Winding Numbers on Discrete Surfaces

NICOLE FENG, MARK GILLESPIE, and KEENAN CRANE, Carnegie Mellon University, USA

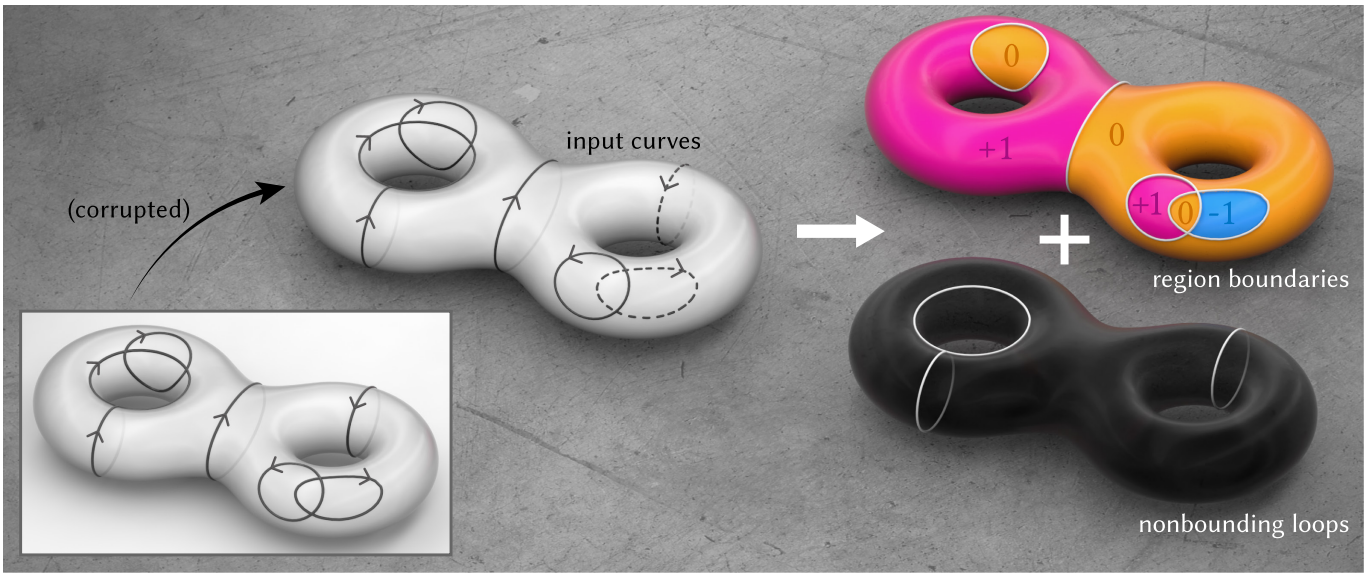


Fig. 1. Any collection of closed loops can be expressed as the boundary of some collection of regions, plus additional nonbounding loops. We describe an algorithm for recovering this decomposition, even from input curves that exhibit topological errors and ambiguities such as gaps and self-intersections.

In the plane, the *winding number* is the number of times a curve wraps around a given point. Winding numbers are a basic component of geometric algorithms such as point-in-polygon tests, and their generalization to data with noise or topological errors has proven valuable for geometry processing tasks ranging from surface reconstruction to mesh booleans. However, standard definitions do not immediately apply on surfaces, where not all curves bound regions. We develop a meaningful generalization, starting with the well-known relationship between winding numbers and harmonic functions. By processing the derivatives of such functions, we can robustly filter out components of the input that do not bound any region. Ultimately, our algorithm yields (i) a closed, completed version of the input curves, (ii) integer labels for regions that are meaningfully bounded by these curves, and (iii) the complementary curves that do not bound any region. The main computational cost is solving a standard Poisson equation, or for surfaces with nontrivial topology, a sparse linear program. We also introduce special basis functions to represent singularities that naturally occur at endpoints of open curves.

CCS Concepts: • **Computing methodologies** → **Shape analysis**; • **Mathematics of computing** → **Partial differential equations**.

Authors' address: Nicole Feng, nfeng@cs.cmu.edu; Mark Gillespie, mgillesp@cs.cmu.edu; Keenan Crane, kmcrane@cs.cmu.edu, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, USA, 15213.

0730-0301/2023/8-ART1 $15.00
https://doi.org/10.1145/3592401

Additional Key Words and Phrases: discrete differential geometry, geometry processing, winding number, solid angle

ACM Reference Format:
Nicole Feng, Mark Gillespie, and Keenan Crane. 2023. Winding Numbers on Discrete Surfaces. *ACM Trans. Graph.* 42, 4, Article 1 (August 2023), 14 pages. https://doi.org/10.1145/3592401

1 INTRODUCTION

Winding numbers are a basic concept from differential geometry [Do Carmo 2016, Section 5.7], and arise naturally in settings ranging from surface reconstruction [Kazhdan et al. 2006] to mesh booleans [Zhou et al. 2016] to tetrahedral meshing [Hu et al. 2018]. This concept does not, however, have a standard extension to curves on surfaces, posing challenges for surface processing algorithms that seek to distinguish inside from outside. The basic issue is that, on surfaces, a closed loop may not be the boundary of a well-defined region (Figure 2). Standard tools from topology, such as basic simplicial homology, are unfortunately insufficient when curves have topological errors, as often arise in applications. Conversely, existing algorithms for robust inside-outside tests do not handle curves on surfaces [Jacobson et al. 2013]. We address both issues.

Basic Problem. Any collection Γ of unbroken (*i.e.*, closed), oriented loops can be decomposed into subsets that do and do not bound regions of a surface M (Figure 1). Informally, we wish to label all regions meaningfully bounded by Γ, and explicitly identify curves that do not bound regions. A more precise problem statement is given in Section 3. Unfortunately, computing such a decomposition

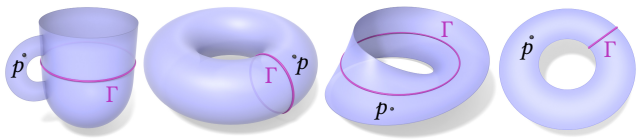


Fig. 2. Is the point p "inside" or "outside" the curve Γ? On surfaces, this question does not always have a meaningful answer.

is not as simple as classifying each loop as bounding or nonbounding: for one thing, a region boundary may be comprised of several independent nonbounding loops (*e.g.* Figure 6). Moreover, the input Γ may be given as an arbitrary set of oriented edges, which are not *a priori* decomposed into distinct loops (Figure 7). When there are errors in the input (*e.g.*, gaps or spurious intersections) one must also somehow complete the input to form closed curves.

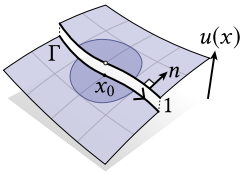
Basic Approach. The classic winding number is a special case of the signed *solid angle* function, which is itself a particular *harmonic function*, *i.e.*,

$$\boxed{\text{winding number} \subset \text{solid angle} \subset \text{harmonic functions}}$$

Since classic winding numbers are not well-defined for broken curves, Jacobson et al. [2013] consider the solid angle function, which they call the *generalized winding number (GWN)*. In turn, solid angle is not well-defined for curves on surfaces—leading us to consider more general harmonic functions u which jump across Γ. For instance, if Γ is simple (*i.e.*, has no self-intersections), then u satisfies a Laplace equation with jump boundary conditions, namely

$$\begin{aligned} \Delta u &= 0, && \text{on } M \setminus \Gamma, \\ u^+ - u^- &= 1, && \text{on } \Gamma, \\ \partial u^+/\partial n &= \partial u^-/\partial n, && \text{on } \Gamma. \end{aligned} \tag{1}$$

Here $u^\pm(x) := \lim_{\varepsilon \to 0} u(x \pm \varepsilon n(x))$ is the value of u on either side of Γ in the normal direction n (see [Krutitskii 2001] for a careful treatment in the case $M = \mathbb{R}^2$). On surfaces, however, u can be "polluted" by the influence of nonbounding curves (Figure 3). Our approach is to filter out this influence via *de Rham cohomology*: rather than work with u itself, we process its gradient vector field (or more properly, the *differential 1-form* du), and recover curves from this processed field. For broken input curves, where there are many possible choices, we also introduce a regularizer that leads to a unique solution. Overall, just as solid angle has provided robust tools for geometry processing in Euclidean space, "cohomological geometry processing" provides a robust approach to processing submanifolds in domains of more general topology.



1.1 Related Work

Connections between winding numbers, solid angles, and harmonic functions have long appeared in mathematics, physics, and scientific computing [Binysh and Alexander 2018]. Both Euler [1781] and Lagrange [1798] give formulas for the solid angle of a triangle; Gauss [1838, Sections 37-38] notes the relationship of solid angle to magnetic potential; Maxwell [1881, Articles 409-11, 417-21] further

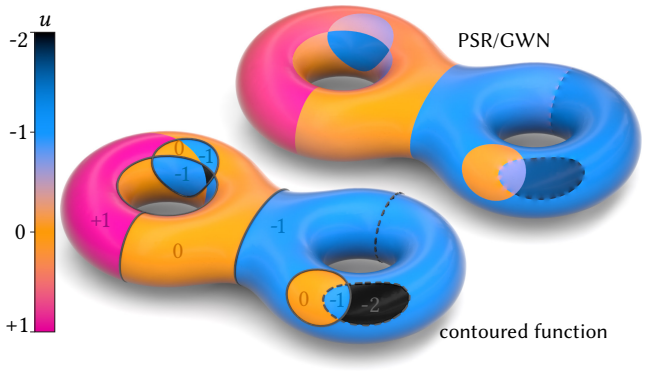
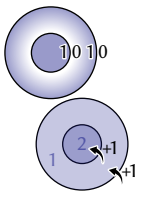


Fig. 3. Both Poisson surface reconstruction (PSR) and generalized winding numbers (GWN) compute a harmonic function subject to jump conditions across the input curves. On surfaces, contouring this function can yield regions that do not follow the input curves, and may give bogus winding numbers that jump across nonbounding curves (compare with Figure 1).

makes connections to jump conditions *à la* Equation 1. Methods for approximating solid angles also play an integral role in *boundary element methods (BEM)* for the Laplace equation [Ning et al. 2010]. To date, however, there has been little focus on the surface case.

1.1.1 Computer Graphics. In computer graphics, winding numbers were first applied to *point-in-polygon queries* [Shimrat 1962; Haines 1994]. Solid angle also plays a key role in rendering, *e.g.*, for finite element radiosity [Goral et al. 1984] or importance sampling for direct illumination [Veach and Guibas 1995, Section 2.1].

1.1.2 Geometry Processing. In geometry processing, the utility of the solid angle function has been rediscovered twice, via both *Poisson surface reconstruction (PSR)* [Kazhdan et al. 2006] and the *generalized winding number (GWN)* [Jacobson et al. 2013]. These methods are in turn key components of a wide variety of applications [Hu et al. 2018; Zhou et al. 2016; Chi and Song 2021; Müller et al. 2021; Dvořák et al. 2022; Collet et al. 2015; Chang et al. 2017]. As briefly noted by Barill et al. [2018, Section 2.1], PSR and GWN are ultimately different numerical discretizations of the same continuous problem: PSR effectively adopts the PDE perspective from Section 1, focusing on reconstruction from oriented points. GWN instead uses a boundary integral formulation, adopting hierarchical methods from the BEM literature to obtain a fast approximation [Barill et al. 2018]. Either way, contouring the function u does not in general yield useful results for nonbounding curves on surfaces (Figure 3). In fact, GWN is sometimes inadequate even for problems involving regions in the plane (Figure 20), which can be viewed as surfaces with boundary.



Harmonic functions with discontinuities do arise in surface processing, *e.g.*, for *diffusion curves* [Sun et al. 2012] or *SeamCuts* [Lucquin et al. 2017]. However, these methods assign fixed Dirichlet boundary conditions on both sides of the curve (inset, *top*), whereas the harmonic function needed to compute winding numbers must instead satisfy the jump conditions from Equation 1 (inset, *bottom*). Harmonic functions continuous up to jumps also arise naturally in surface parameterization,

e.g., as conjugate harmonic functions in conformal mapping [Gu and Yau 2003; Sawhney and Crane 2017]; our treatment of such functions is similar to Tong et al. [2006].

1.1.3 Turning Numbers on Surfaces. The winding number, which assigns a value to each point, is distinct from the *turning number*, which assigns a single number to a whole curve, counting the rotations experienced by its tangent. Erickson [2017] gives an amusing account of the historical confusion between these two terms, noting that much past work on "winding numbers" on surfaces in fact concerns the turning number [Reinhart 1960, 1963; Chillingworth 1972; Humphries and Johnson 1989; McIntyre and Cairns 1993].

1.1.4 Winding Numbers on Surfaces. McIntyre and Cairns [1993, Lemma 2] also describe a function that behaves like the winding number for bounding curves, but for nonbounding curves must introduce arbitrary discontinuities to keep this function piecewise constant. Chernov and Rudyak [2009] define a so-called *affine winding number*, which is useful only for curves within a common homotopy class. Concurrent with our work, Riso et al. [2022] give a method for computing winding numbers of perfectly closed curves that are already partitioned into distinct loops. Nonbounding components are addressed via user-guided edits. Though there are naïve ways to automate such edits (*e.g.*, remove or duplicate *all* nonbounding loops, or add extraneous homology generators), such strategies deviate significantly from the input. To find the *minimal* valid modification (as we do), Riso *et al.* would have to solve an integer linear program—akin to our LP in Section 3.4 (or the reduced version proposed in Section 6). In contrast, our method is already fully automatic, and does not require that the input already be split into distinct loops. Moreover, we handle broken curves, nonmanifold and nonorientable surfaces, and curves terminating on the boundary, which Riso *et al.* do not. On the other hand, our method takes seconds to minutes, whereas theirs runs at real-time rates.

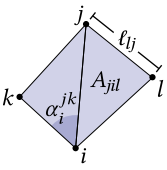
1.1.5 Cohomological Geometry Processing. Our approach is rooted in the theory of *de Rham cohomology*: we reason about curves via the dual perspective of *differential forms*. In a similar spirit, Born et al. [2021] use de Rham cohomology to reason about the topology of noisy maps between surfaces. Our curve completion step can likewise be viewed as a variant of the *optimal homologous chain problem (OHCP)* of Dey et al. [2010], reformulated via harmonic 1-forms. Both the OHCP and our problem can also be viewed as simplicial versions of the nonlinear Hodge theory discussed by Wang and Chern [2021, Appendix A]. A key difference is that this past work applies applies cohomology constraints that are known *a priori*, whereas we use cohomology as a tool to *infer* high-level topological information from noisy input data.

1.2 Outline

We first establish definitions and notation (Section 2) used in our algorithm (Section 3). Section 4 explains how this algorithm naturally arises from a duality between curves and 1-forms, with jump harmonic functions serving as a "bridge" between the two. Section 5 evaluates our method in a geometry processing context, and Section 6 discusses limitations and opportunities for improvement.

2 PRELIMINARIES

2.1 Meshes



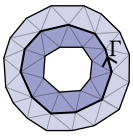
We represent the domain as a triangle mesh $M = (V, E, F)$, with no restrictions on connectivity. We denote vertices by indices $i \in V$, edges by pairs $ij \in E$, and faces by triples $ijk \in F$. For brevity, we often assume that any interior, manifold, oriented edge ij is contained in two triangles labeled ijk, jil, where k and l sit to the left and right of ij, *resp.* We also denote triangle corners by indices $\overset{jk}{i} \in C$. We use ∂M to denote the boundary of M, and E_{int} to denote the set of interior edges, *i.e.*, edges not contained in ∂M. A quantity f at vertex i is denoted by f_i. Similarly, a value at edge ij is denoted by f_{ij}, a value at face ijk by f_{ijk}, and a value at corner i of face ijk by f_i^{jk}. We use ℓ_{ij} for edge lengths, A_{ijk} for triangle areas, and α_i^{jk} for corner angles.

2.2 Curves

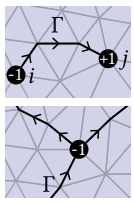
The input to our algorithm is a collection of oriented curves on a mesh M, which can in general be a union of open or closed curves, and may intersect arbitrarily (*e.g.*, two curves can run along the same edge). We often use the terms *broken* and *unbroken* for open and closed curves, *resp.*, as they are more evocative of the application context (*e.g.*, curves that have been corrupted).

We encode any collection of oriented curves as a *1-chain* Γ, *i.e.*, a signed integer $\Gamma_{ij} \in \mathbb{Z}$ for each oriented edge, counting the number of oriented traversals of Γ along ij (hence, $\Gamma_{ij} = -\Gamma_{ji}$). For example, if a curve passes over ij three times, going from i to j once, and j to i twice, then $\Gamma_{ij} = -1$. By abuse of notation, we often refer to Γ as a set, *e.g.*, writing $ij \in \Gamma$ when $\Gamma_{ij} \neq 0$. On nonmanifold meshes, we omit edges ij that pass through nonmanifold vertices by setting $\Gamma_{ij} = 0$, since here it is not clear how different "sides" of the curve should be defined. Instead, we leverage the robustness of our method to deal with these additional gaps.

2.2.1 Regions and Boundaries. A *2-chain* is a signed integer $R_{ijk} \in \mathbb{Z}$ per oriented triangle, encoding a region of the surface (possibly multiply-covered). Its *boundary* is the 1-chain given by $(\partial R)_{ij} := R_{ijk} - R_{jil}$. We call a curve Γ *bounding* if it is the boundary of a 2-chain, and *nonbounding* otherwise. Importantly, on domains with boundary, we do not require bounding curves to include edges $ij \in \partial M$, since we do not want to filter out curves that bound a region in conjunction with the domain boundary (*e.g.*, the curve Γ in the inset). Formally, such curves are congruent to zero in the relative homology group $H_1(M, \partial M)$.



2.2.2 Endpoints. A *0-chain* is likewise a signed integer per vertex, encoding a set of points (possibly with multiplicity). For any 1-chain Γ, its *endpoints* are given by the boundary 0-chain $(\partial\Gamma)_i := -\sum_{ij} \Gamma_{ij}$. For instance, if Γ is a path from i to j, then $\partial\Gamma$ is -1 at i, $+1$ at j, and zero everywhere else. Note that one can also have endpoints where multiple curves meet (see inset). We use *interior endpoints* to refer to endpoints not contained in ∂M. We use V^* to denote the set of vertices minus interior endpoints, and E^* for the set of edges with both endpoints in V^*.

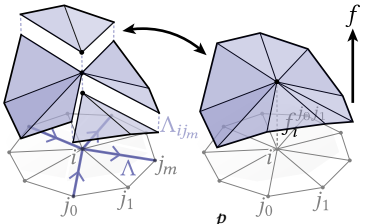


2.3 Jump Harmonic Functions

In the plane, the solid angle of a curve is a harmonic function that jumps in value across the curve. We likewise consider harmonic functions with discontinuous jumps. Such *jump harmonic functions* are encoded by corner values $f_i^{jk} \in \mathbb{R}$, which are linearly interpolated within each triangle, and must be discretely harmonic up to local piecewise constant shifts. More explicitly, there must exist values σ at corners such that for each vertex $i \in V$, adding the shift σ_i^{jk} to each triangle ijk containing i yields a function $\tilde{f}$ that is *discretely harmonic* at vertex i, *i.e.*, $(L\tilde{f})_i = 0$, where L is the *cotan Laplacian* given in Equation 6. As a consequence, the jump across any edge ij will be the same at both endpoints, *i.e.*,

$$f_i^{jk} - f_i^{lj} = f_j^{ki} - f_j^{il}. \tag{2}$$

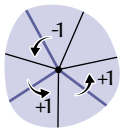
2.3.1 Reduced Coordinates. If the jumps in f are known values Λ_{ij}, then f can be expressed by adding the cumulative sum of jumps around i to a reference value $f_i^{j_0 j_1}$ at some corner:



$$f_i^{j_p j_{p+1}} = f_i^{j_0 j_1} + c_i^{j_p j_{p+1}}, \quad \text{where } c_i^{j_p j_{p+1}} := \sum_{m=1}^{p} \Lambda_{ij_m}. \tag{3}$$

Here the neighbors j_p of i are indexed in counter-clockwise order. At the boundary we assume that the reference value is stored at the most clockwise corner. Since we disallow jumps across nonmanifold edges (Section 2.2), we set $c = 0$ for all corners around such vertices.

2.3.2 Singular Points. Consider a harmonic function f that jumps in value only across the input curve (*i.e.*, $\Lambda = \Gamma$). At an interior endpoint i the jumps Γ_{ij} do not sum to zero, hence there are no corner values compatible with all jumps. The function f thus has a *singular point* at i. In practice, we do not store values of f at interior endpoints of Γ—instead, f is encoded by the $|V^*|$ reference values $f_i^{j_0 j_1}$, which we denote by a vector $f_0 \in \mathbb{R}^{|V^*|}$.



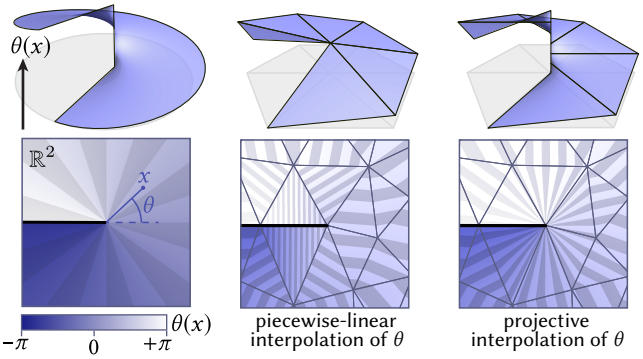


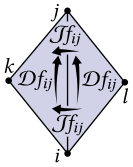
Fig. 4. Near endpoints of a curve Γ, a jump harmonic function behaves like the angular coordinate function $\theta(x)$ (*left*), which is poorly captured by piecewise-linear functions (*center*). Our custom interpolant (*right*) better captures the singular behavior.

Interpolation. To visualize f in a triangle ijk containing a singular vertex i, we perform the following *projective interpolation*, inspired by Knöppel et al. [2015, Section 4.3]:

$$f(\lambda_i, \lambda_j, \lambda_k) := \frac{\lambda_j f_j^{ki} + \lambda_k f_k^{ij}}{\lambda_j + \lambda_k} \tag{4}$$

where $(\lambda_i, \lambda_j, \lambda_k)$ are barycentric coordinates. The interpolated function is constant along rays emanating from i, correctly capturing singular behavior near endpoints (Figure 4).

2.4 Derivative Operators



Since a jump harmonic function f can have discontinuities, we must be careful when defining (discrete) derivatives. In particular, the *jump derivative* $\mathcal{J}f$ (Section 2.4.1) describes discontinuous jumps across edges, whereas the *Darboux derivative* $\mathcal{D}f$ (Section 2.4.2) captures the complementary continuous change along edges. The *jump Laplacian* L_J (Section 2.4.3) measures the failure of a function to be a jump harmonic function.

2.4.1 Jump Derivative. We use $\mathcal{J}$ to denote the *discrete jump derivative*, which measures the size of the jump across edge ij:

$$(\mathcal{J}f)_{ij} := f_i^{jk} - f_i^{lj},$$

which by Equation 2 is the same as the jump $f_j^{ki} - f_j^{il}$. We let $(\mathcal{J}f)_{ij} := 0$ at boundary edges, and have $(\mathcal{J}f)_{ij} = 0$ (no jumps) at nonmanifold edges (Section 2.2). Formally, $\mathcal{J}f$ is a 1-chain.

2.4.2 Darboux Derivative. We also define the *discrete Darboux derivative* $\mathcal{D}$, which for each edge $ij \in E^*$ is given by

$$(\mathcal{D}f)_{ij} := f_j^{ki} - f_i^{jk}. \tag{5}$$

Since f is continuous up to jumps, $\mathcal{D}f$ is the same no matter which side of the edge is used to evaluate it (Equation 2). Note that this definition also applies to nonmanifold edges, where all jumps are zero. For edges $ij \notin E^*$ we let $(\mathcal{D}f)_{ij} := 0$, since the interpolated function is constant along ij which points in the radial direction (Section 2.3.2). Formally, $\mathcal{D}f$ is a discrete 1-form (Section 2.5); if all jumps are multiples of 2π, it discretizes the usual continuous Darboux derivative (see Corman and Crane [2019, Section 1.5]).

2.4.3 Jump Laplacian. Consider a function f at corners which is not necessarily harmonic, but still has known jumps Λ. The *jump Laplacian* L_J measures the failure of f to be jump harmonic. Explicitly, let $w_{ij} := \frac{1}{2}\sum_{ijk \in F} \cot \alpha_k^{ij}$ be the usual *cotan weights* [MacNeal 1949, Section 3.2], and let $L \in \mathbb{R}^{|V^*| \times |V^*|}$ be the standard cotan matrix on V^* with nonzero entries

$$\begin{aligned} L_{ij} &= L_{ji} = -w_{ij}, && \forall ij \in E^*, \\ L_{ii} &= \textstyle\sum_{ij \in E^*} w_{ij}, && \forall i \in V^*. \end{aligned} \tag{6}$$

We also define $b \in \mathbb{R}^{|V^*|}$ to be the vector of values

$$b_i = \frac{1}{2} \sum_{ij \in E^*} w_{ij}(c_j^{ik} - c_i^{jk}), \tag{7}$$

where c is defined as in Equation 3, and for each edge ij in the sum, ijk is any triangle containing ij. Substituting Equation 3 into the usual expression for the cotan Laplacian (written as a sum over

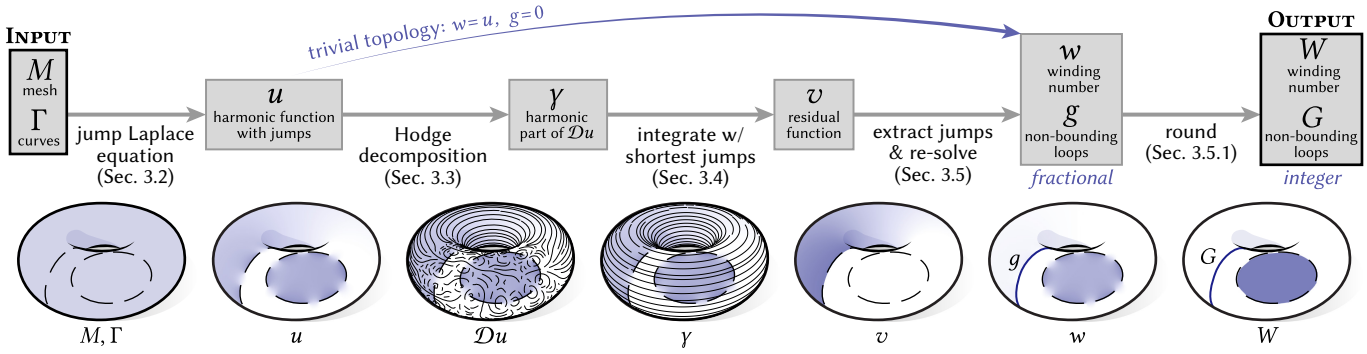


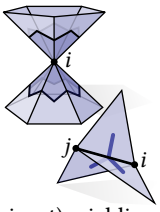
Fig. 5. *Top*: main steps of our algorithm—notice that on surfaces with trivial topology, we need only solve a single Poisson equation. *Bottom*: visualization of the functions and derivatives used at each step. Though we round the final output to obtain integer region labels W and nonbounding loops G, real-valued data w, g provides richer information about uncertainty.

triangles) shows that the Laplacian of the locally shifted function can be expressed as the ordinary Laplacian L applied to the reference values f_0, minus a constant vector b that depends only on Λ:

$$L_J f = L f_0 - b. \tag{8}$$

We omit vertices $i \in V \setminus V^*$, since a function with jumps Λ cannot be harmonic at interior endpoints—in the smooth setting, such functions locally behave like the angle function θ shown in Figure 4, *left*. Note that $L_J f = 0$ discretizes Equation 1, by absorbing the jump conditions into the definition of the operator.

2.5 Discrete Differential Forms



We use *discrete exterior calculus* [Desbrun et al. 2006] to perform the Hodge decomposition in Section 3.3; see Crane et al. [2013, Chapter 3] for a more thorough introduction. However, since we want to work with general (possibly nonmanifold) triangle meshes, we follow Sharp et al. [2019a] and define the *discrete Hodge star* operators by taking volume ratios involving all incident elements (see inset), yielding diagonal matrices with entries $(*_0)_i := \frac{1}{3}\sum_{ijk \in F} A_{ijk}$ for all $i \in V$, $(*_1)_{ij} := \mathrm{w}_{ij}$ for all $ij \in E$, where w_{ij} are the cotan weights from Section 2.4.3, and $(*_2)_{ijk} := 1/A_{ijk}$ for all $ijk \in F$. Otherwise, we use the standard *discrete exterior derivative* matrices d_k; the *discrete codifferential* is then $\delta_k := *_{k-1}^{-1} d_{k-1}^T *_k$.

2.5.1 Helmholtz-Hodge Decomposition. As discussed by Desbrun et al. [2006, Section 7], any 1-form ω can be expressed via a *Helmholtz-Hodge decomposition*

$$\omega = d\alpha + \delta\beta + \gamma$$

where α and β are 0- and 2-forms, and γ is a harmonic 1-form. As detailed in Crane et al. [2013, Chapter 8], this decomposition can be computed by solving a pair of Poisson equations

$$\Delta_0 \alpha = \delta_1 \omega \qquad \text{and} \qquad \Delta_2 \beta = d_1 \omega, \tag{9}$$

where $\Delta_0 := *_0^{-1} d_0^T *_1 d_0$ and $\Delta_2 := d_1 *_1^{-1} d_1^T *_2$ are the discrete 0- and 2-form Laplacians, *resp.*, with their usual zero-Neumann boundary conditions.

3 ALGORITHM

The steps of our algorithm are shown in Figure 5; see Section A of the supplement for pseudocode. Given a collection Γ of input curves on a mesh M, the output is an integer labeling W of regions bounded by Γ, and closed curves G that do not bound any region. We also compute a real-valued function w analogous to GWN, and a real-valued 1-chain g corresponding to G. Just as w provides confidence about inside/outside classification [Jacobson et al. 2013], g provides confidence about nonbounding loops in the input.

3.1 Overview

We seek a harmonic function w that (i) jumps in value across the input curve Γ, and (ii) approaches a piecewise-integer function as the size of gaps in Γ goes to zero. The starting point is to compute a harmonic function u that jumps across Γ (Section 3.2). If Γ has *nonbounding* components, this function will not look like a region labeling. We therefore compute a *residual function* v corresponding to this nonbounding part, and subtract it from u to get w.

Just by inspecting u, it is hard to determine the residual v, *i.e.*, it is hard to say what part comes from nonbounding loops. However, if u were a piecewise constant region labeling, then its Darboux derivative $\omega := \mathcal{D}u$ would be zero everywhere. Hence, a nonzero ω must be the derivative of v, and can thus be integrated to obtain v.

3.1.1 Broken Curves. More generally, we must modify this basic algorithm to make it robust to defects in Γ. In particular, we replace three steps with "best fit" versions, which effectively provide a guess for what the original, uncorrupted curve might have been:

- When Γ has no defects, ω is a *harmonic 1-form*, since locally it is the derivative of a harmonic function. Hence, for a broken curve we use Helmholtz-Hodge decomposition to find the harmonic 1-form γ closest to ω (Section 3.3).
- Likewise, when Γ is unbroken, the residual v jumps only across Γ. Hence, when Γ has gaps we seek to minimize the length of the jump discontinuity between the gaps (Section 3.4).
- Finally, subtracting v directly may introduce new discontinuities in w wherever gaps were filled. Hence, we instead solve for a harmonic function w that jumps by $\Gamma - \mathcal{J}v$ (Section 3.5).

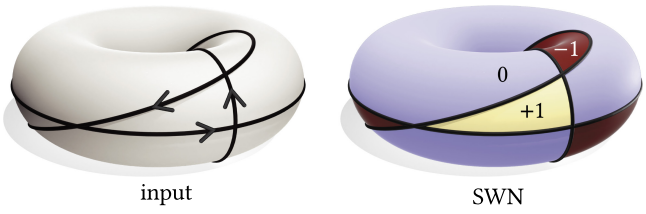


Fig. 6. Even if individual loops do not bound regions, they can conspire to define a meaningful partition—here for instance we produce a correct labeling, reproducing an example from Riso et al. [2022, Figure 4].

The functions v and w are further processed to obtain integer labels, a completion of the input curve, and a classification of bounding vs. nonbounding components (Section 3.6). When M is topologically trivial (*i.e.* simply-connected), there are by definition no nonbounding loops, so we need not filter out their contributions. Here our algorithm boils down to a single linear solve (Equation 1), followed by rounding and curve completion (Sections 3.5 and 3.6).

3.2 Jump Laplace Equation

We first solve for a harmonic function u that jumps in value by Γ_{ij} across each edge ij. In particular, we solve the Laplace equation $L_J u = 0$ with jumps $\Lambda = \Gamma$, or equivalently

$$Lu_0 = b, \tag{10}$$

where b is the constant vector defined in Equation 7, and $u_0 \in \mathbb{R}^{|V^*|}$ provides reference values at each vertex. Unlike a discrete Poisson equation, b need not be multiplied by area weights, since it simply encodes the jump values. The corner values $u_i^{jk} = (u_0)_i + c_i^{jk}$, with c_i^{jk} defined as in Equation 3, then describe a jump harmonic function. Note that since we do not fix any boundary values (and instead impose jump boundary conditions), the solution u to Equation 10 is determined only up to a constant shift. Since our next step is to differentiate u, the constant does not matter here (though we will carefully shift before rounding—see Section 3.5).

3.3 Derivative Processing

We next adjust the Darboux derivative $\omega = \mathcal{D}u$ (Section 2.4.2) to account for the fact that Γ may be broken. In general, the discrete exterior derivative df of a harmonic 0-form f is a harmonic 1-form. If Γ is an unbroken curve, then ω is harmonic, since u is a jump harmonic function, for which $\mathcal{D}u$ behaves like the discrete exterior derivative. However, if Γ is broken, ω will not be harmonic, and we use Hodge decomposition (Section 2.5.1) to extract its harmonic part γ. In this case, only $\delta\beta$ will be nonzero, due to singular behavior near interior endpoints (Appendix A). Hence, we need only solve the second Poisson equation $\Delta_2\beta = d\omega$, then evaluate $\gamma \leftarrow \omega - \delta\beta$.

3.4 Residual Function

Our goal is now to find a residual function v whose Darboux derivative looks like γ, and hence describes the nonbounding part of our input curves. If we imagine this nonbounding part is a 1-chain G, then v must jump across G, and should not jump across the complementary bounding component $\Gamma \setminus G$. However, the choice of G is in general ambiguous (Figure 7). Hence, we look for the *minimal* jumps needed for v to integrate γ. Also, the jumps in v should never be bigger than Γ, which would effectively add additional copies of the input curves. Suppose for example that Γ consists of two parallel nonbounding loops Γ_1, Γ_2 with same orientation (see inset): if we do not limit the size of the jump, then we may end up jumping by +2 across Γ_1 rather than +1 across both Γ_1 and Γ_2. Finally, since Γ may be broken, we allow v to jump across edges of the mesh not originally included in Γ. By making jumps across such edges expensive, and using a sparsity-inducing ℓ_1-norm, we promote short completions. Overall, we get an optimization problem

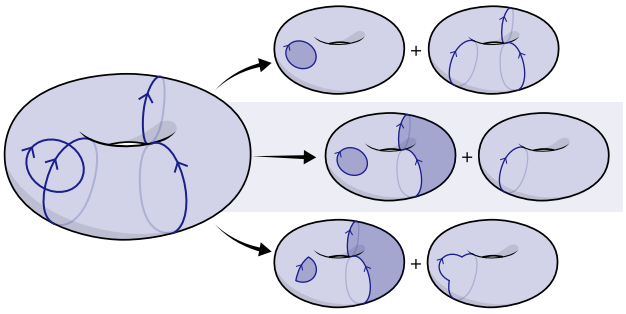
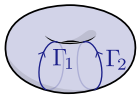
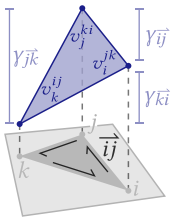


Fig. 7. A collection of loops can be decomposed into bounding and nonbounding components in many different ways. We look for the decomposition whose residual is shortest (*middle*).



$$\begin{aligned} \min_{v \in \mathbb{R}^{|C|}} \quad & \sum_{ij \in \Gamma \cap E_{\text{int}}} \ell_{ij} |(\mathcal{J}v)_{ij}| \;+\; \tfrac{1}{\varepsilon} \sum_{ij \in E_{\text{int}} \setminus \Gamma} \ell_{ij} |(\mathcal{J}v)_{ij}| \\ \text{s.t.} \quad & v_j^{ki} - v_i^{jk} = \gamma_{\vec{ij}}, \quad \forall \vec{ij} \in S, \\ & 0 \le (\mathcal{J}v)_{ij}/\Gamma_{ij} \le 1, \quad \forall ij \in \Gamma. \end{aligned}$$

Here we use $\vec{ij} \in S$ to denote an oriented side within a triangle; hence, the equality constraint ensures that v exactly integrates γ within each triangle. The parameter ε in the objective controls the relative cost of jumps across edges in Γ versus jumps elsewhere (we use $\varepsilon = 0.01$ in all examples). The inequality constraint ensures that v jumps by no more than Γ—since we take a quotient, the sign of Γ does not impact the direction of the inequality.



We reduce the problem size from $3|F|$ to $|F|$ via a change of variables. First, we integrate γ in each triangle ijk to get local values $\mathring{v}_i^{jk} := 0$, $\mathring{v}_j^{ki} := \gamma_{\vec{ij}}$, and $\mathring{v}_k^{ij} := \gamma_{\vec{ij}} + \gamma_{\vec{jk}}$. Next, along each edge ij we let $s_{ij} := \mathring{v}_i^{jk} - \mathring{v}_i^{lj}$ be the jump between reference values. Rather than optimize individual corner values, we can now just optimize per-triangle shifts σ_{ijk}, measuring the jump across each edge as

$(\sigma_{ijk} - \sigma_{jil}) - s_{ij}$. Our final problem is then

$$\begin{aligned} \min_{\sigma \in \mathbb{R}^{|F|}} \quad & \sum_{ij \in \Gamma \cap E_{\text{int}}} \ell_{ij} |(\sigma_{ijk} - \sigma_{jil}) - s_{ij}| \; + \\ & \tfrac{1}{\varepsilon} \sum_{ij \in E_{\text{int}} \setminus \Gamma} \ell_{ij} |(\sigma_{ijk} - \sigma_{jil}) - s_{ij}| \\ \text{s.t.} \quad & 0 \leq ((\sigma_{ijk} - \sigma_{jil}) - s_{ij})/\Gamma_{ij} \leq 1, \quad \forall ij \in \Gamma. \end{aligned} \tag{11}$$

The final corner values are then recovered via $v_i^{jk} = \mathring{v}_i^{jk} + \sigma_{ijk}$. As usual, Equation 11 can be transformed into a linear program in standard form by introducing slack variables. In the nonmanifold case, we require that there be no jumps across edges ij incident on a nonmanifold vertex i, which can be imposed as additional linear constraints $\sigma_{ijk} - \sigma_{jil} = s_{ij}$. This step is the bottleneck in our method; Section 6 discusses a possible way to significantly reduce its size.

3.5 Winding Number Function

Finally, we filter out the influence of nonbounding loops from u, using the residual function v (Figure 8). If Γ is unbroken, we can simply subtract v from u to get w. In general, however, a simple subtraction will introduce discontinuities, since v may jump across edges which are not part of Γ. Instead, to get w, we solve Equation 10, but for edges $ij \in \Gamma$ now use jumps $\Lambda_{ij} = \Gamma_{ij} - (\mathcal{J}v)_{ij}$ to define c (hence b).

3.5.1 Integer Winding Numbers. To get an integer function W, we round the real-valued function w. However, our Laplace equation determines w only up to a constant shift, which affects rounding. To determine a reasonable shift, we compute an average shift over edges in Γ, where the values of w are most reliable. More explicitly, for each edge ij positively oriented along Γ (*i.e.*, $\Gamma_{ij} > 0$), let $w_{ij}^+ := (w_j^{ki} + w_i^{jk})/2$, *i.e.*, the average value on one side of the curve. Our global shift τ is then the mean of the per-edge shifts $\mathsf{round}(w_{ij}^+) - w_{ij}^+$, where round gives the closest integer. Our final per-face region labels are then $W_{ijk} := \mathsf{round}((w_i^{jk} + w_j^{ki} + w_k^{ij})/3 + \tau)$. See Section 6 for discussion of contouring beyond simple rounding.

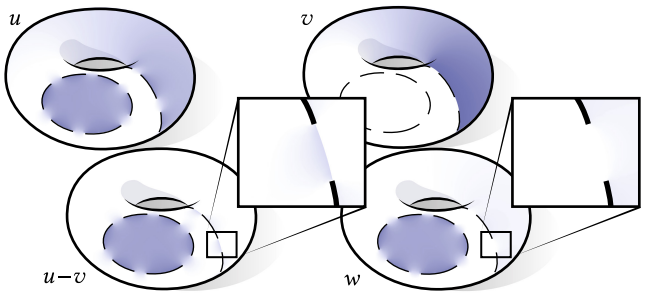


Fig. 8. The residual function v captures the nonbounding part of the input curves Γ (*top right*). Simply subtracting v from u introduces additional jump discontinuities (*bottom left*), so we obtain our winding number function w (*bottom right*) by solving a second jump Laplace equation (Section 3.5).

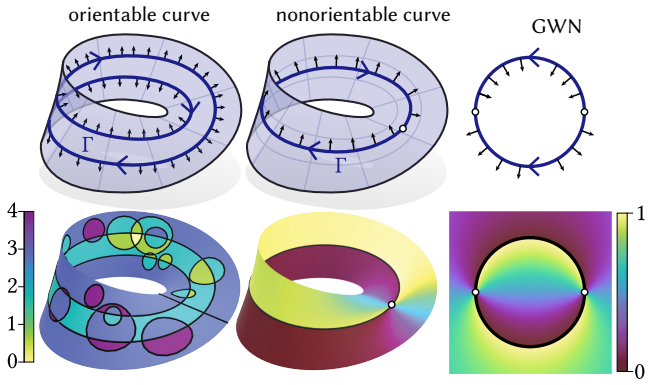


Fig. 9. For a nonorientable surface like the Möbius strip, Γ is an *orientable curve* if it can be assigned a consistent normal direction *(top left)*, and is otherwise a *nonorientable curve (top center)*. Our algorithm works as expected for collections of orientable curves *(bottom left)*. As with GWN *(right)* we do not obtain meaningful region labeling for curves with inconsistent orientation.

3.6 Curve Completion

Bounding Curves. The final output bounding curves are given by the boundary ∂W of the integer winding number function, *i.e.*, the 1-chain with values $(\mathcal{J}W)_{ij}$ at each edge $ij \in E$, where each corner is assigned the value of its corresponding face (*e.g.*, $W_i^{jk} = W_{ijk}$). These curves are always closed, since they are region boundaries.

Nonbounding Curves. To extract nonbounding curves, we likewise compute the jumps in v. The resulting 1-chain $g := \mathcal{J}v$ has no interior endpoints: around any interior vertex i, we have a telescoping sum

$$-(\partial g)_i = \sum_{ij \in E} g_{ij} = \sum_{ij \in E} v_i^{jk} - v_i^{lj} = \sum_{ijk \in F} v_i^{jk} - v_i^{jk} = 0,$$

i.e., the value at the corner of each triangle containing i appears twice, with opposite signs. To get an integer version of this chain G, we simply let $G_{ij} := \mathsf{round}(g_{ij})$ for all ij.

3.7 Nonorientable Surfaces

Our algorithm works on nonorientable domains, and will filter out nonbounding curves so long as they are *orientable curves*, *i.e.*, can be assigned a continuously-varying normal field (Figure 9). In practice, we need only make one small change to the specification of the input. Ordinarily, we assume that jumps increase in the direction obtained by rotating the tangent 90 degrees counter-clockwise. On a nonorientable surface, however, there is no consistent notion of counter-clockwise—even though curves can still meaningfully bound regions. Instead, we can represent the curve as a *dual 1-chain*, *i.e.*, a value $\Gamma_{ijk,jil}$ for each edge $ij \in E$. For edges ij in the curve, the sign of this value determines the normal direction, or equivalently, whether the jump goes from ijk to jil or vice-versa. Away from the curve, $\Gamma = 0$. Note that since we already set Γ to zero for nonmanifold edges (Section 2.2), we need not define jump directions here.

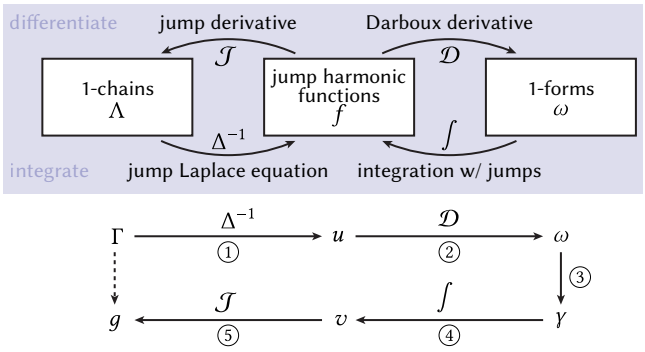


Fig. 10. *Top:* The derivative of any jump harmonic function f can be decomposed into a 1-chain $\Lambda = \mathcal{J}f$ describing discontinuous jumps, and a 1-form $\omega = \mathcal{D}f$ describing the continuous change in f. In the other direction, we can find a function f that jumps by Λ by solving a Laplace equation, or an f that explains ω by solving an integration problem. *Bottom:* from this perspective, our algorithm for extracting the nonbounding part G of the input Γ amounts to a round trip around this diagram.

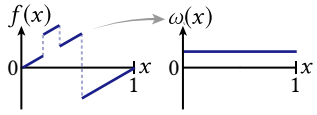
4 DISCUSSION

Jump harmonic functions provide a natural "bridge" between curves and 1-forms (or in the language of differential topology: between homology and de Rham cohomology). This perspective, detailed below, makes it easy to understand and motivate the algorithm in Section 3. In particular, our strategy for extracting the nonbounding part g of the input Γ amounts to a round trip around the diagram in Figure 10, *top*, as illustrated in Figure 10, *bottom*.

From Functions to Derivatives. A key observation is that the derivative of any jump harmonic function can be decomposed into a continuous part, and a "jump part." A useful didactic analogy is piecewise smooth periodic functions f on the interval $[0, 1]$. The distributional derivative of any such function can be expressed as

$$f'(x) = \omega(x) + \sum_i \Lambda_i \delta_{x_i}$$

where ω is a periodic piecewise smooth function, and Λ_i is the size of the jump at x_i (Figure 11). Likewise, we decompose the change in a jump harmonic function f into a 1-form describing continuous change in f, given by the Darboux derivative $\omega := \mathcal{D}f$, and a 1-chain describing discontinuous jumps, given by the jump derivative $\Lambda := \mathcal{J}f$. Just as $\omega(x)$ "forgets" about the jumps in a 1D piecewise linear function (see inset), $\mathcal{D}f$ forgets about jumps across region boundaries on a surface.



From Derivatives to Functions. We can also try to go the other direction. For instance, given a set of jumps $\Lambda_i \delta_{x_i}$ on the periodic interval $[0, 1]$, what is a piecewise smooth function that exhibits these jumps? There are many possibilities; a canonical choice is perhaps a piecewise linear function with constant slope. Likewise, the jump Laplace equation in Section 3.2 provides a canonical way to construct a function f on M that represents any 1-chain Λ. Similarly, given a periodic function ω on $[0, 1]$, we can try to find a piecewise

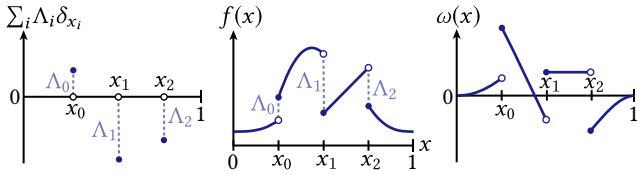
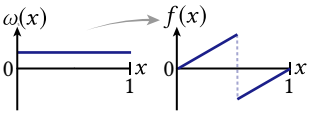


Fig. 11. The derivative of any piecewise smooth function $f(x)$ on a periodic interval (*center*) can be decomposed into a piecewise smooth function $\omega(x)$ (*right*) plus a sum of delta functions (*left*). The former captures continuous changes in f while the latter captures jumps in f.

differentiable function f such that the continuous part of f' equals ω. Ordinarily this function would be determined (up to a constant) via standard integration, but for a periodic function there may be no continuous solution—*e.g.*, if ω is strictly positive. Instead, we must decide where f should jump. Likewise, the linear program in Section 3.4 constructs a harmonic function f on M that represents any harmonic 1-form ω, while choosing a sparse set of jump curves.



Algorithm Interpretation. Our algorithm can now be understood from this perspective. Starting with step ① of Figure 10 we solve a Laplace equation for a harmonic function u that jumps across Γ. This function does not yet meaningfully label regions, since it encodes both bounding and nonbounding components of Γ. The Darboux derivative $\omega = \mathcal{D}u$ in step ② forgets the bounding components, retaining information only about the nonbounding part. Since any harmonic 1-form describes unbroken curves, step ③ finds the harmonic 1-form γ that best explains ω, via Helmholtz-Hodge decomposition. To recover explicit nonbounding curves, step ④ looks for another jump harmonic function v that integrates γ. Since this function is not unique, and could jump across many possible curves, this step uses the input chain Γ to constrain the search for the new jumps, along with an objective which encourages this curve to be as short as possible. Finally, step ⑤ extracts the jumps $g = \mathcal{J}v$ to yield the nonbounding part g of the input Γ. As suggested by the diagram, extracting the nonbounding part g of a 1-chain Γ parallels extraction of the harmonic part γ from a 1-form ω. The key difference is that in the former case we use an L^1-norm rather than an L^2-norm to find a solution concentrated on a low-dimensional subset, rather than a smooth function supported on the entire domain.

5 EVALUATION AND RESULTS

Incomplete oriented curves arise in many settings, ranging from curves projected onto noisy surfaces (Section 5.4), to strokes painted on a noisy mesh (Section 5.3), to imperfect user selections (Section 5.5). Here we apply our method, abbreviated as *surface winding numbers (SWN)* to several such tasks, and evaluate its robustness (Section 5.1) and performance on a large benchmark (Section 5.2). We visualize the shifted function $w + \tau$ (Section 3.5.1), but for brevity label it as w. Except for Figure 27, all examples (including the benchmark) involve broken curves, nonmanifold or nonorientable surfaces, and/or curves that terminate on the boundary, and hence cannot be handled by past methods such as Riso et al. [2022].

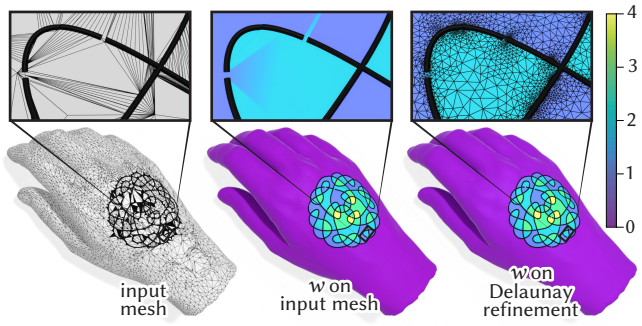


Fig. 12. Even on meshes with low element quality, SWN can produce reasonable region labels (*center*). Since our formulation is intrinsic, any remaining artifacts can be eliminated via *intrinsic Delaunay refinement* (*right*).

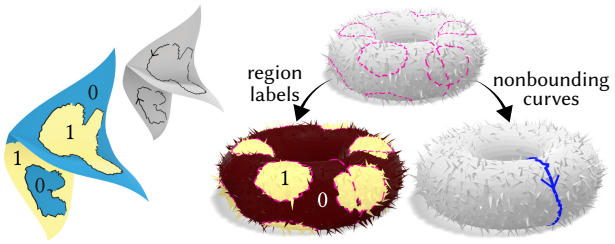


Fig. 13. Even on highly non-manifold meshes, SWN can produce an effective region labeling and completions of nonbounding curves.

5.1 Robustness and Uncertainty

Like PSR and GWN (Section 1.1), SWN is robust to defects in the input curves (*e.g.*, Figures 1, 20 and 23). Especially if gaps are reasonably small, we generally recover the same regions as for equivalent closed curves (Figure 14). In practice our method is also robust to low-quality geometry (Figure 19), meshes with low-quality elements (Figure 12), and highly nonmanifold connectivity (Figure 13), owing to the strong regularity of elliptic problems. Since it is purely intrinsic, surface self-intersections do not result in region misclassification. Moreover, an intrinsic formulation also enables us to use robust methods for *intrinsic retriangulation* if the mesh is particularly bad [Sharp et al. 2019b; Gillespie et al. 2021; Sharp et al. 2021], as illustrated in Figure 12.

As with GWN and PSR, the real-valued function w provides rough information about uncertainty—*e.g.*, the gradient norm $|\nabla w|$ will tend to be nonzero (yet finite) near gaps in a curve; see Sellán and Jacobson [2022] for an in-depth discussion in the PSR context. Likewise, the magnitude of $\mathcal{J}w$ and g, *resp.* roughly captures the confidence that a piece of the curve comes from a bounding or nonbounding curve (*resp.*) in the original, uncorrupted input (Figure 1, *bottom-left*); see for instance Figure 14, *bottom*. One might also try using jump sizes to decompose the nonbounding component into distinct loops, though we do not pursue that idea here.

5.2 Benchmark

5.2.1 Data Set. To measure the success rate of our algorithm, we constructed a synthetic dataset of models with ground truth regions

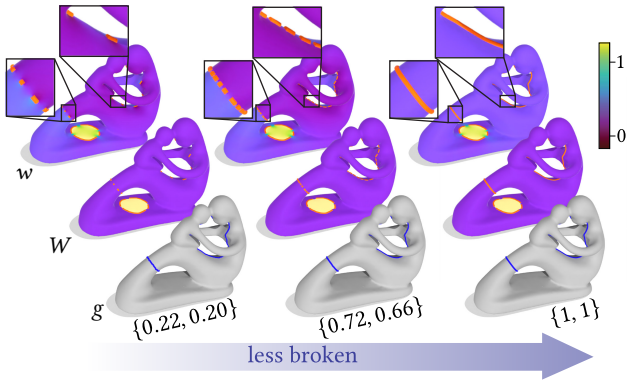


Fig. 14. As Γ becomes less broken, w approaches the expected winding number function, and the coefficients on nonbounding loops g approach 1. Throughout, the rounded winding number W yields the correct inside-outside classification, filtering out nonbounding components even for very broken inputs.

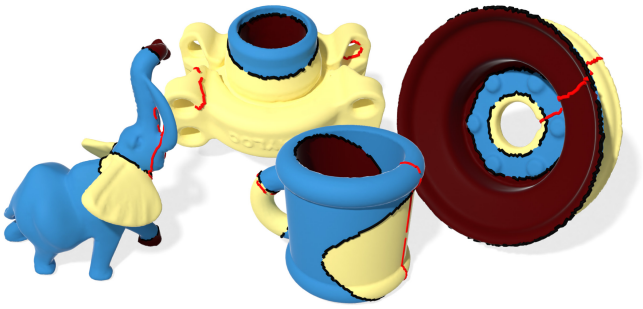


Fig. 15. Here we show four of the 934 test cases in our synthetic benchmark (Section 5.2). Each model is assigned ground truth region labels (indicated by colors), along with broken boundaries for those regions (black), and additional broken nonbounding loops (red).

and nonbounding loops (Figure 15). We started with the meshes from Myles et al. [2014], remeshed them to resolutions between 10k and 90k vertices, and generated random regions by taking sublevelsets of low-frequency Laplacian eigenfunctions. To obtain nonbounding loops, we computed a greedy homology basis [Erickson and Whittlesey 2005], picked a random subset of the loops, and straightened them slightly using *FlipOut* [Sharp and Crane 2020b] before snapping them back to mesh edges. We then deleted random segments from these curves. In total, we obtained 934 test cases of which 451 were defined on nonsimply-connected surfaces (*i.e.*, those with nontrivial topology).

5.2.2 Performance and Accuracy. We implemented SWN in C++, using geometry-central for mesh processing [Sharp et al. 2019a], CHOLMOD for linear systems [Chen et al. 2008] and Gurobi [Gurobi Optimization, LLC 2023] (via CoMISo [Bommes et al. 2012]) for linear programs. Timings were measured on an Intel i9-9980XE with 32 GB of RAM. For each test case, we quantify error as the percentage of surface area mislabeled by our method.

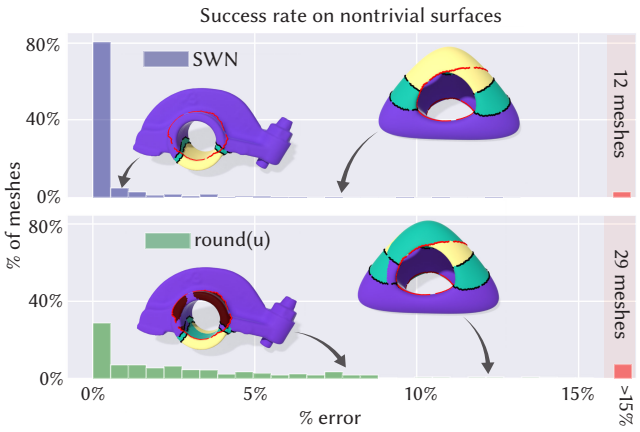


Fig. 16. Error rates for SWN (*top*) compared to naïve rounding of u *à la* GWN (*bottom*). Error is quantified as percentage of mislabeled surface area. The two highlighted examples show how naïve rounding can fail to filter out nonbounding loops (in red) which are correctly identified by SWN.

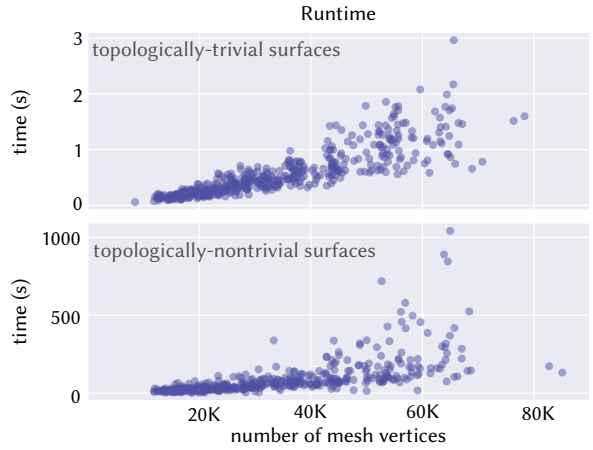


Fig. 17. *Top*: on topologically trivial surfaces, our method boils down to a quick linear solve. *Bottom*: on surfaces with nontrivial topology we must also solve a linear program, which becomes the computational bottleneck.

On simply-connected surfaces our method typically takes less than two seconds (see Figure 17, *top*), and achieves a mean/max error of only 0.14%/5%. On nonsimply-connected surfaces, there was occasionally fundamental ambiguity in the input, yielding results quite different from the ground truth (Figure 18), but in general our method remains quite accurate, achieving errors under 0.5% on 80% of models. More importantly, SWN performs much better than naïve rounding of the function u *à la* GWN, which can create phantom curves (Figure 3) which significantly degrade the accuracy of the final labels (Figure 16). The linear program takes much longer than the single linear solve, but still runs in a matter of minutes (Figure 17, *bottom*); see Section 6 for discussion of possible acceleration strategies.

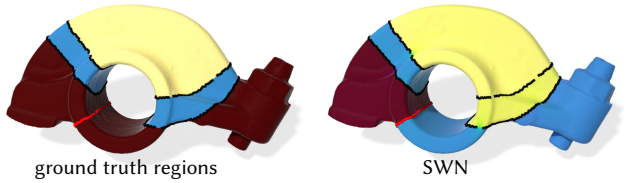


Fig. 18. Since SWN always extracts the shortest collection of nonbounding curves (Figure 7), it may not always reproduce the ground truth—but still gives a reasonable segmentation.

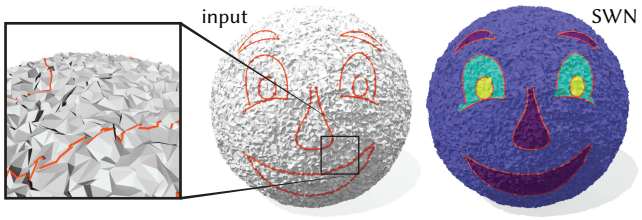


Fig. 19. We can robustly identify regions even on geometry with severe noise, intersections, and fold-over. Here, several strokes quickly painted in screen space are used to color regions on the surface.

5.3 Sketching on Surfaces

Recent methods enable manipulation of perfect closed curves on surfaces, making them appropriate for surface-based analogies of classic vector graphics [Mancinelli et al. 2021; Riso et al. 2022]. In contrast, SWN robustly handles *imperfect* broken curves, making it more appropriate for tasks like surface sketching and painting, where user input is far less precise. For instance, in Figure 19 a user sketches very reasonable yet broken curves; SWN yields a nice coloring of the sketched regions, which can be further refined by the user. Figure 20 demonstrates the utility of SWN even in 2D, where a user draws rough strokes to segment a complex shape. Here, GWN yields undesirable results—despite being a 2D method—since the influence of open strokes leaks across the domain boundary, whether or not the boundary itself is included in Γ. Likewise, GWN may not produce the expected result for 2D regions with holes—for instance, directly rounding the function u in Figure 21 would yield the same kind of phantom curves seen in Figure 3.

5.4 Stamping and Booleans

We can also perform robust boolean operations on surfaces, even for defective domains and/or curves. To get initial shapes, we can for instance "stamp" existing vector graphics onto the surface (Figures 23 and 24). Rather than worry about numerically robust intersection, we can lean on SWN to ensure we obtain well-defined regions. Boolean operations are then trivially computed via element-wise logical operations (Figure 22). Unlike *BoolSurf* [Riso et al. 2022], we can perform these operations for imperfect, broken curves—albeit at larger computational cost. Note also that unlike extrinsic mesh booleans [Zhou et al. 2016], we need not worry about self-intersections of the surface itself.

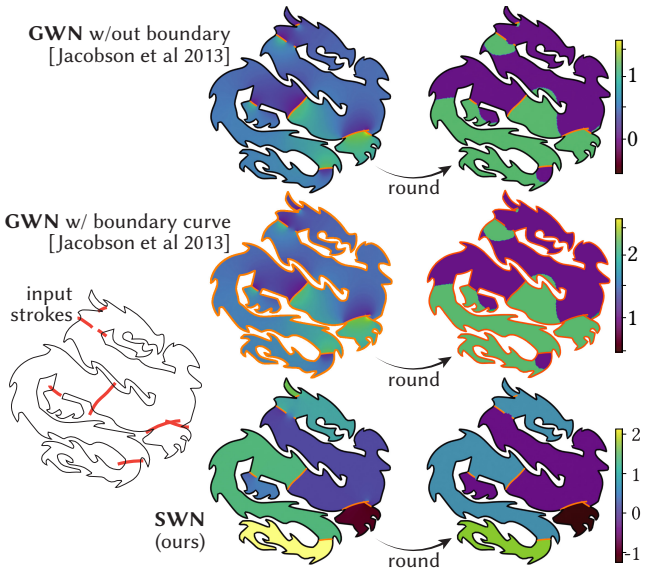


Fig. 20. *Left:* a user makes rough strokes to select regions of a 2D shape. *Top right:* GWN produces the wrong result, since the influence of strokes "leaks" across the domain boundary. *Middle right:* including the boundary curve just shifts GWN's solution by +1. *Bottom right:* SWN produces the desired result, robustly handling gaps, misclicks, and intersecting strokes.

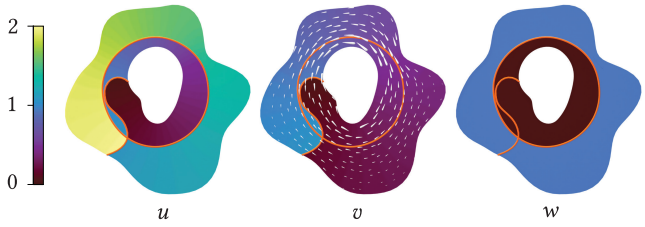


Fig. 21. Even for planar regions, one must think carefully about how curves do (or do not) bound regions. Here, SWN correctly filters out the influence of a nonbounding curve connecting two boundary components.

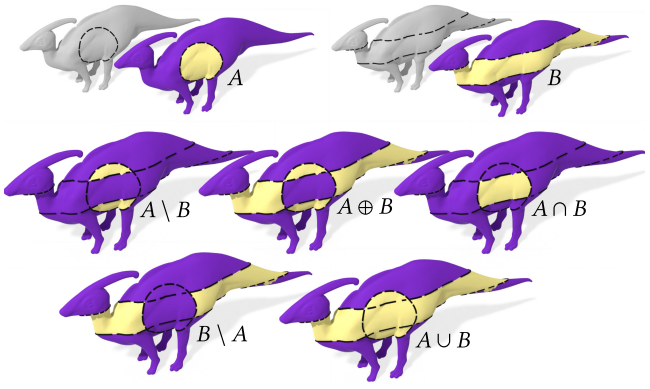


Fig. 22. Unlike previous methods, we can compute boolean operations on regions defined by imperfect, broken curves on surfaces.

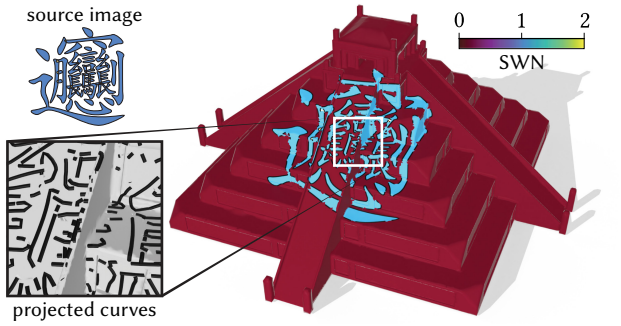


Fig. 23. A complicated shape is projected onto a ziggurat with sharp overhangs, creating broken curves; SWN nicely fills in the bounded regions.

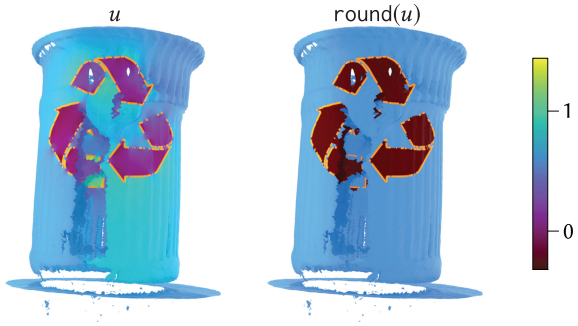


Fig. 24. A recycling logo is projected onto a noisy 3D scan of a trash can from [Choi et al. 2016], creating a highly broken curve. Despite large holes in the scan, SWN produces a reasonable region labeling.

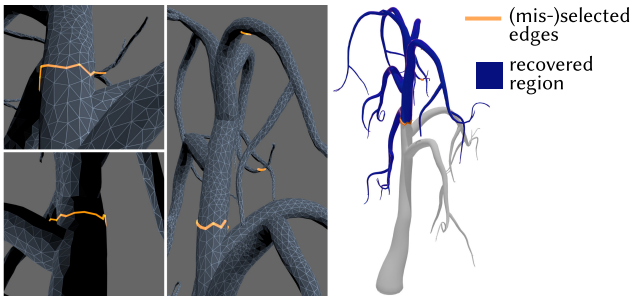


Fig. 25. A common frustration with screen-space selection is that distant edges are often selected unintentionally. SWN filters out spurious parts of the selection, and completes loops to yield the expected segmentation.

5.5 Region Selection

Selection of regions on geometrically or topologically complex 3D models is a challenging user interface design problem. SWN is a valuable component for building such tools. For instance, Figure 25 highlights a common frustration when selecting mesh edges in screen space; here SWN automatically filters out misselected edges, capturing the user intent. Similarly, Figure 26 shows how SWN can be used to repair loops that are not easily chosen via edge-based selection tools common to 3D modelers. Other tools provide facilities

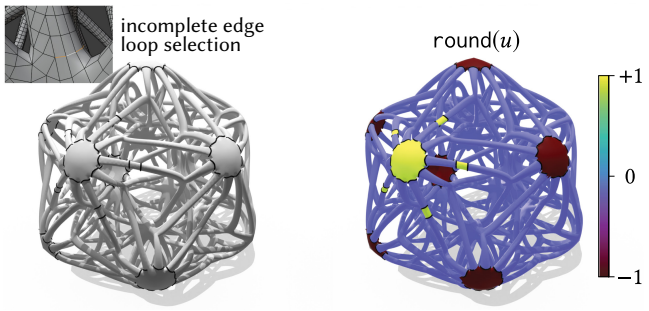


Fig. 26. Many 3D modeling tools provide edge loop selection tools, but are easily tripped up by irregular connectivity such as this mixed quad-triangle mesh *(left)*. By reasoning about functions rather than edges, we robustly infer user intent *(right)*, even on this topologically complex model.

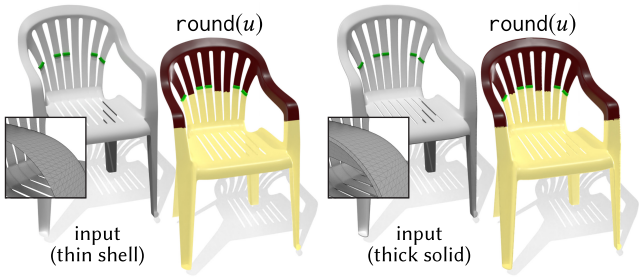


Fig. 27. In some scenarios, directly contouring the function u can yield useful results. Here for instance a user avoids the misery of selecting every small loop in a complex model. We also obtain a similar segmentation whether working with a solid *(right)* or shell-like model *(left)*.

for directly selecting regions rather than curves, *e.g.*, using a lasso or "fat" paintbrush. Here, however, one encounters the same problem: a region selected in screen space might inadvertently highlight distant, unintentional pieces of the surface. One could likewise use SWN to filter the *boundary* of such a selection. Finally, Figure 27 shows an example where one might *not* want to filter out nonbounding loops. Here, rather than process the function u, we simply apply the contouring procedure from Section 3.5.1, yielding loops that did not belong to the input, yet automatically complete the implied segmentation.

6 LIMITATIONS & FUTURE WORK

Our method shares the same basic limitations as GWN: input curves must be (mostly) consistently oriented, and we make no effort to classify curves as open vs. closed. Any open segments in the input are interpreted as subsets of (unknown) closed loops. As with GWN, a significant challenge is dealing with ambiguity in the input, and one must acknowledge that for many inputs (*e.g.*, a random subset of edges) there is no objectively "correct" solution. However, as long as Γ comes from some collection of closed loops, we recover a meaningful decomposition as the size of gaps goes to zero.

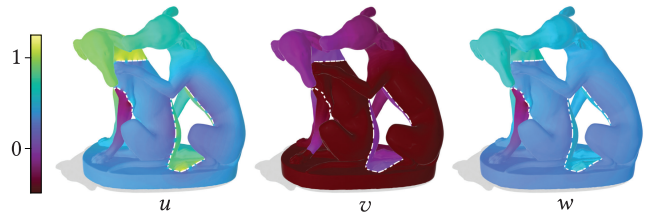


Fig. 28. Because we use a shortest completion, v jumps across the thinnest part of the front leg rather than across the input curve, but SWN still accurately corrects for loops around the other two handles (*right*).

Performance and Discretization. In contrast to GWN, which can independently evaluate the solution at any point without discretizing the domain, our method must triangulate the surface and solve a global PDE. Notably, however, GWN considers exclusively flat, Euclidean domains; the use of discretization seems inevitable to account for the geometry of general curved surfaces. An obvious performance bottleneck is the need to solve an LP for surfaces with nontrivial topology (Section 5.2). We made no attempt to optimize this step, and there are some obvious strategies to try. For instance, we could apply a change of variables $\widetilde{v}_{ij} := v_j^{ki} - v_i^{jk}$ to transform our LP into a simpler bounds-constrained ℓ_1 minimization problem. Alternatively, rather than rely on a sparsity-inducing norm for curve completion, we could use a much simpler shortest path heuristic (via Dijkstra), and compute only one shift σ per edge-connected set of faces in Equation 11—dramatically reducing the size of our LP.

Curve Completion. Some inputs are inherently ambiguous—*e.g.*, our method may not yield the expected result if large pieces of the input are missing. Consider, for example, an input loop Γ with large gaps on a surface with a strongly tapered handle (like a *Dupin cyclide*). If the circumference of this handle is less than the total gap length, SWN may prefer to place jumps around the handle (Figure 28 shows a similar example). Importantly, however, our method will still yield the correct result as gaps become smaller (Figure 14).

Contouring. The question of how to best contour fractional winding numbers is unclear, even for GWN and PSR. For instance, Jacobson et al. [2013, Section 5] suggest a *graph cut* algorithm, though this heuristic is not used in the reference implementation for GWN [Jacobson et al. 2018]. Likewise, for PSR Kazhdan et al. [2020] show that better contouring can be achieved by adding *envelope constraints* based on visibility, though this approach is not meaningful in the surface case. We likewise find that the heuristic given in Section 3.5.1 sometimes fails to detect contours that are "obvious" to the eye—see for example Figure 29. Better contouring strategies for GWN, PSR, and SWN is hence an interesting question for future work.

Nonmanifold and Nonorientable Surfaces. Our algorithm runs on nonorientable surfaces so long as Γ is consistently oriented (Section 3.7); dealing with nonorientable components of Γ requires further thought. Likewise, our method empirically works well for nonmanifold meshes, but several steps (such as Helmholtz-Hodge decomposition) are not given a rigorous justification. However, nonmanifold treatments of fundamental objects like the Laplacian have

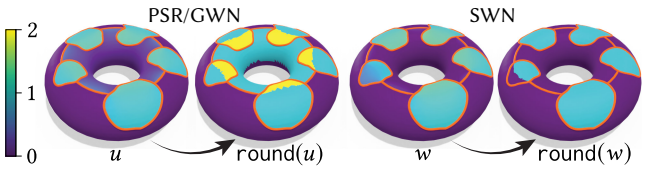


Fig. 29. Although SWN generally does better than naïvely rounding u *(left)*, there are still cases where it may not agree with human intuition about the bounded regions *(right)*. In such cases the real-valued function w still appears to capture the right information—suggesting that more work is needed on contouring strategies.

recently been explored [Sharp and Crane 2020a], and might be extended to the objects needed for our algorithm.

Uncertainty. For broken curves, non-integer values in the winding number function w give qualitative information about confidence: a small value of $|\nabla w|$ on a point of the reconstructed curve indicates uncertainty about the exact position of this curve. However, as discussed by Sellán and Jacobson [2022], such values do not provide quantitative probabilities—extending their framework to surfaces is an interesting future direction.

Higher Dimensional Winding Numbers. Region identification in nontrivial three-manifolds also arise naturally in some scenarios—*e.g.*, noisy periodic surfaces acquired from X-ray crystallography of material structures [Yuan et al. 2022]. Moreover, just as one might need to segment flat 2D regions with boundary (Figure 20), a 3D extension of SWN might prove useful for cutting up solid models with complex geometry and topology. Our "cohomology processing" approach should in principle extend to tetrahedral meshes, where the duality between 1-forms and curves becomes an analogous duality between 1-forms and surfaces.

ACKNOWLEDGMENTS

This work was funded by an NSF CAREER Award (IIS 1943123), NSF Award IIS 2212290, a Packard Fellowship and gifts from Facebook Reality Labs, and Google, Inc.

REFERENCES

Gavin Barill, Neil Dickson, Ryan Schmidt, David I.W. Levin, and Alec Jacobson. 2018. Fast Winding Numbers for Soups and Clouds. *ACM Trans. Graph.* (2018).

Jack Binysh and Gareth P Alexander. 2018. Maxwell's theory of solid angle and the construction of knotted fields. *Journal of Physics A: Mathematical and Theoretical* 51, 38 (2018), 385202.

David Bommes, Henrik Zimmer, and Leif Kobbelt. 2012. Practical mixed-integer optimization for geometry processing. In *Curves and Surfaces: 7th International Conference, Avignon, France, June 24-30, 2010, Revised Selected Papers 7*. Springer, 193–206.

J. Born, P. Schmidt, M. Campen, and L. Kobbelt. 2021. Surface Map Homology Inference. *Computer Graphics Forum* 40, 5 (2021), 193–204. https://doi.org/10.1111/cgf.14367

Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. 2017. Matterport3d: Learning from rgb-d data in indoor environments. *arXiv:1709.06158* (2017).

Yanqing Chen, Timothy A Davis, William W Hager, and Sivasankaran Rajamanickam. 2008. Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate. *ACM Transactions on Mathematical Software (TOMS)* 35, 3 (2008).

Vladimir Chernov and Yuli B. Rudyak. 2009. On generalized winding numbers. *St. Petersburg Mathematical Journal* 20, 5 (2009), 837–849.

Cheng Chi and Shuran Song. 2021. GarmentNets: Category-Level Pose Estimation for Garments via Canonical Space Shape Completion. In *The IEEE International Conference on Computer Vision (ICCV)*.

David RJ Chillingworth. 1972. Winding numbers on surfaces, I. *Math. Ann.* 196, 3 (1972), 218–249.

Sungjoon Choi, Qian-Yi Zhou, Stephen Miller, and Vladlen Koltun. 2016. A Large Dataset of Object Scans. *arXiv:1602.02481* (2016).

Alvaro Collet, Ming Chuang, Pat Sweeney, Don Gillett, Dennis Evseev, David Calabrese, Hugues Hoppe, Adam Kirk, and Steve Sullivan. 2015. High-quality streamable free-viewpoint video. *ACM Trans. Graph.* 34, 4 (2015), 1–13.

Etienne Corman and Keenan Crane. 2019. Symmetric Moving Frames. *ACM Trans. Graph.* 38, 4 (2019).

Keenan Crane, Fernando de Goes, Mathieu Desbrun, and Peter Schröder. 2013. Digital Geometry Processing with Discrete Exterior Calculus. In *ACM SIGGRAPH 2013 courses* (Anaheim, California) *(SIGGRAPH '13)*. ACM, New York, NY, USA, 126.

Mathieu Desbrun, Eva Kanso, and Yiying Tong. 2006. Discrete differential forms for computational modeling. In *ACM SIGGRAPH 2006 Courses*. 39–54.

Tamal K Dey, Anil N Hirani, and Bala Krishnamoorthy. 2010. Optimal homologous cycles, total unimodularity, and linear programming. In *Proceedings of the forty-second ACM symposium on Theory of computing*. 221–230.

Manfredo P. Do Carmo. 2016. *Differential Geometry of Curves & Surfaces* (second ed.). Dover, New York.

Jan Dvořák, Zuzana Káčereková, Petr Vaněček, Lukáš Hruda, and Libor Váša. 2022. As-rigid-as-possible volume tracking for time-varying surfaces. *Computers & Graphics* 102 (2022), 329–338. https://doi.org/10.1016/j.cag.2021.10.015

Jeff Erickson. 2017. Generic and Regular Curves. https://jeffe.cs.illinois.edu/teaching/comptop/2017/chapters/05-regular-homotopy.pdf

Jeff Erickson and Kim Whittlesey. 2005. Greedy optimal homotopy and homology generators. In *SODA*, Vol. 5. 1038–1046.

Leonhard Euler. 1781. De mensura angulorum solidorum. *Acta Academiae Scientiarum Imperialis Petropolitanae* (1781), 31–54.

Carl Friedrich Gauss. 1838. General Theory of Terrestrial Magnetism. (1838). https://hgss.copernicus.org/articles/5/11/2014/hgss-5-11-2014.pdf

Mark Gillespie, Nicholas Sharp, and Keenan Crane. 2021. Integer Coordinates for Intrinsic Geometry Processing. *ACM Trans. Graph.* 40, 6 (2021).

Cindy M Goral, Kenneth E Torrance, Donald P Greenberg, and Bennett Battaile. 1984. Modeling the interaction of light between diffuse surfaces. *ACM SIGGRAPH computer graphics* 18, 3 (1984), 213–222.

Phillip Griffiths and Joseph Harris. 2014. *Principles of algebraic geometry*. John Wiley & Sons.

Xianfeng Gu and Shing-Tung Yau. 2003. Global conformal surface parameterization. In *Eurographics Symposium on Geometry Processing*. 127–137.

Gurobi Optimization, LLC. 2023. Gurobi Optimizer Reference Manual.

Eric Haines. 1994. *Point in Polygon Strategies*. Academic Press Prof., Inc., USA, 24–46.

Allen Hatcher. 2002. *Algebraic Topology*. Cambridge University Press.

Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2018. Tetrahedral Meshing in the Wild. *ACM Trans. Graph.* 37, 4, Article 60 (July 2018), 14 pages. https://doi.org/10.1145/3197517.3201353

Stephen P Humphries and Dennis Johnson. 1989. A generalization of winding number functions on surfaces. *Proc. of the London Math. Soc.* 3, 2 (1989), 366–386.

Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. 2013. Robust Inside-Outside Segmentation Using Generalized Winding Numbers. *ACM Trans. Graph.* 32, 4, Article 33 (July 2013), 12 pages. https://doi.org/10.1145/2461912.2461916

Alec Jacobson, Daniele Panozzo, C Schüller, Olga Diamanti, Qingnan Zhou, N Pietroni, et al. 2018. libigl: A simple C++ geometry processing library.

Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. 2006. Poisson Surface Reconstruction. In *Symposium on Geometry Processing (SGP '06)*. Eurographics Association, 61–70.

Misha Kazhdan, Ming Chuang, Szymon Rusinkiewicz, and Hugues Hoppe. 2020. Poisson surface reconstruction with envelope constraints. In *Computer Graphics Forum (SGP)*, Vol. 39. 173–182.

Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. 2015. Stripe Patterns on Surfaces. *ACM Trans. Graph.* 34, 4 (2015).

Pavel A Krutitskii. 2001. The jump problem for the Laplace equation. *Applied Mathematics Letters* 14, 3 (2001), 353–358.

Joseph Louis Lagrange. 1798. *Solutions de quelques problèmes relatifs aux triangles sphériques, avec une analyse complète de ces triangles.*

Victor Lucquin, Sebastien Deguy, and Tamy Boubekeur. 2017. SeamCut: Interactive Mesh Segmentation for Parameterization. In *ACM SIGGRAPH 2017 Technical Briefs*.

Richard H MacNeal. 1949. *The solution of partial differential equations by means of electrical networks*. Ph.D. Dissertation. California Institute of Technology.

Claudio Mancinelli, Giacomo Nazzaro, Fabio Pellacini, and Enrico Puppo. 2021. B/Surf: Interactive Bézier Splines on Surfaces. *arXiv preprint arXiv:2102.05921* (2021).

James Clerk Maxwell. 1881. *A Treatise on Electricity and Magnetism*. Vol. II. Oxford University Press.

Margaret McIntyre and Grant Cairns. 1993. A new formula for winding number. *Geometriae Dedicata* 46, 2 (1993), 149–159.

Lea Müller, Ahmed A. A. Osman, Siyu Tang, Chun-Hao P. Huang, and Michael J. Black. 2021. On Self-Contact and Human Pose. In *Proceedings IEEE/CVF Conf. on Computer*

Vision and Pattern Recognition (CVPR).
James R Munkres. 1984. *Elements of Algebraic Topology* (1st. ed.). CRC press.
Ashish Myles, Nico Pietroni, and Denis Zorin. 2014. Robust Field-Aligned Global Parametrization. *ACM Trans. Graph.* 33, 4 (2014).
De-zhi Ning, Bin Teng, Hai-tao Zhao, and Chun-ling Hao. 2010. A comparison of two methods for calculating solid angle coefficients in a BIEM numerical wave tank. *Engineering Analysis with Boundary Elements* 34, 1 (2010), 92–96.
Konstantin Poelke and Konrad Polthier. 2016. Boundary-aware Hodge decompositions for piecewise constant vector fields. *Computer-Aided Design* 78 (2016), 126–136.
Bruce L Reinhart. 1960. The winding number on two manifolds. In *Annales de l'institut Fourier*, Vol. 10. 271–283.
Bruce L Reinhart. 1963. Further remarks on the winding number. In *Annales de l'institut Fourier*, Vol. 13. 155–160.
Marzia Riso, Giacomo Nazzaro, Enrico Puppo, Alec Jacobson, Qingnan Zhou, and Fabio Pellacini. 2022. BoolSurf: Boolean Operations on Surfaces. *ACM Trans. Graph.* 41, 6 (2022), 1–13.
Rohan Sawhney and Keenan Crane. 2017. Boundary First Flattening. *ACM Trans. Graph.* 37, 1, Article 5 (dec 2017), 14 pages. https://doi.org/10.1145/3132705
Günter Schwarz. 2006. *Hodge Decomposition-A method for solving boundary value problems*. Springer.
Silvia Sellán and Alec Jacobson. 2022. Stochastic Poisson Surface Reconstruction. *ACM Trans. Graph.* (2022).
Nicholas Sharp and Keenan Crane. 2020a. A Laplacian for Nonmanifold Triangle Meshes. *Computer Graphics Forum (SGP)* 39, 5 (2020).
Nicholas Sharp and Keenan Crane. 2020b. You Can Find Geodesic Paths in Triangle Meshes by Just Flipping Edges. *ACM Trans. Graph.* 39, 6 (2020).
Nicholas Sharp, Keenan Crane, et al. 2019a. GeometryCentral: A modern C++ library of data structures and algorithms for geometry processing. https://geometry-central.net/. (2019).
Nicholas Sharp, Mark Gillespie, and Keenan Crane. 2021. Geometry Processing with Intrinsic Triangulations. (2021).
Nicholas Sharp, Yousuf Soliman, and Keenan Crane. 2019b. Navigating Intrinsic Triangulations. *ACM Trans. Graph.* 38, 4 (2019).
M. Shimrat. 1962. Algorithm 112: Position of Point Relative to Polygon. *Commun. ACM* 5, 8 (aug 1962), 434. https://doi.org/10.1145/368637.368653
Xin Sun, Guofu Xie, Yue Dong, Stephen Lin, Weiwei Xu, Wencheng Wang, Xin Tong, and Baining Guo. 2012. Diffusion curve textures for resolution independent texture mapping. *ACM Trans. Graph.* 31, 4 (2012), 1–9.
Y. Tong, P. Alliez, D. Cohen-Steiner, and M. Desbrun. 2006. Designing Quadrangulations with Discrete Harmonic Forms. In *Symposium on Geometry Processing (SGP '06)*. Eurographics Association, 201–210.
Eric Veach and Leonidas J Guibas. 1995. Optimally combining sampling techniques for Monte Carlo rendering. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. 419–428.
Stephanie Wang and Albert Chern. 2021. Computing Minimal Surfaces with Differential Forms. *ACM Trans. Graph.* 40, 4 (August 2021), 113:1–113:14. https://doi.org/10.1145/3450626.3459781
Yakun Yuan, Dennis S Kim, Jihan Zhou, Dillan J Chang, Fan Zhu, Yasutaka Nagaoka, Yao Yang, Minh Pham, Stanley J Osher, Ou Chen, et al. 2022. Three-dimensional atomic packing in amorphous solids with liquid-like structure. *Nature Materials* 21, 1 (2022), 95–102.
Qingnan Zhou, Eitan Grinspun, Denis Zorin, and Alec Jacobson. 2016. Mesh Arrangements for Solid Geometry. *ACM Trans. Graph.* 35, 4, Article 39 (July 2016), 15 pages. https://doi.org/10.1145/2897824.2925901

A HODGE DECOMPOSITION FOR BROKEN CURVES

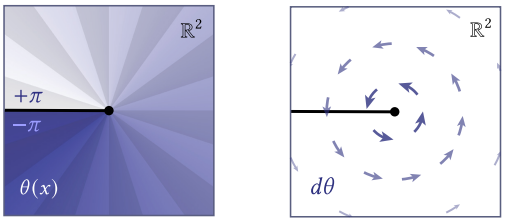
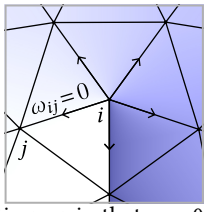


Fig. 30. At endpoints of Γ, the harmonic function u locally behaves like the angle function $\theta(x)$ (*left*). Intuitively, the derivative $\omega := \mathcal{D}u$ is divergence-free, but has singular curl, analogous to $d\theta$ (*right*). More precisely, $\delta\omega = 0$, but $d\omega \neq 0$.

Consider the Hodge decomposition $\omega = d\alpha + \delta\beta + \gamma$, where ω is the Darboux derivative of the harmonic function u from Section 3.2. In this appendix we will show that if Γ is broken, then $d\alpha$ is still zero everywhere, but $\delta\beta$ is nonzero due to the endpoints of Γ—motivating our need to perform a Hodge decomposition.



Recall that in Section 2.4.2 we define $\omega_{ij} := 0$ for all edges ij incident on an interior endpoint $i \in V \setminus V^*$ (see inset). Hence, for any singular vertex i, we have $(\delta\omega)_i = 0$, since the operator δ uses only edges ij incident on i. For all other vertices $i \in V^*$, one can easily verify that $\delta\omega = 0$ by comparing the expressions for $(\delta\omega)_i$ and the jump Laplacian $(L_J u)_i$ (noting again that $\omega = 0$ for edges connected to interior endpoints). In turn, since $\delta\omega = 0$ everywhere, $d\alpha$ must also be zero (Equation 9). However, $d\omega$ is not zero everywhere—in particular, for any triangle ijk incident on an interior endpoint i we have $(d\omega)_{ijk} = \cancelto{0}{\omega_{ij}} + \omega_{jk} + \cancelto{0}{\omega_{ki}}$. Moreover, at least one of the ω_{jk} around i must be nonzero, since (due to the jump) the value of u has to increase by a total value $(\partial\Gamma)_i \neq 0$. Hence, again due to Equation 9, we know $\delta\beta$ cannot be zero—and must be solved for in order to carry out the Hodge decomposition.

Winding Numbers on Discrete Surfaces (Supplemental Material)

This supplement provides detailed pseudocode (Section A) for the *surface winding number (SWN)* method of Feng *et al.* 2023, and discusses the homological perspective on this method (Section B).

A PSEUDOCODE

Our pseudocode is expressed via a halfedge mesh data structure encoding a triangle mesh $M = (V, E, F)$, and use $\overrightarrow{ij}$ to denote the halfedge from i to j. Subroutines not defined here are described in the list below; many correspond to standard libraries/data structures.

- IsManifold(M, i) — returns true if i is a manifold vertex of M.
- IsBoundary(M, ij) — returns true if ij is a boundary edge of M.
- Orientation$(M, \overrightarrow{ij})$ — returns true if the orientation of halfedge $\overrightarrow{ij}$ matches the canonical orientation of edge ij in M, and false otherwise.
- Twin$(M, \overrightarrow{ij})$ — returns the twin of halfedge $\overrightarrow{ij}$ in M.
- Prev$(M, \overrightarrow{ij})$ — returns the previous halfedge in the face containing halfedge $\overrightarrow{ij}$ of M.
- OppositeVertex$(M, \overrightarrow{ij})$ — returns the vertex k opposite $\overrightarrow{ij}$ in face ijk of M.
- CornersOf(M, i) — returns the set of corners ${}^{jk}_{i}$ incident on vertex i of M.
- EndpointsOf(M, Γ) — returns the set $V \setminus V^*$ of vertices comprising the interior endpoints of a discrete 1-chain Γ on M.
- InteriorVertices(M, Γ) — returns the set of vertices which are not interior endpoints of the discrete 1-chain Γ on M.
- OutgoingHalfedgeOnCurve(M, i, Γ) — for a vertex i in M, returns an arbitrary halfedge $\overrightarrow{ij}$ whose tail is i, such that $\Gamma_{ij} \neq 0$. If i is a boundary vertex, instead return the most clockwise halfedge.
- SolvePositiveSemidefinite(A, b) — solves a sparse positive semidefinite linear system $\mathsf{Ax} = \mathsf{b}$, returning x (and picking an arbitrary shift if A has constants in its null space).
- SolveLinearProgram$(M, \ell, \Gamma, \varepsilon, s)$ — solves the linear program in Equation 11, for a mesh M with edge lengths ℓ, curve Γ, parameter ε and shifts s.

Algorithm 1 SurfaceWindingNumber$(M, \ell, \theta, \Gamma, \varepsilon)$

Input: A 1-chain $\Gamma \in \mathbb{Z}^{|E|}$, on a mesh $M = (V, E, F)$ with edge lengths ℓ, corner angles θ, and a parameter ε for the linear program.

Output: The winding number function w defined on corners of M (Section 3.5).

$c \leftarrow$ ComputeReducedCoordinates(M, Γ) ▷ §2.3.1
$u \leftarrow$ SolveJumpEquation(M, θ, Γ, c) ▷ §2.4.3, §3.2
$\omega \leftarrow$ DarbouxDerivative(M, Γ, u) ▷ §2.4.2
$\gamma \leftarrow$ HarmonicPart(M, θ, ω) ▷ §3.3
$\mathring{v} \leftarrow$ IntegrateLocally(M, γ)
$s \leftarrow$ ComputeRelativeJumps$(M, \mathring{v})$
$\sigma \leftarrow$ SolveLinearProgram$(M, \ell, \Gamma, \varepsilon, s)$ ▷ §3.4
$v \leftarrow$ RecoverSolution$(M, \mathring{v}, \sigma)$
$\tilde{c} \leftarrow$ SubtractJumpDerivative(M, Γ, v, c) ▷ §2.4.1
$w \leftarrow$ SolveJumpEquation$(M, \theta, \Gamma, \tilde{c})$ ▷ §3.5
return w

Algorithm 2 ComputeReducedCoordinates(M, Γ)

Input: A 1-chain $\Gamma \in \mathbb{Z}^{|E|}$ on a mesh $M = (V, E, F)$.

Output: A function $c \in \mathbb{Z}^{|C|}$ expressing values at corners relative to some reference value (Section 2.3.1).

$c \leftarrow 0^{|C|}$
for $i \in$ InteriorVertices(M, Γ) **do**
  if IsManifold$(M, i) =$ False **then continue**
  $\overrightarrow{ij_0} \leftarrow$ OutgoingHalfedgeOnCurve(M, i, Γ)
  $\overrightarrow{ij} \leftarrow \overrightarrow{ij_0}$
  sum $\leftarrow 0$
  do
    if IsBoundary$(M, ij) =$ False **then**
      $k \leftarrow$ OppositeVertex$(M, \overrightarrow{ij})$
      jump $\leftarrow$ Orientation$(M, \overrightarrow{ij})$? $\Gamma_{ij} : -\Gamma_{ij}$
      sum += jump
      $c_i^{jk} \leftarrow$ sum
    $\overrightarrow{ij} \leftarrow$ Twin$(M,$ Prev$(M, \overrightarrow{ij}))$ ▷ *next outgoing halfedge*
  while $\overrightarrow{ij} \neq \overrightarrow{ij_0}$
return c

Algorithm 3 SolveJumpEquation(M, θ, Γ, c)

Input: A 1-chain $\Gamma \in \mathbb{Z}^{|E|}$ on a mesh $M = (V, E, F)$ with corner angles θ, and reduced coordinates $c \in \mathbb{R}^{|C|}$.

Output: A function $u \in \mathbb{R}^{|C|}$ defined on corners of M, where u solves Equation 10. Values at corners adjacent to endpoints of Γ are left undefined, to be interpolated using Equation 4.

$L \leftarrow$ BuildLaplacian(M, θ, Γ)
$b \leftarrow$ BuildJumpLaplaceRHS(M, θ, Γ, c)
$u_0 \leftarrow$ SolvePositiveSemidefinite(L, b)
$u \leftarrow 0 \in \mathbb{R}^{|C|}$ ▷ *Apply shifts to recover u (Section 3.2).*
for ${}^{jk}_{i} \in C$ **do** $u_i^{jk} \leftarrow u_0 + c_i^{jk}$
return u

Algorithm 4 DarbouxDerivative(M, Γ, u)

Input: A 1-chain $\Gamma \in \mathbb{Z}^{|E|}$, and a function $u \in \mathbb{R}^{|C|}$ with integer jumps across edges of a mesh $M = (V, E, F)$.

Output: The Darboux derivative $\omega \in \mathbb{R}^{|E|}$ of u, as a discrete 1-form on edges of M (Section 2.4.2).

$\omega \leftarrow 0 \in \mathbb{R}^{|E|}$
for $ij \in E$ **do**
  if $i \in$ EndpointsOf(M, Γ) or $j \in$ EndpointsOf(M, Γ) **then**
    continue
  $k \leftarrow$ OppositeVertex$(M, \overrightarrow{ij})$
  $\omega_{ij} \leftarrow u_j^{ki} - u_i^{jk}$
return ω

Algorithm 5 BuildLaplacian(M, θ, Γ)

Input: A 1-chain $\Gamma \in \mathbb{Z}^{|E|}$ on a mesh $M = (V, E, F)$ with corner angles θ.

Output: The operator $L \in \mathbb{R}^{|V^*|\times|V^*|}$ of Equation 10.

$L \leftarrow 0 \in \mathbb{R}^{|V^*|\times|V^*|}$ ▷ *initialize empty sparse matrix*
for $pqr \in F$ **do**
  for $ijk \in C(pqr)$ **do** ▷ *C: circular shifts*
    if $i \in$ EndpointsOf(M, Γ) or $j \in$ EndpointsOf(M, Γ) **then**
      continue
    $L_{ii},\ L_{jj}$ += $\frac{1}{2}\cot(\theta_k^{ij})$
    $L_{ij},\ L_{ji}$ −= $\frac{1}{2}\cot(\theta_k^{ij})$
return L

Algorithm 6 BuildJumpLaplaceRHS(M, θ, Γ, c)

Input: A 1-chain $\Gamma \in \mathbb{Z}^{|E|}$ on a mesh $M = (V, E, F)$ with corner angles θ, and reduced coordinates $c \in \mathbb{R}^{|C|}$ (Section 2.3.1).

Output: The vector $b \in \mathbb{R}^{|V^*|}$ in Equation 10.

$b \leftarrow 0 \in \mathbb{R}^{|V^*|}$
for $i \in$ InteriorVertices(M, Γ) **do**
  for $i^{jk} \in$ CornersOf(M, i) and $j, k \notin$ EndpointsOf(M, Γ) **do**
    b_i −= $\frac{1}{2}\cot(\theta_k^{ij}) \cdot c_i^{jk}$
    b_j += $\frac{1}{2}\cot(\theta_k^{ij}) \cdot c_i^{jk}$
    b_i −= $\frac{1}{2}\cot(\theta_j^{ki}) \cdot c_i^{jk}$
    b_k += $\frac{1}{2}\cot(\theta_j^{ki}) \cdot c_i^{jk}$
return b

Algorithm 7 HarmonicComponent(M, θ, ω)

Input: A co-closed 1-form $\omega \in \mathbb{R}^{|E|}$ on a mesh $M = (V, E, F)$ with corner angles θ.

Output: A harmonic 1-form $\gamma \in \mathbb{R}^{|E|}$.

$d_1 \leftarrow$ BuildOneFormExteriorDerivative(M)
$*_1 \leftarrow$ BuildOneFormHodgeStar(M, θ)
$\widetilde{\beta} \leftarrow$ SolvePositiveSemidefinite$(d_1 *_1^{-1} d_1^T, d_1\omega)$
$\delta\beta \leftarrow *_1^{-1} d_1^T \widetilde{\beta}$
$\gamma \leftarrow \omega - \delta\beta$
return γ

Algorithm 8 SubtractJumpDerivative(M, Γ, v, c)

Input: A 1-chain $\Gamma \in \mathbb{Z}^{|E|}$ on a mesh $M = (V, E, F)$, residual function $v \in \mathbb{R}^{|C|}$, and reduced coordinates $c \in \mathbb{R}^{|C|}$ associated with Γ.

Output: Updated reduced coordinates $\tilde{c}$ encoding new jump constraints for the jump Laplace equation (Section 3.5).

for $i \in$ InteriorVertices(M, Γ) **do**
  if IsManifold(M, i) = False **then** continue
  for $i^{jk} \in$ CornersOf(M, i) and $j, k \notin$ EndpointsOf(M, Γ) **do**
    if IsBoundary(M, ij) **then** continue
    $\ell \leftarrow$ OppositeVertex$(M,$ Twin$(M, \overrightarrow{ij}))$
    $\tilde{c}_i^{jk} = c_i^{jk} - (v_i^{jk} - v_i^{\ell j})$
return $\tilde{c}$

Algorithm 9 BuildOneFormExteriorDerivative(M)

Input: A mesh $M = (V, E, F)$.

Output: A sparse matrix $d_1 \in \mathbb{Z}^{|F|\times|E|}$ representing the discrete exterior derivative on 1-forms.

$d_1 \leftarrow 0 \in \mathbb{Z}^{|F|\times|E|}$ ▷ *initialize empty sparse matrix*
for $pqr \in F$ **do**
  for $ijk \in C(pqr)$ **do** ▷ *C: circular shifts*
    $(d_1)_{pqr,ij} \leftarrow$ Orientation$(M, \overrightarrow{ij})$? 1 : −1
return d_1

Algorithm 10 BuildOneFormHodgeStar(M, θ)

Input: A mesh $M = (V, E, F)$ with corner angles θ.

Output: A sparse diagonal matrix $*_1 \in \mathbb{Z}^{|F|\times|E|}$ representing the Hodge star acting on discrete 1-forms.

$*_1 \leftarrow 0 \in \mathbb{Z}^{|E|\times|E|}$ ▷ *initialize empty sparse matrix*
for $pqr \in F$ **do**
  for $ijk \in C(pqr)$ **do** ▷ *C: circular shifts*
    $(*_1)_{ij,ij}$ += $\frac{1}{2}\cot\theta_k^{ij}$
return $*_1$

Algorithm 11 IntegrateLocally(M, γ)

Input: A harmonic 1-form $\gamma \in \mathbb{R}^{|E|}$ on a mesh $M = (V, E, F)$.

Output: Corner values $\mathring{v}_i^{jk}$ integrating γ in each triangle of M.

for $ijk \in F$ **do**
  $g_{\overrightarrow{ij}} \leftarrow$ Orientation$(M, \overrightarrow{ij})$? $\gamma_{ij} : -\gamma_{ij}$
  $g_{\overrightarrow{jk}} \leftarrow$ Orientation$(M, \overrightarrow{jk})$? $\gamma_{jk} : -\gamma_{jk}$
  $\mathring{v}_i^{jk} \leftarrow 0$
  $\mathring{v}_j^{ki} \leftarrow g_{\overrightarrow{ij}}$
  $\mathring{v}_k^{ij} \leftarrow g_{\overrightarrow{ij}} + g_{\overrightarrow{jk}}$
return $\mathring{v}$

Algorithm 12 ComputeRelativeJumps$(M, \mathring{v})$

Input: A value $\mathring{v}_i^{jk}$ per corner of a mesh $M = (V, E, F)$.

Output: Values $s \in \mathbb{R}^{|E|}$ that give the jump between locally integrated values across each edge of M.

$s \leftarrow 0 \in \mathbb{R}^{|E|}$ ▷ *initialize zero vector*
for $ij \in E$ and IsBoundary(M, ij) = False **do**
  $s_{ij} \leftarrow \mathring{v}_i^{jk} - \mathring{v}_i^{lj}$
return s

Algorithm 13 RecoverSolution$(M, \mathring{v}, \sigma)$

Input: A value $\mathring{v}_i^{jk}$ per corner of a mesh $M = (V, E, F)$, and per-triangle shifts $\sigma \in \mathbb{R}^{|F|}$.

Output: A value v_i^{jk} per corner describing the residual function.

for $i^{jk} \in C$ **do** $v_i^{jk} \leftarrow \mathring{v}_i^{jk} + \sigma_{ijk}$
return v

B HOMOLOGICAL PERSPECTIVE

Here we discuss the homological perspective on SWN, starting with the case of closed, oriented surfaces (B.1) before proceeding to surfaces with boundary (B.2) and nonorientable surfaces (B.3). The basic tools are the first homology group $H_1(M)$ and cohomology group $H^1(M)$ of the surface M, which provide dual descriptions of its topology. Throughout we assume that M is manifold: while we find that SWN works on nonmanifold meshes in practice, the duality theorems formally apply only to manifolds.

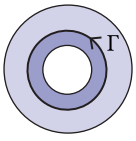
B.1 Overview of the Homological Picture

Homology is the theory of boundaries. A closed curve Γ on a surface M is said to be *nullhomologous* if Γ is the boundary of a region. Conversely, the homology group $H_1(M)$ describes loops which are not region boundaries. Munkres [1984, Chapters 1 & 5] gives a detailed introduction to homology and the dual theory of cohomology.

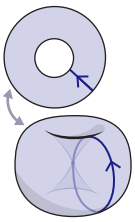
The connection to SWN is simplest when Γ is a closed curve on a closed, oriented surface M. In this case, the 1-form $\gamma = \mathcal{D}u$ computed in Section 3.3—known as the *Poincaré dual* of Γ—encodes Γ's homology class. Formally, Poincaré duality provides a canonical isomorphism $\varphi : H_1(M) \to H^1(M)$ [Munkres 1984, §65]. Concretely, this map provides a harmonic 1-form $\varphi(\Gamma)$ such that for any loop η, the integral $\int_\eta \gamma$ counts the signed number of intersections between η and Γ [Griffiths and Harris 2014, p.56]. Two closed curves Γ_1 and Γ_2 map to the same harmonic 1-form if and only if the curves are homologous. Hence, any jump harmonic function integrating $\gamma = \varphi(\Gamma)$—*e.g.* the function v in Section 3.4—must jump across a chain homologous to Γ. Consequently, the linear program used to compute v minimizes the ℓ^1 norm of the jump $g = \mathcal{J}v$ subject to the constraint that g is homologous to Γ. In this case, one could avoid cohomology and directly solve an optimal homologous chain problem *à la* Dey et al. [2010]. However, harmonic 1-forms are essential in our generalization to broken curves.

When Γ is broken, it lacks a well-defined homology class. Consequently, the 1-form $\mathcal{D}u$ is no longer harmonic for broken curves. Nonetheless, we can take the harmonic component γ of $\mathcal{D}u$, which we interpret as an "approximate homology class" for Γ. SWN then searches for the optimal nonbounding loop $g = Jv$ within this homology class. Among other things, the homology class constraint ensures that g is always a closed loop, even when Γ is broken.

B.2 Relative Homology for Surfaces with Boundary

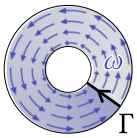


To make sense of our algorithm on surfaces with boundary—and in particular to justify Equation 12—we need to extend the discussion of homology to include *relative homology*. When M has no boundary, nullhomologous curves are precisely the curves enclosing regions, and nonbounding loops are characterized by the usual *absolute homology group* $H_1(M)$. However, the situation is more complicated if M has a boundary. For instance, an annulus has a single homology generator: a loop Γ wrapping around the middle. Though Γ separates the annulus into two components, it is not itself the boundary of any region since each component's boundary also includes a circle from the annulus' boundary.



Instead, nonbounding loops on a surface with boundary are described by the relative homology group $H_1(M, \partial M)$. On an annulus, *e.g.*, this group is generated by a curve connecting the boundary circles. Formally, it is the first homology group of M after collapsing ∂M to a point [Munkres 1984, §9]. *E.g.* collapsing the boundary of the annulus yields a sphere with two points identified, whose homology generator corresponds to the nonbounding curve on the annulus.

Relative Cohomology. Similarly, a surface with boundary has both absolute and relative cohomology groups. The absolute group $H^1(M)$ consists of harmonic 1-forms tangent to the boundary, while the relative group $H^1(M, \partial M)$ consists of harmonic 1-forms normal to the boundary [Poelke and Polthier 2016]. *Lefschetz duality* provides a map between $H_1(M, \partial M)$ and $H^1(M)$ [Munkres 1984, §70]. On an annulus, *e.g.*, the relative homology generator maps to a 1-form circulating around the center. Since nonbounding loops correspond to the relative homology group, our dual harmonic 1-forms are members of $H^1(M)$ and must thus lie tangent to ∂M.



Hodge Decomposition. On manifolds with boundary, one can decompose a k-form ω using the Hodge-Friedrichs-Morrey decomposition [Schwarz 2006, Corollary 2.4.9]:

$$\Omega^k = d\Omega_D^{k-1} \oplus \delta\Omega_N^{k+1} \oplus \left(\mathcal{H}^k \cap d\Omega^{k-1}\right) \oplus \mathcal{H}_N^k \tag{12}$$

$$= d\Omega_D^{k-1} \oplus \delta\Omega_N^{k+1} \oplus \left(\mathcal{H}^k \cap \delta\Omega^{k+1}\right) \oplus \mathcal{H}_D^k \tag{13}$$

Here a subscript D (for Dirichlet) denotes a space of forms with zero tangential component on ∂M, a subscript N (for Neumann) denotes a space of forms with zero normal component on ∂M, and $\mathcal{H}$ denotes the space of harmonic fields, (*i.e.* k-forms satisfying $d\gamma = \delta\gamma = 0$).

To extract a tangential harmonic 1-form, we apply the first decomposition (Equation 12). Multiplying both sides by d and δ yields a pair of equations determining the tangential harmonic component of a 1-form ω. A short calculation shows that these equations are the standard equations solved to perform Hodge decomposition on closed surfaces with zero-Neumann conditions on the boundary.

B.3 Local Coefficients for Nonorientable Surfaces

As discussed in Section 3.7, our algorithm also extends to nonorientable surfaces so long as one explicitly provides curve normals which specify which direction the surface winding number should jump across the curve. Such a choice of normals makes Γ into an element of the first homology group with local coefficients in the sense of Hatcher [2002, Section 3.H], which is Poincaré dual to the ordinary first cohomology group [Hatcher 2002, Theorem 3H.6].

Hodge Decomposition. Our discussion of Hodge decomposition used the codifferential $\delta := *d*$, which may look ill-defined on nonorientable surfaces: the definition uses $*$ which depends on the orientation of M. However, reversing orientation multiplies $*$ by -1, so because δ uses $*$ twice the signs cancel and δ remains well-defined on nonorientable surfaces. Hence, Hodge decomposition still works via the usual linear systems.

Received January 2023