# GauWN: Gaussian-smoothed Winding Number and its Derivatives

HAORAN SUN, State Key Lab of CAD&CG, Zhejiang University, China
JINGKAI WANG, State Key Lab of CAD&CG, Zhejiang University, China and Shanghai Jiao Tong University, China
HUJUN BAO, State Key Lab of CAD&CG, Zhejiang University, China
JIN HUANG*, State Key Lab of CAD&CG, Zhejiang University, China

For a fixed polygon, one can easily determine whether a point is inside or outside it using the winding number. However, deforming a given polygon based on a set of points with expected inside/outside labeling is much more difficult. It asks the winding number to be differentiable with respect to locations of the inside/outside test point and the polygon vertices. We propose a method to address this even for a possibly intersected 2D polygon through Gaussian kernel convolution. Our method can be applied to various problems such as resolving embedding issues (*e.g.*, intersections), editing curves using an in-out brush, and offsetting curves with feature preservation.

It may seem difficult to compute the value and derivatives of this smoothed winding number (GauWN) efficiently, but the cost is only 4 to 6 times that of the vanilla one. To achieve this efficiency, we employ two key strategies: 1) For value computation, we extend the divergence theorem to handle self-intersected cases and transform the convolution into a line integral that can be computed efficiently. 2) For derivatives, we utilize local decomposition to find a line integral form and leverage the radial symmetry and orthogonal separability of the Gaussian kernel. With this differentiable winding number, we can solve the aforementioned problems efficiently by formulating them to involve both the explicit boundary and its implicit field. Surprisingly, there is no need to create a background mesh despite the involvement of an implicit field, making our method easy to apply.

CCS Concepts: • **Computing methodologies** → **Shape modeling**.

Additional Key Words and Phrases: Winding number, geometry processing, differentiable

**ACM Reference Format:**
Haoran Sun, Jingkai Wang, Hujun Bao, and Jin Huang. 2024. GauWN: Gaussian-smoothed Winding Number and its Derivatives. In *SIGGRAPH Asia 2024 Conference Papers (SA Conference Papers '24), December 3–6, 2024, Tokyo, Japan.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3680528.3687569

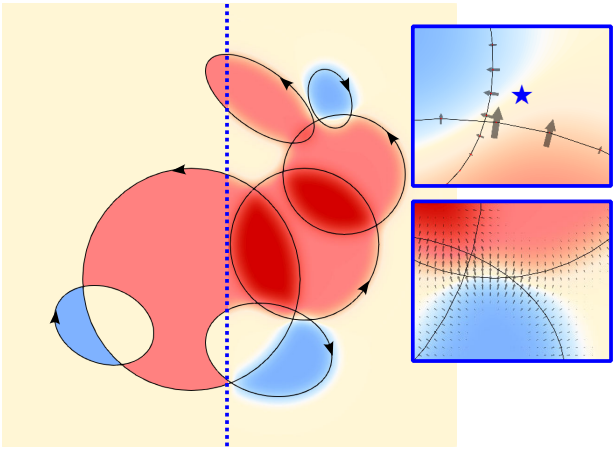


Fig. 1. The winding number (WN) closely connects the boundary and the generalized in-out field (*Left*). We propose GauWN, a differentiable approximation of WN achieved through Gaussian kernel convolution (*Middle*). This allows for easy computation of the gradient of GauWN at a point ★ with respect to the curve vertices (*Right, top*) and spatial locations (*Right, bottom*). With these differentiability properties, GauWN enables expressing the objective in an implicit manner while still using the explicit representation as the degrees of freedom, without the need for a background grid.

## 1 INTRODUCTION

For a solid region in Euclidean space, boundary representations (*i.e.*, explicit representations) and volumetric representations (*i.e.*, implicit representations) have different features and both are widely used. The winding number is well-known for its ability to transform a boundary representation into a field that characterizes the inside/outside labeling, and serves as a classical inside/outside test tool for many applications involving 2D polygons (*e.g.*, rasterization [Doan 2004]).

This feature has been significantly enhanced by extending the winding number definition from 2D to 3D, and to non-closed boundaries [Barill et al. 2018; Jacobson et al. 2013]. All these extensions bring more interesting applications [Jacobson et al. 2013; Xu et al. 2023] to the winding number. For example, one can compute the generalized winding number to approximate the inside/outside indicator field from a problematic boundary with holes and intersections, then complete the boundary and extract the part "inside" this boundary [Hu et al. 2020, 2018], or resolve the intersections [Zhu et al. 2019]. [Feng et al. 2023] even shows that it can work similarly to correct broken and intersected curves on a discrete surface.

In most of the aforementioned boundary correction and completion techniques, the resulting boundary is determined in two sequential stages: implicit field generation (from the boundary) and explicit boundary extraction (from the field). Such a two-stage procedure makes it difficult to control the boundary in its explicit and implicit representations simultaneously. Additionally, it imposes the requirement of creating a suitable background mesh to store

*Corresponding author.

Authors' addresses: Haoran Sun, hrsun@zju.edu.cn, State Key Lab of CAD&CG, Zhejiang University, Hangzhou, Zhejiang, China; Jingkai Wang, ohg@zju.edu.cn, State Key Lab of CAD&CG, Zhejiang University, Hangzhou, Zhejiang, China, and Shanghai Jiao Tong University, Shanghai, China; Hujun Bao, Jin Huang, {bao, hj}@cad.zju.edu.cn, State Key Lab of CAD&CG, Zhejiang University, Hangzhou, Zhejiang, China.


*SA Conference Papers '24, December 3–6, 2024, Tokyo, Japan*

ACM ISBN 979-8-4007-1131-2/24/12
https://doi.org/10.1145/3680528.3687569

the intermediate implicit field, which requires the users to carefully decide the resolution to strike a balance between efficiency and robustness.

It is worth mentioning that the recent work [Xu et al. 2023] is not a two-stage procedure, as it simultaneously optimizes both the winding number field and the explicit boundary (normals at the given points). This work demonstrates the power of optimizing both the explicit representation (*e.g.*, aligning the normal to the pole of the Voronoi cell) and the implicit representation (*e.g.*, achieving balanced winding numbers of 0 and 1 around the boundary). Bringing such a power to other applications is attractive. For example, to resolve self-intersections, we want to: (a) preserve the topology and mesh connectivity, (b) limit shape changes, often through regularizations that penalize deformation, (c) enforce a winding number of 0 at all exterior points and 1 at all interior points. Requirements (a) and (b) are easy to formulate in an explicit representation, while an implicit representation is much more suited for (c). Unfortunately, the winding number field for a closed boundary is piecewise constant, and its derivatives are either trivially zero or infinity. Therefore, it is hard to solve an optimization problem directly formulated with the winding number.

### 1.1 Overview

To address this issue, we propose to apply the Gaussian kernel to the winding number field. We demonstrate that after this simple convolution, the field becomes differentiable *w.r.t.* both the spatial locations and locations of the polygon vertices. This enables a two-way coupling between the winding number field and its associated polygon. More importantly, by utilizing the extendend divergence theorem, local decomposition, and the radial symmetry and orthogonal separability of the Gaussian kernel, we derive an efficient computation form, though the formulation may initially appear challenging to compute efficiently. Unlike many previous works, we do not rely on a background mesh or prescribed points to approximate the winding number field, simplifying its use in various applications that heavily rely on implicit representation. All of these help to formulate problems involving both representations into numerical friendly forms.

The contributions of our work can be summarized as follows:

- Convolving the piecewise constant winding number field with the Gaussian kernel to obtain a smooth field for differentiability,
- deriving efficient numerical methods to compute the values and derivatives of the field with linear complexity (*w.r.t.* the number of polygon vertices), and utilizing spatial hierarchical structures for further acceleration,
- and demonstrating the ability to solve optimization problems involving implicit-field-based objectives without the need for background meshes.

## 2 RELATED WORK

A key aspect of our method is to make the winding number field smooth. While there are techniques available to create smooth implicit fields from polygons, it is challenging to obtain a smooth field that accurately captures the properties of the winding number field and allows for efficient derivative computation.

The signed distance field (SDF) is a candidate for the inside/outside test. However, it has non-smooth points at the medial axis. Besides, it is not well-defined for polygons with embedding issues (*e.g.*, intersection) shown in Section 4.2, but handling such cases is a key advantage of the winding number. Some implicit polygon methods (*e.g.* [Rvachev et al. 2001]) derive a smooth field based on the signed distance field of a contour, but they still cannot be applied to polygons with embedding issues. Moreover, such methods do not allow for efficient derivative computation *w.r.t.* the vertices of the polygon. The same applies to antialiased polygon rasterization methods [Duff 1989; Manson and Schaefer 2011, 2013].

Among all such smooth fields, the regularized double layer potential method [Beale et al. 2016] is very interesting. It is well-known that the winding number can be computed by integrating Green's function along the boundary. Unlike our method that smooths the resulting winding number field directly, this method smooths the Green's function before integration. However, it is unclear how to compute its derivatives efficiently.

It should be noted that the famous Poisson point reconstruction method [Kazhdan et al. 2006] shares a strong similarity with the winding number field. It computes a smooth indicator field (1 for inner and 0 for outer). However, the indicator field is solved numerically on a mesh, while the winding number in Euclidean space is evaluated in a mesh-less fashion.

The concurrent work [Minarcík et al. 2024] demonstrates that many geometric predicates can also be made differentiable through the use of Minkowski penalties. This allows for solving optimization problems for various tasks, such as shape arrangement. Different from [Minarcík et al. 2024] that applies constraints to each object pair, our method relies on a single field contributed by all the objects.

## 3 GAUSSIAN-SMOOTHED WINDING NUMBER

For an oriented closed curve[1] $\Gamma$, its winding number (WN) field $w^\Gamma(\mathbf{x})$ is piecewise constant, and the gradient is zero everywhere except at the curve. This characteristic makes it difficult to use the winding number to guide gradient-based optimization. To address this issue, we propose to smooth the winding number with a Gaussian kernel. The Gaussian-smoothed winding number, GauWN, at $\mathbf{q}$ is defined as

$$W_\sigma^\Gamma(\mathbf{q}) \coloneqq \iint_{\mathbb{R}^2\setminus\Gamma} w^\Gamma(\mathbf{x}) G(\mathbf{x}-\mathbf{q};\sigma^2\mathbf{I})\,\mathrm{d}S, \tag{1}$$

where $G(\mathbf{x};\sigma^2\mathbf{I}) = g(\mathbf{x}.x;\sigma^2)g(\mathbf{x}.y;\sigma^2)$, and $g(x;\sigma^2)$ is the 1D Gaussian kernel with mean 0 and variance $\sigma^2$ evaluated at $x$.

The naïve way to compute Equation (1) is to construct a grid and perform the convolution with the Gaussian kernel. However, even though WN itself is fast enough with BVH, this method would still be too slow and inaccurate (see Figure 3 in the supplementary material). Fortunately, there are special techniques to efficiently compute both Equation (1) and its gradient if the curve is piecewise linear.

[1] In this paper, the term "curve" actually refers to a "multi-curve", *i.e.*, an immersion of one or more disjoint circles [Chang and Erickson 2017].

### 3.1 Value of $W_\sigma^\Gamma$

To calculate the convolution over a domain quickly and accurately, we adopt the idea of previous works like [Manson and Schaefer 2011, 2013], which transform the area integrals into line integrals. To handle closed boundaries with possible self-intersections, we apply the following extendend divergence theorem:

Lemma 3.1. *For a closed 2D piecewise smooth curve (possibly with self-intersections) $\Gamma$ and a smooth vector field $\mathbf{F}$ on the plane, we have*

$$\iint_{\mathbb{R}^2\setminus\Gamma} w^\Gamma(\mathbf{x})\nabla\cdot\mathbf{F}\,\mathrm{d}S = \oint_\Gamma \langle\mathbf{F},\mathbf{n}\rangle\,\mathrm{d}s, \tag{2}$$

*where $\mathbf{n}$ is the unit normal vector of $\Gamma$.*

Please refer to Section A of the supplementary material for the proof. Lemma 3.1 makes it possible to compute the convolution *w.r.t.* the winding number simply by computing the line integral along the curve, without bothering to perform geometric intersection explicitly.

It is not hard to verify

$$\mathbf{F}_\mathbf{q}(\mathbf{x}) = [g(\mathbf{x}.y-\mathbf{q}.y;\sigma^2)\Phi(\mathbf{x}.x-\mathbf{q}.x;\sigma^2),0]^\top, \tag{3}$$

where $\Phi(x;\sigma^2)$ is the 1D Gaussian cumulative distribution with mean 0 and variance $\sigma^2$, satisfies

$$\nabla\cdot\mathbf{F}_\mathbf{q}(\mathbf{x}) = g(\mathbf{x}.x-\mathbf{q}.x;\sigma^2)g(\mathbf{x}.y-\mathbf{q}.y;\sigma^2) = G(\mathbf{x}-\mathbf{q};\sigma^2\mathbf{I}). \tag{4}$$

For a piecewise linear curve, $\oint_\Gamma \langle\mathbf{F}_\mathbf{q},\mathbf{n}\rangle\,\mathrm{d}s$ has a closed form in terms of the bivariate normal cumulative distribution function that can be computed efficiently:

$$\oint_\Gamma \langle\mathbf{F}_\mathbf{q},\mathbf{n}\rangle\,\mathrm{d}s = \sum_{\gamma=\overline{\mathbf{v}_i\mathbf{v}_j}} \Delta y\,\mathrm{O}''_\sigma\big(0,1,\mathbf{v}_i.y-\mathbf{v}_j.y,\mathbf{v}_j.y,\ \mathbf{v}_i.x-\mathbf{v}_j.x,\mathbf{v}_j.x,\mathbf{q}.y,\mathbf{q}.x\big). \tag{5}$$

The $\mathrm{O}''_\sigma$ is defined as follows, and details are provided in Section B of the supplementary material:

$$\begin{aligned}
&\mathrm{O}(Y,a,b) \coloneqq \mathrm{BvN}\left(\frac{a}{\sqrt{1+b^2}},Y;\frac{-b}{\sqrt{1+b^2}}\right),\\
&\mathrm{O}_\sigma(Y,a,b,\mu_1,\mu_2) \coloneqq \mathrm{O}\left(\frac{Y-\mu_1}{\sigma},\frac{a+b\mu_1-\mu_2}{\sigma},b\right),\\
&\mathrm{O}'_\sigma(h,k,a,b,\mu_1,\mu_2) \coloneqq \mathrm{O}_\sigma(k,a,b,\mu_1,\mu_2)-\mathrm{O}_\sigma(h,a,b,\mu_1,\mu_2),\\
&\mathrm{O}''_\sigma(h,k,a,b,e,f,\mu_1,\mu_2) \coloneqq \frac{1}{a}\mathrm{O}'_\sigma\left(ah+b,ak+b,f-\frac{be}{a},\frac{e}{a},\mu_1,\mu_2\right),
\end{aligned} \tag{6}$$

where BvN is the bivariate normal cumulative distribution function that can be computed efficiently with [Tsay and Ke 2023].

### 3.2 Gradient of $W_\sigma^\Gamma$

When $\sigma$ and the topology of $\Gamma$ are fixed, $W_\sigma^\Gamma(\mathbf{q})$ is a function of the vertices $\mathbf{v}_i$ of $\Gamma$ and $\mathbf{q}$, and is therefore denoted as $W_\sigma^\Gamma(\mathbf{q};\mathbf{v}_1,\cdots,\mathbf{v}_n)$. This implies that there are two types of gradients of $W_\sigma^\Gamma$: the gradient *w.r.t.* the vertices $\mathbf{v}_i$ and the gradient *w.r.t.* the spatial locations $\mathbf{q}$.

It is easy to see that the winding number is invariant under translation, *i.e.*, for any $\Delta\in\mathbb{R}^2$, $W_\sigma^\Gamma(\mathbf{q}+\Delta;\mathbf{v}_1,\cdots,\mathbf{v}_n) = W_\sigma^\Gamma(\mathbf{q};\mathbf{v}_1-\Delta,\cdots,\mathbf{v}_n-\Delta)$. Taking the derivative *w.r.t.* $\Delta$ on both sides, we get

$$\frac{\partial W_\sigma^\Gamma}{\partial\mathbf{q}} = -\sum_{i=1}^n \frac{\partial W_\sigma^\Gamma}{\partial\mathbf{v}_i} =: \nabla W_\sigma^\Gamma. \tag{7}$$

Therefore, we can focus on the gradient $\partial W_\sigma^\Gamma/\partial\mathbf{v}_i$.

For any single vertex $\mathbf{v}_i$, we have

$$\begin{aligned}
\frac{\partial W_\sigma^\Gamma}{\partial\mathbf{v}_i} &= \frac{\partial}{\partial\mathbf{v}_i}\iint_{\mathbb{R}^2\setminus\Gamma} w^\Gamma(\mathbf{x})G(\mathbf{x}-\mathbf{q};\sigma^2\mathbf{I})\,\mathrm{d}S\\
&= \iint_{\mathbb{R}^2\setminus\Gamma} G(\mathbf{x}-\mathbf{q};\sigma^2\mathbf{I})\frac{\partial}{\partial\mathbf{v}_i}w^\Gamma(\mathbf{x})\,\mathrm{d}S.
\end{aligned} \tag{8}$$

The challenge lies in computing the integration in Equation 8 due to the presence of the gradients of the winding number, which has singular behavior. However, we can use a local decomposition technique to handle this integration.

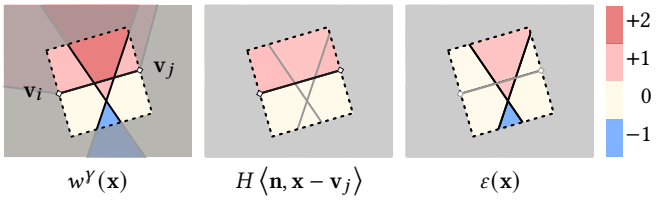



Fig. 2. In a neighborhood $U(\gamma)$ of the edge $\gamma=\overline{\mathbf{v}_i\mathbf{v}_j}$, the winding number can be decomposed into a step function $H$ jumping at $\gamma$ and contributions $\varepsilon$ from other vertices. Black lines are the edges involved in the functions, while the gray lines are the other edges of $\Gamma$ but not involved.

For any edge $\gamma=\overline{\mathbf{v}_i\mathbf{v}_j}$ of $\Gamma$, we notice that there exist some neighborhoods $U(\gamma)$, where we can locally decompose the winding number as

$$w^\gamma(\mathbf{x}) = H\langle\mathbf{n},\mathbf{x}-\mathbf{v}_j\rangle + \varepsilon(\mathbf{x}), \tag{9}$$

where $H$ is the Heaviside step function whose derivative is the Dirac delta function $\delta$, and $\varepsilon(\mathbf{x})$ is some other term that is not related to $\mathbf{v}_i$ (see Figure 2 for an illustration). Therefore, within this neighborhood, we have

$$\begin{aligned}
\frac{\partial}{\partial\mathbf{v}_i}w^\gamma(\mathbf{x}) &= \frac{\partial}{\partial\mathbf{v}_i}H\langle\mathbf{n},\mathbf{x}-\mathbf{v}_j\rangle\\
&= \delta\langle\mathbf{n},\mathbf{x}-\mathbf{v}_j\rangle\frac{\partial}{\partial\mathbf{v}_i}\langle\mathbf{n},\mathbf{x}-\mathbf{v}_j\rangle.
\end{aligned} \tag{10}$$

The Dirac delta and the auxiliary $U$ can be eliminated using its composition property [Hörmander 2003, Theorem 6.1.5], *i.e.*,

$$\int_U f(\mathbf{x})\delta(h(\mathbf{x}))\,\mathrm{d}S = \int_{\mathbb{R}^n} f(\mathbf{x})\delta(h(\mathbf{x}))\,\mathrm{d}S = \int_{h^{-1}(0)}\frac{f(\mathbf{x})}{|\nabla h|}\,\mathrm{d}s, \tag{11}$$

where $h^{-1}(0)\subset U$.

Again, the calculation of the gradient becomes a line integral along the edges:

$$\begin{aligned}\frac{\partial}{\partial \mathbf{v}_i} W_\sigma^\Gamma &= \sum_{\gamma=\overline{\mathbf{v}_i\mathbf{v}_j}} \int_\gamma G(\mathbf{x}-\mathbf{q};\sigma^2\mathbf{I}) \frac{\partial/\partial \mathbf{v}_i \left\langle \mathbf{n}, \mathbf{x}-\mathbf{v}_j \right\rangle}{\left|\partial/\partial \mathbf{x} \left\langle \mathbf{n}, \mathbf{x}-\mathbf{v}_j \right\rangle\right|} \,\mathrm{d}s \\ &= \sum_{\gamma=\overline{\mathbf{v}_i\mathbf{v}_j}} -g(d;\sigma^2)/l \Big[\sigma^2 g(-\mu;\sigma^2) - \sigma^2 g(l-\mu;\sigma^2) \\ &\qquad + \mu\Phi(l-\mu;\sigma^2) - \mu\Phi(-\mu;\sigma^2)\Big]\mathbf{n}.\end{aligned} \tag{12}$$

The details can be found in Section C of the supplementary material.

### 3.3 Acceleration with bounding volume hierarchy

When computing $W_\sigma^\Gamma(\mathbf{q})$ or its gradient, it is unnecessary to keep all the fine details of the entire curve $\Gamma$, as the parts far away from $\mathbf{q}$ have little contribution to the result. Similar to previous work on the winding number [Barill et al. 2018], we can utilize a bounding volume hierarchy (BVH) to accelerate the computation by clustering distant complex elements into simpler ones. Although the winding number can be generalized to incomplete curves and point clouds (*e.g.* [Jacobson et al. 2013]), we only consider the closed situation, as indicated by the usage of Lemma 3.1. It is important to note that the approximated integral must also be a closed curve integral to avoid significant bias, even when the clustering occurs far away (see Figure 3).

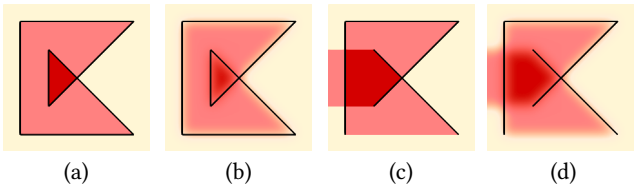
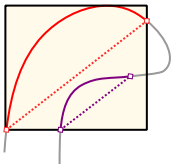



Fig. 3. GauWN (b) is the result of smoothing the winding number field (a) by the Gaussian kernel. When the curve is not closed, there will be significant bias. This behavior is invariant to $\sigma$ (c): $\sigma = 10^{-6}$,(d): $\sigma = 0.05$.

We employ a special clustering method to meet this requirement. We use a standard AABB tree for the hierarchy and store the two ends of each connected component in the nodes. Then, a segment connecting the two ends is used to approximate each connected component (see the inset). By leveraging the properties of the AABB tree, we ensure that each end is a vertex of the original curve $\Gamma$ and is also an end in the nearby AABB node. Consequently, the approximation curve remains closed (see Figure 4).



The gradient case is simpler. Due to the fast-decay property of the Gaussian kernel, when computing the gradient *w.r.t.* the spatial locations, we can simply discard the contributions from nodes that are sufficiently far away. Similarly, when computing the gradient of GauWN at a point $\mathbf{q}$ *w.r.t.* locations of the vertices, we only need to consider the vertices in the nearby nodes. The details are provided as pseudocode in the supplementary material.

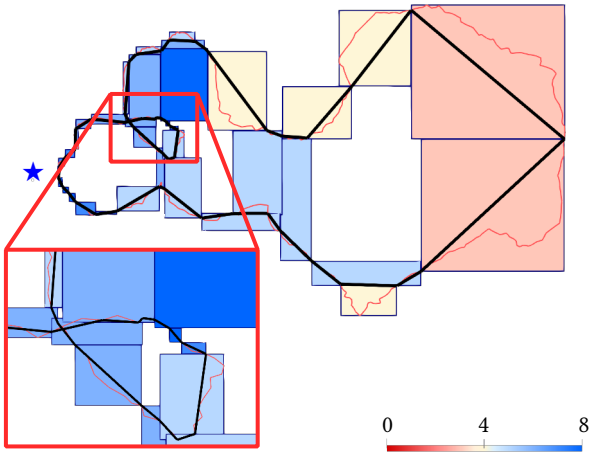



Fig. 4. When using the AABB tree to evaluate the GauWN of the red thin curve at ★ , it is equivalent to evaluating the GauWN of the black thick curve, which is a closed curve approximated using the AABB tree. The colors indicate the depth of each AABB node.

## 4 RESULTS AND APPLICATIONS

In this section, we will demonstrate the performance of GauWN, showcase some applications of GauWN, and discuss its behaviors. All experiments were conducted on a desktop computer with an AMD Ryzen™ 9 5900X processor running at 4.4GHz. The input curves were normalized to have bounding boxes with unit length along their major axis. The core code of GauWN is available at https://github.com/ZJUCADGeoSim/GauWN2D.

### 4.1 Performance

To measure the performance, we built a $100 \times 100$ grid on the aforementioned bounding box and evaluated the values and gradients on the 10000 grid nodes using 20 threads for parallelization. For better visualization in the plots, we fit piecewise linear curves to the scatter points. The curves have 4 uniformly distributed nodes, and the sum of squared time differences to the points is minimized.

*Complexity.* The time complexity of WN is $O(\|L\|)$ without BVH and $O(\log \|L\|)$ with BVH, where $\|L\|$ is the number of segments. GauWN has a similar complexity but with some differences because BvN has no closed form and requires numerical computation. To experimentally demonstrate the complexity of GauWN, we started with a curve with 180 vertices and resample it to have up to 18000 vertices. Figure 5 shows that without BVH, although slower than WN, the time cost of GauWN and its derivatives generally increases linearly with the number of vertices.

*Speed comparison on the dataset.* In this experiment, we used the boundary polygons of 426 parameterized 2D shapes from [Li et al. 2018] as input.

From Figures 6 and 7, we can observe that the influence of $\sigma$ is not obvious when BVH is applied. Indeed, $\sigma$ has little impact on the

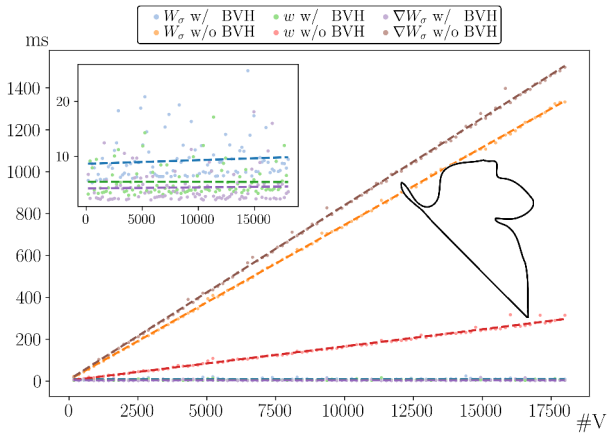



Fig. 5. Time cost of 10000 evaluations of WN, GauWN, and the spatial gradient of GauWN on the same curve up-sampled to different resolutions. *Left-top*: zoom-in for time w/ BVH.

performance if no BVH is used as well. Therefore, we only show the performance of GauWN with $\sigma = 0.01$ for other comparisons.

In this dataset, without BVH, evaluating the value of GauWN is about 3 ∼ 4 times slower than WN, but it is less than 2 times slower after applying BVH. Since the gradient *w.r.t.* the spatial locations and the gradient *w.r.t.* locations of the vertices share the same underlying form, we only exhibit the performance of the spatial gradient (*i.e.*, $\nabla W_\sigma^\Gamma$). As explained earlier, spatial gradient evaluation can ignore the contribution from distant vertices, so it is faster than value evaluation if BVH is applied.

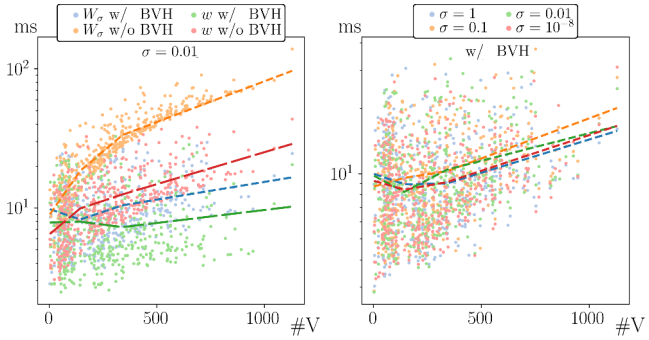



Fig. 6. Time cost of 10000 evaluations of GauWN and WN on each model in the dataset. *Left*: GauWN is slower than WN but is less than two times slower when BVH is applied. *Right*: The performance of GauWN depends on the parameter $\sigma$, but the differences are small.

## 4.2 Embedding issue resolving

Preserving the topology of the input curve and resolving its embedding issues (*e.g.*, intersections) is an important application of GauWN. The approach involves sampling points in the domain as test points

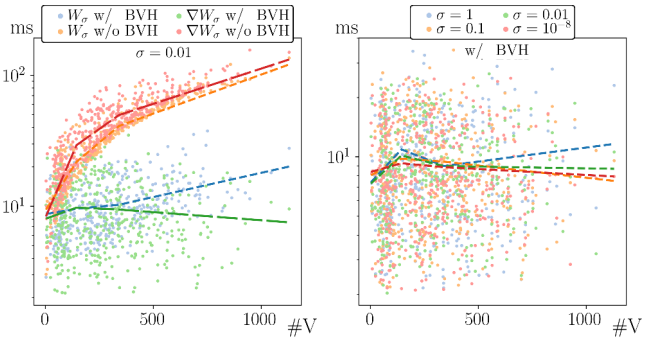
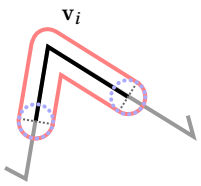



Fig. 7. Time cost of 10000 evaluations of the spatial gradient of GauWN. *Left*: The gradient evaluation benefits more from BVH acceleration than the value evaluation. *Right*: $\sigma$ has little influence as well.

and using the differences in WN to identify inconsistencies between the current embedding and the desired objective.




It is easy to see that if a curve $\Gamma$ has no embedding issues, WN in a small neighborhood of the curve must be consistent with the intrinsic topology. In other words, the winding number $w^\Gamma$ should be 0 on its right side and 1 on its left side respectively. As illustrated in the inset, we sample some test points $S = \{\mathbf{x}\}$ on the locus of points with a small distance $\varepsilon$ to the dual region $\angle_i$ of each vertex $\mathbf{v}$ (*i.e.*, the near halves of adjacent edges of $\mathbf{v}_i$), *i.e.*,

$$S \subset \left\{\mathbf{y} \in \mathbb{R}^2 \;\middle|\; \min_{\mathbf{p}\in\angle_i} \|\mathbf{y}-\mathbf{p}\| = \varepsilon\right\} \setminus \left\{\mathbf{y} \in \mathbb{R}^2 \;\middle|\; \min_{\mathbf{p}\in\partial\angle_i} \|\mathbf{y}-\mathbf{p}\| = \varepsilon\right\}. \tag{13}$$

If a test point $\mathbf{x}$ is sampled on the right side, we assign $\hat{w}^\Gamma(\mathbf{x}) = 0$ as the expected winding number, and vice versa. To *detect* the inconsistencies, it is sufficient to compare $\hat{w}^\Gamma$ with $w^\Gamma$ for the points $\mathbf{x} \in S$. However, with GauWN, we can *penalize* the inconsistencies using the following differentiable objective function:

$$E_D(\Gamma, \sigma) \coloneqq \sum_{\mathbf{x}\in S} \left(W_\sigma^\Gamma(\mathbf{x}) - \hat{w}^\Gamma(\mathbf{x})\right)^2. \tag{14}$$

Its gradient *w.r.t.* a single vertex $\mathbf{v}_i$ of $\Gamma$ is:

$$\frac{\partial E_D(\Gamma,\sigma)}{\partial \mathbf{v}_i} = 2\sum_{\mathbf{x}\in S} \left(W_\sigma^\Gamma(\mathbf{x}) - \hat{w}^\Gamma(\mathbf{x})\right) \frac{\partial W_\sigma^\Gamma(\mathbf{x})}{\partial \mathbf{v}_i}. \tag{15}$$

Because the test point $\mathbf{x}$ depends on the shape of $\Gamma$, it may move if $\mathbf{v}_i$ moves. To consider the partial derivatives of $\mathbf{x}$ *w.r.t.* $\mathbf{v}_i$, we denote $\mathbf{x}$ as $\mathbf{x}(\mathbf{v}_i)$ and $W_\sigma^\Gamma(\mathbf{x})$ as $W(\mathbf{v}_i, \mathbf{x}(\mathbf{v}_i))$ in the following for convenience. Then, we have:

$$\begin{aligned}\frac{\partial W_\sigma^\Gamma(\mathbf{x})}{\partial \mathbf{v}_i} &= \frac{\mathrm{d}}{\mathrm{d}\mathbf{v}_i} W(\mathbf{v}_i, \mathbf{x}(\mathbf{v}_i)) \\ &= \frac{\partial}{\partial \mathbf{v}_i} W(\mathbf{v}, \mathbf{x}(\mathbf{v}_i)) + \left(\frac{\partial \mathbf{x}}{\partial \mathbf{v}_i}\right)^\top \frac{\partial}{\partial \mathbf{x}} W(\mathbf{v}_i, \mathbf{x}(\mathbf{v}_i)).\end{aligned} \tag{16}$$

Along with the regularization term $E_R(\Gamma) = \sum_{\mathbf{v}_i\in\Gamma} \|\mathbf{v}_i - \mathbf{v}_i^{(0)}\|^2$, we solve the following variational problem to resolve the embedding

issues of $\Gamma$:

$$\min_{\Gamma} E_D(\Gamma, \sigma) + \lambda E_R(\Gamma). \tag{17}$$

Other types of regularization can also be used. Since $\lim_{\sigma \to 0} W_\sigma^\Gamma(\mathbf{x}) = w^\Gamma(\mathbf{x})$, it is natural to end the optimization with a small $\sigma$, similar to the approach in [Poranne et al. 2017]. Among many possible ways to reduce $\sigma$, we adopt a simple strategy: starting with a relatively large $\sigma$ (0.01) and gradually decreasing it by 5% in each iteration (see Algorithm 1). We use a gradient-based method CCSAQ [Svanberg 2002] from NLopt [Johnson 2007] to solve Equation (17) in each iteration.

**ALGORITHM 1:** Embedding issue resolving algorithm

```
Input  : Γ^(0);                          // the initial polygon
Input  : σ_0 ;                  // the initial standard deviation
Input  : k_max ;                         // the maximum iterations
k ← 1, σ ← σ_0;
while k ≤ k_max do
    Γ^(k) ← arg min_Γ E_D(Γ, σ) + λE_R(Γ);
    σ ← σ × 0.95;
    k ← k + 1;
end
```

As mentioned in Section 1, both explicit and implicit representations have their own advantages. In Equation (17), it is clear to see that $E_D$ is related to the implicit field and $E_R$ is related to the explicit curve. The advantages of both representations can be easily combined in one optimization problem via GauWN.

Unlike existing methods for resolving self-intersections, using GauWN to resolve embedding issues does not require an initial interior tessellation [Chen et al. 2023] and does not alter the input topology [Li and Barbič 2018]. For example, when triangulating a CAD patch, initial intersections of the boundary curves may occur due to insufficient sampling density or defects in the CAD model itself, making it challenging to use constrained Delaunay triangulation. Figure 8 shows an example of using GauWN to resolve different types of embedding issues, including cases where it fails. It is important to note that GauWN is primarily suitable for addressing local embedding issues, as its gradient can only reflect the local information of the embedding.

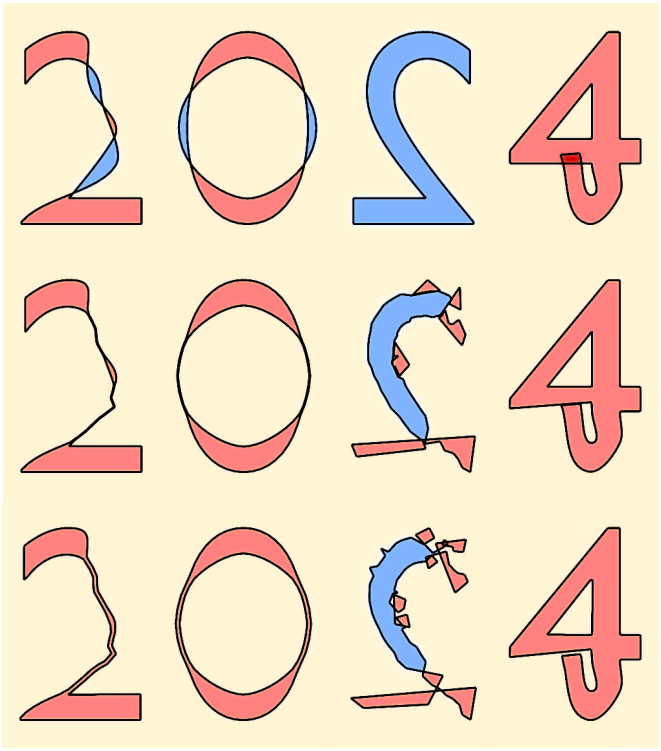
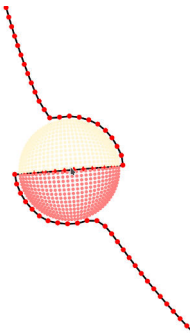


Fig. 8. There may be various types of embedding issues, *e.g.*, the *first "2"* has local flipping and intersections, the *"0"* has intersections between different curves, the *second "2"* has global flipping but no intersections, and the *"4"* has self-intersections but no flipping. GauWN is suitable for issues that can be resolved locally. *Top*: the input curves. *Middle*: the results with the sampling distance $\varepsilon = 0.004$. *Bottom*: the results with sampling distance $\varepsilon = 0.02$.

## 4.3 Curve modeling

GauWN can be used to (interactively) model curves. This application shares a similar idea with resolving embedding issues: driving the curve by the inconsistencies between the GauWN values and the user-specified winding numbers at the test points. The difference is that the test points are not sampled on the two sides of the curve, but provided by the user via an in-out brush consisting of points $\mathbf{x}$ with expected $\hat{w}(\mathbf{x})$. The optimization problem is still Equation (17), but $\partial \mathbf{x} / \partial \mathbf{v}_i$ is zero in this case, and only the gradient *w.r.t.* locations of the vertices is used.



After each edit, the curve is re-sampled (optionally) to maintain appropriate vertex density. Figure 10 shows an example curve modeled by interactively moving the brush. More demonstrations can be found in the supplementary video. One can try to achieve this by simply tracing the isoline of the field modified by the brush, but such a method usually requires a background mesh and may change the topology unexpectedly.

## 4.4 Curve offsetting by a flow

GauWN can also be used in a non-variational manner, for example, to define a flow. As shown in Section 4.5, the spatial gradient of GauWN near the curve behaves like a smoothed gradient of SDF. By optimizing a vertex $\mathbf{v}_i$ to have a larger/smaller value of GauWN, it naturally offsets it inwards/outwards. This can be described as the following optimization problem:

$$\begin{aligned}\left[\mathbf{v}_1^{(k)}, \cdots, \mathbf{v}_n^{(k)}\right] = \underset{[\mathbf{v}_1, \cdots, \mathbf{v}_n]}{\arg\min}\ & \lambda \sum_{i=1}^{n} \left\| \mathbf{v}_i - \mathbf{v}_i^{(k-1)} \right\|^2 \\ & + \sum_{i=1}^{n} \left( W_\sigma^{\Gamma^{(k-1)}}(\mathbf{v}_i) - W_\sigma^{\Gamma^{(k-1)}}(\mathbf{v}_i^{(k-1)}) - o \right)^2,\end{aligned} \tag{18}$$

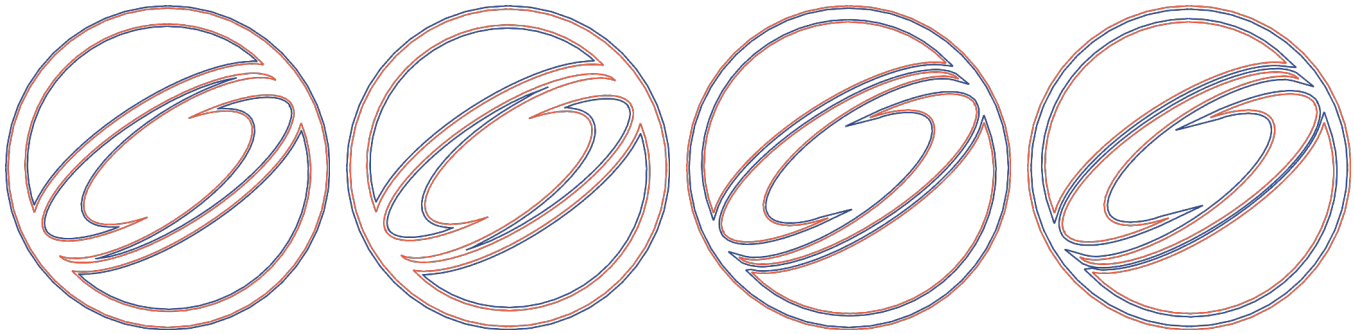


Fig. 9. Offsetting using GauWN. The red curve is the input curve and the blue curve is the offset curve. The sharp feature can be preserved in two offsetting directions.

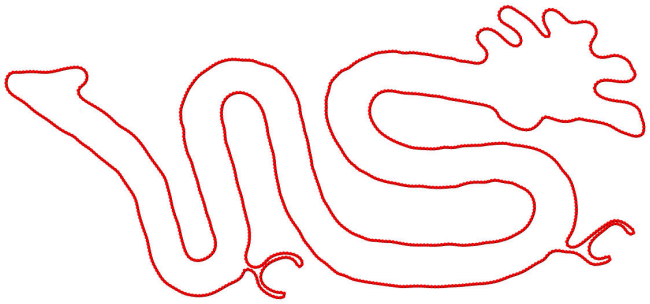


Fig. 10. A "Chinese dragon" curve modeled with GauWN. Please refer to the supplementary video for the modeling procedure.

where $o$ is the offset of GauWN for each iteration, and $\lambda$ is the regularization weight. This offsetting method can keep the one-to-one mapping between the original curve and the offset curve, without requiring spatial discretization like methods that rely on the distance field (*e.g.*, [Chen et al. 2020]). Figure 9 illustrates the inward and outward offsetting results of the SIGGRAPH logo with different distances. The sharp features can be preserved when the offsetting distance is not excessively large.

For large offsetting distances, although there is no well-defined ground-truth, our method can still be applied (Figure 11). However, when the offset curve comes close to touching itself, applying Equation (18) may result in intersections or unpredictable outcomes. To mitigate this issue, it is recommended to incorporate the energy for resolving embedding issues discussed in Section 4.2 and adjust $o$ adaptively (*e.g.*, setting $o$ to $0.1o$ for vertices with a moving distance from the input larger than 1.5 times the average moving distance). The next subsection provides further discussion on this scenario.

### 4.5 Discussion on behaviors of GauWN

The behavior of GauWN can be intuitively understood as applying Gaussian blur to WN. By the nature of convex combination, it is easy to see that $\min w^\Gamma \leq \min W_\sigma^\Gamma \leq \max W_\sigma^\Gamma \leq \max w^\Gamma$. As shown in Figure 12, when $\sigma$ is very large, GauWN becomes over-smoothed and approaches a zero constant function, and becomes not useful.

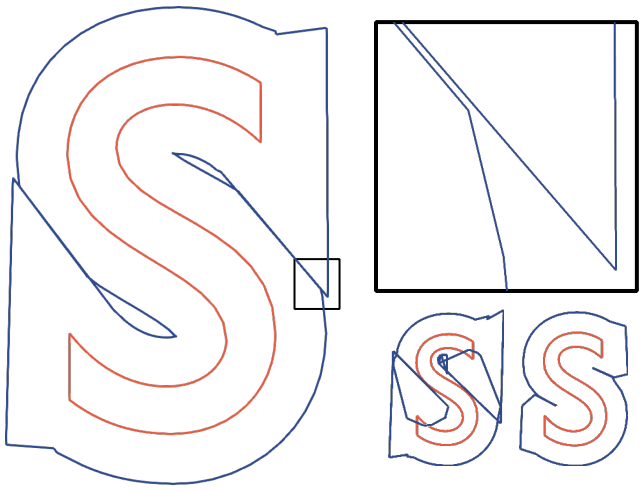


Fig. 11. One can perform a large offset (*left*) using the flow defined in Equation (18) with parameters $o = 0.2, k = 50, \sigma = 0.005$. The offset curve may have self-intersections (*bottom middle*) when the offset is too large ($k = 70$). With the energy for embedding issue resolving in Section 4.2 and adaptive $o$, this issue can be largely resolved (*bottom right*).

Thus, we only consider relatively small $\sigma$, and will approximately assume that the support of the kernel $G(\mathbf{x} - \mathbf{q}; \sigma^2\mathbf{I})$ is just a small disk $N(\mathbf{q}, 3\sigma) \subset \mathbb{R}^2$ centered at $\mathbf{q}$ with a radius of $3\sigma$ to simplify the following discussion.

If the disk $N(\mathbf{q}, 3\sigma), \mathbf{q} \in \Gamma$ only includes a single line segment in $\Gamma$, considering the fact that $N$ is subdivided into two parts with WN of $w, w+1$ respectively, we will have $W_\sigma^\Gamma(\mathbf{q}) \approx w + 0.5, \forall \mathbf{q} \in \Gamma$ approximately. In other words, moving $\mathbf{q}$ along the single segment will not change $W_\sigma^\Gamma$. Therefore, the spatial gradient will be orthogonal to the segment approximately.

At sharp corners (see Figure 13), the 0.5-contour will drift away from $\mathbf{q}$, and the spatial gradient may not align with the normal of the polygon. Intuitively speaking, GauWN fillets sharp corners. In-out testing may be wrong around the sharp corner, and this issue can be alleviated by taking a smaller $\sigma$ (see the strategy shown in Algorithm 1). Such behavior is different from SDF and implicit

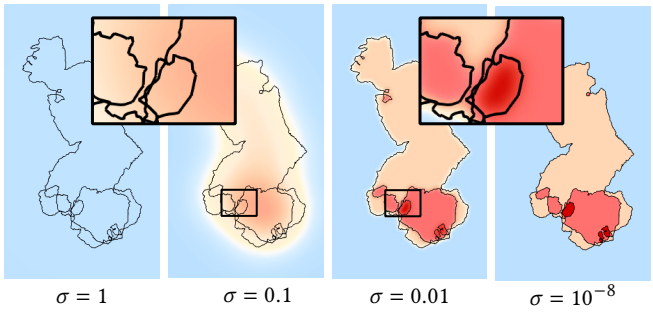



Fig. 12. When $\sigma$ is as small as $10^{-8}$, the result of GauWN is very close to that of WN. As we increase $\sigma$, the result of GauWN becomes smoother, offering a useful gradient for optimization. However, if we continue to increase $\sigma$, GauWN will further smooth out features, and the result will eventually become an almost zero constant function.

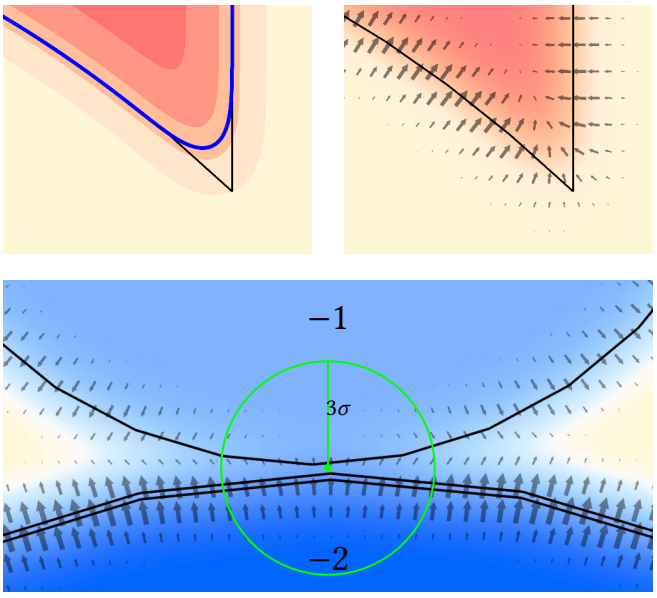



Fig. 13. *Top*: The contour and spatial gradient around a sharp corner. *Bottom*: Due to the smoothing, the gradient may have opposite directions compared to the normal of the boundary curve with a small local feature size.

polygon [Rvachev et al. 2001], which force a contour to exactly align with the sharp corner. However, precisely aligning the sharp corners makes the field non-differentiable at corners. In other words, filleting the corners is an inevitable behavior for differentiability. This behavior also implies that the polygon tracing from a contour of a differentiable implicit field cannot have sharp corners, and our strategy of deforming an explicit representation is more attractive if sharp features are required (*e.g.*, in the offset application).

If the disk covers parts with a small local feature size, GauWN may significantly differ from WN due to over-smoothing. As illustrated in the bottom of Figure 13, four zones with winding numbers $-1, 0, -1, -2$ in top-down order are covered by a disk. The spatial gradient is opposite to the outward normal for some points on the top curve. Reducing $\sigma$ weakens such a phenomenon, but the field will be nearly non-differentiable.

## 5 CONCLUSION

In this paper, we have shown that the winding number can be extended to be differentiable with fast numerical computation. This extension endows the ability of efficient two-way coupling between the explicit representation and implicit representation. As shown in our example applications, such an ability makes some difficult problems much easier to formulate and solve, especially considering that no background mesh is required.

The most obvious limitation of our work is that we only achieve an efficient method in 2D. The high-level idea of smoothing a WN field via a Gaussian kernel can be generalized to 3D without difficulty, but efficiency becomes a significant challenge. In our preliminary experiments, using a simple numerical quadrature for the integration of the 3D Gaussian kernel did not yield satisfactory performance (see Figure 14).

The derivatives are near zero far away from the boundary. This behavior arises from the nature of Gaussian kernel convolution, which is suitable for smoothly deforming the current boundary. However, it can cause issues if one wants the explicit representation to respond to distant changes in the implicit field. For instance, in the curve modeling application, if the user places the in-out brush far away from the current curve, the curve will remain unchanged, and the optimization will get stuck in the current local minimum.

The applications provided in this paper serve as illustrations and can be further improved individually. For example, a better sampling strategy may help to avoid missing detection of embedding issues, and various kinds of regularization may be applied to achieve desired behavior. Additionally, more advanced numerical methods and carefully designed optimization schemes may enhance their performance. Furthermore, integrating our extension of the winding number to other geometries, such as incomplete boundaries and curved manifolds, is also an interesting area for future work.

## ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their comments and suggestions, especially the Shepherd reviewer who offered great help in improving the language of the paper. We also thank the SIGGRAPH English Review Service volunteers. This work was supported in part by a few friendly companies.

## 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 (jul 2018), 12 pages. https://doi.org/10.1145/3197517.3201337

J. Thomas Beale, Wenjun Ying, and Jason R. Wilson. 2016. A Simple Method for Computing Singular or Nearly Singular Integrals on Closed Surfaces. *Communications in Computational Physics* 20, 3 (2016), 733–753. https://doi.org/10.4208/cicp.030815.240216a

Hsien-Chih Chang and Jeff Erickson. 2017. Untangling Planar Curves. *Discrete Comput. Geom.* 58, 4 (Dec. 2017), 889–920. https://doi.org/10.1007/s00454-017-9907-6

He Chen, Elie Diaz, and Cem Yuksel. 2023. Shortest Path to Boundary for Self-Intersecting Meshes. *ACM Trans. Graph.* 42, 4, Article 146 (jul 2023), 15 pages. https://doi.org/10.1145/3592136

Zhen Chen, Daniele Panozzo, and Jérémie Dumas. 2020. Half-Space Power Diagrams and Discrete Surface Offsets. *IEEE Transactions on Visualization and Computer*

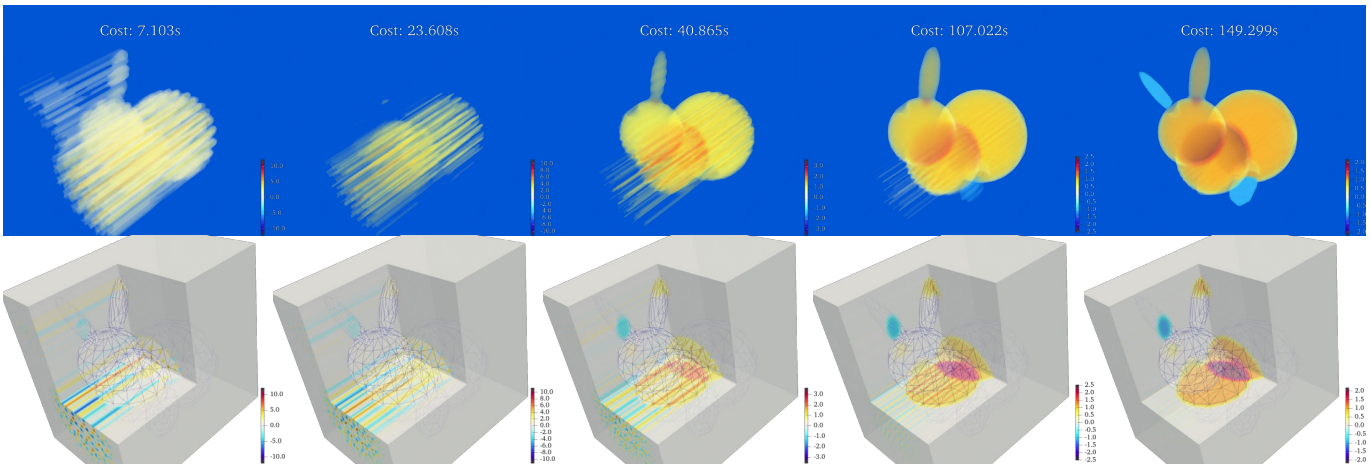



Fig. 14. Results of computing 3D version of the boundary integration using quadrature. Top: volumetric view; Bottom: clip view. From left to right, there are 1, 4, 7, 19, 27 quadrature points per triangle. There are $100 \times 100 \times 100$ queries and the mesh contains 1584 faces.

*Graphics* 26, 10 (2020), 2970–2981. https://doi.org/10.1109/TVCG.2019.2945961

Khanh P. V. Doan. 2004. Antialiased Rendering of Self-Intersecting Polygons using Polygon Decomposition. In *12th Pacific Conference on Computer Graphics and Applications, PG 2004, Seoul, South Korea, October 6-8, 2004*. IEEE Computer Society, 383–391. https://doi.org/10.1109/PCCGA.2004.1348369

Tom Duff. 1989. Polygon scan conversion by exact convolution.. In *International Conference On Raster Imaging and Digital Typography*. Cambridge, 154–168.

Nicole Feng, Mark Gillespie, and Keenan Crane. 2023. Winding Numbers on Discrete Surfaces. *ACM Trans. Graph.* 42, 4 (2023), 36:1–36:17. https://doi.org/10.1145/3592401

Lars Hörmander. 2003. *The Analysis of Linear Partial Differential Operators I.* Springer, Berlin, Germany. https://link.springer.com/book/10.1007/978-3-642-61497-2

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 (July 2020), 18 pages. https://doi.org/10.1145/3386569.3392385

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

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

Steven G. Johnson. 2007. The NLopt nonlinear-optimization package. https://github.com/stevengj/nlopt.

Michael M. Kazhdan, Matthew Bolitho, and Hugues Hoppe. 2006. Poisson surface reconstruction. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing, Cagliari, Sardinia, Italy, June 26-28, 2006 (ACM International Conference Proceeding Series, Vol. 256)*, Alla Sheffer and Konrad Polthier (Eds.). Eurographics Association, 61–70. https://doi.org/10.2312/SGP/SGP06/061-070

Minchen Li, Danny M. Kaufman, Vladimir G. Kim, Justin Solomon, and Alla Sheffer. 2018. OptCuts: joint optimization of surface cuts and parameterization. *ACM Trans. Graph.* 37, 6 (2018), 247. https://doi.org/10.1145/3272127.3275042

Yijing Li and Jernej Barbič. 2018. Immersion of Self-Intersecting Solids and Surfaces. *ACM Trans. Graph.* 37, 4, Article 45 (jul 2018), 14 pages. https://doi.org/10.1145/3197517.3201327

J. Manson and S. Schaefer. 2011. Wavelet Rasterization. *Computer Graphics Forum* 30, 2 (2011), 395–404. https://doi.org/10.1111/j.1467-8659.2011.01887.x

Josiah Manson and Scott Schaefer. 2013. Analytic Rasterization of Curves with Polynomial Filters. *Computer Graphics Forum* 32, 2pt4 (2013), 499–507. https://doi.org/10.1111/cgf.12070 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12070

Jirí Minarcík, Sam Estep, Wode Ni, and Keenan Crane. 2024. Minkowski Penalties: Robust Differentiable Constraint Enforcement for Vector Graphics. In *ACM SIGGRAPH 2024 Conference Papers, SIGGRAPH 2024, Denver, CO, USA, 27 July 2024- 1 August 2024*, Andres Burbano, Denis Zorin, and Wojciech Jarosz (Eds.). ACM, 2. https://doi.org/10.1145/3641519.3657495

Roi Poranne, Marco Tarini, Sandro Huber, Daniele Panozzo, and Olga Sorkine-Hornung. 2017. Autocuts: Simultaneous Distortion and Cut Optimization for UV Mapping. *ACM Trans. Graph.* 36, 6, Article 215 (nov 2017), 11 pages. https://doi.org/10.1145/3130800.3130845

V.L. Rvachev, T.I. Sheiko, V. Shapiro, and I. Tsukanov. 2001. Transfinite interpolation over implicitly defined sets. *Computer Aided Geometric Design* 18, 3 (2001), 195–220. https://doi.org/10.1016/S0167-8396(01)00015-2

Krister Svanberg. 2002. A class of globally convergent optimization methods based on conservative convex separable approximations. *SIAM Journal on Optimization* 12 (2002), 555–573. https://doi.org/10.1137/s1052623499362822

Wen-Jen Tsay and Peng-Hsuan Ke. 2023. A simple approximation for the bivariate normal integral. *Communications in Statistics - Simulation and Computation* 52, 4 (2023), 1462–1475. https://doi.org/10.1080/03610918.2021.1884718

Rui Xu, Zhiyang Dou, Ningna Wang, Shiqing Xin, Shuang-Min 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 (2023), 111:1–111:15. https://doi.org/10.1145/3592129

Jiang Zhu, Yurio Hosaka, and Hayato Yoshioka. 2019. A Robust Algorithm to Remove the Self-intersection of 3D Mesh Data without Changing the Original Shape. *Journal of Physics: Conference Series* 1314, 1 (oct 2019), 012149. https://doi.org/10.1088/1742-6596/1314/1/012149

# *Supplementary Material*
# GauWN: Gaussian-smoothed Winding Number and its Derivatives

HAORAN SUN, State Key Lab of CAD&CG, Zhejiang University, China
JINGKAI WANG, State Key Lab of CAD&CG, Zhejiang University, China and Shanghai Jiao Tong University, China
HUJUN BAO, State Key Lab of CAD&CG, Zhejiang University, China
JIN HUANG*, State Key Lab of CAD&CG, Zhejiang University, China

## A PROOF OF THE EXTENDEND FORM OF THE DIVERGENCE THEOREM

In this section, we provide the proof of the extendend form of the divergence theorem that can be applied to boundaries with possible intersections.

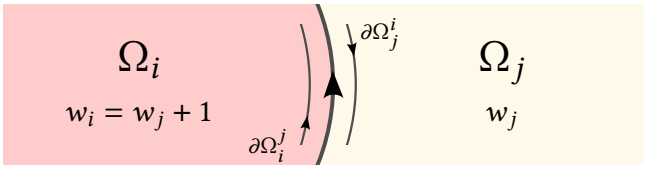



Fig. 1. When crossing a curve from right to left, the winding number must increase by exactly 1. For a pair of locally neighboring regions $\Omega_i$ and $\Omega_j$, the sum of the integrals $\int \langle \mathbf{F}, \mathbf{n} \rangle$ over their respective oriented boundaries $\partial\Omega_i^j$ and $\partial\Omega_j^i$, weighted by their corresponding winding numbers $w_i$ and $w_j$, is equal to the integration over the curve weighted by a factor of 1.

Lemma 3.1. *For a closed 2D piecewise smooth curve (possibly with self-intersections) $\Gamma$ and a smooth vector field $\mathbf{F}$ on the plane, we have*

$$\iint_{\mathbb{R}^2 \setminus \Gamma} w^\Gamma(\mathbf{x}) \nabla \cdot \mathbf{F} \, \mathrm{d}S = \oint_\Gamma \langle \mathbf{F}, \mathbf{n} \rangle \, \mathrm{d}s, \tag{1}$$

*where $\mathbf{n}$ is the unit normal vector of $\Gamma$.*

Proof. The curve $\Gamma$, which may have self-intersections, divides the plane into regions $\Omega_1, \Omega_2, \cdots, \Omega_n$. For each region $\Omega_i$, there is a constant winding number $w_i$. Therefore,

$$\begin{aligned} \iint_{\mathbb{R}^2 \setminus \Gamma} w^\Gamma(\mathbf{x}) \nabla \cdot \mathbf{F} \, \mathrm{d}S &= \sum_{i=1}^n w_i \iint_{\Omega_i} \nabla \cdot \mathbf{F} \, \mathrm{d}S \\ &= \sum_{i=1}^n w_i \oint_{\partial\Omega_i} \langle \mathbf{F}, \mathbf{n} \rangle \, \mathrm{d}s. \end{aligned} \tag{2}$$

First, we discuss the case where the curve has no overlapping segments, *i.e.*, there is only one curve segment between (geometrically) adjacent regions. In other words, all intersection points are isolated. For such a curve, we can further divide each oriented boundary $\partial\Omega_i$ by the adjacent regions:

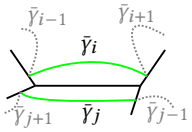
$$\partial\Omega_i = \bigcup_{\Omega_i \cap \Omega_j \neq \varnothing} \partial\Omega_i^j. \tag{3}$$

Note that $\partial\Omega_i^j$ and $\partial\Omega_j^i$ are the same curve between $\Omega_i$ and $\Omega_j$ but with opposite orientations, *i.e.*, opposite normal vectors $\mathbf{n}$. Thus, $\int_{\partial\Omega_i^j} \langle \mathbf{F}, \mathbf{n} \rangle \, \mathrm{d}s = -\int_{\partial\Omega_j^i} \langle \mathbf{F}, \mathbf{n} \rangle \, \mathrm{d}s$.

Furthermore, by the property of the winding number, if $\partial\Omega_i^j$ has the same orientation as $\Gamma$, then $w_i = w_j + 1$ and vice versa (see Figure 1). Without loss of generality, we assume $\partial\Omega_i^j$ has the same orientation as $\Gamma$. Then,

$$\begin{aligned} &\sum_{i=1}^n w_i \oint_{\partial\Omega_i} \langle \mathbf{F}, \mathbf{n} \rangle \, \mathrm{d}s \\ &\quad = \sum_{i=1}^n w_i \sum_{\Omega_i \cap \Omega_j \neq \varnothing} \int_{\partial\Omega_i^j} \langle \mathbf{F}, \mathbf{n} \rangle \, \mathrm{d}s \\ &\quad = \sum_{\Omega_i \cap \Omega_j \neq \varnothing} \left( w_i \int_{\partial\Omega_i^j} \langle \mathbf{F}, \mathbf{n} \rangle \, \mathrm{d}s - w_j \int_{\partial\Omega_i^j} \langle \mathbf{F}, \mathbf{n} \rangle \, \mathrm{d}s \right) \\ &\quad = \sum_{\Omega_i \cap \Omega_j \neq \varnothing} \int_{\partial\Omega_i^j} \langle \mathbf{F}, \mathbf{n} \rangle \, \mathrm{d}s = \oint_\Gamma \langle \mathbf{F}, \mathbf{n} \rangle \, \mathrm{d}s. \end{aligned} \tag{4}$$

Denoting $L(\Gamma) = \iint_{\mathbb{R}^2 \setminus \Gamma} w^\Gamma(\mathbf{x}) \nabla \cdot \mathbf{F} \, \mathrm{d}S$ and $R(\Gamma) = \oint_\Gamma \langle \mathbf{F}, \mathbf{n} \rangle \, \mathrm{d}s$, we have proved that $L(\Gamma) = R(\Gamma)$ if $\Gamma$ is a closed curve and does not have overlapped segments. Now we consider the curve $\Gamma$ with non-isolated intersection points, *i.e.*, there are multiple curve segments in $\Gamma$ between two (geometrically) adjacent regions $\Omega_i$ and $\Omega_j$. As $w^{\Gamma_1+\Gamma_2} = w^{\Gamma_1} + w^{\Gamma_2}$, we also have $L(\Gamma_1 + \Gamma_2) = L(\Gamma_1) + L(\Gamma_2)$. Similarly, we have $R(\gamma_1 + \gamma_2) = R(\gamma_1) + R(\gamma_2)$ due to the line integral form. Now, there always exists a segmentation $\Gamma = \sum_{i=1}^k \gamma_i$, such that each $\gamma_i$ does not overlap with itself. It is not hard to construct another closed curve $\bar{\Gamma} = \sum_{i=1}^k \bar{\gamma}_i$, satisfying: (a) $\bar{\gamma}_i$ does not overlap with $\gamma_i$, (b) $\gamma_i - \bar{\gamma}_i$ is a closed circle, and (c) $\bar{\Gamma}$ has no self-overlaps. That is, both $\bar{\Gamma}$ and $\gamma_i - \bar{\gamma}_i$ are closed curves with no non-isolated intersection points, and Lemma 3.1 applies. Therefore, we have




*Corresponding author.

Authors' addresses: Haoran Sun, hrsun@zju.edu.cn, State Key Lab of CAD&CG, Zhejiang University, Hangzhou, Zhejiang, China; Jingkai Wang, ohg@zju.edu.cn, State Key Lab of CAD&CG, Zhejiang University, Hangzhou, Zhejiang, China, and Shanghai Jiao Tong University, Shanghai, China; Hujun Bao, Jin Huang, {bao, hj}@cad.zju.edu.cn, State Key Lab of CAD&CG, Zhejiang University, Hangzhou, Zhejiang, China.

$L(\gamma_i - \bar{\gamma}_i) = R(\gamma_i - \bar{\gamma}_i), L(\bar{\Gamma}) = R(\bar{\Gamma})$. And finally,

$$\begin{aligned} L(\Gamma) &= L(\Gamma - \bar{\Gamma} + \bar{\Gamma}) = L(\Gamma - \bar{\Gamma}) + L(\bar{\Gamma}) \\ &= L(\sum_{i=1}^{k} \gamma_i - \sum_{i=1}^{k} \bar{\gamma}_i) + L(\bar{\Gamma}_i) \\ &= L(\sum_{i=1}^{k} (\gamma_i - \bar{\gamma}_i)) + R(\bar{\Gamma}_i) \\ &= \sum_{i=1}^{k} L(\gamma_i - \bar{\gamma}_i) + \sum_{i=1}^{k} R(\bar{\gamma}_i) \\ &= \sum_{i=1}^{k} R(\gamma_i - \bar{\gamma}_i) + \sum_{i=1}^{k} R(\bar{\gamma}_i) \\ &= \sum_{i=1}^{k} R(\gamma_i) - \sum_{i=1}^{k} R(\bar{\gamma}_i) + \sum_{i=1}^{k} R(\bar{\gamma}_i) \\ &= \sum_{i=1}^{k} R(\gamma_i) = R(\Gamma). \end{aligned} \tag{5}$$

□

It should be noticed that the above lemma also applies to piecewise smooth curves as long as the measure of the set of non-smooth points (*e.g.*, corners of a polygon) is zero. One can simply skip the points without a well-defined normal when computing the boundary integral in Equation (1).

## B DETAILS OF VALUE EVALUATION

In this section, we provide the details of the efficient evaluation of the Gaussian-smoothed winding number by utilizing the extendend divergence theorem Lemma 3.1 and special functions related to Gaussian integrals.

Applying Lemma 3.1, we have

$$\begin{aligned} W_\sigma^\Gamma(\mathbf{q}) &= \iint_{\mathbb{R}^2 \setminus \Gamma} w^\Gamma(\mathbf{x}) G(\mathbf{x} - \mathbf{q}; \sigma^2 \mathbf{I}) \, \mathrm{d}S \\ &= \oint_\Gamma \left\langle \mathbf{F}_\mathbf{q}(\mathbf{x}), \mathbf{n} \right\rangle \mathrm{d}s \\ &= \sum_{\gamma = \overline{\mathbf{v}_i \mathbf{v}_j}} \int_\gamma \left\langle \mathbf{F}_\mathbf{q}(\mathbf{x}), \mathbf{n} \right\rangle \mathrm{d}s \\ &= \sum_{\gamma = \overline{\mathbf{v}_i \mathbf{v}_j}} \int_\gamma \mathbf{F}_\mathbf{q}.x \, \mathrm{d}y - \mathbf{F}_\mathbf{q}.y \, \mathrm{d}x. \end{aligned} \tag{6}$$

As

$$\mathbf{F}_\mathbf{q}(\mathbf{x}) = \begin{bmatrix} g(\mathbf{x}.y - \mathbf{q}.y; \sigma^2) \Phi(\mathbf{x}.x - \mathbf{q}.x; \sigma^2) \\ 0 \end{bmatrix}, \tag{7}$$

only the first component is non-zero, and we have:

$$\begin{aligned} \int_\gamma \mathbf{F}_\mathbf{q}.x \, \mathrm{d}y &= \int_\gamma g(\mathbf{x}.y - \mathbf{q}.y; \sigma^2) \Phi(\mathbf{x}.x - \mathbf{q}.x; \sigma^2) \, \mathrm{d}y \\ &= \Delta y \int_0^1 g(\mathbf{v}_j.y + t(\mathbf{v}_i.y - \mathbf{v}_j.y) - \mathbf{q}.y; \sigma^2) \\ &\quad \cdot \Phi(\mathbf{v}_j.x + t(\mathbf{v}_i.x - \mathbf{v}_j.x) - \mathbf{q}.x; \sigma^2) \, \mathrm{d}t. \end{aligned} \tag{8}$$

Equation (8) can be computed using Equation 10,010.1 from [Owen 1980], *i.e.*,

$$\int_{-\infty}^{Y} g(x) \Phi(a + bx) \, \mathrm{d}x = \mathrm{BvN}\left( \frac{a}{\sqrt{1 + b^2}}, Y; \frac{-b}{\sqrt{1 + b^2}} \right), \tag{9}$$

where $g(x)$ and $\Phi(x)$ are respectively the probability density function and cumulative distribution function of a 1D Gaussian distribution with a mean of 0 and a variance of 1, and BvN is the bivariate normal cumulative distribution function:

$$\begin{aligned} &\mathrm{BvN}(a, b; \rho) \coloneqq \\ &\frac{1}{2\pi\sqrt{1 - \rho^2}} \int_{-\infty}^{a} \int_{-\infty}^{b} \exp\left( -\frac{x^2 - 2\rho x y + y^2}{2(1 - \rho^2)} \right) \mathrm{d}x \, \mathrm{d}y. \end{aligned} \tag{10}$$

More concretely, we have

$$\begin{aligned} &\mathrm{O}(Y, a, b) \\ &\quad \coloneqq \int_{-\infty}^{Y} g(x) \Phi(a + bx) \, \mathrm{d}x = \mathrm{BvN}\left( \frac{a}{\sqrt{1 + b^2}}, Y; \frac{-b}{\sqrt{1 + b^2}} \right) \\ &\mathrm{O}_\sigma(Y, a, b, \mu_1, \mu_2) \\ &\quad \coloneqq \int_{-\infty}^{Y} g(x - \mu_1; \sigma^2) \Phi(a + bx - \mu_2; \sigma^2) \, \mathrm{d}x \\ &\quad = \mathrm{O}\left( \frac{Y - \mu_1}{\sigma}, \frac{a + b\mu_1 - \mu_2}{\sigma}, b \right) \\ &\mathrm{O}'_\sigma(h, k, a, b, \mu_1, \mu_2) \\ &\quad \coloneqq \int_{h}^{k} g(x - \mu_1; \sigma^2) \Phi(a + bx - \mu_2; \sigma^2) \, \mathrm{d}x \\ &\quad = \mathrm{O}_\sigma(k, a, b, \mu_1, \mu_2) - \mathrm{O}_\sigma(h, a, b, \mu_1, \mu_2) \\ &\mathrm{O}''_\sigma(h, k, a, b, e, f, , \mu_1, \mu_2) \\ &\quad \coloneqq \int_{h}^{k} g(at + b - \mu_1; \sigma^2) \Phi(et + f - \mu_2; \sigma^2) \, \mathrm{d}t \\ &\quad = \frac{1}{a} \mathrm{O}'_\sigma\left( ah + b, ak + b, f - \frac{be}{a}, \frac{e}{a}, , \mu_1, \mu_2 \right). \end{aligned} \tag{11}$$

and finally

$$\begin{aligned} &\int_\gamma \mathbf{F}_\mathbf{q}.x \, \mathrm{d}y \\ &\quad = \Delta y \int_0^1 g(\mathbf{v}_j.y + t(\mathbf{v}_i.y - \mathbf{v}_j.y) - \mathbf{q}.y; \sigma^2) \\ &\qquad \cdot \Phi(\mathbf{v}_j.x + t(\mathbf{v}_i.x - \mathbf{v}_j.x) - \mathbf{q}.x; \sigma^2) \, \mathrm{d}t \\ &\quad = \Delta y \, \mathrm{O}''_\sigma\left( 0, 1, \mathbf{v}_i.y - \mathbf{v}_j.y, \mathbf{v}_j.y, \mathbf{v}_i.x - \mathbf{v}_j.x, \mathbf{v}_j.x, \sigma, \mathbf{q}.y, \mathbf{q}.x \right). \end{aligned} \tag{12}$$

Equations (11) and (12) are the equations needed for implementation. Although there is no closed form for BvN, there are many efficient numerical approximations available. We use the method from [Tsay and Ke 2023], implementation provided by [Cortes 2022].

## C DETAILS OF GRADIENT EVALUATION

In this section, we derive the efficient gradient evaluation of the Gaussian-smoothed winding number using radial symmetry and orthogonal separability properties of the Guassian kernel.

The line integration form of the gradient is

$$\frac{\partial}{\partial \mathbf{v}_i} W_\sigma^\Gamma = \sum_{\gamma=\overline{\mathbf{v}_i\mathbf{v}_j}} \int_\gamma G(\mathbf{x}-\mathbf{q};\sigma^2\mathbf{I}) \frac{\frac{\partial}{\partial l}\left\langle \mathbf{n}, \mathbf{x}-\mathbf{v}_j\right\rangle \mathbf{v}_i}{\left|\frac{\partial}{\partial l}\left\langle \mathbf{n}, \mathbf{x}-\mathbf{v}_j\right\rangle \mathbf{x}\right|} \mathrm{d}s, \tag{13}$$

where

$$\begin{aligned}
&\frac{\partial}{\partial \mathbf{x}}\left\langle \mathbf{n}, \mathbf{x}-\mathbf{v}_j\right\rangle = \mathbf{n} \\
&\frac{\partial}{\partial \mathbf{v}_i}\left\langle \mathbf{n}, \mathbf{x}-\mathbf{v}_j\right\rangle \\
&\quad = \frac{\partial}{\partial \mathbf{v}_i}\left\langle \frac{\mathbf{R}_{90}(\mathbf{v}_i-\mathbf{v}_j)}{\|\mathbf{v}_i-\mathbf{v}_j\|}, \mathbf{x}-\mathbf{v}_j\right\rangle \\
&\quad = \left(\frac{\mathbf{I}_2}{\|\mathbf{v}_i-\mathbf{v}_j\|} - \frac{(\mathbf{v}_i-\mathbf{v}_j)(\mathbf{v}_i-\mathbf{v}_j)^\top}{\|\mathbf{v}_i-\mathbf{v}_j\|^3}\right)\mathbf{R}_{90}^\top(\mathbf{x}-\mathbf{v}_j),
\end{aligned} \tag{14}$$

and $\mathbf{R}_{90}$ is the 90° rotation matrix $\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$.

Since we will only integrate over $\left\langle \mathbf{n}, \mathbf{x}-\mathbf{v}_j\right\rangle = 0$, we can focus on the case where $\mathbf{x} = \mathbf{v}_j + t(\mathbf{v}_i-\mathbf{v}_j)/l, t \in [0,l], l = \|\mathbf{v}_i-\mathbf{v}_j\|$. We can further simplify Equation (14) as follows:

$$\begin{aligned}
&\frac{\partial}{\partial \mathbf{v}_i}\left\langle \mathbf{n}, \mathbf{x}-\mathbf{v}_j\right\rangle \\
&= \frac{t}{l}\left(\frac{\mathbf{R}_{90}^\top(\mathbf{v}_i-\mathbf{v}_j)}{\|\mathbf{v}_i-\mathbf{v}_j\|} - \frac{(\mathbf{v}_i-\mathbf{v}_j)\overbrace{(\mathbf{v}_i-\mathbf{v}_j)^\top\mathbf{R}_{90}^\top(\mathbf{v}_i-\mathbf{v}_j)}^{\langle \mathbf{R}_{90}(\mathbf{v}_i-\mathbf{v}_j),\mathbf{v}_i-\mathbf{v}_j\rangle=0}}{\|\mathbf{v}_i-\mathbf{v}_j\|^3}\right) \\
&= -\frac{t}{l}\mathbf{n}.
\end{aligned} \tag{15}$$

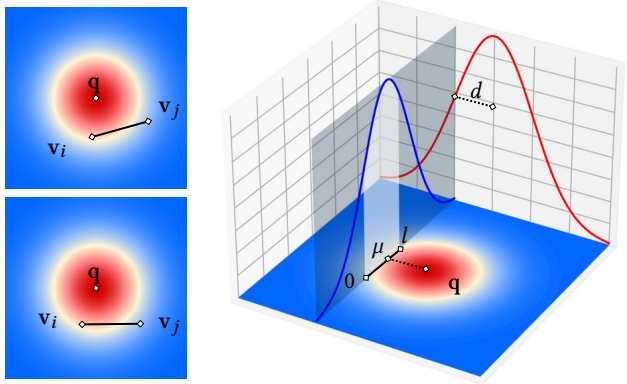



Fig. 2. The two-dimensional Gaussian kernel is the product of two one-dimensional kernels. This means that if we rotate $\gamma$ to align with an axis, the Gaussian kernel would be constant along the other axis.

It is apparent that the Gaussian kernel in 2D exhibits two properties that simplify the computation: radial symmetry and orthogonal separability. Radial symmetry implies that if $\|\mathbf{x}-\mathbf{q}\| = \|\mathbf{x}'-\mathbf{q}\|$, then $G(\mathbf{x}-\mathbf{q};\sigma^2\mathbf{I}) = G(\mathbf{x}'-\mathbf{q};\sigma^2\mathbf{I})$. Orthogonal separability states that $G(\mathbf{x}-\mathbf{q};\sigma^2\mathbf{I}) = g(\mathbf{x}.x-\mathbf{q}.x;\sigma^2)g(\mathbf{x}.y-\mathbf{q}.y;\sigma^2)$. Consequently, the line integral can be simplified (see Figure 2):

$$G(\mathbf{v}_i + t(\mathbf{v}_j-\mathbf{v}_i)/l - \mathbf{q};\sigma^2\mathbf{I}) = g(d;\sigma^2)g(t-\mu;\sigma^2), \tag{16}$$

where $d = \sqrt{\|\mathbf{q}-\mathbf{v}_j\|^2 - \left\langle \mathbf{q}-\mathbf{v}_j, (\mathbf{v}_i-\mathbf{v}_j)/l\right\rangle^2}$ represents the distance from $\mathbf{q}$ to the edge, and $\mu = \left\langle \mathbf{q}-\mathbf{v}_j, \mathbf{v}_i-\mathbf{v}_j\right\rangle/l$.

Using Equation 101 from [Owen 1980], *i.e.*,

$$\int xg(a+bx)\,\mathrm{d}x = -\frac{1}{b^2}g(a+bx) - \frac{a}{b^2}\Phi(a+bx), \tag{17}$$

where $\Phi(t) = \int_{-\infty}^t g(x)\,\mathrm{d}x$, and

$$\begin{aligned}
\int tg(t-\mu;\sigma^2)\,\mathrm{d}t &= \frac{1}{\sigma}\int tg\left(\frac{-\mu+t}{\sigma}\right)\mathrm{d}t \\
&= -\sigma g\left(\frac{-\mu+t}{\sigma}\right) + \mu\Phi\left(\frac{-\mu+t}{\sigma}\right) \\
&= -\sigma^2 g(t-\mu;\sigma^2) + \mu\Phi(t-\mu;\sigma^2),
\end{aligned} \tag{18}$$

we can now obtain the closed form of the integral in Equation (13) as

$$\begin{aligned}
&\frac{\partial}{\partial \mathbf{v}_i} W_\sigma^\Gamma \\
&= \sum_{\gamma=\overline{\mathbf{v}_i\mathbf{v}_j}} -\frac{\mathbf{n}}{l}\int_0^l tG(\mathbf{v}_j + t(\mathbf{v}_i-\mathbf{v}_j)/l - \mathbf{q};\sigma^2\mathbf{I})\,\mathrm{d}t \\
&= \sum_{\gamma=\overline{\mathbf{v}_i\mathbf{v}_j}} -g(d;\sigma^2)/l \\
&\left[\sigma^2 g(-\mu;\sigma^2) - \sigma^2 g(l-\mu;\sigma^2) + \mu\Phi(l-\mu;\sigma^2) - \mu\Phi(-\mu;\sigma^2)\right]\mathbf{n}.
\end{aligned} \tag{19}$$

## D PSEUDOCODES OF THE BVH

**ALGORITHM 1:** GauWNBVH

```
Input  : q ;                                   // the query point
Input  : node ;                                // the AABB node
Use    : wnseg ;                               // Equation (6)
Use    : far_enough
Output : wg

wg ← 0;
if far_enough(node, q) then
    for v1, v2 in approximated segments of node do
        wg ← wg + wnseg(v1,v2, q);
    end
    return wg;
else
    if node is leaf then
        for v1, v2 in children of node do
            wg ← wg + wnseg(v1, v2, q);
        end
    else
        for child in children of node do
            wg ← wg + GauWNBVH(q, child);
        end
    end
    return wg;
end
```

**ALGORITHM 2:** GauWNSpatialGradBVH

```
Input  : q ;                                   // the query point
Input  : node ;                                // the AABB node
Use    : gradvert ;                            // Equation (19)
Use    : far_enough
Output : grad

grad ← 0;
if not far_enough(node, q) then
    if node is leaf then
        for v in children of node do
            grad ← grad − gradvert(v, q);
        end
    else
        for child in children of node do
            grad ← grad + GauWNSpatialGradBVH(q, child);
        end
    end
    return grad;
end
```

**ALGORITHM 3:** GauWNVertexGradBVH

```
Input  : q ;                                   // the query point
Input  : node ;                                // the AABB node
InOut  : grads []
Use    : gradvert ;                            // Equation (19)
Use    : far_enough

if not far_enough(node, q) then
    if node is leaf then
        for v in children of node do
            grads[v] ← grads[v] +gradvert(v, q);
        end
    else
        for child in children of node do
            GauWNVertexGradBVH(q, child, grads);
        end
    end
end
```

## E ALTERNATIVE COMPUTATION METHODS

One might wonder how much benefit we can get from the derivations we have made. To answer this question, we provide two alternative methods to compute GauWN. The first method is to compute the convolution directly using WN sampled at the vertices of a $6\sigma\times6\sigma$ grid centered at each query point. The second method is to compute the line integrals in Equation (6) using the midpoint rule for quadrature. BVH is used in both methods. The results and errors (our proposed computation as ground-truth) are shown in Figure 3. We can find that the errors of numerical convolution are mainly concentrated near the boundary, where the region of interest usually is, while the midpoint rule exhibits striped artifacts when the number of quadrature points is small. As the number of samples or quadrature points increases, the errors decrease faster in the line integration method. However, it will generally still be much slower than GauWN to get an acceptable result.

## F QUADRATURE IN 3D

The extension of GauWN in 3D *w.r.t.* a surface $\mathcal{P}$ is straightforward:

$$W_\sigma^{\mathcal{P}}(\mathbf{q}) \coloneqq \iiint_{\mathbb{R}^3\setminus\mathcal{P}} w^{\mathcal{P}}(\mathbf{x}) G_3(\mathbf{x}-\mathbf{q};\sigma^2\mathbf{I})\,\mathrm{d}S, \tag{20}$$

where $G_3(\mathbf{x};\sigma^2\mathbf{I}) = g(\mathbf{x}.x;\sigma^2)g(\mathbf{x}.y;\sigma^2)g(\mathbf{x}.z;\sigma^2)$.

Similar to the 2D setting,

$$\mathbf{F}_\mathbf{q}(\mathbf{x}) = \begin{bmatrix} g(\mathbf{x}.y-\mathbf{q}.y;\sigma^2)g(\mathbf{x}.z-\mathbf{q}.z;\sigma^2)\Phi(\mathbf{x}.x-\mathbf{q}.x;\sigma^2) \\ 0 \\ 0 \end{bmatrix} \tag{21}$$

satisfies

$$\nabla\cdot\mathbf{F}_\mathbf{q}(\mathbf{x}) = G_3(\mathbf{x}-\mathbf{q};\sigma^2\mathbf{I}). \tag{22}$$

And the extended form of the divergence theorem is still valid in 3D. Though we have not yet found the closed form of

$$\oiint_{\mathcal{P}} \left\langle \mathbf{F}_\mathbf{q}, \mathbf{n}\right\rangle \mathrm{d}s, \tag{23}$$

it can be computed using triangle quadrature methods, though the performance is far from satisfactory:

$$\oiint_{\mathcal{P}} \left\langle \mathbf{F}_\mathbf{q}, \mathbf{n}\right\rangle \mathrm{d}s = \sum_{\triangle\in\mathcal{P}} \sum_{(\mathbf{x},w)\in\mathrm{RULE}(\triangle)} w \left\langle \mathbf{F}_\mathbf{q}, \mathbf{n}\right\rangle. \tag{24}$$

The rule used in Figure 12 of the main paper is [Dunavant 1985].

## REFERENCES


David Cortes. 2022. ApproxCDF. https://github.com/david-cortes/approxcdf/blob/master/src/other.cpp.

D. A. Dunavant. 1985. High degree efficient symmetrical Gaussian quadrature rules for the triangle. *Internat. J. Numer. Methods Engrg.* 21, 6 (1985), 1129–1148. https://doi.org/10.1002/nme.1620210612

D. B. Owen. 1980. A table of normal integrals. *Communications in Statistics - Simulation and Computation* 9, 4 (1980), 389–419. https://doi.org/10.1080/03610918008812164

Wen-Jen Tsay and Peng-Hsuan Ke. 2023. A simple approximation for the bivariate normal integral. *Communications in Statistics - Simulation and Computation* 52, 4 (2023), 1462–1475. https://doi.org/10.1080/03610918.2021.1884718

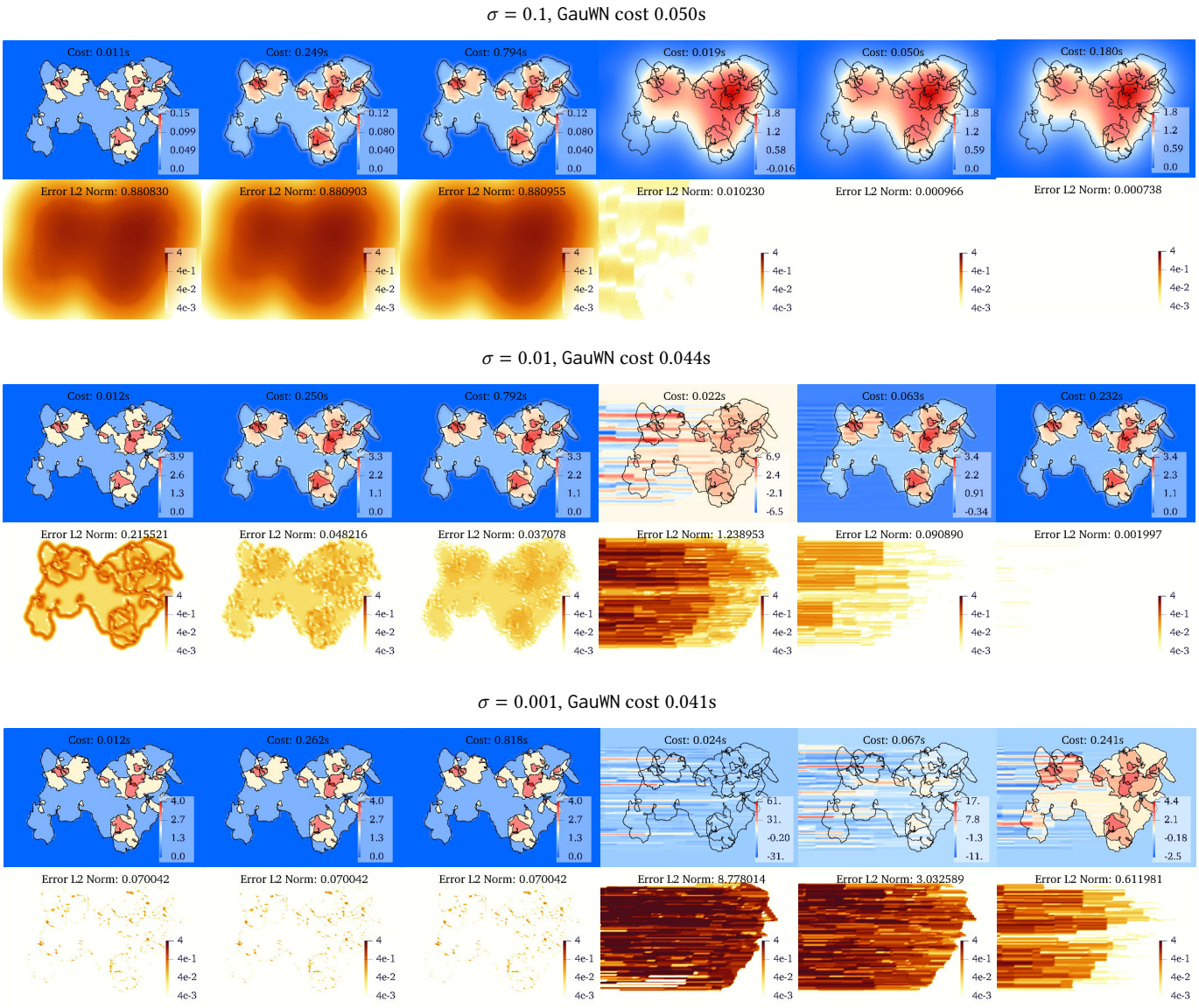



Fig. 3. *Left*: numerical convolution results with 1, 25, 81 samples per query point. *Right*: midpoint rule results with 1, 4, 16 quadrature points per edge. "GauWN cost" is the time cost using expressions of Section B.