Contents

Volume 0 (1981), Number 0 pp. 1–17 COMPUTER GRAPHICS *forum*

Seamless Parametrization of Spheres with Controlled Singularities

Zohar Levi

Victoria University of Wellington, New Zealand

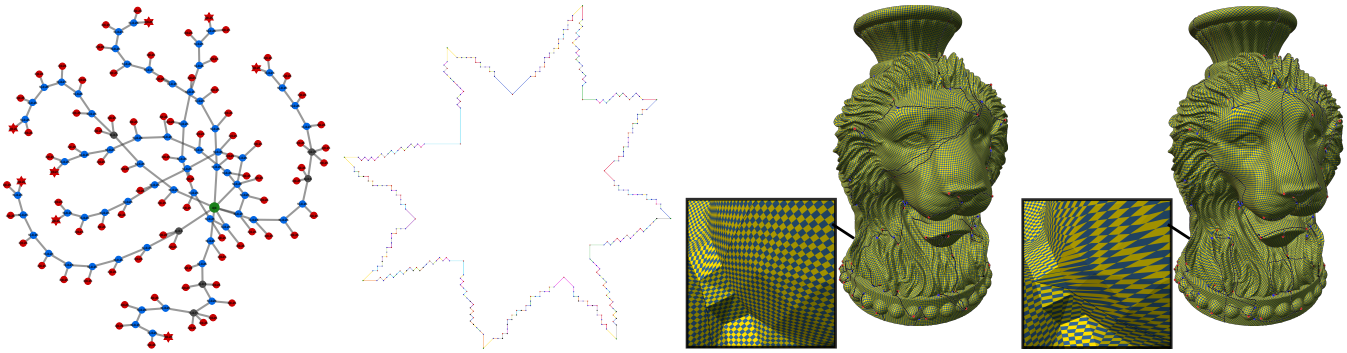


Figure 1: *From left to right: the (uncut) seam graph, the domain boundary polygon, our final result, and [MPZ14]. Our result contains a modest amount of refinement of 2%. The optimized symmetric Dirichlet energy is 4.24. [MPZ14] result contains 53% refinement; due to nearly collapsed triangles, the mapping posed a problem for the optimizer, and the symmetric Dirichlet energy is 111,000, which we consider a failure. See table 2 for more details. See section 3.1 for figure notations.*

Abstract

We present a method for constructing seamless parametrization for genus-0 surfaces, which can handle any feasible cone configuration, thus allowing users to arbitrarily design and tailor a mapping as desired. The method directly constructs a self-overlapping metapolygon of the domain boundary of the mapped cut mesh, which relieves the need of using an auxiliary surface. This simplifies the pipeline and allows for a necessary optimization of the boundary polygon before mapping the interior. Moreover, it enables handling larger meshes with more cones than previous methods can handle.

Our construction is purely combinatorial, and it guarantees that the mapping is locally injective—a prerequisite to today's advanced optimization methods. This is achieved via careful construction of a simple domain boundary polygon, where existence of such a polygon is proven for all cases. We offer a numerically robust algorithm to automate the construction, which involves a solution of two linear problems. We offer a full pipeline, suggesting elegant solutions to sub-problems, and demonstrate robustness through extensive experiments.

CCS Concepts

• ***Computing methodologies*** → ***Mesh geometry models;***

1. Introduction

Surface parametrization is a central operation in geometry processing. With additional constraints, a parametrization can be made seamless, which is useful for quad meshing. The quality of a seamless mapping determines the quality of the quad mesh that is generated from it. One important requirement from a mapping is local injectivity.

A seamless parametrization induces a flat metric over the surface, where curvature vanishes all over the surface except for a few points, the cone singularities. Cones can be the result of an optimization process that utilizes them to lower the amount of distortion, or they can be a result of careful design. In this paper, we address the problem of seamless parametrization that preserves given cone singularities. For further motivation see appendix A.

In recent years, mapping optimization methods were developed to reduce distortion and guarantee local injectivity.

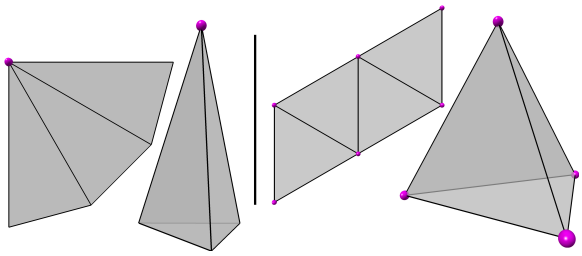


Figure 2: *Isomeric seamless parametrization of primitives, which requires low valence cones. Left: an open cone shape with a single 1-cone. Right: a regular tetrahedron with four 2-cones.*

Since they use non-convex energies, all these methods require an initial feasible point to start the optimization from (otherwise, these barrier energies would be infinite). For an unconstrained parametrization, a Tutte mapping to a convex domain is a common tool for generating an initial locally injective mapping. However, for seamless parametrization, this initial mapping does not satisfy the seamlessness constraints and is not applicable. Generating an initial mapping that satisfies seamlessness constraints—preserving parametric cone angles—and is guaranteed to be locally injective is a challenging task, which we address in this paper. The core part of our method generates a simple domain boundary polygon, which is trivially *weakly self-overlapping*—a requirement for a locally injective mapping [WZ14].

We offer a novel approach to solve the problem, which is arguably simpler than previous approaches (see discussion in the end of section G.3). Unlike previous methods that construct an auxiliary surface, we work directly with the input mesh. This, for one, relieves the need to map the cones between an auxiliary mesh and the input surface (see appendix H). Besides simplifying the pipeline, our approach has a crucial advantage. Our generated domain boundary polygon is constructed as a metagraph with complexity that depends on the number of cones rather than the auxiliary mesh complexity (which can be significantly larger than the input mesh). The size of the boundary polygon affects the run time, robustness, and amount of refinement of subsequent steps in the pipeline, such as the important boundary polygon optimization (appendix D) and mapping the interior of the cut mesh (appendix E).

Our construction guarantees local injectivity and is combinatorial: it is mesh-independent, and the domain polygon is purely based on the set of input cones. We offer a numerically robust algorithm to automate the construction, involving a solution of two linear problems, which is arguably simpler and easier to implement than previous work. We handle 1-cones (and 2-cones) in a uniform way, which were ignored in previous work due to added complexity. These two types of cones are equally important in reducing mapping distortion since they fit better in high curvature places, even for the simplest models, e.g. fig. 2.

We offer a complete pipeline, suggesting novel solutions to independent related sub-problems, which contribute to the robustness of our method. We demonstrate robustness through comprehensive benchmark tests that include challenges that state-of-the-art fail to handle.

Our method currently handles only sphere topology shapes. The new machinery will be employed in future work, aiming to solve the problem for higher genus surfaces. These surfaces have homology generator (or noncontractible) loops, where prescribing their holonomy angles is no less important than preserving cone angles and is still an open problem (section 11).

1.1. Contribution Summary

- A new construction of a simple domain polygon for a given set of cones. Properties of the construction:
  - The polygon is constructed directly, as opposed to previous methods which use an auxiliary quad mesh.
  - For the first time, 1-cones are addressed.
  - The construction is of a metapolygon as opposed to a full polygon (definition 5).
- An automatic method to realize the construction.
- An improved algorithm for mapping into a self-overlapping polygon.
- An algorithm to improve the domain polygon w.r.t. the minimal angle.
- For the first time, a fully robust pipeline, and a comprehensive evaluation to challenge it.

1.2. Outline

After reviewing related work (section 2), we go over basic definitions (section 3) and set some new ones (section 4). The sections following that handle the general case of any cone valence. Each of these sections contains a clear detailed algorithm that is simple to implement.

Before starting these sections that describe the construction, we encourage the reader to read appendix B. It gives an illustrative introduction to the main idea in the paper in the simpler setting of cone valence ≥ 3, which have a small number of cases. We only describe the construction, which can easily be translated into an algorithm.

We begin the construction of the general case (cone valence ≥ 3) with the *foundation polygon* (section 5). A *foundation set* of cones is identified, where there are 10 possible sets, and a foundation polygon is constructed.

After identifying the foundation sets, the rest of the cones are divided into sets with a 0 field-index sum (section 6).

These sets are connected into a seam (tree) graph, having a single component (section 7). First, connections are made between set members, and then the sets are connected through chains of negative

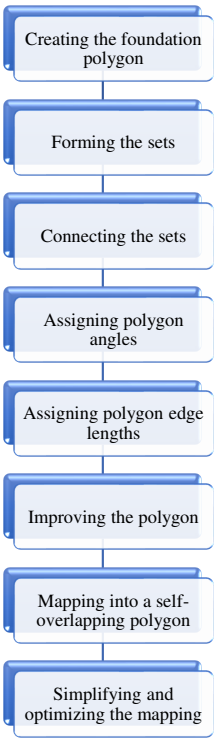


Figure 3: *The pipeline.*

cones. This special seam ensures that each cone has enough copies to create a simple polygon in the domain. Creating a simple polygon trivially ensures that the polygon is self-overlapping. Moreover, it ensures a sequence of interleaving positive and negative cone copies, which can balance each other in terms of polyline turns, creating monotone polylines.

The tracing itself of the seam over the surface is described in appendix C. Specific refinement rules for tracing the seam over the surface that keep the refinement moderate are given.

Afterwards, domain angles are assigned to the polygon vertices to create monotone polylines that ensure that the polygon is intersection-free (section 8).

The last step of constructing the boundary polygon is assigning polygon edge lengths (section 9). Both finding polygon angles and finding edge lengths are performed using linear programming.

To increase the quality of the mapping, the boundary polygon is improved by maximizing the minimal angle of the coarse metapolygon triangulation (appendix D) that is used as convex subdomains for mapping the interior.

In appendix E, we describe a new algorithm for mapping a surface into an arbitrary self-overlapping polygon. The mapping is guaranteed to be locally injective.

After constructing a valid seamless initial mapping, we complete the process by optimizing the mapping distortion and removing redundant refinement (appendix F).

The algorithm is evaluated in section 10, and comparison to the state-of-the-art is performed in appendix G.

The pipeline is illustrated in figs. 3 and 4.

The appendices and the second part of the figures are in the supplement.

2. Related Work

The problem of seamless parametrization and the application for quad meshing is discussed in [BLP*13]. We review work related to generating locally injective seamless parametrization with guarantees for a given set of cone singularities over a triangle mesh.

Campen et al. [CSZZ19] use discrete conformal mapping to parameterize a mesh. The mapping is seamless up to edge lengths. This involves non-linear numerical optimization, which is prone to numerical issues. They follow with a padding step to equalize the edge lengths of twin half-edges that ensures seamlessness. The case of a 1-cone is not treated.

Chen et al. [CZK*19] take a similar route, using discrete conformal mapping to parameterize a mesh. However, ensuring seamlessness is not detailed.

Section G.2 tests the robustness of [CSZZ19], reveals numerical issues, and points out a fundamental flaw in the conformal mapping approach.

Myles et al. [MPZ14] trace the motorcycle graph over the surface to create an initial quad patch partition. This is followed by an algorithm modifying the quad layout structure to ensure that consistent parametric lengths can be assigned to the edges. Each patch is mapped to create an initial locally injective mapping. The algorithm may add cones to the final mapping, violating our requirement to find a mapping for a restricted set of cones.

Recently, Zhou et al. [ZTZC20] follow the approach in [CSZZ19], but instead of using discrete conformal mapping, they construct quad patches with the desired singularities in a combinatorial way, which guarantees result validity. The patches are padded to have matching borders, and a full quad mesh is constructed. The surface is mapped into the quad mesh with the corresponding domain boundary polygon. The domain polygon is full (definition 5), which can cause issues (see section G.3 for comparison and discussion), and the paper provides results only of the generation of the initial boundary polygon. Besides that, their construction is limited to cone valence ≥ 2, a restriction necessary for the 0-genus case ([ZTZC20], section 6.5).

We continue reviewing combinatorial methods for generating a quad mesh for a given set of singular vertices. See appendix H for details on how an auxiliary mesh can be used to generate a valid seamless mapping.

Grunbaum [Gru69] generates a quad mesh for a given set of singularities. The paper handles sphere-topology surfaces and is limited to cone valence ≥ 3. Similarly, Jucovic and Trenkler [JT73] generate a quad mesh with the same restriction on the cone valence but for any surface topology. Both papers are purely theoretical: no clear algorithm is given for the constructions, and there are no implementation or experiment details.

We take a different approach altogether that constructs a boundary metapolygon directly. This bypasses the need to use an auxiliary mesh with its shortcomings (section G.3). Compared to previous work, our construction is arguably simpler and easier to implement. By handling the case of 1-cones for sphere topology, we prove existence of a mapping in this setting for the first time. Finally, unlike previous work, we handle the whole pipeline, and provide extensive tests and results of its robustness.

3. Background

Given a surface triangle mesh, a parametrization (mapping) assigns *UV* coordinates to mesh vertices, creating a (flat) *layout* in a planar domain. The mapping induces a flat metric, where an inner regular vertex has a total domain angle 360° and otherwise, it is considered a cone singularity. The desired cone angles of 90°-multiple (in the mapping domain) are given as input, and they can be, for example, user-defined or from a smooth cross field [BZK09].

Before flattening the surface, it is cut into a disk along a tree of seam edges that spans the cones. Each seam edge on the surface is mapped into two *twin half-edges* in the domain, corresponding to the two triangles sharing the surface edge. This creates *vertex copies*. Each seam vertex would have as many domain copies as its (uncut) seam graph valence. The seam goes through all the cones such that all cone copies lie on the domain boundary polygon, and

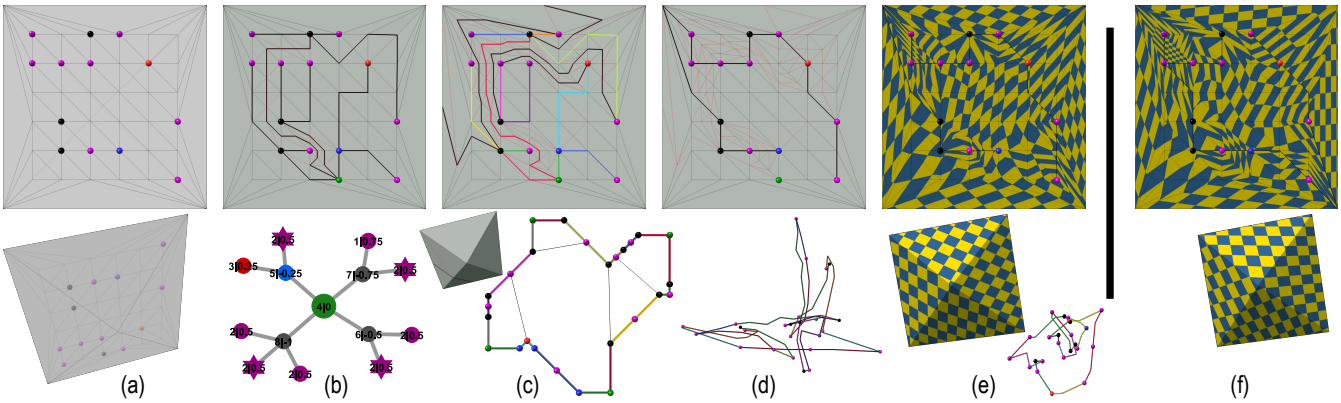


Figure 4: *A pipeline illustration on a base of a flipped pyramid. See section 3.1 for figure notations. (a) The input mesh with (random) cone singularities. (b) Tracing a special seam (marked in thicker lines) over the surface. The (uncut) seam graph is at the bottom. Thin, red lines mark new (non-original) edges, resulting from the refinement. (c) Solving for a simple polygon's angles and edge lengths, and laying it out. The polygon is decomposed into convex sub-domains. This produces four inner (thin, black) edges that are traced over the surface. The surface is mapped into the sub-domains, creating a locally injective mapping. (d) Changing the seam to pass through all cones (and the regular foundation vertex in green is not needed anymore) using only original edges. Since the mapping is seamless, it remains unchanged, and the new polygon remains self-overlapping. Removing the seam from new edges allows contracting and simplifying them. (e) Optimizing isometric distortion (3.7 symmetric Dirichlet energy) and illustrating the final result with a checkered pattern. Areas around the (random) cones are distorted as expected, attesting to the challenge of producing a locally injective mapping for the given (random) cones. (f) Comparison with a state-of-the-art, field-based method [BZK09]. This method generates a smooth field for the given singularities, which is used as frames to lay out the mesh and create an initialization for the distortion optimization procedure (4.7 symmetric Dirichlet energy). While a field-based approach is not guaranteed to succeed, this example is easy enough for it to produce a valid result. In general, if it happens to succeed, then we expect it to produce a similar result to our approach that guarantees local injectivity. Like in this example, our approach may produce a slightly better mapping (in terms of distortion) due to our required refinement, which is not part of the field-based method.*

its interior remains flat (consists of regular vertices). The angle of a vertex in the domain is measured by the sum of all the vertex's copy angles. For an illustration of cutting the seam and mapping it into a polygon, see fig. 11 (*top-left*).

For a parametrization to be locally injective, it is necessary that the boundary polygon is *weakly self-overlapping*; see definition and illustration in [WZ14]. From here on out, we use self-overlapping to refer to weakly self-overlapping.

For a (domain boundary) polygon vertex with an angle α (= sum of incident triangle corner angles), we define its angle defect as $180^\circ - \alpha$.

Definition 1 (seamless parametrization)**.** *We say a parametrization is* seamless *if the domain coordinates for corresponding vertex copies of twin half-edges e, e' are related by*

$$(u', v') = R^r_{90^\circ}(u, v) + (s, t), \quad r \in \{0, 1, 2, 3\}, \quad (s, t) \in \mathbb{R}^2 \qquad (1)$$

where $R^r_{90^\circ}$ is an $r \cdot 90^\circ$ rotation. The rotation and translation are per edge (shared by the edge's vertices).

For a detailed background on seamless parametrization (with the relevant material on cross fields and quad meshes) we refer the reader to [BZK09; MZ12; Lev21].

Seamlessness implies that cone angles are 90°-multiple. The other direction is known as well:

Proposition 1. *For a genus-0 surface, if all cone angles in the layout polygon are 90°-multiple (and twin half-edges have the same length), then the parametrization is seamless.*

See proof in appendix J.

For a sphere, if all the cone copies of a seamless parametrization lie on a grid, the parametrization becomes *integer seamless*, and a quad mesh can be extracted. Our objective is seamless parametrization, and making it integer is a separate problem [CBK15; LCBK19]. Some of the methods in section 2 generate integer seamless parametrization, and with minor effort our construction can be made one as well. However, since the construction results in a mapping with high distortion that only serves as an initial point for an optimization process, there is no advantage in it being integer seamless.

The cone singularities of a triangle mesh translate into singular vertices in the generated quad mesh, with corresponding graph valence. Considering the relation between cone singularities in a cross field and quad mesh singularities, we will use the following definition.

Definition 2. *A k-cone is a cone singularity of valence k. It has the equivalent following properties [BZK09]:*

- *A corresponding vertex with valance k in the (generated) quad mesh.*
- *A (domain/field) cone angle $k \cdot 90^\circ$.*

- *Field index* $1-\frac{k}{4}$ *if it is an inner vertex, and* $1/2-\frac{k}{4}$ *if it is a border vertex. The field index determines if a cone is negative (negative field index) or positive (and zero means it is a regular vertex).*
- *A (domain) angle defect* $360-k\cdot 90^\circ$ *if it is an inner vertex and* $180-k\cdot 90^\circ$ *if it is a border vertex.*

We define a cone configuration (set) as feasible if there exists a locally injective parametrization with corresponding (domain) cone angles. Every set of cones is feasible if its total field index is equal to the Euler characteristic of the surface (Poincaré–Hopf theorem), and every cone index is smaller than 1. Previous work has shown that there is one exception, a torus with a pair of {3,5}-cones, when considering the case of cone valence ≥ 2 [CSZZ19] (previously, cone valence ≥ 3 [JT73]). For the case of cone valence ≥ 1, we show that there are no further exceptions for sphere topology, and the problem remains open for higher genera.

3.1. Figure Notations

In the figures, the vertices are color coded: 3-cones in red, 5-cones in blue, 6-cones and higher quad valence in black, 2-cones and 1-cones in magenta, and regular metavertices (seam valence ≥ 2) in green.

In the force graphs of the (uncut) seam graph (over the surface), *foundation cones* (section 5) are hex-star-shaped. Edges around a vertex do not necessarily correspond to the order on the surface. The labels on the vertices show the quad valence on the left and the field index on the right. In some of the figures, dashed edges between negative cones indicate that the cones belong to the same set after a set union was performed (section 6).

Some of the figures contain vector images, which can be zoomed-in and magnified without compromising quality.

Regarding notations in general, all angles in the paper are measured in degrees.

4. Preliminary Definitions

Some preliminary definitions pertaining the boundary polygon that would be used throughout the paper.

Definition 3 (a foundation set)**.** *A set of cones that is designated to account for the Euler characteristic* χ*. The set also includes one regular cone (with 0 field index), which is described in section 5. The field indices of the set sum up to* χ *(except for one special case, detailed in appendix I).*

Definition 4 (a foundation polygon)**.** *A simple domain polygon that consists of the vertex copies of the foundation set. It satisfies seamlessness constraints and respects the (total) cone angles of the corresponding surface vertices.*

Definition 5 (a metapolygon)**.** *A boundary polygon that consists only of cone copies and copies of the (single) foundation regular vertex. It is the domain cut graph of a seam metagraph that passes through the corresponding vertices on the surface. We distinguish it from a full polygon that also includes all the copies of regular vertices on the boundary of the cut mesh (the seam).*

	1	2	3	4	5	6	7	8	9	10
3-cones	8	6	4	2		5	3	1	2	
2-cones		1	2	3	4		1	2		1
1-cones						1	1	1	2	2

Table 1: *All 10 possible sets of foundation cones. Each column is a set, and each row gives the number of cones of a certain type in each set.*

The size of a boundary polygon is given in the following theorem.

Theorem 1. *Let n be the number of vertices (cone and regular) that a seam graph traverses over the surface. The size of the domain boundary polygon of the cut mesh is* $2(n-1)$*.*

See proof in appendix J.

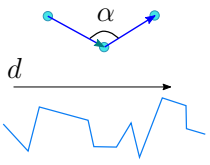
Definition 6 (a foundation edge)**.** *The part of a metapolygon between two (consecutive) copies of the foundation regular vertex.*

Definition 7 (a foundation polyline)**.** *A foundation polyline is the polyline (consisting of cone copies) between two consecutive copies of members of the foundation set (cones or the regular vertex). Each foundation edge consists of two foundation polylines (which can be viewed as twin metahalf-edges).*

Definition 8 (polygon vertex angle defect)**.** *In the context when there is no triangulation (the interior is not mapped yet), the angle* α *at a (CCW-oriented) polygon vertex is the internal angle between the two incident edges. The internal (unsigned) angle* α *of a polygon (or polyline) vertex is to the left of the incident edge vectors. The angle defect at the polygon vertex is* $180^\circ-\alpha$*.*

Definition 9 (a monotone polyline)**.** *A polyline is monotone with respect to a direction d if the (unsigned) angle between a polyline edge direction and d is at most* 90° *(see inset).*

See inset for illustration of a polyline vertex angle (*top*) and a monotone polyline (*bottom*).

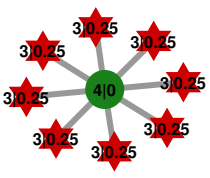


5. The Foundation Polygon

To simplify exposition, we assume in the next two sections that a positive (foundation) set of cones with total field index χ exists. The end of section 6 references appendix I that addresses the special case of a deficit set.

We start by identifying a foundation set of positive cones with field index summing up to χ. Considering input fields with only positive cones (having total field index 2), there are 10 possible foundation sets; see table 1.

The foundation polygon is the basis for the general shape of the final polygon. In fig. 5, we offer an arbitrary domain construction for each one of the 10 sets. The construction adds a regular (0 field index) vertex to a foundation set. On the surface,



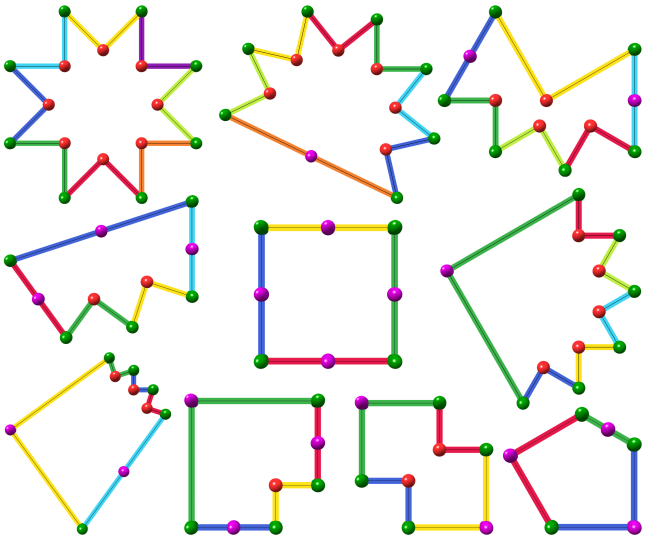


Figure 5: *The polygon of each foundation set.*

the seam tree is constructed as a star graph, where the regular vertex is set as its root in the middle, and the cones are its child leaves. The inset shows (see section 3.1 for notations) the (uncut) seam graph of the first polygon in fig. 5.

In the domain, the regular vertex has as many copies as the number of cones, and they are interleaved between the cone copies. (Note that this is different than our choice in appendix B, where the choice there was made to create a simple rectangle-like foundation polygon). The angles of the cone copies are already determined (since a positive cone is a leaf in the constructed seam tree, it has a single domain copy, which accounts for the entire cone angle). We split the regular vertex angle (360°) evenly between its copies.

The purpose of the regular vertex is to easily create a simple polygon—ensuring that it is self-overlapping—for a foundation set, using a simple general rule.

This is, however, an arbitrary choice to ease implementation. Alternatively, one can use rectilinear foundation polygons. Then, adding appropriate integer constraints (which may not be needed in practice) to eq. (2), it is possible to create a (final) rectilinear domain boundary polygon.

The edge lengths were found using the algorithm in 9, combined with non-intersection constraints; see appendix K for details.

Note that a suggested polygon is offered for a specific ordering of connecting the set on the surface. For flexibility, we want to be able to construct a polygon for any order of a set. Due to symmetry of the patterns, this is easily achieved, and it does not affect the suggested procedure.

6. Forming the Sets

After identifying the foundation set (an arbitrary choice) that accounts for χ, the field indices of the rest of the cones sum up to zero (Poincaré–Hopf theorem). We divide these cones into sets such that in the final construction:

Algorithm 1: Forming the sets

Input: A set of m negative cones $\{n_i\}_{i=1}^m$

for $i = 1$ *to* m **do** // initialize sets $\{s_i\}_{i=1}^m$
  $s_i = \{n_i\}$ // a singleton
  $w_i = FieldIndex(n_i)$ // $w \in \mathbb{R}^m$
Put $\{s_i\}_i$ in a (minimum) priority queue Q, using w as keys
while *not* $Q.empty()$ **do** // expand sets
  $s_i = Q.pop()$
  p = find a free (positive) cone with the largest field index
  $\omega = FieldIndex(p)$
  if $\omega \le |w_i|$ **then** // add the positive cone to s_i
    $s_i.add(p)$
    $w_i = w_i + \omega$
  else // unite sets
    $s_j = Q.pop()$
    $s_i = s_i \cup s_j$
    $w_i = w_i + w_j$
    Delete s_j
  if $w_i \neq 0$ **then**
    $Q.push(s_i)$

- Each set's field indices sum up to zero.
- Each cone has enough copies to construct a simple polygon.

The algorithm is given in alg. 1. We begin by defining a set s_i for each negative cone n_i, and we keep track in w_i of the total field index of the set. Next, we select a set with a minimal (negative) total field index. We add to it a *free* (not in any other set—including the foundation set) cone with a maximal (positive) field index.

We continue adding positive cones to the sets until they are all *complete* (zero field index sum). When needed, we unite sets to lower their total field index to balance a free positive cone. For example, if there are two sets with a single 5-cone each, and there is a free 2-cone, then the two sets would unite in order to balance the addition of the positive cone to the *united set*.

The algorithm is greedy, giving priority to sets or cones with maximal absolute field index. This is done to reduce the number of united sets, reserving sets or cones with minimal absolute field index to fill deficiencies in other sets. It is also used to prove correctness later on.

See appendix I for treatment of the special deficient-set case.

Theorem 2. *There can be at most three negative cones in each set.*

See proof in appendix J.

7. Connecting the Sets

We connect the cones over the surface using non-intersecting paths to form a seam tree (of meta-edges between cones). This is done in three steps.

Paths within a set. We begin by connecting all the negative cones in a set (three at most) to form a short chain. Next, we divide the positive cones in the set between the set's negative cones, connecting them as leaves to the negatives.

Long negative group chains. Given n foundation cones, divide the sets evenly into n groups, each associated with a foundation cone. If there are not enough sets, then some groups will remain empty; also, one group may have less sets than the others. In each group, connect all the (short) negative chains within the sets into one long chain of negative cones (with positive cone leaves).

Connecting the chains. Pick an arbitrary regular vertex as the foundation regular vertex. For each *negative group chain*, connect one side to the regular vertex and the other to the foundation cone that is associated with the group.

In the first step, when dividing the positive cones between the negatives inside a set, the objective is that the seam-graph valence of each negative cone is high enough so there are enough domain copies of the cone to split its angle such that each copy angle would be $< 360^\circ$. This is stated in the following theorem and corollary:

Theorem 3. *In the (uncut) seam graph (that was created above), the valence of a negative cone with an angle α is at least c, where $\frac{\alpha}{c} < 360^\circ$.*

See proof in appendix J.

Corollary 1. *We can divide a (positive or negative) cone angle between its domain copies such that each copy angle would be strictly smaller than 360°. If appropriate edge lengths exist (satisfying seamlessness constraints and avoiding intersections), then the polygon would be simple—ensuring that the polygon is self-overlapping.*

8. Assigning Polygon Angles

One option to assign polygon angles is to generalize the approach in appendix B. In appendix B, we found a general rule for connecting the sets and assigning angles. Since we now allow $\{1,2\}$-cones, we would need more rules to address all cases arising from the new possibilities for a set.

For a given set, there are multiple ways that its cones can be connected. Specifically, each way corresponds to how the edges connecting positive cones in the set to a negative cone are distributed between the two edges that connect the negative cone with other negative cones. Each of these ways creates a new case. To reduce the number of cases, one option is to always connect a set's members in one specific way. This requires that a specific order of seam edges around a negative cone would be observed. That is, when connecting cones over the surface, we would need to consider in which sector incident to a cone an edge should emanate from [SJZP19]. While this would reduce the number of cases, it would still be tedious to address each case manually.

Instead, we offer an automatic method that solves for the polygon angles (and later for appropriate edge lengths). This method is oblivious to the edge order around a vertex, which simplifies implementation. That is why, for example, we can allow for edges in force graphs in the figures to have arbitrary order around vertices.

The angles of foundation cone copies were determined in section 5. To guarantee that the polygon is not self-intersecting, we assign angles for cones on a foundation polyline such that the polyline would be monotone (definition 9). This can be formulated as a linear programming problem:

$$\min_{\substack{\alpha\in\mathbb{R}^n \\ t\in\mathbb{R}}} \quad t \tag{2a}$$

$$\text{s.t.} \quad A\alpha = \bar{\alpha}, \tag{2b}$$

$$|B_j(180^\circ\cdot\mathbb{1}-\alpha)| \le t\cdot\mathbb{1},\ j=1..b\ , \tag{2c}$$

where α is a column vector of polygon angles ordered CCW. t is an auxiliary variable that bounds the maximal absolute *sequence angle defect* (sum of angle defects of consecutive polygon vertices) of sequences along foundation polylines.

Equation (2b) constrains the sum of the copy angles of a cone to be equal to the cone angle. $\bar{\alpha}\in\mathbb{R}^m$ is a given column vector of cone angles (from the input). A is an $m\times n$ binary matrix, where the rows represent the cones, the columns represent the domain copies, and a cell is 1 iff the copy belongs to the cone.

b is the number of foundation polylines (between the $\frac{b}{2}$ foundation cone copies and $\frac{b}{2}$ copies of the foundation regular vertex). Equation (2c) bounds the *maximal absolute sequence angle defect* along a foundation polyline e_j with c inner vertices. It considers the c angle sequences: $(\alpha_{i_1}),(\alpha_{i_1},\alpha_{i_2}),...,(\alpha_{i_1},...,\alpha_{i_c})$. It bounds the absolute sum of angle defects in each sequence. If the bound t is at most 90°, then e_j is monotone. $\mathbb{1}$ is a column vector of ones of an appropriate (inferred) dimension. Let $[i_1...i_c]$ be the (consecutive) indices in α of the c (vertex) angles in e_j; (to avoid cyclic indexing,) WLOG, assume $i_1 \le i_c$ (assuming e_j is not empty). Let L be a $c\times c$ lower triangular binary matrix, where all the entries on the diagonal and below are one. Then, B_j is a $c\times n$ binary block matrix

$$B_j = \left[\mathbb{0}_{c\times(i_1-1)}\ L\ \mathbb{0}_{c\times(n-i_c)}\right]\ ,$$

where $\mathbb{0}$ is a matrix of zeros (dimension in subscript).

The foundation copy angles are added to the problem as linear equality constraints.

Theorem 4. *A solution to eq. (2) exists with maximal absolute angle sequence defect (along a foundation polyline) $t \le 90^\circ$, implying $90^\circ \le \alpha_i \le 270^\circ$ and monotone foundation polylines.*

See proof in appendix J.

9. Assigning Polygon Edge Lengths

After solving for the angles, we can infer the polygon edge directions (unit vectors) $d\in\mathbb{R}^{2\times n}$. Using these, we solve the following linear programming problem to find the (simple) polygon vertex positions:

$$\min_{\substack{x\in\mathbb{R}^{2\times n} \\ l\in\mathbb{R}^n}} \quad \|l-l_0\|_1 \tag{3a}$$

$$\text{s.t.} \quad l_i = l_j,\ \forall (i,j) \in I_{corr}, \tag{3b}$$
$$x_{i+1} - x_i = l_i d_i,\ i = 1,\ldots,n, \tag{3c}$$
$$l_i \geq 1,\ i = 1,\ldots,n, \tag{3d}$$
$$l_i \leq l_{short},\ i \notin I_{foundation}, \tag{3e}$$
$$l_i \geq l_{long},\ i \in I_{foundation}\ , \tag{3f}$$

where x is the coordinates of n vertices, $x_i \in \mathbb{R}^2$. l is the edge lengths. l_0 is given edge lengths to aspire to.

Equation (3b) enforces the same length to corresponding twin half-edges, where $I_{corr} \subset \mathbb{N}^2$ holds pairs of indices of twin half-edges in the polygon.

Equation (3c) converts direction and length into an edge vector.

$I_{foundation} \subset \mathbb{N}$ consists of indices of edges that are incident to a foundation cone. l_{short} and l_{long} are two (suitable) scalar constants (set to 1.5 and 3 respectively in our experiments). The objective of eq. (3e) and eq. (3f) is to set the turns along a foundation polyline as small features to prevent global intersection with other foundation polylines, keeping a rough shape of the foundation polygon.

l_0 is initialized as a vector of ones. The l_1-norm in the objective promotes sparsity ([Ela10], section 1.6), i.e. concentrating the error in a few edges, expecting most of the edge lengths to remain unchanged. This is an advantage for the iterative variant of the algorithm that uses intersection constraints: i) a solution is found faster, where the solution from the previous iteration provides a good initialization; ii) less intersections occur after adding non-intersection constraints in the following iteration.

Theorem 5. *A solution to eq. (3) exists.*

See proof in appendix J.

In the algorithm, we first determine foundation edge lengths (solving only for the foundation polygon, e.g. fig. 5) by solving eq. (3) with added non-intersection constraints; see appendix K. In the iterative process that is described there, l_0 is set to the l solution from the previous iteration (and to a vector of ones in the first iteration). Alternatively, the foundation polygon has limited number of options which can be calculated once and used as constants (e.g. fig. 5).

We determine the sets' edge lengths by solving another instance of eq. (3)—a single iteration without non-intersection constraints since the polylines are monotone. The long edges in l_0 are set to the corresponding foundation edge lengths that were previously found.

In practice, the separation (to ensure that the MI problem is small and tractable) into two problems was not necessary, and we solved for both foundation and set edge lengths together as a single problem, initializing l_0 as a vector of ones. Only in a few cases, an additional iteration with a few additional non-intersection constraints (involving only a small number of integer variables) was necessary. That is, due to the polylines' monotonicity, the expected number of intersections (if any) is minimal.

This concludes the combinatorial part of the pipeline that constructs the seam. See the appendices for details on how to trace the seam over the surface (appendix C) and the rest of the pipeline that improves the domain polygon (appendix D), maps the interior (appendix E), and simplifies and optimizes the mapping (appendix F). You can use section 1.2 as a road map.

model	F_1	cones	Myles14		ours		
			F_2	energy	F_2	F_3	energy
deformed_armadillo	100K	179	155K	4.15	269K	102K	4.16
focal-octa	20K	116	48K	137K	52K	21K	5.37
isidore_horse	100K	127	146K	4.14	222K	101K	4.14
vase-lion100K	100K	160	153K	111K	211K	102K	4.24

Table 2: *Details of figs. 1 and 9. "F_1": number of facets in the input model. "F_2": number of facets in the initial mapping, before simplification. "F_3": number of facets in the final result. "energy": the symmetric Dirichlet energy. Energy above 100 indicates a failure—a mapping with nearly collapsed triangles that cannot be optimized using an external solver [SPS*17].*

10. Evaluation

We performed extensive stress tests for the whole pipeline. Normally, cones are placed by a field optimization method and serve to lower the mapping distortion, thus assisting a layout optimization method; for example, the cones in [MPZ14]'s dataset. Instead, we positioned random valence cones at random places over the surface. These cones not only do not help to reduce the distortion but significantly increase it, making it more challenging to the layout optimization method [SPS*17].

This is the first time that such a test was performed, testing the whole pipeline—including the final part of the layout optimization that uses an external solver. The models proved to be an extraordinary, new challenge to the state-of-the-art. In our experiments, [SPS*17] performed significantly better than other methods. The comprehensive testing asserts that our generated initial mapping has adequate quality for further optimization. We also report timings for the whole pipeline even though most of the time is spent on the external solver. This, again, to test and report the quality of the generated mappings.

Smaller models. We selected 11 (small) models to stress-test the pipeline. For each model, we created 100 random cone configurations (1100 in total), 50 cones in each. We rejected configurations which are impossible to lay out without refinement (i.e. having a high valence cone without enough incident triangles, which would pose a problem to methods that keep the triangulation intact). The models vary in triangulation quality and test the non-combinatorial part of the algorithm. Conversely, for the combinatorial part of the algorithm that constructs the boundary polygon, the choice of the model does not matter, and it depends only on the type of cones in the input. 50 cones are a small enough number for a reasonable chance for special cases to occur randomly and to be tested.

One sample of each of the 11 models is shown in figs. 6 and 22. Each of the first 11 rows in table 3 is the average of 100 test results (i.e. 1100 random inputs in total). The amount of refinement in the initial mapping is reasonable, and it is mainly due to mapping the

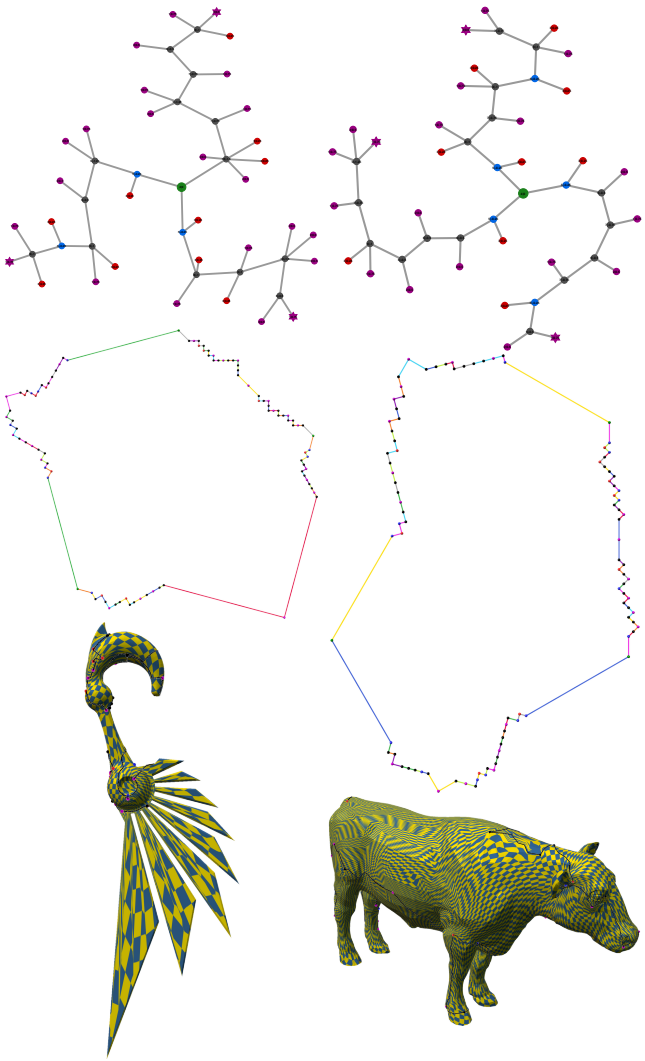


Figure 6: *A sample of each of 2/11 benchmark models; see fig. 22 for more. Rows: the seam graph, boundary polygon, and final result.*

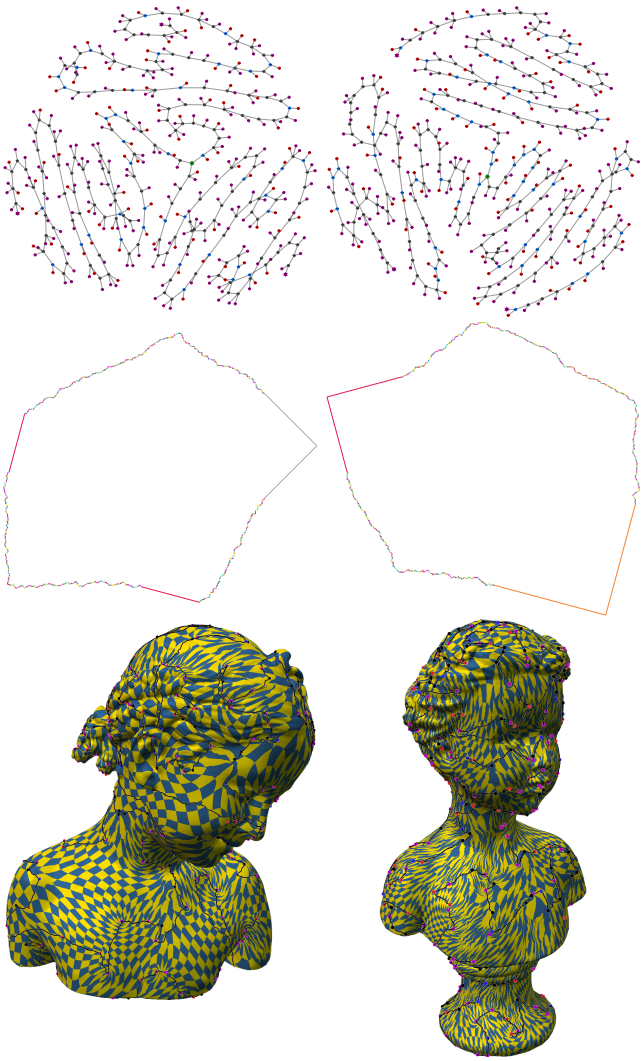


Figure 7: *Two of the larger models; see fig. 23 for the rest. Rows: the seam graph, boundary polygon, and final result.*

interior (required in any combinatorial approach) into a non-convex polygon; more specifically, tracing the inner edges of the convex partition. In the final result, only a modest refinement of 0.5-4.5% triangles remains after simplification.

Since the cones are randomly chosen, we do not expect from the mappings to have low distortion, and the textured squares are not well-shaped and attest to the cone configuration difficulty. The symmetric Dirichlet energy is below 100, which indicates that the optimization [SPS*17] converged successfully.

Larger models. In figs. 7 and 23, we tested larger models with 500 and 1000 random cones. Details are given in table 3. The remaining refinement in the final result is between 1-21% triangles. The final distortion (energy) indicates successful convergence.

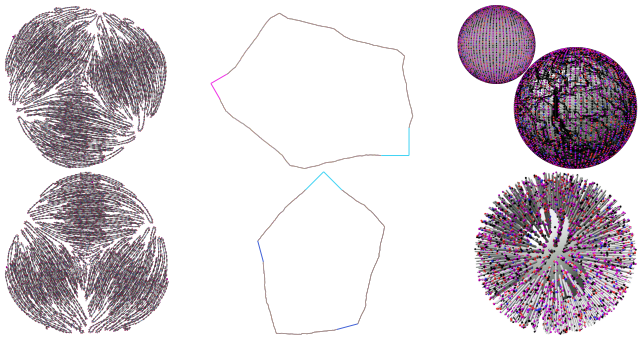


Figure 8: *A sphere with a (random) cone at each of its 10002 vertices and a squishy ball with 10000 random cones. Details in the last two rows in table 3.*

#	model	cones	F_1	F_2	F_3	%	energy
1	blade_earring	50	23K	54K	24K	0.8	17.43
2	cow2	50	9K	24K	9K	2.6	7.26
3	dente	50	19K	38K	19K	0.6	4.78
4	duck	50	19K	40K	19K	0.6	4.68
5	focal-octa	50	20K	43K	20K	0.5	4.73
6	homer	50	10K	26K	10K	1.7	6.49
7	moai	50	20K	41K	20K	0.6	4.47
8	pear	50	22K	52K	22K	1.0	5.74
9	retinal	50	7K	20K	7K	1.6	5.36
10	shark	50	20K	46K	20K	1.2	8.65
11	triceratops	50	6K	18K	6K	4.5	7.31
12	sphere_100-cone	36	1K	4K	3K	179	72.50
13	bimba100K	500	100K	440K	101K	0.7	4.74
14	buste	500	100K	758K	103K	3.0	8.54
15	gargoyle100K	500	100K	629K	103K	3.3	11.70
16	lucy100k	500	100K	927K	108K	8.3	28.76
17	ramses	500	100K	688K	101K	0.9	5.20
18	rgb_dragon	500	105K	1.0M	108K	2.4	9.39
19	vase-lion100K	499	100K	522K	102K	1.9	5.54
20	armchair	1000	100K	1.3M	121K	21.0	13.93
21	bunnyBotsch	1000	111K	1.1M	124K	11.6	7.22
22	camille_hand100K	1000	100K	979K	116K	15.9	10.75
23	deformed_armadillo	1000	100K	1.1M	114K	14.2	9.75
24	eros100K	1000	100K	796K	111K	10.7	5.79
25	isidore_horse	1000	100K	1.4M	119K	18.6	12.33
26	magalie_hand100K	1000	100K	1.1M	116K	15.8	9.66
27	pensatore	1000	100K	765K	112K	12.0	5.79
28	santa	1000	152K	1.9M	175K	15.8	17.81
29	spiked_ball100k	999	112K	1.1M	134K	20.2	14.35
30	igea200k	5000	200K	1.2M			
31	sphere100_all	10002	20K	949K			
32	squishy_ball	10000	1.1M	7.0M			

Table 3: *Details of figs. 6 to 8, 22 and 23. "F_1": number of facets in the input model. "F_2": number of facets in the initial mapping, before simplification. "F_3": number of facets in the final result. "%": percentage of remaining refinement. "energy": the symmetric Dirichlet energy. Dashed lines hint about the models partition into figures.*

For the last three models in table 3, we used 5000-10002 random cones. The boundary polygon construction was successful, but we did not proceed with the rest of the pipeline since the polygon was too large (due to the large number of cones) for the step of improving the polygon; see details on this limitation in appendix D.

Run time. The experiments were performed on a laptop with a quad-core 2.8GHz CPU and 16GB RAM, using a single-threaded implementation. Table 4 summarizes the running time. The performance of the final step of simplifying the mesh and optimizing the distortion depends on the chosen external optimization method [SPS*17].

The main challenge that the optimization method was facing is the initial distortion, which is very high due to nearly collapsed triangles (e.g. 10^{11} symmetric Dirichlet energy); see fig. 10. As explained in appendix D, what currently important to us is that the optimization method is successful (at all), which depends on the minimal domain triangle angle in the initial mapping and not necessarily its energy (as observed). A guaranteed initial mapping that has low distortion is a luxury that no one offers yet. The challenge is due to the current advancement in mapping the interior of the boundary polygon. This remains an open problem for future methods to try and construct an initial mapping with low distortion.

model	cones	trace	angles	lengths	improve	map	opti
11 benchmark	50	6	0.1	0.1	5	6	119
larger models	500	151	0.1	0.1	260	336	5467
larger models	1000	310	0.1	0.1	290	1576	7870
igea200k	5000	1720	35	3			
sphere100_all	10002	2543	188	9			
squishy_ball	10000	15091	161	11			

Table 4: *Run time summary. The columns provide the average time in seconds of processing the set of models in a row. "trace": forming the sets and tracing the seam over the surface. "angles" and "lengths": solving the two linear programming problems for the boundary polygon angles and edge lengths. "improve": improving the polygon. "map": tracing inner edges and mapping into convex sub-domains. "opti": simplifying the mesh and optimizing the distortion using an external solver [SPS*17].*

Comparison with the state of the art is done in appendix G.

11. Conclusion

We presented a method that guarantees local injectivity of a sphere-like surface parametrization for any valid cone configuration. Our method consists of a fully-robust (theoretical guarantees where possible and resilience in experiments) pipeline that employs novel, elegant solutions to sub-problems. Robustness was demonstrated through comprehensive experiments, including a novel stress test of random cones.

The first part of the pipeline asserts the existence of and guarantees to construct a simple polygon—which is self-overlapping by definition—for a given (valid) cone configuration that includes, for the first time, cones of quad valence ≥ 1. The construction is purely combinatorial, independent of the geometry: the domain polygon is purely based on the input set of cones. A numerically robust algorithm that automates the implementation of the construction is offered, which handles all cases uniformly, including the cases and complexity introduced by 1-cones.

The algorithm divide cones into 0-field-index-sum sets and traces a special seam tree to span them, providing enough domain cone copies for constructing a simple polygon. Afterwards, two linear problems are solved to determine the polygon angles and edge lengths. Each of the problems can be translated into a few lines of Matlab code [Löf04].

Since angles of cone copies are guaranteed to be between 90° and 270°, there are no precision issues when solving the first linear

problem in eq. (2). Similarly, due to the limited range of the angles and the arbitrary choice of the minimal edge length, no numerical issues are expected in solving the second linear problem in eq. (3). In summary, this part of the algorithm is free from numerical and precision issues. Due to the two linear programs to automate the construction, the automatic construction algorithm cannot be technically considered combinatorial, but it is as robust.

The main advantage of the generated polygon—compared to the state-of-the-art—is being a small metapolygon, which is crucial for subsequent steps.

The second part of the pipeline maps the surface into the polygon. First, the boundary polygon is improved to alleviate numerical issues in later steps that are caused by nearly collapsed triangles. Then, an algorithm is suggested for mapping a surface into a self-overlapping boundary, which was developed independently from [SJZP19] and suggests a novel approach of tracing inner edges over the surface by shooting rays in a disk domain.

The third part of the pipeline simplifies the mesh and optimizes the isometric distortion using an external state-of-the-art solver. Part of the process is alleviating numerical issues and encouraging faster convergence through mesh beautification.

We note that the second and third part of the pipeline are responsible for most of the mesh refinement (dominated by the interior mapping) and run time (dominated by the external isometric optimizer). It relies on independent problems that any competing method would need to solve. So far, only [ZTZC20] offers an explicit algorithm and test results for a combinatorial construction of a boundary polygon. Our method is the first to address the full pipeline, which includes improving the boundary polygon before the interior mapping and the optimization in the third part of the pipeline.

Now that the full pipeline is addressed and tested for the first time, we see that there is room for improvement, and enhancing each of the following is a future goal:

- Typically, the initial generated mapping has very high distortion. This is the main culprit for the long run time in the optimization step.
- The interior mapping is the main culprit of the refinement size.
- The optimization method is slow.

We clarify that our method is intended to handle fields with arbitrary cones, where standard field-based methods fail to generate a valid mapping. If a field contains good cones—placed in order to lower distortion—and a valid mapping is successfully generated with simpler methods (which is the common case, e.g. see the success rate of the state of the art in [Lev21]), then it would be easier and simpler to employ these significantly simpler methods instead for the task.

The main limitation of our method is that it can handle only sphere topology surfaces. Let us discuss the challenges in extending the method to higher genera. First, in a higher genus surface, the foundation set of cones is different—not necessary limited to 10 possible sets, requiring different foundation polygons—and the homology loops (not present in the spherical case) need to be a part of it. Besides that, prescribing holonomy angles to homology generator loops introduces a new challenge. This task is no less important than prescribing cone angles, and it is still an open problem ([CSZZ19; ZTZC20] only offer seamlessness along the loops).

An example of the importance of a homology loop angle is figure 5 in [CZ17]. Even though the example is a T-mesh, it is a simple illustration of the influence of homology loop holonomies on the edge flow. Another field-related example is figure 6 in [RVLL08]. A related discussion is given in [MPZ14], supplement, section 1.

The new machinery presented in this paper is going to be used to tackle this problem in future work.

From a theoretical point of view, we proved quad mesh existence for vertex valence ≥ 1 for spheres. Generalizing our construction to higher genera would prove existence for vertex valence ≥ 1 for any genus surface.

Another limitation is that we do not handle sharp features. Sharp features are important, e.g. in CAD models, and they would require special handling.

References

[BLP*13] BOMMES, DAVID, LÉVY, BRUNO, PIETRONI, NICO, et al. "Quad-Mesh Generation and Processing: A Survey". *CGF* 32.6 (2013), 51–76 3.

[BZK09] BOMMES, DAVID, ZIMMER, HENRIK, and KOBBELT, LEIF. "Mixed-integer Quadrangulation". *ACM Trans. Graph.* 28.3 (2009), 77:1–77:10 3, 4.

[CBK15] CAMPEN, MARCEL, BOMMES, DAVID, and KOBBELT, LEIF. "Quantized Global Parametrization". *ACM Trans. Graph.* 34.6 (2015), 192:1–192:12 4.

[CSZZ19] CAMPEN, MARCEL, SHEN, HANXIAO, ZHOU, JIARAN, and ZORIN, DENIS. "Seamless parametrization with arbitrary cones for arbitrary genus". *ACM Transactions on Graphics* 39.1 (2019), 1–19 3, 5, 11.

[CZ17] CAMPEN, MARCEL and ZORIN, DENIS. "Similarity maps and field-guided T-splines: a perfect couple". *ACM Transactions on Graphics (TOG)* 36.4 (2017), 1–16 11.

[CZK*19] CHEN, WEI, ZHENG, XIAOPENG, KE, JINGYAO, et al. "Quadrilateral mesh generation I: Metric based method". *Computer Methods in Applied Mechanics and Engineering* 356 (2019), 652–668 3.

[Ela10] ELAD, MICHAEL. *Sparse and redundant representations: from theory to applications in signal and image processing*. Springer Science & Business Media, 2010 8.

[Gru69] GRUNBAUM, BRANKO. "Planar maps with prescribed types of vertices and faces". *Mathematika* 16.1 (1969), 28–36 3.

[JT73] JUCOVIČ, E. and TRENKLER, M. "A theorem on the structure of cell–decompositions of orientable 2–manifolds". *Mathematika* 20.1 (1973), 63–82 3, 5.

[LCBK19] LYON, MAX, CAMPEN, MARCEL, BOMMES, DAVID, and KOBBELT, LEIF. "Parametrization quantization with free boundaries for trimmed quad meshing". *ACM Transactions on Graphics* 38.4 (2019), 1–14 4.

[Lev21] LEVI, ZOHAR. "Direct Seamless Parametrization". *ACM Transactions on Graphics* 40.1 (2021), 1–14 4, 11.

[Löf04] LÖFBERG, J. "YALMIP : A Toolbox for Modeling and Optimization in MATLAB". *Proceedings of the CACSD Conference*. 2004 10.

[MPZ14] MYLES, ASHISH, PIETRONI, NICO, and ZORIN, DENIS. "Robust Field-aligned Global Parametrization". *ACM Trans. Graph.* 33.4 (2014), 135:1–135:14. ISSN: 0730-0301 1, 3, 8, 11.

[MZ12] MYLES, ASHISH and ZORIN, DENIS. "Global parametrization by incremental flattening". *TOG* 31.4 (2012), 109 4.

[RVLL08] RAY, NICOLAS, VALLET, BRUNO, LI, WAN CHIU, and LÉVY, BRUNO. "N-symmetry direction field design". *ACM Transactions on Graphics* 27.2 (2008), 10 11.

[SJZP19] SHEN, HANXIAO, JIANG, ZHONGSHI, ZORIN, DENIS, and PANOZZO, DANIELE. "Progressive embedding". *ACM Transactions on Graphics* 38.4 (2019), 32 7, 11.

[SPS*17] SHTENGEL, ANNA, PORANNE, ROI, SORKINE-HORNUNG, OLGA, et al. "Geometric Optimization via Composite Majorization". *ACM Trans. Graph.* 36.4 (2017) 8–10.

[WZ14] WEBER, OFIR and ZORIN, DENIS. "Locally injective parametrization with arbitrary fixed boundaries". *TOG* 33.4 (2014), 75 2, 4.

[ZTZC20] ZHOU, JIARAN, TU, CHANGHE, ZORIN, DENIS, and CAMPEN, MARCEL. "Combinatorial Construction of Seamless Parameter Domains". *Computer Graphics Forum*. Vol. 39. 2. 2020, 179–190 3, 11.

Volume 0 (1981), Number 0 pp. 1–17 COMPUTER GRAPHICS *forum*

Seamless Parametrization of Spheres with Controlled Singularities—Supplement

Zohar Levi

Victoria University of Wellington, New Zealand

A. Motivation for Cone Fidelity

In a seamless mapping, the choice of the seam is not unique, and it can be changed while keeping the mapping isometric. In contrast, cone angles and placement are unique, and for sphere-type surfaces, they completely determine seamlessness constraints. Therefore, they are good candidates to characterize a group of seamless mappings, as well as quad meshes (e.g. through defining a base complex). A similar argument is given in [RVLL08], where cone singularities are treated as holes in the surface, and a cross field homotopy class is defined as homology basis cycles around cones (along with homology generators in a higher genus). When optimizing a seamless parametrization, we would like to preserve this mapping characteristic by preserving the cones (placements and angles).

To motivate this preservation, consider, for example, the following application. The most popular methods to generate a quad mesh are field-based. They provide the user with convenient tools to control the singularities and produce a low distortion mapping that a high quality quad mesh can be extracted from. The singularities play an important role of defining a quad mesh. For example, the number of singularities affects the complexity of the base complex, and the singularity placement at less prominent locations is preferred. Also, in a typical quad mesh generation process, the resolution of the generated quad mesh depends on the minimal distance between the cones. Thus, after generating an initial cross field, a post-processing step of cone collapse and relocation is common to keep the quad mesh resolution low.

In summary, the final cone angles and positions are usually a product of a process of careful design, which begins with the main objective of reducing mapping distortion (leading to well-shaped quad facets) and is followed by subsequent steps to improve the final generated quad mesh quality. Therefore, when constructing a locally injective seamless mapping, we would like to preserve the exact input cone angles and locations, allowing for arbitrary designing and tailoring of a mapping.

B. The Simple Case of Cone Valence ≥ 3

To illustrate our approach, we describe a "manual" construction of the boundary polygon of the mapping domain in the simpler setting of [Gru69] for spheres, where the maximum cone field index is $\frac{1}{4}$ (cone valence ≥ 3). Given a triangle mesh and a (valid) set of prescribed cones on it, we construct a simple polygon that satisfies the seamlessness constraints.

We begin the construction with the *foundation polygon* (definition 4). In the current setting (defined above), the χ of a watertight sphere is 2, and the foundation set consists of eight 3-cones.

On the surface, the seam graph is a tree that spans the foundation set, and its *metagraph* (definition 5) is illustrated in the *top-left* of fig. 11. The seam is cut and laid out as a foundation polygon.

In the current setting, we define an (single) arbitrary foundation polygon, which is depicted in the *bottom-left* of fig. 11. It has four foundation meta-edges.

Next, we divide the remaining cones into sets of minimal size (cannot be broken into smaller sets), where in each set, the sum of field indices is zero. In the current setting, it means that each set would have a single negative cone with field index $-\frac{k}{4}$ and k 3-cones. Some set examples:

- A pair of {3,5}-cones with corresponding field indices $\{\frac{1}{4}, -\frac{1}{4}\}$.
- A set of {3,3,6}-cones with corresponding field indices $\{\frac{1}{4}, \frac{1}{4}, -\frac{2}{4}\}$.
- A set of {3,3,3,3,8}-cones with corresponding field indices $\{\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, -\frac{4}{4}\}$.

We add the sets to the foundation polygon one by one, where we maintain a valid seamless polygon after each addition. Each addition consists of two polylines that are inserted in the beginning and end of an (arbitrary) foundation meta-edge.

For example, consider the addition of a pair of {3,5}-cones in fig. 11 (middle). On the surface, the 5-cone is connected to the seam tree between the regular vertex and a foundation cone. The 3-cone is connected to it as a leaf. In the domain, we add the pair to the bottom meta-edge: one (blue) 5-cone copy is inserted on the left, and the rest are inserted on the right. The sum of angle defects of each polyline is zero. We can connect each polyline without changing the meta-edge beginning (the vector of the first segment) and end (the vector of the last segment) directions. The single 5-cone copy on the left is assigned 180°, and the rest of the 5-cone copies

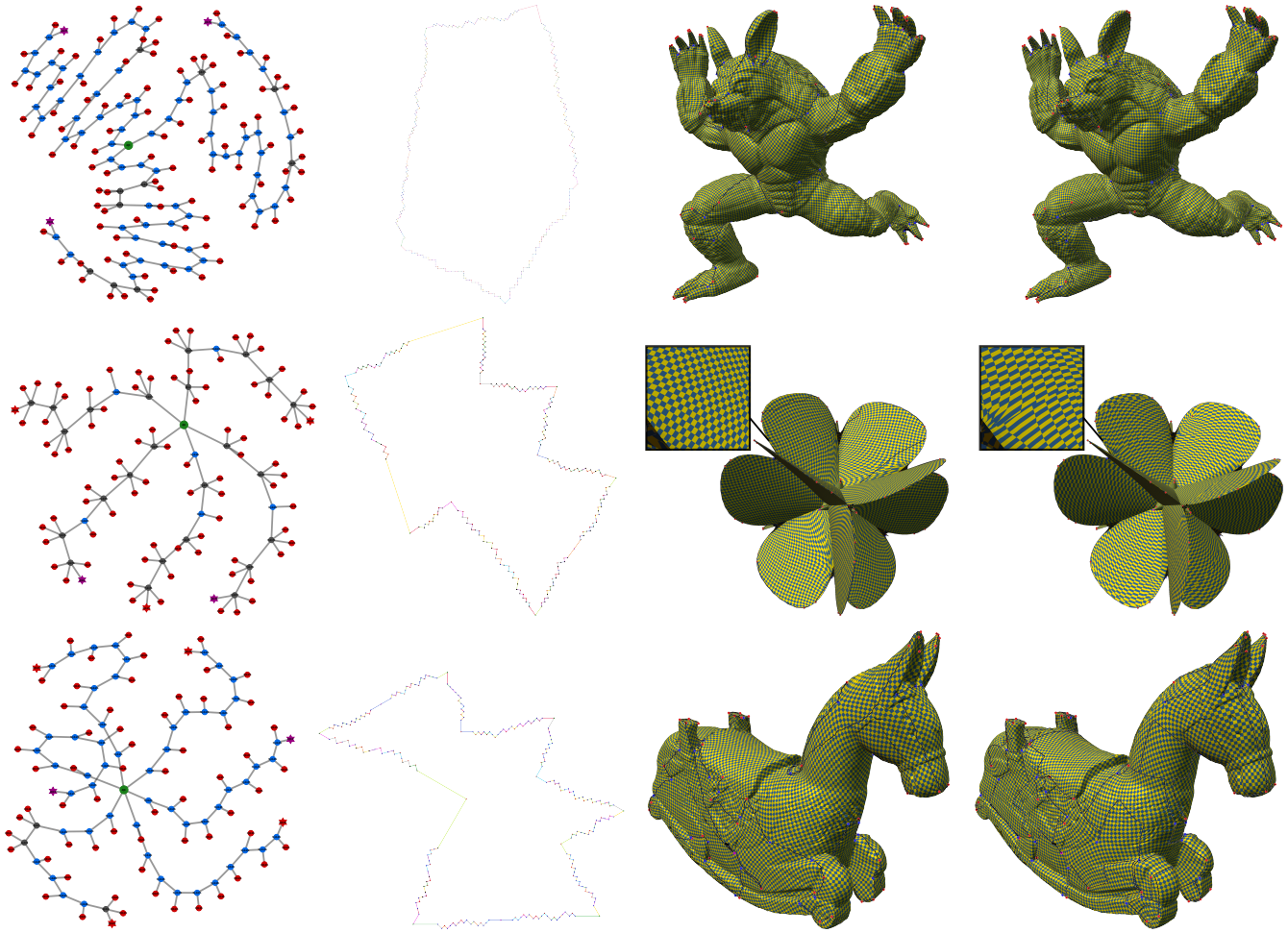


Figure 9: *Using input fields from [MPZ14]. From left to right: the (uncut) seam graph, the boundary polygon, our final result, and [MPZ14]'s result. Details in table 2.*

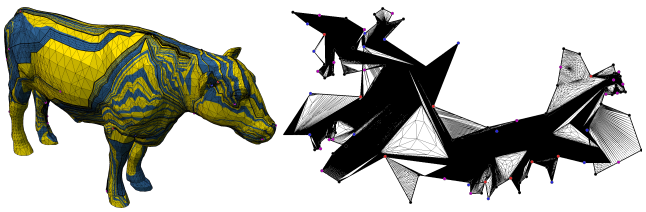


Figure 10: *An initial mapping before optimization (*76000 *symmetric Dirichlet energy).*

are assigned 135° each. The negative copy angles sum up to 450°. Except for the foundation polygon, each positive cone has a single copy.

Next, consider the addition of a set of {3,3,6}-cones in fig. 11 (*right*). On the surface, the 6-cone is connected to the seam tree between the regular vertex and a foundation cone, and the 3-cones are connected to it as leaves. In the domain, the (black) 6-cone copies are added to the bottom foundation edge. One copy is added on its left side, and the rest of the copies are added to the right, interleaved with 3-cones. On the left side, the single negative copy is assigned 180°. On the right side (of the bottom foundation edge), the first and last negative copies are assigned 135° each, and the middle copy is assigned 90°. The negative copy angles sum up to 540°.

The general rule of inserting a polyline for a set with a negative cone with $-\frac{j}{4}$ field index, i.e. a $(j+4)$-cone, becomes apparent. On the surface, the negative cone is connected to the seam tree near the regular foundation vertex, and the 3-cones are connected to it as leaves. In the domain, the negative cone has $j+2$ copies. One negative copy is added to the bottom foundation edge on the left and is assigned 180°. The rest of the $j+1$ negative copies are added to the right of the foundation edge, and they are interleaved with the j 3-cone copies of the set. The first and last copies on the right side are assigned 135° each, and the rest are assigned 90°. The negative copy angles sum up to $180^\circ + 2\cdot 135^\circ + (j-1)90^\circ = 360^\circ + j\cdot 90^\circ$ as required.

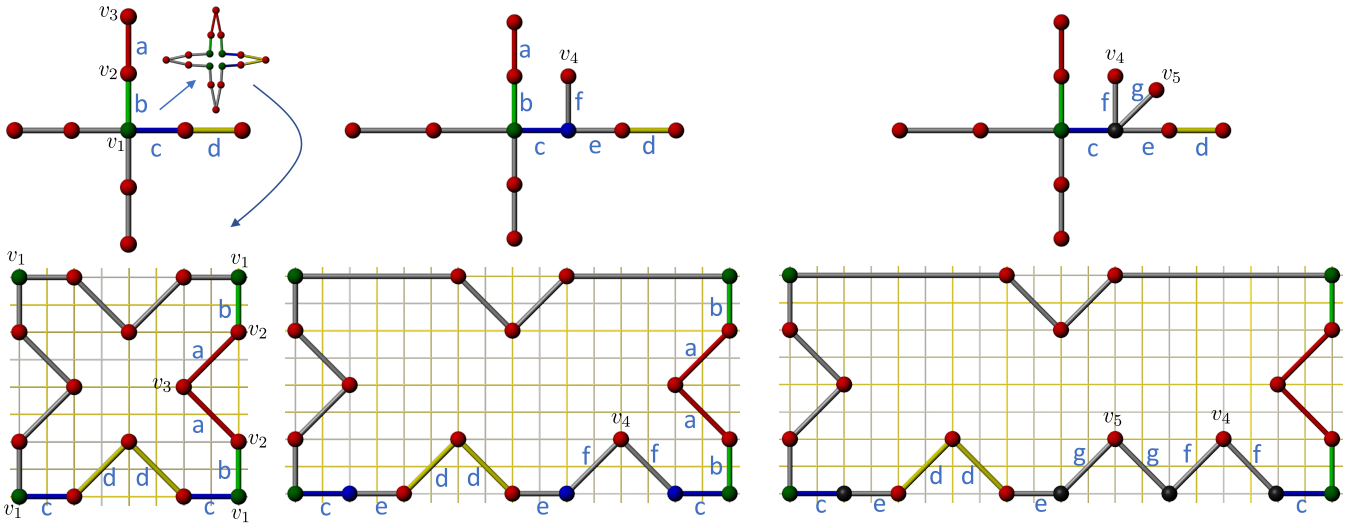


Figure 11: *Manual construction. Top: the (uncut) seam graph (over the surface). The regular vertex in green, 3-cones in red, 5-cones in blue, and the other negative cones in black. Bottom: the domain polygon after cutting the seam at the top image. Some of the vertex correspondence is labeled. For example, the label v_1 at the bottom-left is assigned to four vertex copies of the green vertex at the top-left. Some of the edges are colored and labeled to illustrate the correspondence. For example, the label 'a' is assigned to the two red twin half-edges at the bottom-left that correspond to the red seam edge in the seam graph at the top-left. Left: the foundation set. At the bottom, the foundation polygon consists of four green vertex copies of the regular vertex v_1, four vertex copies of four 3-cones (e.g. v_3)—each has a single vertex copy, and eight copies of four 3-cones (e.g. v_2—each has two vertex copies. Middle: the foundation set with an added pair of {3,5}-cones. Right: the foundation set with an additional set of {3,3,6}-cones.*

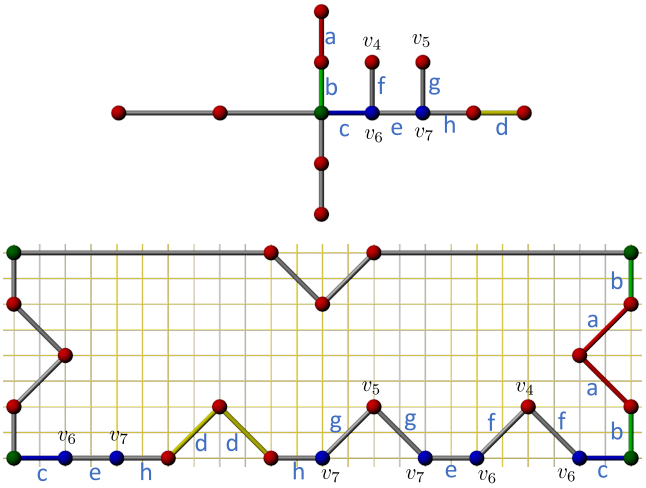


Figure 12: *Manual construction of the foundation set with additional two pairs of {3,5}-cones.*

After determining the order of the cone copies and assigning the angles, we need to set the edge lengths. There are three constraints: i) twin half-edges have the same length, ii) the polygon is closed, and iii) the polygon stays simple. It is easy enough at this setting to come up with edge lengths that satisfy these constraints; we will address that in detail later in the general setting of any cone valence. Note that if one is careful to place the cone copies on a grid, then after mapping the interior, we get an integer seamless parametrization, which a quad mesh can be extracted from.

Figure 12 illustrates consecutive addition of two pairs of {3,5}-cones.

The manual construction follows the following principles:

- A set of cones is identified as the foundation set that accounts for χ.
- All other cones are divided into sets, where the field index of each set sums up to zero. As a consequence, each set can be added as a "local feature" without disturbing the general shape of the polygon (that is set by the foundation set).
- The polygon is simple and hence, self-overlapping.

The general case of any cone valence follows the same principles.

C. Tracing the Seam

In section 7, we needed to connect the cones over the surface using non-intersecting paths. To find a single (simple) path, we use (Dijkstra-like) front propagation from a source vertex in the mesh to find the closest vertex with the required properties.

This is a crucial point to keep the refinement minimal. For example, finding a vertex v_j of a certain valence to connect to a vertex v_i. Instead of first picking a specific v_j from the list of vertices, and then looking for the shortest path between v_i and v_j, we start a front propagation from v_i, finding the closest vertex v_j with the required

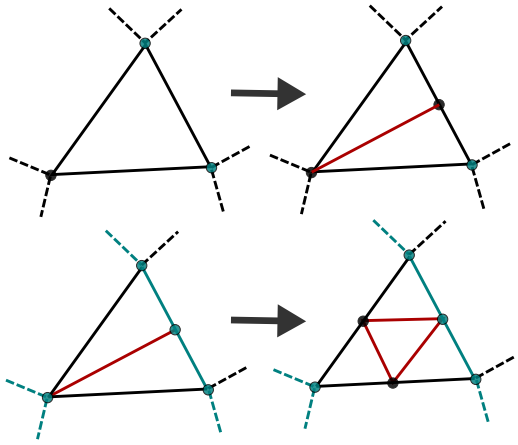


Figure 13: *Two edge-refinement examples, one in each row. The Left column shows the triangulation before refinement, and the right column shows after. Vertices in blue are cones or the seam pass through them. Blue edges are part of the seam. New edges are in red.*

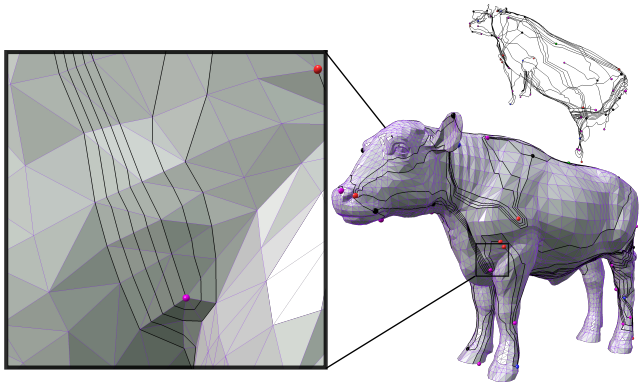


Figure 14: *An example of tracing the seam. Original edges are in blue, and the seam is in black.*

valence. The general rule is then to keep the objective as general as possible.

Before finding a path that connects a pair of cones, a refinement step is performed to guarantee that a path exists (in the mesh triangulation) between each pair of vertices.

Definition 10 (an original edge)**.** *In a refined mesh, an original edge is an edge that exists in the source mesh or is part of a source-mesh edge. Otherwise, it is a new edge.*

We limit the refinement to original edges. An original edge requires refinement if:

- Each of its vertices is either a cone (or the special regular foundation vertex) or a seam vertex.
- It is not a seam edge.

The steps for refining an edge e:

- Remove new edges from the two facets incident to e.
- Split e by adding a vertex to its center.

After all the edges are split, we triangulate each facet (incident to the new split edges) using alg. 2.

Notes:

- Before finding the first path, we test all mesh edges that are incident to cones if they require refinement. Before finding subsequent paths, we iterate only over edges incident to the last path that was found.
- The refinement operation is topological. A new vertex that splits an edge is positioned arbitrarily in the middle of the edge.
- Since we refine only original edges, each vertex in the mesh is incident to at least two original edges. This is a crucial property that keeps the refinement moderate, where otherwise it can grow substantially.

Figure 13 shows two refinement examples. In the first row, before the refinement, there are two neighboring cones. After the refinement, the right edge between them is split, and the new vertex is connected to a vertex that is not a cone and not on the seam.

In the second row, before the refinement, the seam passes through the left vertex and the right edge. After the refinement, the new edge is removed, and the left and bottom edges are split. The facet is then triangulated using alg. 2. Note that if we had not replaced the new edge that was there before the refinement, then we would have needed to refine it (increasing the refinement unnecessarily).

After constructing the seam, we simplify the mesh, removing all unused refinement. More specifically, we remove all vertices that are not on the seam and do not have at least three incident original edges (and triangulate as necessary).

Figure 14 shows a final result of tracing the seam. Note in the blowup image how a few seam paths cross a single original facet. Since we limit new vertices to original edges and we do not refine seam edges, each refined path has only a single segment inside a facet. On the other hand, if we allowed new edges to cross, resulting in new vertices inside the facets, then each path may have crossed new inner edges of previous paths, leading to exponential growth of the number of vertices.

Algorithm 2: Facet triangulation

Input: A facet f to triangulate
if *f is a triangle* **then**
  return
v_1 = a vertex in f that is not on the seam
if *$\exists v$ with two incident seam edges* **then** `// provide access to a cone`
  $v_1 = v$
v_2 = a vertex ($\neq v_1$) that is not a neighbor of v_1, as well as it is not a cone or on the seam if v_1 is
Split f with an edge $v_1 v_2$
Perform a recursive call on each of the two new facets incident to the new edge

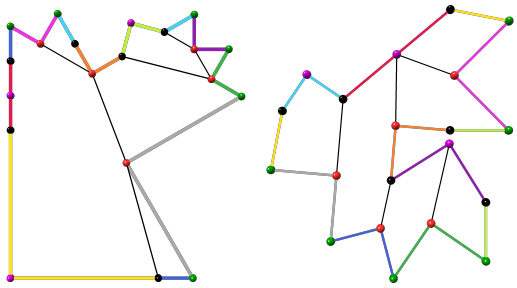


Figure 15: *The convex decomposition of the polygon on the left has a minimal facet angle* 15°*. After improving it, the polygon on the right has a minimal facet angle* 45°*. The cone set is:* $\{1,1,2,3,3,3,3,3,6,7\}$*-cones. Seamlessness constraints and total cone angles are preserved. Border edge color is arbitrary, and it does not match between the two polygons.*

D. Improving the Boundary Polygon

In order for the initial mapping result to be useful for a distortion optimization step, we need to avoid nearly collapsed triangles. Otherwise, numerical issues may arise that hinder subsequent optimization (appendix F). Through experiments, we concluded that the best measure for a mapping quality is the minimal triangle angle in the domain. The minimal angle on the surface is less crucial since it can be remedied (e.g. through vertex perturbation) without compromising local injectivity. Also, since we start with *beautification-based* optimization (appendix F), the surface triangle shape is irrelevant when optimizing the initial mapping. This explains why a distortion energy, e.g. symmetric Dirichlet, cannot determine conclusively the initial mapping quality: it is based on the singular values of the mapping, which depend on the surface triangle shape.

While the minimal triangle area may also give indication in some cases, it is not conclusive either. There could be a small, well-shaped triangle with a small area that does not cause any issues; alternatively, there could be a sliver triangle with a large area that poses an issue.

When mapping the interior (appendix E), mapping into a nearly collapsed triangle is bound to produce triangles with even sharper angles. Therefore, we need to improve the boundary polygon.

When constructing the polygon, there is no triangulation yet. The only practical way to guarantee that it is self-overlapping is to make the polygon simple (consider that even determining if a polygon is self-overlapping is not easy, and e.g. Shor–van Wyck algorithm [SV92] is needed for that). Once we have a self-overlapping polygon, we can triangulate it. Then, keeping the polygon self-overlapping is just a matter of preserving the total vertex domain angles.

Given a triangulated self-overlapping polygon, we solve the following NLP (nonlinear programming) problem to maximize the minimal triangle angle:

$$\min_{\substack{l\in\mathbb{R}^n\\ \alpha\in\mathbb{R}^m\\ \mu\in\mathbb{R}}} \quad \mu \tag{4a}$$

$$\text{s.t.} \quad \alpha_i = arccos\left(\frac{l_b+l_c-l_a}{\sqrt{2l_bl_c}}\right),\ (l_a,l_b,l_c) = sel(\alpha_i), i = 1...m, \tag{4b}$$

$$\sum_{\alpha_i\in cor(v)} \alpha_i = ang(v),\ \forall v \in V, \tag{4c}$$

$$\mu \le \alpha_i,\ i = 1...m, \tag{4d}$$

$$l_a \le l_b + l_c,\ (l_a,l_b,l_c) = sel(\alpha_i), i = 1...m, \tag{4e}$$

$$l_i = l_j,\ \forall (i,j) \in I_{corr}\ , \tag{4f}$$

where n is the number of edges in the polygon, and $m := 3(n-2)$ is the number of angles in its triangulation. l is a vector of squared edge lengths of the polygon. α is a vector of the triangulation angles. μ is a lower bound on a triangle angle.

Equation (4b) uses the law of cosines to relate squared edge lengths to triangle angles. Given an angle, the function $sel()$ returns squared edge lengths of three triangle edges a,b and c, where a is the edge opposite the angle, and b and c are the edges that enclose the angle inside a triangle.

For a vertex v, eq. (4c) constrain the sum of cone copy angles to be equal to the cone angle. The function $cor()$ returns the domain *corner* angles incident to the cone copies, and the function $ang()$ returns the cone angle. This constraint ensures local injectivity.

Equation (4d) bounds the minimal angle.

Equation (4e) verifies that the triangle inequality holds.

Equation (4f) is seamlessness constraints.

Note that there is no need for explicit constraints to keep edge lengths positive [CLW16] (in essence, edge lengths are tied to the restricted angles through the cosine theorem).

Given a boundary polygon, we improve it as follows. We triangulate the polygon using the Delaunay rule. Then, we solve eq. (4) for a new polygon. Since this modifies the triangulation immersion such that it may not be Delaunay any more, we repeat these two steps until convergence.

While the initial polygon is simple, in later iterations it may be only self-overlapping. Also, we would like to keep the procedure general to serve other methods, e.g. section G.3, which do not generate a simple polygon. In this case, we use the version of Shor–van Wyck algorithm suggested in [WZ14] to decompose the polygon into simple parts, which is also required for mapping the interior (appendix E). This algorithm is based on dynamic programming and has space complexity $O(n^2)$ and time complexity $O(n^3)$, which makes it impractical for large polygons.

Given the decomposition, we triangulate each (simple) sub-polygon. Note that since the decomposition edges are arbitrary, we need to modify the triangulation of the whole weakly self-overlapping polygon to be Delaunay (through edge flips).

We used the off-the-shelf interior-point solver IPOPT [WB06] to solve eq. (4). The derivative of eq. (4b) can be found in [CLW16], appendix C. In our experiments, typically the initial mapping had a minimum triangle angle $< 1^\circ$, and the procedure converged within eight outer iterations, increasing the minimum angle to about 20°.

While reaching a global minimum is not guaranteed, usually when the minimum angle was $> 5^\circ$, there were no numerical issues in later steps. See fig. 15 for illustration of improving the minimal angle of a polygon triangulation.

E. Mapping into a Self-overlapping Polygon

After constructing a self-overlapping boundary polygon in the domain that corresponds to the (cut) seam, the next step is to map the interior of the cut mesh. One option is to use dual-harmonic mapping [WZ14], where the surface and the polygonal domain (which is triangulated arbitrarily) are both mapped to an intermediate unit-circle domain, and the two triangle layouts are intersected to create a composed mapping. However, in our experiments, this method did not scale well (generated too much refinement and became too slow), and we came up with an alternative similar to [SJZP19] (but was developed independently).

Our approach is to decompose the domain polygon into convex sub-domains while tracing a similar decomposition over the surface. Then, we use Tutte embedding to map into each convex sub-domain bijectively. This leads to a *piece-wise Tutte embedding* of the surface.

Convex decomposition of a self-overlapping polygon is done by triangulating the polygon [WZ14] and deleting inner edges as long as the union of their two coinciding facets remains convex.

Each inner edge in the (decomposed) domain polygon needs to be traced over the surface between the two corresponding vertices. One option to accomplish that is to use the tracing algorithm in appendix C. In this approach, one needs to be mindful about keeping the correct order of meta-edges around a vertex. This requires starting a path from the correct *edge sector* incident to the vertex. An approach along these lines was suggested in [SJZP19]. Unlike our tracing algorithm, their tracing algorithm is naive, allowing refinement of inner edges, and thus leading to substantial amount of refinement.

We suggest a better alternative for tracing the inner edges over the surface. We map the surface into a unit circle using Tutte embedding. Then, we trace an inner polygon edge by simply intersecting the line that passes through the two corresponding boundary vertices with the mapped triangulation. Since the circle is convex, we are guaranteed that no two inner paths cross. The resulting 2D inner line segments are easily reproduced on the surface: given a 2D inner line segment ℓ and a 2D facet f that it intersects, we use the linear mapping of f to map the intersection of ℓ and f to the 3D surface.

In this approach, paths are guaranteed to be in the correct edge sector incident to a vertex. Moreover, in our experiments, on larger polygons, the algorithm required less refinement than tracing the paths directly over the surface.

For planar path intersection we used the exact computation in [CGA20]. This method was robust no matter the amount of distortion in the triangulation.

In comparison to dual-harmonic mapping, piece-wise Tutte embedding performed significantly better on all accounts: speed, amount of refinement, and numerical stability. To give intuition for that, consider mapping into a convex boundary. The piece-wise Tutte requires no refinement, and it does not entail a mid-step that introduces distortion. In contrast, the dual-harmonic mapping intersects two triangulation that are mapped to an intermediate unit-circle domain.

F. Simplifying and Optimizing the Mapping

Our generated seamless parametrization is intended to be used as initialization to a distortion optimization process. In this section, we perform such an optimization along with further simplification of the mesh refinement that was previously performed.

In the initial mapping, the refinement is necessary to "bend" (original) triangle edges so they could fit within the boundary polygon. When optimizing the mapping and reducing the isometric distortion, these edges straighten, and most of the refinement is no longer necessary.

The simplification—the removal of new vertices—is done through edge contraction as long as the mapping stays valid (it is locally injective, and cone angles are preserved in the domain). Due to our careful refinement process, a new vertex is incident to exactly two *original* edges. A new vertex is removed by, first, deleting non-original incident edges. Then, contracting one of the two incident original edges (joining the new vertex with a neighbor). Finally, the two facets incident to the other original edge are triangulated. This way, the refined and simplified mesh are faithful to the input mesh, preserving its original vertices and edges (which may be refined).

The current seam may pass through new edges which prevents simplification of incident new vertices. Since the parametrization is seamless, we relocate the seam so it passes only through original edges. (This is always possible since in the original mesh there is a tree graph that spans the cones.)

To optimize distortion, we used the method in [SPS*17], minimizing the symmetric Dirichlet energy.

In difficult cases, the quality of the initial mapping was not good enough, and the solver had numerical issues and converged prematurely. The problem was due to nearly-collapsed slivers in the 3D surface. Simply perturbing incident vertices was not enough to resolve the problem. To address that, we used a *beautification-based* optimization: instead of using a surface triangle as source for optimizing the mapping, we used an equilateral triangle with unit-length edges. From an energy viewpoint, it means that the singular values (that the energy is based on) depend only on the quality of the domain triangles, where as normally, they measure the (anisotropic) scaling between a surface and domain triangle. This approach not only resolved the numerical issues but also significantly increased the convergence rate.

We alternated between simplification and beautification until convergence. Then, all new (refined) vertices that were left were distributed uniformly along original edges (between a pair of original vertices) over the surface. Lastly, a final optimization was run using the surface triangles as source, producing a low distortion parametrization with moderate refinement.

A related approach to our *beautification-based* optimization was offered by Liu et al. [LYNF18]: instead of using the input mesh as a

reference to define the objective function, they introduce a *progressive* reference mesh that contains bounded distortion with respect to the current parametrization and is as close as possible to the input mesh. The distortion bound is increased iteratively until the reference is equal to the input mesh. In comparison, we have only two stages: in the first, we use a mesh with unit equilateral facets as a reference, and in the second stage, we use the input mesh as a reference.

G. Comparison with the State of the Art

We compared with the state of the art. Our method is the only one to pass successfully all the tests.

G.1. Field-Based Methods

Bommes et al. [BZK09]. Field smoothing is a popular approach to seamless parametrization and quad meshing, generating high quality results. However, none of the methods can guarantee the validity of the mapping (local injectivity and cone preservation), which limits their use.

We tested the field-smoothing approach on the benchmark models. Given cones, we used [BZK09] to generate a smooth cross field. The cross field was used as frames to generate an initial layout, using the isometric convexification-based optimization described in [Lip12; APL14; LZ14; Lev21]. Only if the optimization is successful, the resulting mapping will be locally injective. Moreover, even if it satisfies seamlessness constraints and is locally injective, it still may not preserve the cone angles (which requires a stronger non-convex constraint). A successful initial mapping was used in a subsequent optimization [SPS*17] to minimize the symmetric Dirichlet energy.

On the 1100 inputs of smaller models, 8.8% of the tests failed, and the rest had 15.25 symmetric Dirichlet energy on average. The successful results have worse distortion than our algorithm results, but we allow refinement while they do not. On the larger models, the fail-rate was close to 50%.

Myles et al. [MPZ14]. In figs. 1 and 9, we compare with [MPZ14], and the input fields are taken from the paper dataset. These input fields were generated using [BZK09], followed by collapsing neighboring cones. Therefore, the fields are mainly designed for an isometric mapping (along with some consideration for the final resolution of a generated quad mesh), where the cones assist in lowering the distortion.

[MPZ14] traces a motorcycle graph over the surface using a given field, and it guarantees that the result is locally injective. However, it may add cones, violating our problem requirement to preserve given cone angles. Moreover, as previously noted (appendix D), local injectivity is not an assurance for mapping quality to be good enough for further optimization—and if it is not, then we consider the mapping to be invalid. That is, extreme slivers that pose numerical issues to further processing can be considered, for all intents and purposes, collapsed triangles, which are no better than foldovers: for example, quality aside, they would pose an issue when extracting a quad mesh from a seamless parametrization. The final step of [MPZ14] uses Tutte mapping into a convex domain, but since no care is taken in shaping the boundary (appendix D), it is prone to result in nearly collapsed triangles, which may invalidate the result.

From their dataset—which was mostly tackled successfully by using [BZK09] frames in the layout step as described above—we selected four of the more challenging models (in this dataset—which we consider easy, as previously explained). On three of them (deformed_armadillo, isidore_horse, and vase-lion100K), the [BZK09] approach failed. [MPZ14] succeeded in producing locally injective mappings for all four models, and adding cones was not necessary. However, results for two of the models (focal-octa and vase-lion100K) contain nearly collapsed triangles. While the Dirichlet energy that was used in [MPZ14] does not penalize nearly collapsed triangles, the symmetric Dirichlet energy that we used does. Due to the nearly collapsed triangles, these two mapping results posed a problem when used as an initialization for the symmetric Dirichlet optimization [SPS*17], which failed to make progress due to numerical issues. We view these two mappings as failures. We do not expect the method to perform any better on our more challenging benchmark dataset.

From a refinement perspective, our final results have 1-5% additional triangles. [MPZ14] results have 46-140% additional triangles, which may be simplified further (but it was not addressed in that paper) for the two valid mappings that are suitable for optimization. See table 2 for more details.

Comparing run time, [MPZ14] reports timings of 30 seconds and less for its dataset. In comparison, our run time is slower due to the tracing of our specific seam, which requires substantial refinement when there are many cones.

G.2. Conformal Mapping

In the continuous case, through conformal equivalence of two Riemannian metrics, a flat metric can be prescribed to a disk-topology surface, along with boundary curvature, which ensures local injectivity. In the discrete case, however, we have already seen that local injectivity does not ensure mapping quality. Our experiments, detailed in this section, show that a discrete conformal mapping may result in collapsed triangles due to numerical issues.

Springborn et al. [SSP08] offered Conformal Equivalence of Triangle Meshes (CETM), a method to evolve a discrete metric into a flat one. While the energy is convex and the method is guaranteed to converge (to a flat metric), the resulting metric may be invalid, violating the triangle equality. Sawhney and Crane [SC17] suggested a faster version of CETM but less accurate, treating CETM as a baseline to compare with. We ran CETM on the 1100 small benchmark models, and it failed to produce a valid metric on all of them. This is not surprising since even on [MPZ14]'s dataset it did not perform so well [CLW16].

An improvement was suggested to CETM [CZ17a; CZ17b; CSZZ19] to make it more robust (albeit slower to converge). The idea is to detect in the line search of the Newton method when the triangle inequality is going to be violated (in the spirit of [SS15]) and flip the involved edge as a remedy. We tested this variation

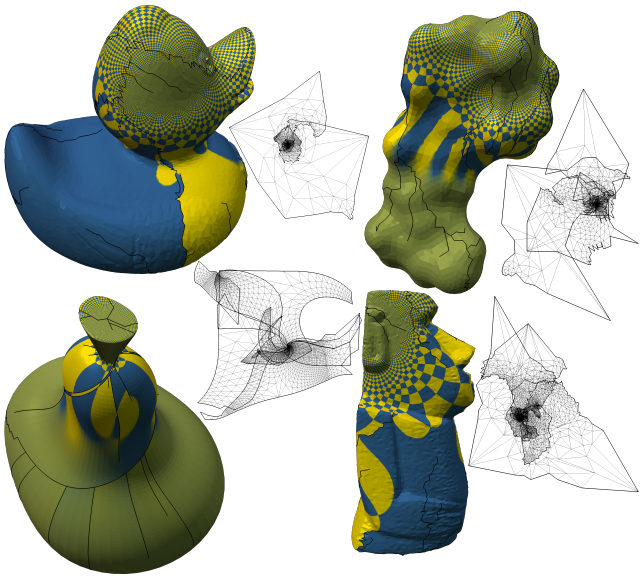


Figure 16: *Conformal mapping. Results of CETM with edge flips [CZ17b; CZ17a; CSZZ19] on four models from the benchmark set (of 1100 models with 50 cones). The mappings contain collapsed triangles (ranging between 16%-51%).*

(using the authors' code) on the 1100 small benchmark models, and it was successful in producing a valid mapping on only 336 of the models. On the rest of the models, the result was a mapping with collapsed triangles (in some cases, more than 50%). Analyzing the collapsed triangles in a failed model with a triangle orientation predicate, only a few of the collapses were pure collapses, but most were inversions, albeit with a negligible negative triangle area (e.g. -10^{-9}). Due to the collapses, the mapping could not be optimized (for lowering isometric distortion [SPS*17]—our main objective) due to numerical issues. In some instances, the method failed to make progress after a handful of iterations, failing to flatten the metric. Figure 16 shows the results of four models, where the mapping is invalid (contains collapsed triangles).

From the results in fig. 16, we can see that the mappings suffer from a scaling issue, which is typical for conformal mapping. More specifically, the same scaling issue was apparent in the successful mappings as well, which also contained triangles that are nearly collapsed—but not severely enough, and they could still be handled by the isometric optimizer.

This implies a fundamental problem in the conformal mapping approach to the problem. Since the conformal mapping is unique, and the approximation of CETM of the smooth case is good enough, we cannot expect from a better (approximation-wise) discrete conformal method to perform much better. That is, the mapping is expected to be badly scaled (even in the continuous case) and would present a numerical challenge to any discrete conformal mapping method. Even if exact numbers are used, once the result is converted to limited precision, numerical issues would surface. Therefore, an optimization method to somehow improve the scaling, which can work with exact numbers, would be needed.

model	cones	F	ours		auxiliary	
			poly	F	poly	F
pear	50	22K	100	52K	2,324	83K
pear	100	22K	200	71K	4,214	102K
pear	200	22K	400	260K	16,304	
bimba100K	50	100K	100	173K	3,128	261K
bimba100K	100	100K	200	184K	5,058	302K
bimba100K	200	100K	400	331K	12,522	

Table 5: *Comparing with an auxiliary-based method. "F": number of facets in the source model and in the result without simplification. "poly": the size of the domain boundary polygon. A missing result is due to a too large polygon for processing the interior mapping on our machine (not enough memory). An alternative for Shor–van Wyck algorithm to decompose the polygon into simple parts is needed for these cases.*

G.3. Combinatorial Methods Using an Auxiliary Mesh

[Gru69; JT73] suggest combinatorial approaches to construct a quad mesh for any given (valid) set of singularities, with the limitation of cone valence ≥ 3. These are theoretical papers that address the question of existence of such quad meshes, and they do not supply explicit algorithms for the construction (which can be involved) nor tests. However, being able to construct such a quad mesh would provide a way to generate seamless parametrization for the given singularities; see appendix H for details.

Recently, Zhou et al. [ZTZC20] offered a combinatorial method, along with an algorithm and results. The results are for generating the boundary polygon only and not the full pipeline. Without improving the arbitrary boundary polygon (appendix D), mapping the interior is bound to result in having nearly-collapsed triangles, which would pose a numerical issue. The idea of [ZTZC20] is also to generate an auxiliary quad mesh (a domain cut mesh), where a self-overlapping polygon that respects given singularities can be extracted from. One limitation is the restriction to cone valence ≥ 2. Another limitation is that the construction of the auxiliary quad mesh is costly. For example, from table 1 in that paper, the thai_statue has 80K faces and is the model with the most cones, 366. The construction of its quad mesh takes 7403 seconds. This is only for the construction of the boundary polygon without interior mapping or further optimization, and it is comparable to the sum of the first three columns in table 4 of our results.

A fundamental drawback of using an auxiliary mesh is that the generated domain boundary polygon is a full and not metapolygon, which makes it substantially larger and more complex. The problem with that is that subsequent operations are badly affected:

- Mapping the interior takes longer and results in a larger mesh. For example, it requires more interior edges for the convex decomposition in appendix E. Tracing the interior edges over the surface is responsible for most of the refinement.
- The algorithm for improving the boundary polygon (appendix D)—which is critical for generating a valid mapping that can be optimized—does not scale well.
- An issue, which can cause a failure to generate a mapping, is that the large boundary polygon is self-overlapping but not sim-

ple. The interior mapping method (appendix E) has a problem decomposing a large polygon into simple parts using the Shor–van Wyck algorithm [SV92]: the algorithm has space complexity $O(n^2)$ and time complexity $O(n^3)$, which limits the size of the polygon.
Since a mapping of the full mesh is available in [ZTZC20], it can be exploited to perform the decomposition into simple parts. However, the decomposition edges will not be simple segments anymore and would consist of inner mesh vertices, further increasing the full polygon size, and exacerbating the previous points. Other alternatives can be considered: i) perform the convex decomposition on the full triangulation, and ii) use dual-harmonic mapping [WZ14]. These alternatives, however, are usually more expensive.

To illustrate these issues, we performed the following experiment. We generated an (experimental) auxiliary parameterized mesh using the first steps of the pipeline, extracted a boundary polygon from it, and mapped the input mesh into this boundary. Compared to [ZTZC20], our (experimental) auxiliary mesh is faster to generate and, more importantly, significantly smaller, resulting in a smaller boundary polygon. Nevertheless, this (experimental) polygon is still unavoidably a full polygon. It is considerably larger than a metapolygon generated by our standard pipeline that constructs a boundary polygon directly on the input mesh. More specifically, the size of a metapolygon is twice the number of cones while the size of a full polygon depends on the mesh size (theorem 1).

Since all the other methods generate an arbitrary polygon (which is similarly a full polygon), our experiment adequately represents the performance of methods that use an auxiliary mesh. The steps of the algorithm used in the experiment:

- Follow the main pipeline (fig. 3) to generate a boundary polygon and an inner mapping for the input mesh without simplification or further optimization. We define the result as the auxiliary mesh, which has the same cones as the input mesh.
- Trace a corresponding seam on both meshes (similar to appendix H). This seam is simpler than the special seam that is traced in the main pipeline (which is required for generating a simple polygon).
- Cut and lay out the auxiliary mesh according to the new seam using the metric of its parametrization (without distorting the current mapping). This is a simple change-of-seam procedure that is allowed due to the seam-invariance nature of a seamless mapping, and both mappings are isometric.
- Extract a self-overlapping boundary polygon from the new layout.
- Improve the extracted (full) boundary polygon (appendix D).
- Cut the input mesh and map it to the polygon (appendix E).

Table 5 provides statistics of the experiment. We use a small and a large model, with different number of cones. First, we note that the resulting mapped mesh from the auxiliary pipeline is larger (and timings were affected accordingly). There are two failures to map into a polygon using the auxiliary-mesh pipeline with 200 cones. The full polygon that was extracted from the auxiliary mesh was too large to be handled by our implementation of Shor–van Wyck algorithm on our machine (not enough memory). The alternative that was suggested for the decomposition into simple parts that would result in a larger polygon is needed for these cases.

Algorithm simplicity. [ZTZC20] is the only paper that outlines a combinatorial algorithm. However, it adopts ideas from [CSZZ19], which needs to be referred to for details. Moreover, parts of its algorithm are not described explicitly, but more in the spirit of our manual construction in B. In contrast, our automatic construction is detailed explicitly as pseudo code or linear problems that can be easily translated to a few lines of Matlab code. Except for the construction, the rest of the pipeline is not addressed in [ZTZC20] and cannot be compared with.

H. Mapping Using an Auxiliary Mesh

We summarize how to utilize a given auxiliary quad mesh with the same desired input singularities to generate seamless parametrization of a triangle mesh with corresponding cones; a similar approach is used in [ZTZC20].

Endow the auxiliary quad mesh with a metric of unit length edges and right angles. Define a seam tree over the quad mesh that spans the singularities and cut the quad mesh along it. Begin by mapping a seed quad facet into a unit square with integer vertex coordinates in the domain. Proceed by laying out neighboring facets, as unit squares, until the whole mesh is parameterized. The parametrization is locally injective, and a weakly self-overlapping boundary polygon that satisfies the input cone angles is extracted from the domain.

A corresponding seam that goes through the cones in the same order—preserving seam edge order around a vertex—is traced over the input mesh. One option to avoid the need to observe the seam edge order around a vertex is to trace the seam as a simple path on the input mesh and a similar one on the auxiliary mesh. This, though, may require (depending on the density of the cones over the mesh) solving a Hamiltonian-path-like problem or alternatively, some refinement of the input mesh. The last step is cutting the input mesh along the seam and mapping it into the boundary polygon of the auxiliary mesh as described in appendix E and proceeding with the rest of the pipeline (section 1.2).

I. The Deficient Set

There are cases where the input field does not contain a set of foundation positive cones with field index sum equals to χ (2 for spheres). That is, there is not a set of positive cones to match any of the 10 sets in table 1. We treat this case by identifying the foundation set after forming the other sets as described next.

When forming the sets using alg. 1, we end up with a single deficient set with a negative total field index that cannot balance any free positive cone. At this point, we break the loop and finish alg. 1 with one *deficient set*. The remaining free positive cones are marked as the foundation set (that do not sum up to χ).

When connecting the sets in section 7, for simplicity (of later proofs), we do the following. One of the copies of the unbalanced (negative) cone in the deficient set will be connected to the end of a negative group chain and to the foundation cone that is associated

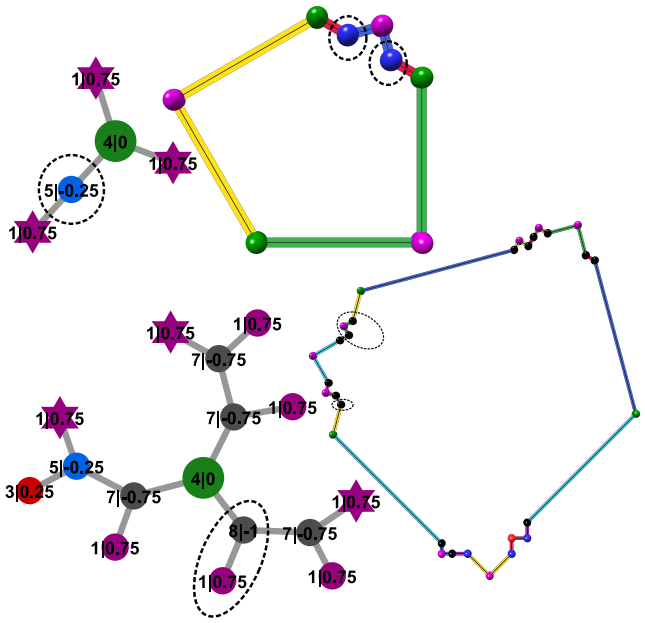


Figure 17: *Two examples of deficient sets. Cones that belong to deficient sets are circled with dashed lines. Left: seam graphs. Right: boundary polygons. Top: a minimal field. The deficient set contains a 5-cone. Compare the deficient set along with the incident foundation cone to the 2-cone in the last foundation polygon in fig. 5. Bottom: the deficient set contains $\{1,8\}$-cones. Note that the set in practice does not need to be incident to a foundation cone. In this example, the maximal absolute sequence angle defect is 45°.*

with the group chain. Intuitively, it is easier to view the unbalanced negative copy in the deficient set and the foundation cone (that the set is connected to) as a single foundation cone—with field index $\frac{1}{2}$ (consisting of a foundation 1-cone and a negative copy with field index $-\frac{1}{4}$, as proved below). In practice, though, no such special handling (of connecting the deficient set in a specific place) is required.

Considering the possible foundation sets and the greedy nature of alg. 1, we note the following for the deficient set case:

- The input field cannot have a 2-cone. Otherwise, any cone configuration that creates a deficient set and accounts for χ would be required to have a set of positive cones that together with the 2-cone would complete one of the 10 foundation sets (a contradiction to requiring a deficient set).
- For the same reason, the input field can have a single 3-cone at most (since two 3-cones have a total field index equals to a 2-cone's field index).
- Therefore, the positive cones in the input field (that requires a deficient set) can be only 1-cones and, possibly, a single 3-cone.
- The deficient set has a total field index of -0.25 and has at most two negative cones.
- The foundation set for this field would contain three 1-cones (with a total field index of 2.25).

In summary, the deficient set occurs only for a very specific input field, where the only positive cones are 1-cones except for, possibly, a single 3-cone. Handling it in practice requires only a minor change (detecting the case and breaking the loop) to alg. 1, which we omitted for clarity. See fig. 17 for deficient-set examples.

J. Proofs

Proposition 1. *For a genus-0 surface, if all cone angles in the layout polygon are 90°-multiple (and twin half-edges have the same length), then the parametrization is seamless.*

Proof. A dual loop ℓ of the 1-ring triangles incident to a vertex u has holonomy equals to the angle defect of u. Consider a seam tree that spans the cones over the surface. A cone leaf in the seam tree has a single incident seam edge e. Since the holonomy of ℓ that crosses e is known, we can assign an integer matching to e (since the angle of u is 90°-multiple).

Next, consider a seam path that leads from u to its parent v in the seam tree. We can propagate the matching on e along the path: each visited vertex on the path would have one seam edge with a known matching, and an integer matching on the other incident seam edge can be deduced from the vertex's field index.

In this fashion, starting from the cone leaves, the matchings are propagated along the seam tree to the root, where in each visited vertex, the propagation proceeds after deducing and setting the last unknown matching on an incident seam edge.

Since all the matchings are integer, seamlessness follows.

Surfaces of higher genera require additional conditions on holonomy angles of homology loops. □

Theorem 1. *Let n be the number of vertices (cone and regular) that a seam graph traverses over the surface. The size of the domain boundary polygon of the cut mesh is $2(n-1)$.*

Proof. Since the seam graph is a tree, it consists of $n-1$ edges. Each edge is cut into two domain twin half-edges. The size of the polygon follows.

Let m be the number of cones in the input. Note that in our construction of a metapolygon, the metaseam spans $m+1$ metavertices (cones plus the additional regular foundation vertex), and thus the metapolygon size is $2m$. In contrast, the size of a full polygon, e.g. one extracted from an auxiliary mesh, would depend on the size of the mesh. □

Theorem 2. *There can be at most three negative cones in each set.*

Proof. The field index is a 0.25-multiple. The maximal (positive) field index is 0.75 (a 1-cone). Before uniting sets, each set can have at most -0.25 field index deficiency (from having a zero field index sum). Therefore, at most, three sets (with one negative cone each) would need to unite to balance a 1-cone. This would be the limit due to the greedy nature of the algorithm. □

Theorem 3. *In the (uncut) seam graph (that was created above), the valence of a negative cone with an angle α is at least c, where $\frac{\alpha}{c} < 360^\circ$.*

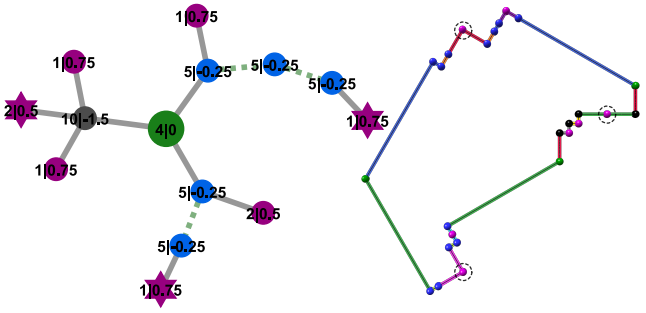


Figure 18: *The construction consists of: a foundation set of* $\{1,2,2\}$*-cones, a set of* $\{10,1,1\}$*-cones, and two united sets (more than one negative cone in each):* $\{2,5,5\}$*-cones and* $\{1,5,5,5\}$*-cones.*

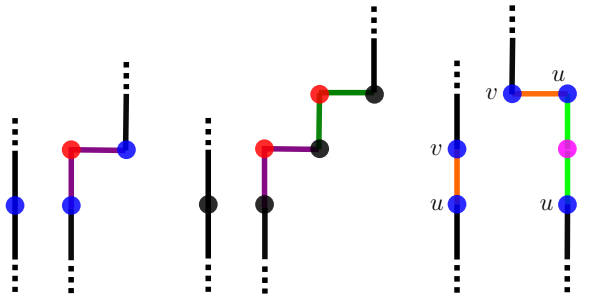


Figure 19: *Illustrating three cases in the proof of theorem 4. Vertical edges are oriented upwards, and the polygon interior is on the left. Left: twin polylines of a set of* $\{3,5\}$*-cones. Middle: twin polylines of a set of* $\{3,3,6\}$*-cones. Right: the minimal united set* $\{2,5,5\}$*.*

Proof. Consider a set with a single negative cone with a field index $I \leq -0.25$ and an angle $\alpha := 360^\circ(1-I)$. Since a positive cone can have at most field index 0.75, the number of positive cones in the set to balance the negative cone is $c \geq -\frac{I}{0.75}$. Besides the positive cones, the negative cone has two other seam graph connections, each to a negative cone or a foundation cone (or the foundation regular vertex). Since each connection results in a new domain copy, the average negative copy angle is:

$$\alpha_c := \frac{\alpha}{c+2} \leq 360^\circ \frac{1-I}{2-\frac{I}{0.75}} = 360^\circ \frac{3-3I}{6-4I} < 270^\circ$$

For a united set with two negative cones, the worst case is two negative cones (possibly, already connected to some positive cones) balancing a 1-cone, or more specifically (if there are not other connected positive cones) the set $\{1,5,6\}$-cones. If the 5-cone is the one that is connected to the 1-cone, then the 6-cone would have only two connections (as being part of a *negative group chain*), and its average domain copy angle would be $540^\circ/2$.

For a united set with three negative cones, the worst case is three negative cones balancing a 1-cone, or more specifically the set of $\{1,5,5,5\}$-cones (fig. 18). Then, two negative cones only have two connections, and their average domain copy angle is $450^\circ/2$.

In the deficient-set case (appendix I), the worst case is an input field that consist of $\{1,5,5,5\}$-cones. In this case, a 5-cone has only two connections, and its average domain copy angle is $450^\circ/2$. □

Theorem 4. *A solution to eq. (2) exists with maximal absolute angle sequence defect (along a foundation polyline)* $t \leq 90^\circ$*, implying* $90^\circ \leq \alpha_i \leq 270^\circ$ *and monotone foundation polylines.*

Proof. A *foundation edge* consists of sets. We prove that the maximal absolute *sequence angle defect* for a set is $\leq 90^\circ$, which would then apply to the whole foundation edge. A foundation edge is a *long negative group chain* with positive cone leaves protruding from it (section 7). Consider first a set with a single negative cone. It has two *negative group chain* connections, and the others are positive cone leaves. In the domain, a foundation edge is split into two twin half-edges (foundation polylines).

We will show a construction that satisfies the *maximal-absolute-sequence-angle-defect* requirement, and it would serve as a possible solution to eq. (2), proving existence.

A set consists of two sequences of domain copies, each on a foundation twin half-edge (polyline). A set has a zero field index sum and $k \cdot 360^\circ$ cone-copy angle sum. We split the set's copy angles between the two sequences such that each sequence would have $(2k-1)180^\circ$, $k \in \mathbb{N}$, assigned to the angles of the set copies in the sequence. We treat each of the two sequences separately to have total angle defect equals to zero.

We note that a positive cone copy (there is only one for each positive cone, and the whole cone angle is assigned to it) is surrounded by two negative copies (since the cone is a leaf in the seam tree that is always connected to a negative cone).

We address, first, a set with a single negative cone. The smallest such set consists of a single pair of cones; there are three possible sets: $\{3,5\}$-cones, $\{2,6\}$-cones, and $\{1,7\}$-cones. See fig. 19 (*left*) for an example of twin polylines of a set of $\{3,5\}$-cones. The set copies are divided into two sequences (one in each foundation twin half-edge). One sequence consists of a single negative copy, which we assign an angle of 180° to. The other sequence has two negative copies surrounding a positive copy. We assign 180° to one copy and the rest of the negative cone angle to the other copy. In the result, copies with 180° do not affect the sequence angle defect, and otherwise a negative copy compensates for the turn (or the angle defect) of an incident positive copy. More generally, to satisfy the requirement that the *maximal absolute sequence angle defect* is $\leq 90^\circ$, the following rule is our guideline: a negative cone copy either does not affect the sequence angle defect (180°), or it compensates for the turn (or angle defect) of a neighbor copy.

Next, consider a larger set with one negative and two positive cones. See the fig. 19 (*middle*) for an example of twin polylines of a set of $\{3,3,6\}$-cones. We add a positive copy followed by a negative copy to the construction of the previous case and follow the guideline when assigning angles to them. That is, the new negative copy would be assigned an angle to compensate for the new positive copy (such that their angle defect sum is equal to zero) while the other copies are assigned angles like in the previous case. In

this fashion, we treat a set with one negative and any number of positive cones (correctness is proved by induction).

Next, we treat a united set with two negative cones and a single positive, e.g. the minimal united set $\{2,5,5\}$ in fig. 19 (*right*). One negative cone, u, is connected to the positive cone, and the other negative cone, v, has no positive leaves (i.e. it has two negative copies). As before, the set copies are divided into two sequences. The first sequence contains one negative copy of each of the two negative cones, which is assigned 180°. The two other copies of u are surrounding the positive cone in the second sequence. As before, one copy is assigned 180°, and the remaining 90° of u's angle are assigned to the last copy. Note that since u cannot balance the positive cone field index by itself, it needs v's copy to balance the last turn. The second copy of v in the second sequence is assigned the rest of the angle, 270°, which balances the turn from the neighboring negative copy of u. The balancing of the turns is guaranteed due to the zero field index sum of the set.

More generally, we divide a set with the two negative cones into two subsets, each with one negative cone and its positive leaves. Each subset is treated as a set with a single negative cone, following the angle assignment that was described previously. However, each subset ends with an extra turn. Both subsets complement each other and balance this turn. The case of a united set with three negative cones is treated in a similar fashion.

One way to view the problem in eq. (2) to gain intuition for the proof is to consider the negative copies as right-turn degrees of freedom (left, right, or keeping straight), which the optimizer utilizes to compensate for the positive cone copy turns (left, right, or straight) in order to minimize the *maximal absolute sequence angle defect*, keeping the polyline monotone. Each positive copy has a negative copy that can compensate for it. We know that the number of negative copy turns can balance the positive copy turns due to the zero field index sum of the set.

In the deficient-set case (appendix I), a deficient set is unbalanced, and one of its (two) sequences would end up with a 90° turn (extra angle defect). We use a nearby foundation cone to balance it. For simplicity (of the proof), when constructing the angle sequences in eq. (2) of the foundation edge that contains the deficient set, we do the following. We set the angle of one of the copies of the unbalanced cone in the deficient set to be the last in the last angle sequence of the foundation edge (i.e. incident to the foundation cone). This way, it does not affect the *maximal absolute sequence angle defect* t in eq. (2). In practice, this extra step is not needed (but it makes the proof easier), and it does not affect the algorithm. □

Theorem 5. *A solution to eq. (3) exists.*

Proof. When the input field contains only positive cones, i.e. the domain polygon is a foundation polygon, we illustrated in fig. 5 possible edge lengths (found using the algorithm in appendix K).

Consider an input field with a single set (besides the foundation set). As previously described, the set copies form two (monotone) sequences in the boundary polygon, each on a foundation twin half-edge. Since the polyline of each sequence is monotone, arbitrary edge lengths can be prescribed without worrying about self-intersection within the polyline. Furthermore, if the foundation edges are long enough (intuitively, making the sets small features on a foundation edge), a set polyline will not intersect (globally) any other part of the polygon.

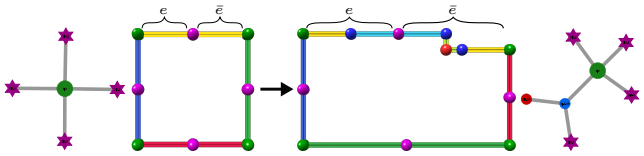
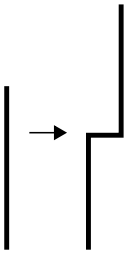


Figure 20: *The foundation polygon on the left has four foundation edges: left, top, right, and bottom. The top foundation edge consists of two twin foundation polylines e and $\bar{e}$. Since they do not have sets on them, seamlessness constraints apply to them. On the right, we added to the polygon a set of $\{3,5\}$-cones. In order to accommodate the set, e and $\bar{e}$ are not of the same length anymore, and seamlessness constraints do not apply to them (but to their segment sequences instead). The right foundation edge can change its length to compensate for the set addition to the top edge while the bottom and left foundation edges can remain intact (although, in this example, the bottom foundation edge has changed as well). (Edge color is arbitrary and does not correspond between the two polygons.)*

Due to the positive cone leaves, the length of the two (twin) sequences is different in general. This means that before they are added to one of the foundation edges in a foundation polygon, the foundation twin half-edges have the same length, and after the addition, they do not.

It is easy to see that in every polygon in fig. 5, (at least) one foundation half-edge can change its length arbitrarily (ignoring the seamlessness constraint for the edge—which would now be applied instead to its inner sequences) while keeping the rest of the polygon valid (same angles and seamlessness constraints hold for the rest of edges). More specifically, there is another pair of twin half-edges in the polygon that changing their length would compensate for the modified length of the first pair while keeping the rest of the polygon intact and valid. This allows us to add a set to the foundation edge, where its twin foundation polylines would not have the same length anymore. See fig. 20 for illustration.



A more thorough feasibility test is to replace in each of the 10 foundation polygons one twin polyline (which is just a straight segment in a foundation polygon) with the dummy polyline in the inset (which is longer and wider—in terms of its bounding box). Then, using a *brute-force* algorithm (appendix K), solve for polygon lengths such that the dummy polyline is strictly longer than its twin and seamlessness constraints apply to all other foundation edges as usual. For example, consider replacing the foundation polyline $\bar{e}$ in the foundation polygon on the *left* in fig. 20 with a dummy polyline.

Adding more sets to the same foundation edge does not affect the argument. Adding all the sets to one foundation edge is enough for the algorithm correctness (keeping the proof simple). In prac-

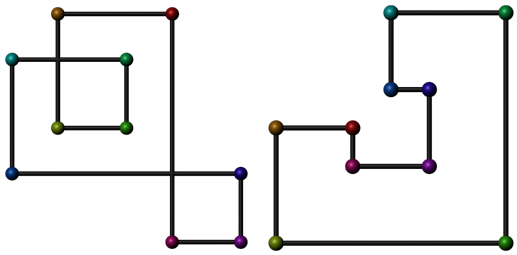


Figure 21: *Solving intersections of the left polygon by changing the edge lengths, e.g. the polygon on the right. Vertex correspondence between the two polygons is color coded. Polygon angles (or edge directions) are preserved. In this illustration, there are no seamlessness constraints.*

tice, distributing the sets between the foundation edges (which improves the results in later stages of the pipeline) still produces a valid polygon.

□

K. Non-intersection Constraints

Consider the problem in section 9 of finding the vertex coordinates (or edge lengths) of a simple polygon P, given its angles (or edge directions). A related problem is: given a self-intersecting polygon, find new edge lengths that produce a simple polygon with the same angles; see fig. 21 for an illustration. For small input fields, such as those that contain only positive cones (see table 1), using brute force (an MI solver) to solve the problem becomes practical. We formulate linear non-intersection constraints using binary variables that can be added to a given problem.

Consider two (non-consecutive) polygon edge segments p and q:

$$\begin{aligned} p(s) &= p_0 + su,\ s \in [0, l_p] \\ q(t) &= q_0 + tv,\ t \in [0, l_q] \end{aligned} \quad ,$$

where $p_0, q_0 \in \mathbb{R}^2$ are the beginning of the edges, l_p and l_q are the edge lengths, and $u, v \in \mathbb{R}^2$ are the given edge directions (unit column vectors). Let $A := \begin{bmatrix} u & -v \end{bmatrix}$, $A \in \mathbb{R}^{2\times 2}$. We find the point of intersection of the supporting lines of the two segments by solving $p(s) = q(t)$:

$$A \begin{bmatrix} s \\ t \end{bmatrix} = q_0 - p_0 \quad .$$

Theorem 6. *If p and q are parallel ($det(A) = 0$) and intersecting (i.e. overlapping), then there exists a (different) pair of segments in P that also intersect but are not parallel.*

Proof. Let $\bar{p}$ be the (maximal) line segment that consists of p and all the consecutive segments that are parallel to it. Similarly, define $\bar{q}$. Since p and q intersect and they are not incident, $\bar{p} \neq \bar{q}$, or else P has a vertex with an absolute angle defect 180° ($> 90^\circ$), where one incident edge points to the opposite direction of the other incident edge. The intersection (or overlap) between the two segments $\bar{p}$ and $\bar{q}$ must contain one of the four ends, a vertex η. WLOG, $\eta \in \bar{p}$. Since $\bar{p}$ and $\bar{q}$ are not incident, $\eta \in r$ for some different segment r in P. Since $\bar{p}$ is maximal, r is not parallel to $\bar{p}$. Then, r intersects one of the *subsegments* of $\bar{q}$ (at η) and is not parallel to it. □

Corollary 2. *If each non-parallel (non-consecutive) pair of edge segments in P does not intersect, then P is simple.*

Therefore, if u and v are parallel, then there is no need to check if they are intersecting. Hence, we proceed with the assumption that they are not parallel and A is invertible. We have:

$$\begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} u & -v \end{bmatrix}^{-1} (q_0 - p_0) \quad .$$

One of the following needs to hold to ensure there is no intersection:

$$\begin{aligned} s &< -\varepsilon_1 \\ s &> l_p + \varepsilon_1 \\ t &< -\varepsilon_1 \\ t &> l_q + \varepsilon_1 \end{aligned} \quad ,$$

where $\varepsilon_1 \geq 0$ is a small constant. We formulate this condition using MI (mixed-integer) constraints that add real variables s and t and four binary variables $b_i \in \{0, 1\}$ to the problem:

$$\begin{aligned} s &\leq b_1 \mu - \varepsilon_2 \\ l_p + \varepsilon_2 &\leq s + b_2 \mu \\ t &\leq b_3 \mu - \varepsilon_2 \\ l_q + \varepsilon_2 &\leq t + b_4 \mu \\ b_1 + b_2 + b_3 + b_4 &\leq 3 \end{aligned} \quad ,$$

where μ is a large constant that bounds the maximal value of s and t (we used $\mu = 10^5$), and $\varepsilon_2 \geq 0$ is a small constant.

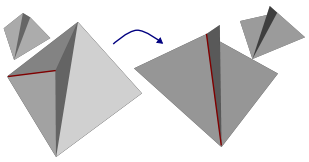
In situations such as the two 3-cones in the 9th polygon in fig. 5, we not only want to prevent intersection, but we further want to maintain some minimal distance between the cones. ε_1 and ε_2 control the padding between the segments, where ε_1 is used to detect intersections and ε_2 is used to prevent them (we used $\varepsilon_1 = 0.4$ and $\varepsilon_2 = 2$).

We employ the non-intersection constraints as follows. For a given input, we begin by solving eq. (3). Then, we check for intersections between all (non-consecutive, non-parallel) pairs of edge segments, add a constraint (along with the necessary auxiliary variables) to the problem for each intersecting pair, and solve again. We iterate this process until there are no intersections. Since we proved polygon existence, the process is guaranteed to converge to a feasible solution.

We used [Gur18] to solve the MILP (mixed integer linear programming) problem that results from adding non-intersection constraints to eq. (3). We used this approach to generate the foundation polygons in fig. 5, which took a fraction of a second. It is not practical to use this MI approach on problems with many integer variables, which is why we took care to generate monotone polylines (that cannot self-intersect) for the non-foundation sets in the input field.

L. Implementation Tips

Edge manifoldness. Most simple operations on a 3D surface, such as edge contraction, an edge flip, or triangulating a polygon, require a preliminary test that every edge involved in the operation remains manifold. A requirement from a manifold edge is that it is shared by two faces at the most.

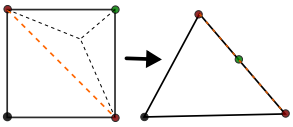


However, consider flipping the red edge in the inset. While technically it remains shared by only two facets, its two incident vertices now have more than one edge in common. Loading such a mesh into a DCEL structure using a popular library such as [CGA20] would pose a problem.

Therefore, when checking for edge manifoldness, one needs to check the stricter requirement that no two vertices share more than a single edge after contraction. It is also possible to test the link conditions [DEGN98; TNB10] before contracting an edge.

When applying similar operations on a 2D mapping of the surface, it is enough to keep the mapping locally injective to guarantee manifoldness.

Tutte embedding. One of the requirements for Tutte embedding is that the graph is 3-connected. However, this requirement is too strict, and a more relaxed and sufficient requirement is that no dividing edge (connecting two non-consecutive boundary vertices) is mapped wholly into one side of the boundary polygon ([Flo03], theorem 6.1).



In practical terms, given a dividing edge between two boundary vertices p_i and p_j, one needs to test the mapping of the two boundary polylines between p_i and p_j (one polyline in each direction). Testing the mapping of a polyline can be done by choosing an inner vertex, e.g. p_{i+1} for the CCW polyline (or p_{i-1} for the CW polyline), and verifying that the mapping of p_i, p_{i+1}, and p_j is not collinear.

For example, the orange dividing edge in the inset violates the condition since its two blue vertices along with the inner pink vertex are mapped to a straight line. There is no need for such a test when mapping to a strictly convex polygon, e.g. sampled from a circle.

A suggested remedy for an edge that violates the condition is to split the edge (insert a new vertex at its middle). Normally, a graph does not contain many dividing edges, and the difference between splitting all dividing edges or splitting only required edges to maintain 3-connectedness is insignificant. However, since we kept the refinement in appendices C and E relatively minimal, the number of dividing edges (that split narrow passages—strips of triangles—between seam paths) is more substantial than usual while only a few of them violate the condition and require splitting.

References

[APL14] AIGERMAN, NOAM, PORANNE, ROI, and LIPMAN, YARON. "Lifted bijections for low distortion surface mappings". *TOG* 33.4 (2014), 69 7.

[BZK09] BOMMES, DAVID, ZIMMER, HENRIK, and KOBBELT, LEIF. "Mixed-integer Quadrangulation". *ACM Trans. Graph.* 28.3 (2009), 77:1–77:10 7.

[CGA20] CGAL. *CGAL User and Reference Manual*. 2020. URL: https://www.cgal.org/ 6, 14.

[CLW16] CHIEN, EDWARD, LEVI, ZOHAR, and WEBER, OFIR. "Bounded Distortion Parametrization in the Space of Metrics". *ACM Trans. Graph.* 35.6 (2016), 215:1–215:16 5, 7.

[CSZZ19] CAMPEN, MARCEL, SHEN, HANXIAO, ZHOU, JIARAN, and ZORIN, DENIS. "Seamless parametrization with arbitrary cones for arbitrary genus". *ACM Transactions on Graphics* 39.1 (2019), 1–19 7–9.

[CZ17a] CAMPEN, MARCEL and ZORIN, DENIS. "On Discrete Conformal Seamless Similarity Maps". *arXiv preprint arXiv:1705.02422* (2017) 7, 8.

[CZ17b] CAMPEN, MARCEL and ZORIN, DENIS. "Similarity maps and field-guided T-splines: a perfect couple". *ACM Transactions on Graphics (TOG)* 36.4 (2017), 1–16 7, 8.

[DEGN98] DEY, TAMAL K, EDELSBRUNNER, HERBERT, GUHA, SUMANTA, and NEKHAYEV, DMITRY V. "Topology preserving edge contraction". *Publ. Inst. Math.(Beograd)(N.S.)* 1998 14.

[Flo03] FLOATER, MICHAEL. "One-to-one piecewise linear mappings over triangulations". *Mathematics of Computation* 72.242 (2003), 685–696 14.

[Gru69] GRUNBAUM, BRANKO. "Planar maps with prescribed types of vertices and faces". *Mathematika* 16.1 (1969), 28–36 1, 8.

[Gur18] GUROBI. *Gurobi Optimizer Reference Manual*. 2018. URL: http://www.gurobi.com 13.

[JT73] JUCOVIČ, E. and TRENKLER, M. "A theorem on the structure of cell–decompositions of orientable 2–manifolds". *Mathematika* 20.1 (1973), 63–82 8.

[Lev21] LEVI, ZOHAR. "Direct Seamless Parametrization". *ACM Transactions on Graphics* 40.1 (2021), 1–14 7.

[Lip12] LIPMAN, YARON. "Bounded distortion mapping spaces for triangular meshes". *ACM Transactions on Graphics (TOG)* 31.4 (2012), 108 7.

[LYNF18] LIU, LIGANG, YE, CHUNYANG, NI, RUIQI, and FU, XIAO-MING. "Progressive parameterizations". *ACM Transactions on Graphics* 37.4 (2018), 1–12 6.

[LZ14] LEVI, ZOHAR and ZORIN, DENIS. "Strict minimizers for geometric optimization". *ACM Transactions on Graphics (TOG)* 33.6 (2014), 185 7.

[MPZ14] MYLES, ASHISH, PIETRONI, NICO, and ZORIN, DENIS. "Robust Field-aligned Global Parametrization". *ACM Trans. Graph.* 33.4 (2014), 135:1–135:14. ISSN: 0730-0301 2, 7.

[RVLL08] RAY, NICOLAS, VALLET, BRUNO, LI, WAN CHIU, and LÉVY, BRUNO. "N-symmetry direction field design". *ACM Transactions on Graphics* 27.2 (2008), 10 1.

[SC17] SAWHNEY, ROHAN and CRANE, KEENAN. "Boundary first flattening". *ACM Transactions on Graphics* 37.1 (2017), 1–14 7.

[SJZP19] SHEN, HANXIAO, JIANG, ZHONGSHI, ZORIN, DENIS, and PANOZZO, DANIELE. "Progressive embedding". *ACM Transactions on Graphics* 38.4 (2019), 32 6.

[SPS*17] SHTENGEL, ANNA, PORANNE, ROI, SORKINE-HORNUNG, OLGA, et al. "Geometric Optimization via Composite Majorization". *ACM Trans. Graph.* 36.4 (2017) 6–8.

[SS15] SMITH, JASON and SCHAEFER, SCOTT. "Bijective Parameterization with Free Boundaries". *ACM Trans. Graph.* 34.4 (2015), 70:1–70:9 7.

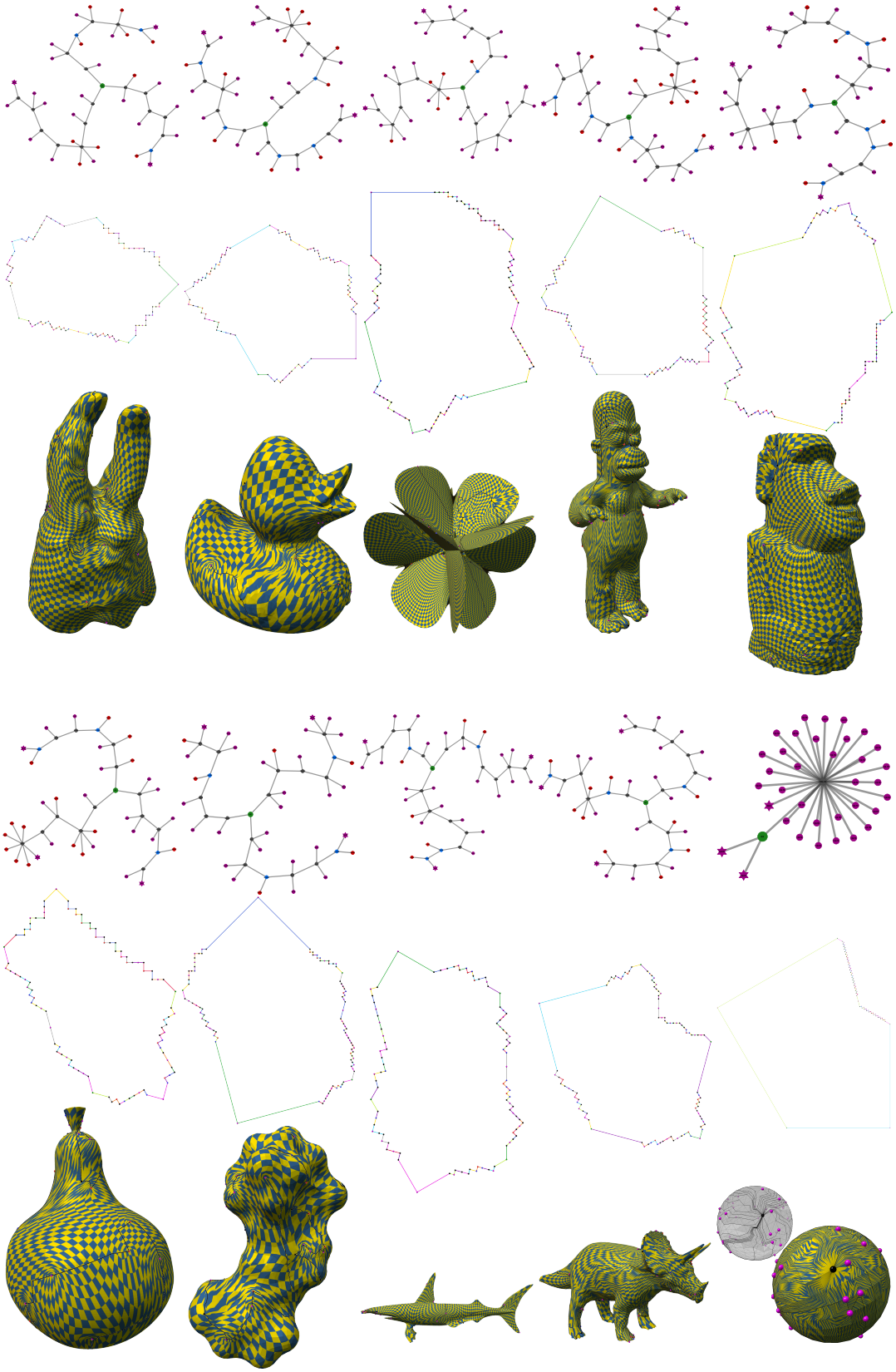
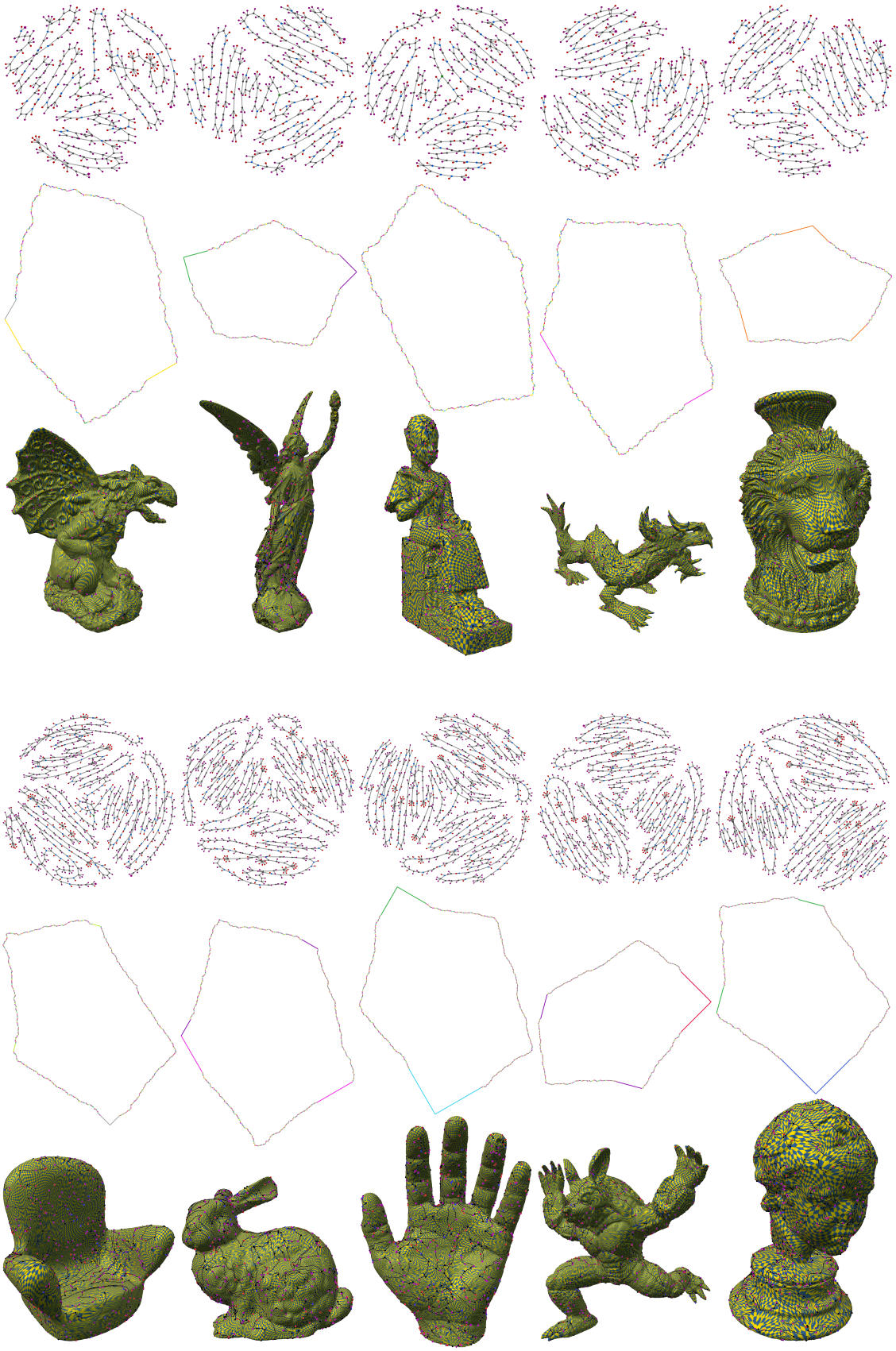


Figure 22: *A sample of each of 9/11 benchmark models and a sphere with a 100-cone (-24 field index). Rows: the seam graph, boundary polygon, and final result.*



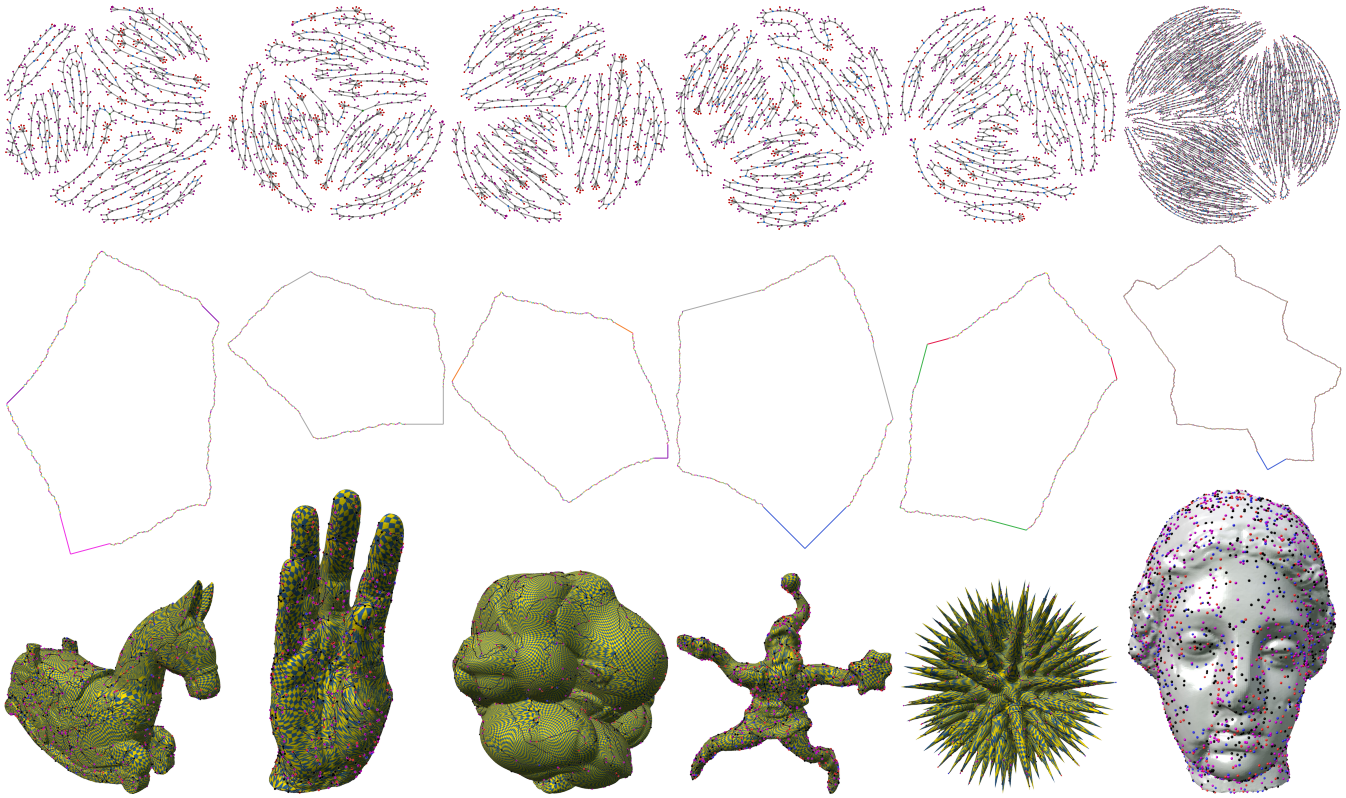


Figure 23: *Larger models. Rows: the seam graph, boundary polygon, and final result.*

[SSP08] SPRINGBORN, BORIS, SCHRÖDER, PETER, and PINKALL, ULRICH. "Conformal equivalence of triangle meshes". *TOG* 27.3 (2008), 77 7.

[SV92] SHOR, P.W. and VAN WYK, C.J. "Detecting and decomposing self-overlapping curves". *Computational Geometry* 2.1 (1992), 31–50 5, 9.

[TNB10] THOMAS, DILIP MATHEW, NATARAJAN, VIJAY, and BONNEAU, GEORGES-PIERRE. "Link conditions for simplifying meshes with embedded structures". *IEEE Transactions on Visualization and Computer Graphics* 17.7 (2010), 1007–1019 14.

[WB06] WÄCHTER, ANDREAS and BIEGLER, LORENZ T. "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". *Mathematical programming* 106.1 (2006), 25–57 5.

[WZ14] WEBER, OFIR and ZORIN, DENIS. "Locally injective parametrization with arbitrary fixed boundaries". *TOG* 33.4 (2014), 75 5, 6, 9.

[ZTZC20] ZHOU, JIARAN, TU, CHANGHE, ZORIN, DENIS, and CAMPEN, MARCEL. "Combinatorial Construction of Seamless Parameter Domains". *Computer Graphics Forum*. Vol. 39. 2. 2020, 179–190 8, 9.