Stripe Embedding: Efficient Maps with Exact Numeric Computation

MARCO LIVESU, CNR IMATI, Italy

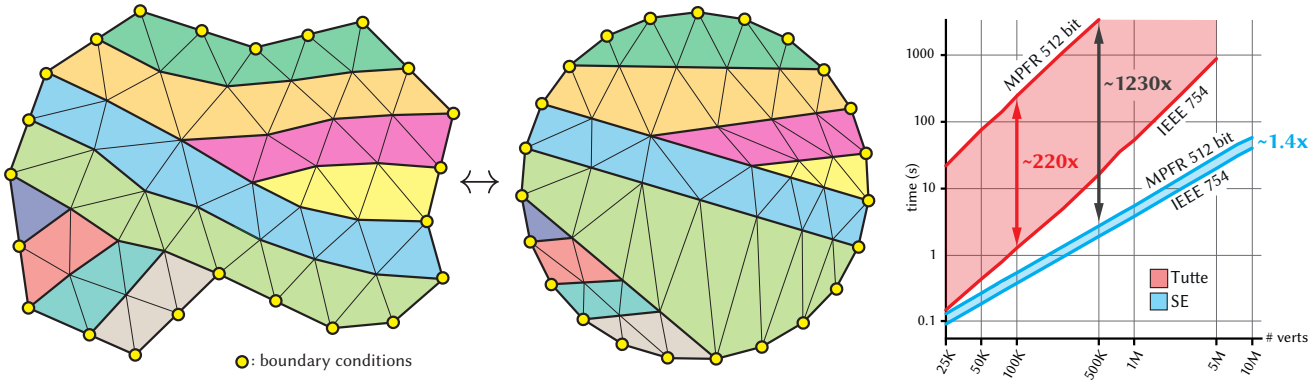


Fig. 1. Left: decomposing a mesh into triangle strips and rectifying them in the target domain yields an injective map. Right: the Stripe Embedding algorithm exploits this principle to compute planar maps with extreme efficiency at any scale, outperforming prior art by more than three orders of magnitude (black arrow). Most notably, switching from IEEE 754 floating points to exact or higher precision numerical models introduces only a minor slowdown (blue shaded band). Conversely, prior methods increase running times by more than two orders of magnitude, as highlighted by the red shaded band in the plot.

We consider the fundamental problem of injectively mapping a surface mesh with disk topology onto a boundary constrained convex domain. We start from the basic observation that mapping a strip of triangles onto a rectangular shape always yields a valid embedding, if the vertices that bound the strip are sorted coherently along the sides of the rectangle. Based on this intuition, we propose a straightforward algorithm, called Stripe Embedding, that operates by decomposing the input mesh into a set of triangle strips and then embeds each strip into the target domain by means of linear interpolation between two previously embedded vertices. Thanks to its simplicity, Stripe Embedding is extremely efficient and permits to switch to an exact implementation without almost increasing its running times. Stripe Embedding is up to three orders of magnitude faster than the Tutte embedding for same numerical model and, even when implemented with costly rational numbers, it is faster than any floating point implementation of prior methods at any scale.

ACM Reference Format:
Marco Livesu. 2024. Stripe Embedding: Efficient Maps with Exact Numeric Computation. *ACM Trans. Graph.* 43, 6 (December 2024), 14 pages. https://doi.org/10.1145/3687915

1 INTRODUCTION

Exact Numeric Computation (ENC) is a scientific computation paradigm that was established almost 30 years ago as an alternative to the floating point paradigm [Yap and Dubé 1995]. It is based around the idea that to transform *theoretical* guarantees into *software* guarantees all numerical values should be represented exactly and all branching decisions should be error-free. In practical terms, ENC advocates the use of multi-precision arithmetic and is nowadays achieved thanks to a variety of numerical models and robust geometric predicates. In computer graphics, the importance of ENC is well documented by numerous failures on the resolution of even basic geometric tasks [Kettner et al. 2008]. To date, ENC is extensively used to reliably solve critical problems involving mesh generation and shape modeling (Section 2.1).

Unlike these fields, the computation of injective maps still relies on non robust numerical approaches [Fu et al. 2021; Naitsat et al. 2021]. Recent literature clearly shows that even mapping methods that provide strict theoretical guarantees of correctness exhibit a significant number of failures when implemented in fixed precision [Finnendahl et al. 2023; Shen et al. 2019], preventing their use as trusted black boxes that can reliably process large amounts of data (e.g., for training in data-driven approaches). The robust computation of injective planar maps has received increasing attention, but scalable methods that successfully combine efficiency and guarantees into the same software are yet to be found (Section 2.2).

It is important to realize that theoretical guarantees of correctness cannot always transition to a fully robust software implementation. Mapping algorithms must be *designed* to be compatible with some exact numerical model that can be implemented in a computer (Section 2.1). To make a concrete example, algorithms such as [Campen et al. 2021; Gillespie et al. 2021] are both guaranteed correct under the assumption of infinite precision, but they internally rely on transcendental numbers that cannot be evaluated exactly, making these methods intrinsically incompatible with the principles of ENC. In [Gillespie et al. 2021] the authors report 2.3% failure rate on Thingi10K (see §8.3.2). Robustness issues can be mitigated by

Author's address: Marco Livesu, CNR IMATI, Italy, marco.livesu@gmail.com.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
0730-0301/2024/12-ART $15.00
https://doi.org/10.1145/3687915

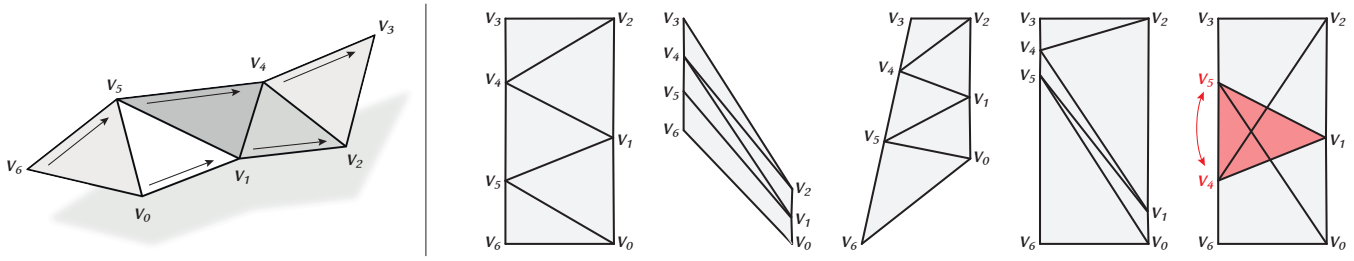


Fig. 2. A triangle strip (left) can always be injectively mapped to a rectangular domain (right) if the lateral vertices of the strip are coherently sorted along the sides of the rectangle. Flipped elements can only arise if the vertex sorting is violated (rightmost column, red triangle). Note that the same argument also holds for degenerate strips where one of the two lateral chains contains a single vertex. In such a case the strip is a triangle fan and the domain is triangular.

Table 1. Number of cases in which Tutte and Stripe Embedding failed to produce a bijective map when implemented in fixed precision and launched on 12.1K mapping tasks. Four alternative numerical models have been considered. In the top line for each model we indicate how many binary digits are used to represent the mantissa in the floating point encoding.

	IEEE 754 53 bit	MPFR 128 bit	MPFR 256 bit	MPFR 512 bit
Tutte	146	32	4	0
	1.2%	0.26%	0.03%	0%
Stripe Embedding	207	27	3	0
	1.7%	0.22%	0.02%	0%

using floating points with extended precision, but this requires to fix the machine epsilon a priori, meaning that a correct result can only be achieved via trial and error, with no guarantees of success (Table 1). Besides numerical issues, the other major shortcoming of robust computation is the overhead introduced by multi-precision arithmetic, which is known to increase running times by orders of magnitude (Figure 1, right).

In this paper we introduce a novel algorithm that opens to the fully robust and highly efficient computation of planar maps in an unprecedented way, exhibiting scalability and performance features that are nowhere found in previous methods (Section 4). Specifically, we consider the computation of injective embeddings onto a boundary constrained convex polygon. Maps of this kind find direct application in the computation of shape correspondences and generation of high quality uv maps (Section 2).

Our mapping strategy is based on a fairly simple geometric intuition: a triangle fan can always be injectively mapped to a triangular domain if the lateral vertices of the fan are sorted coherently (Figure 2). We exploit this core idea by decomposing the input mesh into a set of triangle fans, embedding each one separately. The actual map computation simply amounts to positioning the lateral vertices that bound each fan via linear interpolation between two previously embedded vertices, meaning that the algorithm solely relies on an arithmetically straightforward operation that is closed under rational numbers. This makes Stripe Embedding intrinsically much faster than alternative approaches, which are either based on the resolution of a sparse linear system [Tutte 1963] or involve the computation of segment-segment intersections [Livesu 2024].

Our extensive analysis shows that in terms of efficiency and scalability Stripe Embedding is superior than prior art by a large margin (Section 4). We considered more than 12K mapping tasks, on meshes that span from 150 to 10M vertices. Stripe Embedding resulted faster on any mesh, at any scale, with any numerical model, exact or approximate. Measured speed up factors are even beyond three orders of magnitude, also exhibiting a better asymptotic behavior. All in all, we believe that the Stripe Embedding algorithm provides a convenient all-round drop-in replacement of any other existing method for the computation of boundary constrained mappings. To promote early adoption of our algorithm, we release an open source implementation of our tool.

2 RELATED WORKS

Mapping a 2D signal to a 3D surface is a fundamental tool for texturing, remeshing, and generation of shape correspondences, to name a few [Hormann et al. 2008]. Surface maps must contain little geometric distortion and be injective. Distortion energies are non linear, hence methods in this class necessarily need an initial guess to start the optimization process. The injectivity of the map is guaranteed only if the initial guess does not contain inverted elements. All existing methods fulfill this goal by using a map to a convex polygon as initial guess [Fargion and Weber 2022; Fu and Liu 2016; Jiang et al. 2017; Liu et al. 2018; Rabinovich et al. 2017; Smith and Schaefer 2015; Su et al. 2020; Wang et al. 2023; Ye et al. 2020]. Therefore, methods for boundary constrained mapping are the key ingredient to bootstrap these pipelines and are expected to be fully robust, both theoretically and in practice. In this section we first discuss available numerical tools to transfer theoretical guarantees into a software implementation. We then report on existing provably correct methods, also commenting on their practical robustness and compatibility with existing numerical models.

2.1 Numerical models

A primary distinction between quantities that can be exactly represented in a computer and quantities that cannot is the one between *algebraic* and *transcendental* numbers. Algebraic numbers are the root of a non zero polynomial. All integer and rational numbers are algebraic, but also all squared roots with rational argument and some complex and irrational numbers such as the golden ratio φ, which is the root of $x^2 - x - 1$. All numbers that are not

algebraic are called trascendental. This category includes most of the irrational numbers but also trigonometric functions. It follows that when designing an exact algorithm one should always avoid using quantities such as π or functions such as $\cos(x), \sin(x)$ or $\cot(x)$. Otherwise, even if the algorithm is formally correct from a mathematical standpoint, guarantees can hardly be preserved by any software implementation.

Besides the ability to represent quantities exactly, the overhead introduced by the numerical model of choice should always be considered. Software tools such as LEDA [Burnikel et al. 1996] and CORE [Karamcheti et al. 1999] support all algebraic numbers hence provide great flexibility. However, they are notoriously slow and tend to increase running times by several orders of magnitude with respect to a plain floating point implementation [Diazzi et al. 2023]. These numbers are often used by algorithms that unavoidably require the computation of squared roots (e.g. for vector normalization or for the insertion of Steiner points in a CDT [Diazzi et al. 2023; Si and Gärtner 2005]).

A slightly restrictive but more efficient choice is the design of algorithms that are closed under rational numbers. These numbers are internally represented using integers with arbitrary length. Rationals are well supported by a variety of tools such as GMP [Granlund 1996], LEDA [Burnikel et al. 1996] and CORE [Karamcheti et al. 1999]. The CGAL library can conveniently wrap any of these tools, also coupling them with lazy evaluation to reduce the number of costly exact evaluations to the minimum [Pion and Fabri 2011]. Many robust mesh processing algorithms employ rationals [Cherchi et al. 2022; Fang et al. 2022; Guo et al. 2023; Hu et al. 2018; Khanteimouri et al. 2022; Yang et al. 2022; Zhou et al. 2016], also to compute maps [Campen et al. 2016; Livesu 2024].

Assuming the input mesh coordinates are integer, exact numerical computation can also be implemented using integers with arbitrary length. This numerical model is faster than the previous ones but for typical floating point inputs it can only be exploited after the coordinates are snapped to the integer grid, introducing an initial approximation that may or may not be compatible with the application at hand. In the general case robust integer snapping is an open problem by itself [Devillers et al. 2018], hence this operation cannot be always implemented efficiently. The mapping algorithm E^3A [Finnendahl et al. 2023] and some of the best performing exact mesh boolean algorithms [Nehring-Wirxel et al. 2021; Trettner et al. 2022] operate on integer numbers, although for maximum efficiency they internally use integers with fixed length.

Mapping methods such as [Campen et al. 2021] increase accuracy by operating on floating point numbers that are bigger than the ones supported by the IEEE 754 standard. A popular tool to operate on these numbers is MPFR [Fousse et al. 2007]. In this case a fixed precision must be determined a priori. Failure cases may therefore arise, even when using large floating point numbers (Table 1).

Robust predicates. Certain geometric tasks only require to correctly evaluate the sign of a polynomial, without the need to know the exact value. This is the case for many mesh generation problems, where one wants to typically answer basic geometric queries such as: *is this angle acute or concave? is this point above or below this plane?* These yes/no questions are critical to determine the correct algorithmic flow and can be reliably answered using robust geometric predicates [Lévy 2016; Shewchuk 1997]. These predicates are designed to operate directly on the input mesh coordinates, which are assumed exact by definition, but cannot reliably operate on intermediate constructions because they may be affected by approximation errors. Recent extensions [Attene 2020; Lévy 2024] also permit to exactly query intermediate quantities derived from the input points. These latter predicates are the key ingredient to robustly implement mesh Booleans and CSG pipelines [Cherchi et al. 2020, 2022; Diazzi and Attene 2021; Diazzi et al. 2023; Lévy 2024], but they only support only one level of indirection and are therefore not compatible with cascaded constructions. For this reason, they are not directly usable for the mapping problem that we consider.

The algorithm that we propose is closed under rationals, hence it can be implemented exactly using these numbers, or approximately using IEEE 754 floating points or MPFR for increased (fixed) precision. In Section 4 we evaluate its performances with all these numerical models, comparing it with prior art. When non exact numerical models are used, robust predicates are used to verify map injectivity.

2.2 Maps To Boundary Constrained Domains

The Tutte embedding [Floater 1997; Tutte 1963] is by far the most widely used algorithm for the computation of injective maps to a boundary constrained convex domain. It operates by positioning each internal point at the barycenter of its neighbors and, assuming infinite precision, it is guaranteed to always produce a correct result.

Shen and colleagues [2019] showed that, despite being formally correct, Tutte fails to provide an injective map in a significant number of cases when implemented in floating point. They isolated 80 failures out of 2718 mapping tasks from Thingi10K (2.94%). The alternative floating point method they proposed, Progressive Embedding (PE), is more robust and reached 100% success rate on the same dataset but it was also shown to fail on midpoint refinements of the same 80 meshes [Finnendahl et al. 2023]. To better understand its behavior we repeated this experiment independently, splitting each triangle into four sub-triangles. PE failed on 3 out of 80 models (3.75%). The coarsest failure mesh contains less than 18K vertices (model id: #40051). Applying one additional step of refinement PE consistently fails on the vast majority of models, after hours or even days of computation. This is because PE achieves its floating point robustness by performing Newton-based distortion optimization after each vertex insertion, thus creating a conspicuous computational overhead.

From an implementative standpoint, the Tutte embedding requires solving a sparse linear system, hence it is fully compatible with a multi-precision implementation. Conversely, Progressive Embedding is specifically designed to operate on floating points and internally relies on thresholds that intrinsically limit exact numerical models. Furthermore, in our experiments the parallel floating point code provided by the authors (bootstrapped with floating point Tutte) is orders of magnitude slower than our multi-precision implementation of Tutte itself. We therefore conclude that PE is not an interesting candidate for Exact Numeric Computation.

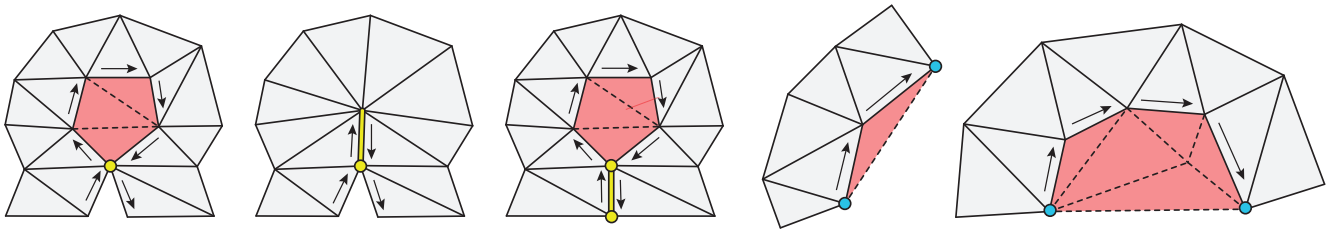


Fig. 3. Triangle strips that do not admit a valid embedding because one of the chains that bound them from one side contains duplicated elements (yellow) or vertices that are not adjacent in the chain but directly adjacent in the underlying mesh (blue). In all these cases, rectifying the chain in the embedding introduces flipped elements or vanishes entire mesh regions (red).

Advancing Front Mapping [Livesu 2024] was recently proposed as an alternative construction for the computation of provably injective planar maps. AFM supports a wider range of target domains (star-shaped polygons) and uses mesh refinement to provably find a valid solution. Thanks to the superior simplicity of our geometric constructions, our method is one order of magnitude faster than AFM and also does not suffer from cascading issues with rationals (Section 4.3), which occasionally stretch their running times beyond one day of computation.

The E^{3}A algorithm [Finnendahl et al. 2023] operates on integer numbers and is, to our knowledge, the fastest existing algorithm for the exact computation of boundary constrained mappings. However, it only supports maps to 3-sided domains, hence it solves a much less general version of the problem that we consider.

Certain methods for boundary constrained mapping exploit map composition to support a class of domains that is wider than convex polygons [Kraevoy and Sheffer 2004; Livesu 2021, 2023; Schreiner et al. 2004; Weber and Zorin 2014]. Since these methods internally use the Tutte embedding for the computation of the atomic maps, they offer the same guarantees of correctness provided by Tutte, which also bounds the performances they can achieve.

3 STRIPE EMBEDDING

The Stripe Embedding algorithm (SE) inputs a triangle mesh M with disk-like topology and a bijective function

$$\Phi_\partial : \partial M \leftrightarrow \partial\Omega$$

that maps its boundary vertices on the perimeter of a convex domain $\Omega \subset \mathbb{R}^2$. Function Φ_∂ is provided in discrete form, in the sense that it simply amounts to a set of prescribed target positions for all boundary vertices of M. The output of the algorithm is a completion of the input map, which is extended to the interior of the two domains

$$\Phi : M^* \leftrightarrow \Omega\,.$$

The output mesh M^* may occasionally be a refined version of M. As discussed in Section 4.7 such a refinement is almost never necessary in practical cases and, even when performed, it has negligible effect on the output mesh size.

Our mapping strategy is based on a stripification of the input mesh (Figure 1, left). Stripe decomposition has been historically used in graphics to optimize caching for efficient rendering [Hoppe 1999; Porcu and Scateni 2006] or for mesh compression [Rossignac 1999; Taubin and Rossignac 1998], but these methods optimize for criteria that do not relate with our goal. The whole algorithm is based on a straightforward yet powerful geometric observation: if we can sort the lateral vertices of a triangle strip along the sides of a convex rectangle, then the triangles always maintain a correct orientation, realizing an injective (per stripe) map, as shown in Figure 2.

3.1 Mapping a single strip

Let us focus on a single strip to better illustrate the mapping process. From a topological perspective, a strip is entirely defined by two disjoint chains of vertices that bound it from the two sides and by two mesh edges that connect the chains' endpoints and bound it at the top and bottom. Without loss of generality we can consider just one of the two lateral chains, which we can imagine as a sorted sequence of n vertices $v_0, \ldots, v_{n-1}$. Assuming the embedded coordinates of the endpoints v_0, v_{n-1} are known, an order preserving embedding for all intermediate vertices $v_1, \ldots, v_{n-2}$ can be computed with trivial linear interpolation

$$v_i = (1-t)\cdot v_0 \;+\; t\cdot v_{n-1}, \quad \text{with} \quad t = \frac{n-1-i}{n-1}. \tag{1}$$

Indeed, this is the only arithmetic operation that the algorithm does to compute the output map Φ and it always preserves injectivity if the polygon that bounds the strip is convex (Section 3.5). The simplicity of this mapping function has important implications on the efficiency of the algorithm and on its ability to switch from approximate to exact numerical models almost without increasing its running times (Sections 4.3 and 4.4).

3.2 Stripe requirements

To be compatible with our algorithm, the chains of vertices that bound a triangle strip must fulfill two important requirements:

- there must exist a coherent vertex sorting that can be reproduced in the target domain via linear interpolation. This is always possible if the chain does not contain duplicated elements. The condition is violated when the chain is self-tangent along a vertex or edge, or when the strip winds around a mesh vertex (first three columns in Figure 3). In all these cases, strip triangles incident to the duplicated elements become flipped in the embedding and the mesh portions that remain trapped inside the chain completely vanish;

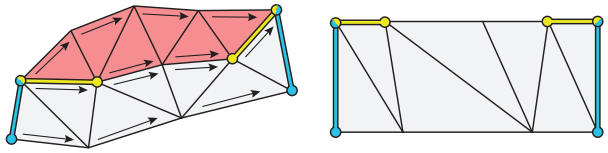


Fig. 4. The red strip starts and terminates at the thick yellow edges. The gray strip at the thick blue edges. Since yellow edges belong to the lateral chain that bounds the gray strip from one side, when the gray strip is rectified in the embedding the red one entirely collapses to a segment (right).

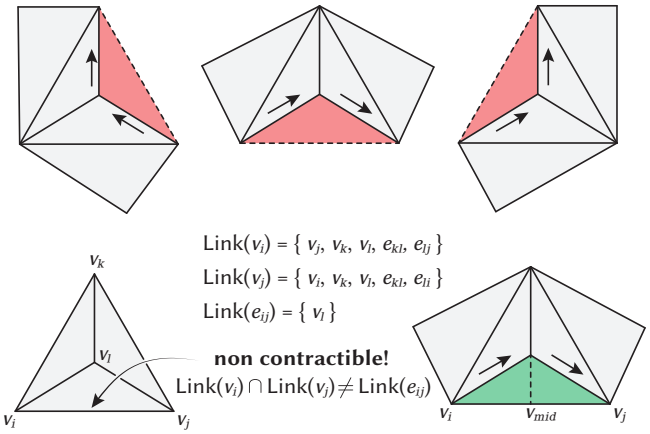


Fig. 5. Top: an outer triangle split into three sub-triangles does not admit a valid stripification. The middle valence three vertex can be traversed by a chain in three alternative ways, but in all such cases the chain contains all three vertices of a sub-triangle, making it inevitably vanish when the chain is rectified in the embedding. Bottom: these pathological configurations arise in the presence of non contractible edges that do not satisfy the link condition [Dey et al. 1999]. Splitting non contractible edges ensures the existence of a solution (bottom right).

- non consecutive vertices in the chain must not be directly connected by a mesh edge (last two columns in Figure 3). Should this happen, the portion of mesh that is external to the strip and is sandwiched between the chain and the offending edge collapses to a segment in the target domain.

Additionally, we must also ensure compatibility between adjacent strips. In particular, a triangle strip cannot start and terminate from the same side of another strip, otherwise its four endpoints will become colinear in the embedding, entirely collapsing it (Figure 4).

3.3 Stripe decomposition

The space of all possible stripifications is linked with the input mesh topology and grows exponentially with the number of triangles. Exploring such a big space to find a stripification that fulfills all the requirements above and entirely covers the input mesh may be inefficient. Furthermore, the constraints above raise an important theoretical question: does a valid stripification always exist for a given mesh? As shown in the top part of Figure 5 the answer to this question is, sadly, no. Our stripification strategy addresses both efficiency and feasibility issues thanks to two key observations.

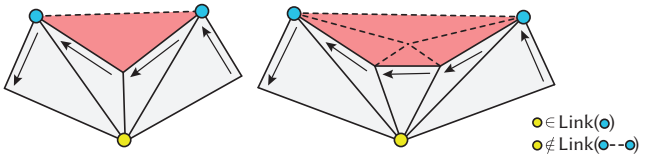


Fig. 6. Pathological strips containing two non consecutive chain vertices (blue) that form an edge in the underlying mesh (dashed line). Since the opposite chain contains a single vertex (yellow), such a vertex will by definition be part of the link of the blue ones. At the same time, since the red triangles are not part of the strip, none of them will contain the yellow vertex, meaning that the dashed edge is always non contractible. Notice that this is true only for the special case in which one chain contains a single vertex. For the more general cases shown in the last two columns of Figure 3, the offending edges are contractible.

For efficiency, we observe that since the pathological cases may involve distant vertices in the same chain, reducing the chain length to the minimum also minimizes the computational cost for sanity checks. We push this idea to the limit, considering only degenerate triangle strips where one of the two lateral chains is composed of a single vertex. As a result, for any given strip only one lateral chain must be checked for validity and its size is bounded by the local vertex valence, which is typically a very low number (6 on regular meshes).

For feasibility, the failure case in the top part of Figure 5 is just a minimal example of what can go wrong. Arbitrarily complex non feasible configurations can be easily created by further refining the inner sub-triangles. Our main intuition is that the key pathological element that makes a configuration non feasible is the presence of *non contractible* edges. An edge is contractible if it can be collapsed to a single point without changing the topology of the mesh [Dey et al. 1999]. Indeed, all dashed edges in the figure are non contractible and any further refinement of the inner sub-triangles does not affect this property. Edge contractibility has been largely studied in discrete math to understand topology preserving reductions of simplicial complexes. Let us define the link of a vertex as the set of vertices and edges in its one ring. Similarly, the link of an edge is the set of vertices that are connected to it trough some other edge or triangle. Contractibility can be easily verified with a local test: given two vertices v_i and v_j and the edge e_{ij} connecting them, e_{ij} is contractible if it satisfies the *link condition*

$$\text{Link}(v_i) \cap \text{Link}(v_j) = \text{Link}(e_{ij}), \tag{2}$$

it is non contractible if it does not satisfy it (Figure 5, bottom). We ensure that the input mesh M admits a valid stripification by detecting and splitting all non contractible edges in pre-processing. In Appendix A we prove that this process always terminates in a finite number of splits.

The combination of using minimal strips and removing non contractible edges jointly provide a great additional advantage. In fact, they also ensure that none of the pathological cases shown in Figure 3 can arise, almost entirely removing the need for explicit sanity

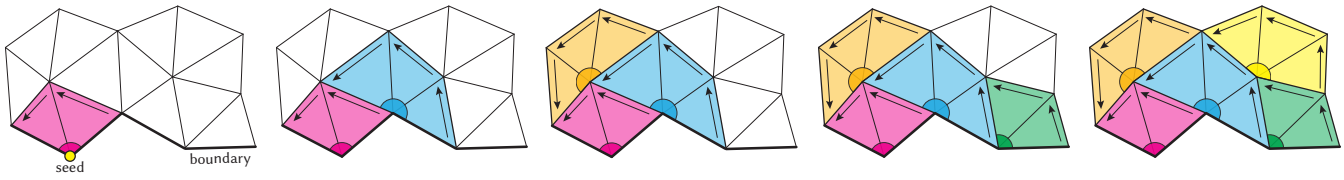


Fig. 7. Strip propagation from a boundary seed vertex. For maximum efficiency we generate minimal triangle strips that cover just the unvisited portion of the one ring of the vertex. Each time a new strip is generated, we push into a queue the chain of neighbors opposite to the current seed (arrow chains). The process continues until the whole mesh is flooded.

checks. Specifically, duplicated elements along the same chain cannot arise if the chain is a subset of the one ring of a mesh vertex, otherwise the mesh would not be a valid manifold. Furthermore, if one chain contains a single vertex, then the presence of mesh edges that connect non consecutive vertices in the opposite chain implies non contractibility, which is avoided by our refinement (Figure 6). If follows that the only sanity check that needs to be implemented explicitly is the compatibility between adjacent strips (Figure 4), which is extremely efficient because it only involves a local check at the chain endpoints, to make sure that they are not incident to a shared previous chain.

3.4 Putting everything together

We are now ready to assemble all the ingredients to create Stripe Embedding, which is fully described in Algorithm 1. As can be noticed the pseudo-code is quite simple, which greatly contributes not only to the efficiency but also to the ease of implementation and deployment of SE.

Stripe Embedding is a progressive algorithm: it starts from the perimeter of the target domain $\partial\Omega$ and embeds the strips proceeding inwards. We use a queue Q to control the stripification progress. In the initialization phase, Q contains all boundary vertices of M, which all form a convex angle in $\partial\Omega$ due to the convexity of the domain. During the iterative step, we pop a vertex v from the queue and create a new triangle strip that is bounded by v itself from one side and by a portion of its one ring at the other side. The portion of one ring is selected by first detecting a previously embedded vertex that is adjacent to a vertex that is not yet embedded. Then, following the topological sorting around v, we conquer all vertices in the ring until another embedded vertex is found (Figure 7). If the so generated chain is incompatible with a previously embedded one it is discarded (Figure 4), otherwise it is embedded via linear interpolation and all its vertices are pushed into Q. Vertex v is also pushed again in the queue for further processing if some of its neighbors have not been embedded yet. This may happen for example when the one ring of a vertex touches the boundary at multiple connected components, yielding more triangle strips that bridge the various portions of boundary. The process terminates when the queue is empty and all vertices have been embedded.

The algorithm is guaranteed to always converge, because the stripe compatibility condition in Figure 4 cannot be globally violated hence all non visited vertices will be embedded. The proof of this statement follows by the famous *ear theorem* [Meisters 1975], which ensures that all simple polygons have at least two ears. In our construction convex ears arise either at the boundary of the target domain or at the intersection point of two alternative stripes, always ensuring the existence of at least two vertices to insert a new valid strip.

In Figure 8 (first two columns) we show a typical stripe decomposition obtained with our algorithm and its induced embedding. Note that since the algorithm is vertex-based and terminates when all vertices are positioned in the domain, triangles whose three vertices have been embedded due to their presence in previous strips are not explicitly involved in the stripification process.

ALGORITHM 1: Stripe Embedding (SE)

inputs : a triangle mesh M with disk-like topology and a function $\partial\Phi : \partial M \leftrightarrow \partial\Omega$ that maps its boundary vertices to the perimeter of a convex domain Ω.
outputs: a mapping function $\Phi : M^* \leftrightarrow \Omega$, where M^* is a possibly refined version of the input mesh M.

$M^* = M$, with all non contractible edges split;
Q = queue with all border vertices of M^* that form a valid ear in $\partial\Omega$;
Mark all inner vertices of M^* as not embedded;
while Q *is not empty* **do**
  $v = Q$.pop();
  N_v = ordered one ring of v;
  Find index i such that n_i is embedded and $n_{(i+1) \bmod |N_v|}$ is not;
  $C_v = \{n_i\}$;
  repeat
    $i = (i+1) \bmod |N_v|$;
    Append n_i in C_v;
  until n_i *is not embedded*;
  if $C(v)$ *violates stripe compatibility* **then**
    continue; (Figure 4)
  end
  Embed chain C_v in Ω via linear interpolation; (Section 3.1)
  Mark all vertices in C_v as embedded;
  Push all vertices in C_v inside Q;
  if N_v *still contains not embedded vertices* **then**
    push v inside Q;
  end
end

3.5 Injectivity

Stripe Embedding is guaranteed to always produce an injective map where all triangles are non degenerate and share a globally coherent orientation. A major point is played by the convexity

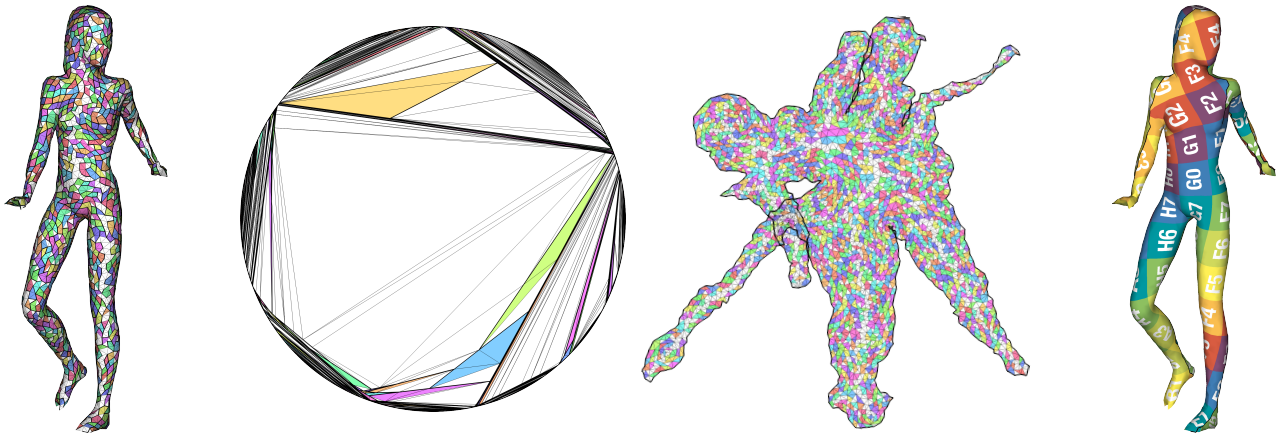


Fig. 8. Stripification of a triangle mesh (left), then embedded in the unit circle (middle left) and used to bootstrap SLIM [Rabinovich et al. 2017] to reduce the Symmetric Dirichlet distortion (middle right), generating a high quality surface map (right). Color coded triangles represent the various strips. White triangles denote elements whose three vertices were embedded prior to be involved in a strip. This happens when *opposite* expansion fronts meet.

of the computational domain, which is guaranteed by the input requirements and preserved by construction because we always split Ω along linear cuts. In a sense, Stripe Embedding operates similarly to earcut [Eberly 2008], with the only difference that in this case the ears we cut from Ω do not form a single triangle but rather a fan of triangles that are all incident to the center of the ear. In this regard, the method has analogies with [Livesu 2020], where the author used earcut to produce an injective map, exploiting a similar mechanism for the preservation of convexity. However, this method only operates in a 2d-to-2d fashion and does not support surfaces embedded in a higher dimensional space.

Due to the preservation of convexity, to demonstrate injectivity we only need to prove that a single triangle strip S injectively maps to a convex portion of the target domain $\Omega_S \subset \Omega$. We focus here on the more general case in which the two vertex chains that bound S contain an arbitrary number of vertices, hence Ω_S is a convex rectangle. The proof naturally extends to degenerate strips containing a chain with only one vertex. In this case, the domain is triangular and not rectangular.

Let us assume that the strip S contains n vertices, distributed among its two lateral chains. If Ω_S was strictly convex, then any triplet of vertices would yield a valid triangle. Furthermore, any triplet that obeys to the same circular vertex ordering yields triangles with coherent orientation. In our case the embedding of S is only *weakly* convex, because the n vertices linearly sample two opposite sides of Ω_S. However, triangles in a strip always contain two subsequent vertices on a chain and one vertex on the opposite chain, meaning that triplets of colinear vertices cannot arise. Therefore, the triangulation properties for strictly convex polygons extend to weakly convex polygons too if the triangles to be inserted form a valid strip, ensuring that all triangles that we insert are non degenerate and share a coherent orientation.

4 DISCUSSION

We implemented Stripe Embedding in C++, using CinoLib [Livesu 2019] for mesh processing. A reference implementation is available at https://github.com/mlivesu/cinolib/blob/master/include/cinolib/stripe_embedding/stripe_embedding.h. Our tool supports a variety of different numerical models, both approximate and exact (Section 2.1). Embedded mesh coordinates can be expressed approximately, according to the IEEE 754 standard for double precision but also exactly, using GMP [Granlund 1996] rational numbers coupled with the CGAL kernel for lazy evaluation [Pion and Fabri 2011]. Additionally, we also support floating point numbers with arbitrary length for extended fixed precision, via MPFR [Fousse et al. 2007]. Note that this latter numerical model is not exact because precision must be fixed a priori, hence it cannot be sufficient for a specific mapping task (Table 1). In this section we report on numerous experiments that involve all these numeric types.

It should be noted that Stripe Embedding operates natively in the numerical model of choice, without requiring lossy conversions from higher precision numbers. If the numeric type is exact, then the algorithm is guaranteed to produce an injective map. Otherwise, the output map may not be fully injective. In terms of compatibility with downstream applications that require the use of standard IEEE 754 floating point numbers, Stripe Embedding performs similarly to Tutte. As detailed in Table 1, both methods fail to provide a fully injective map only in a marginal number of cases.

Figure 8 (second column) shows a typical embedding computed with our algorithm directly in floating point and then passed to a downstream application. As can be noticed the initial map contains a significant amount of geometric distortion. This is a common behavior for all methods in this class [Finnendahl et al. 2023; Livesu 2024; Shen et al. 2019; Tutte 1963] and is not to be considered a significant flaw. In fact, these algorithms are typically used to bootstrap low-distortion mapping methods with a valid initial solution, as demonstrated in the same image and discussed in Section 2. Indeed efficiency, injectivity and numerical robustness are the key fundamental properties that must be fulfilled.

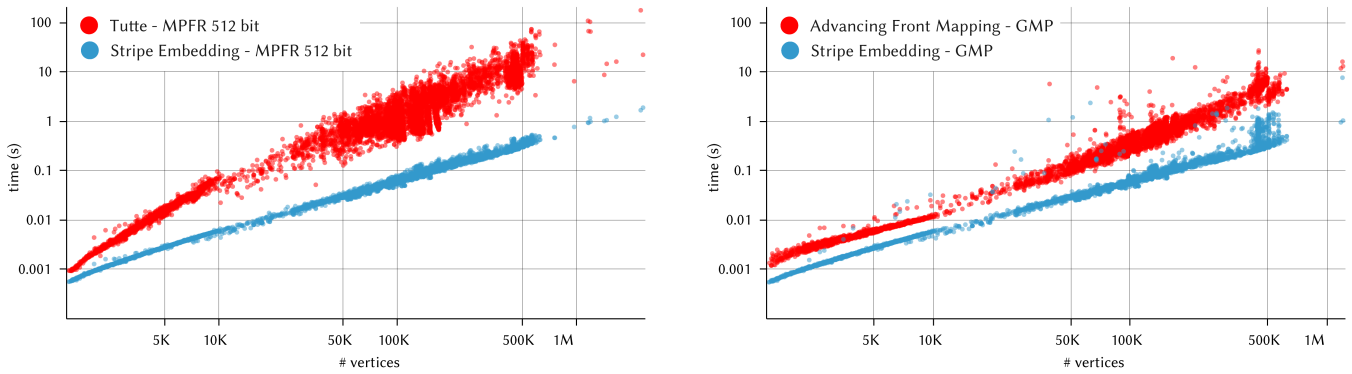


Fig. 9. Left: running time comparison between Stripe Embedding and Tutte on 12.1K mapping tasks, computed using floating points with 512 bits to represent the mantissa. Right: comparison between an exact rational implementation of Stripe Embedding and Advancing Front Mapping [Livesu 2024] on the same dataset. All tests were performed on a Mac Book M1 Pro with 32GB of RAM.

4.1 Large Scale Validation

We validated the robustness and performances of Stripe Embedding on a large variety of mapping tasks. Our testing dataset comprises both 2D and 3D meshes that come from different sources, exhibiting large variations in terms of geometry, connectivity and vertex count. We collected 1K meshes from the 2D Shape Structure Dataset [Carlier et al. 2016] and around 11K surface meshes that were used by the authors of [Chai et al. 2018; Liu et al. 2018] to validate their mapping methods. These two datasets together were already used in [Livesu 2024] for the validation of AFM. Additionally, we considered the 80 meshes from Thingi10K where Tutte is known to fail when implemented in floating point [Shen et al. 2019] and also a midpoint refinement of the same 80 meshes, which greatly challenge inexact numerical implementations. Overall, this amounts to 12.1K meshes.

Our baseline for comparative analysis is composed of Tutte [1963] and Advancing Front Mapping [Livesu 2024]. Both methods offer equivalent theoretical guarantees of correctness that, depending on the numerical model of choice, can more or less be effectively transferred to a software implementation. We do not consider Progressive Embedding [Shen et al. 2019] for direct comparative analysis because its reference floating point implementation is not robust enough (it fails on some of the refinements of the 80 meshes from Thingi10K) and is not compatible with exact numerical models. We also observed that on complex mapping tasks running Tutte in higher precision is much more efficient than bootstrapping PE with a non injective mapping produced with Tutte in floating point (Section 2.2). We do not consider E^3A [Finnendahl et al. 2023] either, because we are not aware of any available implementation, yet because E^3A only supports mappings to 3-sided domains, hence its scope is much narrower than ours and ultimately not compatible with our large scale dataset.

4.2 Comparison with Tutte

Computing the Tutte embedding amounts to solving a sparse linear system. We implemented it in Eigen [Guennebaud et al. 2010], using its built in Simplicial Cholesky solver for efficient LL^T matrix factorization. Note that Cholesky decomposition uses the squared root of matrix entries during pivot selection, making this code not compatible with rational numbers. We could consider richer numerical models that also support squared roots (Section 2.1) but they would introduce a theoretically unnecessary major slowdown. We therefore decided to use floating point numbers with extended precision, which are natively supported by Eigen (via MPFR [Fousse et al. 2007]). A similar choice for comparative analysis was also made in [Shen et al. 2019]. We did a few tests to find a suitable precision, considering 128, 256 and 512 bits to represent the mantissa. Tutte was able to provide an injective map for all models only at the highest precision (Table 1). We performed the same test also with our tool, which exhibited a comparable robustness for same numerical model and also achieved 100% success rate only at the highest precision. For performances, running times are reported in the left part of Figure 9. Both methods exhibit a linear behavior, although the slope of our results denotes a better scalability as mesh size grows. For meshes with more than 50K Tutte also exhibits larger variations, whereas the linear behavior of Stripe Embedding remains clear at all sizes. Overall, Stripe Embedding completed its task in 18.34 minutes, Tutte in 8.77 hours (28.71×). Stripe Embedding performed better than Tutte at all scales, completing the map in a shorter amount of time in the 100% of the cases. Speedups were in range in between 1.57× (for a mesh with 152 vertices) to 223.18×.

4.3 Comparison with AFM

We considered the reference implementation released by the author, which is contained in CinoLib [Livesu 2019]. Their software prototype only operates on exact rational numbers, which cannot be disabled. We therefore compare with AFM with respect to this numerical model. We used the tool with its default parameters, including a timeout threshold of 2 seconds for each advancing move. This is used to stop the execution in the presence of cascading issues with rational numbers, which is a declared limitation of AFM that can stretch running times up to a day of computation [Livesu 2024]. AFM stopped its execution due to timeout on 1449 models (12%), it produced a valid map in all the remaining cases. Timeouts almost entirely include the most challenging tasks in our dataset: 56 of

Table 2. Scalability comparison between Tutte [1963], AFM [Livesu 2024] and Stripe Embedding considering the IEEE 754 standard in double precision, MPFR floating points with 128 and 512 bits to encode the mantissa, and exact rational numbers. We report running times (in seconds) and slowdowns w.r.t. to a IEEE 754 implementation. For our method we also report speedups w.r.t. prior art for same numerical model. In the final line we report average speedups and slowdowns. Light green cells denote speedups by more than two orders of magnitude, darker green cells by more than three. Light red cells denote slowdowns by more than two orders of magnitude when Tutte switches to higher precision. All tests were executed on a Mac Book M1 Pro with 32GB RAM. This amount of memory was insufficient for big meshes in some configuration (fail entries). AFM also had timeout issues due to cascading of rational numbers (timeout entries).

	Tutte					AFM	Stripe Embedding										
	IEEE 754	MPFR 128 bit		MPFR 512 bit		Rationals	IEEE 754		MPFR 128 bit			MPFR 512 bit			Rationals		
# verts	s	s	slowdown w.r.t. IEEE 754	s	slowdown w.r.t. IEEE 754	s	s	speedup w.r.t. Tutte	s	speedup w.r.t. Tutte	slowdown w.r.t. IEEE 754	s	speedup w.r.t. Tutte	slowdown w.r.t. IEEE 754	s	speedup w.r.t. AFM	slowdown w.r.t. IEEE 754
25K	0.15	14.45	96.66×	21.68	145.09×	0.45	0.09	1.62×	0.12	119.78×	1.31×	0.13	164.79×	1.43×	0.13	3.56×	1.38×
50K	0.43	57.38	132.39×	75.08	173.24×	1.10	0.19	2.34×	0.25	233.60×	1.33×	0.27	280.95×	1.44×	0.26	4.26×	1.39×
75K	0.79	125.58	159.19×	140.64	178.29×	1.67	0.28	2.80×	0.37	339.27×	1.31×	0.41	343.65×	1.45×	0.39	4.28×	1.38×
100K	1.26	209.64	165.97×	240.51	190.40×	2.45	0.38	3.36×	0.49	422.43×	1.32×	0.54	446.07×	1.43×	0.53	4.67×	1.40×
250K	4.99	927.95	185.91×	1115.83	223.56×	8.78	0.94	5.29×	1.24	744.06×	1.32×	1.36	822.14×	1.44×	1.31	6.71×	1.39×
500K	16.04	2840.66	177.07×	3370.59	210.11×	**timeout**	1.91	8.42×	2.50	1136.15×	1.31×	2.72	1238.05×	1.43×	2.65		1.39×
750K	35.13					**timeout**	2.84	12.37×	3.76		1.32×	4.11		1.45×	4.00		1.41×
1M	52.72					**timeout**	3.80	13.89×	5.03		1.32×	5.45		1.44×	5.35		1.41×
2.5M	256.11					**timeout**	9.82	26.07×	12.87		1.31×	14.31		1.46×	13.82		1.41×
5M	874.25					**fail**	19.85	44.03×	26.59		1.34×	29.73		1.50×	**fail**		
7.5M	**fail**					**fail**	30.50		40.91		1.34×	45.38		1.49×	**fail**		
10M	**fail**					**fail**	39.78		53.10		1.33×	57.36		1.44×	**fail**		
			152.86×		186.78×			12.02×		499.22×	1.32×		549.28×	1.45×		4.70×	1.40×

the 80 challenging models highlighted in [Shen et al. 2019] and 69 of the 80 midpoint refinements of such models. Stripe Embedding succeeded in the 100% of the cases, without ever incurring into similar cascading issues. Based on our understanding, the different behavior of the two methods when implemented with rational numbers is intimately linked with the complexity of the geometric constructions they use. AFM embeds critical vertices by computing exact line-line intersections, which amounts to solve a 2×2 linear system. This potentially doubles the number of bits that are necessary to represent rationals, leading to an exponential growth in the rational representation when such constructions are cascaded. Conversely, in our linear sampling (Equation 1), we only multiply vertex coordinates with a parameter t whose complexity is almost constant (it depends on the chain size, which is in turn bounded by the local vertex valence), thus leading to linear growth in the rational representation.

In the right part of Figure 9 we show a log-log plot of running times. We restrict this part of the analysis to the 10655 meshes were AFM converged. Also in this case the two algorithms exhibit a clearly linear behavior. The line slope is also similar, indicating comparable performances in terms of scalability. Stripe Embedding terminated in 16.9 minutes, AFM in 2.57 hours (9.12×). AFM was occasionally faster than SE, in 17 out of 10655 cases. In the worst case Stripe Embedding was 6.59× slower than AFM. In the best case it was 260.72× faster.

4.4 Scalability

We evaluate the scalability of ours and the other algorithms on meshes with growing size. This test is similar to the one proposed in [Finnendahl et al. 2023], but we consider more methods and a variety of alternative numerical models, both exact and approximate. We randomly sampled the unit square $[0, 1]^2$ at different densities

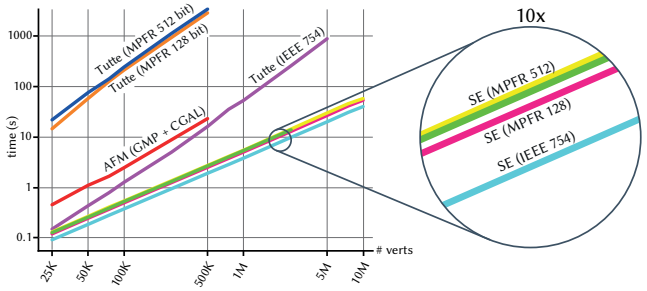


Fig. 10. Log-log plot of the scalability test reported in Table 2. Switching to higher precision increases Tutte's runtime by more than two orders of magnitude. Stripe Embedding reports only marginal slowdowns (closeup). The green line in the closeup refers to the performances of Stripe Embedding with exact rationals.

and computed a sequence of Delaunay triangulations that we passed in input to SE, Tutte [1963] and AFM [Livesu 2024] to compute planar maps to the unit circle. Mesh size spans from 25K to 10M vertices. For Tutte we considered three alternative types of numbers: IEEE 754 floats in double precision and MPFR with 128 and 512 bits to encode the mantissa. For AFM we considered exact rational numbers. For Stripe Embedding we considered all available numeric models.

Runtimes are reported in Table 2 and a log-log plot of the same data can be found in Figure 10. Physiological fluctuations are counterbalanced by always considering the best out of three runs. All methods clearly exhibit a linear asymptotic behavior, although the slope of Stripe Embedding lines yield growing efficiencies as the mesh size grows.

Stripe Embedding resulted faster than Tutte and AFM at all scales with any numerical model. It resulted faster than floating point implementations of prior art even when implemented with exact

rationals. Stripe Embedding is 2.5 orders of magnitude faster than Tutte implemented with MPFR on average, reaching speed up factors that go well beyond 1000× for meshes with 500K vertices.

It may be argued that the poor scalability performances of Tutte are due to the choice of the LL^T solver (Section 4.2) and not to an inherent deficiency of the method. As a sanity check, we also performed a few experiments with an iterative solver, which is typically better suited for larger mesh sizes. Specifically, we considered the mesh with 5M vertices and computed the Tutte embedding using the Eigen's BiCGSTAB iterative solver with row-major matrix encoding and a diagonal preconditioner, as suggested in the Eigen documentation. Indeed, memory-wise BiCGSTAB is more efficient (9GB of RAM vs 15.2GB for LL^T), but time-wise LL^T was still faster, completing the map in half the time required by BiCGSTAB. In principle, one could also implement simpler iterative schemes such as Jacobi or Gauss-Seidel and even stop the solver as soon as the map is injective. However, in such a case the overhead introduced by exactly checking injectivity after each iteration would cause major slowdowns, not to mention the fact that in challenging cases stopping the solver prematurely does not work, as even fully solving the system may not provide an injective map (Table 1).

The other important aspect that emerges from this test is the ability of Stripe Embedding to switch from lower to higher precision numbers without almost increasing its running time. Specifically, considering a IEEE 754 implementation as the baseline, Stripe Embedding runs 1.32× slower when implemented with MPFR 128, 1.45× slower when implemented with MPFR 512, and only 1.40× slower when implemented with exact rational numbers. This behavior highly contrasts with typical geometric algorithms, which tend to increase their running times by two or more orders of magnitude when implemented in extended precision, as witnessed by the performances of Tutte. We conjecture that this nice property is fully determined by the simplicity of our arithmetic construction, which is solely based on simple linear interpolation.

In terms of numerical robustness, Tutte and Stripe Embedding never introduced flipped elements in their embeddings, not even when using IEEE 754 floating points. Advancing Front Mapping never introduced flipped elements when implemented with rational numbers, but failed to produce an injective map in floating point at all scales.

Due to the limited amount of memory of our testing hardware (32 GB) some methods failed at the highest resolutions. This happened to Tutte already in double precision, for meshes having 7.5 and 10M vertices. Implementations based on rational numbers (AFM and SE) both failed on meshes with 5M vertices or more. Besides that, Stripe Embedding successfully completed all mapping tasks.

4.5 Initialization for Low Distortion Mapping

To evaluate the practical usefulness of our algorithm we implemented a pipeline for the generation of low distortion uv maps, minimizing the Symmetric Dirichlet energy with the implementation of SLIM [Rabinovich et al. 2017] available in libIGL [Jacobson et al. 2013], which we bootstrapped with Tutte and with our embeddings. To make the comparison fair, we always executed Tutte

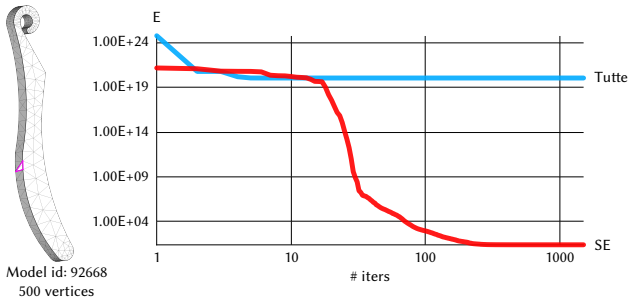


Fig. 11. Embedding an entire mesh to a single triangle yields a highly distorted map that is difficult to optimize. Here we plot the Symmetric Dirichlet energy minimized with SLIM [Rabinovich et al. 2017] warm started with Tutte and Stripe Embedding. When initialized with Tutte, SLIM gets stuck at energy 1.08366E+20 after six iterations and the distortion never decreases (we continued for 3K iterations). When initialized with SE, SLIM is able to effectively decrease the energy up to a final distortion of 25.5127.

on the (possibly) refined mesh produced by SE, ensuring that both methods operate on the same amount of degrees of freedom.

We first observe that on *easy* cases Tutte performs better than Stripe Embedding. We considered the 1237 meshes of the 2D Shape Structure Dataset [Carlier et al. 2016]. Since these are planar meshes, we expected the output Symmetric Dirichlet energy to reach its minimum, which is 4 in 2D. Indeed, all output energies were below $4+10^{-5}$ for both methods, but SLIM converged faster when initialized with Tutte on the majority of cases (1055 out ot 1237), using roughly half of the iterations on average. This is because Tutte tends to produce an initial solution with lower geometric distortion, which is closer to the final result. We did not observe significant differences in the cost per iteration, therefore also running times are roughly halved due to a faster convergence.

Moving to more challenging tasks, Stripe Embedding consistently provides better initial solutions. We considered the 80 challenging meshes isolated in [Shen et al. 2019] and also their midpoint refinements, for a total of 160 mapping tasks. Stripe Embedding produced a valid initial solution in 44 cases, Tutte in 15. We used the 15 cases in which both methods produced a valid floating point embedding to bootstrap SLIM, setting the maximum number of iterations to 3000. In all cases SLIM provided solutions with lower geometric distortion when initialized with Stripe Embedding. Differently from the previous experiment, where the final results were identical regardless of the initial guess, in this case the output maps were heavily affected by the starting solution. In particular, we observed that SLIM tends to get stuck when optimizing the initial embedding provided by Tutte, likely due to precision issues with the floating point system. A representative example of this failure is shown in Figure 11. SE resulted superior than Tutte also in terms of cost per iteration. Overall, for the same number of iterations, SE+SLIM resulted 50% faster than Tutte+SLIM on average.

4.6 Alternative Stripes

Our choice to rely on strips of minimal length is rewarding in terms of maximum efficiency and reduced cost for sanity checks. However,

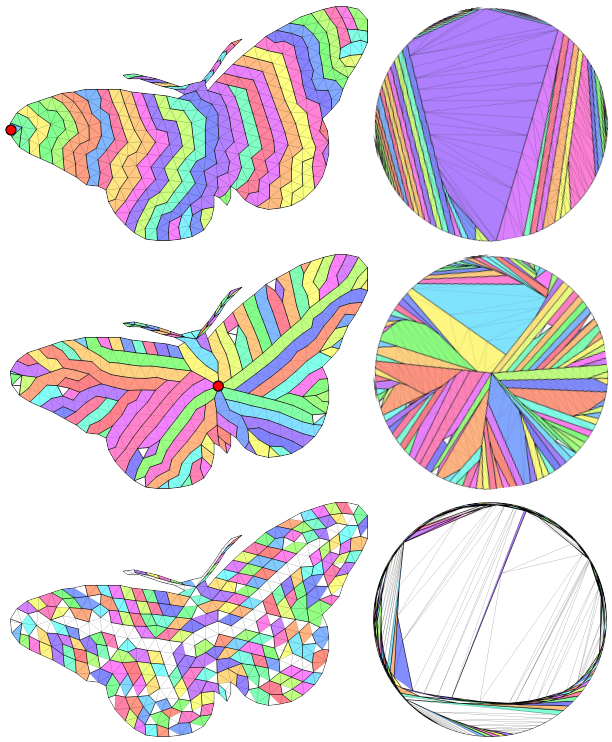


Fig. 12. Three alternative embeddings obtained by computing stripes according to different policies. From top to bottom: parallel expansion, spanning tree, our expansion. Red circles indicate the seeds used for the parallel and tree expansions.

any stripification that fulfills the requirements in Section 3.2 is guaranteed to produce a valid embedding (Section 3.5). In this regard the idea at the core of this paper is representative of a *family* of algorithms, which can greatly differ from one another in terms of output results. The space of alternative stripifications grows exponentially with the mesh size. Besides minimal stripes incident to a single vertex, we also explored alternative decompositions, expanding parallel fronts from an initial boundary seed vertex and growing a spanning tree from an inner node. As shown in Figure 12 these alternative versions of Stripe Embedding produce remarkably different maps, also suggesting a superior potential in terms of mesh distortion and floating point robustness, due to a minor presence of almost degenerate triangles. On the negative side, we observed that the parallel expansion introduces a substantial overhead due to the need of explicit sanity checks to ensure strip validity, also making the implementation cumbersome. For the spanning tree propagation, we observed that in complex cases it may introduce illegal strips that wind around a mesh vertex (first column in Figure 3), which may not be easy to sanitize. We attempted to fix such strips with a local heuristic and tested the resulting algorithm on our large scale benchmark. To give numbers, we failed to compute a valid stripification in 164 cases, but we were also able to obtain an injective floating point map in 11920 cases out of 11940, reducing the floating point failure rate from 1.7% to 0.17%. All in all, we believe that our current solution is just scratching the surface of the problem and the quest for alternative stripifications may produce even better results than the ones showcased in this article.

4.7 Impact of Mesh Refinement

As mentioned in Section 3.3, Stripe Embedding ensures the existence of a valid stripification by splitting all non contractible edges in preprocessing. We show here that the impact of this step in the output mesh size is negligible.

For the 12.1K models in the large scale benchmark, Stripe Embedding split at least one edge in 7414 meshes (61.67%). The impact of this refinement was in between 1% and 2% of the input mesh size for 12 models, in between 2% and 5% for 3 models, and was 11.5% for only one model (234 vertices, 34 edges split). For all the remaining cases the impact was below 1%. In absolute numbers, in the two worst cases Stripe Embedding split 383 edges, on two meshes having 39K vertices.

We emphasize that our large scale benchmark contains many meshes with poor connectivity, hence it does not truly reflect the topological properties of an average good mesh (e.g. Delaunay). Indeed, if we consider the sequence of Delaunay meshes that we used for the scalability test, Stripe Embedding split only 3 edges in the whole dataset, one on a mesh with 2.5M vertices, one on a mesh with 5M vertices and one on a mesh with 7.5M vertices. In all the other meshes the output connectivity was identical to the input one.

4.8 Parallelization

Even though the software we used for our experiments is single threaded, Stripe Embedding is highly compatible with a parallel implementation. Assuming to perform computation on n cores, one could trivially split the mesh boundary into n chunks and grow regions of roughly equal size that jointly cover the entire surface. Once the chains of edges that separate the various clusters have been embedded along straight lines, the map for all clusters can be completed in parallel, without the need for any synchronization. To roughly estimate the gain that this strategy yields, we launched Stripe Embedding in parallel on 8 meshes with nK vertices on our 8-core testing hardware and compared the running time with Stripe Embedding executed on a single mesh of $8n$K vertices. We repeated this experiment for $n = 5, 10, 50, 100$ and estimated an average speed up of 5.24×. Switching to a GPU implementation would likely greatly improve our performances.

5 CONCLUSIONS

We have presented a novel paradigm for the robust computation of planar maps which is based on the idea to decompose a mesh into a sequence of triangle strips that can be embedded by means of simple linear interpolation. The article shows minimal conditions that triangle strips must fulfill to enable such a mapping and proposes a local stripification strategy that maximizes efficiency.

Results show that Stripe Embedding matches prior methods in terms of numerical robustness, but is up to three orders of magnitude faster and allows to almost seamlessly switch from floating point to numeric types with higher precision. Based on these evidences, we

believe that Stripe Embedding is practically superior than any alternative existing method for the computation of boundary constrained mappings, for which is a convenient drop-in replacement.

Besides the obvious future works, such as providing a proper parallel implementation (Section 4.8), we believe that this paper defines a new *family* of embeddings, opening the doors to interesting avenues for further research. In particular, the space of valid stripe decompositions is exponentially big and largely unexplored. We plan to continue the preliminary experimentation discussed in Section 4.6 to devise smarter stripification strategies that optimally combine efficiency, numerical robustness and geometric distortion. As discussed in Sections 4.5 and 4.6 this positively impacts both floating point robustness and convergence speed when computing low distortion uv maps.

In a broader perspective, we foresee two additional interesting open related questions to address. The first one regards the ability to bridge fully robust mapping software with downstream applications. When not implemented in floating point the output embeddings are not directly usable by third party applications, which hardly support multi-precision arithmetic. Snapping the output mesh coordinates to the 2D floating point grid is a solved problem for the case of a single mesh [Halperin and Packer 2002], but existing methods collapse mesh elements, hence they would destroy an underlying map. Devising some *map-aware* snap rounding algorithm would allow to reliably transfer the results of exact map computation to third party software, completing a fully robust end-to-end pipeline in the spirit of Exact Numeric Computation [Yap and Dubé 1995].

The second interesting question regards the possibility to bring the stripe embedding intuition one dimension up, devising similar strategies to embed a tetrahedral mesh into a convex polyhedron. This operation is known to be not possible in the general case for Tutte [Alexa 2023] and, more in general, the fully robust computation of injective volume maps is a scientific open problem [Hinderink and Campen 2023; Nigolian et al. 2023] with important applications in graphics and engineering [Pietroni et al. 2023] (§8.2).

ACKNOWLEDGMENTS

Funded by the European Union - Next Generation EU and by the Ministry of University and Research (MUR), National Recovery and Resilience Plan (NRRP), Mission 4, Component 2, Investment 1.5, project "RAISE - Robotics and AI for Socio-economic Empowerment" (ECS00000035). All Authors are part of RAISE Innovation Ecosystem.

REFERENCES

Marc Alexa. 2023. Tutte embeddings of tetrahedral meshes. *Discrete & Computational Geometry* (2023), 1–11.

Marco Attene. 2020. Indirect predicates for geometric constructions. *Computer-Aided Design* 126 (2020), 102856.

Christoph Burnikel, Kurt Mehlhorn, and Stefan Schirra. 1996. The LEDA class real number. (1996).

Marcel Campen, Ryan Capouellez, Hanxiao Shen, Leyi Zhu, Daniele Panozzo, and Denis Zorin. 2021. Efficient and robust discrete conformal equivalence with boundary. *ACM Transactions on Graphics (TOG)* 40, 6 (2021), 1–16.

Marcel Campen, Cláudio T Silva, and Denis Zorin. 2016. Bijective maps from simplicial foliations. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–15.

Axel Carlier, Kathryn Leonard, Stefanie Hahmann, Geraldine Morin, and Misha Collins. 2016. The 2d shape structure dataset: A user annotated open access database. *Computers & Graphics* 58 (2016), 23–30.

Shuangming Chai, Xiao-Ming Fu, Xin Hu, Yang Yang, and Ligang Liu. 2018. Sphere-based cut construction for planar parameterizations. *Computers & Graphics* 74 (2018), 66–75.

Gianmarco Cherchi, Marco Livesu, Riccardo Scateni, and Marco Attene. 2020. Fast and robust mesh arrangements using floating-point arithmetic. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–16.

Gianmarco Cherchi, Fabio Pellacini, Marco Attene, and Marco Livesu. 2022. Interactive and Robust Mesh Booleans. *ACM Transactions on Graphics (TOG)* 41, 6 (2022), 1–14.

Olivier Devillers, Sylvain Lazard, and William J Lenhart. 2018. 3D Snap Rounding. In *34th International Symposium on Computational Geometry (SoCG 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Tamal Dey, Herbert Edelsbrunner, Sumanta Guha, and Dmitry Nekhayev. 1999. Topology preserving edge contraction. *Publications de l'Institut Mathématique* 66 (1999).

Lorenzo Diazzi and Marco Attene. 2021. Convex polyhedral meshing for robust solid modeling. *ACM Transactions on Graphics (TOG)* 40, 6 (2021), 1–16.

Lorenzo Diazzi, Daniele Panozzo, Amir Vaxman, and Marco Attene. 2023. Constrained Delaunay Tetrahedrization: A Robust and Practical Approach. *ACM Transactions on Graphics (TOG)* 42, 6 (2023), 1–15.

Xingyi Du, Danny M. Kaufman, Qingnan Zhou, Shahar Z. Kovalsky, Yajie Yan, Noam Aigerman, and Tao Ju. 2021. Optimizing Global Injectivity for Constrained Parameterization. *ACM Trans. Graph.* 40, 6, Article 260 (2021), 18 pages. https://doi.org/10.1145/3478513.3480556

David Eberly. 2008. Triangulation by ear clipping. *Geometric Tools* (2008), 2002–2005.

Xianzhong Fang, Mathieu Desbrun, Hujun Bao, and Jin Huang. 2022. Topocut: fast and robust planar cutting of arbitrary domains. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–15.

Guy Fargion and Ofir Weber. 2022. Globally injective flattening via a reduced harmonic subspace. *ACM Transactions on Graphics (TOG)* 41, 6 (2022), 1–17.

Ugo Finnendahl, Dimitrios Bogiokas, Pablo Robles Cervantes, and Marc Alexa. 2023. Efficient Embeddings in Exact Arithmetic. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–17.

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

Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. 2007. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software (TOMS)* 33, 2 (2007), 13–es.

Xiao-Ming Fu and Yang Liu. 2016. Computing inversion-free mappings by simplex assembly. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 1–12.

Xiao-Ming Fu, Jian-Ping Su, Zheng-Yu Zhao, Qing Fang, Chunyang Ye, and Ligang Liu. 2021. Inversion-free geometric mapping construction: A survey. *Computational Visual Media* 7 (2021), 289–318.

Mark Gillespie, Boris Springborn, and Keenan Crane. 2021. Discrete conformal equivalence of polyhedral surfaces. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–20.

Torbjörn Granlund. 1996. Gnu mp. *The GNU Multiple Precision Arithmetic Library* 2, 2 (1996).

Gaël Guennebaud, Benoit Jacob, et al. 2010. Eigen. *URl: http://eigen. tuxfamily. org* 3, 1 (2010).

Jia-Peng Guo, Wen-Xiang Zhang, Chunyang Ye, and Xiao-Ming Fu. 2023. Robust Coarse Cage Construction with Small Approximation Errors. *IEEE Transactions on Visualization and Computer Graphics* (2023).

Dan Halperin and Eli Packer. 2002. Iterated snap rounding. *Computational Geometry* 23, 2 (2002), 209–225.

Steffen Hinderink and Marcel Campen. 2023. Galaxy maps: Localized foliations for bijective volumetric mapping. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–16.

Hugues Hoppe. 1999. Optimization of mesh locality for transparent vertex caching. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. 269–276.

Kai Hormann, Konrad Polthier, and Alla Sheffer. 2008. Mesh parameterization: theory and practice. In *ACM SIGGRAPH ASIA 2008 courses*. 1–87.

Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2018. Tetrahedral meshing in the wild. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–14.

Alec Jacobson, Daniele Panozzo, C Schüller, Olga Diamanti, Qingnan Zhou, N Pietroni, et al. 2013. libigl: A simple C++ geometry processing library. *Google Scholar* (2013).

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

Vijay Karamcheti, Chen Li, Igor Pechtchanski, and Chee Yap. 1999. A core library for robust numeric and geometric computation. In *Proceedings of the fifteenth annual symposium on Computational geometry*. 351–359.

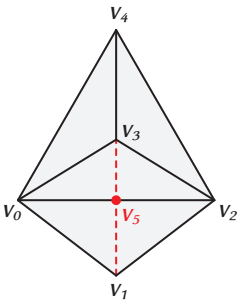
Lutz Kettner, Kurt Mehlhorn, Sylvain Pion, Stefan Schirra, and Chee Yap. 2008. Classroom examples of robustness problems in geometric computations. *Computational Geometry* 40, 1 (2008), 61–78.

Payam Khanteimouri, Manish Mandad, and Marcel Campen. 2022. Rational Bézier Guarding. In *Computer Graphics Forum*, Vol. 41. Wiley Online Library, 89–99.

Vladislav Kraevoy and Alla Sheffer. 2004. Cross-parameterization and compatible remeshing of 3D models. *ACM Transactions on Graphics (ToG)* 23, 3 (2004), 861–869.
Bruno Lévy. 2016. Robustness and efficiency of geometric programs: The Predicate Construction Kit (PCK). *Computer-Aided Design* 72 (2016), 3–12.
Bruno Lévy. 2024. Exact predicates, exact constructions and combinatorics for mesh CSG. *arXiv preprint arXiv:2405.12949* (2024).
Ligang Liu, Chunyang Ye, Ruiqi Ni, and Xiao-Ming Fu. 2018. Progressive parameterizations. *ACM Trans. Graph.* 37, 4 (2018), 41.
Marco Livesu. 2019. cinolib: a generic programming header only C++ library for processing polygonal and polyhedral meshes. *Transactions on Computational Science XXXIV* (2019), 64–76.
Marco Livesu. 2020. A Mesh Generation Perspective on Robust Mappings. In *Smart Tools and Applications in Graphics (STAG)*. The Eurographics Association. https://doi.org/10.2312/stag.20201234
Marco Livesu. 2021. Scalable Mesh Refinement for Canonical Polygonal Schemas of Extremely High Genus Shapes. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 27, 1 (2021), 254–260. https://doi.org/10.1109/TVCG.2020.3010736
Marco Livesu. 2023. Towards a robust and portable pipeline for quad meshing: Topological initialization of injective integer grid maps. *Computers & Graphics* 112 (2023), 50–59.
Marco Livesu. 2024. Advancing Front Surface Mapping. *Computer Graphics Forum* (2024). https://doi.org/10.1111/cgf.15026
Gary H Meisters. 1975. Polygons have ears. *The American Mathematical Monthly* 82, 6 (1975), 648–651.
Alexander Naitsat, Gregory Naitzat, and Yehoshua Y Zeevi. 2021. On inversion-free mapping and distortion minimization. *Journal of Mathematical Imaging and Vision* 63 (2021), 974–1009.
Julius Nehring-Wirxel, Philip Trettner, and Leif Kobbelt. 2021. Fast exact booleans for iterated CSG using octree-embedded BSPs. *Computer-Aided Design* 135 (2021), 103015.
Valentin Zénon Nigolian, Marcel Campen, and David Bommes. 2023. Expansion cones: A progressive volumetric mapping framework. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–19.
Nico Pietroni, Marcel Campen, Alla Sheffer, Gianmarco Cherchi, David Bommes, Xifeng Gao, Riccardo Scateni, Franck Ledoux, Jean-Françoise Remacle, and Marco Livesu. 2023. Hex-Mesh Generation and Processing: a Survey. *ACM Transactions on Graphics* 42 (2023), 1–44. Issue 2. https://doi.org/10.1145/3554920
Sylvain Pion and Andreas Fabri. 2011. A generic lazy evaluation scheme for exact geometric computations. *Science of Computer Programming* 76, 4 (2011), 307–323.
M Porcu and Riccardo Scateni. 2006. Partitioning Meshes into Strips using the Enhanced Tunnelling Algorithm. In *Proceedings of VriPhys' 06*. Eurographics, 61–70.
Michael Rabinovich, Roi Poranne, Daniele Panozzo, and Olga Sorkine-Hornung. 2017. Scalable locally injective mappings. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1.
Jarek Rossignac. 1999. Edgebreaker: Connectivity compression for triangle meshes. *IEEE transactions on visualization and computer graphics* 5, 1 (1999), 47–61.
John Schreiner, Arul Asirvatham, Emil Praun, and Hugues Hoppe. 2004. Inter-surface mapping. In *ACM SIGGRAPH 2004 Papers*. 870–877.
Hanxiao Shen, Zhongshi Jiang, Denis Zorin, and Daniele Panozzo. 2019. Progressive embedding. *ACM Transactions on Graphics* 38, 4 (2019).
Jonathan Richard Shewchuk. 1997. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry* 18 (1997), 305–363.
Hang Si and Klaus Gärtner. 2005. Meshing piecewise linear complexes by constrained Delaunay tetrahedralizations. In *Proceedings of the 14th international meshing roundtable*. Springer, 147–163.
Jason Smith and Scott Schaefer. 2015. Bijective parameterization with free boundaries. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 1–9.
Jian-Ping Su, Chunyang Ye, Ligang Liu, and Xiao-Ming Fu. 2020. Efficient bijective parameterizations. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 111–1.
Gabriel Taubin and Jarek Rossignac. 1998. Geometric compression through topological surgery. *ACM Transactions on Graphics (TOG)* 17, 2 (1998), 84–115.
Philip Trettner, Julius Nehring-Wirxel, and Leif Kobbelt. 2022. EMBER: exact mesh booleans via efficient & robust local arrangements. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–15.
William Thomas Tutte. 1963. How to draw a graph. *Proceedings of the London Mathematical Society* 3, 1 (1963), 743–767.
Qi Wang, Wen-Xiang Zhang, Yuan-Yuan Cheng, Ligang Liu, and Xiao-Ming Fu. 2023. Practical construction of globally injective parameterizations with positional constraints. *Computational Visual Media* 9, 2 (2023), 265–277.
Ofir Weber and Denis Zorin. 2014. Locally injective parametrization with arbitrary fixed boundaries. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 1–12.
Jinlin Yang, Shibo Liu, Shuangming Chai, Ligang Liu, and Xiao-Ming Fu. 2022. Precise High-order Meshing of 2D Domains with Rational Bézier Curves. In *Computer Graphics Forum*, Vol. 41. Wiley Online Library, 79–88.
Chee Yap and Thomas Dubé. 1995. The exact computation paradigm. In *Computing in Euclidean Geometry*. World Scientific, 452–492.
Chunyang Ye, Jian-Ping Su, Ligang Liu, and Xiao-Ming Fu. 2020. Memory-Efficient Bijective Parameterizations of Very-Large-Scale Models. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 1–12.
Qingnan Zhou, Eitan Grinspun, Denis Zorin, and Alec Jacobson. 2016. Mesh arrangements for solid geometry. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–15.

A CONVERGENCE OF THE EDGE SPLITTING

We prove that splitting all non contractible edges in a mesh always converges in a finite number of steps. To do so, we demonstrate that splitting a non contractible edge does not introduce new illegal edges, meaning that the set of non contractible edges reduces by one unit after each split, eventually vanishing because its size is bounded from above by the total number of mesh edges.



Let us consider the edge e_{02} in the figure aside. This edge is non contractible because it violates the link condition in Eq. (2). In fact, vertex v_4 and edge e_{34} are in the link of vertices v_0 and v_2 but not in the link of edge e_{02}, therefore

$$\text{Link}(v_0) \cap \text{Link}(v_2) \neq \text{Link}(e_{02}).$$

Contractibility can be locally restored by splitting e_{02} at vertex v_5. This operation deletes edge e_{02} from the mesh, inserting vertex v_5 and four new edges, $e_{05}, e_{15}, e_{25}, e_{35}$, incident to it. As a result, the link of v_5 is the quadrilateral perimeter of the two triangles incident to the split edge e_{02}, that is

$$\text{Link}(v_5) = \{v_0, v_1, v_2, v_3, e_{01}, e_{12}, e_{23}, e_{03}\}.$$

We first observe that only the edges in $\text{Link}(v_5)$ can become non contractible. This is because the split vertex v_5 will only appear in the link of such edges. To prove that any other mesh edge cannot be affected by this local change of connectivity it is sufficient to observe that: (i) such edge would not have v_5 in its link; (ii) at least one of its endpoints would not have v_5 in its link either, therefore the intersection between the two vertex links would not yield v_5. Combining these two points we obtain that the link condition $\text{Link}(v_i) \cap \text{Link}(v_j) = \text{Link}(e_{ij})$ holds for any edge $e_{ij} \notin \text{Link}(v_5)$ that was contractible prior to splitting e_{02}.

Let us now focus on the elements in $\text{Link}(v_5)$. The links of its mesh elements are as follows

$$\begin{aligned}
\text{Link}(v_0) &= \{v_1, v_5, v_3, v_4, e_{15}, e_{35}, e_{34}\} \cup O_0 \\
\text{Link}(v_1) &= \{v_0, v_5, v_2, e_{05}, e_{25}\} \cup O_1 \\
\text{Link}(v_2) &= \{v_1, v_5, v_3, v_4, e_{15}, e_{35}, e_{34}\} \cup O_2 \\
\text{Link}(v_3) &= \{v_0, v_5, v_2, v_4, e_{05}, e_{25}, e_{24}, e_{04}\}
\end{aligned}$$

$$\begin{aligned}
\text{Link}(e_{01}) &= \{v_5\} \cup O_{01} \\
\text{Link}(e_{12}) &= \{v_5\} \cup O_{12} \\
\text{Link}(e_{23}) &= \text{Link}(e_{03}) = \{v_4, v_5\}.
\end{aligned}$$

Symbols O_i, O_{ij} indicate additional (or *Other*) mesh elements that are not depicted in the figure but may be in the link of vertex v_i or edge e_{ij}. Applying the link condition to edges e_{03}, e_{23} we obtain

$$\begin{aligned} \text{Link}(v_0) \cap \text{Link}(v_3) &= \\ \text{Link}(e_{03}) &= \\ \text{Link}(e_{23}) &= \\ \text{Link}(v_2) \cap \text{Link}(v_3) &= \{v_4, v_5\}\,. \end{aligned}$$

Instead, for edges e_{01}, e_{12} we obtain

$$\begin{aligned} \text{Link}(v_0) \cap \text{Link}(v_1) &= \{v_5\} \cup (O_0 \cap O_1) \\ \text{Link}(v_1) \cap \text{Link}(v_2) &= \{v_5\} \cup (O_1 \cap O_2). \end{aligned}$$

It follows that the the link conditions

$$\begin{aligned} \text{Link}(v_0) \cap \text{Lin k}(v_1) &= \text{Link}(e_{01}) \\ \text{Link}(v_1) \cap \text{Link}(v_2) &= \text{Link}(e_{12}) \end{aligned}$$

are satisfied if and only if

$$\begin{aligned} O_0 \cap O_1 &= O_{01} \\ O_1 \cap O_2 &= O_{12}. \end{aligned}$$

But if edges e_{01}, e_{12} were contractible prior to splitting edge e_{02}, then these conditions are surely satisfied. In fact, elements in O_0, O_1, O_2, O_{01}, O_{12} already belonged to the links of these mesh elements and, therefore, already contributed to satisfy the link condition. Consequently, e_{01}, e_{02} can only be non contractible if they already were prior to splitting edge e_{02}. □