Locally Injective Parametrization with Arbitrary Fixed Boundaries

Ofir Weber
Bar Ilan University, Israel

Denis Zorin
New York University, USA

Abstract

We present an algorithm for mapping a triangle mesh, which is homeomorphic to a disk, to a planar domain with arbitrary fixed boundaries. The algorithm is guaranteed to produce a globally bijective map when the boundary is fixed to a shape that does not self-intersect. Obtaining a one-to-one map is of paramount importance for many graphics applications such as texture mapping. However, for other applications, such as quadrangulation, remeshing, and planar deformations, global bijectively may be unnecessarily constraining and requires significant increase on map distortion. For that reason, our algorithm allows the fixed boundary to intersect itself, and is *guaranteed* to produce a map that is injective locally (if such a map exists). We also extend the basic ideas of the algorithm to support the computation of discrete approximation for extremal quasiconformal maps. The algorithm is conceptually simple and fast. We demonstrate the superior robustness of our algorithm in various settings and configurations in which state-of-the-art algorithms fail to produce injective maps.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

Keywords: parametrization, conformal mapping, bijective mapping, quasiconformal, deformation, quadrangulation, remeshing

Links: DL PDF

1 Introduction

Computing a surface-to-plane map is one of the most fundamental tasks in geometry processing. Many computer graphics and geometry processing applications rely heavily on solving the mapping problem. Among the most important applications are: texture mapping, image and shape deformation-and-animation, finding correspondence between shapes for comparison and analysis, remeshing, mesh improvement-and-fairing, morphing, and image retargeting.

Given a triangle mesh, the goal is to find a piecewise-linear map (parametrization) to a planar domain, satisfying different types of boundary constraints. Among the typical ones are position, orientation (alignment) and curvature constraints. Depending on the application, different properties of the map are important: smoothness, amount of metric or angle distortion, and the ability to handle general boundary conditions. One of the most important properties that is often required but seldom guaranteed is injectivity.

In the smooth setting, a map with strictly positive Jacobian is locally injective. If in addition, the image of the boundary does not intersect itself, the map will be a global bijection. In the discrete setting, a piecewise-linear map of a triangle mesh will be a global bijection if two conditions hold: (1) the orientations of all the triangles in the image are positive (i.e. no flipped triangles) and (2) the boundary does not intersect itself. However, in contrast to the smooth setting, (1) alone is not sufficient for *local* injectivity. A counter example is shown in Figure 1a. A piecewise-linear map is locally injective only if the sum of (unsigned) triangle angles around each internal vertex is precisely 2π.

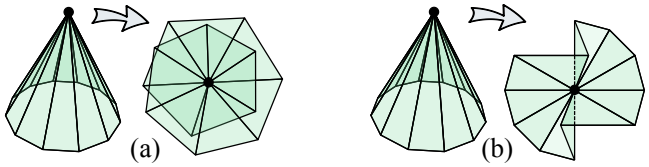

Figure 1: *Non-injective discrete maps of a cone shaped mesh to the plane with fixed boundary. (a) None of the triangles are flipped but the angle sum around the center vertex is 4π rather than 2π. (b) It is impossible to position the center vertex in a way that will produce discrete injective map.*

Even in the smooth setting, a locally injective map does not always exist, if the boundary is allowed to intersect itself. For example, there is no smooth (or discrete) locally injective map that can satisfy the boundary configuration of Figure 1a. A self-intersecting curve is called *self-overlapping*, if there exists a locally injective map with this curve as its boundary (more details in Section 3.1). Another difficulty that distinguishes the discrete setting from the smooth one is illustrated in Figure 1b. Here, the boundary is a simple curve, hence, it is obvious that a *smooth* bijective map exists. However, for this particular coarse discretization, there is only one degree of freedom that corresponds to the position of the center vertex. Regardless of how we choose to set this degree of freedom, at least one of the triangles will have negative orientation, meaning that a discrete injective solution does not exist.

Few existing algorithms guarantee global or local injectivity. Most importantly, the classic algorithm of Tutte computes a globally bijective map from a 3-connected mesh to a planar polygon with a convex boundary, and requires only a single linear solve.

Several existing algorithms for *free-boundary* parametrization with no constraints provide local injectivity guarantees, typically conditional on convergence (we discuss these algorithms in greater detail in Section 2). Fixing the entire boundary position makes the problem more difficult.

We propose an algorithm for mesh parametrization problems with arbitrary *fixed* boundaries, which *guarantees* that the resulting map is locally injective. The only restriction is that the boundary will be a *self-overlapping* polygon (otherwise, a solution does not even exist in the smooth setting). To handle connectivity problems similar to Figure 1b we relax the original problem by allowing mesh refinement. The amount of refinement needed by our algorithm is bounded; although the worst-case bound is quadratic in the number of mesh vertices, the observed number of added vertices is very small. It is important to note that even though the algorithm may change the *connectivity* of the input mesh, the actual *shape* of the mesh is never altered.

The main idea of our algorithm is quite simple. First, we construct a triangulation (denoted as target mesh) of the user prescribed fixed boundary polygon. Then, we compute two globally bijective maps of the source and target meshes into an intermediate convex domain. Finally, the source-to-target map is obtained by composing the first map with the inverse of the second one.

The concept of using two bijective maps to a common intermediate domain has been already explored in the context of finding dense correspondence between two meshes with different connectivity [Kanai et al. 1997; Lipman and Funkhouser 2009]. We improve on this approach in several important ways: we demonstrate how domains with self-overlaps can be handled (Section 3.1), we show how to minimize mesh refinement needed for the map to be locally injective (Section 4), and we demonstrate how to extend the basic setup to compute discrete approximations of extremal quasiconformal maps.

We show how our algorithm can be used to ensure local injectivity in a number of different settings. We further compare the results of our algorithm with the state-of-the-art algorithms and demonstrate the uncompromised robustness of our algorithm in cases where other state-of-the-art algorithms fail.

2 Related work

Detailed reviews of parametrization literature can be found in [Hormann et al. 2007; Sheffer et al. 2006]; we are primarily concerned with the work related to injectivity. The idea of [Tutte 1963], introduced to geometric modeling in [Floater 1997] is one of the few algorithms that provides guarantees on injectivity; it is limited to maps to convex domains. [Kanai et al. 1997] uses a variant of Tutte's algorithm to establish correspondence between two meshes with different connectivity. This was done by mapping both meshes to a common convex domain, an idea that we also use. [Lipman and Funkhouser 2009] established bijective correspondence between meshes by mapping both source and target to a unit disk, using a discrete conformal map.

Conformal methods. Another setting for which parametrization algorithms with guarantees were proposed is *free-boundary* parametrization. To the best of our knowledge, the only algorithm that has injectivity guarantees, bounds on refinement *and convergence guarantees* is the algorithm of [Kharevych et al. 2006]. However, it requires an intrinsic Delaunay triangulation of a surface, which may not be a regular triangulation. ABF [Sheffer and de Sturler 2001] and [Sheffer et al. 2005] guarantee injectivity, if a solution is found. As these methods require solving a nonlinear nonconvex problem, they provide no guarantees. Finally, [Springborn et al. 2008], describes a convex optimization formulation for conformal mapping in terms of per-vertex scale factors. However, the problem requires nonconvex constraints to guarantee the existence of a planar mesh for the optimal choice of scale factors.

Fixed boundary methods. MIPS parametrization [Hormann and Greiner 2000] is a nonlinear method that aims to reduce angular distortion and can handle fixed position constraints. However, the MIPS energy is nonlinear and nonconvex. As-rigid-as-possible (ARAP) parametrization [Liu et al. 2008], measures deviation of the Jacobian from being a rotation matrix. The corresponding energy is also nonconvex and even when the global minimum is reached, local injectivity is not guaranteed. [Xu et al. 2011] uses a nonlinear method to embed a triangle mesh in a given simple planar boundary. This is done by minimizing the unsigned area of the planar triangulation, which is a nonsmooth nonlinear function. The method produce bijective maps in cases where the starting point is close to being bijective but suffer from numerical issues otherwise.

[Schüller et al. 2013] show how to modify any deformation energy to guarantee a locally injective map. This is done by introducing a barrier term to prevent flipped triangles. For the method to work successfully, an initial map with no flipped triangles should be used. In order to use this method for parametrization with arbitrary (nonconvex) fixed boundary, one has to first obtain a valid parametrization, and then continuously deform the boundary till it is aligned with the user prescribed boundary. For this to succeed, the boundary at each step must be self-overlapping. The Composite Mean Value algorithm [Schneider et al. 2013] produces smooth (as opposed to discrete) globally bijective maps from one planar simple polygon to another. Similar to [Schüller et al. 2013], the algorithm requires a smooth deformation of the boundary from source to target. In contrast to [Schüller et al. 2013], which considers injectivity locally, here the source and target polygons, as well as all the inbetween polygons, must be simple.

The work of [Lipman 2012], aims at producing piecewise-linear maps of triangle meshes with a bounded amount of conformal distortion. Forcing the distortion of a triangle to be smaller than a given threshold is a nonconvex inequality constraint, which is convexified, at the expense of shrinking the feasible space. The strength of [Lipman 2012] is that the chosen convex subspace is shown to be optimal, in the sense that it is the largest convex subspace contained in the (nonconvex) full space. However, in extreme cases it is possible that the method will not find a bijective solution even when one exists. [Bommes et al. 2013] uses a similar idea, but the specific convex subspace being used is different.

Quasiconformal maps are locally injective maps with a bounded amount of conformal distortion. Consider the space of all quasiconformal maps from a given source domain to a target domain, satisfying a set of boundary constraints. An extremal quasiconformal map f is a map within this space, that has the smallest maximal distortion. Formally, f minimizes the following nonlinear, nonconvex, nonsmooth functional: $\inf_f(\sup_x(k(x)))$ where k is the conformal distortion. Unlike conformal maps, these maps are flexible enough to allow for position constraints to be prescribed and can handle multiply-connected domains as well as surfaces with high genus [Ahlfors 1966; Gardiner and Lakic 2000]. Recently, [Weber et al. 2012] described a practical algorithm for the computation of discrete approximation to extremal quasiconformal maps. Although the method approximates extremal maps well in the average sense, injectivity is not guaranteed. In Section 5.3, we show how to combine this method with our method, in order to guarantee locally injective maps.

As far as we know, no algorithms with injectivity guarantees were proposed for arbitrary fixed boundaries.

3 Algorithm

Overview of the algorithm. The input of our algorithm is a mesh of disk topology to parametrize and a target boundary planar polygon P (which may be self-intersecting), optionally annotated with angle information as explained in detail in Section 3.2.

The algorithm consists of two main parts: the first part constructs a triangulation of the target boundary polygon P (in the sense defined below), simultaneously determining if the polygon can be the boundary of a locally injective map for *any* mesh. As a second step, we construct two bijective maps to an intermediate domain, from the triangulated target boundary and the original mesh, and use their composition to find the source-to-target map. In the process, the original mesh is possibly refined adaptively to guarantee injectivity.

3.1 Admissible inputs and self-overlapping polygon triangulation

The input to our algorithm is an arbitrary mesh M of disk topology and a piecewise-linear map f_b from the boundary of M to the plane. We assume that no triangle of M is degenerate. We compute a locally injective piecewise-linear map f from the mesh M_r (obtained by refining M), to the plane, such that $f|_{\partial M_r} = f_b$, and the amount of refinement is minimized.

Denote by Θ_i the sum of triangle angles around vertex v_i at the image of the map f. f is locally injective if all triangles in the image have positive orientation, $\Theta_i = 2\pi$ for each internal vertex, and $\Theta_i \in (0, 2\pi)$ for each boundary vertex.

Existence of solutions. The first question we ask is: under what conditions on f_b the problem has a solution? A necessary, but not sufficient condition for a related problem for smooth curves first

appeared in [Whitney 1937], and more complete algorithmic tests were described in [Marx 1974]. The foundation of our approach is [Shor and Van Wyk 1992], which considers *self-overlapping* curves and polygons.

As we are interested in piecewise-linear maps of meshes, we specialize definitions to this case. We say that a polygon P is self-overlapping, if there is (any) mesh M, homeomorphic to a disk, and a piecewise-linear locally injective map f from M to the plane, such that $f(\partial M) = P$ (see Figure 2 for illustration).

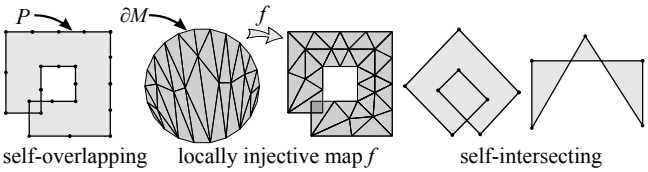

Figure 2: *Self-overlapping and non-self-overlapping polygons. (left to right) A self-overlapping polygon P. A locally injective map f from a mesh M satisfying $f(\partial M) = P$. Two self-intersecting polygons which are not self-overlapping.*

While natural for smooth curves, this definition is not entirely natural for polygons. Start with a planar mesh M, and consider continuous deformation of this mesh, such that for any time t, $f(M,t)$ is nondegenerate (i.e. no triangle has zero area). It is natural to include $f(\partial M,t)$ in the class of polygons we consider. Yet, as shown in Figure 3, not all such polygons are included in the class that we consider, because they may lack injectivity at the boundary.

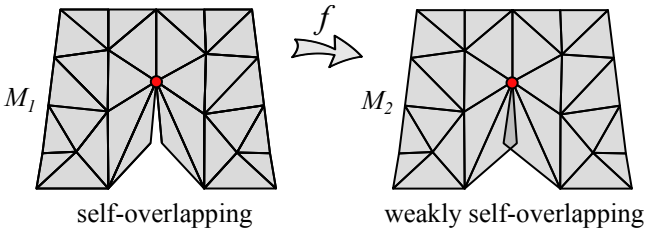

Figure 3: *Weakly self-overlapping polygon. (left) The "pants" mesh is bounded by a simple polygon. (right) The boundary polygon is slightly perturbed to form a self-intersecting polygon which is not self-overlapping. While there is no locally injective map from M_1 (or any other mesh) to M_2, f is locally injective everywhere beside at the singular (marked red) boundary vertex. The boundary of M_2 is called a weakly self-overlapping polygon.*

Instead we consider a broader class of polygons which we call *weakly self-overlapping*. We use the same essential definition, except that the map f is only required to be locally injective in the interior of M. Specifically, a polygon P is weakly self-overlapping, if there is a map f from (any) mesh M, homeomorphic to a disk, such that $f(\partial M) = P$, all triangles are mapped with positive orientation, and $\Theta_i = 2\pi$ for each internal vertex.

Proposition 1. *If a polygon P is weakly self-overlapping, for any mesh M and a map f, such that $P = \partial f(M)$, there is a simplified mesh M' with no interior vertices and a map f' which is locally injective in the interior of M' and satisfies $P = \partial f'(M')$.*

[Shor and Van Wyk 1992] shows that the property holds for self-overlapping polygons, we extend it to *weakly* self-overlapping polygons. The proof is given in Appendix A. Proposition 1 allows us to drop the condition $\Theta_i = 2\pi$ for internal vertices, which greatly simplifies the task of determining whether a polygon is weakly self-overlapping or not. Determination is done by checking whether the polygon can be triangulated with positively oriented triangles and no internal vertices.

We assume that our input polygon P is weakly self-overlapping; then we guarantee that for a refinement of the original mesh, a locally injective parametrization with P as the boundary will be constructed. Note that this assumption is not tautological: we say that we can construct a locally injective map from a *given* mesh, if it exists for *any* mesh at all.

Triangulation of weakly self-overlapping polygons. Next, we describe an algorithm (a version of Shor-van Wyck algorithm), that determines if a polygon P is weakly self-overlapping, and simultaneously triangulates it. In the next section, we extend the algorithm to support a more complex setting, where exact prescription of singular boundary vertices is given.

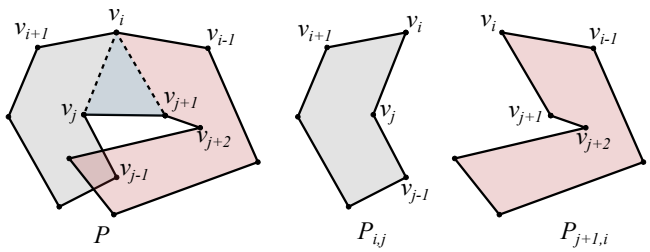

Figure 4: *The weakly self-overlapping polygon P is split into two weakly self-overlapping polygons, $P_{i,j}$ and $P_{j+1,i}$.*

Suppose v_i are the vertices of the polygon P enumerated counterclockwise. Note that one can always find a triangle in a triangulation $\mathcal{T}$ of P with one side on the boundary; removing this triangle splits $\mathcal{T}$ into two separate triangulations with no interior vertices. So for any weakly self-overlapping polygon, it is possible to find 3 vertices v_i, v_j, v_{j+1} on the boundary, two of them adjacent, such that removing the edge (v_j, v_{j+1}), replicating vertex v_i and adding edges (v_i, v_{j+1}) and (v_j, v_i), yields two separate weakly self-overlapping polygons (Figure 4). Furthermore, note that a triangle is weakly self-overlapping if and only if it has positive (counter-clockwise) orientation. So for a given polygon, one can determine if it is weakly self-overlapping, by recursively splitting it in all possible ways by valid choices of positively oriented triangles $\triangle(i, j, j+1)$ and checking if each part is weakly self-overlapping. This yields an algorithm for determining if a polygon is weakly self-overlapping (and simultaneously generating a triangulation), which is a simplified version of the Shor-van Wyck algorithm (for which testing the conditions ensuring local injectivity on the boundary adds complexity).

Shor-van Wyck algorithm summary. A subpolygon $P_{i,j}$ of P is formed by joining all edges that lie between v_i and v_j in counterclockwise order: $i, i+1, \ldots, j-1, j$ and then closing the polygon by adding an edge that connects v_j to v_i (see Figure 5 left).

The algorithm constructs a table Q of size $n \times n$ (n is the number of vertices in P), so that $Q_{i,j}$ is 1 if the subpolygon $P_{i,j}$ is weakly self-overlapping. Note that $P_{i,i+1}$ is a polygon with only two vertices, so by convention, $Q_{i,i+1} = 1$. Also note that $P_{i,i-1}$ is equivalent to P. This means that if $Q_{i,i-1} = 1$, for any i, the whole polygon can be triangulated, so the algorithm can terminate as soon as one of these is found.

The table Q is updated using dynamic programming. More constructively, one proceeds in the order of increasing counterclockwise distances d between i and $j = (i+d) mod\ n$, starting with $d = 2$. In order to set $Q_{i,j}$, the algorithm searches the table for a splitting vertex k. Vertex k is a splitting vertex, if the orientation of $\triangle(i,k,j)$ is positive, $Q_{i,k} = 1$, and $Q_{k,j} = 1$ (see Figure 5 left). Note that the entries $Q_{i,k}$ and $Q_{k,j}$ are already updated at this point since the distance between i and k, and the distance between k and j are smaller than d. The splitting vertices are stored in a second table for future reference.

Once Q is fully updated, the subdiagonal elements, $Q_{i,i-1}$ are checked. If no subdiagonal element is 1, the algorithm declares

that the polygon is not weakly self-overlapping. Otherwise, the triangulation itself can be reconstructed in a top down manner. Suppose i is the first value such that $Q_{i,i-1} = 1$. Then add the triangle $\triangle(i,k,i-1)$ to the triangulation and continue recursively to triangulate $P_{i,k}$ and $P_{k,i-1}$.

The complexity of the algorithm is $O(n^3)$, where n is the size of the boundary polygon. For a typical mesh, the size of the boundary is $O(\sqrt{m})$, where m is the size of the mesh, so in practice, the complexity is $O(m^{1.5})$, which is the same as that of a direct linear solver (which will be used in Section 4).

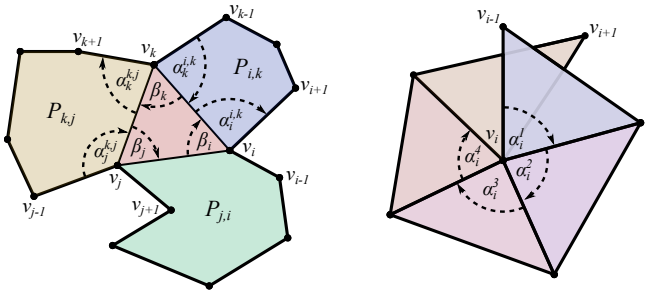

Figure 5: *Notations for the triangulation algorithm.*

3.2 Angle compatibility

In some cases, it is important to distinguish between different valid triangulations of P. A weakly self-overlapping polygon may have *singular* vertices on the boundary for which the map is not locally injective. It is often useful to control which vertices are allowed to be singular. We now extend the basic algorithm to support precise prescription of singularities.

Let α_i be the clockwise angle between the vectors $v_i v_{i-1}$ and $v_i v_{i+1}$, in the range $(0, 2\pi]$ (i.e., the angle on the left side of the polygon). For every vertex v_i of the boundary of the triangulation $\mathcal{T}$, let $\triangle_i^j$ be the triangles of $\mathcal{T}$ sharing the vertex i. Let α_i^j be the angles at v_i of $\triangle_i^j$ (see Figure 5 right).

We call $\alpha_i' = \sum_j \alpha_i^j$ the *triangulation angle* of $\mathcal{T}$ at v_i. Note that in general, $\alpha_i' \neq \alpha_i$, however, because all triangles are oriented in the same way, $\alpha_i' = \alpha_i + 2R_i\pi$. So if P is given, the triangulation angles can be defined by assigning additional integers (*full rotation indices*) R_i to each vertex. We say that triangulations $\mathcal{T}_1$ and $\mathcal{T}_2$ are *angle-compatible*, if their triangulation angles agree at all vertices.

For example, with a parametrization intended for quadrangulation (cf. [Kälberer et al. 2007; Bommes et al. 2009]), some vertices on the boundary of the domain are cones with angles $m\pi/2 + 2\pi R_i$, where $m = 0,1,2,3$. The cones of real interest are $3\pi/2$ and $5\pi/2 = \pi/2 + 2\pi$, so there is only one commonly occurring case with $R_i \neq 0$, $R_i = 1$.

Constructing a triangulation compatible with a given set of triangulation angles. We would like to make sure that the triangulation we construct has given triangulation angles. The simplest approach is to consider joining two subpolygons valid only if the resulting angle changes are compatible with the given triangulation angles.

At each step, the triangulation of a polygon $P_{i,j}$ is obtained from subpolygons $P_{i,k}$ and $P_{k,j}$ by gluing along $\triangle(i,k,j)$ (Figure 5 left). Let the angles of $\triangle(i,k,j)$ be β_i, β_k, β_j. The assumption is that for a subpolygon $P_{i,j}$ all the triangulation angles are identical to those of P (as provided by the user) beside the two triangulation angles at i and j. The algorithm maintains two additional $n \times n$ tables, to keep track of these two triangulation angles for each subpolygon.

When the gluing of $P_{i,j}$ happens, only two triangulation angles are updated, at vertices i and j. The triangulation angles needed for the update are: $\alpha_i^{i,k}$, $\alpha_k^{i,k}$, $\alpha_k^{k,j}$, and $\alpha_j^{k,j}$. The superscript indicates which subpolygon this triangulation angle belongs to, and the subscript indicates the vertex index (see Figure 5 left).

Then the triangulation angles of $P_{i,j}$ that needs to be updated are given by:

- $\alpha_i^{i,j} = \alpha_i^{i,k} + \beta_i$;
- $\alpha_j^{i,j} = \alpha_j^{k,j} + \beta_j$.

Gluing $P_{i,k}$ and $P_{k,j}$ is valid, if and only if the following conditions hold:

- $\alpha_i^{i,j} \leq \alpha_i'$;
- $\alpha_j^{i,j} \leq \alpha_j'$;
- $\alpha_k^{k,j} + \beta_k + \alpha_k^{i,k} = \alpha_k'$.

In practice, it is highly undesirable to keep track of angles, due to numerical accuracy problems. Instead, it is sufficient to keep track of the full rotation indices R_i, and use robust orientation tests with exact predicates [Shewchuk 1996] to determine how to update full rotation indices when gluing polygons. Detailed explanation is given in Appendix B. Note that this algorithm includes the algorithm of Shor-van Wyck as a special case, which corresponds to having all full rotation indices set to zero.

Acceleration of the triangulation algorithm. As explained in [Shor and Van Wyk 1992], it is possible to simplify a self-overlapping curve without altering its self-overlappingness as long as self-intersection points are preserved, and no new intersection points are created. This decreases the complexity of the triangulation algorithm to $O(k^3)$, where k is the number of self-intersections. A simplification technique for self-overlapping curves is provided in [Shor and Van Wyk 1992]. However, rather than trying to adapt this technique to the *weakly* self-overlapping case, we pursued a different strategy for polygon simplification which is easier to understand and implement. Our technique is based on the "ear clipping" algorithm [Eberly 1998] which is one of the simplest polygon triangulation methods. While, it is not guaranteed that our procedure will reduce the polygon to $O(k^3)$, in practice it provides considerable acceleration (up to two order of magnitudes). We describe this optional acceleration procedure in Appendix C.

4 The remapping algorithm

The second stage of our algorithm is the construction of two globally bijective maps to a common intermediate domain, one from the source mesh and the other from the triangulation of P constructed at the first stage. Then, a locally injective map is being constructed by composing two bijective maps and refining the original mesh (if necessary).

Let the source mesh be M, and let Ω_i be a polygonal intermediate domain with boundary B, with the number of edges matching the boundary of M. Let $h_{s\to i}$ be a piecewise-linear globally bijective map from the source M to the intermediate domain Ω_i, mapping vertices of ∂M to vertices of B one-to-one. M_t is the target mesh, obtained by triangulating P, and $h_{t\to i}$ is a piecewise-linear globally bijective map from the target mesh M_t to the same intermediate domain Ω_i (illustrated in Figure 6).

For the purposes of our construction, the shape of the intermediate domain Ω_i, and the manner in which $h_{s\to i}$ and $h_{t\to i}$ are constructed are irrelevant, as long as $h_{s\to i}$ and $h_{t\to i}$ are bijective. However, currently the only mapping algorithm that can fully guarantee a globally bijective map with fixed boundary is the algorithm of Tutte's and its variants [Tutte 1963; Floater 1997]. In order to obtain a Tutte map, one has to fix the boundary vertices to form a strictly convex polygon (in a one-to-one manner). The position of the internal vertices is obtained by solving a single linear system, where each vertex is positioned at a convex combination of its neighbor

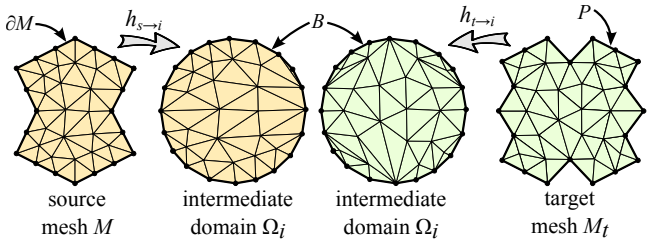

Figure 6: *Two globally bijective maps to a common intermediate domain. $h_{s\to i}$ maps the source mesh M to the intermediate domain Ω_i and $h_{t\to i}$ maps the target mesh M_t to the same intermediate domain.*

vertices. We choose B to be a polygon with evenly spaced vertices on a unit disk, and $h_{s\to i}$ and $h_{t\to i}$ are chosen to be Tutte's maps.

Meshing the target domain. The triangulation constructed at Section 3.2 provides a coarse mesh bounded by P with no internal vertices. We would like to replace the triangulation of this mesh with another, *angle compatible* triangulation, which is finer. The bijectivity of the result does not depend on how coarse or fine the mesh is, but the quality of the map does. As illustrated in Figure 7b, the initial triangulation may contain badly shaped triangles, so we first simplify this mesh by removing as many internal edges as possible. An internal edge can be removed if the union of the two subpolygons that shares that edge is a simple polygon. This simplification process result in a partition of P into a typically small number of simple subpolygons. The interior edges of this partition are then refined. Finally, each simple subpolygon is independently triangulated to obtain a high quality triangulation, while preserving the boundary edges (we use [Shewchuk 2005]). The meshing process is illustrated in Figure 7. For all the examples in this paper, we set the density of the target mesh to be the same as the source mesh. The effect of using different densities on the quality of the final map is illustrated in Figure 8.

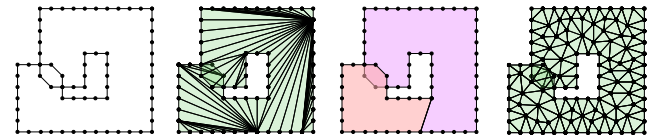

Figure 7: *Meshing the target domain. (a) The input self-overlapping polygon. (b) Coarse mesh. (c) The mesh is simplified into a partition with two simple subpolygons by repeatedly removing internal edges. (d) Each subpolygon is refined to obtain a high quality triangulation.*

The composition $f = h_{t\to i}^{-1} \circ h_{s\to i}$ defines a bijective map from M to M_t. However, this map is not piecewise-linear on triangles of M. Let M_{join} be the polygonal mesh on the intermediate domain obtained by intersecting $h_{s\to i}(M)$ and $h_{t\to i}(M_t)$. Then $f = h_{t\to i}^{-1} \circ h_{s\to i}$ is piecewise-linear on $h_{s\to i}^{-1}(M_{join})$, i.e. on a refinement of M obtained by mapping M to the intermediate domain and intersecting with the image of M_t. The worst-case complexity of M_{join} is $O(m^2)$ where m is the total number of vertices in two meshes, but worst-case scenarios are unlikely for large m. Nonetheless, even in a typical situation, M_{join} is likely to have an order of magnitude more triangles compared to M and M_t, and many triangles are likely to be poorly shaped. Instead, we use $h_{t\to i}(M_t)$ to guide the refinement of M to make f injective.

Source mesh refinement. To motivate our decision to allow mesh refinement, we emphasise that for a mesh with a given connectivity, some configurations of the boundary B do not permit an injective map. A straight forward example is shown in Figure 1b. However,

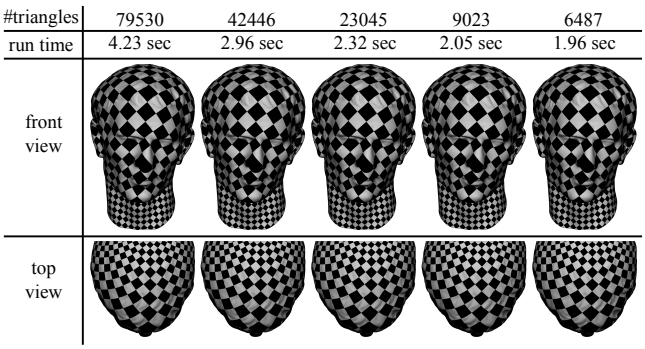

Figure 8: *Target mesh density variation. The Julius model with* 39164 *triangles is parameterized using our method with different target mesh densities. As evident from the visual results, the resolution of the target has little effect on the overall map quality.*

similar problems can occur for arbitrarily large meshes. For example, consider the snake-like mesh that appears in Figure 9a. Our goal then is to construct a bijective map, which is piecewise-linear on a refined mesh M_r, with minimum amount of refinement.

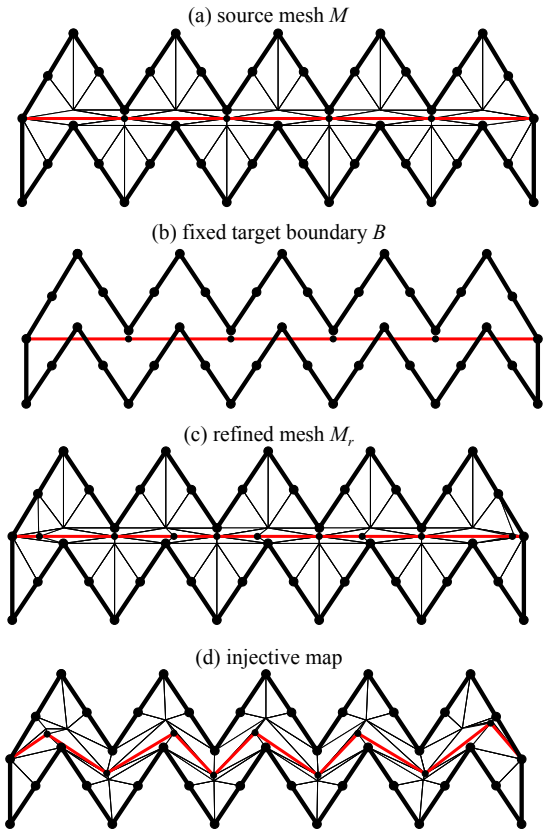

Figure 9: *Mandatory refinement for a planar deformation example. (a) A snake-like source mesh. (b) The boundary undergoes a very small deformation. Nevertheless, it is impossible to position the chain of 5 red edges inside the deformed target polygon. (c) Our algorithm automatically refine M by adding 5 new vertices along the red edge chain. (d) An injective map where the refined red chain lies fully inside the polygon.*

Consider the image of a triangle $\triangle$ of the source mesh M under the map f (Figure 10). $h_{s\to i}(\triangle)$ is still a triangle in Ω_i, as $h_{s\to i}$ is piecewise-linear on M. If we insert all points obtained by intersecting edges of $h_{t\to i}(M_t)$ with the edges of the triangle in Ω_i, we can

compute the exact image $f(\triangle)$ by evaluating f at all intersection points. The image is a "curved" triangle, which may potentially have self intersections, but is guaranteed to be a self-overlapping polygon.

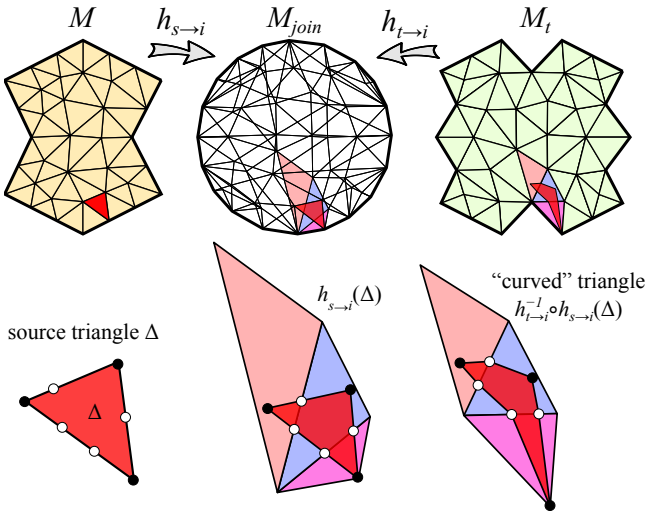

Figure 10: *The image of a triangle $\triangle$ under $f = h_{t\to i}^{-1} \circ h_{s\to i}$. (top row) The source and target meshes and the intersection of both maps at the intermediate domain. (bottom row) Zooming in on the red triangle at all three domains. The black points are the original vertices. The white points are generated by intersecting $h_{s\to i}(\triangle)$ with $h_{t\to i}(M_t)$.*

In order to efficiently invert $h_{t\to i}$, as well as for computing intersections robustly, we use CGAL 2D Arrangements with exact predicates and exact constructions [Wein et al. 2013]. We start with a piecewise linear map f^r on M, obtained by evaluating f on every vertex v of M. If all the triangles of M are mapped by f^r to triangles with positive orientation, the algorithm terminates and there is no need to perform any refinement. Otherwise, we use the following refinement algorithm.

```
place all triangles △ of M having f^r(△) with negative orientation on a stack
while stack not empty do
    pop the first polygon p from the stack
    I = the intersection points of h_{s→i}(p) with h_{t→i}(M_t)
    refine p by adding I to p
    let p_1, p_2, p_3 be the neighboring polygons across p edges
    for all i ∈ {1,2,3} do
        if f^r(p_i) is not self-overlapping then
            push p_i into the stack
        end if
    end for
end while
```

The above algorithm terminates, because if all points of intersection are inserted, the images of all triangles of M are self-overlapping. Note that all the new vertices added by the algorithm lie on the original edges of the mesh M. Hence, the refined triangles of M become polygons, but maintain exactly the same triangular shape.

It is typically possible to further reduce the number of points added by the refinement algorithm. Next, we describe a simplification procedure that strives to remove as many as possible unnecessary points from M_r. Obviously, we only try to remove points that were added by the refinement algorithm, leaving out the original vertices of M. Since new points are never inserted on the boundary, each new point belongs to exactly two polygons on M_r. Let these polygons be p_1 and p_2. A point can be removed from M_r if $f^r(p_1)$ and $f^r(p_2)$ remains self-overlapping after its removal (Figure 11).

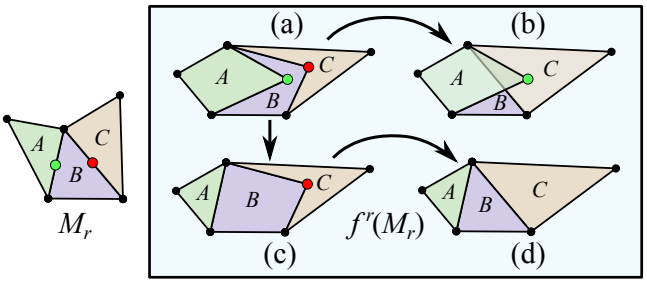

Figure 11: *The simplification procedure. (left) A mesh M_r with three triangles. Two points (red and green) were added during the refinement step. (a) The image of M_r under the map f^r before simplification. (b) Removal of the red point is illegal since it causes polygon B to stop being self-overlapping. (c) Removal of the green point is possible since both neighboring polygons (A and B) stays self-overlapping. (d) Once the green point is removed, the red one can be removed as well, and the algorithm terminates since all added points are removed.*

The simplification procedure goes over all previously added points, removing each point that does not violates the condition described above. In some cases, a point that could not be removed at a certain time, becomes removable after the removal of another point (Figure 11). For that reason, we repeat the above procedure until no more points can be removed. Typically, 1-4 simplification rounds are required to reach a state where no more points can be removed.

Upon termination of the algorithm, we convert the refined mesh M_r into a triangle mesh. Each refined triangle and its "curved" triangle image are triangulated in a compatible manner to obtain the final mesh (Figure 12). It is important to note that while M_r may have more vertices than M, both meshes have identical geometry.

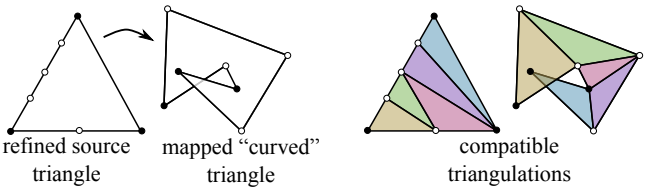

Figure 12: *Compatible triangulation. (left to right) A refined triangle of the mesh M_r. The "curved" triangle image (which is a self-overlapping polygon). Compatible triangulations of these two polygons.*

Summary. To summarize, this part of the algorithm requires: (a) Meshing the target domain M_t. (b) Computing two Tutte's maps $h_{s\to i}$ and $h_{t\to i}$ which amounts to solving two linear systems. (c) Refining M adaptively to obtain M_r and evaluating the map on each vertex of M_r. (d) Triangulating the faces of M_r.

5 Map quality

Till now, we focused mainly on map validity without paying much attention to the quality of the map. Our algorithm succeeds as long as the two maps, $h_{t\to i}$ and $h_{s\to i}$, are bijective. However, we have a lot of freedom in selecting the intermediate domain Ω_i as well as the maps $h_{t\to i}$ and $h_{s\to i}$.

5.1 Dual harmonic maps

Radó's theorem [Duren et al. 2004] states that a smooth harmonic map of a surface S to a convex planar domain will be a global bijection (even for highly curved S). We call a dual harmonic smooth map, a map of the form $g_{s\to t} = h_{t\to i}^{-1} \circ h_{s\to i}$, where $h_{t\to i}$ and $h_{s\to i}$ are smooth harmonic maps, and Ω_i is convex.

Dual harmonic maps possess various appealing properties. Most importantly, since $h_{t\to i}$ is bijective, so is $h_{t\to i}^{-1}$ and $g_{s\to t}$. Moreover, $g_{s\to t}$ is C^∞ which is very important for graphics applications (denoted as the *smoothness* property). Let $g_{t\to s}$ be the inverse of $g_{s\to t}$ (i.e., $g_{t\to s} = h_{s\to i}^{-1} \circ h_{t\to i}$), then $g_{t\to s}$ is also dual harmonic (the *symmetry* property). Note that conformal and extremal quasiconformal maps also possess this property, but regular harmonic maps do not. Finally, if the source domain Ω_s is planar and $g_{s\to t}(\partial\Omega_s) = Id$, then $g_{s\to t} = Id$ (the *reproduction* property).

In order for our *discrete* maps to follow the properties of smooth dual harmonic maps, we replace the two Tutte's maps used before with discrete approximation of harmonic maps (obtained by the standard cotangent weights FEM discretization [Pinkall and Polthier 1993]). A sufficient condition for a discrete harmonic map to a convex domain to be bijective is that all Laplacian weights are positive. Unfortunately, this is not the case for a general triangle mesh. Nevertheless, we point out that having strictly positive weights is sufficient, but not a necessary condition for bijectivity, and that in practice (for fine meshes), the map turns out to be bijective even at the presence of (many) negative weights. In the exceptional case that flipped triangles do appear, we treat the obtained parametrization (with foldovers) as a new mesh, and map it again to the same convex domain, only this time we use positive mean value weights [Floater 2003]. The choice not to use mean value weights directly, is related to its lack of convergence properties.

Figure 13 demonstrates the reproduction property of our discrete dual harmonic maps, where a planar mesh is being deformed by our algorithm, such that the boundary of the target is identical to that of the source. The source and target domains are highly concave, and their images under the discrete harmonic maps is highly distorted. Nevertheless, the obtained source-to-target map, approximates the identity map with high fidelity.

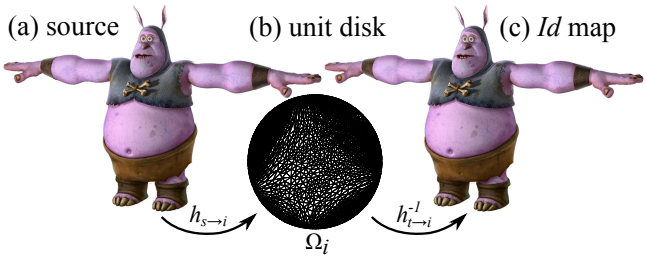

Figure 13: *Identity reproduction of discrete dual harmonic map. (a) A planar mesh M with* 3400 *triangles and texture applied. (b) Discrete harmonic map $h_{s\to i}(M)$ to a unit disk. Note how extreme the deformation is due to the dramatic difference between ∂M and the boundary of the disk. (c) The deformed mesh, using the same texture coordinates as in (a). The two images of the troll are visually indistinguishable.*

In Figure 14, we show a planar deformation example using the technique described in this section. The user prescribes a self-overlapping polygon as the boundary of the deformed raptor shape. Our dual harmonic map algorithm produces a smooth and natural deformation with very low conformal distortion and without introducing mesh refinement.

Figure 15 contains another deformation example for which the prescribed target polygon is *weakly* self-overlapping. The head and tail of the horse are bent sharply, creating two singular vertices.

Figure 16 shows a comparison of parametrization obtained using the method of [Springborn et al. 2008] and our dual harmonic map. A seam is introduced such that the cut meshes become homeomorphic to a disk. We extract the boundary position obtained from [Springborn et al. 2008] and feed it to our algorithm.

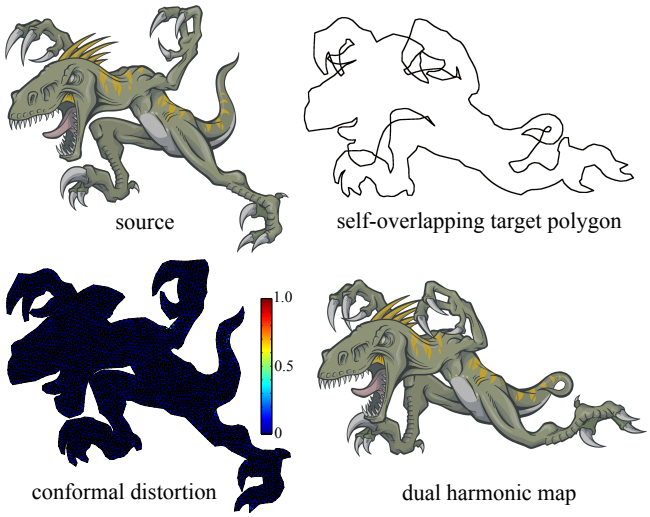

Figure 14: *Image deformation. The raptor source planar mesh with* 4554 *triangles is being deformed. The user prescribes the shape of the deformed boundary such that it intersect itself. Our dual harmonic map algorithm produces a smooth and natural deformation of the image within a total time of 0.524 sec. The color visualization shows the conformal distortion of the map. No mesh refinement was needed.*

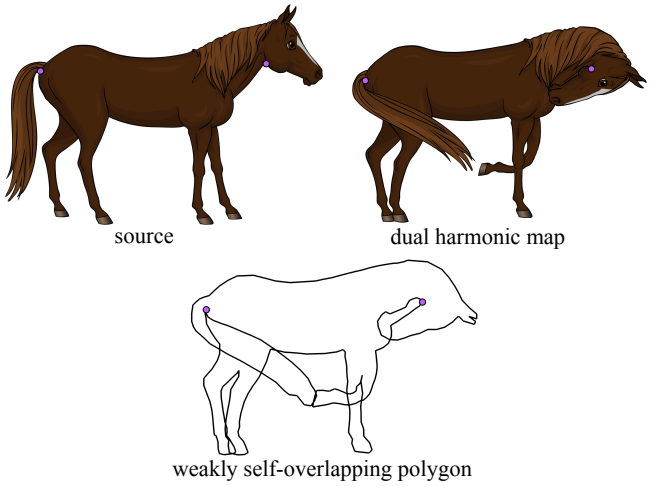

Figure 15: *Deformation of the horse. A sharp bend of the head and tail leads to a weakly self-overlapping polygon with two singular vertices (marked pink). Each singular vertex has full rotation index* 1*, meaning the triangulation angle at these vertices is in the range* $[2\pi, 4\pi)$*. The obtained dual harmonic map is natural and smooth, even in the vicinity of the singular vertices.*

5.2 Variable metric

Dual harmonic maps possess various appealing properties (injectivity, smoothness, symmetry, reproduction). However, depending on the application, additional properties of the parametrization, such as low isometric or angle distortion, are desired. We next explain, how dual harmonic maps can be combined with other mapping algorithms to ensure injectivity while maintaining other properties.

The idea is to use a target parametrization (e.g., a parametrization with low isometric distortion) to define a metric on the surface. More specifically, let $T : M \to \mathbb{R}^2$ be a parametrization, and assign lengths $\ell_i = |T(v_m) - T(v_n)|$ to all edges $e_i = (v_n, v_m)$. Let M_ℓ be the mesh M equipped with a metric $\ell = (\ell_1, \ldots \ell_N)$, where N is the number of edges. Recall that the entries of the discrete Lapla-

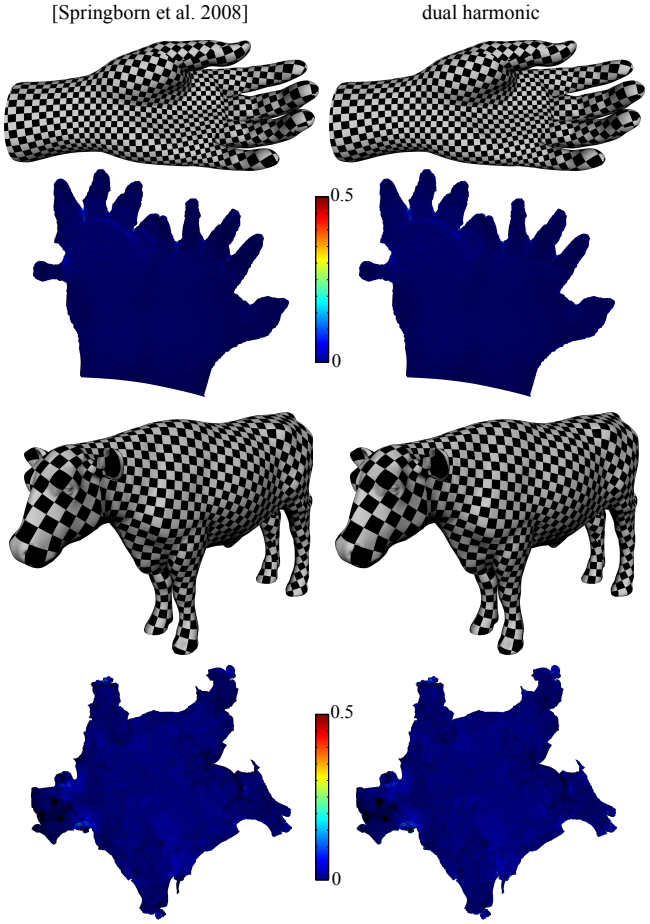

Figure 16: *Parametrization of the hand and bull models with sphere topology. (left) [Springborn et al. 2008]. (right) Our dual harmonic map. We fix the boundary of the dual harmonic maps based on the result of [Springborn et al. 2008], such that both methods have identical boundary. The discrete conformal maps and the dual harmonic maps are remarkably similar and possess low amount of conformal distortion. No refinement was needed for the dual harmonic maps.*

cian matrix can be computed from the edge lengths only, so one can compute harmonic maps on M_ℓ.

If T is locally injective, then ℓ is a flat metric, and by solving for a dual harmonic map h on M_ℓ with boundary conditions $h(v) = T(v)$ for vertices $v \in \partial M$ *we obtain the map T itself.*

Indeed, the Laplacian matrix L_ℓ we construct on M, is identical to the matrix L_T we would obtain from the mesh $T(M)$, thus, the values of the first harmonic map in a dual harmonic map satisfy the same equations whether the domain is M_ℓ or $T(M)$. If on $T(M)$ we specify identity boundary conditions $h_T(v) = v$, and solve for a dual harmonic map $h_T : T(M) \to \mathbb{R}^2$, then, by the reproduction property, the whole of h_T is identity. The values of h with the same boundary conditions, i.e. $h(v) = T(v)$ for any $v \in \partial M_\ell$, and the same intermediate domain, will be the same at vertices of M_ℓ as at corresponding vertices of $T(M)$, i.e. $h(v) = T(v)$.

Obviously, reproducing an already injective map T is not very useful. However, suppose T has some desired properties but is *almost* injective, as it is often the case. Then the metric ℓ is no longer flat (vertices shared by triangles with opposite orientation may have nonzero curvature) and the dual harmonic map $h = h[T]$ will be different from T. However one can expect to have only a small difference, between these maps, while $h[T]$ is guaranteed to be injective (in contrast to T). The nonlinear operator $P : T \to h[T]$ can be viewed as a projection from the space of all maps $T : M \to \mathbb{R}^2$ with self-overlapping boundary, to the space of locally injective maps.

Figure 17 shows the result of our algorithm with a metric obtained from running the Mixed-Integer (MI) algorithm [Bommes et al. 2009] with sharp features alignment. The parametrization of MI contained flipped triangles around two of the cones, even after running many iterations of the stiffening procedure (as described in MI). The mesh is cut into a topological disk where the image of the boundary forms a weakly self-overlapping polygon with prescribed singularities (Section 3.2).

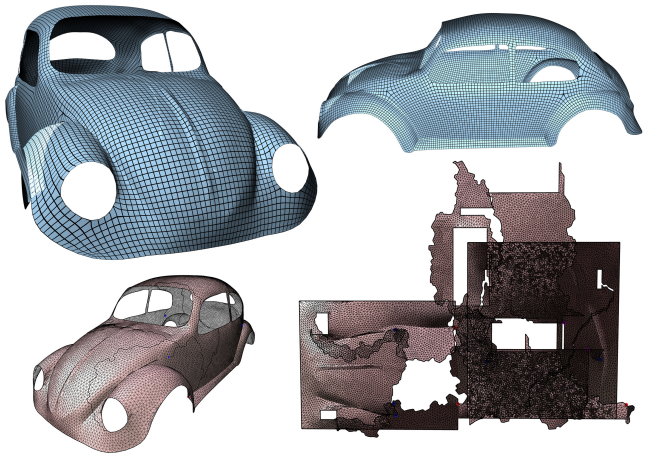

Figure 17: *The beetle model mapped using our algorithm to create a seamless global parametrization with alignment to sharp borders. 10 new vertices were added. The weakly self-overlapping polygon and the singularities were created by the Mixed-Integer algorithm.*

5.3 Extremal quasiconformal maps

We consider one example of ensuring injectivity of a map using the approach of Section 5.2 in greater detail, as in this case, using dual harmonic maps makes it possible to guarantee injectivity simultaneously with improving robustness of a mapping algorithm.

A natural extension to the space of conformal maps is the richer space of quasiconformal maps of bounded conformal distortion, i.e., orientation-preserving maps for which at any point x, $K(x) = \sigma_1/\sigma_2 < K$, where $\sigma_1 \geq \sigma_2$ are singular values of the Jacobian at x, and K is a constant bound. Unlike conformal maps, these maps allow prescribing all boundary values, for a sufficiently high allowed conformal distortion K. The *extremal map* $f^M : M \to \mathbb{R}^2$ is a bounded distortion map with minimal K, for a given boundary conditions.

Recently, [Weber et al. 2012] presented an algorithm for computing piecewise linear approximations of extremal quasiconformal maps for genus-zero surfaces with boundaries. The core idea is based on the fact that in all practically relevant cases, the extremal map is unique, and it has an explicit characterization of a Teichmüller map. A Teichmüller map f has a constant distortion $K(x) = K$ for all points; it turns out that under weak conditions on the domains, the only such maps are extremal maps.

Computing Teichmüller maps. To explain our algorithm we need to introduce more explicit notation for Teichmüller maps. It is convenient to use complex notation: let $z \in \mathbb{C}$ be the coordinate in the tangent plane of M at a point x (in the discrete case, on a triangle). The values of the map in the plane can also be viewed as complex numbers. Viewing the Jacobian of the map as a pair of complex functions $f_x(z)$, $f_y(z)$, we can introduce complex derivatives $f_z = \frac{1}{2}(f_x - \mathrm{i} f_y)$ and $f_{\bar{z}} = \frac{1}{2}(f_x + \mathrm{i} f_y)$. An important quantity

associated with a Teichmüller map is a *quadratic differential*, which is an *analytic* complex function $\phi(z)$. The map f satisfies

$$\frac{f_{\bar{z}}}{f_z} = k\frac{\bar{\phi}}{|\phi|}$$

where *little dilatation* $k = const$, in the range $[0,1)$, is related to K by $K = (1+k)/(1-k)$. Up to a uniform scale, the *Beltrami coefficient* $\mu = k\frac{\bar{\phi}}{|\phi|}$ defines the metric of f: specifically, the $\frac{1}{2}\text{Arg}\mu$ is the maximal stretch direction, and the ratio of singular values is given by $K = (1+|\mu|)/(1-|\mu|)$. (see [Weber et al. 2012] for details).

The algorithm of [Weber et al. 2012] aims to minimize a nonlinear nonconvex energy, the *least squares Beltrami energy* that measures deviation of the map f from being a Teichmüller map:

$$E_{LSB} = \int_M \left| f_{\bar{z}} - k\frac{\bar{\phi}}{|\phi|} f_z \right|^2 dA$$

The optimization is based on alternating-descent and is composed of three main steps:

1. Optimize for ϕ, assuming that f and k are fixed.
2. Solve for the single scalar k, assuming f and ϕ are fixed.
3. Compute the map f, assuming k and ϕ are fixed.

Improving robustness with dual harmonic maps. A crucial observation, also used in [Weber et al. 2012], is an interpretation of the algorithm above in terms of metric. For any map $f : M \to \mathbb{R}^2$, the metric tensor $G_f = J_f^T J_f$, where J_f is the Jacobian of f, defines the map uniquely, up to a rigid transform. As the image of the map is a subset of the plane, the metric has to be flat.

The first two steps of the algorithm compute an approximation to μ; from the theory of quasiconformal mappings, it is known that G_f can be computed from μ, up to a uniform pointwise scale s which cannot be inferred from μ. s can be computed by first computing a unit-norm metric tensor G_μ from μ, and then using the condition that $G_f = sG_\mu$ is flat. If we denote by M_f the surface M with metric G_f, this is equivalent to computing a *conformal* map from M_f to the target domain. Such a map exists only if the exact μ for the extremal map is known; in the algorithm iteration, μ is approximate. As discussed in [Weber et al. 2012], minimizing E_{LSB} for a fixed μ, is actually equivalent to computing a harmonic map in the metric G_μ which, in turn, is the same as computing a "least-squares conformal" map for fixed boundaries [Lévy et al. 2002].

While we have observed that in all but extreme distortion cases the algorithm of [Weber et al. 2012] converges to a close approximation of a Teichmüller map; unlike a smooth Teichmüller map, the resulting map does not guarantee injectivity. Combining it with a dual harmonic map allows to add an injectivity guarantee *and* improve robustness of convergence.

The simplest approach, addressing the first challenge only, is to use the method of Section 5.2 directly. Figure 18 shows the result of our algorithm when a mesh is equipped with a metric obtained from the map of [Weber et al. 2012]. Note that the initial map f, while being close to having uniform K, has flipped triangles near the four corners of the square, so f is not bijective. The dual harmonic map based on the nonflat metric defined by f remains close but is bijective (at the cost of adding 9 new vertices to M).

In extreme cases (boundaries with extreme concavities) the nonlinear optimization of [Weber et al. 2012] can converge to a local minimum which is quite far from Teichmüller maps. While using the metric of this map for a dual harmonic map ensures injectivity, it does not bring the map closer to the global minimum.

Instead, we suggest to use the dual harmonic mapping framework in a slightly different way. To this end, we use the following two simple facts: 1) A Teichmüller map is invariant to pre-composition and post-composition with a conformal map. 2) We can make the map $h_{t\to i}$ in the dual map conformal by choosing a suitable domain.

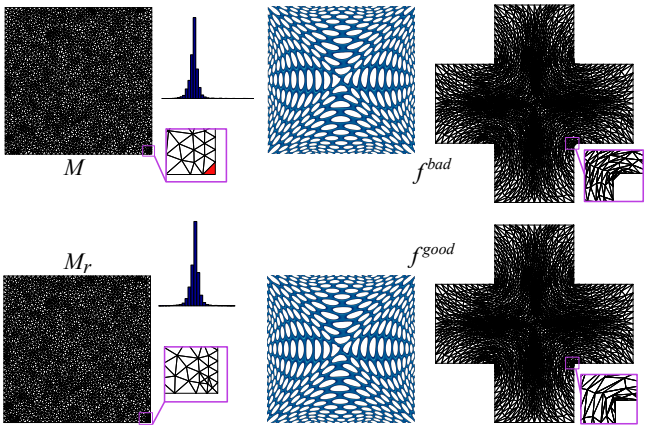

Figure 18: *Dual harmonic map with variable metric. (top row) a squared shape mesh M is mapped to a cross shape using the method of [Weber et al. 2012]. The histograms show the distribution of the conformal distortion. As evident by the histogram and the applied texture, the map is a good approximation to the extremal map. Nonetheless, the map contains flipped triangles. (bottom row) Our method uses the nearly flat metric obtained using [Weber et al. 2012], to produce a bijective map from a refined mesh M_r (with 9 new vertices) to the same target. Note that M is not 3-connected (see the red "ear" triangle at the corner), so a bijective map from M to the cross domain does not exist. Hence, any existing parametrization method that prohibit mesh refinement will fail on this particular input.*

We replace the harmonic map $h_{t\to i}$ with a conformal map; we no longer can fix the target domain, but we can add constraints ensuring that it is convex: we prescribe (positive) curvatures at the vertices of the boundary, while allowing the lengths of edges to vary freely. We use the convex optimization of [Springborn et al. 2008]; in this framework, the variables are scale factors at vertices, which directly determine edge scaling, and boundary curvatures can be easily prescribed. While [Springborn et al. 2008] is not guaranteed to produce a valid solution (the resulting edge lengths may not satisfy triangle inequality), we found that even in such cases we can still use a nonflat metric based on scale factors, as in [Ben-Chen et al. 2008] and compute a discrete bijective harmonic map closely approximating a conformal map, along with a convex domain M_c.

We modify [Weber et al. 2012]: instead of mapping directly to the target domain, we map to the convex intermediate domain M_c first. As a result, we have two maps: $h_{s\to i} : M \to M_c$, which approximates a Teichmüller map, and $h_{t\to i} : M_t \to M_c$, which approximates a conformal map. Since, the inverse of a conformal map is also conformal, the composition $g_{s\to t} = h_{t\to i}^{-1} \circ h_{s\to i}$ is also an approximation of a Teichmüller map.

The advantage of this modified algorithm is that the target domain for the Teichmüller map computation is always convex; while the LSB energy remains nonconvex, convexity of the (intermediate) target significantly improves robustness.

Figure 19 shows the result of our algorithm on extreme deformation of a planar shape, along with a comparison with other methods. The algorithm of [Weber et al. 2012] converged to a non-injective map, quite far from being extremal. The algorithm of [Lipman 2012], which can be used to produce extremal maps by a binary search for optimal k, failed to find a feasible solution with a bound on conformal distortion set to $k = 1$. Our method produces a high quality approximation to the extremal map.

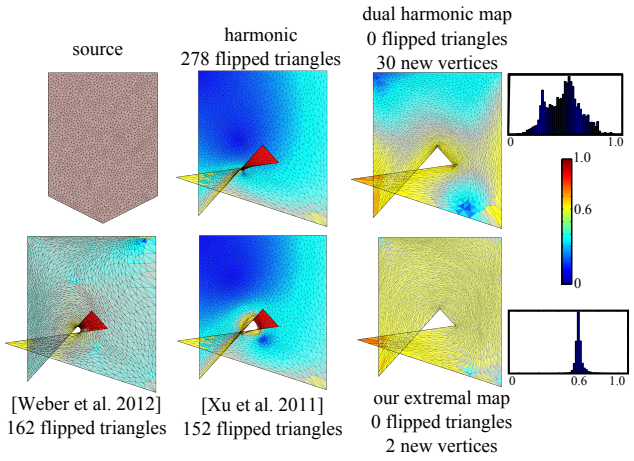

Figure 19: *Computation of extremal quasiconformal map using our method. The color visualization shows the conformal distortion, where red indicates flipped triangles. Our method (bottom right) produced a high quality approximation to the extremal map, as evident by the nearly constant conformal distortion.*

6 Evaluation and discussion

We presented an algorithm for parametrization of meshes with arbitrary fixed boundaries. The basic algorithm described in the previous sections focuses on ensuring that the parametrization is locally injective away from the boundary. To the best of our knowledge, this is the only algorithm of its kind that provides full guarantees on injectivity. Depending on the context, our algorithm can be combined with a variety of other techniques for parametrization. Our algorithm is considerably faster compared to competing methods [Lipman 2012; Weber et al. 2012]. See Table 1 for a summary of running times for the models that appear in the paper.

	statistics				run time (seconds)			
model	#vertices	#triangles	#boundary vertices	#vertices inserted	triangulation	two harmonic maps	composition + refinement	total time
Snake	55	66	42	5	0.002	0.034	0.023	0.059
Troll	1839	3400	276	0	0.006	0.084	0.108	0.198
Raptor	2556	4554	556	0	0.090	0.198	0.236	0.524
Horse	1574	2670	476	0	0.061	0.088	0.109	0.258
Hand	37234	72958	1508	0	0.085	2.189	2.971	5.245
Bull	17918	34504	1330	0	0.733	1.124	1.555	3.412
Beetle	17908	34728	1298	10	2.130	1.220	1.800	5.150
Square-Cross	2096	4048	142	9	0.005	0.097	0.129	0.231
Extremal q.c.	2079	4022	139	2	0.027	0.099	0.142	0.268
Fandisk	2026	3741	309	0	0.005	0.150	0.166	0.321
Doughtnut	6560	12720	398	0	0.062	0.302	0.473	0.837

Table 1: *Mesh statistics and running times of all steps of our algorithm on an Intel i7-3770 machine with 16GB memory. Running time of [Weber et al. 2012], which was needed to produce the "Square-Cross" and "Extremal q.c." examples are excluded.*

Our algorithm has two main limitations. First, it is limited to meshes with disk-topology. As shown in Figures 16, 17, 21, and 22, it is possible to cut a mesh with arbitrary topology to a disk-like mesh on which our algorithm can operate. However, by doing so, one has to provide additional input for the algorithm which may not always be trivially obtained. Namely, the position to which the cuts should be mapped to.

The second limitation is the fact that our algorithm may require that the input mesh will be refined. As evident from Table 1 (see column labeled "#vertices inserted"), in the majority of practical cases, our algorithm manages to avoid mesh refinement. However, mesh refinement is unavoidable in some cases (for any algorithm). It remains to understand when exactly refinement is mandatory, and how one can ensure that only the minimal possible refinement is done. Figure 20 provide a first step in answering this challenging open question. The figure illustrates a way of using our algorithm in conjunction with the method of [Lipman 2012], for the sake of producing an injective map without the need to perform refinement. We found out that the result obtained by our algorithm (without refinement) provides an excellent initialization to [Lipman 2012]. This dramatically increase the chances of [Lipman 2012] to find a feasible solution (in case it exists).

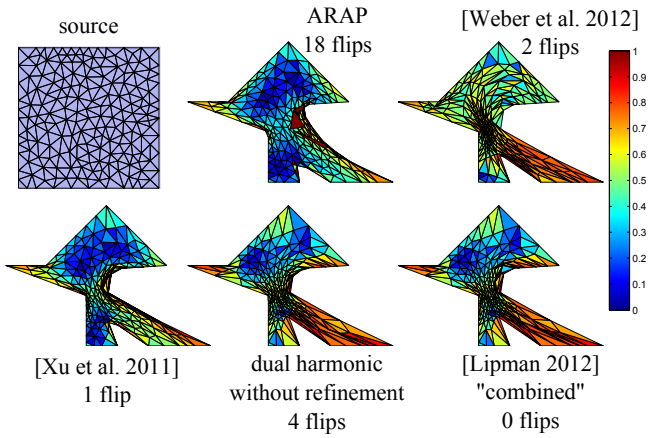

Figure 20: *Combining our method with the method of [Lipman 2012] in order to avoid mesh refinement. A mesh is deformed into an unnatural highly distorted configuration. We applied the methods of [Liu et al. 2008; Xu et al. 2011; Weber et al. 2012] which all failed to produce bijective maps. We also applied the algorithm of [Lipman 2012], by setting the upper bound on conformal distortion to the maximum ($k = 1$). The method failed to find a bijective solution within the restricted convex subspace in which it operates, for several different initializations (as explained in [Lipman 2012]). Running our method (Section 5.1) produced a bijective map by adding 9 new vertices. Finally, we used the result of our algorithm without refinement (which contained 4 flipped triangles) to initialize Lipman's algorithm. This time, the algorithm of Lipman's produced a bijective result. The color visualization shows the conformal distortion k.*

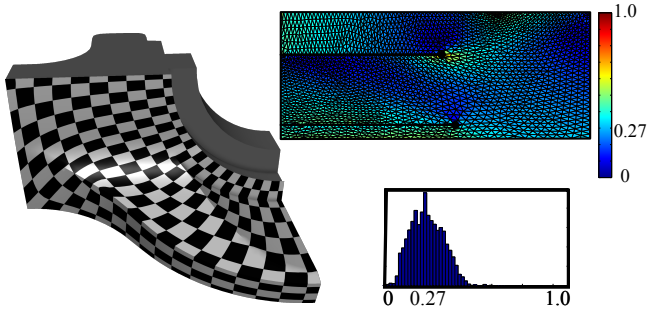

Figure 21: *Parametrization with aligned sharp features and boundary. The front of the fandisk model is cut to a topological disk and mapped using our algorithm to obtain an injective feature aligned parametrization. The image of the boundary is a weakly self-overlapping polygon with two singularities of index 1. The triangulation angles (Section 3.2) at the singularities (marked black) are precisely 2π. The color visualization and the histogram show the conformal distortion. No refinement was needed.*

Acknowledgements

This research was supported by grant number 2012264 from the United States-Israel Binational Science Foundation (BSF). We

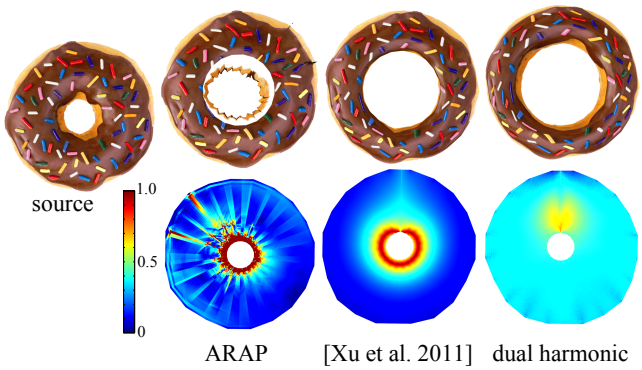

Figure 22: *Image deformation. The thickness of the doughnut is changed by scaling its inner boundary. A single straight cut is made, connecting the inner and outer boundaries. ARAP produced a map with 1,802 flipped triangles out of 12,720. [Xu et al. 2011] produced a bijective map, but the conformal distortion is high and is mostly concentrated near the inner boundary. Our dual harmonic map (Section 5.1) produced a bijective map, with overall low distortion, without any mesh refinement.*

gratefully acknowledge the support of NVIDIA Corporation with the donation of the Quadro k6000 GPU used for this research.

References

AHLFORS, L. 1966. *Lectures on quasiconformal mappings*, vol. 38. Amer. Mathematical Society.

BEN-CHEN, M., GOTSMAN, C., AND BUNIN, G. 2008. Conformal Flattening by Curvature Prescription and Metric Scaling. In *Computer Graphics Forum*, vol. 27, Blackwell Synergy, 449–458.

BOMMES, D., ZIMMER, H., AND KOBBELT, L. 2009. Mixed-integer quadrangulation. *ACM Trans. Graph. 28*, 3, 77.

BOMMES, D., CAMPEN, M., EBKE, H.-C., ALLIEZ, P., KOBBELT, L., ET AL. 2013. Integer-grid maps for reliable quad meshing. *ACM Trans. Graph. 32*, 4.

DUREN, P. L., BOLLOBAS, B., FULTON, W., KATOK, A., KIRWAN, F., AND SARNAK, P. 2004. *Harmonic mappings in the plane*, vol. 156. Cambridge University Press Cambridge.

EBERLY, D. 1998. Triangulation by ear clipping. *Geometric Tools, LLC. http://www. geometrictools. com*.

FLOATER, M. 1997. Parametrization and smooth approximation of surface triangulations* 1. *Computer Aided Geometric Design 14*, 3, 231–250.

FLOATER, M. S. 2003. Mean value coordinates. *Computer Aided Geometric Design 20*, 1, 19–27.

GARDINER, F., AND LAKIC, N. 2000. *Quasiconformal Teichmüller theory*, vol. 76 of *Mathematical Surveys and Monographs*. American Mathematical Society.

HORMANN, K., AND GREINER, G. 2000. Mips: An efficient global parametrization method. *Curve and Surface Design: Saint-Malo 99*, 153.

HORMANN, K., LÉVY, B., AND SHEFFER, A. 2007. Mesh parameterization: Theory and practice. *SIGGRAPH Course Notes*.

KÄLBERER, F., NIESER, M., AND POLTHIER, K. 2007. QuadCover: Surface Parameterization using Branched Coverings. *Computer Graphics Forum 26*, 3, 375–384.

KANAI, T., SUZUKI, H., AND KIMURA, F. 1997. 3d geometric metamorphosis based on harmonic map. *pg*, 97.

KHAREVYCH, L., SPRINGBORN, B., AND SCHRÖDER, P. 2006. Discrete conformal mappings via circle patterns. *ACM Trans. Graph. 25*, 2, 412–438.

LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2002. Least squares conformal maps for automatic texture atlas generation. In *ACM Transactions on Graphics (TOG)*, vol. 21, ACM, 362–371.

LIPMAN, Y., AND FUNKHOUSER, T. 2009. Möbius voting for surface correspondence. In *ACM Transactions on Graphics (TOG)*, vol. 28, ACM, 72.

LIPMAN, Y. 2012. Bounded distortion mapping spaces for triangular meshes. *ACM Transactions on Graphics (TOG) 31*, 4, 108.

LIU, L., ZHANG, L., XU, Y., GOTSMAN, C., AND GORTLER, S. 2008. A local/global approach to mesh parameterization. In *Computer Graphics Forum*, vol. 27, Wiley Online Library, 1495–1504.

MARX, M. 1974. Extensions of normal immersions of s 1 into r 2. *Transactions of the American Mathematical Society 187*, 309–326.

PINKALL, U., AND POLTHIER, K. 1993. Computing discrete minimal surfaces and their conjugates. *Experimental mathematics 2*, 1, 15–36.

SCHNEIDER, T., HORMANN, K., AND FLOATER, M. S. 2013. Bijective composite mean value mappings. In *Computer Graphics Forum*, vol. 32, Wiley Online Library, 137–146.

SCHÜLLER, C., KAVAN, L., PANOZZO, D., AND SORKINE-HORNUNG, O. 2013. Locally injective mappings. In *Computer Graphics Forum*, vol. 32, Wiley Online Library, 125–135.

SHEFFER, A., AND DE STURLER, E. 2001. Parameterization of Faceted Surfaces for Meshing using Angle-Based Flattening. *Engineering with Computers 17*, 3, 326–337.

SHEFFER, A., LÉVY, B., MOGILNITSKY, M., AND BOGOMYAKOV, A. 2005. ABF++: fast and robust angle based flattening. *ACM Trans. Graph. 24*, 2, 311–330.

SHEFFER, A., PRAUN, E., AND ROSE, K. 2006. Mesh parameterization methods and their applications. *Foundations and Trends® in Computer Graphics and Vision 2*, 2, 171.

SHEWCHUK, J. 1996. Robust adaptive floating-point geometric predicates. In *Proceedings of the twelfth annual symposium on Computational geometry*, ACM, 141–150.

SHEWCHUK, J. R. 2005. Triangle: A two-dimensional quality mesh generator and delaunay triangulator. *Computer Science Division, University of California at Berkeley, Berkeley, California*, 94720–1776.

SHOR, P., AND VAN WYK, C. 1992. Detecting and decomposing self-overlapping curves. *Computational Geometry 2*, 1, 31–50.

SPRINGBORN, B., SCHRÖDER, P., AND PINKALL, U. 2008. Conformal equivalence of triangle meshes.

TUTTE, W. 1963. How to draw a graph. *Proc. London Math. Soc 13*, 3, 743–768.

WEBER, O., MYLES, A., AND ZORIN, D. 2012. Computing extremal quasiconformal maps. In *Computer Graphics Forum*, vol. 31, Wiley Online Library, 1679–1689.

WEIN, R., BERBERICH, E., FOGEL, E., HALPERIN, D., HEMMER, M., SALZMAN, O., AND ZUKERMAN, B. 2013. 2D arrangements. In *CGAL User and Reference Manual*, 4.3 ed. CGAL Editorial Board.

WHITNEY, H. 1937. On regular closed curves in the plane. *Compositio Math 4*, 276–284.

XU, Y., CHEN, R., GOTSMAN, C., AND LIU, L. 2011. Embedding a triangular graph within a given boundary. *Computer Aided Geometric Design 28*, 6, 349–356.

Appendix

A Proof of proposition 1

Let M be a mesh and f be a p.w. linear map $f : M \to \mathbb{R}^2$, locally injective in the interior.

Consider a *singular* boundary vertex p of M for which f is not injective, $f(p) = v$. For any $\varepsilon > 0$, we can construct a piecewise linear path b with auxiliary vertices $(p_1, p_2, \dots p_k)$ on the 1-ring of p, connecting two boundary segments incident at p, such that the vertices $w_i^v = f(p_i)$ of the path $f(b)$ in P are on a circle S_ε of radius ε, centered at v. We call vertices v and w_i^v the group of v. Non-singular vertices contain a single group. Let M_c be the mesh obtained by removing the neighborhood cut out by b and retriangulating triangles of M that became trapezoids, and $P' = f(\partial M_c)$, a new polygon obtained by replacing a small part of the boundary of P with $f(b)$. Repeating this operation, we obtain a self-overlapping polygon P', as away from cut out points v, f is locally injective. By [Shor and Van Wyk 1992], there is a simplified mesh M_c' with no interior vertices and a p.w. linear map $f_c' : M_c' \to \mathbb{R}^2$, compatible with the restriction of the map f to M_c. This defines a triangulation T' of P' with no interior points, with consistently oriented triangles. (M_c', f_c') define a *triangulation* of P', i.e. a set of possibly overlapping triangles in the plane, sharing some edges, such that the set of non-shared edges of these triangles coincides with P'. We construct a triangulation T of P from a triangulation of P' by connecting each singular vertex v to corresponding vertices w_i^v. The only interior points of the new triangulation are points w_i^v. We transform the triangulation T to one without interior vertices by collapsing all edges (v, w_i^v).

To show that this operation does not result in any triangle inversions, we consider triangles affected by the operation. Triangles containing two or three auxiliary vertices corresponding to the same vertex of v are eliminated by the collapse operation. It remains to consider triangles with no more than one vertex from the set.

We define visibility in P as follows: if for an edge connecting two points (a,b) in P, there is a subset U of M, such that f restricted to M is bijective, and $f(U)$ contains (a,b).

Consider all pairs of vertices (a,b) of P that do not see each other, i.e. for any choice of U, (a,b) intersects the boundary of $f(U)$. For each such pair there is an $\varepsilon(a,b) > 0$, such that any two points in $U_\varepsilon(a)$ and $U_\varepsilon(b)$ do not see each other. For every singular point v, we choose ε for constructing w_i, so that it is less than $\varepsilon(v,b)$ for all b not visible from v. Then if points (a,b) do not see each other, same is true for any pair of points (w_i^a, w_i^b).

If an edge (w_i^a, w_j^b) is an edge of a triangulation of P', w_i^a and w_j^b are visible to each other; by the choice of ε above, a and b must also be visible. Thus, (a,b) we get from (w_i^a, w_j^b) after a collapse, is a valid edge for a triangulation of P. Same is true for edge of the form (w_i^a, b). It remains to show that no triangle is inverted. For a triangle not to collapse to an edge, its vertices have to belong to the expansions of different vertices of P; For any triple a,b,c of mutually visible, non-collinear vertices, there is a radius $\varepsilon(a,b,c)$ such that any triangle with vertices within ε of a, b and c has the same orientation as (a,b,c). If in addition to the constraints on ε above we add this constraint, then no triangle can be inverted. Finally, if (a,b,c) are collinear, some triangles may degenerate as a result of collapse. We eliminate these triangles by removing the longest edge, and merging the triangle with the face across that edge.

B Robust angles update

Tracking and summing angles may lead to floating points numerical accuracy problems. Instead, it is sufficient to keep track of the full rotation indices R_i (which are integers), and use robust orientation tests with exact predicates [Shewchuk 1996] to determine how to update full rotation indices when gluing two polygons.

Given three vertex positions, v_{i-1}, v_i, v_{i+1}, forming a triangle in the plane, a robust orientation test precisely decides whether the orientation of the triangle is negative, positive or zero (collapsed). By convention a positive orientation indicates that the vertices are ordered counterclockwise.

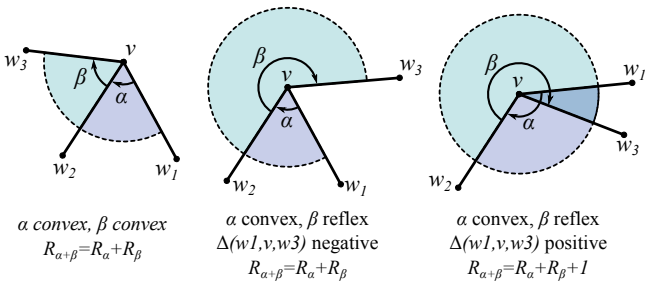

Figure 23: *Computing full rotation indices $R_{\alpha+\beta}$ for 3 different configurations.*

Suppose we attach two angles α and β, at a vertex v, associated with full rotation indices R_α and R_β. Our goal is to compute the full rotation index $R_{\alpha+\beta}$ associated with $\alpha+\beta$. Let 3 vectors involved, all starting at v, be vw_1, vw_2 and vw_3: α is the clockwise rotation of vw_1 to vw_2, and β of vw_2 to vw_3, both in the $(0, 2\pi)$ range (Figure 23). Two situations are possible: the full rotation index $R_{\alpha+\beta}$ is $R_\alpha + R_\beta + \delta$, with either $\delta = 0$ or 1. δ is 1 if and only if, one of the following conditions holds:

- both α and β are reflex;
- α is reflex, β is convex, and $\triangle(w1, v, w3)$ has positive orientation;
- β is reflex, α is convex, and $\triangle(w1, v, w3)$ has positive orientation.

α is convex, if the $\triangle(w_1, v, w_2)$ has positive orientation, reflex otherwise; similarly, β is convex, if $\triangle(w_2, v, w_3)$ has positive orientation. Thus, one can always properly update the full rotation indices by using robust orientation tests only.

C Simplification procedure for weakly self-overlapping polygons

An ear of a weakly self-overlapping polygon with prescribed triangulation angles is a triangle formed by three consecutive vertices v_{i-1}, v_i, v_{i+1} such that:

- v_i is strictly convex;
- the full rotation indices R_{i-1}, R_i, R_{i+1} are zero;
- the segments (v_{i-1}, v_i), $(v_i, v_i + 1)$, $(v_{i-1}, v_i + 1)$ do not intersect any other edge of the polygon.

Removing an ear from the polygon decrease its size while keeping it weakly self-overlapping. In contrast to simple polygons, self-overlapping polygons cannot be reduced completely by repetitively removing ears. Hence, we use a variant of [Eberly 1998] to clip as many ears as possible from the polygon, followed by running our algorithm (Section 3.2) on the reduced size polygon. The final triangulation is obtained by unifying the clipped ears with the triangulation of the reduced weakly self-overlapping polygon. Similarly to [Eberly 1998], our simplification procedure maintains two active lists. One for the polygon vertices and another for all ears. Initialization of the ear list is done in $O(n^2)$ as it involves testing whether each edge of the polygon intersect any of the other edges. Whenever an ear v_{i-1}, v_i, v_{i+1} is removed by removing v_i from the vertex list, we also check the ear status of $v_{i-2}, v_{i-1}, v_{i+1}$ and $v_{i-1}, v_{i+1}, v_{i+2}$ and update the ear list. This is done in $O(n)$ and since any polygon have up to $n-2$ ears, the entire algorithm runs in $O(n^2)$.