair techniques eliminate self-intersections, outputting a watertight mesh which may then be meshed. But these repairs either delete [SOS04, Att10] or fuse intersecting pieces [JKSH13]. Our method accommodates self-intersections without any modification to the original surface—local or otherwise (see Figure 10).

Naive methods are generally not an option. Having meshed the entire convex hull, one could segment based on the winding number (analogous to the "nonzero-rule" of SVG and OpenGL, [FvDFH90, JKSH13]). This not only leads to incorrectly deleting or joining entire regions, but also easily results in non-manifold output. It is also tempting to consider decomposing the input into intersection-free pieces, meshing each independently and reconnecting them. Constructing, let alone combinatorially reconnecting, such a decomposition is not obvious for complicated or multiple overlaps. Further, one must ensure discretization coherency across cuts. Luo et al. [LBRM12] consider planar cuts to decompose shapes for 3D printing, but even assuming self-intersection free input they show that managing these requires care. Our method avoids combinatorial decisions.

Instead of decomposing or modifying the input surface, we find a new embedding for it via conformalized mean-curvature flow (cMCF), which removes self-intersections [KSBC12]. Several common flows converge to the sphere, e.g. the Willmore flow, volume-preserving mean-curvature flow, heat diffusion flow, etc. [Wil00]. Unlike mean-curvature flow, many of these flows are known to create new self-intersections even in their absence in the input [MS00, MS03], making them unsuitable for our purposes. We also enjoy the simplicity, performance and robustness of cMCF for triangle mesh discretizations.

We optimize a surface-based, elastic energy with dynamics to reverse the flow while preventing self-intersections with safe contact constraints and repulsion forces at the vertex level. Harmon et al. [HPSZ11] similarly prevent self-intersections during interactive deformation, speeding up computation by grouping collision responses. Alternatively, we could employ this method, but grouping runs the risk of locking early on during the reverse flow. Further, our simulation conveniently does not require expensive and complicated gradient computation for the deformation energy. There is a large amount of literature in physically-based simulation on robust and efficient handling of collisions [Har10]. Given our intersection-free state we could treat the original surface as a rest-state and run any off-the-shelf surface simulation with safe collision handling (e.g. [BFA02]). However, this quickly results in locking (see Figure 7). Our use of the forward flow surfaces at intermediary time steps avoids this. Another option would be to avoid the flow altogether and attempt to untangle the self-intersections [BWK03]. Tailored to open cloth surfaces, this heuristic appears to succeed for small overlaps, but purposefully assumes no knowledge of a possible intersection-free state (e.g. a previous frame in their simulation). It relies instead on heuristic global topological analysis of the input.

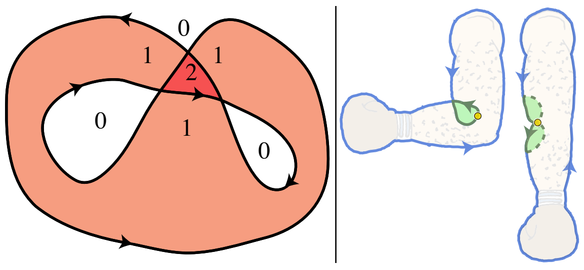

Figure 4: *Left (with winding numbers indicated): contrary to [MGR11], this curve is not a "self-crossing loop". Middle: This* elbow case *is also not a self-crossing loop, and so it is not considered by any existing 2D method. Our method succeeds by allowing the mapping to not be locally injective on the boundary (at the yellow dot), visualized in a hypothetical untangled state (right).*

A few works have defined similar problems in $\mathbb{R}^2$: discretizing or charting the area inside an overlapping curve in the plane. Unfortunately their solutions are slow and do not extend to $\mathbb{R}^3$ [SVW89, EM09, MGR11]. It is very difficult to have a definition of valid input that is not based on the existence of valid output, or in other words, defining what valid input is such that validity is easy to verify. The observations in Section 2.4 of [MGR11] make use of the *winding number*, the signed number of times a curve wraps around a point. Unfortunately, these observations are necessary but not sufficient, since they would accept the infeasible curve in Figure 4 (left). This curve is not even a valid "self-crossing loop", i.e. the boundary curve of a locally-injectively deformed circular disc [SVW89]. Interestingly, the existence-of-output definition of self-crossing loops used in [SVW89] and all following works does not include the *elbow case* (Figure 3b of [SVW89]), which arises frequently during surface deformations (see Figure 4, right). Our formulation is more general and handles this case by allowing a zero-measure subset of the boundary where the computed map is not locally injective. A similar situation in 3D is shown in Figure 9.

3 Problem description

Let our input be $\mathcal{M}$, a closed, orientable $(d-1)$-manifold embedded in $\mathbb{R}^d$, with possible self-intersections. From now on we assume $d = 3$, but our problem and solution generalize

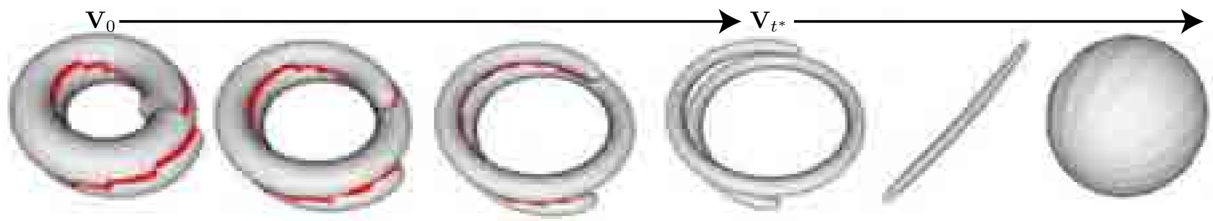

Figure 5: *We evolve a self-intersecting surface (intersections are shown in red) with the conformalized Mean Curvature Flow. After some time t^* the surface reaches a self-intersection free state. The limit is a conformal mapping to the unit sphere.*

for $d \geq 2$. Our goal is to chart the volume *inside* $\mathcal{M}$. That is, we wish to find a continuous map $\Omega : \mathcal{D} \to \mathrm{R}^3$ where $\mathcal{D}$ is an abstract 3-manifold with boundary and our mapping Ω meets the following requirements:

$$\Omega(\partial\mathcal{D}) = \mathcal{M}, \tag{1}$$

$$\Omega(\mathcal{D} \setminus \partial\mathcal{D}) \quad \text{is differentiable}, \tag{2}$$

$$|J_\Omega(\mathbf{p})| > 0 \quad \forall \mathbf{p} \in \mathcal{D} \setminus \partial\mathcal{D}, \tag{3}$$

where $|J_\Omega(\mathbf{p})|$ is the determinant of the Jacobian matrix of the map Ω evaluated at point $\mathbf{p}$. The first requirement states that Ω should map the boundary of $\mathcal{D}$ to the input surface $\mathcal{M}$. The second and third requirements ensure local injectivity on the interior. We do not require local injectivity on the boundary. This allows isolated "hinge points" on $\partial\mathcal{D}$ to appear, necessary for handling *elbow cases* (see Figure 4). With the local injectivity requirement on the boundary our definition would be equivalent to "self-crossing loops" [SVW89]. In practice, these requirements imply that our output tetrahedral mesh conforms to the input boundary, has no flipped (negative signed volume) tetrahedra, and has proper connectivity.

Forward flow. As in [KSBC12], we define the conformalized mean-curvature flow (cMCF) $\Phi_t : \mathcal{M} \to \mathrm{R}^3$ to be a smooth family of immersions, each the solution to the partial differential equation:

$$\frac{\partial \Phi_t}{\partial t} = \sqrt{|g_t^{-1} g_0|} \Delta_{g_0} \Phi_t, \tag{4}$$

where $g(\cdot,\cdot)_t$ is the metric induced by the immersion at time t, and Δ_{g_0} is the Laplace-Beltrami operator defined with respect to the original metric g_0 of $\mathcal{M}$.

Kazhdan et. al prove that "if cMCF converges, than [sic] it converges to a map on the sphere if and only if the limit map is conformal". This is of special importance to us because the mapping to the sphere will have removed all self-intersections. They observed this convergence in all their tests with sphere-topology examples, and we confirm this empirically, too. In general, self-intersections disappear long before reaching the sphere. Let t^* and Φ_{t^*} be the time at which this occurs and the corresponding immersion, respectively (see Figure 5).

Reverse flow. In the discrete setting, the exponential scaling in Φ_t introduces numerical issues that prevent us from directly mapping the volumetric closure of $\Phi_{t^*}(\mathcal{M})$ back to $\mathcal{M}$. We alleviate this by finding an *intrinsic* reverse flow. Let $\Psi_t : \mathcal{M} \to \mathrm{R}^3$ be a family of immersions defined as the optimum of the nonlinear, constrained optimization problem:

$$\underset{\Psi_t}{\text{argmin}} \quad E_{\text{surf}}(g_t, \tilde{g}_t), \tag{5}$$

$$\text{subject to:} \quad \Psi_t \text{ is injective} \tag{6}$$

where $\tilde{g}(\cdot,\cdot)_t$ is the metric induced by Ψ_t and $E_{\text{surf}}(g_t, \tilde{g}_t)$ measures the *similarity* of the metrics g_t and $\tilde{g}_t$. In this way, our optimization finds a non-self-intersecting immersion Ψ_t which is as close as possible to Φ_t (see Figure 6). Since Ψ_t is defined w.r.t. discontinuous constraints, we cannot write that it is a *smooth* family of immersions w.r.t. t, but this is not an issue for us as we are only concerned with the quality of Ψ_0.

We can immediately notice that if cMCF converges, the feasible set of solutions is non-empty: the sphere and thus also Ψ_{t^*} are intersection-free. In the ideal world, E_{surf} would measure the similarity of the *volumes* within Φ_t and Ψ_t, but of course without the unknown discretization of the volume within Φ_t this is elusive in practice. Hence, we choose E_{surf} to be a cumulative measure of local surface rigidity, namely the surface-based elastic energy discussed in [CPSS10].

For our purposes, we are only concerned with the final reversed flow Ψ_0, but in practice we compute Ψ_t at the same samples in time as Φ_t. These intermediary solutions will be essential as feasible, initial guesses to the subsequent steps back in time $(t - \delta)$ until reaching time 0.

Volumetric parameterization. Now we no longer need to treat $\mathcal{D}$ as an abstract domain: let $\mathcal{D}$ be the closure of $\Psi_0(\mathcal{M})$. The problem of finding a suitable volumetric mapping Ω reduces to a volumetric parameterization problem with a fixed boundary. We can write this volumetric parameterization Ω as the optimum of:

$$\underset{\Omega}{\text{argmin}} \quad E_{\text{vol}}(\Omega), \tag{7}$$

$$\text{subject to:} \quad |J_\Omega(\mathbf{p})| > 0 \quad \forall \mathbf{p} \in \mathcal{D} \setminus \partial\mathcal{D}, \tag{8}$$

$$\Omega(\partial\mathcal{D}) = \mathcal{M}, \tag{9}$$

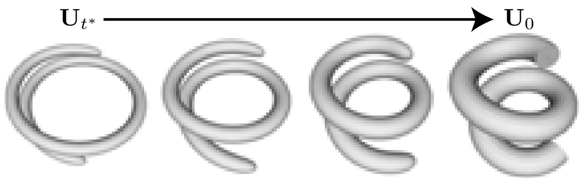

Figure 6: *We evolve the intersection-free surface obtained with cMCF to a one that is similar to the input surface intrinsically, but without self-intersections.*

where E_{vol} is an arbitrary non-negative energy. In fact, since we only care about the map Ω insofar as we care about the constraints (8-9), it is helpful conceptually to consider simply $E_{\text{vol}}(\Omega) = 0$. This is in contrast to the typical, variational parameterization or deformation problems, where energies are carefully crafted to minimize distortion or satisfy problem-specific needs.

4 Discretization

We have described our solution in the continuous case, and now we must discretize it in order to mesh self-intersecting input surfaces. Let us restrict our input shape in $\mathbb{R}^3$ to be a closed, orientable, triangle mesh described by a list of n vertices $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n\}$, $\mathbf{v}_i \in \mathbb{R}^3$ and a list of m triangle facets $\mathbf{F} = \{f_1, f_2, \dots, f_m\}$ where $f_i \in \{1, 2, \dots, n\}^3$. Our goal is then to find a set of tetrahedral elements $\mathbf{E} \subset \{1, \dots, k\}^4$ defined over a set of vertices $\mathbf{V_E} \supseteq \mathbf{V}$ which represent the overlapping volume of $(\mathbf{V}, \mathbf{F})$. In general, $k > n$ since we may add Steiner points.

The discrete analogs to the requirements (1-3) are:

1. all triangles in **F** appear as faces of boundary tets in **E**,
2. the signed volume of each tet in **E** is positive, and
3. **E** forms a combinatorial 3-manifold with boundary.

Forward flow. Discretizing the cMCF Φ_t follows exactly as described in [KSBC12]. We compute the cotangent Laplacian of the original mesh $(\mathbf{V}, \mathbf{F})$ and then for each discrete step δ forward in time we update the mass matrix according to vertex positions $\mathbf{V}_t$ of the current immersion Φ_t. In addition to updating the flow, at each time step we detect if any self-intersections remain. We do this by computing all triangle-triangle intersections using the exact predicates (but inexact construction) kernel in [CGAL]. We, of course, stop early as soon as an intersection is found (see Figure 5). To efficiently compute the intersections we use the box intersection implementation provided by [CGAL] that exploits spatial hierarchies.

Reverse flow. We follow forward in time by discrete steps δ ($\sim 10^{-4}$) until no self-intersections remain, resulting in $\mathbf{V}_{t^*}$ (see Figure 5). Then, we flow *in reverse*. Starting with $t = t^*$ and initializing $\mathbf{U}_{t^*} \leftarrow \mathbf{V}_{t^*}$, we minimize the surface-based elastic energy, which treats $\mathbf{V}_{t-\delta}$ as the rest-state, using $\mathbf{U}_t$ as the initial guess of the unknown positions $\mathbf{U}_{t-\delta}$ (see Figure 6).

We implement this in a fashion similar to the dynamics method of [CPSS10] with three notable differences. Instead of a volumetric energy we use a surface-based as-rigid-as-possible (ARAP) energy, in particular the "spokes-and-rims" energy described in their Section 4.2. Instead of the Newton solver proposed by Chao et al., we use a "local-global" solver as described by [SA07]. This is simpler to implement and avoids expensive Hessian computations. Finally, we need absolutely safe collision detection and response. To handle this we detect all intersecting triangles for each time step in the simulation, again using [CGAL]. All vertices of each offending triangle are fixed to their previous positions, and the solution for that time step is resolved recursively until no intersections remain. For all vertices of each intersecting pair of triangles we also accumulate repulsion forces to be used for the next time step. These force vectors are the difference between the barycenters of the triangles times a scalar weighting term (for all results in this paper we have defined this weight as 1000, but good results are obtained for the range $[1, 1000]$). The accumulated forces for each vertex are defined as the external forces only for the next time step.

We opt for dynamics, rather than worry about fixing enough vertices in $\mathbf{U}_t$ to remove the translational and rotational degrees of freedom in the under-constrained ARAP energy. Adding dynamics to an ARAP energy optimization is simple. Suppose our unknown positions at *simulation* time i are $\mathbf{U}_t^i$, then we begin with Netwon's second law:

$$\mathbf{f}_{\text{ext}} + \mathbf{f}_{\text{int}} = \mathbf{M}\mathbf{a}^i \tag{10}$$

$$\mathbf{f}_{\text{ext}} + \nabla E_{\text{ARAP}}(\mathbf{U}_t^i) = \mathbf{M}\mathbf{a}^i \tag{11}$$

where $\mathbf{f}_{\text{ext}} \in \mathbb{R}^{n\times 3}$ and $\mathbf{f}_{\text{int}} \in \mathbb{R}^{n\times 3}$ are the external and internal forces respectively, $\mathbf{M} \in \mathbb{R}^3$ is the (diagonalized) mass matrix, and the unknown accelerations are discretized with finite differences in *simulation* time $\mathbf{a}^i = (\mathbf{U}_t^i - 2\mathbf{U}_t^{i-1} + \mathbf{U}_t^{i-2})/\varepsilon^2$, where ε is the *simulation* time step (not to be confused with the flow time step δ). We immediately treat the gradient of our ARAP energy E_{ARAP} as an internal force. The anti-derivative of Equation (11) w.r.t. $\mathbf{U}_t^{i\,\mathsf{T}}$ results in a time dependent energy:

$$E_{\text{dyn}}(\mathbf{U}_t^i, \mathbf{U}_t^{i-1}, \mathbf{U}_t^{i-2}) = \frac{\varepsilon}{2}\mathbf{a}^{i\,\mathsf{T}}\mathbf{M}\mathbf{a}^i - \mathbf{U}_t^{i\,\mathsf{T}}\mathbf{f}_{\text{ext}} + E_{\text{ARAP}}(\mathbf{U}_t^i).$$

Notice that in terms of the implementation of a local-global solver, the addition of dynamics only affects the global Poisson solve, and only the right-hand side changes as simulation time i advances. Thus a Cholesky decomposition may still be prefactored. The local step (SVDs for best-fit rotations) is not affected. More details may be found in [JBK*12]. Intuitively, the dynamic simulation corresponds to introducing an energy term at each iteration that gently pulls the current guess toward the previous guess. This creates drag, but also regularizes the Poisson solve at the "global" step and enables

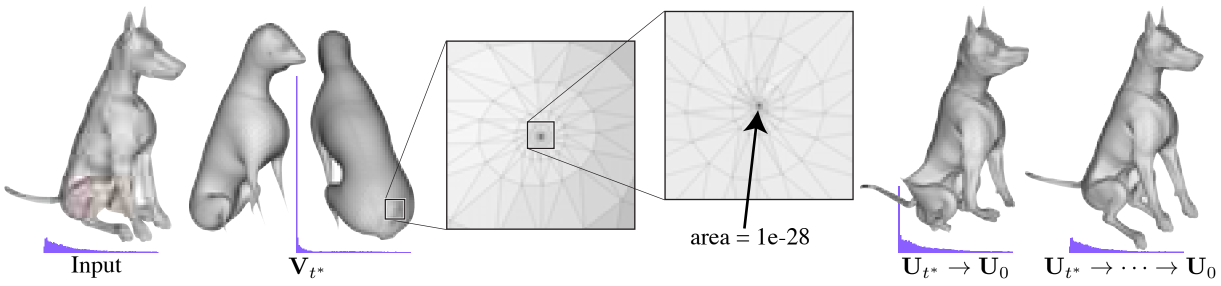
Input
$\mathbf{V}_{t^*}$
area = 1e-28
$\mathbf{U}_{t^*} \to \mathbf{U}_0$
$\mathbf{U}_{t^*} \to \cdots \to \mathbf{U}_0$

Figure 7: *cMCF removes intersections on the* Dog *producing* $\mathbf{V}_{t^*}$*, but also introduces enormous scaling: flowed tail shown in nested insets. Attempting to optimize directly back to the original metric (*$\mathbf{U}_{t^*} \to \mathbf{U}_0$*) finds a self-intersection free immersion, but with heavy distortion. Our reverse flow using intermediary steps (*$\mathbf{U}_{t^*} \to \cdots \to \mathbf{U}_0$*) instead finds a self-intersection free and low distortion solution. Purple histogram overlays show the distribution of triangle areas.*

the aforementioned repulsion forces. Finally, after each simulation is completed, we register $\mathbf{U}_t$ to $\mathbf{V}_t$ with a globally optimal rigid transformation [Sor09].

We repeat this simulation for each discrete flow time t, resulting in a sequence of self-intersection free immersions $\{\mathbf{U}_{t^*}, \dots \mathbf{U}_t, \dots, \mathbf{U}_0\}$ (see Figure 6). Most importantly, $\mathbf{U}_0$ is intrinsically similar to our original mesh geometry $\mathbf{V}_0$, but free of self-intersections. We mesh the interior of $(\mathbf{U}_0, \mathbf{F})$ using TETGEN [Si03], setting parameters to achieve slightly graded elements, but with sufficient circumradius-to-edge ratio (~ 2). This produces a tet-mesh with elements $\mathbf{E}$ and vertex positions $\mathbf{U}_\mathbf{E} \supseteq \mathbf{U}_0$.

Though only auxillary, the intermediary steps $\mathbf{U}_t$ are essential as initial guesses for each step backward in the flow. Immediately optimizing for $\mathbf{U}_0$ treating $\mathbf{V}_{t^*}$ as an initial guess results in many collisions early on, locking the surface in an unsatisfactory shape (see Figure 7).

Volumetric parameterization. The ill-posed energy optimization in Equation (7) could be discretized into a piecewise-linear mapping $\Omega : (\mathbf{U}_\mathbf{E}, \mathbf{E}) \to \mathrm{R}^3$ in a variety of ways. We examine two choices. First we could consider treating the local injectivity constraint in Equation (8) as a *weak* constraint, translating it into an energy term which punishes flipped elements. One available energy is the elastic energy for tetrahedral meshes described in [CPSS10]. We can minimize this energy (subject to the boundary constraints implied by Equation (9)) with the same local-global solver used earlier, only now the energy is volumetric, defined with $(\mathbf{U}_\mathbf{E}, \mathbf{E})$ as the rest state.

This discretized energy punishes flipped elements, but only with finite energy[2]. For most applications, flipped elements are undesired, leading to inaccuracies. For some applications like physically based simulation or mesh refinement, flipped elements are a deal-breaker. While mesh untangling methods such as [FP00, Knu01] could be employed, they are slow and ignore our extra information of a self-intersection free state $(\mathbf{U}_\mathbf{E}, \mathbf{E})$. Therefore, we consider another option: treating the constraints Equation (8) as *hard* constraints.

We employ a solver for exactly this problem: finding locally injective maps [SKPSH13]. Rather than use an arbitrary constant energy for E_{vol}, we notice that, while we do not care about the distortion of the mapping (we can always refine as a post process), choosing an energy that punishes tetrahedra with degenerating unsigned volumes assists [SKPSH13] in finding a locally injective map. To this end, we employ Green's strain energy (refer to [SKPSH13] for the definition).

Additionally, since we do not care about the distortion of the internal mapping, we may further assist [SKPSH13] by refining our tet mesh during optimization. For the volume parameterization and deformation problems considered by [SKPSH13], this is in general not possible or desired. In our case, we notice that this greatly improves the chances of finding a feasible solution. We refine every 100 iterations using TETGEN's coarsen-then-refine option. This first removes existing internal Steiner points and then adds new points, ensuring the same quality as discussed earlier and avoiding an explosion in the mesh size. Refining the current solution to a new mesh $(\mathbf{U}'_\mathbf{E}, \mathbf{E}')$ effectively redefines the domain of our mapping, $\Omega : (\mathbf{U}'_\mathbf{E}, \mathbf{E}') \to \mathrm{R}^3$, but by abuse of notation we are still finding a solution to our original problem.

The problem in R^2 is far more studied, and many options for locally injective mappings exist. While the method of [SKPSH13] outperforms other methods such as [Lip12], [XCGL11] present an even faster solution, guaranteed to detect feasibility. Unfortunately, while extending the *implementation* of [XCGL11] to R^3 is straightforward, there is no proof (yet) that the guarantees extend as well. In our experiments, we found that for simple examples extending [XCGL11] to R^3 succeeds, but for more complicated inputs numerical issues arose before convergence could be reached or infeasibility could be determined.

[2] In terms of the derivation in [CPSS10], this can be chalked up to discretization error as the continuous energy should be infinite.

Input model			Computation time			Output
Name	$\lvert\mathbf{F}\rvert$	#s.i.	Φ_{t^*}	Ψ_0	Ω	$\lvert\mathbf{E}\rvert$
Decimated	500	7	0.1	1	0.8	694
Leg	13230	239	0.4	125	30	23968
Cheese	15944	368	0.4	340	1	24447
Male	30788	1133	0.9	200	223	78647
Hand	44000	1924	1.6	435	1922	86000
Dog	50576	1042	2.2	523	-	-
Polygirl	65800	679	0.6	140	1642	122118

Table 1: *Statistics for the various examples. $\lvert\mathbf{F}\rvert$ is the number of facets in the input 3D surface and #s.i. the number of intersecting pairs of facets. We report timings for each stage of our algorithm in seconds: (Φ_{t^*}) computing the cMCF flow until self-intersections are removed, (Ψ_0) computing reverse flow preventing self-intersections, (Ω) computing volumetric map. The number of elements in the output tet mesh is $\lvert\mathbf{E}\rvert$.*

5 Experiments and results

We report statistics in Table 1. Timings were obtained on an iMac Intel Core i7 3.4GHz computer with 16GB memory. We implement our method primarily as a serial program in MATLAB. Self-intersections are determined using the CGAL C++ library, and meshing is performed with TETGEN. Neither are a bottle-neck. The locally injective mapping method of [SKPSH13] is also implemented as a serial subroutine written in C++.

See the attached zipped folder containing input, intermediary, and output meshes of all results in the paper. Also, see the attached video showing forward (cMCF) and reverse flows for each model. cMCF successful removes all self-intersections in the sphere-topology example we tested, typically after 10-20 iterations. We use [SKPSH13] for all of our examples, except the *Dog* in Figure 7 for which it does not find a solution. Finding locally injective mappings is a difficult and unsolved problem, but as new methods appear in these areas, our approach will immediate see benefits. Switching to ARAP for this example is a possibility though doing so produces 654 flipped tetrahedra out of 147588. We also tried ARAP on other examples where [SKPSH13] succeeds: for the *Hand* in Figure 1, ARAP flips 325 out of 86000. Both the *Dog* and the *Hand* highlight the intrinsic quality of our final reverse flow surface.

The *Dog* model in Figure 7 contains multiple self-overlapping parts, all resolved by cMCF. However, flowing with cMCF comes at a cost: the entire tail has shrunk around a small point on the *Dog*'s behind. The smallest triangle area is far too small for TETGEN to deal with. Instead, our reverse flow restores the intrinsic shape of the dog. This surface, and a tet-mesh generated in it, is ready to use for physically based simulation applications.

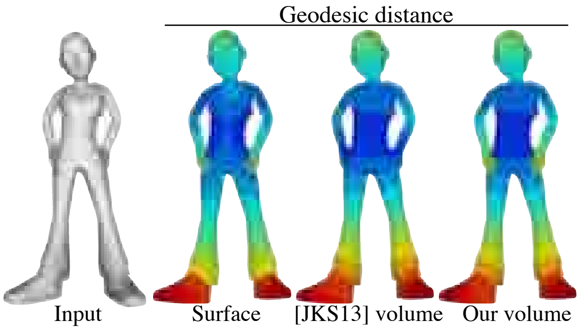

Figure 8: Polygirl*'s hands intersect her waist (left). Geodesic distance to a point on her back computed on the surface correctly separates the hands, but measures around the waist instead of through it. The positive winding number mesh of [JKSH13] has the opposite situation. Using our output volume mesh, both are correct.*

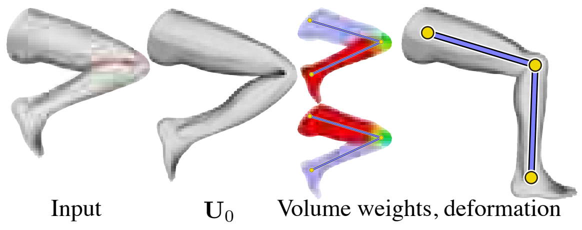

Figure 9: *Our method enables automatic skinning weight computation with [JBPS11].*

In Figure 1, the *Hand* is also restored well by our reverse flow. As desired, when mapped to match the input surface, the generated tetrahedral mesh has overlapping tetrahedra on the fingers: this is made obvious when solving the biharmonic equation with alternating Dirichlet boundary conditions on the finger tips.

Geodesic distance computation is a cornerstone of geometry processing. In Figure 8, we compare *surface* geodesic distances computed on a self-intersecting input with *volumetric* geodesic distances computed on our consistently overlapping output tet mesh and the fused mesh of [JKSH13]. Our mesh reveals the semantically distant waist and hands, without compromising the volumetric distance through the body.

The *Leg* in Figure 9 overlaps itself non-trivially in a 3D analog to the elbow case in Figure 4. Our method resolves this and volumetric skinning weights for a manually defined skeleton are generated using Bounded Biharmonic Weights (BBW) [JBPS11]. Each weight function correctly controls the corresponding portion of the *Leg*, despite the spacial overlap. Deforming the mesh with linear blend skinning reveals this as the *Leg* extends without artifacts.

Mesh repairing algorithms can be used to resolve inter-

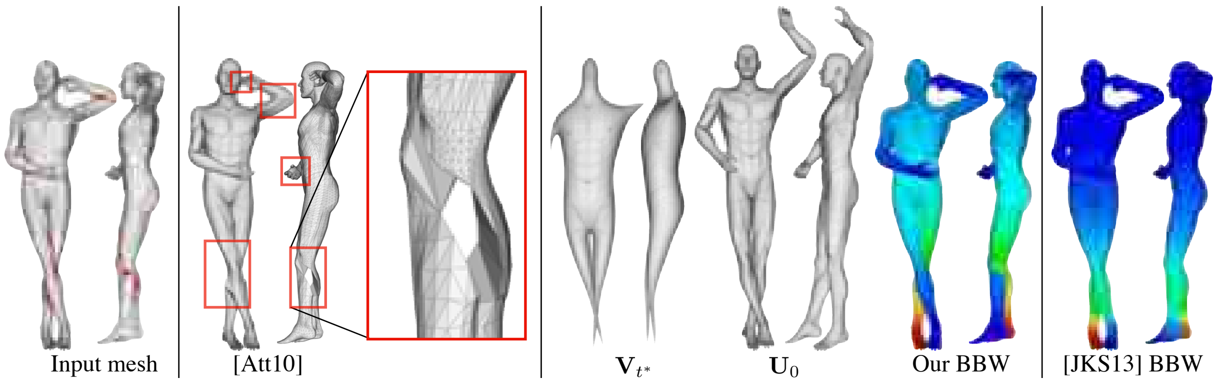

Figure 10: *Left to right: the limbs of* the Male *intersect each other and the body (front and sideview).* MeshFix *of [Att10] successfully removes these intersections, but local modifications are too aggressive. The cMCF flow removes intersections, resulting in* $\mathbf{V}_{t^*}$*, and our reverse flow restores the original shape intrinsics* $\mathbf{U}_0$*. We use [SKPSH13] to map this to the interior of the input surface and compute BBW weights for point handles at each extremity (left foot visualized). Meshing according to positive winding number [JKSH13] glues the limbs together, producing unsatisfactory weights.*

sections from a surface mesh, such that it can be subsequently tetrahedralized. We show in Figure 10 the watertight output of MeshFix [Att10]. Although a tet-mesh could be generated, much of the surface of the *Male* has been deleted or altered (see inside red rectangles). Instead, our method resolves self-intersection without modifying the input's connectivity. The result of our reverse flow is meshed and the mesh is mapped back to the original surface, where BBWs may be computed for each extremity. An alternative, is to consider the interior as defined by the positive winding number [JKSH13]. This glues semantically distant regions together resulting in poor BBWs. Reduced elastic simulations [JBK*12] are computed for these two tet-mesh and BBW results and compared in Figure 11. Notably the limbs in our mesh separate freely, while those of [JKSH13] are awkwardly stuck together.

To perform complex volumetric physical simulations, it is common to first decimate surface meshes to a very coarse level and then tetrahedralize their interior. We show in Figure 12 that decimation can introduce self-intersections and that our method is able to define a tet-mesh for the interior of the decimated surface.

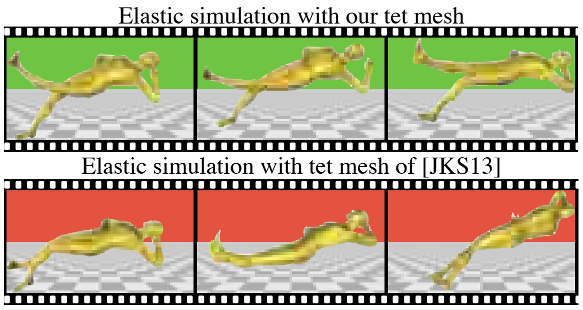
Elastic simulation with our tet mesh
Elastic simulation with tet mesh of [JKS13]

Figure 11: *Our output tetmesh is consistent with the self-intersections of the original shape. This allows limbs to move freely during elastic simulation of* the Male *(top). Meshing according to positive winding number glues semantically distant regions together [JKSH13], causing the legs to stick together and the arms to stick to the belly and head (bottom).*

5.1 Limitations and future work

We would like to improve the computational performance of our reverse flow dynamics in future work, perhaps by employing a subspace reduction method such as [JBK*12], though the incorporation of safe contact response is not obvious.

cMCF and other flows such as Willmore, do not, in general, converge to self-intersection free surfaces. Thus we have no guarantee that our method will work for high-genus shapes. However, we found that self-intersections are often removed by cMCF early on, even for high genus shapes (see Figure 13). This is in contrast to the conformal Willmore flow [CPS13]. Our experiments show that this method removes all intersections at a much later state (see Figure 14). Using [CPS13] as the forward flow in our pipeline would make it difficult for the reverse flow to restore the shape. Although it seems that this new flow works better for high-genus surfaces, there is no guarantee of convergence to a self-intersection free shape. Investigating other flows or modifications of existing flows for removing and *preventing* self-intersections is an interesting direction for future work.

Although we use the state-of-the-art locally-injective mapping method of [SKPSH13], the optimization may sometimes get stuck, unable to satisfy the boundary conditions. Their method treats local injectivity as hard con-

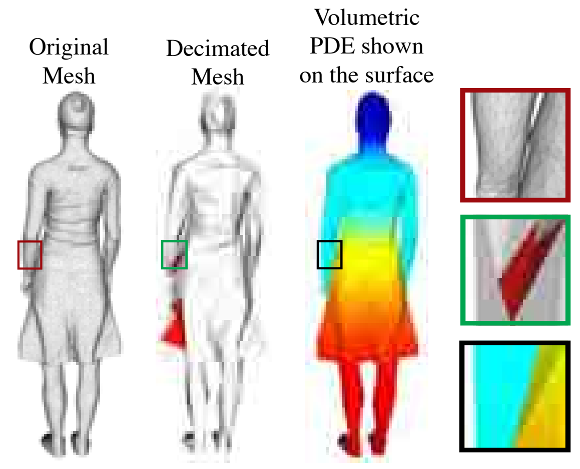

Figure 12: *A dense mesh is decimated, and the result presents self-intersections. Our method successfully defines a tet-mesh for its interior, on which we can solve the biharmonic PDE.*

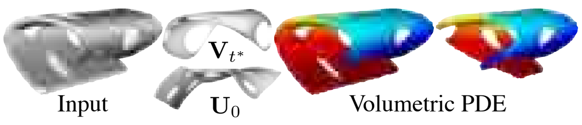

Figure 13: *We observe that for some cases—such as this thick slice of* Cheese*—self-intersections are also removed from high genus shapes before cMCF converges.*

straints using a barrier method, but the positional constraints on the boundary are only enforced with quadratic penalty terms. We do not report the result for the *Dog* model since [SKPSH13] was not able to satisfy all the soft constraints.

In some cases the reverse flow does not succeed in restoring the original shape. In Figure 15 we show an example where the bristles inside a torus need more space to restore their original geometry and get locked by the torus. Histograms of triangle areas (bottom) show that the geometry of the result of the reverse flow is not similar to the input shape.

Finally, even some sphere-topology shapes will not work with our method. Shapes implying regions of negative winding number (see Figure 16) are ill posed as they do not define a clear "interior." Our method can reject these immediately, but it remains to prove the sufficient conditions for invalid input to our problem.

6 Conclusion

Our discretized formulation proves to be a powerful tool for consistently meshing, previously *unmeshable* models. Our reverse flow takes maximal advantage of the cMCF, which

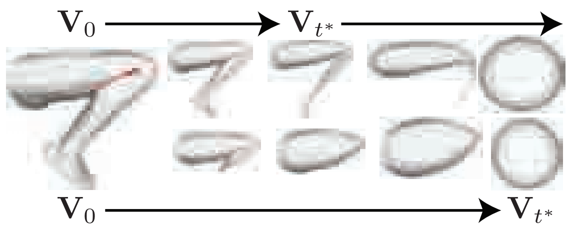

Figure 14: *Conformalized mean curvature flow (top) removes self-intersections much earlier than conformal Willmore flow (bottom).*

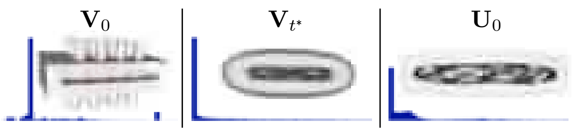

Figure 15: *The input surface* $\mathbf{V}_0$ *is taken to a self-intersection-free state* $\mathbf{V}_{t^*}$*, but the reverse flow cannot restore the original shape due to the lock caused by the external torus on the bristles. Histograms below each surface show the triangle area distribution and confirm that the result of the reverse flow was not able to restore the original areas.*

is computed anyway to remove self-intersections. This complements modern tet-meshing software and state-of-the-art bijective parameterization, forming a useful tool to assist in a variety of geometry processing tasks. We hope that by providing a method to recover volumetric discretizations for self-intersecting surfaces of solid shapes we will encourage volumetric processing at every applicable stage in the geometry processing pipeline.

Acknowledgements

The authors thank Ladislav Kavan, Diego Nehab, Tino Weinkauf, Enrico Puppo and Yotam Gingold for helpful con-

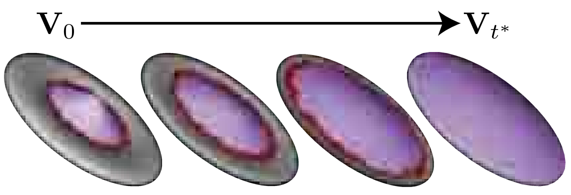

Figure 16: *This* Flying Saucer *shape is created by pulling one side through the other (back-faces are purple). The region in the middle has negative winding number: meshing it is ill-posed, so this is not valid input for our method. Interestingly, cMCF does remove all self-intersections, but flips the entire shape inside-out in the process.*

versations. We are grateful to Hang Si for making the software Tetgen available and for email discussions, Marco Attene for making Meshfix open source and Pascal Frey and the University of Paris VI for releasing Medit into the public domain. Thanks to TOSCA and Aim@Shape projects for providing triangle meshes that were used as input. We also thank Kenshi Takayama for reviewing the submitted version of this work. This work was supported in part by the ERC grant iModel (StG-2012-306877), by an SNF award 200021_137879, an Intel Doctoral Fellowship and an FCS-Switzerland Scholarship ESKAS Nr 2012.0617.

References

[Att10] ATTENE M.: A lightweight approach to repairing digitized polygon meshes. *The Visual Computer 26*, 11 (2010), 1393–1406. 3, 8

[BFA02] BRIDSON R., FEDKIW R. P., ANDERSON J.: Robust treatment of collisions, contact, and friction for cloth animation. *ACM Trans. Graph. 21*, 3 (July 2002), 594–603. 3

[BKP*10] BOTSCH M., KOBBELT L., PAULY M., ALLIEZ P., LÉVY B.: *Polygon Mesh Processing*. AK Peters, 2010. 1

[BPWG07] BOTSCH M., PAULY M., WICKE M., GROSS M.: Adaptive space deformations based on rigid cells. *Comput. Graph. Forum 26*, 3 (2007), 339–347. 2

[BWK03] BARAFF D., WITKIN A., KASS M.: Untangling cloth. *ACM Trans. Graph. 22*, 3 (2003), 862–870. 3

[CGAL] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org. 2, 5

[CPS13] CRANE K., PINKALL U., SCHRÖDER P.: Robust fairing via conformal curvature flow. *ACM Trans. Graph. 32* (2013). 8

[CPSS10] CHAO I., PINKALL U., SANAN P., SCHRÖDER P.: A simple geometric model for elastic deformations. *ACM Trans. Graph. 29*, 4 (July 2010), 38:1–38:6. 4, 5, 6

[DG03] DEY T. K., GOSWAMI S.: Tight cocone: a water-tight surface reconstructor. In *Proc. SM* (June 2003). 2

[EM09] EPPSTEIN D., MUMFORD E.: Self-overlapping curves revisited. Society for Industrial and Applied Mathematics. 3

[FP00] FREITAG L. A., PLASSMANN P.: Local optimization-based simplicial mesh untangling and improvement. *International Journal for Numerical Methods in Engineering 49*, 1-2 (2000), 109–125. 6

[FvDFH90] FOLEY J. D., VAN DAM A., FEINER S. K., HUGHES J. F.: *Computer graphics: principles and practice (2nd ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990. 3

[GR09] GEUZAINE C., REMACLE J. F.: GMSH: A 3-D finite element mesh generator with built-in pre- and post-processing . *Numerical Methods in Engineering* (2009). 2

[Har10] HARMON D.: *Robust, Efficient, and Accurate Contact Algorithms*. PhD thesis, Columbia University, 2010. 3

[HPSZ11] HARMON D., PANOZZO D., SORKINE O., ZORIN D.: Interference aware geometric modeling. *ACM Trans. Graph. 30*, 6 (2011). 2, 3

[JBK*12] JACOBSON A., BARAN I., KAVAN L., POPOVIĆ J., SORKINE O.: Fast automatic skinning transformations. *ACM Trans. Graph. 31*, 4 (2012). 5, 8

[JBPS11] JACOBSON A., BARAN I., POPOVIĆ J., SORKINE O.: Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph. 30*, 4 (2011). 7

[JKSH13] JACOBSON A., KAVAN L., SORKINE-HORNUNG O.: Robust inside-outside segmentation using generalized winding numbers. *ACM Trans. Graph. 32*, 4 (2013), to appear. 3, 7, 8

[Knu01] KNUPP P.: Hexahedral and tetrahedral mesh untangling. *Engineering with Computers 17*, 3 (2001), 261–268. 6

[KSBC12] KAZHDAN M., SOLOMON J., BEN-CHEN M.: Can mean-curvature flow be made non-singular? In *Proc. SGP* (2012). 2, 3, 4, 5

[LBB12] LITMAN R., BRONSTEIN A., BRONSTEIN M.: Stable volumetric features in deformable shapes. *Computers & Graphics 36*, 5 (2012). 2

[LBRM12] LUO L., BARAN I., RUSINKIEWICZ S., MATUSIK W.: Chopper: partitioning models into 3D-printable parts. *ACM Trans. Graph. 31*, 6 (2012). 3

[Lip12] LIPMAN Y.: Bounded distortion mapping spaces for triangular meshes. *ACM Trans. Graph. 31*, 4 (2012). 6

[LS07] LABELLE F., SHEWCHUK J. R.: Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. *ACM Trans. Graph. 26*, 3 (2007). 2

[LSR*12] LI D., SUN X., REN Z., LIN S., TONG Y., GUO B., ZHOU K.: Transcut: Interactive rendering of translucent cutouts. *IEEE TVCG 19*, 3 (2012). 2

[MGR11] MUKHERJEE U., GOPI M., ROSSIGNAC J.: Immersion and embedding of self-crossing loops. In *Proc. SBIM* (2011). 3

[MS00] MAYER U. F., SIMONETT G.: Self-intersections for the surface diffusion and the volume-preserving mean curvature flow. In *Mean Curvature Flow. Differential and Integral Equations* (2000), pp. 1189–1199. 3

[MS03] MAYER U. F., SIMONETT G.: Self-intersections for willmore flow. In *Evolution Equations: Applications to Physics, Industry, Life Sciences and Economics*. 2003, pp. 341–348. 3

[SA07] SORKINE O., ALEXA M.: As-rigid-as-possible surface modeling. In *Proc. SGP* (2007), pp. 109–116. 5

[She12] SHEWCHUK J.: Unstructured Mesh Generation. *Combinatorial Scientific Computing 12* (2012), 257. 2, 3

[Si03] SI H.: TETGEN: A 3D Delaunay tetrahedral mesh generator, 2003. `http://tetgen.berlios.de`. 1, 2, 3, 6

[SKPSH13] SCHÜLLER C., KAVAN L., PANOZZO D., SORKINE-HORNUNG O.: Locally injective mappings. In *Proc. SGP* (2013). 2, 6, 7, 8, 9

[Sor09] SORKINE O.: *Least-squares rigid motion using SVD*. Tech. rep., Courant Institute of Mathematical Sciences, New York University, Feb. 2009. 6

[SOS04] SHEN C., O'BRIEN J. F., SHEWCHUK J. R.: Interpolating and approximating implicit surfaces from polygon soup. *ACM Trans. Graph. 23*, 3 (2004), 896–904. 3

[SVW89] SHOR P. W., VAN WYK C. J.: Detecting and decomposing self-overlapping curves. In *Proc. Symp. Computational Geometry* (1989). 3, 4

[Wil00] WILLMORE T.: Surfaces in conformal geometry. *Annals of Global Analysis and Geometry 18*, 3-4 (2000). 3

[XCGL11] XU Y., CHEN R., GOTSMAN C., LIU L.: Embedding a triangular graph within a given boundary. *Comput. Aided Geom. Des. 28*, 6 (2011). 6