



Robust curve embedding in inconsistent surface meshes

Alexander Schier^{*} , Reinhard Klein

University Bonn, Bonn, Germany

HIGHLIGHTS

- Polygon soups and meshes without connectivity can be assigned a continuous texture map.
- Long curves from a connected proxy can be embedded across disjoint polygons.
- Only minimal, targeted face splits are required to represent seams accurately.
- Unstable numerical predicates are avoided by using local topology information.

ARTICLE INFO

Editor: Konrad Polthier

Keywords:

Curve embedding
Mesh proxies
Geodesic paths
UV seams
Texture transfer
Polygon soups

ABSTRACT

We present a numerically robust algorithm for augmenting inconsistent meshes and polygon soups with curves defined on a mesh proxy. Unlike prior approaches, which transfer only vertex positions to the proxy, our method represents input edges on the proxy by geodesic paths and merges them with proxy-defined edge paths such as UV seams. This enables reliable detection of intersections between geodesic paths and proxy seams and avoids explicit edge-edge intersection tests, which makes the construction stable even on poorly conditioned meshes. We demonstrate the method in texture transfer, where projected input faces are cut along UV seams of the proxy so that the reconstructed result mesh reproduces the proxy seam layout explicitly and avoids distortions that previous approaches could only mitigate heuristically. To construct this result mesh, we first build an overlay curve network consisting of geodesic segments and proxy seam edges, and then reconstruct a manifold mesh aligned with the input mesh. Our experiments show that this reconstruction removes seam artifacts and reduces the associated high-distortion outliers of prior heuristic methods. Beyond UV seams, the construction can be used more generally to transfer proxy-defined curves onto disjoint meshes, such as collections of NURBS patches.

1. Introduction

Parameterizing three-dimensional surface meshes into a two-dimensional UV space is essential for many applications, including visualization, rendering, remeshing, animation, and games. Such a parameterization assigns two-dimensional face-vertex UV coordinates so that each vertex corresponds to a location in the UV plane, enabling textures to be interpolated across mesh faces. For most parameterization algorithms, the mesh must first be segmented into patches that can be embedded in 2D by defining UV seams. In addition, further UV seams may be introduced within a patch to reduce texture distortion. Creating UV parameterizations for such patches is well-studied, but most automatic parameterization algorithms are not suited to handle inconsistent meshes, i.e., meshes with incomplete or missing connectivity between faces.

^{*} Corresponding author.

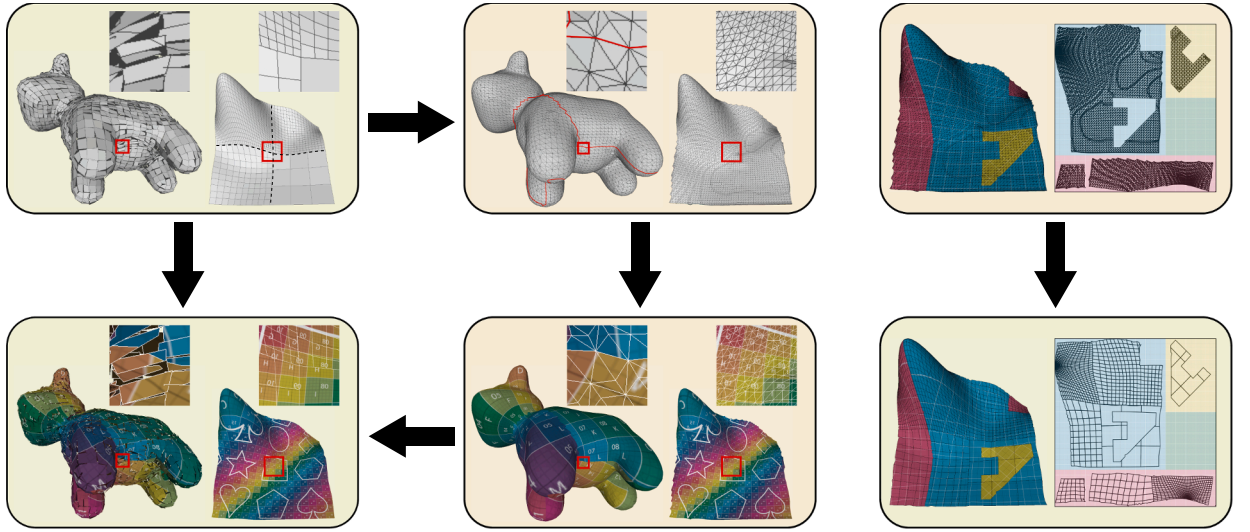
Email addresses: schier@uni-bonn.de (A. Schier), rk@cs.uni-bonn.de (R. Klein).

<https://doi.org/10.1016/j.cagd.2026.102588>

Received 30 January 2026; Received in revised form 10 June 2026; Accepted 16 June 2026

Available online 17 June 2026

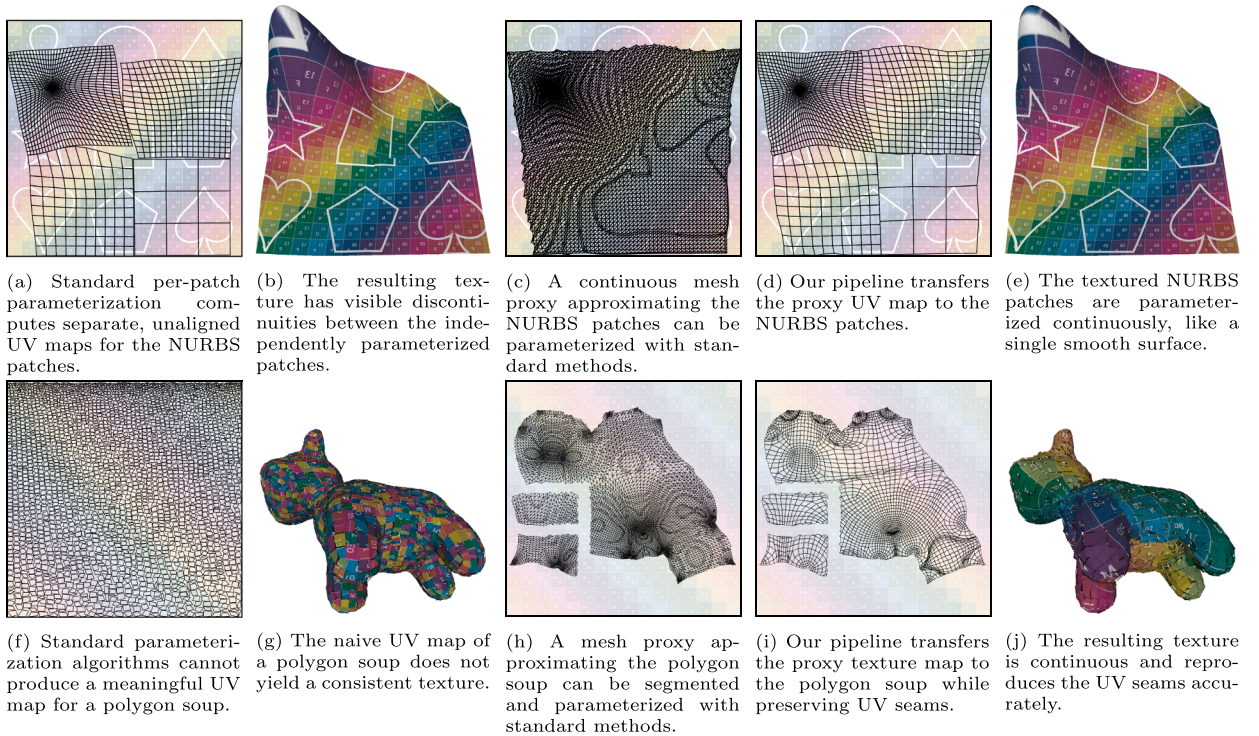
0167-8396/© 2026 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).



(a) Given an inconsistent geometry, such as a polygon soup or a collection of NURBS patches with discontinuous tessellations, we can create a consistent texture map by first computing a UV map for a mesh proxy and then projecting this map together with the UV seams defined on the proxy back onto the original mesh.

(b) As we embed seams in the mesh instead of only defining them in the parameterization, we can merge different meshes or curve networks.

Fig. 1. Our method allows texturing inconsistent geometries using a mesh proxy that is simple to texture. Beyond texturing, the algorithm can be used to merge arbitrary curve networks on a given carrier surface.



(a) Standard per-patch parameterization computes separate, unaligned UV maps for the NURBS patches.

(b) The resulting texture has visible discontinuities between the independently parameterized patches.

(c) A continuous mesh proxy approximating the NURBS patches can be parameterized with standard methods.

(d) Our pipeline transfers the proxy UV map to the NURBS patches.

(e) The textured NURBS patches are parameterized continuously, like a single smooth surface.

(f) Standard parameterization algorithms cannot produce a meaningful UV map for a polygon soup.

(g) The naive UV map of a polygon soup does not yield a consistent texture.

(h) A mesh proxy approximating the polygon soup can be segmented and parameterized with standard methods.

(i) Our pipeline transfers the proxy texture map to the polygon soup while preserving UV seams.

(j) The resulting texture is continuous and reproduces the UV seams accurately.

Fig. 2. Examples of how standard automatic parameterization algorithms handle the inconsistent meshes from Fig. 1 compared with the results of our pipeline.

Two examples of inconsistent meshes are shown in Fig. 2. First, meshes may consist of multiple disconnected components due to the way they are generated. For example, when NURBS patches are adaptively tessellated at varying resolutions according to local curvature, they cannot easily be joined into a single mesh even though they represent a single smooth surface. Second, some meshes provide little or no topological connectivity at all and may have significant geometric noise, which is common in 3D scanned data and other imperfect acquisition pipelines. Despite the lack of reliable connectivity, these meshes often have sufficient geometric structure to support meaningful texturing, but conventional parameterization algorithms cannot produce consistent mappings, as these methods typically minimize deformation energies that depend explicitly on mesh connectivity. Without consistent connectivity, the methods break down, as illustrated in Fig. 2 for our two example meshes. Independently parameterized NURBS patches cannot be aligned with each other due to different mesh resolutions and non-smooth changes in deformation along boundaries (Fig. 2(a) and (b)), and applying standard parameterization algorithms to a polygon soup results in each face being embedded independently, making the texture coordinates meaningless (Fig. 2(f) and (g)). One widely used approach for handling meshes with inconsistent connectivity is to construct a consistent mesh proxy that approximates the input and can be parameterized using standard algorithms. In some applications, this proxy can replace the original mesh, but in others one would like to retain the original mesh and only use the proxy to compute a consistent parameterization. It is therefore desirable to transfer the UV coordinates computed on the proxy (cf. Fig. 2(c) and (h)) to the input mesh (Fig. 2(e) and (j)).

In 2007, Degener and Klein proposed an approach for texture transfer using electrostatic field lines, which, unlike distance-based mappings, can guarantee injectivity, and in 2018, Schier et al. (2018) extended that approach to monopole field lines that allow to use different mesh proxies without recomputing the field lines. Both approaches provide injective and flip-free correspondences between the vertices of the inconsistent geometry and an enclosing parameterized mesh proxy and thereby allow parameterizing the inconsistent input mesh using texture coordinates of that mesh proxy. By only projecting vertices onto the mesh proxy, neither method accounts for UV seams directly, so they need to rely on heuristics instead. Degener and Klein used a refinement strategy that minimized the distortion at the cost of changing the resulting geometry and increasing the number of faces, while Schier et al. avoided geometry modification by using heuristics in UV space to clamp problematic triangles to UV charts, mitigating the issue at the cost of increased texture deformation near UV seams.

Our work proposes to replace these heuristics by cutting faces along UV seams projected back from the mesh proxy to the input mesh. Unlike Degener and Klein (2007), we do not refine until the error is below a threshold, but instead compute the edge paths needed to split faces so that each sub-face is contained in exactly one UV chart, making linear interpolation within each sub-face well defined. Point correspondences at the vertices alone do not determine how an input edge is represented on the proxy surface. Prior work uses linear interpolation in UV space between the projected endpoints of the edge, but the resulting straight path in UV space may correspond to an unintended path on the surface, for example a wrap-around path across a seam. We instead represent each input edge explicitly by the shortest geodesic between its projected endpoints on the proxy surface. For meshes with sufficiently fine resolution, i.e., edge lengths small compared to local feature size and seam length, this yields the intended path in practice.

While we demonstrate our approach in the context of texture coordinate transfer between meshes, the underlying contribution of a geodesic edge projection is more general, as we define a numerically robust method for projecting curves as piecewise geodesics onto a target surface. The potential applications beyond texture mapping include CAD, CAE and CAM applications. For instance, in CAD, it could be used to insert design curves from one mesh into another at their intersection or contact surface. In CAE, it could facilitate the transfer of boundary conditions across non-matching meshes. In CAM, our approach could be used to embed prescribed cutting curves or tool paths directly into a mesh surface.

Key contributions

- **Curve embedding:** Given a polygonal input mesh (possibly with inconsistent connectivity) and a consistent mesh proxy with similar geometry we map the inconsistent input to the consistent proxy and merge it with seam edges defined on the proxy mesh.
- **Interpolation of discontinuous surface properties:** Our scheme enables piecewise-linear interpolation of surface properties (such as UV coordinates) that are discontinuous at seams.
- **Not requiring connectivity:** The curves of the mesh proxy are incorporated into the faces without requiring connectivity between faces. As a result, long curves embedded in the mesh are consistent even across disjoint mesh components.
- **Robustness:** Our method minimizes the use of numerically fragile geometric predicates by exploiting the topological information of the mesh proxy to increase stability on poorly conditioned meshes.
- **Polygonal Meshes:** In contrast to prior work that effectively operates on triangle soups, our approach can handle and preserve polygonal faces.
- **Connectivity in UV space:** Prior approaches assigned unique face-vertex UV coordinates to each face, whereas our approach uses shared UV vertices except at discontinuities.
- **Fully automated pipeline:** We provide an end-to-end pipeline that automatically constructs a proxy, computes a parameterization, and transfers texture coordinates from a watertight proxy to inconsistent input meshes, without manual intervention.

2. Related work

We provide an overview of work related to the main components of our pipeline. We highlight in each section the choices for our algorithm and give a rationale for them.

Mesh proxies

The simplest way to create a mesh proxy is to triangulate a signed distance function using an algorithm like Marching Cubes (Lorensen and Cline, 1987) or Dual Contouring (Schaefer et al., 2007) and adaptive variants thereof (Shu et al., 1995; Greß and Klein, 2004). The survey on the Marching Cubes algorithm by Newman and Yi (2006) provides a comprehensive overview of extensions to the classic algorithm, and the survey by Büscher et al. (2024) gives a broader overview of methods for triangulating isosurfaces. In addition, cage-based methods such as nested cages (Sacht et al., 2015) can be useful in our setting, since field line correspondences can be queried against multiple enclosing proxies without recomputing the field. A recent survey on cage-based methods is provided by Ströter et al. (2024). For distorted or disconnected inputs, robust inside-outside information is also often obtained using generalized winding numbers and their fast approximation (Jacobson et al., 2013; Barill et al., 2018), and watertight proxy reconstruction is commonly addressed by Poisson-type methods (Kazhdan et al., 2006; Kazhdan and Hoppe, 2013). Alternative proxy constructions emphasize coarse or bounding approximations (Yan et al., 2012; Calderon and Boubekur, 2017; Xian et al., 2009; Casti et al., 2019), and the work of Shen et al. (2004) addresses mesh reconstruction from polygon soups directly. In our experiments we use Dual Contouring to generate a simple distance-based offset mesh.

Mesh parameterization

The surveys (Sheffer et al., 2007; Hormann et al., 2008) provide an overview of well-known parameterization methods. More recent work includes distortion-minimizing and locally injective optimization methods such as AMIPS by Fu et al. (2015), inversion-free mapping by simplex assembly (Fu and Liu, 2016), global parameterization by incremental flattening by Myles and Zorin (2012), the scalable locally injective mappings of Rabinovich et al. (2017), and practical construction of globally injective parameterizations with positional constraints (Wang et al., 2023). Discrete conformal approaches based on Ricci, Yamabe, and Calabi flow have also been studied extensively (Chow and Luo, 2003; Luo, 2004; Jin et al., 2008; Zhao et al., 2018; Zhang et al., 2014), along with discrete conformal equivalence methods (Springborn et al., 2008; Gillespie et al., 2021). Other recent developments include seamless parameterization (Zohar, 2021; Levi, 2022, 2023) and rectangular surface parameterization (Corman and Crane, 2025). For our experiments we use Boundary First Flattening (Sawhney and Crane, 2017) due to its speed, robust practical behavior, and available implementation. Other choices are possible, as our algorithm only depends on the mesh proxy carrying a texture atlas that can be transferred to the input mesh or polygon soup.

Mesh segmentation

Many parameterization methods depend on a segmentation of the mesh beforehand, for example to cut it into genus-zero or disk-like patches with boundaries before flattening. Surveys on the topic include the mesh segmentation survey by Shamir (2008), the review by Rashad et al. (2017) for classical methods, and the survey by Rodrigues et al. (2018) on part-based mesh segmentation. In recent years many geometric deep learning approaches have been proposed for mesh segmentation (e.g., Hanocka et al., 2019; Gong et al., 2019; Sharp et al., 2022), although these methods often focus on semantic segmentation rather than the placement of UV chart boundaries. Methods explicitly targeting seam placement for UV mapping include SeamCut by Lucquin and Boubekur (2017), AutoCuts by Poranne et al. (2017), and OptCuts by Li et al. (2018). Learning-based approaches for computing UV seams include GraphSeam (Teimury et al., 2020), DA Wand (Liu et al., 2023), and SeamGPT (Li et al., 2025). We use the cone-based segmentation provided by the reference implementation of Sawhney and Crane (2017) as a readily available preprocessing step, although it is not generally optimized for UV chart quality. For some of our examples, we additionally insert manual seams to split the parameterization into multiple charts. Any segmentation suitable as a preprocessing step for UV parameterization could be used instead, and in practice the choice of segmentation and parameterization should be matched to the class of objects being processed.

Point correspondences

For surveys on classical methods for point correspondences see van Kaick et al. (2011) and Sahillioglu (2020). Recent deep learning-based point cloud registration methods are covered by Zhang et al. (2025). For our algorithm, we build on the monopole field line correspondences of Schier et al. (2018), which extend the field line approach to texture transfer introduced earlier by Degener and Klein (2007). Degener et al. use dipole fields to compute injective correspondences between a triangle soup and a mesh proxy surface, whereas Schier et al. precompute a monopole field on the input surface, which allows correspondences to multiple enclosing proxies to be queried by intersecting the same field lines without recomputing the field. Related electrostatic constructions also appear in approximating winding numbers (Barill et al., 2018), where the winding-number field is expressed as an area-weighted sum of dipole contributions. The monopole approach is also related to harmonic parameterization by electrostatics (Wang et al., 2013), but while the electrostatic potential and its gradient field define a harmonic coordinate system for the embedding space parameterized by a field line and an isocontour, an enclosing offset mesh that is not aligned with an isocontour of the field may be intersected multiple times by a single field line, so the induced surface correspondences are generally injective, but not surjective.

Geodesics

Geodesic distances on polyhedral surfaces are well-studied. An important foundation for exact geodesics is the continuous Dijkstra algorithm of Mitchell et al. (1987), which was later improved using upper bounds and distance estimates (Chen and Han, 1990; Xin and Wang, 2009). Another important direction is approximating geodesic distances using fast marching methods introduced by

Sethian (1996), which model the geodesic distance as a monotonically propagating wavefront. Novotni and Klein (2002) extended this idea by propagating the wavefront using circles around vertices with known distances, and Tang et al. (2007) described an explicit construction of the corresponding geodesic path. Ying et al. (2013) propose a saddle vertex graph that splits the path into segments having the saddle vertices as breakpoints, so that each segment does not pass through saddle vertices. Crane et al. (2013b) showed how to compute approximate geodesics by solving the heat equation, which allows for a speedup by pre-factoring the Laplacian when computing geodesics from many different source points, and Tao et al. (2019) showed how to parallelize this approach. Another direction consists of methods that manipulate the mesh or an intrinsic triangulation directly, such as the selective refinement approach of Kanai and Suzuki (2001), the edge-flip algorithm of Sharp and Crane (2020), and b/Surf by Mancinelli et al. (2023), which traces locally shortest paths on surface meshes. Further methods are discussed in the surveys (Peyré et al., 2010; Bose et al., 2011; Crane et al., 2020). We use the edge-flip algorithm of Sharp and Crane (2020) because it directly provides an ordered sequence of intersection points between the geodesic path and the mesh on which it is computed, which is a core ingredient of our edge-tracing algorithm. Methods that return only distances would require an additional path-extraction step to recover edge intersections, while exact path-tracing methods can be more expensive when many paths have to be recovered explicitly.

3. Motivation

Before presenting our method, we introduce the texture transfer approach of previous work and show how texture interpolation can fail when relying only on vertex correspondences, even when using the proposed UV space correction approaches.

3.1. Mesh proxies for texturing low-quality meshes

Degener and Klein (2007) and Schier et al. (2018) propose methods for assigning texture coordinates to triangle soups using point correspondences between vertices of the triangle soup and a textured mesh proxy obtained by tracing electrostatic field lines. The mesh proxy is constructed such that it satisfies requirements such as being similar to the input and having consistent connectivity that admits common parameterization approaches. Using vertex correspondences, they can texture the original mesh instead of replacing it with an approximation, enabling downstream pipelines that require the original mesh. By the physical laws governing field lines, their point correspondences are injective and flip-free, which is important for consistent texturing.

Both approaches transfer texture coordinates only at vertices and assume that linear interpolation between these UV coordinates is a sufficient approximation inside the input mesh faces. This assumption is usually justified as the mesh proxy is constructed such that it approximates the geometry of the input mesh. Both approaches share a failure case in the presence of UV seams, as linear interpolation between projected coordinates cannot represent the discontinuity along the texture seam on the mesh proxy. Degener and Klein handle this case using an adaptive refinement scheme that confines the artifacts to small triangles, whereas Schier et al. describe UV space mitigations. These mitigations adjust the UV space embedding, so that each triangle is clamped to a single UV chart, thereby avoiding texture distortion at the cost of additional texture deformation due to the modified UV map.

3.2. Linear interpolation in UV space

Linear interpolation in UV space is well-defined on the parameterized mesh proxy, as the images of the UV seams coincide by definition with the boundaries of the UV charts, and the UV map is affine for each face. When projecting vertices of another mesh onto the proxy UV map, this assumption no longer holds as the map from the input mesh to the mesh proxy is only piecewise-affine. For faces that come to lie in a single UV chart of the mesh proxy, the composition of the projection onto the proxy and the parameterization of the proxy usually still yields a useful UV map, because the mesh proxy is built to be similar to the input, generally this is only a practical result without mathematical guarantees.

However, if the projected face intersects a UV seam, the linear interpolation on the input mesh fails to represent the discontinuity in UV space, thereby causing visible texture artifacts. The mitigations applied in previous work aim to avoid such intersections altogether instead of resolving the root cause. Degener and Klein use an adaptive refinement to reduce the size of the affected faces, whereas Schier et al. mitigate the issue by adjusting the resulting texture map by repositioning vertices in UV space.

We show the three problems of linear interpolation in UV space that occur when transferring a texture from the parameterized mesh proxy shown in Fig. 3:

- A *multi-chart triangle* (Fig. 4(a)) has vertices on more than one UV chart, so interpolation in UV space is arbitrary as the charts are not guaranteed to be aligned in any meaningful way.
- A *wrap-around triangle* (Fig. 4(b)) has all vertices in a single UV chart, but the straight edge in UV space corresponds to a path wrapping around the chart instead of intersecting the seam.
- A *seam-intersecting triangle* (Fig. 4(c)) has all vertices in a single UV chart, but a straight edge in UV space crosses a non-convex chart boundary.

The first issue is discussed and resolved in Schier et al. (2018), and the second is addressed in subsequent implementations. To our knowledge, prior work does not handle the third, which could also be treated using a rigid embedding to replace the distorted mapping, but cannot be detected reliably. Examples of the texture artifacts for this example mesh are shown in Section 10.3 and Appendix E.

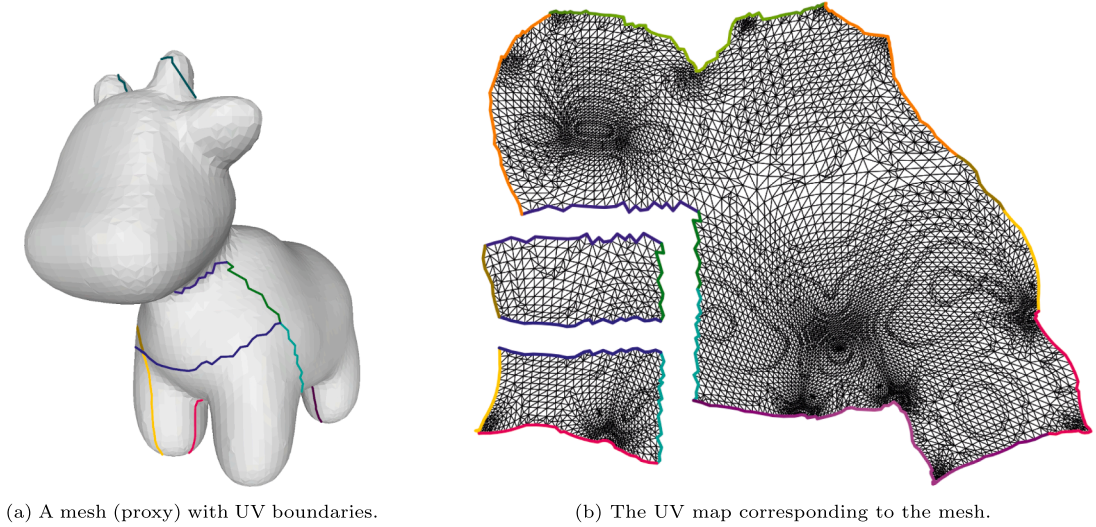


Fig. 3. A parameterized mesh with three UV charts. The colors indicate which seams belong together. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

3.3. Seam-aware interpolation

To address the root cause of these artifacts, we briefly recall the texture mapping approach used in Degener and Klein (2007) and Schier et al. (2018). Both approaches employ a mesh proxy M_p that is similar to an input mesh M_i and is equipped with a standard texture map φ_p that assigns UV coordinates to the vertices and is barycentrically interpolated within faces. For a triangle mesh, this map is affine on each face, so discontinuities can only occur along mesh edges. To transfer this texture onto an input mesh M_i , these approaches define an injective and flip-free map π from the input mesh to the proxy mesh using electrostatic field lines. Composing π with φ_p then yields a texture map $\varphi_i : x \mapsto \varphi_p(\pi(x))$ on M_i . Since evaluating π is computationally expensive, φ_i is not evaluated for every point of M_i , but UV coordinates are computed only at the vertices of M_i and barycentrically interpolated within its faces, and as a result, each edge of M_i is mapped to a straight segment in UV space. Texture artifacts arise because the image of an input edge under π is generally not aligned with edges of M_p and may cross seam edges of the proxy texture map. In that case, the image of the edge under φ_i is discontinuous in UV space and therefore cannot be approximated well by linear interpolation between its endpoints. Our approach addresses this by splitting edges of M_i into segments whose image under φ_i is continuous in UV space, so that each segment can be approximated linearly.

The image of an edge of M_i under π is generally a smooth curve on M_p , so any practical construction short of dense sampling must remain approximate. This is consistent with the overall philosophy of the original approach, where field lines are used primarily because they provide topological guarantees (injectivity and flip-freeness) and not because of geometric exactness. We therefore introduce an additional heuristic and approximate the curve on M_p by a shortest geodesic between the projected edge endpoints. In practice, this provides a good approximation for the relatively short projected edges considered here.

A key advantage of shortest geodesics is that they make seam intersections easy to identify, as on a triangle mesh, a shortest geodesic consists of piecewise-linear segments within faces, with breakpoints only where the geodesic crosses mesh edges or passes through vertices. Linear interpolation is therefore well-defined on each segment, and seam crossings can only occur at breakpoints. By parameterizing the resulting geodesic polyline on M_p by relative length, we can define a bijective map between the input edge of M_i and the geodesic approximation of its image on M_p . If the geodesic is split at every crossed mesh edge, each segment lies within a single face of M_p , where φ_p is affine, so its UV image is linear. Splitting only at seam crossings yields a coarser linear interpolation that in practice still provides a reasonable approximation of the geodesic image, and thus of the original input edge. The incident faces can then be subdivided by inserting seam segments between the edge splits, yielding sub-faces whose image in UV space is continuous, as shown in Fig. 4(d).

4. Overview

We define a pipeline for parameterizing an inconsistent input mesh by using a consistent mesh proxy as a carrier surface. The pipeline can be run fully automatically, but also allows optional manual intervention, for example to edit the proxy parameterization or prescribe custom UV seams. We first construct and parameterize a suitable mesh proxy, then project the input vertices onto it. We then represent the input edges on the proxy by geodesic paths, merge these paths with the UV seam edges of the proxy, and reconstruct a merged mesh in which seam-induced sub-faces are aligned with the UV chart boundaries defined on the mesh proxy (see Fig. 5). A detailed overview of the full pipeline is given in Appendix A.

The next sections are structured as follows. In Section 5 we describe how we project input mesh vertices onto the proxy using field lines as in Schier et al. (2018), and how we then trace shortest geodesic paths on the proxy using the intersection sequences

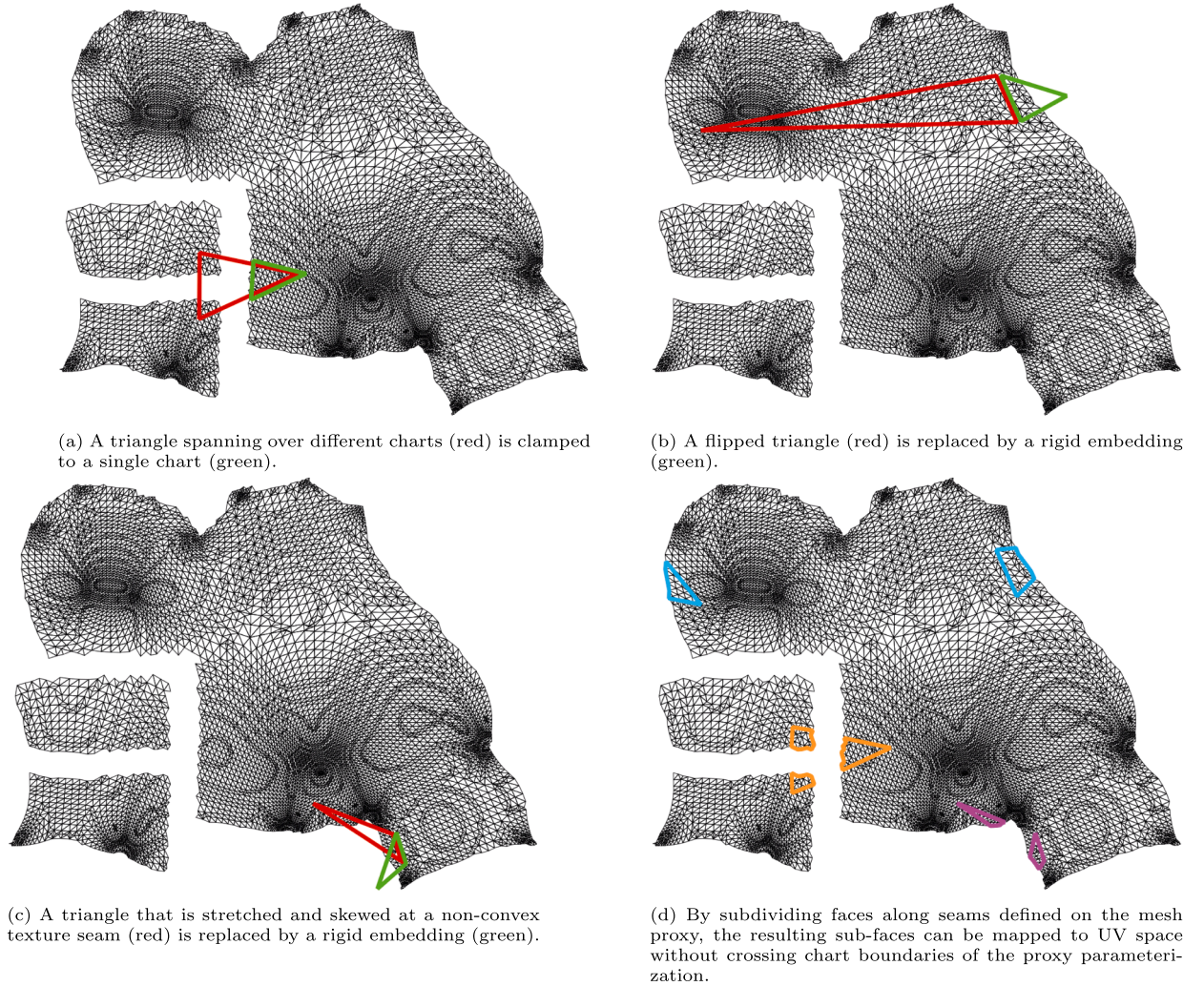


Fig. 4. There are three configurations for seam intersections that need to be treated. The red triangle in each image shows the projected face and the green triangle the UV space mitigation applied in previous work. The last image shows the desired embedding that subdivides the faces instead of heuristically repositioning its vertices. Note that the arrangement of UV charts in the example is chosen to be benign for visualization purposes, and in general, the position and rotation of the charts are arbitrary and may not correspond to the positions on the 3D surface. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

computed by [Sharp and Crane \(2020\)](#). We also describe why inserting multiple geodesics directly into the mesh proxy via refinement is numerically fragile, which motivates the edge-by-edge construction of a non-manifold curve network from these intersection points as presented in [Section 7](#). In [Section 8](#) we explain how to reconstruct a manifold surface mesh with a standard halfedge structure from this non-manifold curve network and how to handle hanging edges that cannot be represented in such a mesh. In [Section 9](#) we describe how we align the reconstructed mesh back with the input mesh, and in [Section 10](#) we present results of our algorithm and compare them to those of [Schier et al. \(2018\)](#). We discuss the limitations of the approach in [Section 11](#) before we conclude in [Section 12](#).

5. Projecting input edges onto the mesh proxy

To split projected input faces by seams on M_p , we first represent each input edge of M_i on the proxy as a shortest-geodesic polyline. These edge representations on the proxy must later be combined with the intersected texture seam edges so that the reconstructed mesh contains all seam-induced splits.

We follow [Schier et al. \(2018\)](#) for projecting the endpoints of each edge in M_i onto the mesh proxy along field lines, and to increase the stability of later algorithms and avoid badly conditioned result meshes, we treat projected points that lie close to existing proxy vertices or edges as belonging to those elements, reusing the vertex or splitting the edge to insert the projected point. For each input edge, we then approximate its image on the proxy by the shortest geodesic path between its projected endpoints, computed using the edge-flip algorithm of [Sharp and Crane \(2020\)](#), which operates on an intrinsic triangulation ([Sharp et al., 2019](#)). For a pair

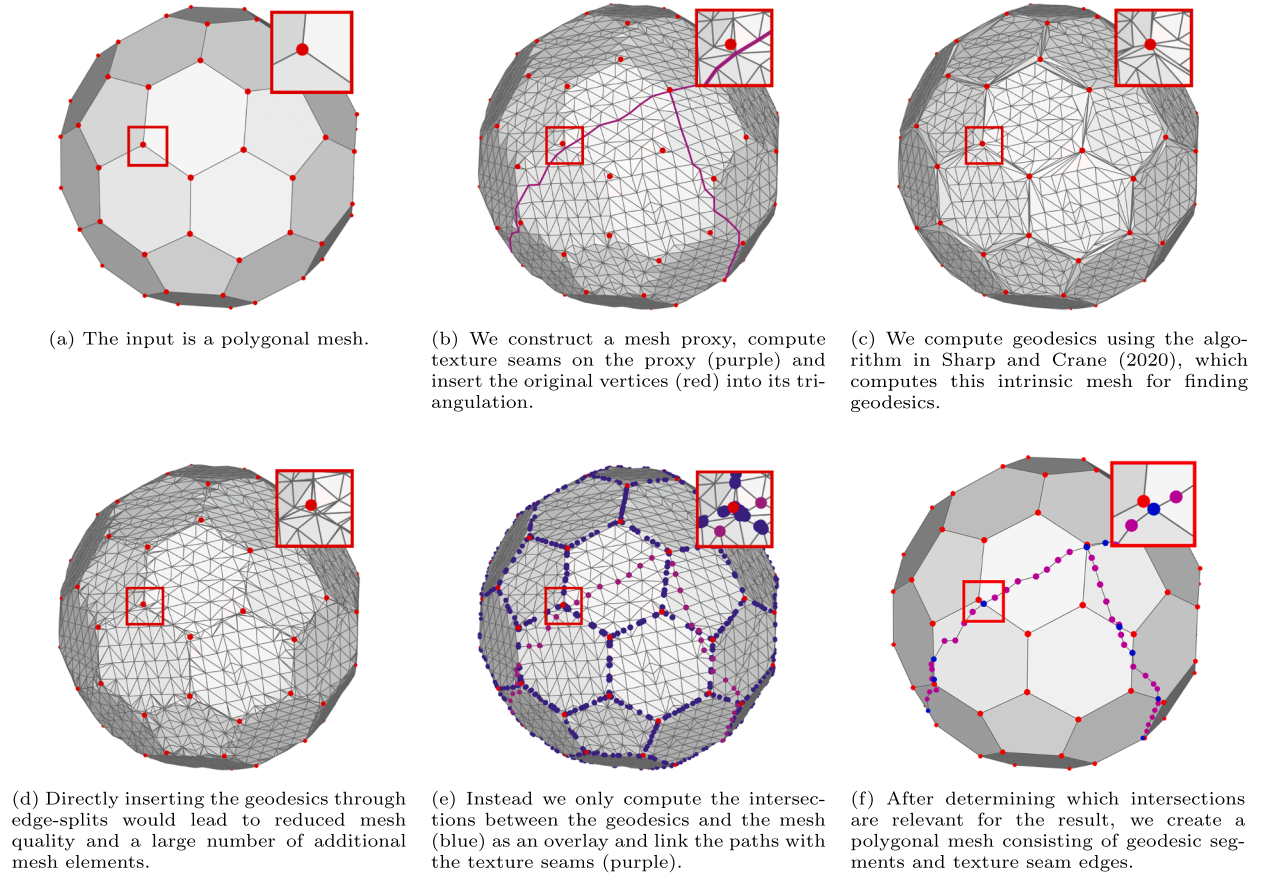


Fig. 5. Overview of how we create a merged mesh by joining geodesic representations of input edges with texture seams.

of proxy vertices, this yields an ordered sequence of intersection points with proxy edges, with each pair of consecutive intersection points lying on the boundary of a shared proxy face, as illustrated in Fig. 6.

Throughout the paper we use the following notation for the meshes and intersections between geodesic paths and mesh edges.

Definition 1. We use the following meshes in the algorithms.

- M_i is a polygonal *input mesh* to be parameterized, which may have topological inconsistencies.
- M_p is a topologically consistent *mesh proxy* that approximates M_i without reproducing the topological inconsistencies.
- M_r is the *result mesh*, obtained by joining segments of projected edges of M_i with seam edges of M_p .

We use halfedge meshes and define for each edge one of its halfedges to be its *primary halfedge* h_0 and the opposite halfedge to be h_1 . The choice is arbitrary but fixed throughout the algorithms.

Definition 2. We call a vertex v of M_p that is part of a geodesic path a *vertex intersection point* of the path. We call a point where the geodesic crosses a mesh edge e of M_p an *edge intersection point* and denote it by $p = (e, t)$, where $t \in [0, 1]$ is the relative position of the intersection along the primary halfedge of e .

5.1. Path embedding through refinement

Given a set of geodesic paths as sequences of intersection points (cf. Fig. 6), a seemingly straightforward strategy would be to refine the mesh proxy M_p directly along these paths, so that the geodesic path becomes a part of the triangulation of M_p , and the desired curve network can later be extracted as a subset of the edges of M_p . To insert the path segments for the sequence of intersection points, it would suffice to perform an edge-split for each intersected edge of M_p to automatically obtain an edge between the split point and the previous intersection point by retriangulating the adjacent faces, as depicted in Fig. 6(b). As this approach maintains M_p as a triangle mesh, it not only inserts the geodesic segment (in backward direction), but also a retriangulation edge in the next triangle. While this edge cannot obstruct the next intersection point of the current path, it may occlude adjacent intersection points in other geodesic paths, as depicted in Fig. 6(c).

Handling this occlusion would require either recomputing geodesic paths after each refinement step or computing additional intersection points to be inserted into the occluded paths. In our experiments, both options quickly produced poorly conditioned

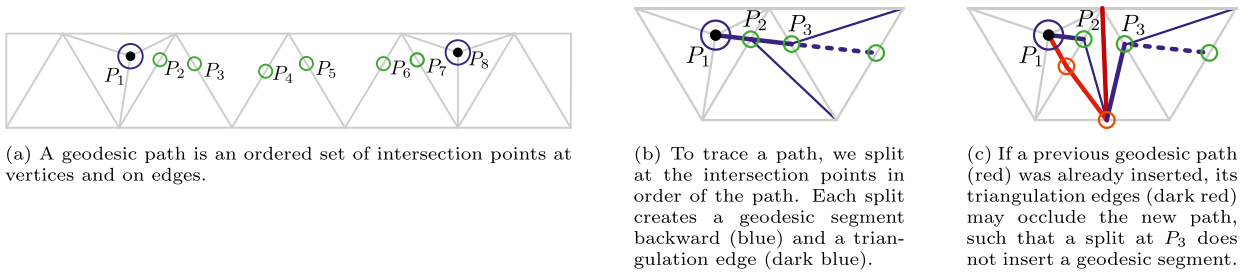


Fig. 6. When multiple geodesic paths are inserted sequentially, splitting only at their intersection points with existing edges is not sufficient, as edges created by earlier insertions can obstruct later paths and prevent a split point from yielding a valid geodesic segment.

triangulations, which in turn caused numerical instabilities both when recomputing geodesics using [Sharp and Crane \(2020\)](#) and when determining new intersection points on the newly inserted edges.

6. Using an overlay mesh

We avoid modifying M_p directly by constructing a separate overlay mesh M_r instead. In the overlay mesh we can insert segments of geodesic paths without invalidating other geodesic paths, as each path is traced on the fixed mesh proxy. Vertices of M_r are given by the breakpoints of geodesic paths, i.e., their endpoints and intersections with edges of M_p , together with the vertices on UV seams of M_p . The edges of M_r are constructed in two ways. Consecutive breakpoints along a projected input edge define geodesic segments, while UV seam edges are given by the connectivity between the corresponding vertices in M_p .

The remaining difficulty is to construct a shared topology between the two curve networks. Since the overlay mesh does not refine M_p , the geodesic segments do not automatically inherit the local connectivity of the proxy triangulation, and in particular, we do not yet know the cyclic order of incident geodesic segments and UV seam edges at shared vertices, which is required to recover the face structure of M_r without relying on unstable geometric predicates.

In the following, we first define the piecewise-geodesic face loops in the overlay on M_p induced by faces in M_i , and then show how the local topology in the neighborhood of each intersection point on M_p allows for inserting seam edges of M_p into the cyclic ordering at the intersection points.

6.1. Overlay faces

Our goal is to represent each face of M_i on M_p by a piecewise-geodesic loop and to insert UV seam edges of M_p in the loop wherever the projected face is crossed by a seam of M_p . This yields one or more overlay loops whose boundaries consist of geodesic segments and, where necessary, UV seam edges.

Definition 3. An *overlay loop* is a closed sequence of geodesic segments and possibly UV seam edges between projected vertices of a single input face. The region on the proxy surface bounded by an overlay loop is called an *overlay face*.

Each overlay loop initially consists of shortest-geodesic paths on M_p between the vertices of a face of M_i inserted into M_p , represented by linear segments lying entirely within single faces of M_p . Where such a loop is split by a UV seam of M_p , the corresponding seam edges are included in the boundary, yielding multiple sub-loops. Since these sub-loops consist only of linear segments within faces of M_p and of existing edges of M_p , they follow the underlying proxy geometry exactly along their boundary (cf. [Fig. 7\(a\)](#)). However, faithfully reproducing all geometric detail is not always the goal, since the mesh proxy is only an approximation of M_i and its small-scale geometric features are not meaningful on their own. For the texturing application, we only need to retain the breakpoints required to recover the topology of the sub-faces created by splitting projected input faces along UV seams. In many cases it is therefore preferable to simplify the overlay loops and keep only the vertices required for correct topology and seam representation, as shown in [Fig. 7\(c\)](#).

Definition 4. A *relevant intersection point* is an intersection point of a geodesic path on M_p that must be kept in the overlay mesh M_r . We consider an intersection point relevant if it (i) is a projected vertex of M_i , or (ii) lies on a UV seam edge of M_p .

We additionally treat vertices of M_p that are endpoints of UV seam edges as relevant vertices of M_r . Although these seam vertices are not intersection points in the above sense, including them allows us to handle geodesic segments and UV seam edges uniformly in the later sections. [Fig. 7\(b\)](#) shows an overlay face consisting of three geodesic polylines and a UV seam splitting the face into two sub-faces. If the goal is only to subdivide faces along UV seams, such that no sub-face is crossed by a seam, it is sufficient to keep the relevant intersection points to represent the topology and the UV seam vertices to ensure that the faces are split exactly along the seam. The simplified sub-faces are depicted in [Fig. 7\(c\)](#) and we show how the simplification affects the overall mesh in [Appendix B](#).

7. Constructing the overlay mesh

We construct the overlay mesh M_r , by joining the curve network given by the geodesic polylines with the UV seam edges of M_p . After introducing the required vertices and edge segments in the previous section, the remaining task is to merge the two disjoint

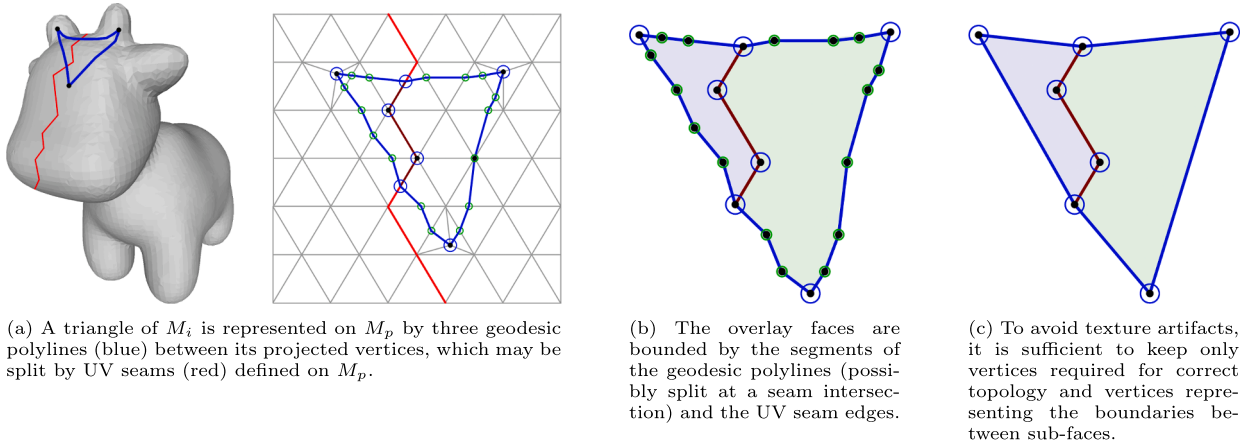


Fig. 7. By tracing geodesics on M_p and joining their segments with UV seams, we obtain closed loops in the overlay mesh that can later be used to construct faces. Topologically relevant intersection points are marked in blue, whereas intersection points marked in green are only geometrically relevant and can be skipped without introducing topological inconsistencies. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

edge-stars present at each edge intersection point on a UV seam edge, namely the edge-star of geodesic segments induced by the connectivity of M_i , and the edge-star of UV seam edges induced by the connectivity of M_p . To obtain a common cyclic ordering without resorting to unstable geometric predicates, we exploit the fact that all relevant intersection points are defined on M_p , so their local topology can still be determined using the proxy even though they are not inserted into it. To find this ordering, it suffices to consider only the direct neighborhood of the intersection point, which is the 1-ring around a vertex intersection point v , or the diamond of an edge for an edge intersection point (e, t) . In either case, we traverse this neighborhood on M_p in clockwise order and record the incident UV seam edges and geodesic segments. After one full cycle, this yields the cyclic ordering of the incident edges required to construct the full edge-star of the corresponding vertex in M_r .

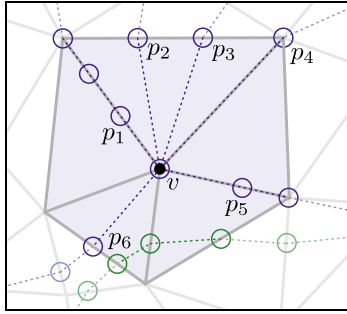
7.1. Creating edge-stars at intersection points

To build the curve network, we process one intersection point at a time and construct its edge-star. As constructing the mesh edge-by-edge is not supported by the standard halfedge structure, we employ our own non-manifold halfedge structure, which is described in [Appendix D](#).

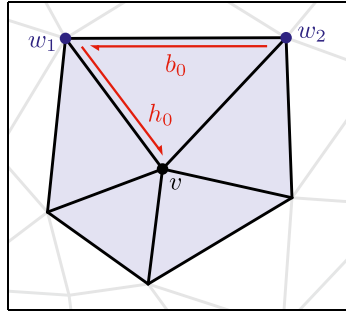
We first insert vertices in M_r , which are defined by the intersection points of the geodesic paths and endpoints of UV seam edges. On each edge of M_p , we sort the intersection points (e, t) by their relative position, which is the only remaining predicate required by our construction. In addition, we precompute the set of related intersection points for each intersection point that are either direct neighbors along the same geodesic path or adjacent on a UV seam edge. This allows the local traversal to skip nearby intersection points that happen to lie in the same 1-ring or edge diamond but are unrelated to the current intersection point. We construct the overlay mesh one intersection point at a time by building the local edge-star of each vertex of M_r . For a vertex intersection point, we traverse the proxy 1-ring in clockwise order alternating between halfedges incoming to v and 1-ring boundary edges, and for an edge intersection point $p = (e, t)$, we traverse the boundary of the edge diamond of e in a similar way as shown in [Fig. 8](#). In each case, we collect the incoming halfedges to the current point and link them cyclically so that the *next* pointer of each incoming halfedge points to the *opposite* of the next incoming halfedge in clockwise order. Since every relevant intersection point is processed and the neighborhood relation is symmetric, every halfedge is eventually inserted into an edge-star. After all edge-stars have been constructed, the resulting faces are linked in the standard counter-clockwise order. [Fig. 9](#) illustrates this construction, and [Algorithm 1](#) and [Algorithm 2](#) give the corresponding pseudocode.

7.2. UV interpolation

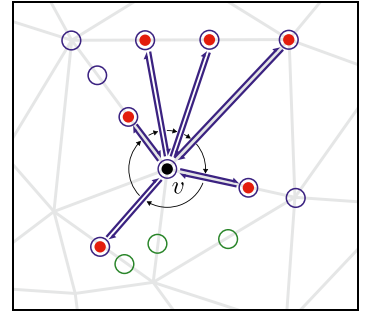
We assign UV coordinates to the halfedges of M_r according to the triangle of the mesh proxy M_p in which the corresponding segment lies. For the edge-star of a vertex v , every halfedge of M_r incoming to v lies in one of the wedges between an incoming halfedge h_0 of M_p and its previous halfedge b_0 on the 1-ring boundary (cf. [Fig. 8\(b\)](#)). All segments in such a wedge lie in the same triangle of M_p and end at the same vertex v , so they are assigned the halfedge UV coordinates of h_0 . For the edge-star of an edge intersection point $p = (e, t)$, every halfedge of M_r incoming to p lies in one of the two triangles of the diamond of e (cf. [Fig. 8\(e\)](#)). We therefore interpolate UV coordinates at p separately along h_0 and h_1 , and assign each incoming halfedge the UV coordinates interpolated along the halfedge of e belonging to the triangle in which its segment lies. If e is a UV seam edge, interpolating along h_0 and h_1 separately automatically assigns different face-vertex UV coordinates to the segments on the two sides of the seam, so seam edges included in M_r inherit the UV discontinuity from M_p .



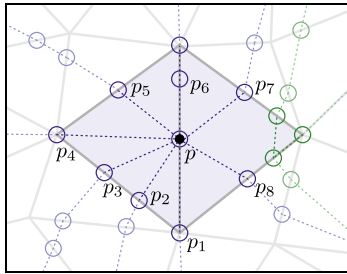
(a) For each vertex v , we walk around its 1-ring and add intersection points (blue) that lie on geodesic paths or UV seams (dashed lines) passing through v in clockwise order. On incident edges, only the point closest to v is added. Intersection points of unrelated paths (green) are skipped.



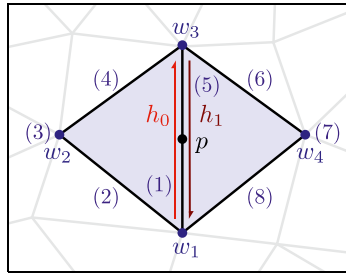
(b) For a vertex v , the algorithm alternates between processing incoming mesh proxy halfedges $h_0 = [w_1, v]$ in the edge-star of v and their previous halfedge $b_0 = [w_2, w_1]$ on the 1-ring boundary around v .



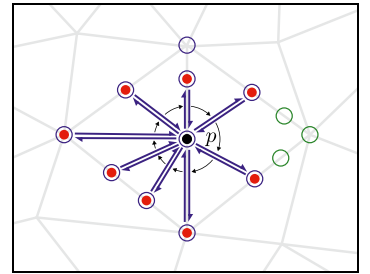
(c) The edge-star around v is constructed in M_r by creating halfedge pairs between the intersection points selected in (a) and v , joined at v in the cyclic order induced by the 1-ring defined by the triangulation of M_p .



(d) Edge intersection stars are constructed analogously to the vertex case, but by walking around the diamond of an edge. We connect the edge intersection point p to the related intersection points (blue) of its diamond: (i) the closest other intersection points on its edge, (ii) intersection points on the diamond edges, and (iii) intersection points at the vertices opposite the edge. Intersection points of unrelated paths (green) are skipped.

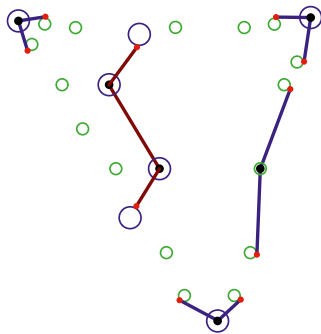


(e) For each edge intersection point p with diamond as in (d), the algorithm processes eight regions of the edge diamond around p : The vertices of the edge (1, 5), the diamond edges (2, 4, 6, 8), and the vertices opposite the edge (3, 7).

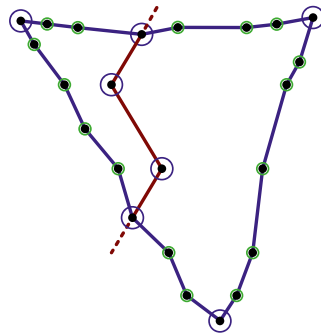


(f) The resulting edge-star around the edge-intersection point p is built in M_r by adding the halfedge pairs between the intersection points selected in (d) and p , joined at p in the cyclic order induced by the diamond defined on M_p .

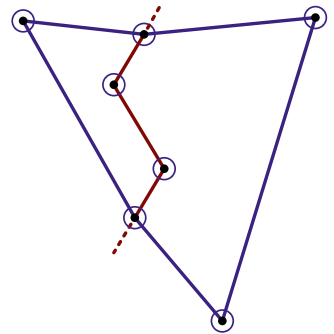
Fig. 8. We build two kinds of edge-stars. The stars at vertices of M_p are built by circulating clockwise over incident edges and 1-ring boundary edges (top row), and stars of edge intersection points are built by circulating clockwise around the diamond formed by the two adjacent triangles (bottom row).



(a) The overlay mesh after adding the edges adjacent to vertices in M_p . The hanging edges (with red endpoints) are not connected to a second vertex yet.



(b) When all segments adjacent to vertex intersection points and edge intersections with proxy edges (green) are created, the curve network forms a manifold mesh.



(c) By keeping only relevant vertices (blue circles), we obtain a minimal overlay mesh, containing only topologically required vertices and UV vertices.

Fig. 9. The overlay mesh is built edge-by-edge. Segments adjacent to vertices are inserted first and afterward the segments at edge intersection points are added. While intermediate steps are non-manifold, the final set of edges forms a manifold mesh.

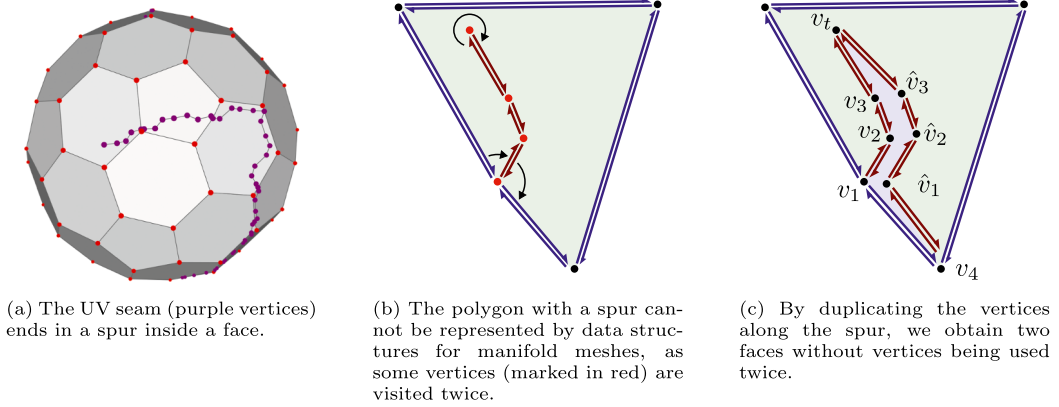


Fig. 10. A UV spur causes a face boundary to visit vertices twice. We first duplicate the repeated vertices to obtain manifold face loops and then add spur faces to close the resulting topological crack. Note that pairs of vertices $v_i, \hat{v}_i$ have the same coordinates and are only moved apart for visualization.

8. Building faces

After the edge-construction step, the halfedges in M_r form a consistent curve network in which the local topology is induced by the curves merged on mesh proxy. Each face of M_i thereby induces a closed boundary loop in M_r through the edge-paths corresponding to its boundary edges. If UV seams split such a loop, tracing one resulting loop follows the seam along one side, while the halfedges bounding the remaining regions remain unvisited and therefore become starting points for the loops of the other sub-faces.

Faces are extracted by tracing these halfedge loops in M_r . Starting from an unvisited halfedge on a geodesic path, we follow the next pointers until the start halfedge is reached again and record the visited vertices in order. For input meshes with boundaries, each halfedge of a geodesic polyline in M_r is associated with the corresponding input halfedge in loop direction, and tracing is started only from halfedges that do not correspond to boundary halfedges of M_i , so the topological boundary of the input mesh is preserved in M_r .

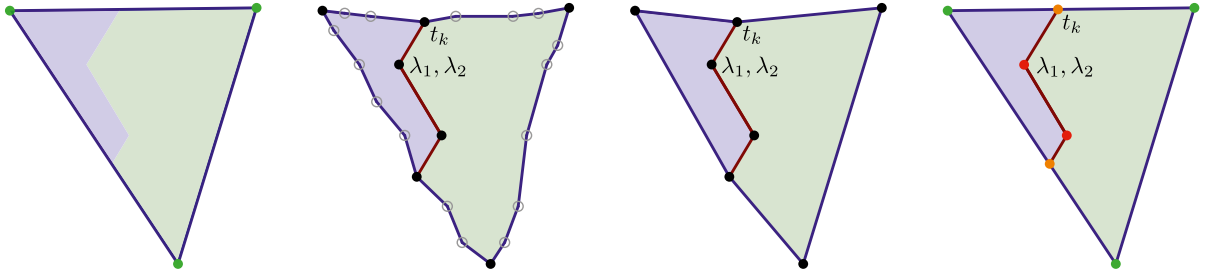
8.1. Topological assumptions

Our construction assumes that the shortest geodesic between the projected endpoints of an input edge provides a suitable surrogate for the projection of the edge on M_p . This is a practical assumption rather than a general guarantee, as the projection onto M_p may already be inaccurate, and even for correctly projected endpoints, the shortest geodesic may follow an unintended topological route or otherwise fail to represent the desired cell boundary. A second limitation is that the geodesic surrogates of different input faces may not form a globally valid non-overlapping cell complex on M_p (cf. Born et al., 2021). For texture transfer, however, this is not required. We trace each input face independently, following only the geodesic segments associated with that face and inserting UV seam edges wherever they split the traced loop. Overlaps between projected boundaries of different faces therefore do not prevent the extraction of consistent sub-faces for each face individually and by reusing vertices, existing connectivity of M_i is preserved without reconstruction on M_r . For texture transfer, we therefore trace each input face independently.

8.2. Non-manifold face boundaries

UV seams may end inside a projected face, forming a UV spur that extends from the projected face boundary into its interior, as shown in Fig. 10(a). If such a spur is included in the boundary loop, tracing the loop revisits some vertices. This does not cause problems for our tracing algorithm, because each halfedge in the loop is still visited only once, and our non-manifold halfedge structure represents faces as loops of halfedges without requiring vertices to occur only once, but the resulting loop cannot be added directly to the desired manifold result mesh, since standard manifold mesh data structures assume simple face boundaries without duplicate vertices. A simple way to adapt faces containing spurs to standard manifold mesh data structures is to duplicate vertices in M_r when they are visited a second time while tracing a face loop, and to use the duplicate vertex instead. Algorithm 3 shows in pseudocode how to trace face loops and duplicate spur vertices.

The downside of duplicating vertices is that it introduces topological cracks in the result mesh, as the seam edges along the spur are implicitly duplicated as well. To restore watertightness, we therefore add additional spur faces along these cracks. To build such a spur face, the algorithm detects when a traced vertex already belongs to the current loop and then stores the previous vertex as the spur tip v_t and collects the duplicated vertices $\hat{v}_n, \dots, \hat{v}_1$ while tracing the spur backward until it reaches the first non-duplicate vertex v_{n+1} (cf. Fig. 10(c)). The loop $[v_{n+1}, \hat{v}_1, \dots, \hat{v}_n, v_t, v_n, \dots, v_1]$ then defines a spur face, which is added to close the gap. As each duplicated vertex lies on top of its original vertex, the spur face has zero geometric area and is only required to topologically close



(a) An input face spanning two UV regions whose common boundary is not represented in the triangulation. (b) The corresponding overlay representation of the face in M_r includes the seam edges defined on M_p . (c) For texturing, we retain only the topologically relevant intersection points and UV seam vertices in M_r . (d) The overlay mesh M_r with subdivided faces backprojected to align with M_i .

Fig. 11. The blue and green regions visualize the proxy UV map restricted to the input face. Because the input vertices lie in different UV regions while the triangulation of M_i does not represent the boundary between them, direct interpolation on the face crosses the UV discontinuity. Representing the face on M_p allows the required seam edges to be inserted, and the resulting overlay mesh is then backprojected to align with the geometry of M_i . (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

the crack along the UV spur. These spur faces therefore do not affect the parameterization or rendering, but only close the gap and restore watertightness.

9. Backprojecting to the input mesh

So far, the overlay mesh M_r lies geometrically on the proxy mesh M_p . Its vertices lie on vertices or edges of M_p , and its edges either run through faces of M_p as geodesic segments or coincide with UV seam edges of M_p . To obtain the final result on M_i , we now backproject the retained vertices of M_r from their locations on M_p to corresponding positions on the input mesh.

In Fig. 11, black vertices denote retained overlay vertices on M_r lying on M_p , while hollow gray vertices denote traced geodesic intersections that are discarded before backprojection. These vertices fall into three classes:

- **Direct correspondences:** Vertices of M_r corresponding to vertices of M_i (green) are assigned the coordinates of their corresponding input vertices.
- **Edge interpolation:** Retained breakpoint vertices on geodesic polylines (orange) are placed on the corresponding edge of M_i by interpolating along that edge using the relative arc-length position of the breakpoint on the geodesic polyline.
- **Face interpolation:** Vertices originating from UV seam vertices of M_p that lie in the interior of an input face (red) are backprojected using the barycentric interpolation described below.

For *direct correspondences*, the 3D position on M_i is exact, since we simply reuse the original input vertex, and for retained breakpoints on projected input edges, we place the backprojected vertices on the corresponding edge of M_i . This preserves the boundary geometry of the input mesh exactly and treats the shortest geodesic on M_p only as a surrogate for determining the relative position of the split along that edge, without reproducing its curvature on the surface of the proxy mesh. The strongest geometric approximation occurs for interior seam vertices, which are backprojected using local barycentric interpolation inside the corresponding input face.

For *edge interpolation*, we parameterize each retained breakpoint by its relative arc-length along the geodesic surrogate of the corresponding input edge. Let $p_1, \dots, p_n$ be the breakpoints along the polyline, ordered from one endpoint to the other, with cumulative arc-lengths $L_1, \dots, L_n$. The normalized parameter $t_k = L_k/L_n$ then gives the relative position of breakpoint p_k along the corresponding input edge, as illustrated in Fig. 11(c) and (d).

Interior vertices, i.e., UV seam vertices of M_p inserted into M_r , are backprojected using *face interpolation* with a local barycentric interpolation. We compute barycentric coordinates λ_1, λ_2 with respect to a triangle spanned by three vertices of the overlay face on M_p and for polygonal faces we choose the largest such triangle to improve numerical stability. We then use the barycentric coordinates computed in the overlay face in M_r to interpolate the vertex position in M_i .

Note that this face interpolation assumes that the corresponding overlay face is approximately planar. Although the region of M_p bounded by a piecewise-geodesic loop is generally not planar, its interior should have only minor curvature when M_p approximates M_i well, so a linear approximation is sufficient in practice. Strongly non-planar input faces are not well suited to our algorithm, and we recommend subdividing them into approximately planar regions beforehand.

10. Evaluation

We evaluate our seam-cutting method on the cow mesh and NURBS examples from Fig. 1 and compare it against the UV heuristics of Schier et al. (2018) for triangles intersected by UV seams and analyze the distortion near seams, mesh complexity, and runtime for different resolutions of the mesh proxy.

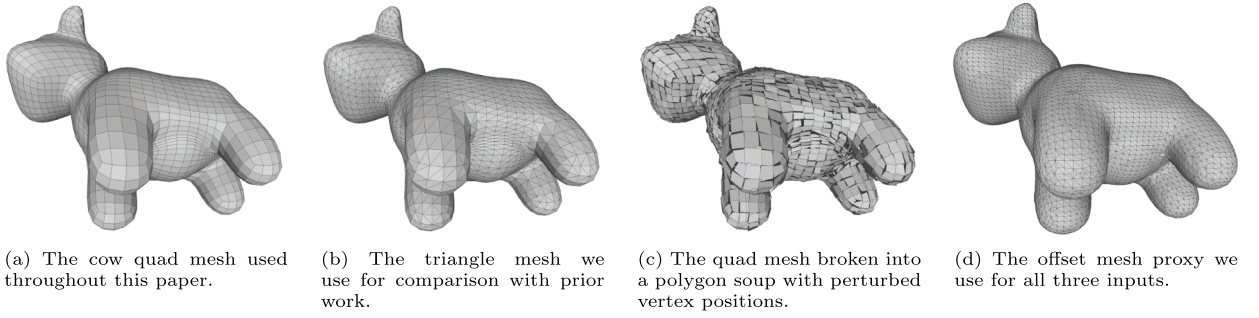


Fig. 12. The input meshes for the algorithms.

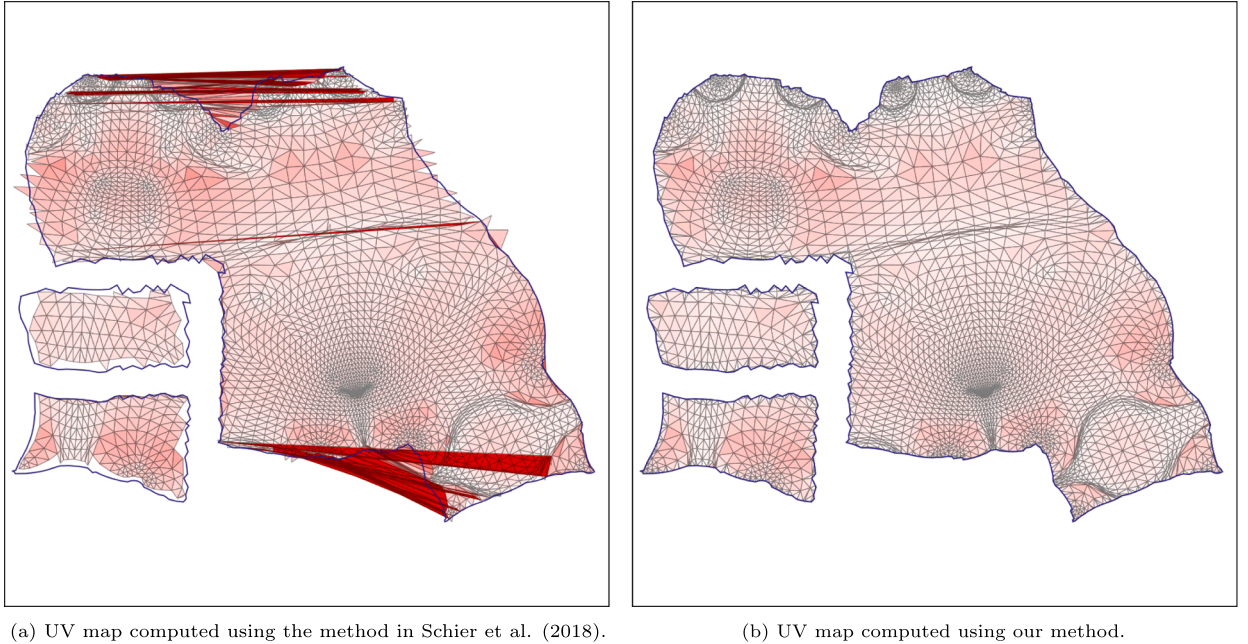


Fig. 13. Comparison of per-triangle Dirichlet energy, log-scaled and visualized using the same colormap. The blue outline shows the UV chart boundaries of the mesh proxy. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

10.1. Experiment setup

In the following experiments we use a common pipeline to compute UV coordinates at the vertices of the input mesh for both the heuristic approach and our method. Given an input mesh M_i , we first construct an offset proxy M_p , create a UV segmentation, and parameterize M_p . We then trace field lines to establish correspondences between vertices of M_i and points on M_p , and use these correspondences for assigning UV coordinates to the vertices. Afterward we create two meshes that address issues near UV seams with the two different methods. In the first, we apply the heuristic UV-fixing procedures of Schier et al. (2018), described in Section 3, and in the second, we use our edge-tracing method to insert the UV seams of M_p as cuts into M_i .

Fig. 12 shows the cow meshes (Crane et al., 2013a) used in our experiments. Our method takes the original quad mesh as input, which we triangulate for the method in Schier et al. (2018) because the flip-detection heuristic requires a triangular mesh. To demonstrate our method on inconsistent meshes, we construct an example by breaking the quad mesh into a polygon soup with perturbed vertex positions. For a comparable triangle mesh, we then triangulate this polygon soup rather than generating an independent triangle soup. Fig. 12(d) shows the offset mesh proxy M_p , which we use for all cow meshes in our benchmarks. For the NURBS patches, we use both the original quadrilateral meshes and a triangulated version as inputs, and construct a single-sided mesh proxy close to the original surface.

10.2. Seam artifacts and dirichlet energy

As both methods use the same parameterized mesh proxy and the same correspondences for the vertices of M_i , they behave similarly away from UV seams. The relevant differences in the resulting UV maps therefore occur near UV chart boundaries, where the heuristic corrections of Schier et al. (2018) fail to represent the discontinuity correctly.

In Fig. 13, we visualize the two UV maps and color them with the per-triangle Dirichlet energy, clamped to the 97.5th percentile of the first UV map. The bright red areas in Fig. 13(a) highlight triangles where the heuristic mitigations fail, particularly triangles spanning non-convex chart boundaries and wrap-around configurations that remain undetected. The result of our method, shown in Fig. 13(b), avoids these large local distortions by cutting the affected faces along the UV seams. The heatmaps also show that the charts produced by the heuristic-based method are not well aligned with the chart boundaries of the mesh proxy parameterization (blue contours). Clamping and rigid embedding can reduce distortion inside individual triangles, but these heuristics cannot recover the correct seam geometry, since the triangulations of the mesh proxy and the result mesh are not aligned. Moreover, the rigid embedding does not attempt to straighten seams, so the resulting UV chart boundaries on M_r can become jagged. In contrast, our method cuts the triangles of M_i along the UV seams of M_p , so the UV chart boundaries of the overlay mesh M_r coincide with those of M_p .

To quantify this behavior globally, Fig. 14 compares the normalized distributions of per-face Dirichlet energy. The overall distributions remain similar, which is expected because both methods share the same proxy UV map and vertex correspondences. The main difference lies in the tail, where the high-energy outliers produced by the heuristic UV corrections are strongly reduced or eliminated by our seam-cutting approach. For the NURBS example, the distributions shown in Fig. 14(e) are close to each other as the main problem solved is the missing representation of the UV cutout (cf. Fig. 16(c) and (d)), and not the distortion in the mapping. In Fig. 14(f), we compare our shared continuous parameterization with four separately computed discontinuous parameterizations of the patches. As expected, the disjoint parameterizations achieve lower Dirichlet energy by allowing discontinuities at the patch boundaries instead of enforcing continuity across the patches.

10.3. Qualitative comparison with prior work

For this comparison, we parameterize the mesh proxy using boundary-first flattening (Sawhney and Crane, 2017) with a cone-based segmentation and then manually split the resulting UV chart into three parts (cf. Fig. 3) so that the UV chart exhibits the seam configurations discussed in Section 3. For the field line method of Schier et al. (2018), we use a triangulated version of the input mesh, since their flip heuristic is defined for triangles and is not easily generalizable to polygons. This choice is favorable to the baseline, as it reduces texture artifacts at the cost of not preserving polygonal connectivity.

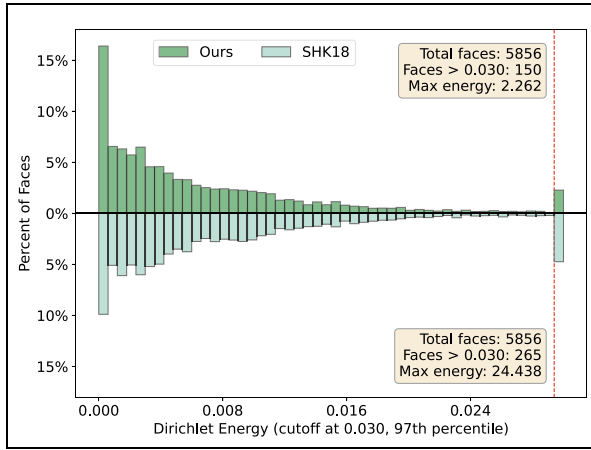
Fig. 15 shows a direct comparison between the heuristic seam correction and our exact cutting approach. Our result reproduces the UV seams of the mesh proxy faithfully, whereas the baseline exposes the seam artifacts described in Section 3. In the upper and lower parts of the large UV chart, where the parameterization unfolds along a seam of the chart with itself, many triangles span across the chart boundary, which leads to severely distorted texture mapping in the affected faces. On the textured model, some seams appear visually smooth because triangles crossing the seam are clamped to chart boundaries, but their position and shape do not match the seams of the mesh proxy as accurately as in our result. The NURBS example in Fig. 16 illustrates a limitation of heuristic UV-space seam correction when the desired seam configuration cannot be represented as an edge path on the input mesh. Here, the mesh proxy contains a yellow cut-out region inside the blue region, but on the coarse input mesh all vertices of that cut-out lie in the interiors of faces rather than on existing mesh edges. Our method traces the seam edges of the patch explicitly and reproduces the cut-out by inserting new vertices along its boundary (Fig. 16(c)), whereas the baseline only corrects the two distorted triangles whose projected UV embedding would span from the blue region into the yellow one by clamping the affected vertices back to the proxy seam position in the surrounding blue chart (Fig. 16(d)). As a result, the yellow region collapses completely and disappears from the parameterization, while the surrounding blue region exhibits additional distortion because the affected vertices are snapped to the proxy seam instead of remaining at positions consistent with the surrounding blue chart.

Fig. 17 shows the transfer of mesh-proxy UVs to a triangle soup created from a cat model in the SHREC2011 dataset (Boyer et al., 2011). The triangle soup consists of 56 262 vertices and 18 754 faces. For the mesh proxy, we use a segmentation into genus-zero patches in which each UV seam runs between two different patches, a configuration favorable to the chart heuristic of Schier et al. (2018). Fig. 17(b) shows that the chart heuristic mitigates the multi-chart failure case described in Section 3 while the choice of segmentation avoids wrap-around triangles and seam-intersecting triangles. As the heuristic clamps multi-chart triangles to a single chart, it still causes visible deviations of the chart boundaries from the mesh proxy UV layout. In contrast, our method splits the affected triangles and reproduces the mesh proxy seams exactly.

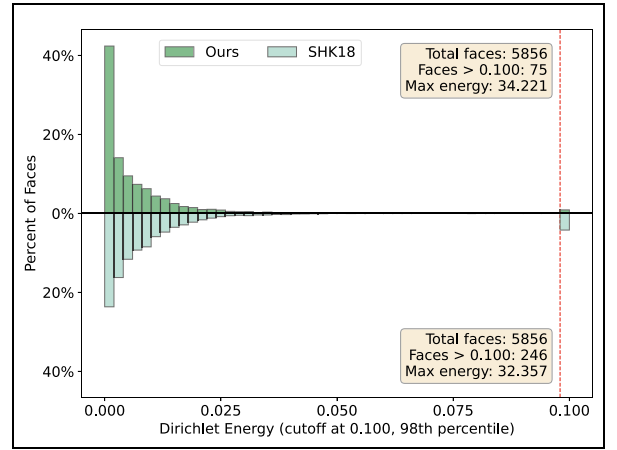
In contrast, for a challenging parameterization we transfer mesh-proxy UVs to a triangle soup created from the Botijo model from the COSEG dataset (Wang et al., 2012), as shown in Fig. 18. The triangle soup consists of 89 202 vertices and 29 734 faces, and before breaking up, the model had genus 5. For the mesh proxy, we intentionally use a less favorable cone-based segmentation with a single chart, which produces long seams and global self-overlaps in the proxy UV map. In this challenging setting, the seam heuristics of Schier et al. (2018) fail at many seams, whereas our algorithm reproduces the proxy seams on the result mesh and thereby enables correct mapping of the sub-faces on both sides of each seam, eliminating the corresponding texture artifacts. This behavior is consistent with the additional ablation study of the baseline heuristics shown in Section E, and it shows in particular, that the two heuristics address different subsets of seam failures. Chart clamping removes multi-chart embeddings robustly, but not wrap-around cases within a single chart, whereas the flip heuristic corrects many flipped triangles but can still miss seam crossings across non-convex chart boundaries.

10.4. Output complexity and runtime

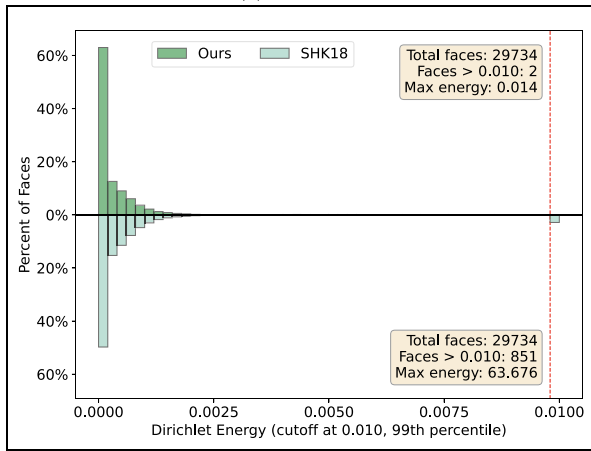
In the overall pipeline, the time required to trace the field lines used to establish vertex correspondences dominates, whereas the time for parameterization and segmentation is negligible for the mesh proxies considered here. Performance evaluations of these steps are reported in Sawhney and Crane (2017); Schier et al. (2018), so we focus in this subsection on the complexity of the result mesh and on the runtime of our own algorithm after these preprocessing steps.



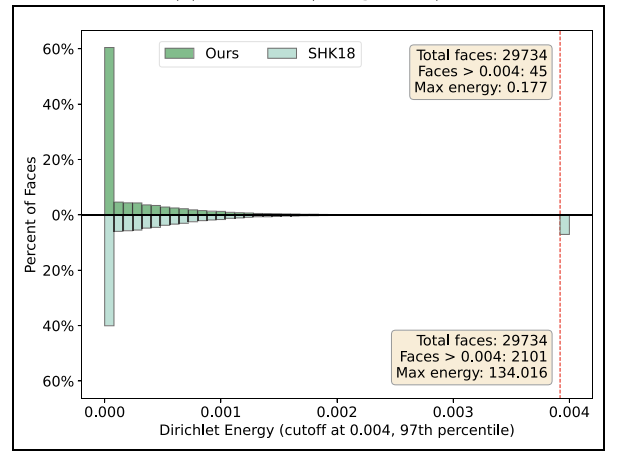
(a) Cow mesh



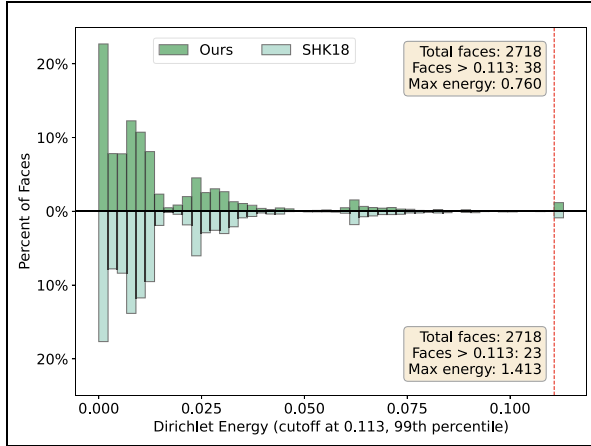
(b) Cow mesh (triangle soup)



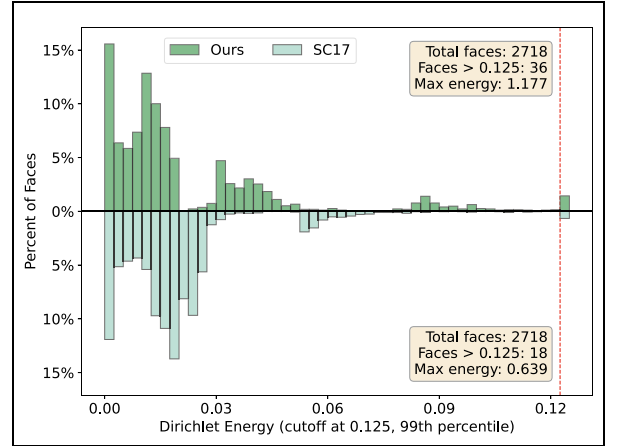
(c) Botijo mesh



(d) Botijo mesh (triangle soup)



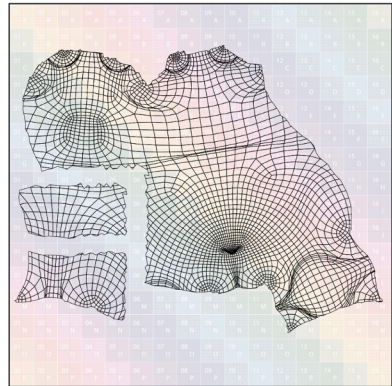
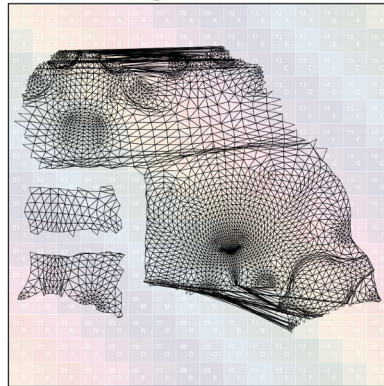
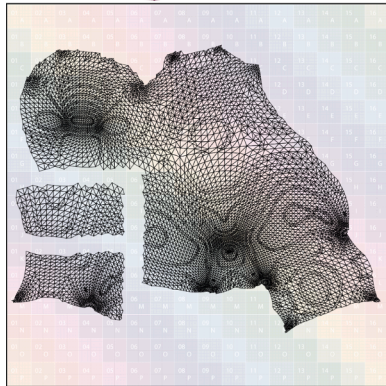
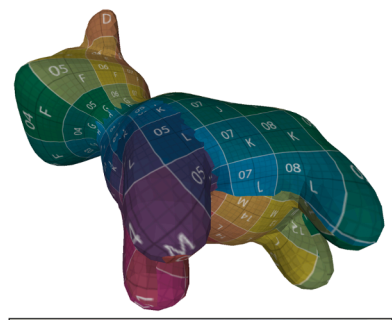
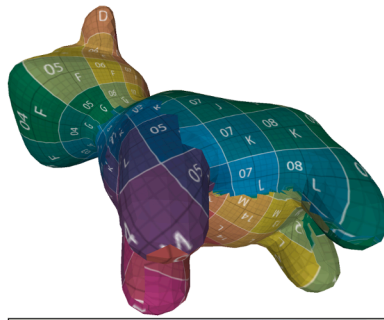
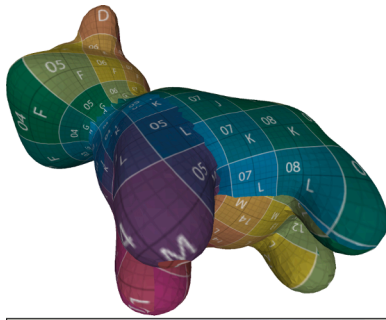
(e) NURBS patches with a projected parameterization containing a cutout region (cf. Fig. 16c).



(f) NURBS patches: our projected continuous parameterization (cf. Fig. 2c) versus four separately computed discontinuous parameterizations (cf. Fig. 2a) obtained directly with Sawhney and Crane (2017).

Fig. 14. Comparison of normalized per-face Dirichlet-energy distributions in the resulting UV maps. For each plot, the cutoff is chosen near the 97th to 99th percentile so that the bulk of the distribution is visible, and values above the cutoff are accumulated in the final bin.

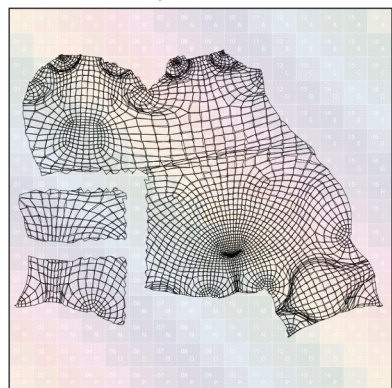
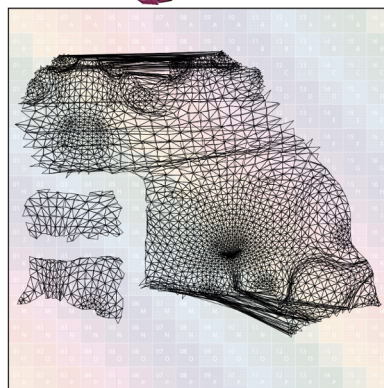
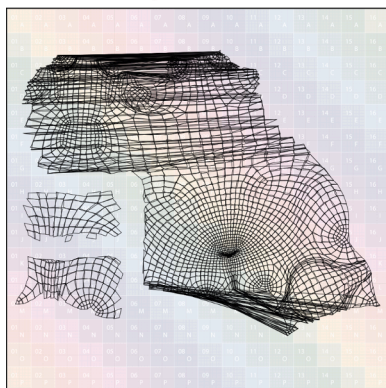
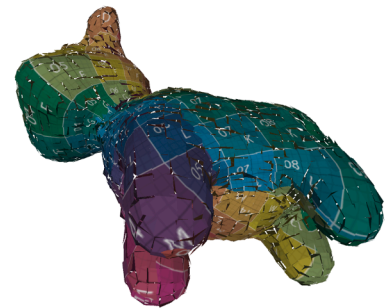
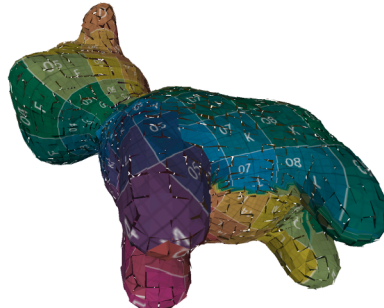
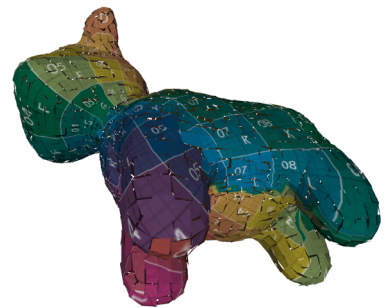
As described in Section 6, our algorithm can construct either a full overlay mesh or a reduced one obtained by filtering relevant intersection points. This reduction removes boundary vertices that are not required for the mesh topology and therefore primarily affects the size of the resulting mesh representation. A more detailed illustration of how this simplification affects the overlay mesh



(a) The texture map of the offset mesh proxy used to parameterize the cow mesh and the polygon soup in both methods.

(b) The projected texture map for the triangle mesh corrected using both UV heuristics.

(c) The projected texture map created using our exact UV cutting method.



(d) The projected texture map for the polygon soup corrected using the chart-based UV heuristic.

(e) The projected texture map for a triangulated version of the polygon soup corrected using both UV heuristics.

(f) The projected texture map for the polygon soup created using our exact UV cutting method.

Fig. 15. The results of texture projection using the heuristics of [Schier et al. \(2018\)](#) and our exact UV cutting method.

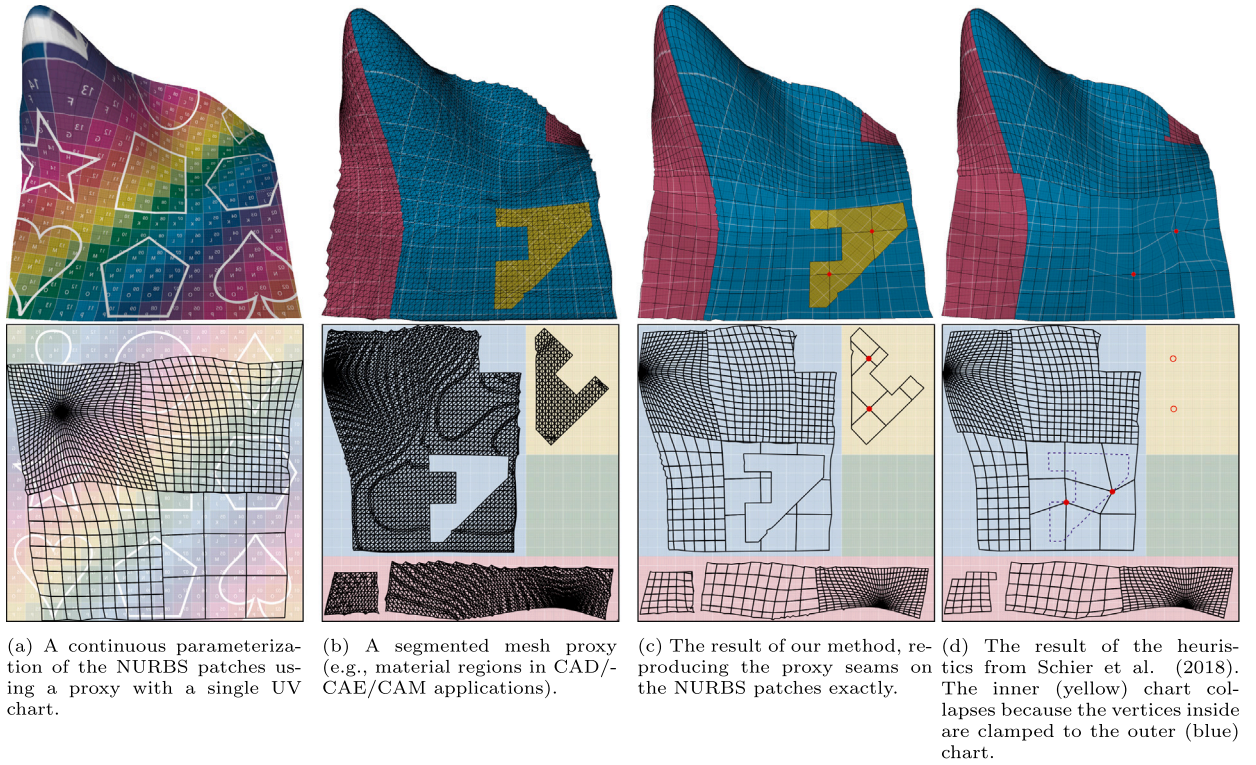


Fig. 16. The result of transferring the texture map of a segmented mesh proxy (or more generally a mesh proxy with seam edges) to the NURBS patches. The red points highlight two vertices of the lower-right NURBS patch that are projected into the yellow region on the proxy. Our method preserves this cut-out region in the UV map by explicitly representing its edges, whereas the baseline method clamps these vertices to the surrounding blue region, thereby collapsing the yellow chart. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

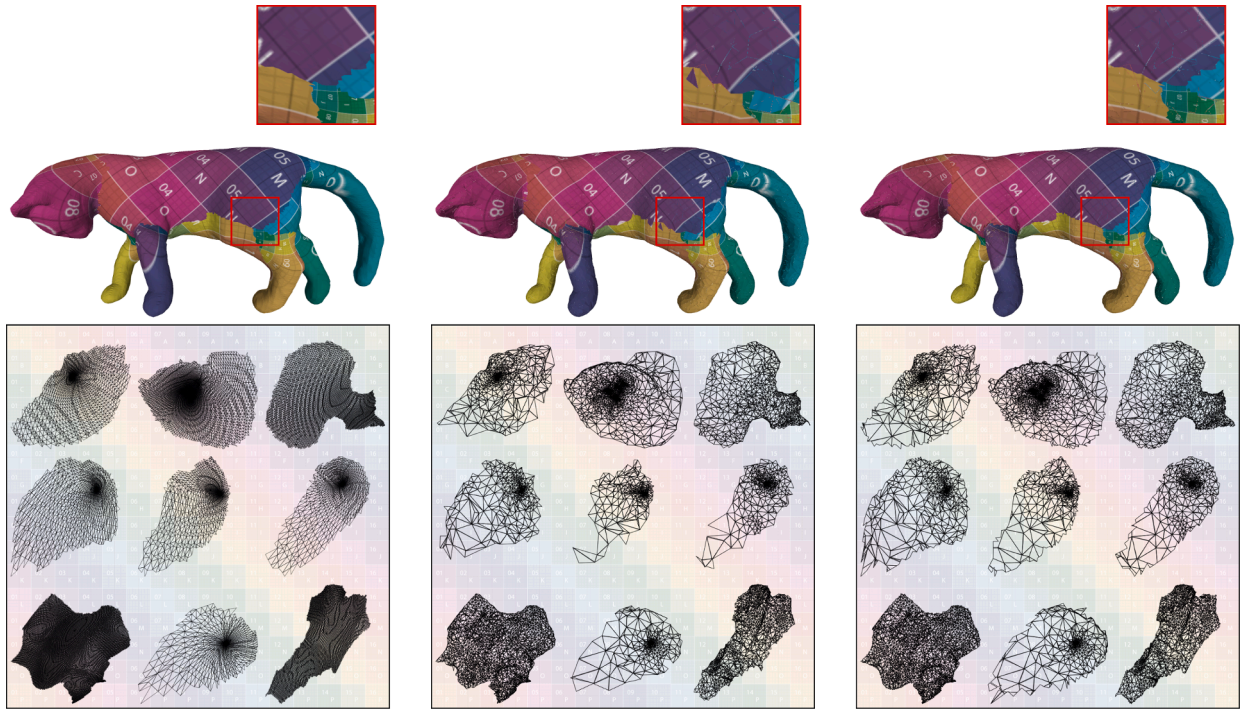
itself is given in [Appendix B](#). The plots in [Fig. 19](#) show how this affects the size of the resulting mesh representation for mesh proxies of increasing resolution, both with and without filtering, and for polygonal as well as triangulated result meshes. Keeping all intersection points makes the size of the result mesh grow strongly with proxy resolution, whereas filtering relevant intersection points keeps both the size of the polygonal and triangulated results largely stable.

For texture transfer, this reduction is usually desirable because the removed vertices lie only on face boundaries, so they increase the size and boundary complexity of the result mesh without introducing additional interpolation degrees of freedom inside the faces. Using a finer mesh proxy M_p beyond what is needed to approximate the input mesh mainly increases the complexity of the result mesh rather than improving the transferred texture map, and if more geometric or texture detail is required, refining the input mesh M_i is usually more effective than refining M_p . For applications in which the exact geometry of transferred paths must be preserved, on the other hand, retaining all intersection points can still be useful. An example is the cutout region in [Fig. 16\(b\)](#), where keeping all vertices on both projected input edges and proxy seam edges yields the most faithful representation, which may be required in engineering applications. The texture map shown in [Fig. 16\(c\)](#) reproduces, for example, the small indentation at the diagonal of the cutout, which would not be reproduced when simplifying UV seams (cf. [Appendix B](#)).

[Fig. 20](#) shows the runtime of our algorithm for fixed input meshes and mesh proxies of increasing resolution using a single-threaded implementation on an AMD Ryzen 9 7950X processor. We show timings for both the full mesh and the reduced mesh, and additionally report the runtime with the time spent on geodesic computation (using the reference implementation of [Sharp and Crane \(2020\)](#)) subtracted, in order to isolate the cost of our own construction steps.

The plots show that the runtime of our method is low and scales approximately linearly with the number of faces of the mesh proxy. For the highest-resolution proxies, the total runtime is about 0.66 s for the cow mesh, 0.3 s for the NURBS example, and about 2.3 s for the Botijo example, including the time for geodesic computation. Computing the reduced mesh is consistently only slightly faster than computing the full mesh, since irrelevant intersection points are filtered only in the final face-tracing step. Thus, the main practical advantage of the reduced representation is not a large runtime reduction, but a substantially simpler and more practical result mesh. This suggests that, for large meshes, earlier filtering could further reduce runtime.

By comparison, the cost of the baseline heuristics is essentially negligible. They require only about 0.002 s on the cow mesh, 0.002 s on the NURBS patches, and 0.012 s on the Botijo mesh. Although the required BVH lookups depend on the offset mesh resolution, this dependency is not measurable in our experiments, as for each of our example meshes, the run-to-run variation is larger than the observed variation across proxy resolutions. We therefore treat the runtimes plotted in [Fig. 20](#) as the effective cost added by our method.



(a) An offset mesh proxy parameterization in which each UV seam lies between two different charts.

(b) The seam heuristics in Schier et al. (2018) produce a UV map free of seam-crossing artifacts, but clamp multi-chart triangles to a single chart.

(c) Our method reproduces the exact UV boundaries by splitting multi-chart triangles along the UV seams defined on the proxy instead of clamping them to a single chart.

Fig. 17. Texture transfer to a triangle soup created from a cat model.

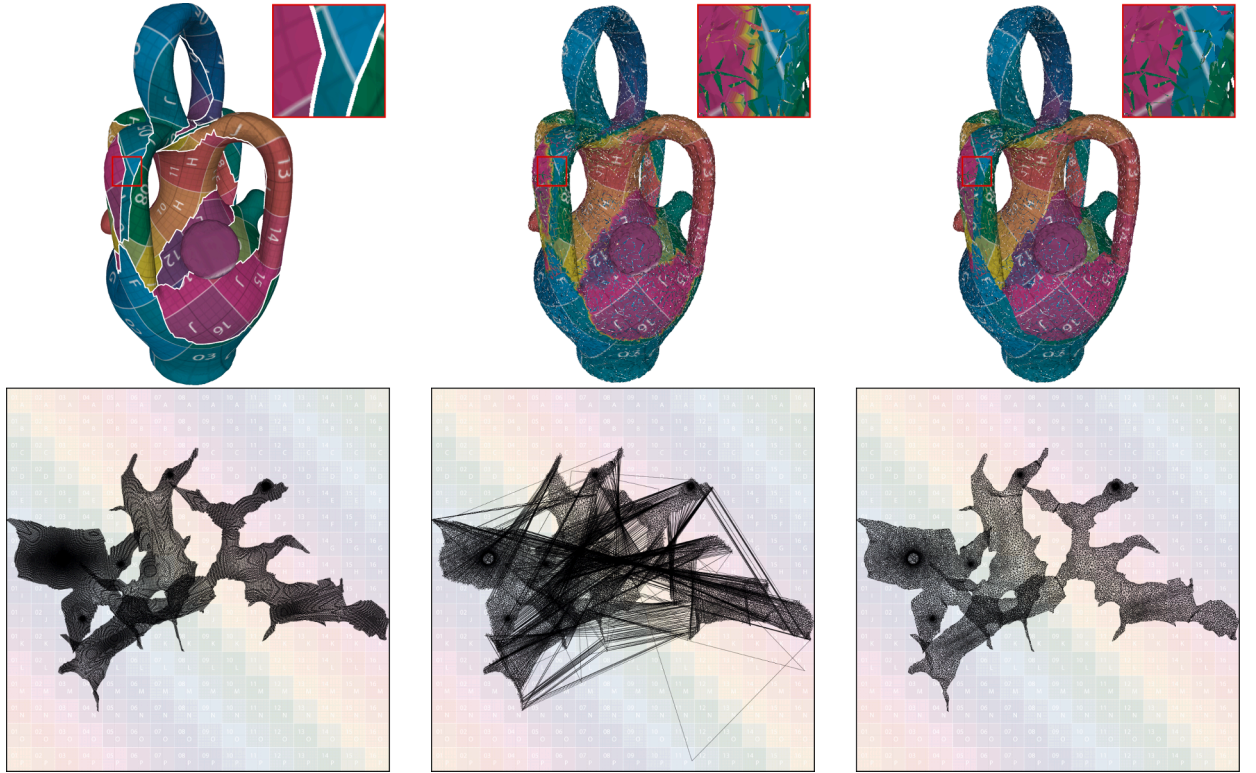
11. Limitations and future work

Our method is numerically robust and can handle inconsistent inputs ranging from collections of well-aligned patches to triangle soups without any connectivity, but it relies on several assumptions that replace unknown or inconsistent information by practical heuristics on a consistent mesh proxy surface. In particular, we assume reasonably well-behaved input polygons, a mesh proxy that approximates the intended geometry sufficiently well, and that projected input edges can be represented well by shortest geodesics on the proxy. These assumptions lead to a number of limitations and trade-offs, which we summarize below together with practical workarounds and directions for future work.

11.1. Mesh assumptions

Our method relies on three main assumptions about the input data. First, we assume that the mesh proxy M_p represents the intended geometry at an appropriate level of detail. It must be fine enough to capture the relevant surface features, but also smooth enough to remove artifacts caused by topological inconsistencies in the input. The difficulty of constructing such a proxy depends on whether the input is only topologically inconsistent or also exhibits geometric flaws that should be repaired in the mesh proxy. Since proxy construction is a well-studied problem, we refer to the literature for a detailed discussion. Second, we assume a well-behaved projection from the input mesh M_i onto M_p . In our implementation, this is enforced by the injective and flip-free field line projection of Schier et al. (2018). Third, some parts of the pipeline assume that input polygons are simple and approximately planar. In particular, the backprojection step uses planar interpolation for interior seam vertices, so highly non-planar faces are problematic for the current formulation.

The implementation described in this paper also does not resolve self-intersections or overlapping regions of the input mesh. Such overlaps are reproduced in the result mesh M_r , which can be desirable when the original input topology should be preserved despite geometric inconsistencies. In principle, exact bookkeeping of traced face boundaries would also allow the reconstruction of non-simple polygons, and our non-manifold halfedge structure can represent such intermediate configurations. In practice, however, this would introduce many additional corner cases and would produce meshes that cannot be represented by typical downstream data structures. For practical applications, we therefore recommend cutting non-simple polygons into simple ones, or triangulating them so that the input faces are simple and planar.



(a) The cone-based mesh proxy parameterization consists of a single chart with a long seam and global self-overlaps.

(b) The seam heuristics in Schier et al. (2018) fail for complex proxy UV maps.

(c) Our method faithfully transfers the complex texture map of the mesh proxy.

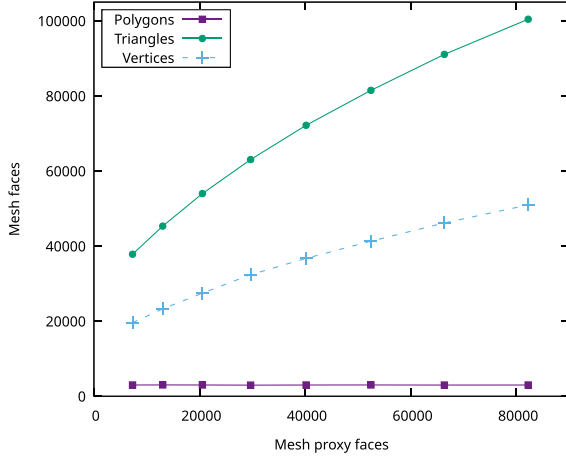
Fig. 18. Texture transfer to a triangle soup created from the Botijo model.

11.2. Mesh proxies with multiple connected components

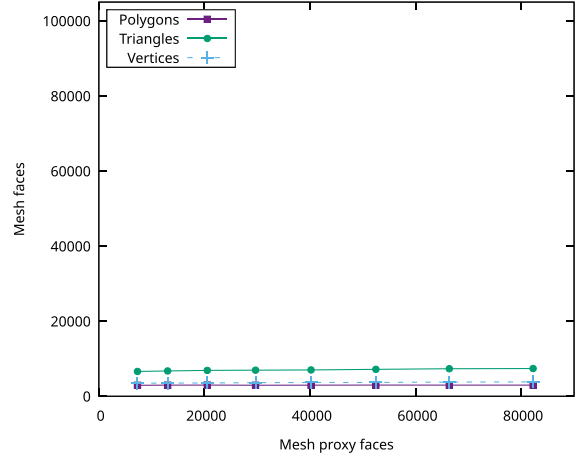
Tracing edges across the mesh proxy assumes that all vertices of an input face are projected onto the same connected component of the proxy mesh. If a mesh proxy consists of multiple connected components (cf. Fig. 21(c)), one must decide how to handle faces whose vertices are projected onto different proxy components. For polygon soups without connectivity, one possible strategy would be to clamp all vertices of a polygon to a single proxy component, for example by inverse-distance voting. For input meshes that are not polygon soups, one would either need to clamp an entire connected component of M_i to a single component of M_p , which may introduce substantial distortion, or cut the input mesh so that all faces of each resulting component are mapped to the same proxy component.

11.3. Edge approximation using geodesics

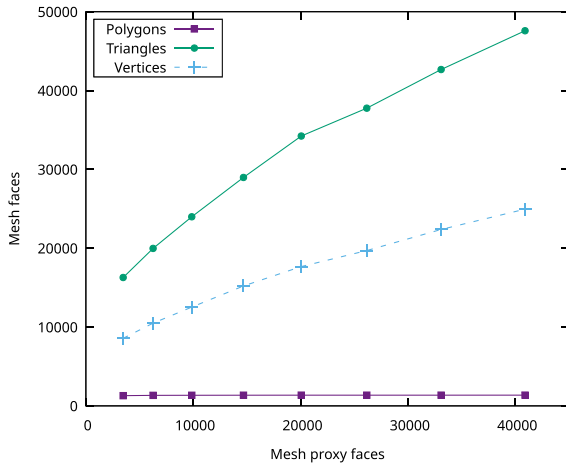
Even in the absence of UV seams, projecting only the endpoints of an input edge does not uniquely determine how that edge should be represented on the proxy surface. On a closed surface, the projected endpoints can be connected by different surface paths, and from their positions alone it is generally unclear which one best represents the projection of the input edge or whether the resulting set of chosen paths admits a valid cellular embedding. Our method resolves this ambiguity in a practical way by representing each projected input edge by a shortest geodesic on the proxy. This is a reasonable approximation when the input mesh is sufficiently fine and the mesh proxy is geometrically similar to the input, but it is not a general guarantee (cf. Born et al., 2021). A possible failure case is that the intended projected edge would follow a rough surface region of the proxy and therefore have a longer geodesic distance, while an alternative path through a smoother region is shorter but topologically incorrect. In such cases, the shortest geodesic no longer reflects the intended edge correspondence. In practice, we recommend refining coarse input meshes so that each edge is short enough to be clearly associated with a geodesic that approximates the edge projection on M_p well.



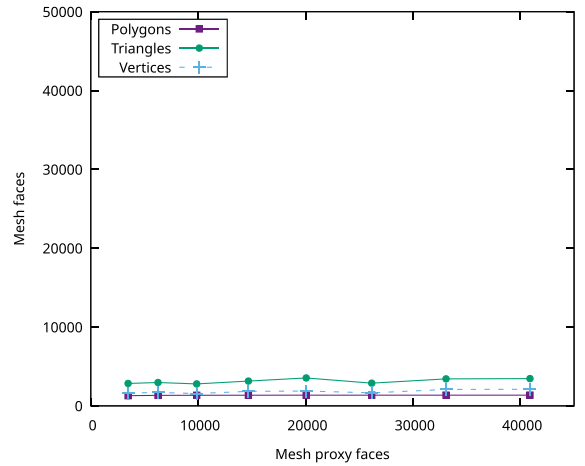
(a) Cow mesh with all intersection points



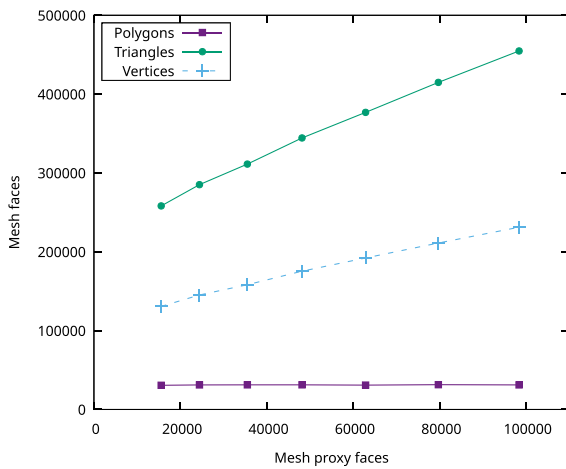
(b) Cow mesh with only relevant intersection points



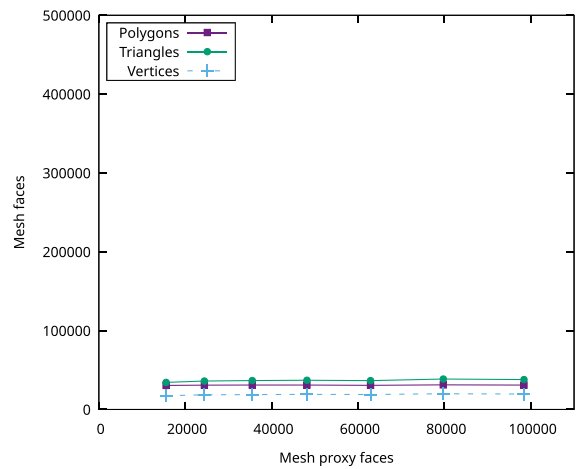
(c) NURBS mesh with all intersection points



(d) NURBS mesh with only relevant intersection points



(e) Botijo mesh with all intersection points



(f) Botijo mesh with only relevant intersection points

Fig. 19. Number of vertices, polygonal faces, and triangles in the overlay mesh M_r as a function of the number of faces of the proxy mesh M_p . Keeping all intersection points makes the output representation strongly dependent on the resolution of M_p , while the reduced mesh keeps the complexity of the result largely independent of proxy resolution.

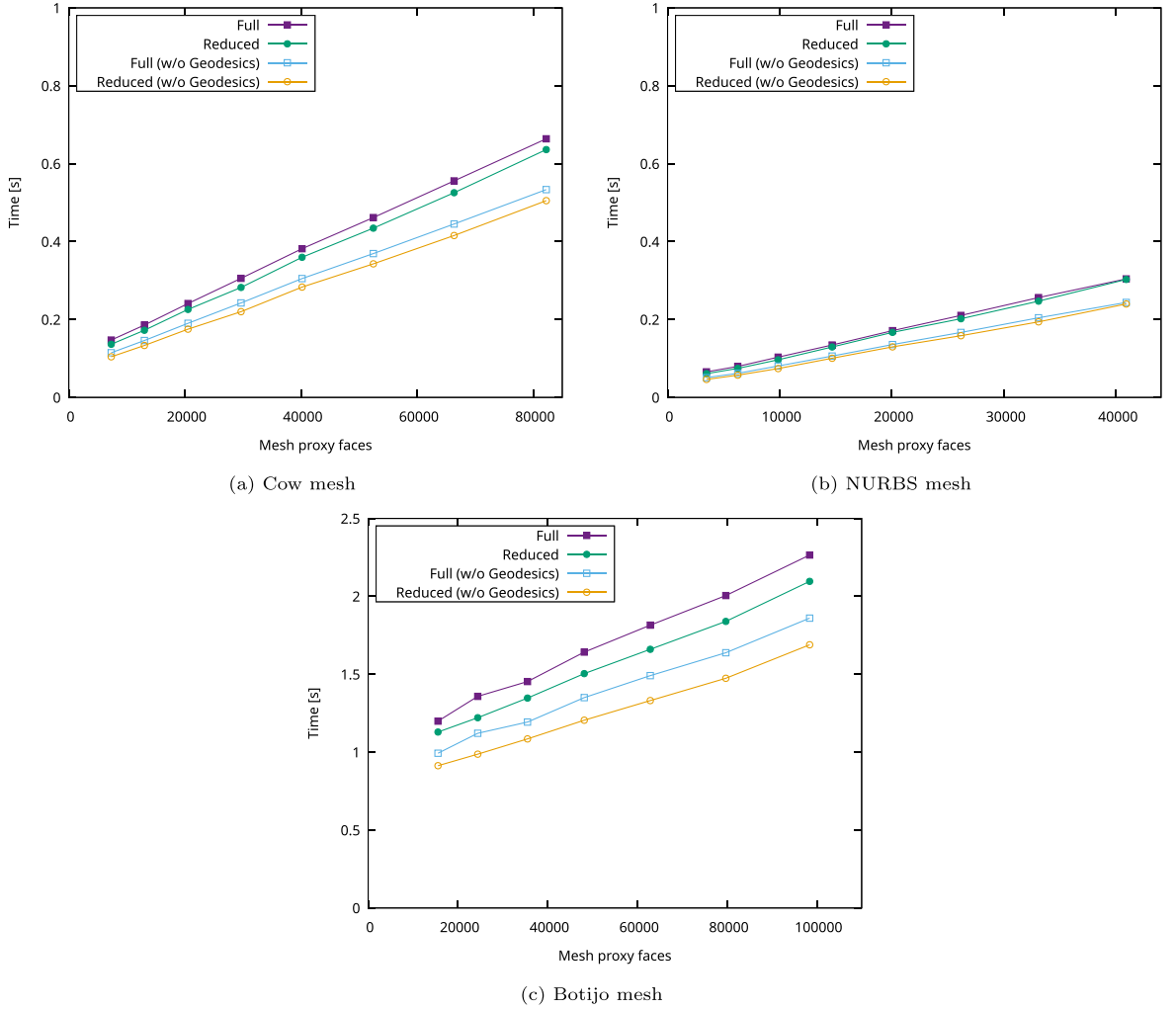


Fig. 20. Runtime of our algorithm for mesh proxies of different resolutions, for the cow and NURBS examples from Fig. 1 and the Botijo example from Fig. 18.

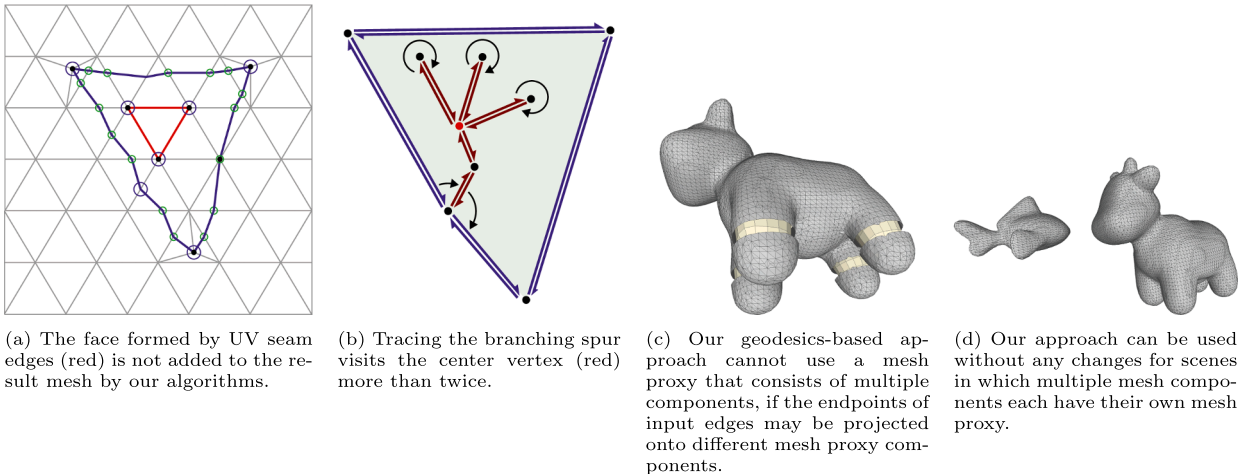


Fig. 21. Examples of limitations of our approach.

11.4. Enclosed UV seam loops

An overlay face may completely enclose a loop of UV seam edges of M_p that does not intersect any projected input edge, as illustrated in Fig. 21(a). Our edge-construction step in Section 7.1 still creates all halfedges for this seam loop, but the face-building algorithm in Section 8 starts tracing only from segments of geodesic paths and never from UV seam edges, so that loops solely consisting of seam edges are not extracted as faces. Therefore these loops are not added to the final result mesh.

In principle, we could detect such cases by checking in Algorithm 3 whether edges of the non-manifold overlay remain unused after face extraction. However, inserting them directly would produce faces with interior boundary loops, which would require additional tessellation before the result could be represented by standard mesh data structures and used in downstream applications. We therefore do not handle this case explicitly and instead assume that sufficiently small enclosed seam loops can be omitted.

11.5. Handling spur faces

While spur faces close the mesh topologically, no meaningful UV coordinates can be assigned to them because the spur itself represents a texture discontinuity. We therefore assign the UV coordinates (0,0) to spur faces to make explicit that they are purely topological auxiliary faces. One way to avoid spur faces completely would be to extend UV spurs until they intersect a geodesic path and thereby split the overlay face in a well-defined manner. However, there are several possible extension criteria, such as choosing the shortest extension, the straightest continuation, or an extension aligned with mesh features, and we leave this design choice to future work. Another challenge arises for more complex spur configurations, for example when a UV spur branches inside a face, as depicted in Fig. 21(b). The face-tracing step can still duplicate revisited vertices in such cases, but the current construction of auxiliary spur faces is described only for a single non-branching spur. Extending this crack-closing step to branching spurs and related corner cases would require a more general treatment and is left to future work.

11.6. Intersections between input edges

If faces overlap, as in polygon soups or meshes with self-intersections, our approach does not attempt to construct intersections between projected input edges. Instead, it reproduces the overlaps and inconsistencies of M_i in the result mesh M_r and leaves their resolution to downstream processing if needed, while itself only creating the seam-induced subdivisions required for including seams.

Resolving such overlaps, for example to recover a more consistent mesh on the proxy surface, would require constructing intersections between projected geodesic edge representations. Unlike seam edges, which are already edges of the mesh proxy M_p , projected input edges are only overlaid on M_p to determine their breakpoints and are not inserted into its triangulation, so our current construction does not provide these intersections automatically. A potentially robust way to handle this would be to detect crossings locally inside a face of M_p from the cyclic order of the polyline intersections with the face boundary. Since each proxy face is a triangle and each geodesic segment is linear within that face, this boundary order determines whether two segments must cross in its interior. After fixing this crossing topology, the corresponding geometric intersection points could then be computed numerically, and numerical errors would affect only the geometric placement of the intersections while the resulting mesh topology remains consistent. This approach would still require additional case distinctions for degenerate configurations, such as tangential contacts, shared endpoints, or multiple geodesics intersecting in the same face. In particular, configurations with more than two intersecting geodesic segments in one face would require an iterative or more global construction and are left for future work.

11.7. Future work

Beyond addressing these limitations, it would be interesting to integrate curve embedding and merging more closely with engineering workflows. In this paper, we merge inconsistent meshes with seam edges on a given mesh proxy, but in principle the same construction could be used to join multiple objects directly. For example, one could project an intersection curve of two objects onto a topologically consistent carrier mesh and then use our algorithm to stitch the meshes along the projected curve. For texturing, it would also be interesting to study how mesh segmentations can be transferred between similar objects using a shared carrier mesh. In such a setting, UV seams would be projected from one source object to the carrier and then transferred to other meshes that reuse the same proxy.

12. Conclusion

We introduced a robust method for embedding curves defined on a mesh proxy into inconsistent input meshes, including polygon soups and disjoint patch meshes. Instead of transferring only vertex positions, we represent input edges on the proxy by shortest-geodesic polylines and merge them with edge paths defined on the proxy, such as UV seams. This yields a reconstructed result mesh in which discontinuities in the UV map are represented explicitly by mesh edges, thereby enabling a consistent parameterization of the inconsistent input using the UV map of the mesh proxy. In contrast to prior approaches that rely on UV-space heuristics or local refinement near seams, our construction inserts the required seam-induced subdivisions directly into the reconstructed mesh and preserves polygonal connectivity where possible. The method is numerically robust because it avoids explicit edge-edge intersection tests and instead recovers the required connectivity from local topology on the mesh proxy. In our texture transfer application, this allows us to reproduce the UV chart boundaries of the mesh proxy accurately in the result mesh and eliminate the visible seam artifacts of prior heuristic methods. The method also handles configurations that heuristic seam correction cannot represent structurally, such as cut-out regions that collapse when projected vertices are clamped to surrounding UV seams. More generally, the proposed construction suggests viewing the mesh proxy as a carrier surface on which one or more curve networks can be defined or projected and then merged using the proxy topology. Beyond UV seams, the method may be useful in engineering workflows where design curves, segmentation boundaries, or interface curves must be transferred consistently between meshes with incompatible or incomplete connectivity.

CRedit authorship contribution statement

Alexander Schier: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Formal analysis, Conceptualization. **Reinhard Klein:** Supervision, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Full parameterization pipeline

Our parameterization pipeline shown in Fig. A.22 consists of two stages. First, we reuse the pipeline of Schier et al. (2018) to project the vertices of M_i onto the mesh proxy M_p and assign UV coordinates from the parameterization of M_p . Starting from this parameterized mesh proxy with projected vertices, we then insert the vertices of M_i into M_p , compute shortest-geodesic edge representations between vertices connected by an edge in M_i , and combine these projected edge paths with the UV seam edges of M_p . We merge the resulting geodesic segments and UV seam edges into an overlaid curve network, from which we reconstruct a merged mesh M_r by tracing closed loops. Since the UV seam edges already carry two-sided UV coordinates, the reconstructed mesh inherits the UV chart boundaries of M_p automatically.

The full pipeline proceeds as follows:

1. Starting from the input mesh, we construct an offset mesh proxy, segment it into UV charts, and parameterize the mesh proxy.
2. In parallel, we compute electrostatic monopole field lines on the input mesh.
3. We then intersect the field lines with the mesh proxy to project the vertices of M_i onto M_p . The projection positions are then used both to assign UV coordinates to the vertices of M_i and to insert the corresponding projected vertices into M_p .
4. For each input edge, we compute a shortest-geodesic polyline on the proxy between its projected endpoints and split it at proxy-edge crossings into geodesic segments.
5. We extract the UV seam edges of the parameterized mesh proxy and join them topologically with the geodesic segments to obtain a merged curve network.
6. Finally, we extract closed edge loops from this network, reconstruct the merged mesh, and align it with the input mesh to obtain the result mesh.

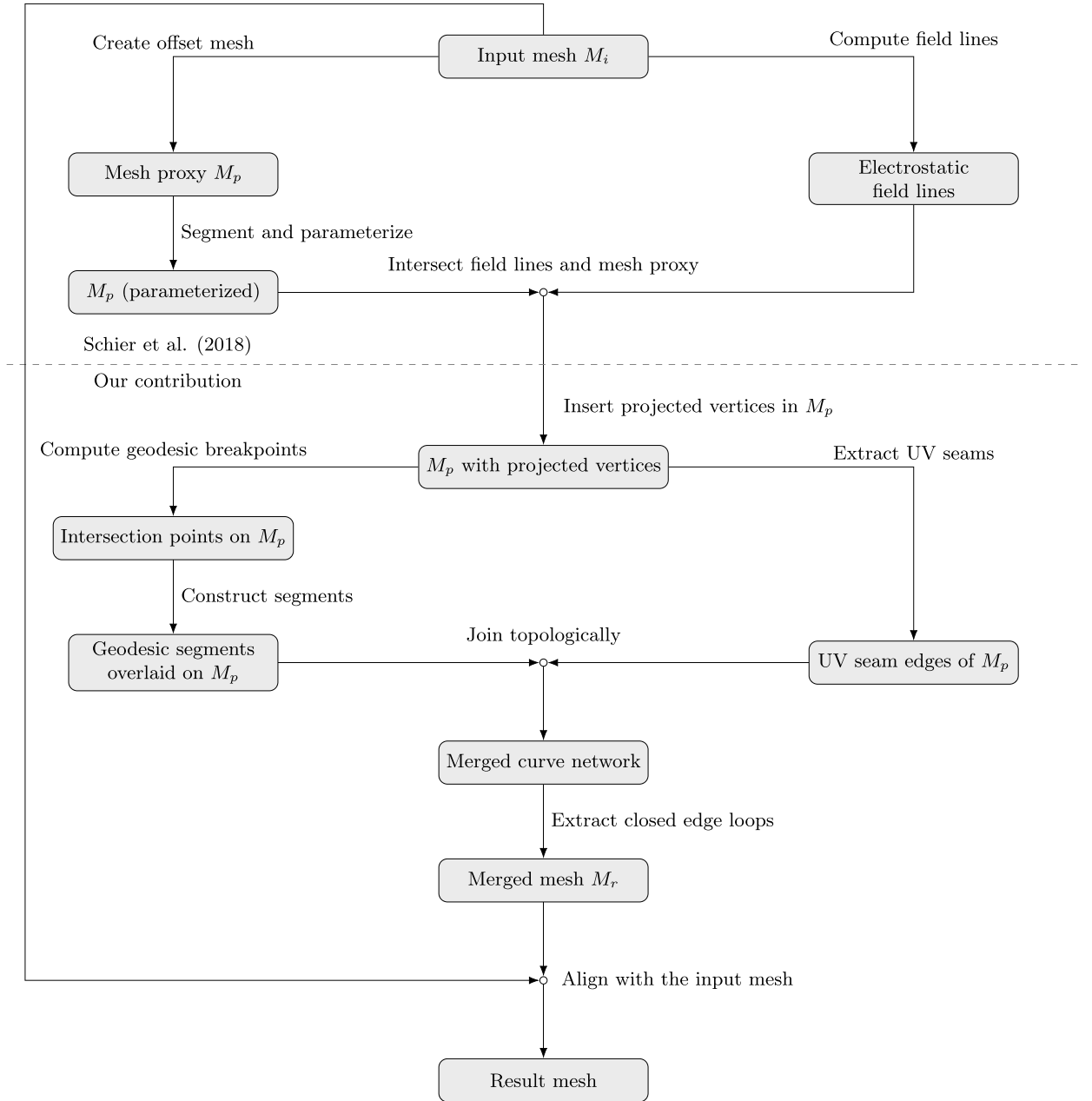
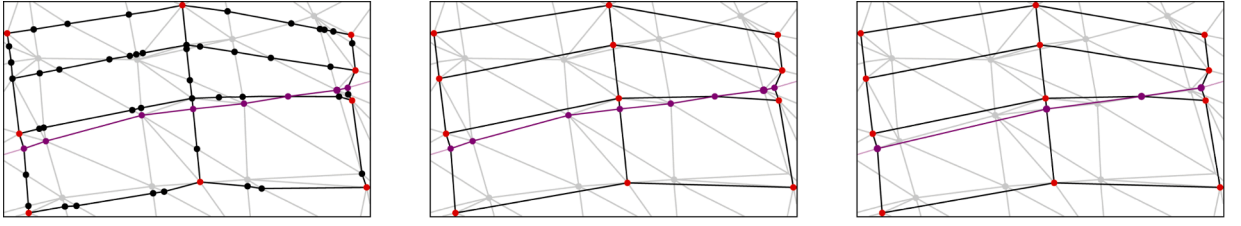


Fig. A.22. The full parameterization pipeline. The upper part follows (Schier et al., 2018) to construct a parameterized mesh proxy and compute vertex correspondences. The lower part is our contribution, which replaces the UV-space seam heuristics by a Seam-aware reconstruction of the result mesh.

Appendix B. Overlay mesh complexity trade-offs

We compare three ways of representing an overlay face on the mesh proxy. In the full representation (Fig. B.23(a)), all traced breakpoints of the geodesic edge paths are kept. This follows the proxy geometry most faithfully, but introduces many short boundary segments that are not required for the seam topology. In the reduced representation (Fig. B.23(b)), we keep only the relevant geodesic intersection points together with the vertices of the UV seam polylines. This is the representation we suggest for most use cases, as it still represents cuts at UV seams exactly while keeping the result mesh substantially simpler.

For some simple configurations, one can simplify the UV seam polylines further by removing inner seam vertices, as shown in Fig. B.23(c). This can further reduce the mesh complexity, but the seam is then only approximated rather than reproduced exactly. In particular, the simplified seam segments may span non-convex chart boundaries or even cross each other, which can lead to overlapping sub-faces.



(a) The mesh containing all edge intersection points captures all details of the mesh proxy, but has many segments that are not strictly required for texture projection.

(b) The reduced mesh only contains topologically required points and points belonging to UV segments, which is sufficient to represent cuts at UV seams exactly.

(c) Removing inner vertices of UV seams can further simplify the mesh at the cost of UV charts boundaries possibly not being followed precisely.

Fig. B.23. Three representations of the overlay mesh M_r (black) on the mesh proxy M_p (gray) as a curve network between projected vertices (red) and UV seams (purple): (a) full representation with all traced geodesic breakpoints, (b) reduced representation keeping only relevant geodesic points and all seam vertices, and (c) further seam simplification obtained by removing inner seam vertices, which only approximates the seam geometry. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Appendix C. Algorithms

The following algorithms construct the local edge-stars of the overlay mesh and then extract the faces of the result mesh from the resulting curve network.

In the pseudocode, we write $v \sim w$ if v and w are adjacent along a geodesic path or UV seam, and $p \in h$ if the intersection point p lies on the halfedge h . For a halfedge $h = [w, v]$, oriented from w to v , we define

$$\text{closest}(h, v) = \begin{cases} \operatorname{argmin}_{p \in h} \|p - v\| & \text{if } \exists p: p \in h, \\ w & \text{otherwise.} \end{cases}$$

If $p_1, \dots, p_n \in h$, we assume that these intersection points are ordered along the orientation of the halfedge h , from $h.\text{from}$ to $h.\text{to}$.

For each undirected edge of M_r , we create at most one halfedge pair. In the pseudocode, $\text{get_halfedge}([v, w])$ returns the halfedge $[v, w]$ and creates the pair $[v, w], [w, v]$ if it does not exist yet. The notation $h.\text{next}$, $h.\text{prev}$, and $h.\text{opp}$ is the usual one for a halfedge mesh. Since the algorithms preserve valid boundary loops, $h.\text{prev}$ can also be treated as an implicit relation, as in the implementation in [Appendix D](#).

Algorithm 1: Create an edge-star at a vertex.

Input: A vertex v of the mesh proxy M_p , and a halfedge $h_{\text{start}} = [w, v]$.

Output: Cyclically ordered list of halfedges incoming to v in M_r .

result $\leftarrow \emptyset$;

$h \leftarrow h_{\text{start}}$;

do

 // Point on the star edge

$p_{\text{edge}} \leftarrow \text{closest}(h, v)$;

if $p_{\text{edge}} \sim v$ **then**

 | result.append(get_halfedge($[p_{\text{edge}}, v]$));

end

 // Points on the boundary edge

foreach $p_i \leftarrow p_1, \dots, p_n \in h.\text{prev.opp}$ **do**

if $p_i \sim v$ **then**

 | result.append(get_halfedge($[p_i, v]$));

end

end

 // Next incoming star edge

$h \leftarrow h.\text{next.opp}$;

while $h \neq h_{\text{start}}$;

 // Build the edge-star of v in M_r

$n \leftarrow |\text{result}|$;

for $i \leftarrow 1$ **to** n **do**

 | result[i].next $\leftarrow$ result[$(i \bmod n) + 1$].opp;

end

Algorithm 2: Create an edge-star at an edge intersection point.**Input:** Intersection point $p = (e, t)$ with $e = (h_0, h_1)$ being an edge in M_p **Output:** Cyclically ordered list of halfedges incoming to p in M_r .

```

result  $\leftarrow \emptyset$ ;
// (1) Segment from  $h_0$ .from side to  $p$ 
if  $p \sim \text{closest}(h_0, p)$  then
  | result.append(get_halfedge([closest( $h_0, p$ ),  $p$ )]));
end
// (2) Diamond edge
foreach  $p_i \leftarrow p_1, \dots, p_n \in h_0.\text{prev.opp}$  do
  | if  $p_i \sim p$  then
  | | result.append(get_halfedge([ $p_i, p$ )]));
  | end
end
// (3) Vertex opposite to  $e$  in the face of  $h_0$ 
if  $h_0.\text{prev.from} \sim p$  then
  | result.append(get_halfedge([ $h_0.\text{prev.from}, p$ )]));
end
// (4) Diamond edge
foreach  $p_i \leftarrow p_n, \dots, p_1 \in h_0.\text{prev.prev.opp}$  do
  | if  $p_i \sim p$  then
  | | result.append(get_halfedge([ $p_i, p$ )]));
  | end
end
// (5) Segment from  $h_1$ .from side to  $p$ 
if  $p \sim \text{closest}(h_1, p)$  then
  | result.append(get_halfedge([closest( $h_1, p$ ),  $p$ )]));
end
// (6) Diamond edge
foreach  $p_i \leftarrow p_1, \dots, p_n \in h_1.\text{prev.opp}$  do
  | if  $p_i \sim p$  then
  | | result.append(get_halfedge([ $p_i, p$ )]));
  | end
end
// (7) Vertex opposite to  $e$  in the face of  $h_1$ 
if  $h_1.\text{prev.from} \sim p$  then
  | result.append(get_halfedge([ $h_1.\text{prev.from}, p$ )]));
end
// (8) Diamond edge
foreach  $p_i \leftarrow p_1, \dots, p_n \in h_1.\text{prev.prev.opp}$  do
  | if  $p_i \sim p$  then
  | | result.append(get_halfedge([ $p_i, p$ )]));
  | end
end
// Build the edge-star of  $p$  in  $M_r$ 
 $n \leftarrow |\text{result}|$ ;
for  $i \leftarrow 1$  to  $n$  do
  | result[ $i$ ].next  $\leftarrow$  result[( $i \bmod n$ ) + 1].opp;
end

```


Algorithm 3: Build faces by tracing loops of halfedges.

Input: The set of halfedges of the non-manifold mesh M_r .
Output: A set of polygonal faces extracted from M_r .
 $\text{faces} \leftarrow \emptyset$;
 $\text{start_halfedges} \leftarrow$ halfedges on geodesic paths that are not boundary halfedges;
while $\text{start_halfedges} \neq \emptyset$ **do**
 $h_{\text{start}} \leftarrow \text{start_halfedges.first}$ // Arbitrary start halfedge
 $h \leftarrow h_{\text{start}}$;
 $\text{face} \leftarrow []$ // Ordered vertex list
 do
 $v \leftarrow h.\text{to}$;
 if $v \in \text{face}$ **then**
 // New vertex with the same coordinates
 $v \leftarrow \text{duplicate}(h.\text{to})$;
 end
 $\text{face.append}(v)$;
 $\text{start_halfedges} \leftarrow \text{start_halfedges} \setminus \{h\}$;
 $h \leftarrow h.\text{next}$;
 while $h \neq h_{\text{start}}$;
 $\text{faces} \leftarrow \text{faces} \cup \{\text{face}\}$;
end

Appendix D. Non-manifold halfedge structure

For the incremental construction of the overlay mesh, we use a minimal non-manifold halfedge structure that supports hanging-edge configurations. Compared to a standard halfedge structure (Muller and Preparata, 1978), it differs by not storing faces, and by allowing the *next* halfedge to point to the *opposite* halfedge. For the texture-transfer application, we additionally store UV coordinates per halfedge. The structure is deliberately kept simple and does not store precomputed previous-pointers, since this would introduce additional consistency constraints in each low-level update operation. Listing 1 gives the minimal data structure needed to store these attributes and Listing 2 defines the routines for creating or retrieving halfedges and inserting them into edge-stars.

```

1 class Vertex {
2     Halfedge outgoing_halfedge;
3     Vec3 position;
4 }
5 class Halfedge {
6     Vertex to;
7     Halfedge opposite;
8     Halfedge next;
9     Vec2 uv;
10    Vertex from() {
11        return opposite.to;
12    }
13    Halfedge previous() {
14        Halfedge prev=next;
15        while(prev.next != this) {
16            prev=prev.next;
17        }
18        return prev;
19    }
20 }
21 class HalfedgePair {
22     Halfedge h0;
23     Halfedge h1;
24 }
25 // Map associating ordered vertex pairs
26 // with halfedges, so only one halfedge pair
27 // is created per edge.
28 Map<Pair<Vertex, Vertex>, Halfedge>halfedges;
```

Listing 1. Our simple data structure stores only the minimal topology and per-element attributes required by the construction without an explicit face representation or stored previous-pointers.

```

1 HalfedgePair create_halfedge_pair(Vertex v, Vertex w) {
2     Halfedge h0, h1;
3     h0.to = w;
4     h1.to = v;
5     h0.opposite = h1;
6     h1.opposite = h0;
7     h0.next = h1;
8     h1.next = h0;
9     if(v.outgoing_halfedge == null) {
10        v.outgoing_halfedge = h0;
11    }
12    if(w.outgoing_halfedge == null) {
13        w.outgoing_halfedge = h1;
14    }
15    halfedges[(v, w)] = h0;
16    halfedges[(w, v)] = h1;
17    return {h0, h1};
18 }
19 Halfedge get_halfedge(Vertex v, Vertex w) {
20     h0 = halfedges[(v, w)];
21     if(h0 == null) {
22         h0 = create_halfedge_pair(v, w).h0;
23     }
24     return h0;
25 }
26 // Insert a new incoming halfedge into the
27 // edge-star of star_halfedge.to
28 // immediately after star_halfedge
29 void insert_after(Halfedge star_halfedge, Halfedge halfedge) {
30     halfedge.next = star_halfedge.next;
31     star_halfedge.next = halfedge.opposite;
32 }

```

Listing 2. Low-level routines for initializing hanging halfedges and inserting them into edge-stars.

Appendix E. Ablation study for the UV heuristics

We also evaluate how the field line method behaves without one or both of the UV heuristics proposed by Schier et al. (2018). In the parameterization without any fixes, shown in Fig. E.24(a), the most noticeable seam artifacts are the rainbow-like stripes. They are caused by triangles that span a large distance in UV space, often together with a flipped UV embedding. Triangles that connect vertices on adjacent charts across only a small gap produce milder distortion, for example at the seam between the leg and the neck. In many real-world UV maps, however, charts may be placed farther apart or rotated differently, which makes these artifacts more severe.

In Fig. E.24(b), we omit the flip heuristic, so each triangle is clamped to a single chart but wrap-around triangles are not corrected. As a result, there are still triangles whose UV embedding spans a long distance within one chart. The seam between the leg and the neck is almost free of compression artifacts, but the seam between the front and rear legs still shows visible distortion. In Fig. E.24(c), we apply only the heuristic that embeds flipped triangles rigidly and do not clamp triangles to a single UV chart. As a result, triangles with vertices on multiple UV charts still span the gaps between charts in the UV map. The rigid embedding corrects triangles that are also detected as flipped, but the other triangles spanning across charts still lead to visible artifacts.

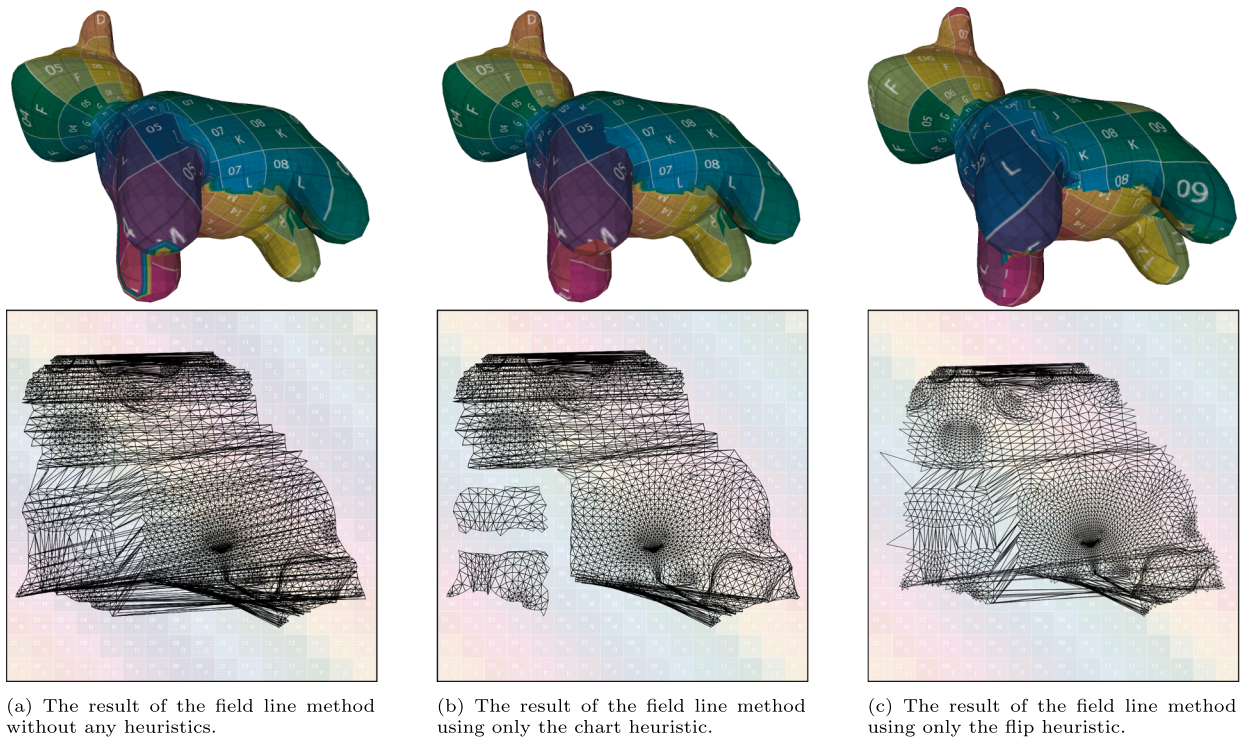


Fig. E.24. Results of the field line projection when one or both of the UV heuristics used in Fig. 15(b) are disabled.

Data availability

Data will be made available on request.

References

- Barill, G., Dickson, N.G., Schmidt, R., Levin, D.I.W., Jacobson, A., 2018. Fast winding numbers for soups and clouds. *ACM Trans. Graph. (TOG)* 37 (4), 1–12.
- Born, J., Schmidt, P., Kobbelt, L., 2021. Layout embedding via combinatorial optimization. *Comput. Graph. Forum* 40 (2), 277–290 (en).
- Bose, P., Maheshwari, A., Shu, C., Wuhler, S., 2011. A survey of geodesic paths on 3D surfaces. *Comput. Geom.* 44 (9), 486–498 (en).
- Boyer, E., Bronstein, A.M., Bronstein, M.M., Bustos, B., Darom, T., Horaud, R., Hotz, I., Keller, Y., Keustermans, J., Kovnatsky, A., Litman, R., Reininghaus, J., Sipiran, I., Smeets, D., Suetens, P., Vandermeulen, D., Zaharescu, A., Zobel, V., Apr 2011, This is a full version of the SHREC'11 report published in 3DOR, pp. SHREC 2011: robust feature detection and description benchmark. In: 3DOR2011 - Eurographics Workshop on 3D Object Retrieval. ACM Siggraph, Llandudno, United Kingdom, pp. 71–78.
- Büscher, K.J., Degel, J.P., Oellerich, J., 2024. A comprehensive survey of isocontouring methods: applications, limitations and perspectives. *Algorithms* 17 (2), 83.
- Calderon, S., Boubekeur, T., 2017. Bounding proxies for shape approximation. *ACM Trans. Graph.* 36 (4), 57–1.
- Casti, S., Livesu, M., Mellado, N., Rumman, N.A., Scateni, R., Barthe, L., Puppo, E., 2019. Skeleton based cage generation guided by harmonic fields. *Comput. Graph.* 81, 140–151.
- Chen, J., Han, Y., 1990. Shortest paths on a polyhedron. In: *Proceedings of the Sixth Annual Symposium on Computational Geometry - SCG '90*. ACM Press, Berkeley, California, United States, pp. 360–369 (en).
- Chow, B., Luo, F., 2003. Combinatorial ricci flows on surfaces. *J. Differ. Geom.* 63 (1), 97–129.
- Corman, E., Crane, K., 2025. Rectangular surface parameterization. *ACM Trans. Graph. (TOG)* 44 (4), 1–21.
- Crane, K., Pinkall, U., Schröder, P., 2013a. Robust fairing via conformal curvature flow. *ACM Trans. Graph. (TOG)* 32 (4), 1–10.
- Crane, K., Weischedel, C., Wardetzky, M., 2013b. Geodesics in heat: a new approach to computing distance based on heat flow. *ACM Trans. Graph. (TOG)* 32 (5), 1–11.
- Crane, K., Livesu, M., Puppo, E., Qin, Y., Jul 2020. A survey of algorithms for geodesic paths and distances, arXiv:2007.10430.
- Degener, P., Klein, R., 2007. Texture atlas generation for inconsistent meshes and point sets. In: *IEEE International Conference on Shape Modeling and Applications 2007 (SMI'07)*. IEEE, pp. 156–168.
- Fu, X.-M., Liu, Y., 2016. Computing inversion-free mappings by simplex assembly. *ACM Trans. Graph. (TOG)* 35 (6), 216.
- Fu, X.-M., Liu, Y., Guo, B., 2015. Computing locally injective mappings by advanced MIPS. *ACM Trans. Graph. (TOG)* 34 (4), 71.
- Gillespie, M., Springborn, B., Crane, K., 2021. Discrete conformal equivalence of polyhedral surfaces. *ACM Trans. Graph. (TOG)* 40 (4), 1–20.
- Gong, S., Chen, L., Bronstein, M., Zafeiriou, S., 2019. Spiralnet + +: a fast and highly efficient mesh convolution operator. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pp. 0.
- Greif, A., Klein, R., 2004. Efficient representation and extraction of 2-manifold isosurfaces using kd-trees. *Graph. Models* 66 (6), 370–397.
- Hanocka, R., Hertz, A., Fish, N., Giryas, R., Fleishman, S., Cohen-Or, D., 2019. Meshcnn: a network with an edge. *ACM Trans. Graph. (TOG)* 38 (4), 1–12.
- Hormann, K., Polthier, K., Sheffer, A., 2008. Mesh parameterization: theory and practice, SIGGRAPH Asia '08. In: *ACM SIGGRAPH ASIA 2008 Courses*. ACM, New York, NY, USA, pp. 12:1–12:87).
- Jacobson, A., Kavan, L., Sorkine-Hornung, O., 2013. Robust inside-outside segmentation using generalized winding numbers. *ACM Trans. Graph. (TOG)* 32 (4), 1–12.
- Jin, M., Kim, J., Luo, F., Gu, X., 2008. Discrete surface ricci flow. *IEEE Trans. Vis. Comput. Graph.* 14 (5), 1030–1043.
- Kanai, T., Suzuki, H., 2001. Approximate shortest path on a polyhedral surface and its applications. *Comput.-aided Des.* 33 (11), 801–811 (en).
- Kazhdan, M., Hoppe, H., 2013. Screened poisson surface reconstruction. *ACM Trans. Graph. (TOG)* 32 (3), 1–13.

- Kazhdan, M., Bolitho, M., Hoppe, H., 2006. Poisson surface reconstruction. In: Proceedings of the Fourth Eurographics Symposium on Geometry Processing, SGP '06. Eurographics Association, Goslar, DEU, pp. 61–70.
- Levi, Z., 2022. Seamless parametrization of spheres with controlled singularities. *Comput. Graph. Forum* 41 (1), 57–68 (en).
- Levi, Z., 2023. Shear-reduced seamless parametrization. *Comput. Aided Geom. Des.* 101, 102179.
- Li, M., Kaufman, D.M., Kim, V.G., Solomon, J., Sheffer, A., 2018. Optcuts: joint optimization of surface cuts and parameterization. *ACM Trans. Graph. (TOG)* 37 (6), 1–13.
- Li, Y., Cheung, V., Liu, X., Chen, Y., Luo, Z., Lei, B., Weng, H., Zhao, Z., Huang, J., Chen, Z., Guo, C., Jun 2025. Auto-regressive surface cutting. [arXiv:2506.18017](https://arxiv.org/abs/2506.18017).
- Liu, R., Aigerman, N., Kim, V.G., Hanocka, R., Jun 2023. DA wand: distortion-aware selection using neural mesh parameterization. In: 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, Vancouver, BC, Canada, pp. 16739–16749 (en).
- Lorensen, W.E., Cline, H.E., 1987. Marching cubes: a high resolution 3D surface construction algorithm. *ACM SIGGR. Comput. Graph.* 21 (4), 163–169 (en).
- Lucquin, V., Boubekur, T., 2017. SeamCut: interactive mesh segmentation for parameterization. In: ACM SIGGRAPH Asia Technical Brief.
- Luo, F., 2004. Combinatorial yamabe flow on surfaces. *Commun. Contemp. Math.* 6 (5), 765–780.
- Mancinelli, C., Nazzaro, G., Pellacini, F., Puppo, E., 2023. b/Surf: interactive bézier splines on surface meshes. *IEEE Trans. Vis. Comput. Graph.* 29 (7), 3419–3435.
- Mitchell, J.S.B., Mount, D.M., Papadimitriou, C.H., 1987. The discrete geodesic problem. *SIAM J. Comput.* 16, 647–668.
- Muller, D.E., Preparata, F.P., 1978. Finding the intersection of two convex polyhedra. *Theor. Comput. Sci.* 7 (2), 217–236.
- Myles, A., Zorin, D., 2012. Global parametrization by incremental flattening. *ACM Trans. Graph. (TOG)* 31 (4), 109.
- Newman, T.S., Yi, H., 2006. A survey of the marching cubes algorithm. *Comput. Graph.* 30 (5), 854–879.
- Novotni, M., Klein, R., 2002. Computing geodesic distances on triangular meshes. In: The 10-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2002 (WSCG'2002).
- Peyré, G., Péchaud, M., Keriven, R., Cohen, L.D., 2010. Geodesic methods in computer vision and graphics. *Found. Trends® Comput. Graph. Vis.* 5 (3–4), 197–397.
- Poranne, R., Tarini, M., Huber, S., Panozzo, D., Sorkine-Hornung, O., 2017. Autocuts: simultaneous distortion and cut optimization for UV mapping. *ACM Trans. Graph. (TOG)* 36 (6), 215.
- Rabinovich, M., Poranne, R., Panozzo, D., Sorkine-Hornung, O., 2017. Scalable locally injective mappings. *ACM Trans. Graph. (TOG)* 36 (2), 16.
- Rashad, M., Khamiss, M., Mousa, M., 2017. A review on mesh segmentation techniques. *Int. J. Eng. Innov. Technol.* 6, 18–26.
- Rodrigues, R.S.V., Morgado, J.F.M., Gomes, A.J.P., 2018. Part-based mesh segmentation: a survey. *Comput. Graph. Forum* 37 (6), 235–274.
- Sacht, L., Vouga, E., Jacobson, A., 2015. Nested cages. *ACM Trans. Graph. (TOG)* 34 (6), 1–14.
- Sahillioğlu, Y., 2020. Recent advances in shape correspondence. *Vis. Comput.* 36 (8), 1705–1721 (en).
- Sawhney, R., Crane, K., 2017. Boundary first flattening. *ACM Trans. Graph. (TOG)* 37 (1), 1–14.
- Schaefer, S., Ju, T., Warren, J., 2007. Manifold dual contouring. *IEEE Trans. Vis. Comput. Graph.* 13 (3), 610–619 (en).
- Schier, A., Hartmann, S., Klein, R., 2018. Fast texture mapping for triangle soups using electrostatic monopole field lines. *Comput. Graph.* 77, 140–155.
- Sethian, J.A., 1996. A fast marching level set method for monotonically advancing fronts. *Proc. Natl. Acad. Sci.* 93 (4), 1591–1595.
- Shamir, A., 2008. A survey on mesh segmentation techniques. *Comput. Graph. Forum* 27 (6), 1539–1556.
- Sharp, N., Crane, K., 2020. You can find geodesic paths in triangle meshes by just flipping edges. *ACM Trans. Graph.* 39 (6), 1–15 (en).
- Sharp, N., Soliman, Y., Crane, K., 2019. Navigating intrinsic triangulations. *ACM Trans. Graph. (TOG)* 38 (4), 1–16.
- Sharp, N., Attai, S., Crane, K., Ovsjanikov, M., 2022. Diffusionnet: discretization agnostic learning on surfaces. *ACM Trans. Graph. (TOG)* 41 (3), 1–16.
- Sheffer, A., Praun, E., Rose, K., 2007. Mesh parameterization methods and their applications. *Found. Trends® Comput. Graph. Vis.* 2 (2), 105–171.
- Shen, C., O'Brien, J.F., Shewchuk, J.R., 2004. Interpolating and approximating implicit surfaces from polygon soup. In: ACM SIGGRAPH 2004 Papers. Association for Computing Machinery, New York, NY, USA, pp. 896–904, SIGGRAPH '04.
- Shu, R., Zhou, C., Kankanhalli, M.S., 1995. Adaptive marching cubes. *Vis. Comput.* 11, 202–217.
- Springborn, B., Schröder, P., Pinkall, U., 2008. Conformal equivalence of triangle meshes. In: ACM SIGGRAPH 2008 Papers. Association for Computing Machinery, New York, NY, USA, SIGGRAPH '08.
- Ströter, D., Thiery, J.M., Hormann, K., Chen, J., Chang, Q., Besler, S., Mueller-Roemer, J.S., Boubekur, T., Stork, A., Fellner, D.W., 2024. A survey on cage-based deformation of 3D models. *Comput. Graph. Forum* 43 (2), e15060 (en).
- Tang, J., Wu, G.-S., Zhang, F.-Y., Zhang, M.-M., 2007. Fast approximate geodesic paths on triangle mesh. *Int. J. Autom. Comput.* 4 (1), 8–13.
- Tao, J., Zhang, J., Deng, B., Fang, Z., Peng, Y., He, Y., 2019. Parallel and scalable heat methods for geodesic distance computation. en, 579–594, [arXiv:1812.06060](https://arxiv.org/abs/1812.06060).
- Teimury, F., Roy, B., Casallas, J.S., MacDonald, D., Coates, M., Dec 2020. GraphSeam: supervised graph learning framework for semantic UV mapping, [arXiv:2011.13748](https://arxiv.org/abs/2011.13748).
- van Kaick, O., Zhang, H., Hamarneh, G., Cohen-Or, D., 2011. A survey on shape correspondence: O. van kaick et al. / A survey on shape correspondence. *Comput. Graph. Forum* 30 (6), 1681–1707 (en).
- Wang, Y., Asafi, S., van Kaick, O., Zhang, H., Cohen-Or, D., Chen, B., 2012. Active co-analysis of a set of shapes. *ACM Trans. Graph. (TOG)* 31 (6), 165.
- Wang, H., Sidorov, K.A., Sandilands, P., Komura, T., 2013. Harmonic parameterization by electrostatics. *ACM Trans. Graph. (TOG)* 32 (5), 1–12.
- Wang, Q., Zhang, W.-X., Cheng, Y.-Y., Liu, L., Fu, X.-M., 2023. Practical construction of globally injective parameterizations with positional constraints. *Comput. Vis. Media* 9 (2), 265–277.
- Xian, C., Lin, H., Gao, S., 2009. Automatic generation of coarse bounding cages from dense meshes. In: 2009 IEEE International Conference on Shape Modeling and Applications. IEEE, pp. 21–27.
- Xin, S.-Q., Wang, G.-J., 2009. Improving chen and han's algorithm on the discrete geodesic problem. *ACM Trans. Graph.* 28 (4), 1–8 (en).
- Yan, D.-M., Wang, W., Liu, Y., Yang, Z., 2012. Variational mesh segmentation via quadric surface fitting. *Comput.-aided Des.* 44 (11), 1072–1082.
- Ying, X., Wang, X., He, Y., 2013. Saddle vertex graph (SVG): a novel solution to the discrete geodesic problem. *ACM Trans. Graph.* 32 (6), 1–12 (en).
- Zhang, M., Guo, R., Zeng, W., Luo, F., Yau, S.-T., Gu, X., 2014. The unified discrete surface ricci flow. *Graph. Models* 76 (5), 321–339.
- Zhang, Y.-X., Gui, J., Yu, B., Cong, X., Gong, X., Tao, W., Tao, D., Feb 2025. Deep learning-based point cloud registration: a comprehensive survey and taxonomy, [arXiv:2404.13830](https://arxiv.org/abs/2404.13830).
- Zhao, H., Li, X., Ge, H., Lei, N., Zhang, M., Wang, X., Gu, X., 2018. Conformal mesh parameterization using discrete calabi flow. *Comput. Aided Geom. Des.* 63, 96–108.
- Zohar, L., 2021. Direct seamless parametrization. *ACM Trans. Graph.* 40 (1), 1–14.