EUROGRAPHICS 2024 / A. Bermano and E. Kalogerakis
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 43 (2024), Number 2

Advancing Front Surface Mapping

M. Livesu

CNR IMATI, Italy

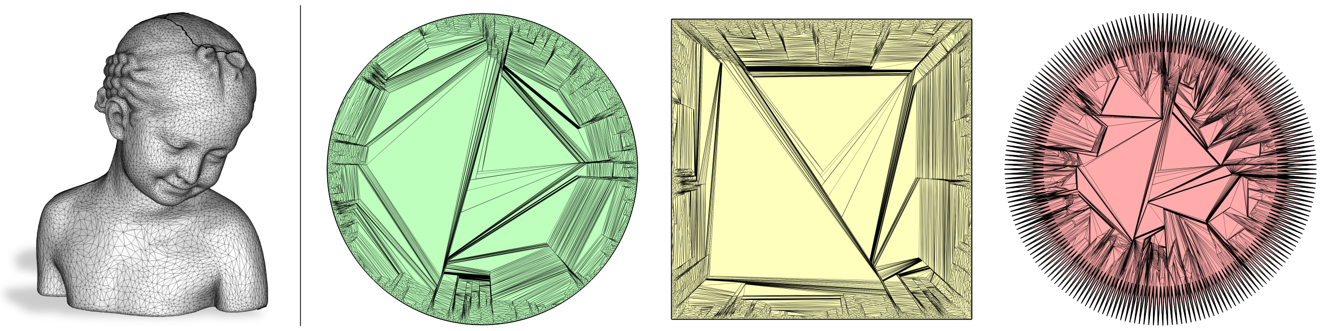

Figure 1: *Given an input mesh with the topology of a disk (left), Advancing Front Mapping embeds it into a strictly convex (green), convex (yellow) or star-shaped (red) polygon. All mappings are guaranteed to be free from degenerate or inverted elements, thus being injective.*

Abstract

We present Advancing Front Mapping (AFM), a novel algorithm for the computation of injective maps to simple planar domains. AFM is inspired by the advancing front meshing paradigm, which is here revisited to operate on two embeddings at once, becoming a tool for compatible mesh generation. AFM extends the capabilities of existing robust approaches, supporting a broader set of embeddings (star-shaped polygons) with a direct approach, without resorting to intermediate constructions. Our method only relies on two topological operators (split and flip) and on the computation of segment intersections, thus permitting to compute a valid embedding without solving any numerical problem. AFM is therefore easy to implement, debug and deploy. This article is mainly focused on the presentation of the compatible advancing front idea and on the demonstration that the algorithm provably converges to an injective map. We also complement our theoretical analysis with an extensive practical validation, executing more than one billion advancing front moves on 36K mapping tasks.

1. Introduction

Surface mapping is arguably one of the most widely studied topics in computer graphics and is at the core of many fundamental techniques in the field [FH05, HPS08, FSZ*21, NNZ21].

In this article we focus on the specific task of mapping a given triangle mesh to a convex or star-shaped domain with fixed boundary. While mappings of this kind are seldom directly useful for downstream applications due to the high geometric distortion they may contain, methods for the robust solution of this problem are internally used in many higher level pipelines, which employ these constructions to create valid initial maps that are subsequently improved based on the needs of the application at hand. Various representative examples of this strategy can be found in the literature, spanning from the robust computation of correspondences between shapes [KS04, SAPH04, SJZP19], to morphing [PSS01] and the generation of low-distortion planar maps for uv mapping, remeshing, and other applications [KSG03, LSS*98, SS15, RPPSH17, LYNF18, FW22, Liv23].

For a surface map to be practically useful, it is required that no triangle becomes degenerate or inverts its orientation. In other words, mappings must be *injective*. This is a hard property to fulfill. Most of the existing methods achieve a good practical robustness and can perform well in most of the cases, but may unexpectedly fail to produce an injective map [NCB23]. Very few methods can be regarded to be *unconditionally robust*, meaning that they provide strict theoretical guarantees of injectivity if implemented using exact numerical models such as integers or rational numbers [SJZP19, FBRCA23]. The design of robust algorithms in this class is an active field of research in geometry processing (Section 2).

In this article we introduce a novel methodology for the computation of provably injective mappings to a fixed domain called *Advancing Front Mapping*. AFM is based on the principles of advancing front mesh generation, a strategy that starts from the boundary of a target domain and proceeds inwards, inserting new elements until a tessellation of the whole domain is completed [GS94]. We adapted this idea to the context of surface map generation, where two fronts are initialized at the boundaries of two alternative domains and are simultaneously advanced while maintaining a one-to-one correspondence between them, reproducing the same topological structure in two embeddings. Since a meshing of the input domain is typically known, AFM uses the connectivity of the input mesh as a guidance, trying to install it also in the target domain. This connectivity is a resource and never a limit: in case a mapping with the given topology does not exist [ASS93], AFM is able to automatically refine the input mesh, opening the space of solutions and always providing a valid result (Figure 2).

Our method is partly inspired by prior art that uses a background mesh and topological operators to advance the front, such as [MW95]. However, operating in two domains at once introduces additional issues which cannot be handled by classical advancing front methods, because they are not able to resolve the many deadlock configurations that arise when no advancing move can be applied to both domains simultaneously. We discuss and resolve all these issues in a provably robust manner, using a completely novel methodology. Remarkably, we do this by using only a minimal set of geometric and topological constructions. In fact, AFM solely relies on two topological moves to advance the front – triangle split and edge flip – and uses simple edge intersections and edge splits to resolve any possible deadlock configuration that may arise. As a result, the AFM approach is entirely constructive and completely avoids the use of numerical optimization routines for the computation of the output map. This makes our algorithm easy to implement, debug and deploy.

AFM compares favorably with alternative robust techniques such as [FBRCA23], Tutte [Tut63] and Progressive Embeddings [SJZP19] in the sense that it enlarges the class of target domains to star-shaped polygons. Composite methods such as [WZ14] still provide more flexibility, permitting mappings to any non convex domain, even self-overlapping. However, [WZ14] internally relies on intermediate mappings to convex polygons, whereas AFM creates the wanted mapping *directly*. To our knowledge this is the only existing method that achieves this result for the class of star-shaped polygons.

To date, our contribution is mostly of theoretical interest because AFM appears to be more sensitive than prior art to floating point implementations, it may suffer from cascading issues with exact (rational) numbers, and may unnecessarily increase mesh size even in cases when refinement is not strictly needed. A detailed discussion of all these aspects is provided in Sections 4 and 5. Nevertheless, we believe that our theoretical results are sufficiently novel and powerful to be of interest for the computer graphics and computational geometry communities, and have the potential to foster more research in the field both for the surface and the volumetric setting (Section 6).

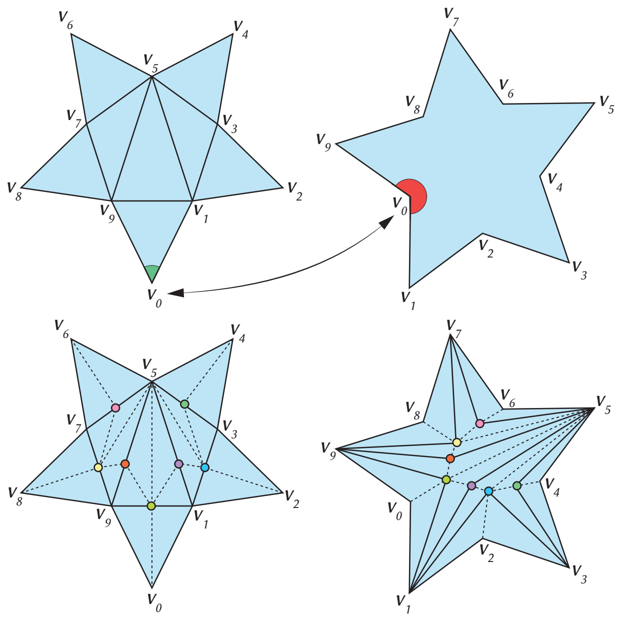

Figure 2: *Top: two identical stars up to a shift of the boundary vertices cannot be mapped to one another without introducing additional vertices, because the convexities of the left one map to the concavities of the right one. Bottom: by automatically refining the input mesh our method successfully opens the space of solutions, producing an injective map. Note: due to the presence of tiny triangles in the original output, the inner nodes in the bottom right star have been slightly relocated to make the figure easier to parse.*

2. Related Works

Our contribution is focused on a very specific instance of the general surface mapping problem. In the remainder of this section we discuss only methods that can compute mappings to planar domains with a fixed boundary and exhibit some theoretical or practical robustness. We point the reader to [HPS08] for a broader perspective on general surface mappings and their related applications.

2.1. Provably Robust Methods

Only a few methods offer strict theoretical guarantees of correctness. Known algorithms to compute a map directly only support simple domains, such as triangles [FBRCA23] or convex polygons [Tut63, SJZP19]. More general methods build on top of them, using intermediate constructions to create the embedding.

Tutte embedding and its derivatives. Solutions to the problem of mapping a surface to a strictly convex domain have been known since 1963, when Tutte showed that positioning internal vertices at the barycenter of their neighbors yields a valid embedding [Tut63]. This approach was brought to the computer graphics community and extended by Floater, who showed that the embedding is still valid when the boundary is not strictly convex and when interior vertices are positioned at a general convex combination of their neighbors [Flo97, Flo03a, Flo03b]. Various subsequent works have shown that a "Tutte-like" embedding could also be computed for alternative topological spaces such as orbifolds [AL15, AL16, AKL17] and, under precise geometric and topological restrictions, even for a restricted set of planar domains containing multiple boundaries, possibly non convex [GGT06, BCW17, KAD*20]. While some of these methods can compute injective mappings for certain star-shaped polygons, none of them covers this class of domains entirely.

Intermediate domains. A successful strategy to further enlarge the family of domains that can be considered is map composition. Methods in this category internally use the Tutte embedding, generating mappings to strictly convex domains that are then combined to obtain more general mappings to manifolds of arbitrary genus [Liv20c, LBG*08, GJGQ05, KS04, SAPH04], grid spaces for quad remeshing [Liv23], and even non-convex planar polygons that may also partially overlap [WZ14]. Being internally based on Tutte, all these methods inherit the same theoretical guarantees of correctness. Our method directly computes a map to star-shaped polygons, without resorting to intermediate constructions.

Progressive Embedding (PE). When implemented in limited precision the Tutte embedding may occasionally contain degenerate elements. Shen and colleagues proposed an alternative approach, called Progressive Embedding [SJZP19], which offers the same theoretical guarantees of correctness but is more reliable when implemented in floating point. Inspired by the concept of Progressive Meshes [Hop96], PE operates by first removing all degenerate elements from an input map through a sequence of edge collapses. Then, it reintroduces the previously removed vertices one at a time, through a reverse sequence of vertex splits. Newly inserted vertices are carefully positioned in the domain, ensuring that no incident triangle inverts its orientation or vanishes, thus preserving injectivity. Similarly to Tutte, also PE is designed to operate on convex domains. At first, it may seem that also star-shaped domains could be supported if, during the simplification phase, all internal vertices outside the kernel of the target domain are removed. However, not all mesh topologies can be successfully embedded into a star-shaped polygon. In the the worst case scenario, it is known that up to a quadratic number of Steiner points are necessary to complete this task [ASS93], making it impossible for methods that do not perform mesh refinement to succeed. We modified the authors' reference implementation to operate in this fashion and the software consistently failed to produce a valid map due to its inability to refine the input mesh, showing that this theoretical limitation arises in all practical cases.

Foliations. Campen and colleagues introduced the concept of simplicial foliations [CSZ16], a method to map both 2D and 3D simplicial meshes to simple base domains. This method is provably robust but it only supports mappings to squares and circles, also not permitting to setup explicit per vertex boundary conditions. Foliations yield maps that are non linear inside each triangle, which can be transformed into piece-wise linear maps only at the expense of massive mesh refinement, increasing mesh size even by orders of magnitude. A recent article, mostly focused on volumes, significantly improved the original algorithm [HC23], alleviating some of these limitations.

Compatible Triangulations. Methods that fill two empty domains with a compatible triangulation are similar in spirit to our method. Various algorithms have been proposed over the years [ASS93, GW97, SG04, Liv20a, Liv20b] but they can only operate in a 2D-to-2D fashion. The case in which the source mesh is a surface embedded in a higher dimensional space is not supported. The concept of compatible triangulations was recently lifted to the surface case [Tak22], but this latter method requires to be initialized with a previously existing map, hence it does not solve the problem we are interested in.

2.2. Numerical Methods

A large body of literature focuses on algorithms that compute a planar embedding by solving a numerical problem. These methods typically minimize some distortion energy, producing high quality surface maps that are already in floating point and are best suited for downstream applications. Modern algorithms are extremely robust and can produce a valid map in the vast majority of the cases [DKZ*22, DAZ*20, GKK*21, POK23, WGS23, GSC21, SLS22], also applying on-demand refinement to open the space of solutions if needed [JHT14]. Despite superior in terms of map quality and practical usefulness, methods in this class do not provide theoretical guarantees of correctness and, when tested on large datasets of models, tend to fail in a higher number of cases [NCB23]. A major shortcoming stems from the fact that explicitly encoding the injectivity constraint yields a non-linear non-convex optimization problem that cannot be solved directly [WZ14]. Various ideas have been explored to relax this constraint and make the problem numerically tractable. We point the reader to [FSZ*21, NNZ21] for recent surveys on the topic.

Robust Initialization. If initialized with a valid embedding, numerical methods can iteratively reduce geometric distortion while provably preserving the injectivity of the given map [SS15, CBSS17, RPPSH17, LYNF18, FW22, JSP17]. Our advancing front algorithm, as well as any other robust method in Section 2.1, can be used to provide a valid initial solution. Note that these pipelines are implemented in floating point, therefore the injectivity of the initial solution and of the final result are only guaranteed up to approximation errors introduced by the numerical system.

3. Advancing Front Mapping (AFM)

AFM takes in input a triangle mesh with disk-like topology, M_1, and an injective mapping of its boundary vertices onto the boundary of a target domain Ω. Such domain is required to be convex or star-shaped, meaning that it contains a non empty kernel from which all its boundary vertices are directly visible. The algorithm outputs

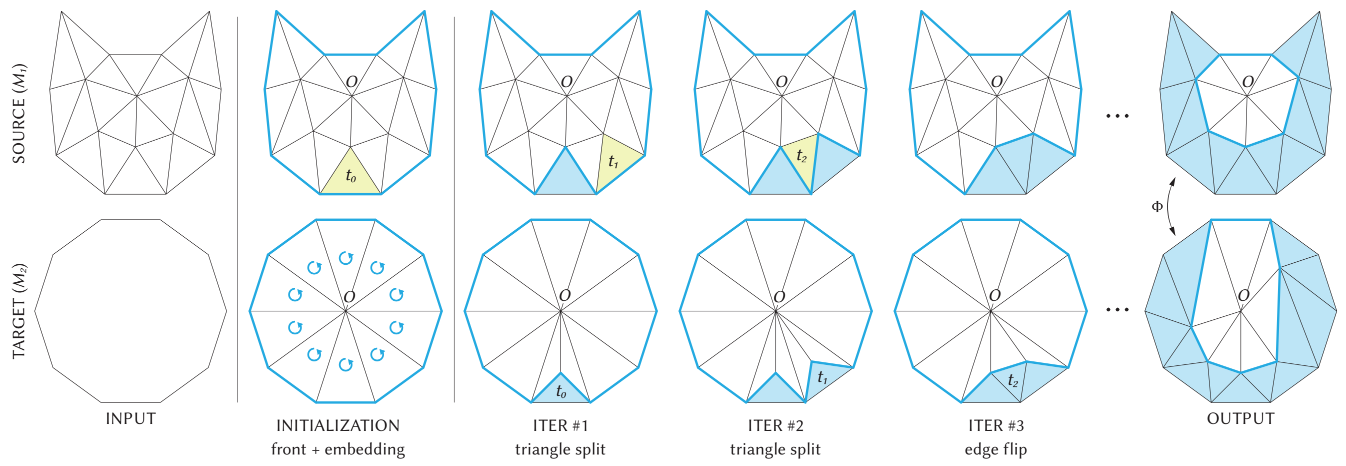

Figure 3: *Pipeline of AFM: we start from a source triangle mesh and a target convex or star-shaped domain (left). In the initialization step (second column) boundary edges are marked as front (thick blue lines) and a starting embedding is created by forming a triangle between each front edge and the front origin (O), selected as an inner mesh vertex. Due to convexity, all triangles share a globally coherent orientation. Then, AFM iteratively reproduces the connectivity of the source mesh in the target domain, advancing both fronts while maintaining a one-to-one correspondence between them. Source triangles having only one edge on the front (t_0 and t_1) are inserted by splitting the triangle formed by the image of such edge and O (iterations #1 and #2). Source triangles having two edges on the front (t_2) are inserted by flipping the edge connecting O with the front vertex shared by the images of such edges (iteration #3). The algorithm stops when all source triangles have been inserted in the target domain, yielding a one-to-one mapping Φ between the two meshes.*

two meshes: a possibly refined version of M_1 and a mesh M_2 that triangulates Ω and has the same connectivity of M_1.

Since the two meshes contain the same number of vertices, connected to form the same triangles, there exists a natural piece-wise linear mapping between them

$$\Phi : M_1 \leftrightarrow M_2.$$

Function Φ is constructed through a direct per vertex correspondence and is linearly extended inside mesh triangles via barycentric interpolation. AFM is guaranteed to always produce a piece-wise linear injective map, meaning that neither M_1 nor M_2 contain zero-area or inverted elements and all triangles share a globally coherent orientation, hence Φ is one-to-one [Lip14].

At a high level the algorithm is rather simple: after having created a coarse mesh M_2 that covers the target domain Ω, the boundary edges of M_1, M_2 are set to form two initial fronts with simple topology (Figure 3, Initialization). These two fronts are then coherently pushed inwards, maintaining a one-to-one correspondence throughout the whole execution. Each advancing front move amounts to conquering an unvisited triangle in M_1, inserting an equivalent triangle in M_2. Advancing moves in the target domain are always executed by either splitting an existing triangle in M_2 or by flipping one of its edges (Figure 3, Iterations). Both moves are guaranteed to not create degenerate or inverted elements if simple local geometric requirements are fulfilled, thus ensuring the injectivity of the map (Appendix B). In case of geometric deadlock configurations that prevent the front to advance any further, vertex relocation or finite local mesh refinement (applied to both meshes) is used to resolve the lock and proceed with the computation, ensuring convergence (Appendix A). Refinement is used parsimoniously and only marginally impacts the input mesh size (Section 5). The algorithm stops when all triangles of M_1 have been successfully inserted into mesh M_2, yielding two meshes with identical connectivity and different embedding. An algorithmic description of AFM is reported in Algorithm 1. In the remainder of this section we discuss details of each step, including initialization, front advancement, and handling of the deadlock configurations that may arise.

3.1. Initialization

In this phase the input mesh M_1 is first refined to avoid predictable degenerate configurations. Then, mesh M_2 is initialized as a polar mesh that entirely covers the target domain Ω. After these operations have been executed, the algorithm is ready for the iterative part and the boundaries of meshes M_1 and M_2 form the two fronts to be advanced.

3.1.1. Refinement

From a topological perspective, advancing a front with simple topology starting from the boundary and proceeding inwards until the whole mesh is conquered corresponds to computing a shelling sequence of M_1. For topological disks, a shelling sequence is always guaranteed to exist if the mesh graph is 2-connected [CSZ16](§4.4). This property can be easily enforced by splitting all internal edges that connect pairs of boundary vertices in the input mesh. This edge splitting strategy also provides additional benefits to our algorithm, because:

- it ensures that the mesh has at least one internal vertex, which is a necessary condition for the initialization of mesh M_2;

- it ensures that no triangle has all its three vertices on the boundary, which may become flipped or degenerate when mapping to concave or non strictly convex domains such as a square (due to concavity or co-linearity of the boundary edges).

3.1.2. Polar Mesh

For the final mapping to be injective, AFM needs to insert the connectivity of M_1 onto the mesh M_2 by always operating onto a *valid* mesh. In our setting a mesh is valid if all triangles share a globally coherent orientation and are not degenerate. Constructing such a mesh is trivial if Ω is convex or star-shaped. In fact, in case of convexity M_2 can be initialized by taking any point strictly inside Ω and forming a triangle between such point and each boundary edge. Similarly, if the domain is star-shaped a valid mesh M_2 can be created with the same procedure, with the additional restriction that the inner point must be located inside the kernel of the domain, that is, any point inside Ω from which all boundary vertices of M_2 are directly visible [SBS22]. A visual example of the so generated mesh can be seen in Figure 3, at the bottom of the second column. Since this mesh contains only one internal vertex that is incident to all boundary triangles, we call such vertex the *front origin*, and denote it with the letter O.

We recall that M_1 and M_2 will eventually contain the same connectivity, therefore an interior vertex in mesh M_1 must also be selected and put in correspondence with O. Thanks to the preliminary refinement step, an interior vertex in M_1 is guaranteed to exist. From a theoretical perspective any interior vertex is equally good. In practice, using a point that is furthest from the boundary promotes a uniform growth of the front, providing higher numerical stability in floating point. We select such vertex by simply running the Dijkstra shortest path algorithm with multiple sources, selecting all boundary vertices as source nodes and flooding the whole mesh until the furthest vertex is found.

ALGORITHM 1: Advancing Front Mapping (AFM)

inputs : (1) a triangle mesh M_1 with the topology of a disk; (2) a one-to-one correspondence between the boundary vertices of M_1 and the boundary of a convex or star-shaped domain Ω.

outputs: (1) a possibly refined version of $M1$; (2) a new mesh M_2 which has the same connectivity of M_1 and that realizes an injective mapping of M_1 onto the target domain Ω.

```
1  refine $M_1$;                                              (Section 3.1.1)
2  mark all triangles in $M_1$ as unvisited;
3  select the front origin $O$ and initialize $M_2$;           (Section 3.1.2)
4  initialize front as the set of boundary edges in $M_1$;
5  push all front edges into a queue $Q$;
6  while $Q$ is not empty do
7      pop edge $e$ from $Q$;
8      select the triangle $t \in M_1$ inside the front and incident to $e$;
9      if $t$ is already marked as visited then
10         continue;
11     end
12     let $E$ be the front edges incident to $t$
13     if $|E| = 1$ then
14         let $v$ be the vertex of $t$ opposite to $e$;
15         if $v$ is on the front or is the front origin $O$ then
16             continue;
17         end
18         advance front in $M_2$ by triangle split;           (Section 3.2.1)
19     end
20     else if $|E| = 2$ then
21         if edges in $E$ form a concave angle then
22             locally convexify front;                        (Section 3.3.1)
23         end
24         let $t_E$ be the triangle denoted by edges in $E$;
25         if $t_E$ contains the front origin $O$ then
26             if the the front is not locally a triangle strip then
27                 continue;
28             end
29             locally concavify front;                        (Section 3.3.2)
30         end
31         advance front in $M_2$ by edge flip;                (Section 3.2.2)
32     end
33     mark $t$ as visited;
34     push new front edges in $Q$;
35 end
```

3.2. Advancing Moves

This is the core of the algorithm. All front edges of the input mesh M_1 are inserted into a queue and then processed one at a time until the queue becomes empty. Given a front edge e, AFM aims to insert into the target domain the image of the triangle t that is incident to e from the inner side of the front. Two cases are possible: t may have either one or two of its edges exposed on the front (including e). Note that the case where all the three edges of t are on the front cannot happen, because the front has simple topology and there is at least one vertex inside the front (the origin O).

3.2.1. One front edge

Let e be the only edge of t exposed on the front and v the vertex of t opposite to it. If vertex v is inside the front and has not been inserted into mesh M_2 yet, we insert the image of t into mesh M_2 by splitting the triangle formed by the image of e and the front origin, O. We do this by using as a split point the image of vertex v, $\Phi(v)$ (Figure 4). This operation produces three sub-triangles, one of which is formed by $\Phi(e)$ and $\Phi(v)$, hence is the image of triangle t. The front in both meshes is then advanced, moving from e to the other two edges of triangle t, which is now located outside of the current front (see iterations #1 and #2 in Figure 3). For the newly generated triangles to preserve the orientation of their father (hence local injectivity) it is sufficient to position point $\Phi(v)$ strictly inside the triangle formed by $\Phi(e)$ and O. In our implementation we position $\Phi(v)$ at

$$\Phi(v) = \frac{99\,\Phi(v_0) + 99\,\Phi(v_1) + 2O}{200},$$

where v_0 and v_1 are the vertices of edge e. This choice avoids the front to rapidly approach the origin O, leaving more room for the insertion of the subsequent mesh elements.
Triangles having one edge and all three vertices exposed on the front are initially skipped because inserting them would break the

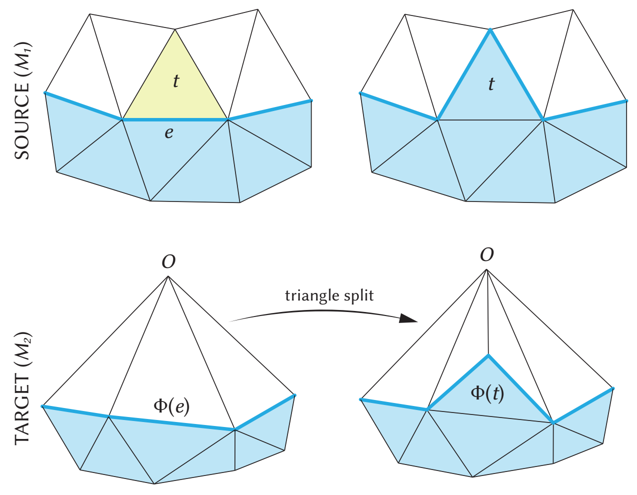

Figure 4: *Advancing the front to reproduce a triangle t having a single front edge e amounts to: (i) locating the triangle formed by the image of e, $\Phi(e)$, and the front origin O; (ii) splitting it into three sub-triangles. The triangle formed by the split point and by $\Phi(e)$ is $\Phi(t)$.*

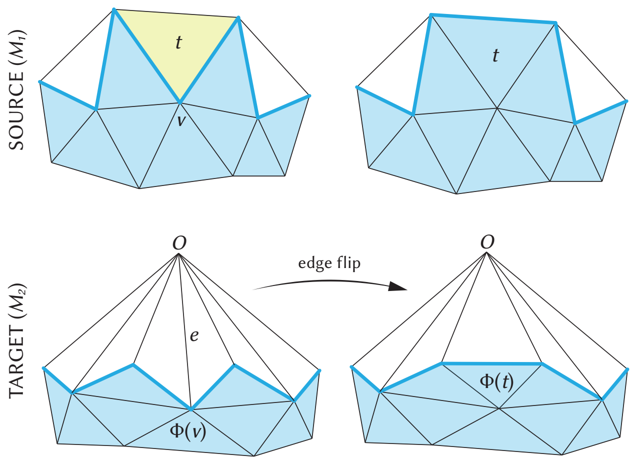

Figure 5: *Advancing the front to reproduce a triangle t having two front edges meeting at a shared vertex v amounts to: (i) locating the edge e connecting the image of v, $\Phi(v)$, with the front origin O; (ii) flipping e. This operation can be performed only if the quad surrounding e (i.e., its* link*) is strictly convex.*

topology of the front, making it a non simple polygon (i.e., the triangle insertion sequence would not be a valid shelling sequence). Note that this case does not need to be handled: the front will advance further through other moves and, at some point, another edge of the same triangle will show up in the front. At that point, the triangle will be inserted with the edge flip move described in the next paragraph.

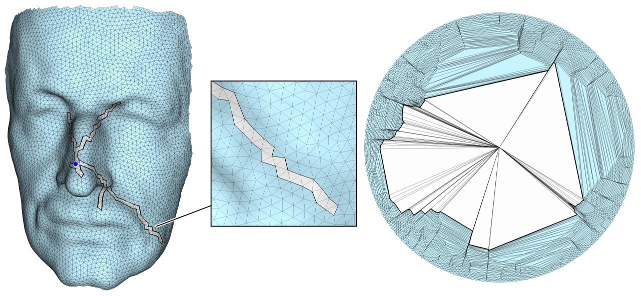

Figure 6: *Naively applying triangle splits and edge flips to advance the front does not always lead to convergence. In the typical case, the front conquers all the mesh vertices, leaving a network of triangle strips (closeup) that cannot be inserted in the target domain because their associated edge flip operations are concave, hence they would introduce flipped elements. The local strategies described in Section 3.3 ensure that any edge flip can be performed without breaking the injectivity of the map, thus always ensuring both validity and convergence.*

3.2.2. Two front edges

Let e_1, e_2 be the two edges of t exposed on the front and let v be the front vertex in between them. In the target domain edges $\Phi(e_1)$ and $\Phi(e_2)$ form two triangles with the front origin O and are adjacent to one another along the edge connecting $\Phi(v)$ with O. The insertion of triangle t inside mesh M_2 can therefore be performed by simply flipping such edge (Figure 5). The front in both meshes is then advanced, moving from e_1, e_2 to the edge of t opposite to them (see iteration #3 in Figure 3). Note that, differently from the triangle split, the edge flip operation cannot always be performed. Deadlock cases that arise when no further flip move is possible are discussed in the next section.

3.3. Deadlocks

An edge can be flipped without introducing inverted elements only if the four-sided polygon formed by the two triangles incident to it is strictly convex. This may create deadlock configurations where the front can no longer be advanced, because all missing triangles have their three vertices exposed on the front and no edge flip move is possible. Situations of this kind typically arise towards the end of the execution, when all input mesh vertices have been inserted in the target domain and a sequence of edge flips is required to *close* the front (Figure 6).

Concavities that prevent the execution of a flip move may arise at both endpoints of the edge to be flipped. In AFM flipped edges always connect a front vertex with the origin O. In the following two subsections we describe the two procedures that we use to unlock illegal edge flips in these two cases.

3.3.1. Convexification

When the front is locally concave at a front vertex v, we unlock the flip of the edge connecting v and O via vertex relocation, pushing one of the front vertices adjacent to v towards O, making the front

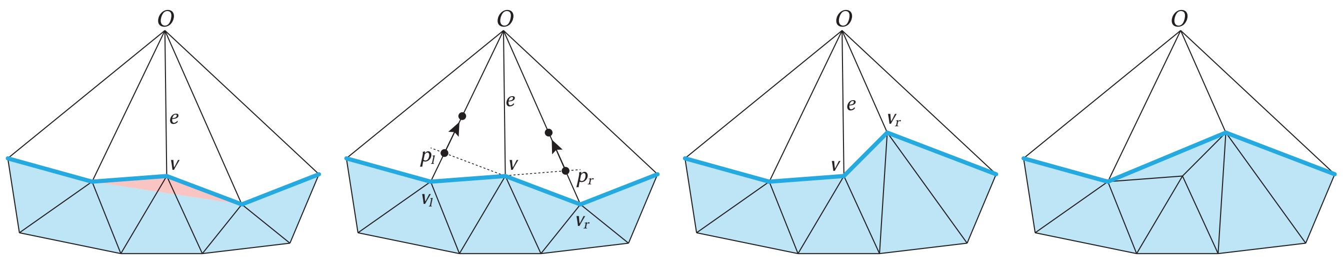

Figure 7: *Left: flipping edge e introduces an inverted element (red) because the front is locally concave at vertex v. Middle left: we compute points p_l, p_r by intersecting the prolongation of edges v, v_l and v, v_r with edges v_l, O and v_r, O. Positioning either v_l or v_r at such intersection points makes the front locally flat. Pushing any of them a bit further towards the origin (black arrows) makes the front locally convex at v. Middle right: since only one vertex motion is necessary, we break ties by moving the vertex that remains furthest from O (in this case v_r), leaving more room for the next advancing moves. For the same reason, in our implementation we only minimally lift vertex v_r, computing its final position as $0.99p_r + 0.01O$. Right: after vertex relocation the front can be advanced by flipping edge e.*

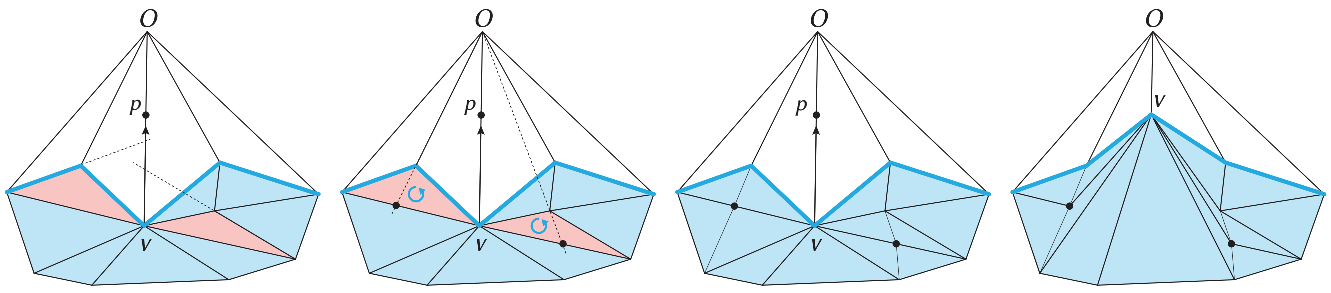

Figure 8: *Left: pushing vertex v towards point p flips the orientation of some triangles (in red) because, for each of them, v and p lie at opposite sides of the supporting line of the edges opposite to v (dashed lines). Middle left: split points to fix this issue can be computed by intersecting, for each red triangle, the edge whose positive half-space contains the front origin O with the line passing through O and the triangle vertex opposite to such edge. Middle right: splitting edges at the so computed intersections yields a new mesh where all triangles remain valid when pushing v towards O (right).*

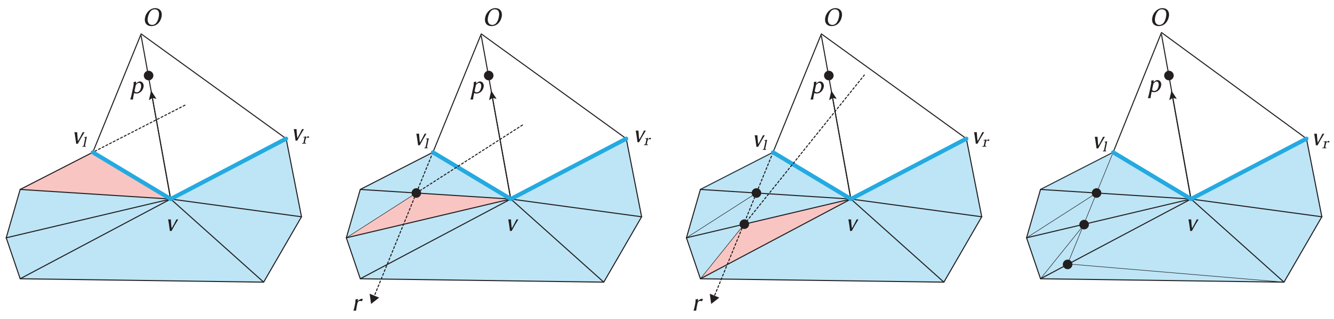

Figure 9: *Left: moving vertex v to p would flip the orientation of the red triangle. As shown in the next three columns the mesh refinement procedure described in Figure 8 may in turn create new triangles that would flip their orientation, thus triggering cascaded refinement. All the newly inserted split points (black dots) are positioned along the ray r that emits from O. Since the number of edges incident to v is finite and its valence never increases, r can only intersect a finite number of edges. Therefore the cascaded refinement is guaranteed to terminate.*

locally convex at v (Figure 7). Once the selected point has been repositioned to the desired location, the edge can be flipped and the front advanced as described in Section 3.2.2.

The convexification strategy described in Figure 7 is not completely safe, because it only ensures that the two triangles generated by the edge flip are not inverted, completely overlooking the effect that this operation has on all the other triangles that are incident to a relocated vertex v. As shown in the left part of Figure 8, previously existing triangles (in red) may flip their orientation due to vertex relocation. We resolve this issue by splitting such triangles

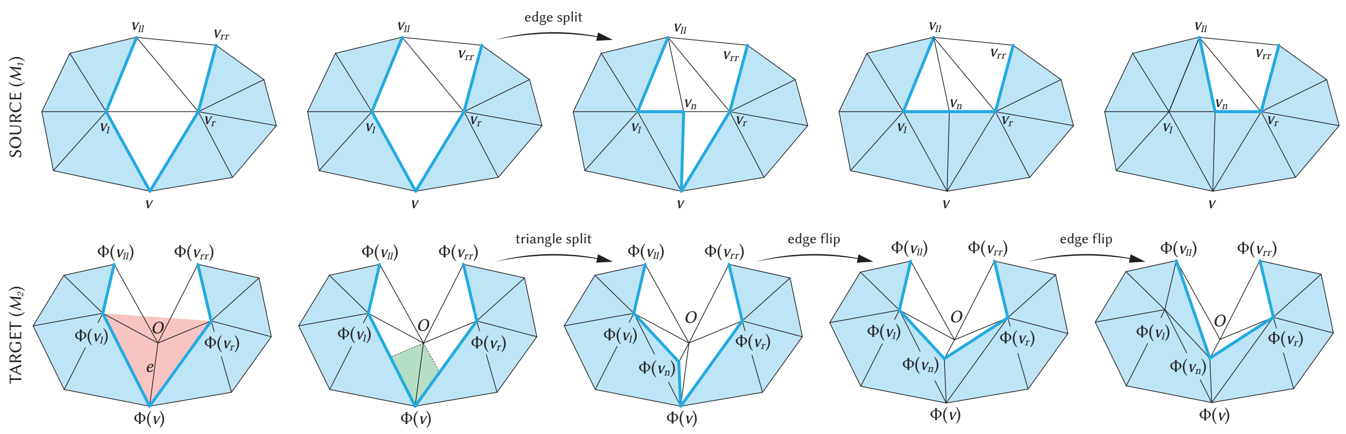

Figure 10: *Left: flipping edge e introduces an inverted element (red) because the triangle $\Phi(v_l, v, v_r)$ contains the front origin O. We resolve this issue by inserting a new vertex v_n in both meshes, locally editing the front to avoid advancing it beyond the origin O. In the input domain vertex v_n splits the edge v_l, v_r at its midpoint. In the target domain, its image $\Phi(v_n)$ is positioned so as to create valid edge flip moves to insert triangles $\Phi(v_n, v, v_r)$ and $\Phi(v_l, v_n, v_{ll})$. The feasible region is highlighted in the second column (green area) and is computed by intersecting the prolongations of edges $\Phi(v_r), O$ and $\Phi(v_{ll}), O$ with the front. The advancing moves are shown in the last three columns. Note that in case $\Phi(v_n)$ is not inserted in the green region, one or both the edge flip moves in the last two columns would in turn suffer from the same issue shown in the first column, thus triggering infinite refinement.*

as described in the same figure, ensuring that, for each triangle t incident to v, the positive half-space of the edge e opposite to v in t contains the front origin O. Since vertices are always relocated at a point along an edge incident to O, this ensures that any possible relocation does not introduce inverted elements in M_2.

This refinement may in turn generate another flipped triangle that is adjacent to the same split edge and the process can potentially iterate, resulting in cascaded refinement. As shown in Figure 9 the entire process is convergent, because all offending triangles are fixed by splitting edges incident to the front vertex v at the intersection points with a fixed ray r. Considering that the valence of vertex v is finite and does not increase during refinement, the number of edges intersected by r is fixed as well, ensuring termination in a finite number of steps.

Note that, differently from the triangle splits in Section 3.2.1, all refined triangles are images of triangles existing in the input mesh. Therefore, these splits are also applied to triangles in M_1, maintaining a full topological compatibility with mesh M_2.

To avoid excessive mesh growth, convexification should be used parsimoniously. AFM is designed to promote the fulfillment of the geometric criteria that ensure the applicability of flip operators and only marginally uses convexification. In our large scale benchmark we performed almost half a billion edge flips. Only in the 15.7% of the cases convexification was necessary and only in the 3.7% of these cases local refinement was used to unlock a flip (Section 5).

3.3.2. Concavification

When the concavity that prevents a flip is located at the front origin O, we unlock the flip operation by adding one degree of freedom along the front, splitting the triangle that would flip its orientation (left of Figure 10, in red) into two sub-triangles that do not contain O. We call this procedure *concavification*, because a portion of the front that is already convex becomes locally concave to accommodate the flip move that we need to perform. Particular care must be taken to ensure that the process terminates. The key to successfully advance the front and ensure convergence consists in positioning the new vertex not only in function of the current advancing move, but also considering the next two, finding a location that ensures that a full sequence of three advancing moves can be performed without any further convexification or concavifications (one split and two flips shown in last three columns of Figure 10). Concavification can only be applied if the two triangles to be inserted form a triangle strip, that is, if all their four vertices are already on the front and no alternative move is locally possible. As detailed in Appendix A this is enough to ensure algorithmic convergence. For this reason, concavification is even more rare to occur than convexification and has negligible impact on the output mesh size. In our large scale benchmark this procedure was applied in only the 0.29% of the cases.

A tempting (apparently simpler) alternative to concavification consists in relocating vertex O outside of the red triangle in Figure 10. While in some cases this would work without requiring extra refinement, it should be noted that not only O but also other portions of the front may sneak inside that triangle, possibly requiring a large number of relocations, each of which may trigger its own refinement (Figure 8). Considering the marginal number of its occurrences, we opted for concavification because it is compact to code and it always has the same complexity, avoiding the necessity to handle challenging corner cases in the code.

4. Implementation Details

The constructions described in the previous two sections theoretically guarantee both the existence and the injectivity of any map-

Domain	#Models	#Conv.	#Timeout	#Adv. Moves	#Triangle Splits	#Edge Flips	#Convex.	#Concav.	Mesh growth	#Flips (rational)	#Flips (double)	Average runtime
Circle	11942	11679	263 96.5% done	314.6M	152.7M 48.5%	161.9M 51.5%	15.1M 9.3%	918.6K 0.56%	avg 4.91% max 42.6%	0	885.5K 3166 models 27.1%	0.77s
Square	11942	11645	297 96.2% done	312.3M	151.6M 48.5%	160.7M 51.5%	15.7M 9.8%	910.1K 0.57%	avg 5.76% max 42.9%	0	866.7K 3231 models 27.7%	0.76s
Star	11942	11567	375 86.1% done	308.1M	149.5M 48.5%	158.6M 51.5%	43.2M 27.2%	901.9K 5.69%	avg 18.1% max 56.1%	0	791.5K 3780 models 32.7%	0.83s
TOTAL	35826	34892	935 92.2% done	934.9M	453.8M 48.5%	481.1M 51.5%	74.0M 15.4%	2.7M 5.68%	avg 9.5% max 56.1%	0	2.5M 10177 models 29.1%	0.78s

Table 1: *Summary of the results obtained in our large scale benchmark. In the second section of the table we report: the total number of input meshes; the number of maps successfully completed and the number of timeouts (i.e., the amount of runs that were stopped because an advancing move took more than* 2*s). For timeouts, we also report the average percentage of triangles already inserted in the target domain when the process was stopped. In the second section of the table we report: the total number of advancing moves; the number and percentage of triangle flips and edge flips, and the number and percentage of convexifications and concavifications that were executed to unlock an illegal edge flip move. We then report the average and maximum growth rate of the meshes, measured as* $(|T|_{out} - |T|_{in})/|T|_{in}$*, and the number of flipped elements in the map, both with rationals and floating point coordinates. Flips in floating point coordinates were computed by forcing, at the end of the execution, a naive snap rounding to double for all mesh vertices. The total number of models containing at least one inverted element and the percentage w.r.t. the total number of converged models are also reported. Finally, in the rightmost column we report the average running time. Statistics on moves, growth, flips and run times do not consider timeouts. Including timeout experiments, more than one billion advancing moves were successfully performed by AFM.*

ping to a convex or star-shaped domain. The proposed approach entirely focuses on mesh *validity* and does not make any attempt to generate an embedding with good geometric quality. As a result, output meshes can be expected to contain badly shaped and nearly degenerate triangles which may be problematic to handle in a real software implementation. In this section we provide low level practical details about AFM, explaining how to transfer its correctness guarantees theory to software.

Numerical models. Many nearly degenerate triangles will be created by AFM and, based on properties computed on such triangles, many algorithmic choices will be made during the execution. Expecting a correct algorithmic flow using limited precision floating points for calculation is practically impossible. Since all points inserted in the domains are computed by means of barycentric interpolation or by the computation of intersections, all point coordinates will be rational numbers. We therefore implemented AFM using exact constructions to express both vertex coordinates and intermediate quantities, ensuring that no approximations will occur during the map computation. Specifically, we used the GMP [Gra10] rational numbers enhanced with the lazy evaluation scheme of CGAL [PF11].

Snap rounding. Despite enhanced by a lazy kernel, rational numbers are known to introduce dramatic slowdowns in the computation [CLSA20, CPAL22]. Even worse, due to our progressive approach vertex insertion is *incremental*, thus introducing a cascading effect that may produce rational numbers with huge complexity. To mitigate the cascading problem and reduce the computational overhead introduced by our numerical model we always try to convert the coordinates of newly inserted vertices into floats. This is done by attempting a naive snap rounding, using `CGAL::to_double()`, and then checking whether the rounded coordinates have inverted the orientation of any triangle incident to such vertex. If no flip is found, the snap rounding is accepted. Otherwise, the coordinates for that specific vertex are kept rational. Based on our experience, this simple strategy reduces running times by almost one order of magnitude.

Algorithmic choices. AFM has been explicitly designed to operate with rationals. This is perhaps visible in the geometric constructions that we used for convexification (Section 3.3.1) and concavification (Section 3.3.2), which entirely avoid the use of normalized vectors (which involve the computation of squared roots) and trigonometric functions. Line searches for vertex repositioning were also avoided because when implemented in infinite precision may produce (almost) infinite loops even in cases when convergence is guaranteed from a theoretical perspective. Alternative geometric constructions may have possibly produced better shaped triangles or smaller vertex displacements, but our approach entirely based on the computation of segment intersections is fully compatible with our numerical model and is *closed form*, in the sense that it is based on finite quantities that can be readily computed, without requiring iterations or numerical algorithms for their estimation.

5. Results

We implemented AFM in C++, using Cinolib [Liv19] for geometry processing and `CGAL::Lazy_exact_nt<CGAL::Gmpq>` for exact computations. Our reference code is directly contained inside Cinolib (example 47) and runs as a single threaded application, supporting mappings to circles, squares and star-like polygon domains as the ones shown in Figures 1 and 12. Code improvements such as parallelization and warm starting are possible and would likely greatly boost our performances and practical usefulness, but are not currently implemented. Considerations about these future extensions can be found in Section 6.

Dataset. To validate our approach we conducted a large-scale experimentation involving roughly 12K meshes collected from the 2D Structure Dataset [CLH*16] and from data released by the authors of [LYNF18, CFH*18]. We composed this testing dataset so as to be heterogeneous in multiple ways, exposing AFM to a wide variety of mesh sizes (from 190 to 230K triangles), shapes (humans, animals, abstract objects, tools, furnitures, silhouettes), surface types (smooth, noisy) and local mesh structures (regular, irregular, high vertex valence).

Setup. Overall, we run 36K experiments, launching AFM three times for each input model, producing mappings to three alternative target domains, both convex and star-shaped. All tests have been executed on commodity hardware (a Mac Book M1 Pro with 32GB of RAM). Considering the limited hardware capabilities at our disposal, we decided to set a strict time limit for each experiment, aborting the map generation if a single advancing move took more than 2 seconds and also setting 24 hours as a maximum time budget for the whole experiment. Comprehensive statistical information for all our tests are reported in Table 1.

Baseline. Our baseline for comparative analysis is composed by Tutte [Tut63] and Progressive Embedding (PE) [SJZP19]. We did not consider [FBRCA23] because it only supports triangular domains. For Tutte, we considered the implementation available in Cinolib [Liv19]. For PE, we used the reference implementation released by the authors. Both implementations use floating point numbers and PE also exploits parallelization to speed up computation. The reference code for PE assumes to receive in input a previously existing mapping, which is only locally modified to remove inverted elements (if any). The authors released two tools to warm start PE: one based on Tutte and one based on randomization. Since in most of the cases Tutte already computes an injective map, bootstrapping PE with Tutte would trigger the execution of PE on a tiny amount of cases (5 out of 12K tests). We therefore initialized an invalid embedding using their `random_init_bin` tool and then processed the so generated file with their tool `untangle_bin`, using option `-e 1`. This configuration ensures that PE is indeed executed on every single mapping task and is asked to reproduce the entire map with its edge collapse/vertex split strategy. We emphasize that this choice of the baseline is not representative of the state of the art. The best existing compromise between performances and floating-point robustness is obtained with PE bootstrapped with Tutte. However, for the sake of a more informative comparison between basic mapping strategies, we decided to not rely on warm starting (which is also compatible with AFM, as discussed in Section 6). For the same reason, we restricted our comparative analysis to strictly convex domains only. Squared and star-shaped domains are not considered, even though a basic pre-processing refinement such as the one described in Section 3.1.1 would have allowed us to extend our comparative analysis at least to convex domains containing co-linear boundary vertices.

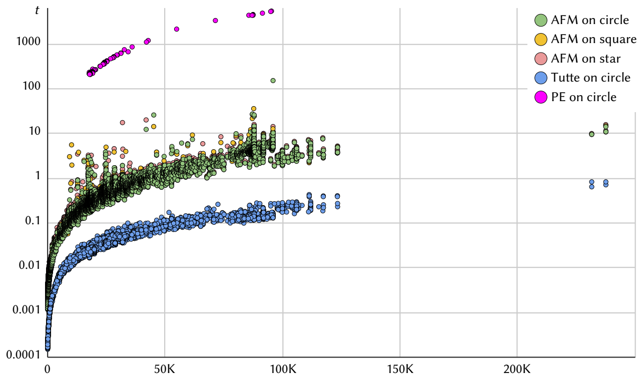

Figure 11: *Running times (in seconds) for AFM, Tutte [Tut63] and Progressive Embedding [SJZP19] on our testing dataset. For AFM we considered mappings to circles, squares and stars. Tutte and PE were only tested on mappings to strictly convex domains.*

Timings. Despite the single thread implementation and the use of computationally expensive rational numbers, on average AFM was able to complete each mapping task in less than a second (avg $0.78s$). As shown in Figure 11, floating point Tutte is by far the fastest algorithm (avg $0.04s$), one order of magnitude faster than AFM. PE with random initialization is much slower (avg $608s$), three and four orders of magnitude slower than AFM and Tutte, respectively. PE is the only method that consumed the whole 24h time budget. After 24h of computation only 139 out of the 12K maps were completed. The evaluation of PE is therefore limited to this restricted set of shapes. Conversely, Tutte was executed on the full set of 12K input shapes, which completed in 8.3 minutes.

For AFM, running times remain consistent across the target domains we considered, with mappings to circles and squares being slightly faster (avg $0.76s$) than mappings to stars (avg $0.83s$). This difference can be explained by observing that star mappings tend to refine more (Figure 13), thus introducing a computational overhead.

In a few cases our tool was stopped because the insertion of a triangle took more than two seconds, violating our testing policy. As can be noticed in Table 1 (percentages in the Timeout column), stopped processes had already completed more than the 92% of the map on average, meaning that only a handful of input triangles were missing in the target domain. Based on our analysis, this slowdown towards the end of the execution is entirely due to cascading issues with the rational numbers, which accumulate complexity throughout the computation. The snap rounding strategy discussed in Section 4 is naive and it only mitigates this issue, without entirely avoiding it. Possible remedies are discussed in Section 6. To give a reference, we considered 10 timed out cases and run AFM until convergence. In the worst case the algorithm took two days to complete. These timings seem compatible with recent exact numerical approaches that suffer from similar issues with cascaded constructions [NCB23, HC23].

Robustness (rational). When implemented with rational numbers AFM is fully robust. The correctness of our topological and geo-

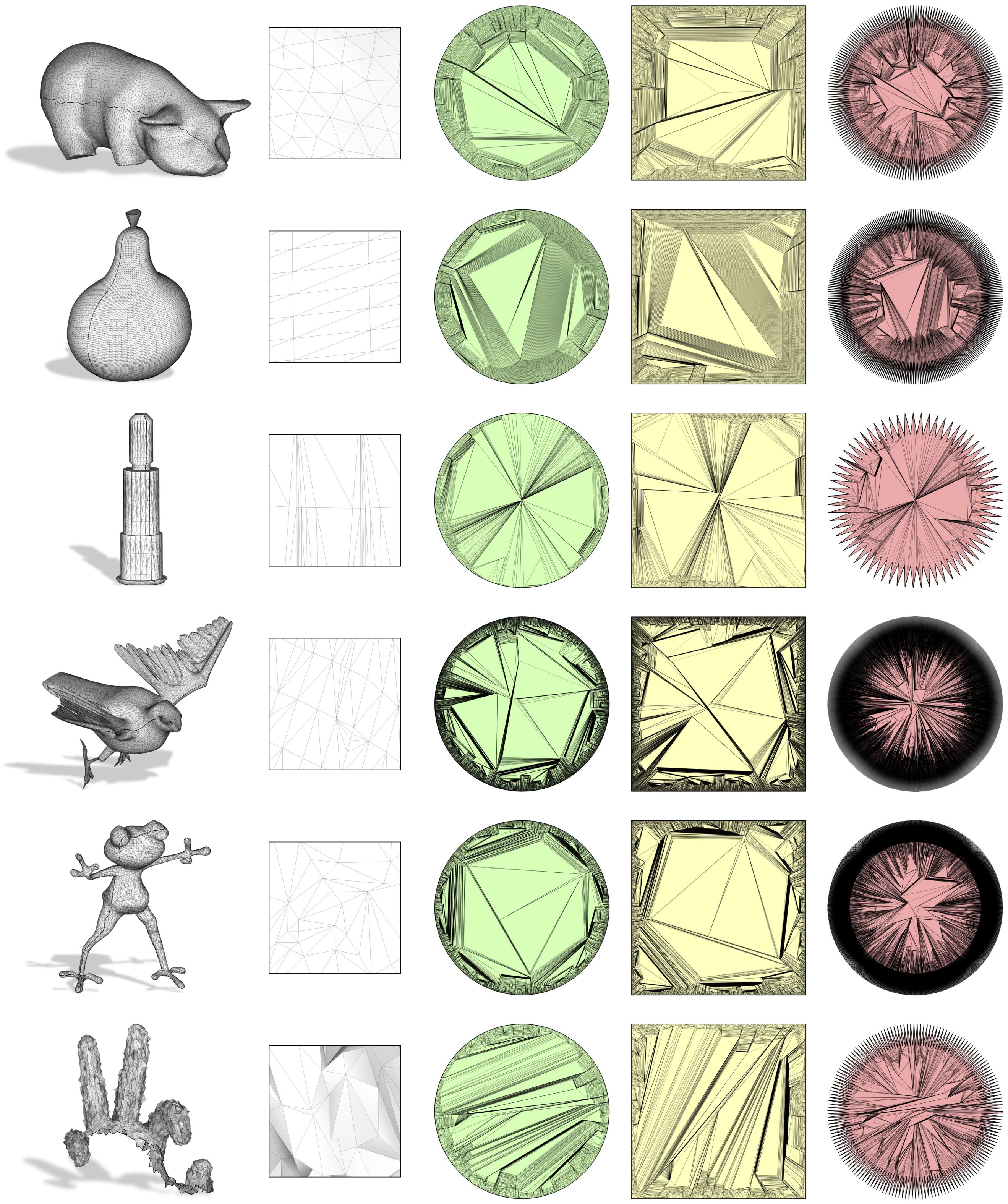
Figure 12: *Results obtained with AFM on a variety of meshes with different geometric and topological properties. All maps are fully injective. The second column shows a closeup of each mesh connectivity. Star domains appear different to one another because they depend on mesh resolution. They are all computed starting from a circular disk and then pushing odd vertices in the sequence inwards, creating the spikes.*

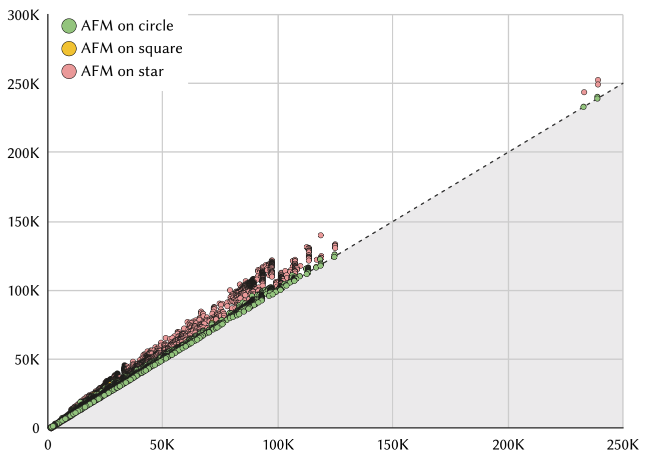

Figure 13: *Plot of output mesh size (vertical) over input mesh size (horizontal) for our 36K mapping tests. Points on the dashed line correspond to maps that introduced zero refinement. AFM only marginally refines the input mesh: mappings to circles and squares perfectly overlap and are almost entirely flat on the dashed line. Mappings to concave (star-shaped) domains are more likely to trigger refinement, but remain just slightly above it.*

	Guaranteed Injective	Floating Point Robust	Running Time	Supported Domains	Smart Refinement
Tutte	yes	99.96% 11946/11950	0.04s serial, float	convex only if a map exists	no
PE	yes	100% 139/139	608s parallel, float	convex only if a map exists	no
AFM	yes	70.8% 24715/34892	0.78s serial, rational	star-shaped	yes

Table 2: *Summary of the outcomes of our large-scale comparative analysis. Percentages in the second column refer to the amount of fully injective mappings in double precision produced by each method over the totality of the attempts. Third column reports average running times on the same attempts. PE was tested on 139 models (mapping to a circle). Tutte was executed 12K times (mapping to a circle) and AMF was executed 36K times (mapping to circles, squares and stars, 12K times each). For the supported domains column, Tutte and PE permit maps to non strictly convex domains (e.g. a square) only if the mesh does not contain triangles having all three vertices on the boundary. Conversely, AFM exploits local mesh refinement to automatically open the space of solutions in case a mapping does not exist.*

metric constructions was practically verified in the 36K runs AFM was tested on. More than one billion advancing moves were successfully completed without ever introducing a degenerate or inverted triangle in the embedding, including the cases when the process was stopped because of the elapsed runtime. As detailed in Table 1, this quantity is almost perfectly split into triangle splits and edge flips, with just a slight predominance of the latter over the former.

Another important aspect to consider when evaluating the practical robustness of a mapping method is its ability to not get stuck due to a vanishing space of solutions. A key advantage of AFM over prior provably robust methods is its ability to exploit on-demand local mesh refinement to create a valid solution even when it does not exist. While basic topological checks like the 2-connectedness of the graph could be executed in a pre-processing phase and remove some of the potential issues (Section 3.1.1), mappings to non convex domains may require non trivial refinement of the input mesh that is hard to determine a priori. In the worst case, the number of Steiner points that are necessary to grant the existence of a valid embedding are quadratic w.r.t. the number of boundary vertices [ASS93]. Our statistical results indicate that AFM indeed introduced more refinement to complete mappings to star-shaped domains, showcasing 18.1% mesh growth on average. Growth for circles and squares was 4.91% and 5.76%, respectively. In the worst case the amount of refinement reached a peak of 56.1% of the input mesh size. Nevertheless, our analysis revealed that the highest percentages of mesh growth belong to very coarse meshes containing either none or just a few internal vertices and edges, such as the one shown in Figure 2. Indeed, the full plot in Figure 13 shows that for the 24K maps to circles and squares input and output mesh sizes were almost identical, whereas for the 12K maps to stars output mesh size was only marginally higher than the input one.

Robustness (float). In many practical cases planar embeddings must be passed to downstream applications that operate in floating point. Alongside exact rational coordinates our implementation also maintains a floating point copy of the embedding, which we exploited to verify the practical usefulness of our tool. In the majority of the cases our floating point embeddings were free from degenerate or inverted elements (70.8%). Percentages are a bit lower for star-shaped domains (67.3%), and higher for circles and squares (72.9% and 72.3%, respectively). Considering that no serious attempts to achieve floating point robustness were made, we believe this is a promising result. Nevertheless, in this regard AFM performs worse than Tutte [Tut63] and PE [SJZP19], which is entirely designed to be floating point robust. In practical cases where a mapping to a convex domain suffices, these methods remain a preferable solution. In Section 6 we discuss how the technical solutions used in [SJZP19] could also be adopted in AFM to match their floating point robustness, possibly at a lower computational cost.

6. Conclusions and Future Works

We have introduced Advancing Front Mapping (AFM), a novel constructive algorithm for the computation of provably injective embeddings to simple planar domains. AFM expands the set of unconditionally robust tools for surface mapping, providing a method that natively supports a wider class of domains (star-shaped polygons) than existing direct methods [Tut63, SJZP19, FBRCA23], without resorting to intermediate constructions [WZ14]. In Appendices A and B we have formally proved the convergence and injectivity properties of AFM, also validating its capabilities extensively with a large-scale experimentation. The open source implementation of our algorithm will be released to the public domain upon acceptance of the article.

As openly discussed in Sections 1 and 5 and conveniently summarized in Table 2, AFM does not currently translate into a practical mapping tool and cannot currently be considered a valid alternative to prior robust methods such as [Tut63, SJZP19, WZ14], because of issues with efficiency and sensitivity to floating point implementations. It should be noted that none of these features are the core business of this article, which is rather focused on the presentation of a novel construction and its theoretical validation. In the remainder of this section we share a few considerations on future extensions of the core algorithm that may provide significant practical benefits in terms of these and other aspects, both in 2D and 3D.

Parallelization. Our current implementation relies on a serial mesh data structure, hence is single threaded. From an algorithmic perspective AFM could be trivially parallelized by applying simultaneous advancing moves in different regions of the active front. Designing fully disjoint advance operations is easy, because each move has a very local footprint. If the move is a triangle split, the only mesh element that changes is the triangle being split. If the advancing move is an edge flip, then two adjacent front edges plus the already inserted triangles that are incident to any of the three front vertices involved may change (due to the possibility of local refinement). In addition to this, it should be noted that even though all triangles inside the front share the same vertex (the front origin), such vertex never moves and its adjacencies are never read. Therefore, for the sake of a better parallel system, the front origin could just be duplicated so as to make each such triangles fully disjoint from the others, merging the various copies of the front origin just in a final step. Considering the 10 cores of our testing hardware, using a parallel-friendly mesh data structure such as [JDH*22] we could reasonably expected a boost factor of 4-5$\times$ or higher.

Warm Start. AFM is currently designed to create a surface map from scratch, always inserting *all* the input triangles in the target domain. Alternative methods such as PE hugely benefit from a warm start initialization (e.g., computed with Tutte), which permits to focus only on the very few spots where the map needs to be fixed. AFM could be similarly modified to enjoy a warm start, identifying the inverted triangles in the input map and then defining small star-shaped neighborhoods that fully contain each of them (e.g., via flooding). At this point, each of these regions could be considered separately and filled with our advancing front methodology. In the worst case scenario, the local neighborhood could grow to conquer the whole mesh, yielding a problem identical to the one we currently solve. Conversely, if smaller sized pockets around each illegal triangle are successfully found, each of them could be processed separately, reducing the computational cost and also providing yet another opportunity for parallelization.

Indirect Predicates and Cascading. As shown in multiple recent articles [CPAL22, CLSA20], switching from rational numbers to Indirect Predicates [Att20] provides great advantages in terms of efficiency without sacrificing robustness. Such a change should provide similar advantages also in our setting. Unfortunately, Indirect Predicates do not support cascading, hence the switch is currently incompatible with our implementation. A tempting idea to resolve the cascading problem consists in expressing all mapped points as a convex combination of the vertices of the polar mesh computed during initialization (Section 3.1.2). Specifically, since such a mesh is a valid tessellation of the target domain, any mesh point is strictly contained either in one of its triangles or edges. Devising alternative construction that allow to exactly represent split point locations with only one indirection is an appealing direction of further research.

Snap Rounding and Regularization. In their current form, some of the embeddings generated with AFM cannot be passed to a downstream application because switching to floating point may introduce degenerate triangles. Switching from an exact to an approximated numerical model in a robust manner is a notoriously complex problem, called 3D snap rounding. Besides the naive rounding to double described in Section 4, our method does not make any serious attempt to robustly generate floating point maps, failing at this task in the 29.2% of the cases (Table 2). Provably correct snap rounding algorithms are still too complex to be practically useful [DLL18]. Effective heuristics often used in the context of mesh arrangements ([ZGZJ16] §6.1) are not usable in our context because they require to collapse triangles in the input mesh, failing to preserve its original geometry. The only possible solution that we envision is to incorporate in our pipeline a regularization step that relaxes the geometry, avoiding almost degenerate configurations. This routine has been successfully used by Progressive Embedding [SJZP19], although in their case it introduces a huge computational overhead because vertex insertions may arise *anywhere* in the mesh, requiring a global optimization. Our approach operates only along the active front, keeping the geometry of the previously inserted triangles frozen. Introducing vertex relaxation only along the current front may permit to find a sweet spot between floating point robustness and numerical overhead. Experiments in this direction have not been attempted yet.

Volume Maps. Last but not least, an appealing property of AFM is its apparent compatibility with a volumetric extension. Provably injective volume mapping is a fundamental yet open problem in the literature ([Liv20b] §2), for which existing robust approaches fall short for their limited applicability [Ale23] or high computational cost and excessive amount of mesh refinement [HC23, NCB23]. Regarding AFM, the generation of the polar mesh in the initialization phase (Section 3.1) and the basic advancing moves by means of triangle splits and edge flips (Section 3.2) have a direct counterpart in tetrahedral meshes, with the only difference that in 3D there are two alternative flip operations to advance the front: a face flip move to conquer a tetrahedron having two faces on the front and an edge flip move to conquer a tetrahedron having three faces on the front (Figure 14). What remains unclear – and is subject to ongoing research – is the handling of locally concave configurations that prevent the execution of a flip. As discussed in Section 3.3 in 2D there are only two possible cases, that are fully addressed by our convexification and concavification strategies, whereas in 3D there are more, possibly more complex.

Akcnowledgements

Thanks are due to Daniele Panozzo, Teseo Schneider and Zhongshi Jiang, with whom I failed to make a preliminar version of this idea

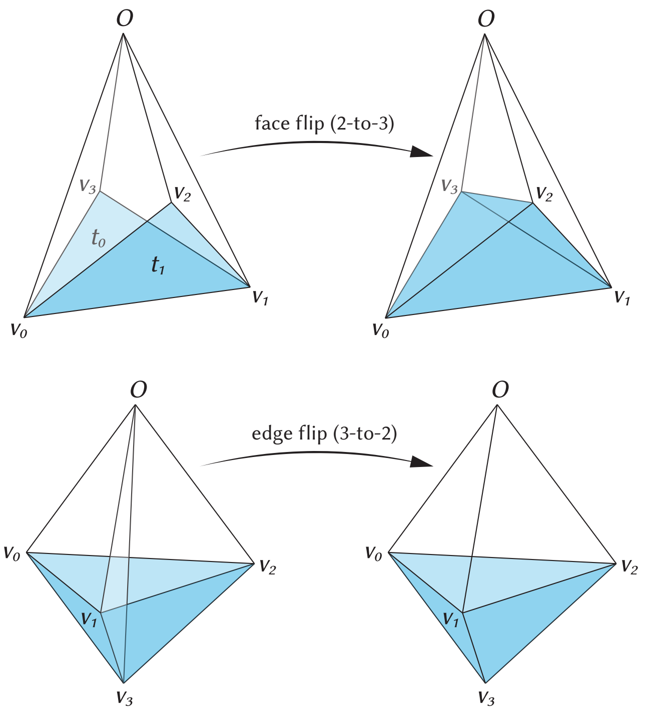

Figure 14: *Advancing front moves in a volume mesh by means of flipping operators. Top: tetrahedra having two triangular faces exposed on the front (t_1, t_2), can be inserted in the domain by flipping the triangle v_0, v_1, O. Bottom: tetrahedra having three faces exposed on the front can be reproduced by flipping edge v_3, O.*

work back in 2019. I am also grateful to Gianmarco Cherchi, Enrico Puppo, Marco Attene and Riccardo Scateni, who have patiently listened to me spelling the steps of this algorithm for years, often pointing me in the right direction to solve unexpected corner cases. This work was supported by the EU project DIGITbrain/ProMED (952071).

References

[AKL17] Aigerman N., Kovalsky S. Z., Lipman Y.: Spherical orbifold tutte embeddings. *ACM Trans. Graph. 36*, 4 (2017), 90. 3

[AL15] Aigerman N., Lipman Y.: Orbifold tutte embeddings. *ACM Trans. Graph. 34*, 6 (2015), 190–1. 3

[AL16] Aigerman N., Lipman Y.: Hyperbolic orbifold tutte embeddings. *ACM Trans. Graph. 35*, 6 (2016), 217–1. 3

[Ale23] Alexa M.: Tutte embeddings of tetrahedral meshes. *Discrete & Computational Geometry* (2023), 1–11. 13

[ASS93] Aronov B., Seidel R., Souvaine D.: On compatible triangulations of simple polygons. *Computational Geometry 3*, 1 (1993), 27–35. 2, 3, 12

[Att20] Attene M.: Indirect predicates for geometric constructions. *Computer-Aided Design 126* (2020), 102856. 13

[BCW17] Bright A., Chien E., Weber O.: Harmonic global parametrization with rational holonomy. *ACM Transactions on Graphics (TOG) 36*, 4 (2017), 1–15. 3

[CBSS17] Claici S., Bessmeltsev M., Schaefer S., Solomon J.: Isometry-aware preconditioning for mesh parameterization. In *Computer Graphics Forum* (2017), vol. 36, Wiley Online Library, pp. 37–47. 3

[CFH*18] Chai S., Fu X.-M., Hu X., Yang Y., Liu L.: Sphere-based cut construction for planar parameterizations. *Computers & Graphics 74* (2018), 66–75. 10

[CLH*16] Carlier A., Leonard K., Hahmann S., Morin G., Collins M.: The 2d shape structure dataset: A user annotated open access database. *Computers & Graphics 58* (2016), 23–30. 10

[CLSA20] Cherchi G., Livesu M., Scateni R., Attene M.: Fast and robust mesh arrangements using floating-point arithmetic. *ACM Transactions on Graphics (SIGGRAPH Asia 2020) 39*, 6 (2020). 9, 13

[CPAL22] Cherchi G., Pellacini F., Attene M., Livesu M.: Interactive and robust mesh booleans. *ACM Transactions on Graphics (TOG) 41*, 6 (2022), 1–14. 9, 13

[CSZ16] Campen M., Silva C. T., Zorin D.: Bijective maps from simplicial foliations. *ACM Transactions on Graphics (TOG) 35*, 4 (2016), 1–15. 3, 4

[DAZ*20] Du X., Aigerman N., Zhou Q., Kovalsky S. Z., Yan Y., Kaufman D. M., Ju T.: Lifting simplices to find injectivity. *ACM Trans. Graph. 39*, 4 (2020), 120. 3

[DKZ*22] Du X., Kaufman D. M., Zhou Q., Kovalsky S., Yan Y., Aigerman N., Ju T.: Isometric energies for recovering injectivity in constrained mapping. In *SIGGRAPH Asia 2022-Computer Graphics and Interactive Techniques Conference-Asia, SA 2022* (2022), Association for Computing Machinery, Inc, p. 36. 3

[DLL18] Devillers O., Lazard S., Lenhart W.: 3d snap rounding. In *Proceedings of the 34th International Symposium on Computational Geometry* (2018), pp. 30–1. 13

[FBRCA23] Finnendahl U., Bogiokas D., Robles Cervantes P., Alexa M.: Efficient embeddings in exact arithmetic. *ACM Transactions on Graphics (TOG) 42*, 4 (2023), 1–17. 2, 10, 12

[FH05] Floater M. S., Hormann K.: Surface parameterization: a tutorial and survey. *Advances in multiresolution for geometric modelling* (2005), 157–186. 1

[Flo97] Floater M. S.: Parametrization and smooth approximation of surface triangulations. *Computer aided geometric design 14*, 3 (1997), 231–250. 3

[Flo03a] Floater M.: One-to-one piecewise linear mappings over triangulations. *Mathematics of Computation 72*, 242 (2003), 685–696. 3

[Flo03b] Floater M. S.: Mean value coordinates. *Computer aided geometric design 20*, 1 (2003), 19–27. 3

[FSZ*21] Fu X.-M., Su J.-P., Zhao Z.-Y., Fang Q., Ye C., Liu L.: Inversion-free geometric mapping construction: A survey. *Computational Visual Media 7* (2021), 289–318. 1, 3

[FW22] Fargion G., Weber O.: Globally injective flattening via a reduced harmonic subspace. *ACM Transactions on Graphics (TOG) 41*, 6 (2022), 1–17. 1, 3

[GGT06] Gortler S. J., Gotsman C., Thurston D.: Discrete one-forms on meshes and applications to 3d mesh parameterization. *Computer Aided Geometric Design 23*, 2 (2006), 83–112. 3

[GJGQ05] Garner C., Jin M., Gu X., Qin H.: Topology-driven surface mappings with robust feature alignment. In *VIS 05. IEEE Visualization, 2005.* (2005), IEEE, pp. 543–550. 3

[GKK*21] Garanzha V., Kaporin I., Kudryavtseva L., Protais F., Ray N., Sokolov D.: Foldover-free maps in 50 lines of code. *ACM Transactions on Graphics (TOG) 40*, 4 (2021), 1–16. 3, 16

[Gra10] Granlund T.: The gnu multiple precision arithmetic library. *http://gmplib.org/* (2010). 9

[GS94] George P. L., Seveno É.: The advancing-front mesh generation method revisited. *International Journal for Numerical Methods in Engineering 37*, 21 (1994), 3605–3619. 2

[GSC21] Gillespie M., Springborn B., Crane K.: Discrete conformal equivalence of polyhedral surfaces. *ACM Transactions on Graphics 40*, 4 (2021). 3

[GW97] Gupta H., Wenger R.: Constructing piecewise linear homeomorphisms of simple polygons. *Journal of Algorithms 22*, 1 (1997), 142–157. 3

[HC23] Hinderink S., Campen M.: Galaxy maps: Localized foliations for bijective volumetric mapping. *ACM Transactions on Graphics (TOG) 42*, 4 (2023), 1–16. 3, 10, 13

[Hop96] Hoppe H.: Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (1996), pp. 99–108. 3

[HPS08] Hormann K., Polthier K., Sheffer A.: Mesh parameterization: theory and practice. In *ACM SIGGRAPH ASIA 2008 courses*. 2008, pp. 1–87. 1, 2

[JDH*22] Jiang Z., Dai J., Hu Y., Zhou Y., Dumas J., Zhou Q., Bajwa G. S., Zorin D., Panozzo D., Schneider T.: Declarative specification for unstructured mesh editing algorithms. *ACM Transactions on Graphics (TOG) 41*, 6 (2022), 1–14. 13

[JHT14] Jin Y., Huang J., Tong R.: Remeshing-assisted optimization for locally injective mappings. In *Computer Graphics Forum* (2014), vol. 33, Wiley Online Library, pp. 269–279. 3

[JSP17] Jiang Z., Schaefer S., Panozzo D.: Simplicial complex augmentation framework for bijective maps. *ACM Transactions on Graphics 36*, 6 (2017). 3

[KAD*20] Kovalsky S. Z., Aigerman N., Daubechies I., Kazhdan M., Lu J., Steinerberger S.: Non-convex planar harmonic maps. *arXiv preprint arXiv:2001.01322* (2020). 3

[KS04] Kraevoy V., Sheffer A.: Cross-parameterization and compatible remeshing of 3d models. *ACM Transactions on Graphics (ToG) 23*, 3 (2004), 861–869. 1, 3

[KSG03] Kraevoy V., Sheffer A., Gotsman C.: Matchmaker: constructing constrained texture maps. *ACM Transactions on Graphics (TOG) 22*, 3 (2003), 326–333. 1

[LBG*08] Li X., Bao Y., Guo X., Jin M., Gu X., Qin H.: Globally optimal surface mapping for surfaces with arbitrary topology. *IEEE Transactions on Visualization and Computer Graphics 14*, 4 (2008), 805–819. 3

[Lip14] Lipman Y.: Bijective mappings of meshes with boundary and the degree in mesh processing. *SIAM Journal on Imaging Sciences 7*, 2 (2014), 1263–1283. 4, 16

[Liv19] Livesu M.: cinolib: a generic programming header only c++ library for processing polygonal and polyhedral meshes. *Transactions on Computational Science XXXIV* (2019), 64–76. 9, 10

[Liv20a] Livesu M.: A Mesh Generation Perspective on Robust Mappings. In *Smart Tools and Applications in Graphics (STAG)* (2020), The Eurographics Association. 3

[Liv20b] Livesu M.: Mapping surfaces with earcut. *arXiv preprint arXiv:2012.08233* (2020). 3, 13

[Liv20c] Livesu M.: Scalable mesh refinement for canonical polygonal schemas of extremely high genus shapes. *IEEE Transactions on Visualization and Computer Graphics 27*, 1 (2020), 254–260. 3

[Liv23] Livesu M.: Towards a robust and portable pipeline for quad meshing: Topological initialization of injective integer grid maps. *Computers & Graphics 112* (2023), 50–59. 1, 3

[LSS*98] Lee A. W., Sweldens W., Schröder P., Cowsar L., Dobkin D.: Maps: Multiresolution adaptive parameterization of surfaces. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (1998), pp. 95–104. 1

[LYNF18] Liu L., Ye C., Ni R., Fu X.-M.: Progressive parameterizations. *ACM Trans. Graph. 37*, 4 (2018), 41–1. 1, 3, 10

[MW95] Marcum D. L., Weatherill N. P.: Unstructured grid generation using iterative point insertion and local reconnection. *AIAA journal 33*, 9 (1995), 1619–1625. 2

[NCB23] Nigolian V. Z., Campen M., Bommes D.: Expansion cones: A progressive volumetric mapping framework. *ACM Transactions on Graphics (TOG) 42*, 4 (2023), 1–19. 1, 3, 10, 13

[NNZ21] Naitsat A., Naitzat G., Zeevi Y. Y.: On inversion-free mapping and distortion minimization. *Journal of Mathematical Imaging and Vision 63* (2021), 974–1009. 1, 3

[PF11] Pion S., Fabri A.: A generic lazy evaluation scheme for exact geometric computations. *Science of Computer Programming 76*, 4 (2011), 307–323. 9

[POK23] Poya R., Ortigosa R., Kim T.: Geometric optimisation via spectral shifting. *ACM Transactions on Graphics 42*, 3 (2023), 1–15. 3

[PSS01] Praun E., Sweldens W., Schröder P.: Consistent mesh parameterizations. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), pp. 179–184. 1

[RPPSH17] Rabinovich M., Poranne R., Panozzo D., Sorkine-Hornung O.: Scalable locally injective mappings. *ACM Transactions on Graphics (TOG) 36*, 4 (2017), 1. 1, 3

[SAPH04] Schreiner J., Asirvatham A., Praun E., Hoppe H.: Inter-surface mapping. In *ACM SIGGRAPH 2004 Papers*. 2004, pp. 870–877. 1, 3

[SBS22] Sorgente T., Biasotti S., Spagnuolo M.: Polyhedron kernel computation using a geometric approach. *Computers & Graphics 105* (2022), 94–104. 5

[SG04] Surazhsky V., Gotsman C.: High quality compatible triangulations. *Engineering with Computers 20* (2004), 147–156. 3

[SJZP19] Shen H., Jiang Z., Zorin D., Panozzo D.: Progressive embedding. *ACM Transactions on Graphics 38*, 4 (2019). 1, 2, 3, 10, 12, 13

[SLS22] Stein O., Li J., Solomon J.: A splitting scheme for flip-free distortion energies. *SIAM Journal on Imaging Sciences 15*, 2 (2022), 925–959. 3

[SS15] Smith J., Schaefer S.: Bijective parameterization with free boundaries. *ACM Transactions on Graphics (TOG) 34*, 4 (2015), 1–9. 1, 3

[Tak22] Takayama K.: Compatible intrinsic triangulations. *ACM Transactions on Graphics (TOG) 41*, 4 (2022), 1–12. 3

[Tut63] Tutte W. T.: How to draw a graph. *Proceedings of the London Mathematical Society 3*, 1 (1963), 743–767. 2, 3, 10, 12, 13

[WGS23] Wang Y., Guo M., Solomon J.: Variational quasi-harmonic maps for computing diffeomorphisms. *ACM Transactions on Graphics (TOG) 42*, 4 (2023), 1–26. 3

[WZ14] Weber O., Zorin D.: Locally injective parametrization with arbitrary fixed boundaries. *ACM Trans. Graph. 33*, 4 (jul 2014). 2, 3, 12, 13

[ZGZJ16] Zhou Q., Grinspun E., Zorin D., Jacobson A.: Mesh arrangements for solid geometry. *ACM Transactions on Graphics (TOG) 35*, 4 (2016), 1–15. 13

Appendix A: Proof of convergence

Since the algorithm operates by inserting one triangle at a time, to prove convergence it is sufficient to show that: (i) each advancing move reduces the number of input triangles to be inserted in the embedding by at least one unit; (ii) it is always possible to execute an advancing move.

For (i): the advance by triangle split (Section 3.2.1) and advance by edge flip (Section 3.2.2) moves clearly reduce the number of triangles by one. The convexification strategy (Section 3.3.1) does not decrease the number of triangles, but it always permits the execution of a subsequent edge flip, thus ensuring that at least one more triangle can be inserted in the embedding. Finally, the concavification strategy (Section 3.3.2) takes two triangles inside the front, splits the edge shared by them creating four sub-triangles, and then inserts in the embedding three of them, thus reducing the total amount of triangles by one unit as well.

For (ii): triangles that touch the front with only one edge can always be inserted with a triangle split (Section 3.2.1), there are no restrictions on the applicability of this move. Triangles that touch the front with two edges can be inserted with an edge flip (Section 3.2.2) only if the four-sided polygonal pocket surrounding the edge to be flipped is convex. If such a polygon is not convex and the concavity lies along the front, the convexification strategy described in Section 3.3.1 is guaranteed to locally edit the front to make it convex, thus ensuring the insertion of the wanted triangle. Note that convexification may trigger iterative refinement, but this operation is always guaranteed to terminate (Figure 9). If the concavity lies at the opposite side (i.e., around the front origin O) the concavification strategy described in Section 3.3.2 can only be applied if the two triangles to be inserted form a triangle strip (i.e., all their vertices are already on the front). If this is the case, concavification can always be applied and the front be pushed forward. Conversely, if such triangles do not form a strip, then at least one of them will have a vertex inside the front, hence such a triangle could be inserted with a triangle split operation (Section 3.2.1), once again ensuring that at least one advancing move can always be executed and the method converges.

Appendix B: Proof of injectivity

We demonstrate injectivity by showing that all advancing and refinement moves used by AFM preserve the orientation of the triangles, which is set to be globally coherent in the initialization phase (Section 3.1.2).

Triangle and edge splits provably preserve the orientation of the refined elements if split points are inserted at a strictly convex combination of their vertices, that is, a point strictly inside a triangle for the triangle split and a point strictly inside the segment for the edge split. This is always the case for the operations described in Sections 3.2.1 and 3.3.1.

The edge flip operator preserves the orientation of the triangles if the pocket containing the edge to be flipped is convex. Our advancing front move by edge flip is always applied in such a case and, in case this property is not fulfilled, provably convergent routines are used to ensure that the convexity requirement holds (Sections 3.2.1 and 3.3.1). Therefore, no inverted triangles can be inserted in this step either.

Combining all these ingredients, we can guarantee that all triangles in the final embedding share a globally coherent orientation. Note that this condition alone is not sufficient to ensure the injectivity of the map because non injective configurations may still arise around a mesh vertex. Failure cases of this kind can be found in Figure 1 in [Lip14] and in Figure 16 in [GKK*21]. Nevertheless, coupling the coherent orientation property with the bijectivity of the boundary map, which in our case is an input requirement (Section 3), the so generated simplicial maps are guaranteed do be bijective. A formal proof of this statement is contained in [Lip14] (Theorem 3) .