Fast and Exact Winding Numbers for Triangle Meshes

PEIYUAN XIE, ISTA, Austria
CHRISTIAN HAFNER, ISTA, Austria
CHRIS WOJTAN, ISTA, Austria

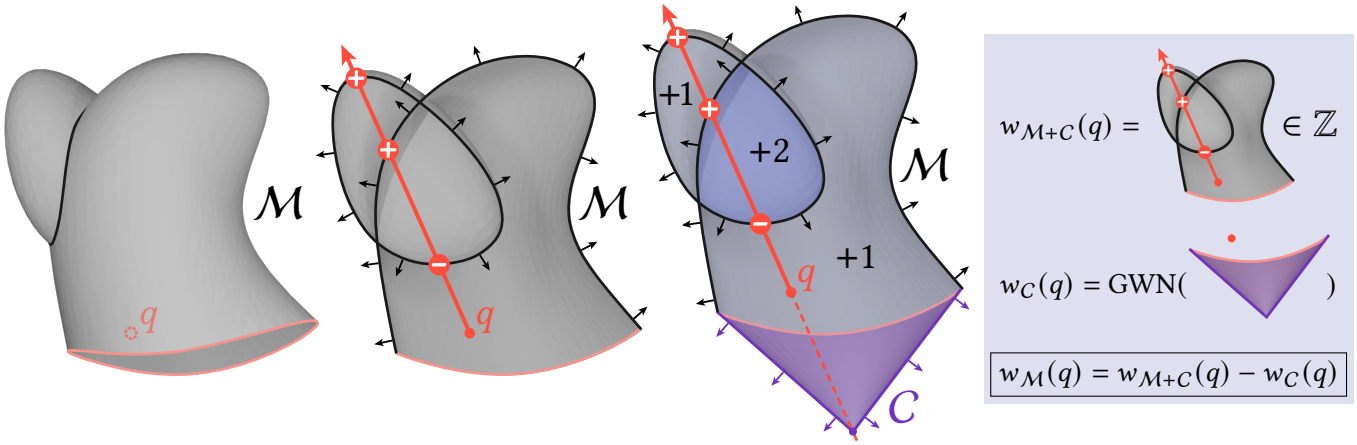


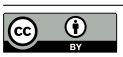
Fig. 1. **Generalized Winding Number (GWN) Algorithm.** *Left:* An input mesh $\mathcal{M}$ with self-intersections (black) and boundaries (pink). The goal is to evaluate the GWN of $\mathcal{M}$ at a query point q. *Center-left:* We shoot a ray from q in an arbitrary direction and count the signed number of intersections with $\mathcal{M}$, shown in a cut-away view. *Center-right:* We attach a generalized cone $\mathcal{C}$ with apex directly behind the ray to the boundary of $\mathcal{M}$. This produces a closed mesh $\mathcal{M} + \mathcal{C}$, and $\mathcal{C}$ will not intersect the ray. *Right:* The GWN of $\mathcal{M} + \mathcal{C}$ equals the signed number of ray intersections with $\mathcal{M}$. The GWN of $\mathcal{C}$ is cheap to evaluate directly because $\mathcal{C}$ has very few faces compared to $\mathcal{M}$. The GWN of $\mathcal{M}$ is the difference between the two quantities.

We revisit the computation of 3D generalized winding numbers, a useful measure for inside-outside classification on triangle meshes with gaps, self-intersections, and open boundaries. At the core of our new method is an analytical reduction of the surface integral that defines the winding number, resulting in a single ray-mesh intersection test and an elementary sum over boundary edges per evaluation. This construction is orders of magnitude more efficient than the state of the art in practice, which we show in an extensive performance benchmark. Conveniently, the method also reduces to the best-available asymptotic complexity in the worst case, and it introduces no approximations apart from floating-point errors. Our algorithm is conceptually simple to understand, straightforward to implement and debug, and it works reliably even on extremely noisy and corrupt input geometry.

CCS Concepts: • **Computing methodologies** → **Shape analysis**; **Mesh models**.

ACM Reference Format:
Peiyuan Xie, Christian Hafner, and Chris Wojtan. 2026. Fast and Exact Winding Numbers for Triangle Meshes. *ACM Trans. Graph.* 45, 4, Article 41 (July 2026), 8 pages. https://doi.org/10.1145/3811339

Authors' Contact Information: Peiyuan Xie, ISTA, Klosterneuburg, Austria, Peiyuan.Xie@ist.ac.at; Christian Hafner, ISTA, Klosterneuburg, Austria, christian.hafner@ist.ac.at; Chris Wojtan, ISTA, Klosterneuburg, Austria, wojtan@ist.ac.at.



ACM 1557-7368/2026/7-ART41
https://doi.org/10.1145/3811339

1 Introduction

The generalized winding number (GWN) [Jacobson et al. 2013] is a common and fundamental tool in geometry processing that gives a measure of how confident we can be that a point is inside or outside (or wrapped multiple times inside) a given triangle mesh. It is useful for computing signed distances and cleaning meshes with self-intersections and topological noise, among other applications. This paper introduces a novel method for computing GWNs for triangle meshes accurately and more efficiently than all existing techniques, including those that give approximate results.

As we explain in Section 2, the generalized winding number is defined as an integral over a surface mesh. Existing methods for computing the GWN vary in accuracy and efficiency: discretizing the integral as a summation over surface triangles [Jacobson et al. 2013] gives the exact answer, but it is computationally expensive; approximating the integral using Barnes-Hut summation [Barill et al. 2018] trades off accuracy for computational speed; recent work by Martens and Bessmeltsev [2025] transforms the GWN surface integral into a lower-dimensional but computationally challenging calculation of finding all self-intersections of spherical polygons; and the work of Spainhour et al. [2024] reduces a curve integral down to a difference computation between boundary points using the fundamental theorem of calculus, but is restricted to two-dimensionsal applications. Spainhour and Weiss [2026] extends Spainhour et al. [2024] to 3D parametric surfaces and uses numerical quadrature to approximate curve integrals, but it is not obvious how to arrive

at an exact closed-form solution for the common case of triangle meshes.

In this paper, we present a transformation of the GWN surface integral into a single ray-mesh intersection calculation and an elementary sum over the triangle mesh boundary, as shown in Fig. 1. This approach always gives the exact answer, and it is orders of magnitude more efficient than the state of the art in practice. It also conveniently reduces to the best-available asymptotic behavior in extreme cases, i.e., closed meshes only require a single ray-mesh intersection test, and a triangle soup requires iteration over all triangles. Our algorithm is conceptually simple to understand, straightforward to implement and debug, and works robustly for extremely noisy and corrupt input geometry.

2 Background

The winding number has a long and varied history. Originally defined for closed planar curves and polygons, this mathematical concept can be traced back to early works by Meister [1769], Gauss [1799], and Möbius [1865]. In the computer graphics community, a generalized formulation of the winding number was introduced by Jacobson et al. [2013] to handle open curves and surfaces, and has since become a powerful tool for robust inside-outside tests. In more recent years, the generalized winding number has been leveraged in a variety of applications, including shape modeling [Hu et al. 2020; Wang et al. 2025a; Xu et al. 2023], sketch colorization [Dong et al. 2025; Scrivener et al. 2024], animation [Dodik et al. 2025; Nuvoli et al. 2022], fabrication [Duenser et al. 2020] and physics-based simulation [Chang et al. 2025; Wang et al. 2025b].

In the remainder of this section, we briefly review the classical notion of the winding number and several equivalent interpretations, which serve as conceptual foundations for our algorithm described later in the paper.

In the classical 2D setting, the winding number of a closed oriented curve at a point is a signed integer, measuring how many times the curve encircles the point. Equivalently, one can interpret the winding number as the total signed angle subtended by the curve with respect to the point, normalized by the full angle 2π. This interpretation leads to a concrete line-integral formula for evaluation: given an oriented curve γ, the winding number at a point $q \in \mathbb{R}^2 \setminus \gamma$ can be expressed as the integral of the differential angle, formally denoted by $d\theta$, subtended at q:

$$w_\gamma(q) = \frac{1}{2\pi} \int_\gamma d\theta(q). \tag{1}$$

This interpretation also naturally extends the notion of winding number to open curves, where the resulting winding number is not restricted to integer values but can vary continuously.

As an alternative, the winding number can be understood through the principle of ray casting: The winding number at a point is equivalent to the average number of signed intersections between the curve and rays cast in all possible directions from the point. While conceptually elegant, this approach is impractical for computation. However, for closed shapes, the sum of signed intersections is independent of the ray direction, which provides a computational simplification,

$$w_\gamma(q) = \sum_i \operatorname{sgn}(r \cdot n_i), \tag{2}$$

where r is the direction of an arbitrary ray emanating from q, and n_i is the normal direction of the curve at the i-th intersection point with the ray. This simplification makes it feasible to use ray tracers to efficiently compute winding numbers for closed shapes.

The notion of winding number naturally generalizes from planar curves to spatial surfaces. In 3D, the generalized winding number measures how many times an oriented surface wraps around a point. By replacing angles with their 3D counterparts—solid angles—the generalized winding number at a point $q \in \mathbb{R}^3 \setminus \mathcal{S}$ can be expressed as a surface integral that denotes the total signed solid angle subtended by the surface $\mathcal{S}$ with respect to the point, normalized by the full solid angle 4π:

$$w_\mathcal{S}(q) = \frac{1}{4\pi} \int_\mathcal{S} d\Omega(q). \tag{3}$$

Similarly, the principle of ray casting is directly applicable to 3D, and the simplification is analogously valid for watertight surfaces.

Apart from efforts to generalize winding numbers across dimensions, other lines of work have explored alternative extensions. Feng et al. [2023] proposed a generalization of winding numbers for curves on surfaces, providing powerful tools for various surface processing algorithms that require meaningful inside-outside distinctions. Sun et al. [2024] introduced a Gaussian-smoothed winding number that can be efficiently differentiated and used for optimization problems. However, this line of research focuses on a distinct problem domain and is beyond the scope of our paper.

3 Related Work

Different techniques have been proposed for computing generalized winding numbers across various geometric representations. For triangle meshes, a direct summation of solid angles over all triangles is straightforward but results in poor asymptotic runtime on large models. To address this limitation, Jacobson et al. [2013] introduced a hierarchical evaluation algorithm that performs asymptotically better than the naive implementation.

Barill et al. [2018] adopted a similar divide-and-conquer idea in designing fast approximation algorithms for more unstructured geometric data, such as oriented point clouds. While their technique can be directly applied to triangle soups and connected meshes, the performance gains come at the expense of exact accuracy, as the resulting values are merely approximations of the actual winding number.

For curved geometry, both of the methods mentioned above require a discretization of the whole surface, which may fail to capture the true geometry and can introduce errors in downstream applications. To ensure geometric fidelity, Spainhour et al. [2024] developed an algorithm to compute winding numbers in 2D, which directly handles collections of planar rational parametric curves and can achieve exact and efficient evaluation without resorting to numerical quadrature. In a different direction, Liu et al. [2025] derived closed-form expressions for the winding number using the residue theorem, yielding concise analytic formulas that can offer improved performance, particularly for query points close to the curve.

Most related to our method are the recent work by Martens and Bessmeltsev [2025] and the work by Spainhour and Weiss [2026]. Both recognized that, for a surface, the generalized winding number can be determined entirely from the surface boundaries together with additional global information that can be obtained through ray or line intersection queries. Accordingly, their algorithms rely only on boundary operations and surface intersection tests, and the computational complexity of their approaches is governed primarily by the complexity of processing the boundary rather than that of the entire surface.

However, these methods have different practical limitations. The OneShot algorithm by Martens and Bessmeltsev [2025] requires pairwise intersection detection between spherical line segments as part of its evaluation procedure. This can be a performance bottleneck and is nontrivial to implement efficiently and robustly in practice. The recent work of Spainhour and Weiss [2026] reformulates the relevant surface integral via Stokes' theorem and reduces it to a line integral over the boundary, which is evaluated using numerical quadrature. Their algorithm is presented for parametric surfaces and naturally extends the work by Spainhour et al. [2024] to three dimensions. Our work shares a similar spirit but uses a different derivation, which results in a closed-form solution for triangle meshes that cannot easily be gleaned from the work by Spainhour and Weiss [2026].

A heuristic method that uses ray casting for inside-outside tests on non-watertight meshes, known as ray stabbing, aggregates the number of mesh intersections with rays in different directions by voting. Variants of this idea have been used for fluid-solid interaction [Houston et al. 2003], mesh repair [Nooruddin and Turk 2003], and skinning-weight computation [Dionne and de Lasa 2014]. On closed meshes, the number of mesh intersections is theoretically independent of the ray direction, but implementations must take special care not to miss or double-count intersections, for example when a ray crosses the mesh exactly in an edge or in a vertex.

We could resolve these degenerate configurations using a fully robust ray tracer such as one that uses exact or adaptive precision [Richard Shewchuk 1997], or simulation of simplicity [Edelsbrunner and Mücke 1990], a technique that symbolically perturbs the geometry to resolve degeneracies and eliminates the need for extensive special-case handling in an implementation [Heiss-Synak et al. 2024]. However, most high-performance ray tracing libraries do not natively provide such support without low-level modifications. Another common approach is to numerically perturb the ray to reduce the likelihood of exact degeneracy [Cherchi et al. 2022; Müller 2009], but this does not provide robustness guarantees and can still fail in challenging configurations. Instead of relying on heuristic techniques, we handle this problem explicitly by detecting intersections that are potentially incorrect and locally falling back to a safe strategy.

4 Method

In this section, we present our motivation and describe the proposed algorithm in the context of triangle meshes. Triangles are ubiquitous in graphics and geometry processing applications. Moreover, they admit an exact formula for computing the solid angle they subtend at a point, which makes evaluation of the generalized winding number both simple and efficient. For this reason, we restrict our derivation and the following experiments to triangle meshes.

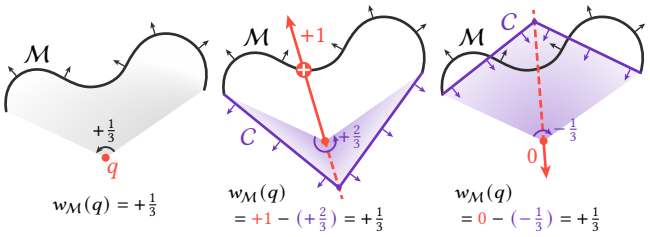


Fig. 2. **2D Example.** *Left:* The GWN of $\mathcal{M}$ at q is defined as the signed subtended angle (as a fraction of 2π). *Center:* We choose a ray (red) emanating from q and attach a cone $\mathcal{C}$ (purple) with apex directly behind the ray to the boundary of $\mathcal{M}$. The GWN is then given by the signed number of ray intersections with $\mathcal{M}$ minus the signed angle subtended by the cone. *Right:* The result is correct regardless of ray direction.

Ray casting is one of the most efficient ways to evaluate the integer winding number for closed meshes, as modern ray tracers are highly optimized for triangles and achieve improved asymptotic performance using acceleration structures. Moreover, a single ray suffices to determine the winding number for all query points that lie along that ray. However, as discussed earlier, it is infeasible to use ray casting alone to compute the exact generalized winding number for meshes with open boundaries.

In contrast, evaluating the generalized winding number by summing the solid angle subtended by each triangle applies to both open and closed meshes. While this formulation is inherently amenable to a parallel implementation, its computational cost still grows linearly with the total number of faces, which can be expensive for large meshes.

The additive property of the generalized winding number bridges these two ideas: For any two oriented meshes $\mathcal{M}$ and $\mathcal{C}$, the winding number of their sum[1] is equal to the sum of their individual winding numbers,

$$w_{\mathcal{M}+\mathcal{C}}(q) = w_{\mathcal{M}}(q) + w_{\mathcal{C}}(q). \tag{4}$$

By choosing $\mathcal{C}$ such that $\mathcal{M}+\mathcal{C}$ forms a consistently oriented closed mesh, this property enables an efficient indirect evaluation of the winding number, which is the key motivation behind the algorithm we describe next.

Given an arbitrary triangle mesh $\mathcal{M}$ with open boundary, we close it with another mesh $\mathcal{C}$ that shares the same boundary and is oriented consistently. Rather than directly evaluating the generalized winding number of $\mathcal{M}$ at a query point, we first compute the integer winding number of the closed mesh $\mathcal{M}+\mathcal{C}$ via a single ray cast, and then subtract the contribution of the closing mesh $\mathcal{C}$

[1]Strictly speaking, this is a sum of two 2-chains, so faces with opposite orientations cancel each other.

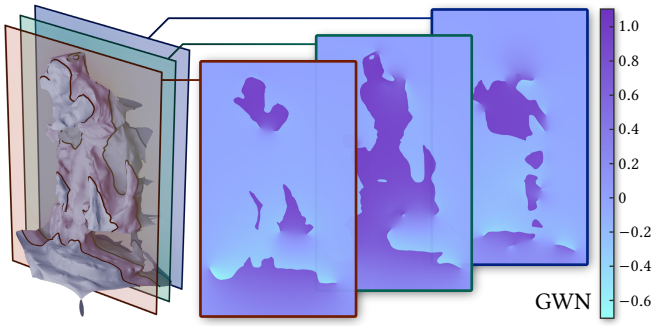


Fig. 3. Generalized winding number field of the Thingi10K model "Nydia" with open boundaries, shown on three cross sections. The GWN field jumps by ±1 across the intersection curves between the model and image plane.

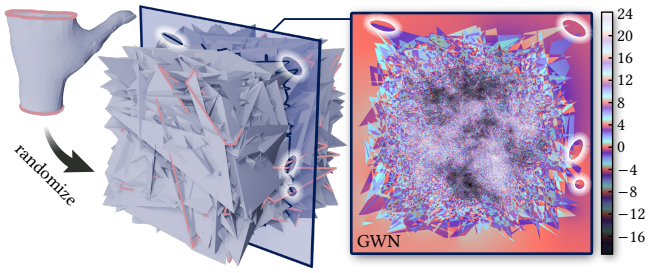


Fig. 4. **Stress Test.** We take a "hand" model *(top-left)* with two open boundary curves (pink) from the Thingi10K database, and assign uniformly random vertex positions *(center)*. The GWN field *(right)* is shown on a cross section of 2000^2 query points, with some corresponding locations marked by white ellipses. Compared to the hierarchical method [Jacobson et al. 2013], the result has an L^∞-error of $2.95 \cdot 10^{-12}$ and an L^1-error of $1.77 \cdot 10^{-8}$.

determined via a sum over its triangles:

$$\begin{aligned} w_{\mathcal{M}}(q) &= w_{\mathcal{M}+\mathcal{C}}(q) - w_{\mathcal{C}}(q) \\ &= \sum_i \operatorname{sgn}(r \cdot n_i) - \frac{1}{4\pi} \int_{\mathcal{C}} d\Omega(q) \\ &= \sum_i \operatorname{sgn}(r \cdot n_i) - \frac{1}{4\pi} \sum_j \Omega_j, \end{aligned} \tag{5}$$

where Ω_j is the solid angle subtended at q by the j-th triangle in $\mathcal{C}$. For a triangle with vertex positions a, b, and c relative to the query point, the solid angle it subtends can be efficiently computed as [Van Oosterom and Strackee 1983]:

$$\Omega = 2 \arctan\left(\frac{\det(a,b,c)}{abc + (a\cdot b)c + (b\cdot c)a + (c\cdot a)b} \right), \tag{6}$$

where $a = \|a\|$, $b = \|b\|$, $c = \|c\|$.

The above argument is true *for any* closing surface $\mathcal{C}$, so we intentionally design $\mathcal{C}$ such that it minimizes computational cost. As illustrated in Fig. 2, the closing mesh $\mathcal{C}$ is constructed as a generalized cone: The cone has the same boundary as the open mesh $\mathcal{M}$, and its apex is chosen along the opposite direction of the ray from the query point. This construction offers two performance advantages. First, the cone is guaranteed to have no intersections with the ray by construction. As a result, casting rays to the combined mesh $\mathcal{M} + \mathcal{C}$ reduces to casting rays just to the original open mesh $\mathcal{M}$, avoiding the need for any acceleration structures or intersection tests with the surface $\mathcal{C}$. Second, the cost of directly evaluating the contribution $w_{\mathcal{C}}(q)$ of the closing mesh scales linearly with the number of boundary edges in $\mathcal{M}$, instead of the total number of faces in $\mathcal{M}$. Since the boundary edges are typically much fewer than the total triangles in practical models, evaluating $w_{\mathcal{C}}(q)$ is significantly cheaper than evaluating $w_{\mathcal{M}}(q)$ via a full surface integral.

Robustness. Most high-performance ray-casting packages do not guarantee exactness in near-degenerate situations, and they might, e.g., double-count ray-mesh intersections located very close to a mesh edge or vertex. To guarantee exactness of our algorithm regardless, we mark each intersection point that is not clearly in the interior of a mesh face. Then we locally extract all triangles adjacent to these intersections and connect their boundary to the apex point behind the ray. This produces a small closed auxiliary mesh whose winding number equals the signed number of ray intersections in this region, the quantity which we were previously unable to determine reliably by means of the ray-caster. This construction allows us to fall back to using Eq. 6 to determine the winding number of the small auxiliary mesh, which then replaces the signed number of ray intersections within this region, when we evaluate Eq. 5. Since such degenerate cases are extremely rare in practice, their impact on the overall performance is negligible.

4.1 Relation to Stokes' Theorem

The GWN at a query point q can be written as a surface flux integral over $\mathcal{M}$ of the vector field $v(x) = (x-q)/\|x-q\|^3$, which is divergence-free. Nevertheless, Stokes' theorem does not apply directly because it is only guaranteed to hold for vector fields defined on a contractible domain, i.e., a domain that can be continuously shrunk to a point; however, the domain of v is $\mathbb{R}^3 \setminus \{q\}$, which is not contractible. Conceptually, we can interpret the ray $\mathbf{R} \subset \mathbb{R}^3$ that appears in our algorithm as a tool for defining the contractible domain $\mathbb{R}^3 \setminus \mathbf{R}$. Contractability guarantees that we can find a surface with the same boundary as $\mathcal{M}$ that is fully contained in $\mathbb{R}^3 \setminus \mathbf{R}$. The cone $\mathcal{C}$ is a particularly simple example of such a surface, and this facilitates the efficient construction in our algorithm.

The concurrent work by Spainhour and Weiss [2026] achieves a dimension reduction from a surface integral to a line integral over the mesh boundary by using a vector potential of v on a domain $\mathbb{R}^3 \setminus \mathbf{L}$, where $\mathbf{L}$ is a line rather than a ray. A domain of this form is not contractible, so one cannot generally find a surface with the same boundary as $\mathcal{M}$ that is fully contained in it. This hinders a direct transformation of the line integral into a sum over GWNs of triangles, for which the closed-form expression Eq. 6 is available.

5 Results

We implemented our algorithm in C++, using Eigen [Guennebaud et al. 2010] and libigl [Jacobson et al. 2018] for geometry processing and Embree [Wald et al. 2014] for ray casting. We represent triangle meshes and evaluate their winding numbers in double-precision. While Embree uses single-precision internally, it is only used to compute the integer winding number of closed meshes and

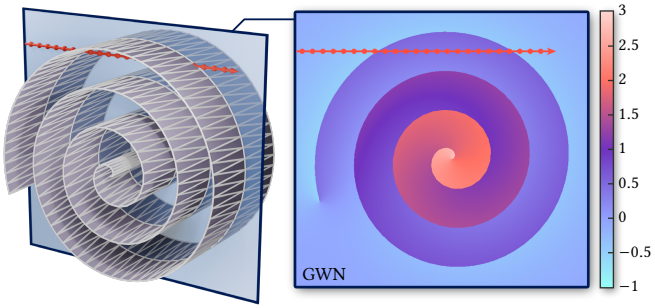
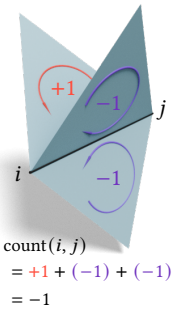


Fig. 5. **Swiss Roll.** *Left:* Synthetic model with wireframe and an image plane chosen to be coplanar to an edge loop of the mesh. This causes every ray (red) used in the GWN computation to intersect the mesh exactly in its edges. *Right:* Our robustness check handles this degenerate situation correctly while enabling ray reuse for all samples on a grid line.

thus will not affect the precision of the final result. Additionally, we fall back to direct winding number evaluation if intersection and query points are in very close proximity. Our implementation can be found at https://git.ista.ac.at/wojtan-group/peiyuan-xie/boundaryblaze. For simplicity, rays are generated from each query point along the $+x$ direction. On a regular grid, a single ray is reused for all points sharing the same y and z coordinates. All performance measurements were obtained via a single-threaded implementation on a MacBook Air M2. To accept inputs with non-manifold edges, as seen in the inset, faces of $\mathcal{C}$ are attached to all *exterior edges* [Jacobson et al. 2013] of $\mathcal{M}$. These are the edges that appear with a non-zero count in the boundary of its incident faces, taking into account orientation.



5.1 Baseline Examples

We evaluate the correctness and robustness of our method on a set of models with significant geometric defects that pose challenges for winding number computation. First, we test a practical mesh from the Thingi10K dataset [Zhou and Jacobson 2016] with self-intersections, open boundaries, non-manifold pieces and multiple connected components. Fig. 3 visualizes the winding number field across three cross-sections of the Nydia model. In contrast to conventional ray-casting and ray-stabbing approaches, our cone-closing algorithm remains insensitive to these geometric defects, yielding stable and accurate results.

To further assess robustness, we stress-test our algorithm on more extreme meshes. To produce Fig. 4, we used a model from the Thingi10K dataset and moved every vertex to a random location in the unit cube, resulting in a highly distorted mesh with numerous self-intersections and irregular boundaries. In Fig. 5, we evaluate the GWN of a "Swiss roll" model on a deliberately positioned slicing plane such that every ray intersects the mesh exactly in a vertex or an edge. Despite these adversarial conditions, our approach consistently resolves all geometric degeneracies and produces accurate winding number fields on the entire cross section.

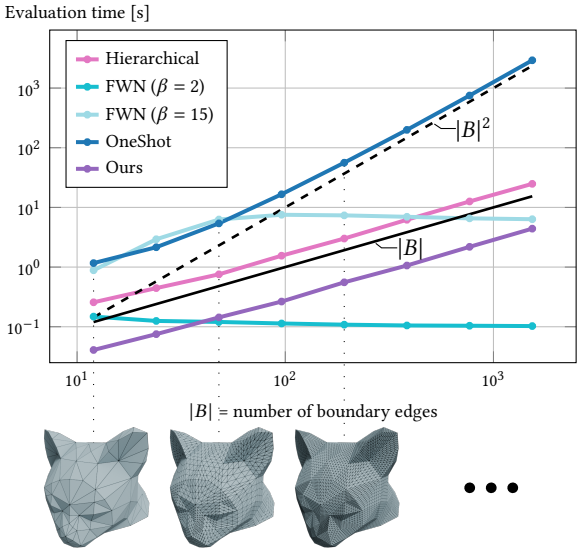


Fig. 6. **Asymptotic performance.** We compare the asymptotic behavior of our method (purple) on the subdivided cat head model *(bottom)* against hierarchical evaluation (pink), FWN with two different accuracy settings (cyan) and the OneShot algorithm (blue). Our approach exhibits the best asymptotic behavior and lowest computation time among the exact methods.

5.2 Asymptotic Performance

In Fig. 6 we demonstrate the asymptotic performance of our algorithm on a sequence of subdivided meshes with boundary. The number of boundary edges $|B|$ grows linearly with the subdivision level, while the number of faces $|F|$ grows quadratically, yielding the asymptotic law $|B| \sim |F|^{0.5}$.

We perform the evaluation of the GWN at 500^2 sample locations on a cross section of the mesh. Thanks to the high efficiency of modern ray tracers and sharing the cost of intersection queries across all query points along the ray, the computational cost of ray-casting is negligible on meshes with non-empty boundary. As a result, the cost of evaluating the generalized winding number of the cone formed by boundary edges dominates in our method. This cost grows linearly with $|B|$, so the overall time complexity of our algorithm is $O(|B|)$.

In contrast, the OneShot algorithm [Martens and Bessmeltsev 2025] has a time complexity of $O(|B|^2)$, or equivalently $O(|F|)$, due to the need to find spherical segment intersections among projected boundary edges. While the hierarchical approach [Jacobson et al. 2013] shows an asymptotic behavior similar to ours, it incurs larger absolute evaluation times due to the more complicated boundaries that arise after their divide-and-conquer subdivisions.

The fast winding number (FWN) algorithm [Barill et al. 2018] has near-constant complexity for a fixed accuracy parameter value and, interestingly, tends to be slightly more efficient on denser meshes. This is likely because smaller triangles can be more effectively grouped by the BVH, allowing the algorithm to avoid many exact evaluations and rely more on aggregated approximations. However, this acceleration strategy is inherently approximate and

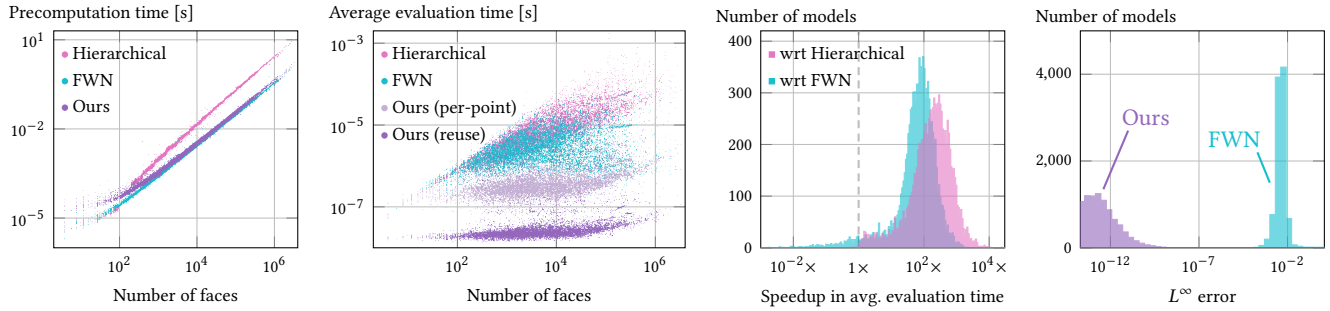


Fig. 7. **Empirical performance.** We compare the performance of our method (purple) against hierarchical evaluation (pink) and FWN with $\beta = 2$ (cyan) on all triangle meshes from the Thingi10K dataset. Our method achieves precomputation times comparable to FWN *(left)*, orders of magnitude faster evaluation times *(center-left)* and significant speedup compared to both methods *(center-right)*, and introduces no approximation errors *(right)*.

cannot be used to produce exact results. With a low accuracy parameter ($\beta = 2$), which controls the relative radius to distinguish near-field and far-field, FWN attains high efficiency but introduces errors up to $5 \cdot 10^{-3}$ in the winding number field. To reduce the mean absolute error for the geometry in Fig. 6 below single-precision floating-point machine epsilon ($\sim 10^{-7}$) across all subdivision levels, FWN requires a larger near-field ($\beta = 15$), which in turn increases computational cost significantly.

While approximate winding numbers suffice for most applications, rigorous error bounds are not available for any existing approximate algorithms. Consequently, approximation errors can be unpredictable and necessitate extensive parameter tuning to achieve the desired performance and precision. In worst-case scenarios, high approximation errors may persist regardless, as we show in Table 1. By contrast, our method's exactness inherently precludes unexpected failure cases and removes the burden of manual parameter selection.

Table 1. **Error statistics on Thingi10K dataset.** We use the results from hierarchical evaluation as the ground truth and compare the average and maximum L^1 and L^∞ errors of the winding number fields generated by FWN ($\beta = 2$), FWN ($\beta = 15$) and our method.

Method	Max L^∞ err	Max L^1 err	Avg L^∞ err	Avg L^1 err
FWN_2	$1.15 \cdot 10^{0}$	$1.06 \cdot 10^{3}$	$6.95 \cdot 10^{-3}$	$4.95 \cdot 10^{1}$
FWN_{15}	$1.15 \cdot 10^{0}$	$1.01 \cdot 10^{3}$	$2.95 \cdot 10^{-3}$	$2.82 \cdot 10^{-1}$
Ours	$1.46 \cdot 10^{-7}$	$8.80 \cdot 10^{-7}$	$2.31 \cdot 10^{-11}$	$4.40 \cdot 10^{-10}$

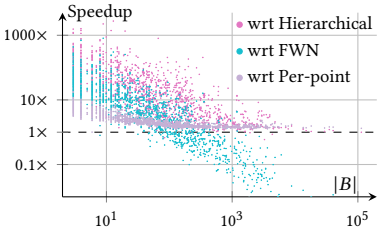
5.3 Empirical Performance

In Fig. 7, we evaluate winding numbers for every triangle mesh in the Thingi10K dataset. Every model from the dataset is loaded from disk and converted to a face-vertex list, which serves as input for the evaluation. This preprocessing step is applied uniformly across all tested methods and is not included in the timings.

For each mesh, we compute the winding number field on a 50^3 lattice spanning its bounding box, and compare precomputation time and average per-sample evaluation time across three methods: hierarchical evaluation [Jacobson et al. 2013], FWN [Barill et al. 2018], and ours. The OneShot algorithm is excluded from this comparison because it is primarily efficient on meshes with many triangles and a simple boundary. Our earlier experiment in Fig. 6 confirms that it underperforms relative to the hierarchical evaluation in general cases. For FWN, we continue to use a second-order approximation with a low accuracy parameter ($\beta = 2$).

Running Times. In the precomputation stage, our method is significantly faster than hierarchical evaluation on large models but slightly more costly than FWN, see Fig. 7 *(left)*. This is because we build additional mesh-adjacency structures during initialization to identify and resolve degenerate cases during ray casting, which is a bit more expensive than constructing the BVH used in FWN.

However, these additional structures facilitate a significantly more efficient evaluation phase, as shown in Fig. 7 *(center-left)*. In terms of per-sample evaluation time, our method achieves orders of magnitude speedup over both baselines on a fixed-size query grid, centered at a performance gain of about 100×, see Fig. 7 *(center-right)*. The lowest evaluation times are naturally achieved on models with no exterior edges, because then our algorithm reduces to N^2 ray-mesh intersections that are reused to compute all GWN samples on an N^3 grid. To isolate the benefits of ray reuse, we also conducted a study by running our algorithm with each query point processed independently; this mimics the setting of computing the winding numbers at unstructured sample locations. While omitting ray reuse slows down the computation on nearly closed meshes by a factor close to N, the impact is negligible on meshes with a large number of boundary edges, where the cone evaluation dominates the runtime. Even this version of the algorithm remains markedly more efficient than other methods on the Thingi10K dataset, as shown in Fig. 7 *(center-left)*. Only for meshes with very little connectivity, such as triangle soups, do exact methods like ours and hierarchical evaluation provide no acceleration

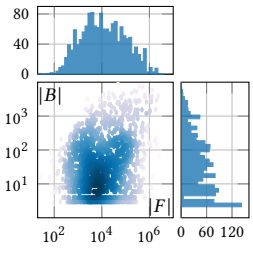


over direct computation. In these cases, approximation methods like FWN are the only viable option for high-performance evaluation.

Errors. Our method is exact in the sense that it introduces no approximations apart from floating-point errors. To demonstrate that these errors are very small in practice, we treat the results obtained from the hierarchical method [Jacobson et al. 2013] as ground truth and report the L^∞ error of our method for each Thingi10K model in Fig. 7 *(right)*. Table 1 shows the maximum and average of the L^∞ and L^1 errors across all models. Here, L^1 error refers to the sum of absolute errors in the GWN value over all query points for a given model, and the L^∞ error is the maximum absolute error over all query points. Since the winding number has a jump discontinuity across the surface by definition, evaluating it in finite precision rather than exact arithmetic introduces numerical instability for near-surface queries. In these regions, it is numerically difficult to robustly determine the orientation of the query point relative to the surface, and the resulting value may vary depending on implementation. To ensure a fair comparison, we exclude query points within a distance of 10^{-14} from the mesh in the statistics for all methods. At these excluded points, the GWN value obtained by different methods may differ by exactly ± 1, which is the height of the jump discontinuity in the GWN field across mesh faces.

We also report errors, measured in the same way, for the FWN method with $\beta = 2$ and $\beta = 15$, both of which we outperform, as shown in Table 1. The L^∞ errors and the maximum L^1 error do not show a clear improvement with a higher accuracy parameter. This could be explained by the criterion FWN uses to determine the near and far fields, which depends on the distance from the query point to the centroid of a BVH cell, rather than to the closest triangle in a cell. As a result, large triangles in the proximity of the query point could be incorrectly categorized as far-field and produce large approximation errors.

Mesh Statistics. To contextualize our performance gains, we analyzed the correlation of the number of faces $|F|$ with that of boundary edges $|B|$ in the Thingi10K dataset. Based on the asymptotic relationship $|B| \sim |F|^{0.5}$ obtained under subdivision, cf. Fig. 6, we expected to find a similar correlation.



Instead, we found that 85% of the meshes have no boundary edges, and the remaining 15% exhibit no strong correlation between face and boundary edge counts, but veer strongly on the side of low boundary complexity. Within these 15% of meshes, whose element counts are plotted in the inset, about 1/3 have less than ten boundary edges, and another 1/3 have between ten and one hundred boundary edges. In relative terms, about 2/3 of meshes within these 15% have at least 100× fewer boundary edges than faces, and 1/4 have between 10× and 100× fewer boundary edges than faces. Only five meshes in total contain more boundary edges than faces.

Although the data is not cleanly correlated, an attempt to fit a power law yields an empirical complexity of $|B| = O(|F|^{0.28})$. This number should not be treated as irrefutable evidence, but it suggests that real-world performance of our algorithm is actually better than the complexity $O(|F|^{0.5})$ measured by the subdivision study in Section 5.2. It also leads us to believe that boundary-centric algorithms are well-suited for winding-number computations on real-world geometries, compared to algorithms impacted by surface discretization.

6 Conclusion

In this paper, we proposed a new algorithm for computing generalized winding numbers on triangle meshes. By combining ray casting with a direct evaluation on a generalized cone, our method produces exact winding numbers while achieving improved performance compared to state-of-the-art approaches across a wide range of test meshes.

The performance benefits demonstrated by our approach depend on a well-connected mesh topology. For highly unstructured geometric data, such as triangle soups, our method provides no efficiency advantage over a naive winding number evaluation.

While the underlying ideas of our algorithm are not inherently restricted to triangle meshes and could, in principle, be extended to more general surface representations such as parametric or implicit surfaces, our current derivation and implementation focus exclusively on meshes. For parametric surfaces, efficient ray tracing algorithms are already available [Reshetov 2019; Xiong et al. 2023], and constructing the corresponding generalized cone is also natural in this setting. However, evaluating the winding number of such a cone remains nontrivial. Extending our framework to support parametric surfaces and other surface representations is an interesting and promising direction for future work.

Acknowledgments

We thank Sadashige Ishida and Ryusuke Sugimoto for their insightful discussions and proofreading and other members of the ISTA Visual Computing Group for their general feedback. This project was funded in part by the European Research Council (ERC Consolidator Grant 101045083 CoDiNA).

References

Gavin Barill, Neil G. Dickson, Ryan Schmidt, David I. W. Levin, and Alec Jacobson. 2018. Fast winding numbers for soups and clouds. *ACM Trans. Graph.* 37, 4, Article 43 (2018), 12 pages.

Yue Chang, Mengfei Liu, Zhecheng Wang, Peter Yichen Chen, and Eitan Grinspun. 2025. Lifting the Winding Number: Precise Discontinuities in Neural Fields for Physics Simulation. In *ACM SIGGRAPH 2025 Conference Papers*. Association for Computing Machinery, New York, NY, USA, 11 pages.

Gianmarco Cherchi, Fabio Pellacini, Marco Attene, and Marco Livesu. 2022. Interactive and Robust Mesh Booleans. *ACM Trans. Graph.* 41, 6, Article 248 (2022), 14 pages.

Olivier Dionne and Martin de Lasa. 2014. Geodesic Binding for Degenerate Character Geometry Using Sparse Voxelization. *IEEE Transactions on Visualization and Computer Graphics* 20, 10 (2014), 1367–1378.

Ana Dodik, Vincent Sitzmann, Justin Solomon, and Oded Stein. 2025. Robust Biharmonic Skinning Using Geometric Fields. *ACM Trans. Graph.* 45, 2, Article 14 (2025), 18 pages.

Yiming Dong, Hongxu Xin, Zhiyang Dou, Rui Xu, Yuan Liu, Shuangmin Chen, Shiqing Xin, Changhe Tu, Taku Komura, and Wenping Wang. 2025. KISSColor: Kinetic and Intuitive Stroke Stretching for Vector Drawing Colorization. *ACM Trans. Graph.* 44, 6, Article 224 (2025), 13 pages.

Simon Duenser, Roi Poranne, Bernhard Thomaszewski, and Stelian Coros. 2020. RoboCut: hot-wire cutting with robot-controlled flexible rods. *ACM Trans. Graph.* 39, 4, Article 98 (2020), 15 pages.

Herbert Edelsbrunner and Ernst Peter Mücke. 1990. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.* 9, 1 (1990), 66–104.

Nicole Feng, Mark Gillespie, and Keenan Crane. 2023. Winding Numbers on Discrete Surfaces. *ACM Trans. Graph.* 42, 4, Article 36 (2023), 17 pages.

Carolus Fridericus Gauss. 1799. *Demonstratio nova theorematis omnem functionem algebraicam.* apvd CG Fleckeisen.

Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen. https://libeigen.gitlab.io.

Peter Heiss-Synak, Aleksei Kalinov, Malina Strugaru, Arian Etemadi, Huidong Yang, and Chris Wojtan. 2024. Multi-Material Mesh-Based Surface Tracking with Implicit Topology Changes. *ACM Trans. Graph.* 43, 4, Article 54 (2024), 14 pages.

Ben Houston, Chris Bond, and Mark Wiebe. 2003. A unified approach for modeling complex occlusions in fluid simulations. In *ACM SIGGRAPH 2003 Sketches & Applications*. Association for Computing Machinery, New York, NY, USA, 1.

Yixin Hu, Teseo Schneider, Bolun Wang, Denis Zorin, and Daniele Panozzo. 2020. Fast tetrahedral meshing in the wild. *ACM Trans. Graph.* 39, 4, Article 117 (2020), 18 pages.

Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. 2013. Robust inside-outside segmentation using generalized winding numbers. *ACM Trans. Graph.* 32, 4, Article 33 (2013), 12 pages.

Alec Jacobson, Daniele Panozzo, et al. 2018. libigl: A simple C++ geometry processing library. https://libigl.github.io/.

Shibo Liu, Ligang Liu, and Xiao-Ming Fu. 2025. Closed-form Generalized Winding Numbers of Rational Parametric Curves for Robust Containment Queries. *ACM Trans. Graph.* 44, 4, Article 75 (2025), 9 pages.

C. Martens and M. Bessmeltsev. 2025. One-Shot Method for Computing Generalized Winding Numbers. *Computer Graphics Forum* 44, 5 (2025), e70194.

Albrecht Ludwig Friedrich Meister. 1769. *Generalia de genesi figurarum planarum et inde pendentibus earum affectionibus.*

August Ferdinand Möbius. 1865. Ueber die bestimmung des inhaltes eines polyëders. *Gesammelte Werke* 2 (1865), 473–512.

Matthias Müller. 2009. Fast and robust tracking of fluid surfaces. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Association for Computing Machinery, New York, NY, USA, 237–245.

F.S. Nooruddin and G. Turk. 2003. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics* 9, 2 (2003), 191–205.

Stefano Nuvoli, Nico Pietroni, Paolo Cignoni, Riccardo Scateni, and Marco Tarini. 2022. SkinMixer: Blending 3D Animated Models. *ACM Trans. Graph.* 41, 6, Article 250 (2022), 15 pages.

Alexander Reshetov. 2019. *Cool Patches: A Geometric Approach to Ray/Bilinear Patch Intersections.* Apress, Berkeley, CA, 95–109.

Jonathan Richard Shewchuk. 1997. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry* 18, 3 (1997), 305–363.

Daniel Scrivener, Ellis Coldren, and Edward Chien. 2024. Winding Number Features for Vector Sketch Colorization. *Computer Graphics Forum* 43, 5 (2024), e15141.

Jacob Spainhour, David Gunderman, and Kenneth Weiss. 2024. Robust Containment Queries over Collections of Rational Parametric Curves via Generalized Winding Numbers. *ACM Trans. Graph.* 43, 4, Article 38 (2024), 14 pages.

Jacob Spainhour and Kenneth Weiss. 2026. Robust Containment Queries over Collections of Trimmed NURBS Surfaces via Generalized Winding Numbers. *ACM Trans. Graph.* (Feb. 2026).

Haoran Sun, Jingkai Wang, Hujun Bao, and Jin Huang. 2024. GauWN: Gaussian-smoothed Winding Number and its Derivatives. In *SIGGRAPH Asia 2024 Conference Papers*. Association for Computing Machinery, New York, NY, USA, Article 89, 9 pages.

A. Van Oosterom and J. Strackee. 1983. The Solid Angle of a Plane Triangle. *IEEE Transactions on Biomedical Engineering* BME-30, 2 (1983), 125–126.

Ingo Wald, Sven Woop, Carsten Benthin, Gregory S. Johnson, and Manfred Ernst. 2014. Embree: a kernel framework for efficient CPU ray tracing. *ACM Trans. Graph.* 33, 4, Article 143 (2014), 8 pages.

Siqi Wang, Janos Meny, Izak Grguric, Mehdi Rahimzadeh, Denis Zorin, Daniele Panozzo, and Hsueh-Ti Derek Liu. 2025a. Solid-Shell Labeling for Discrete Surfaces. In *Proceedings of the SIGGRAPH Asia 2025 Conference Papers (SA Conference Papers '25)*. Association for Computing Machinery, New York, NY, USA, Article 16, 9 pages.

Sinan Wang, Junwei Zhou, Fan Feng, Zhiqi Li, Yuchen Sun, Duowen Chen, Greg Turk, and Bo Zhu. 2025b. Fluid Simulation on Vortex Particle Flow Maps. *ACM Trans. Graph.* 44, 4, Article 91 (2025), 24 pages.

Ruicheng Xiong, Yang Lu, Cong Chen, Jiaming Zhu, Yajun Zeng, and Ligang Liu. 2023. ETER: Elastic Tessellation for Real-Time Pixel-Accurate Rendering of Large-Scale NURBS Models. *ACM Trans. Graph.* 42, 4, Article 133 (2023), 13 pages.

Rui Xu, Zhiyang Dou, Ningna Wang, Shiqing Xin, Shuangmin Chen, Mingyan Jiang, Xiaohu Guo, Wenping Wang, and Changhe Tu. 2023. Globally Consistent Normal Orientation for Point Clouds by Regularizing the Winding-Number Field. *ACM Trans. Graph.* 42, 4, Article 111 (2023), 15 pages.

Qingnan Zhou and Alec Jacobson. 2016. Thingi10K: A Dataset of 10,000 3D-Printing Models. arXiv:1605.04797 [cs.GR] https://arxiv.org/abs/1605.04797