Fast Winding Numbers for Soups and Clouds

GAVIN BARILL, University of Toronto, Canada
NEIL G. DICKSON, Side Effects Software Inc., Canada
RYAN SCHMIDT, Gradientspace, Canada
DAVID I.W. LEVIN* and ALEC JACOBSON*, University of Toronto, Canada

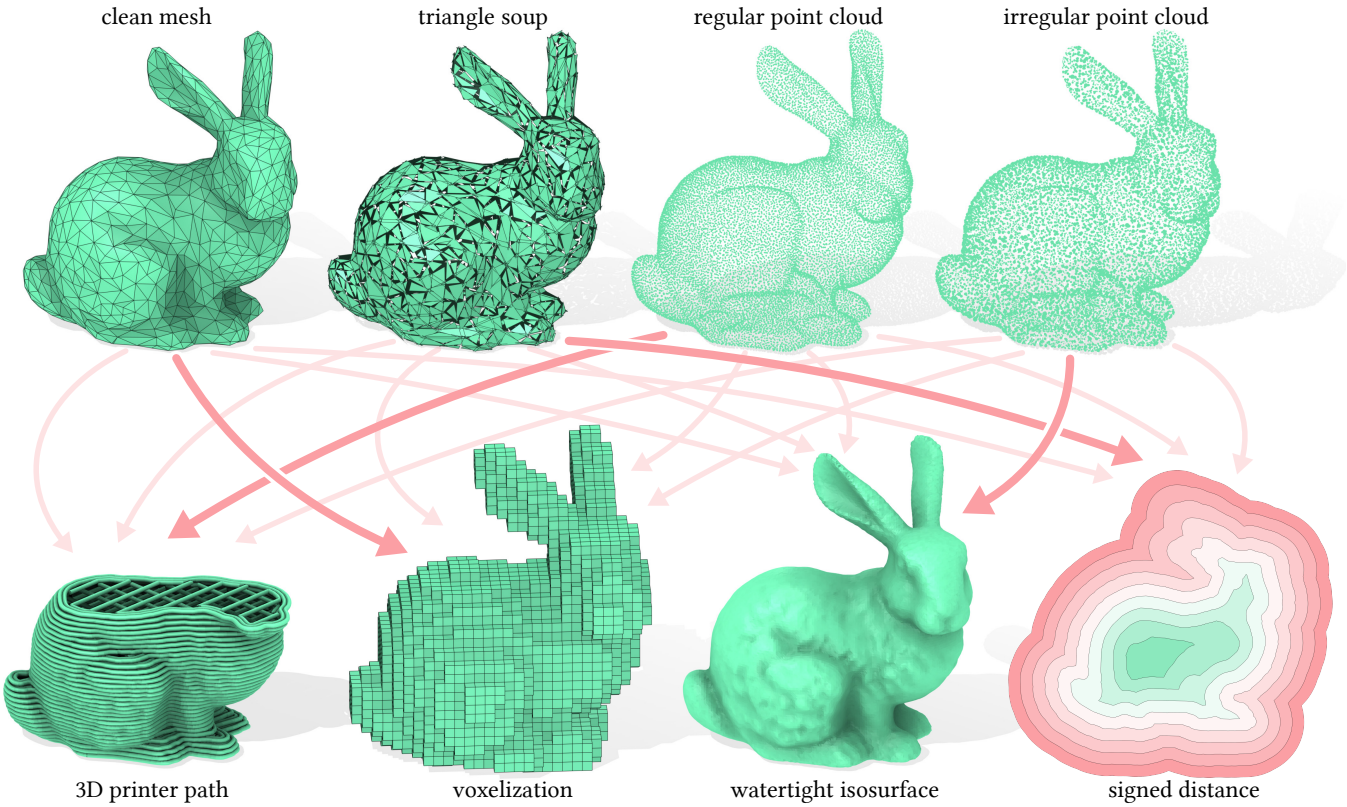

Fig. 1. In this paper, we further generalize the winding number to point clouds and propose a hierarchical algorithm for fast evaluation (up to 1000× speedup). This enables efficient answers to inside-outside queries for a wider class of shape representations (top) during a variety of tasks (bottom).

Inside-outside determination is a basic building block for higher-level geometry processing operations. Generalized winding numbers provide a robust answer for triangle meshes, regardless of defects such as self-intersections, holes or degeneracies. In this paper, we further generalize the winding number to point clouds. Previous methods for evaluating the winding number are slow for completely disconnected surfaces, such as triangle soups or–in the extreme case– point clouds. We propose a tree-based algorithm to reduce the asymptotic complexity of generalized winding number computation, while closely approximating the exact value. Armed with a fast evaluation, we demonstrate the winding number in a variety of new applications: voxelization, signing distances, generating 3D printer paths, defect-tolerant mesh booleans and point set surfaces.

CCS Concepts: • **Computing methodologies** → **Mesh models**; **Point-based models**; *Volumetric models*; Shape modeling;

Additional Key Words and Phrases: generalized winding number, inside-outside segmentation, tree-based algorithm, robust geometry processing

ACM Reference Format:
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 (August 2018), 12 pages. https://doi.org/10.1145/3197517.3201337

*Joint last authors

Authors' addresses: Gavin Barill, David I.W. Levin, Alec Jacobson, UofT, 40 St George St, Toronto, M5S 2E4, Canada, gavin.barill@mail.utoronto.ca.

0730-0301/2018/8-ART43 $15.00
https://doi.org/10.1145/3197517.3201337

1 INTRODUCTION

"Well, are you in or are you out?"

— Jodi Kramer, *Dazed and Confused* (1993)

Determining whether a point is inside or outside of a given shape is one of the most basic geometric questions. Inside-outside segmentation is crucial for: signing distance fields, tetrahedralizing or voxelizing volumes, representing smooth surfaces from point clouds, generating 3D printer path instructions, and surface repair. For analytic shapes and *sufficiently* clean discrete surface meshes, we can answer the question confidently and quickly. Unfortunately, most surface representations found *in the wild* are either completely unstructured (e.g., point clouds) or riddled with defects such as open boundaries, duplicated or degenerate geometry, self-intersections and non-manifold combinatorics.

The classic winding number determines how many times a planar curve encircles a query point (see, e.g., [Meister 1769]). Generalized winding numbers [Jacobson et al. 2013] extend this concept to oriented triangle meshes suffering from the aforementioned defects. For oriented triangle meshes, this is computed as a sum of signed solid angles $\Omega_t(\mathbf{q})$ of each triangle t subtended at a query point $\mathbf{q}$:

$$w_S(\mathbf{q}) = \frac{1}{4\pi} \sum_{t \in \text{triangles}} \Omega_t(\mathbf{q}). \tag{1}$$

For closed watertight meshes, this perfectly reproduces the indicator function (1 inside, 0 outside). For overlapping regions, the winding number measures *how many* times the region is inside the surface. For holey or non-manifold surfaces, the winding number produces a smoothly varying function revealing a fractional measure of insideness. While simple and robust, a direct evaluation of this sum: 1) is slow, requiring $O(nm)$ computation for n queries and an m-triangle mesh; and 2) only applies to triangle meshes.

For large geometries and interactive applications, inside-outside queries need to be efficient. Existing optimizations for winding number computation either merely use parallelization or make heavy assumptions about mesh connectivity. For large, incoherent triangle "soups" often encountered during scanning or modeling, existing methods are too slow.

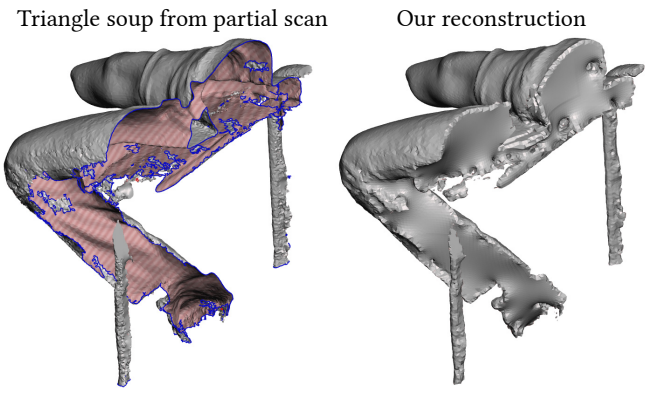

Fig. 2. This soupy scan of an amputee's leg has a veritable archipelago of dangling components, a major challenge for automatic mesh clean up methods. Details of the scan are maintained, while missing regions are filled in with smooth minimal surfaces.

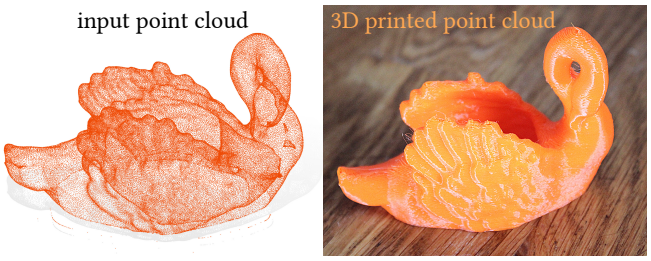

Fig. 3. To 3D print the shape represented by the winding field, we convert it to a stack of 2D polygons, which are then filled with *toolpaths* by the 3D printer software. We extract the polygons using "marching squares" [Lorensen and Cline 1987]) again with a continuation approach. The 3D winding number is used for field evaluations – the 2D winding number along the slice is not the same for open geometry.

Meanwhile, determining the smooth surface interpolating oriented point clouds is equivalent to extracting the boundary between what the points classify as inside or outside. Most existing point set surface methods are based on grid-dependent discretizations or *custom tailored* radial basis functions. These methods focus on level-set extraction, but knowing the answer to the inside-outside question has important applications away from the level-set (e.g., for signed distances, voxelization, solid 3D printing, etc.).

In this paper, we propose a fast method for computing generalized winding numbers on arbitrary triangle *soups* and point *clouds*. We begin by deriving a definition of the winding number for oriented point clouds. This directly enables a novel point set surface representation from the sum of winding number contributions from each point. Our function exactly interpolates a cloud of oriented points — in contrast to smooth surface approximation functions for noisy points. We then *asymptotically* improve the performance of this sum with a tree-based algorithm for computing error-controlled approximations of far away points.

Analytically integrating our definition for points leads to the familiar generalized winding number for triangles. By applying the same integration to our approximations, we generalize our fast evaluation algorithm to triangle soups as well.

We show evidence of the performance and approximation accuracy of our method on a large benchmark, where we achieve up to 1000× speed-up for large triangle soups. We test our method in a variety of applications including: point set surfaces, voxelization, boolean operations on triangle soups, signing distance fields, and mesh cleanup. For an example of mesh cleanup see Fig. 2.

Quickly and robustly answering the inside-outside question allows raw point clouds and triangle soups to travel deeper into the geometry processing pipeline, avoiding lossy representation conversions. As a prototypical example, we show direct 3D printing of point clouds (see Fig. 3).

2 BACKGROUND

The generalized winding number in Equation (2) was introduced by Jacobson et al. [2013] to robustly segment inside from outside

for triangle meshes. Acknowledging that a direct summation is too slow, they propose a divide and conquer algorithm based on cutting the mesh into connected components and computing a direct sum on the scale of the component boundaries. Though this method achieves sub-linear performance for most meshes, the precomputation is involved and overhead during evaluation is much higher than our method: we show up to 1000× speed-ups over this method. Meanwhile, in the worst case, triangle soups can have so many boundaries that their divide and conquer method degenerates into the slow, direct sum. Our approximation ignores mesh connectivity, which enables it to achieve $O(\log m)$ amortized performance. The winding number for triangle meshes has been used in many contexts since its introduction, all of which benefit from improved efficiency.

Applications of the winding number. In this paper, we highlight a number of novel applications of the winding number since its introduction. Others in the literature have also found winding numbers useful. Dionne et al. [2014] use winding numbers to create voxelizations of triangle meshes; they use a ray stabbing heuristic as an initial guess and resort to direct winding number computation for low confidence voxels, citing poor performance as prohibiting its use everywhere (see Fig. 4). Autodesk's Maya software uses this winding number-based voxelization for computing automatic skinning weights, where faster computation would be "extremely useful" [de Lasa 2018]. The direct sum in Equation (2) is straightforward to parallelize (e.g., using the GPU [Dionne and de Lasa 2013]) but doing so does not change the asymptotic performance. Our fast and equally parallelizeable approximation does.

Enormous datasets of 3D shapes such as ShapeNet [Chang et al. 2015] or Thingi10k [Zhou and Jacobson 2016] present tantalizing opportunities for machine learning. Generalized winding numbers are already used to convert between triangle soup representations to signed voxel grids for generalized adversarial network training and shape generation [Jian and Marcus 2017]. Our method provides faster winding number computation, potentially unlocking training on larger datasets at higher resolutions. With our unified definition, triangle soups and point clouds could possibly be homogenized for training purposes. Andrews [2013] advocates for the use of winding numbers to post-process meshes during user-guided inverse 3D modeling. Similarly, Le & Deng [2017] use winding numbers for mesh repair on automatically generated coarse cages for animation and simulation. Ichim et al. [2017] use winding numbers to compute tetrahedral meshes robustly for anatomical simulations of faces.

2.1 Related Works

The work of Jacobson et al. [2013] includes a rich review of literature related to inside-outside segmentation for triangle meshes (e.g., [Murali and Funkhouser 1997; Nooruddin and Turk 2003; Zhou et al. 2008]). We refer to them for a more comprehensive review. Here we focus on contemporary works as well as those related to our methodology for fast approximation and our point set surface representation.

Boundary Element Method. The generalized winding number definition is *equivalent* to solving the Laplace equation ($\Delta u = 0$) using the boundary element method with jump boundary conditions (i.e., integrating double layer potentials with constant density) [Evans 1997]. Orzan et al. [2008] diffusion curves also solve a Laplace equation, where boundary element method can be employed [Ilbery et al. 2013; Sun et al. 2014, 2012; van de Gronde 2010]. Diffusion curve boundary conditions differ from the winding number's, making it useful for blending colors, but not for inside-outside segmentation. Wang et al. [2013] parameterize the space *exterior* to a given solid using the same boundary element method formulation of the Laplace equation as our method, though again with different boundary conditions producing a different solution with different appropriate applications. Da et al. [2016] employ boundary element method for liquid simulation. Zhang et al. [2014] use fast summation over particle potentials for fluid simulation, and briefly mention using their formulation for Poisson surface reconstruction. Da et al. use direct summation and Zhang et al. introduce a fast summation similar to ours, both forgoing "full Fast Multipole Method" (FMM) [Greengard and Rokhlin 1997] due to overhead and complexity. Our experiments with FMM lead to a similar conclusion (common in the literature [Blelloch and Narlikar 1994]). We opt for a more straightforward tree-algorithm and show applications to inside-outside problems for triangle soups and point clouds.

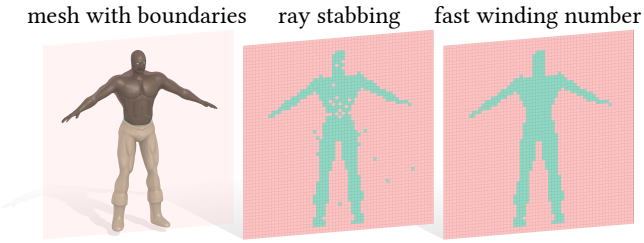

Fig. 4. A common approximation of the winding number is to shoot a random ray and count signed intersections (middle). For triangle soups (left), this leads to floating misclassified voxels. Our fast evaluation of the winding number is also an approximation, but with controlled error: no voxels are misclassified (right).

Point Set Surfaces. There are compelling reasons to *represent* shapes as point clouds [Alexa et al. 2001; Amenta and Kil 2004]. Our method can be used to compute the winding number for point clouds as easily as it can for triangle surfaces. This frees up practitioners to use the best surface representation for a given application, or even mix triangles and points. Just as the introduction of the winding number function in [Jacobson et al. 2013] did not seek to smooth or change input triangle meshes, we do not change the surface: instead, we simply answer whether a query point is inside or outside. The winding number represents a surface as a discontinuity, the definition for triangle soups maintains this property and so does our definition for points. Calderon et al. [2014] define morphological operations for Moving Least Squares point set surfaces, citing generalized winding numbers as a possible way to improve inside-outside classification. We show signed distance fields and offset surfaces for point sets, indicating that this is indeed possible. Sanchez et al. [2012] losslessly convert triangle meshes to a distance field that is signed using angle-weighted pseudonormals [Baerentzen and Aanaes 2005]. This requires not just a closed, watertight mesh but

also good mesh-quality for numerically trustworthy normal computation. Fryazinov et al. 2011 [2011] convert triangle meshes to signed distance fields using BSP-trees. In contrast, we unify triangle meshes and implicits for point sets through a consistent definition of their winding numbers.

Point set surfaces are closely related to noisy point cloud surface reconstruction, though with a different objective. State-of-the-art methods [Boltcheva and Lévy 2017; Fuhrmann and Goesele 2014; Kazhdan and Hoppe 2013; Mostegel et al. 2017] are robust to noise, scale, sampling density diversity, and missing data. While our dipole function is similar in shape to the directional derivative of a Gaussian used by others [Fuhrmann and Goesele 2014; Zagorchev and Goshtasby 2012], that function is non-singular and thus non-interpolating. Seversky and Yin [2012] use an inside-outside segmentation for their surface reconstruction. They create a coarse approximation to the surface, then apply ray stabbing to approximate both a parity map for inside-outside, as well as a confidence map using unsigned solid angle. Our method avoids ray stabbing, and thresholds the winding number for surface extraction, rather than using space carving. Employing a tree-algorithm similar to ours, Carr et al. [2001] extract a surface from a point cloud by using radial basis functions to approximate a signed distance function. To break symmetry, this and other methods create extra "offset points" near input points along their normal directions [Ohtake et al. 2003, 2004]. Points and their normal offsets are sometimes referred to as "dipoles" in the surface reconstruction literature [Shalom et al. 2010] In our paper, we use "dipole" to refer to a single oriented point: we do not use offset points, nor inherit their ambiguities and difficulties (see [Shen et al. 2004]).

Connection to Poisson Surface Reconstruction. Kazhdan et al. [2006] propose solving a Laplace equation with jump boundary conditions enforced with a soft constraint, effectively blurring the boundary conditions during discretization, resulting in a discrete Poisson equation ($\Delta u = f$). Stepping away from the particular discretization, solver employed, or soft constraints, the Poisson surface reconstruction problem *is equivalent* to the winding number. Both seek solutions to the Laplace equation with a jump of ±1 across the surface, reproducing the indicator function for solid shapes and more generally the winding number function for overlapping or surfaces with boundaries. We do not blur our input data or boundary conditions and formulate our point-based winding number in the smooth setting. Consequently, we do not need to solve a system of linear equations to recover the reconstructed surface, just a simple summation which we then accelerate. The linear system solution of [Kazhdan et al. 2006] is only known up to a constant shift; later, Kazhdan et al. [2013] pull this isovalue toward zero by adding additional soft constraints, incurring another parameter. The winding number has an exact value of one inside a solid shape and zero outside (e.g., in the limit of point sampling) and the surface neatly follows the ½-isovalue. Our definition of the winding number for points is consistent with this and – armed with local area estimations per point – extracting along the ½-isovalue accurately identifies the surface.

Contemporary methods to Poisson surface reconstruction have focused on the difficulties of extracting the isosurface of an indicator function [Calakli and Taubin 2011; Lu et al. 2017]. Lu et al. formulate a regularized kernel similar to ours and use the fast multipole method for fast summation. However, the regularization introduces a parameter and unnecessary blurring. Our winding number also has a sharp gradient near the extracted surface, but using root finding during polygonization alleviates mesh extraction concerns.

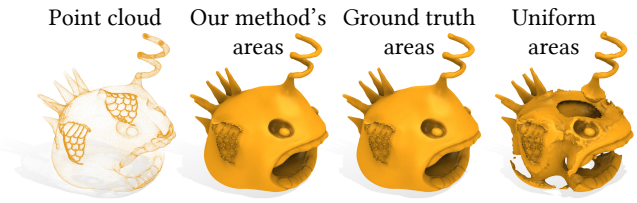

Fig. 5. The area term is crucial to our method. We show the results from point set surfaces (left) using our estimated areas (center-left) the ground-truth area (center-right) and an evenly-distributed area (right).

3 METHOD

The input to our method is either: a set of m oriented points in $\mathbb{R}^3$, represented as a list of positions $\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m\}$ with corresponding unit normal vectors $\{\hat{\mathbf{n}}_1, \hat{\mathbf{n}}_2, \ldots, \hat{\mathbf{n}}_m\}$ *or* a set of m oriented triangles, e.g., represented as a list of vertex position triplets $\{\{\mathbf{v}_{11}, \mathbf{v}_{12}, \mathbf{v}_{13}\}, \ldots, \{\mathbf{v}_{m1}, \mathbf{v}_{m2}, \mathbf{v}_{m3}\}\}$, from which triangle normals may be readily computed. The output of our method is a function $\tilde{w} : \mathbb{R}^3 \to \mathbb{R}$ that quickly approximates the winding number function $w : \mathbb{R}^3 \to \mathbb{R}$ of the input set. While the winding number for oriented triangles was defined by Jacobson et al. [2013], we first must define what we mean by the winding number function for a set of oriented points.

3.1 Point Clouds

Intuitively, the winding number of a continuous surface S evaluated at a query point $\mathbf{q} \in \mathbb{R}^3$ is the sum of signed solid angles subtended by small surface patches. For meshes, patches correspond to triangles and the signed solid angle formula is easily computed via the inverse tangent function (atan2) [van Oosterom and Strackee 1983]. For a smooth surface, patches will be curved. Considering the limit of shrinking patches to points, we arrive at the definition of the winding number as an integral of the *differential* solid angle $d\Omega(\mathbf{q})$ subtended at $\mathbf{q}$:

$$w_S(\mathbf{q}) = \frac{1}{4\pi} \int_S d\Omega(\mathbf{q}). \tag{2}$$

As angles and solid angles are translation invariant, so is the winding number function. We can shift our coordinate system so that the query point $\mathbf{q}$ lies at the origin, where the differential solid angle $d\Omega$ is the differential signed surface area of the surface projected onto the unit sphere:

$$w_S(\mathbf{q}) = w_{S-\mathbf{q}}(\mathbf{0}) = \int_{S-\mathbf{q}} \frac{\mathbf{x} \cdot \hat{\mathbf{n}}}{4\pi \|\mathbf{x}\|^3}\, dx = \int_S \frac{(\mathbf{x}-\mathbf{q}) \cdot \hat{\mathbf{n}}}{4\pi \|\mathbf{x}-\mathbf{q}\|^3}\, dx, \tag{3}$$

where $\mathbf{x}$ is a point on the surface with outward facing unit normal vector $\hat{\mathbf{n}}$.

The integrand $(\mathbf{x}-\mathbf{q})\cdot\hat{\mathbf{n}}/(4\pi\|\mathbf{x}-\mathbf{q}\|^3)$ is commonly found in boundary element method and partial differential equation literature, as it is the normal derivative of the Green's function $G(\mathbf{q},\mathbf{x}) = -1/(4\pi\|\mathbf{x}-\mathbf{q}\|)$ of the Laplace equation in $\mathbb{R}^3$ [Evans 1997, pp. 37]:

$$\frac{(\mathbf{x}-\mathbf{q})\cdot\hat{\mathbf{n}}}{4\pi\|\mathbf{x}-\mathbf{q}\|^3} = \nabla\left(\frac{-1}{4\pi\|\mathbf{x}-\mathbf{q}\|}\right)\cdot\hat{\mathbf{n}} =: G_{\hat{\mathbf{n}}}(\mathbf{q},\mathbf{x}) \tag{4}$$

This function is harmonic ($\Delta G_{\hat{\mathbf{n}}}(\mathbf{q},\mathbf{x}) = 0$) away from $\mathbf{q}$. Because harmonic functions are closed under summation, the winding number function of a surface is harmonic away from the surface, whether the surface is triangulated [Jacobson et al. 2013] or smooth [Narain 2013]. In electrostatics, $G_{\hat{\mathbf{n}}}(\mathbf{q},\mathbf{x})$ appears as the electric potential induced by a *dipole* (two oppositely charged points close together).

If the input oriented points lie on a – perhaps unknown – surface, then we can use them to approximate the integral in Equation (3) as a discrete summation akin to single-point quadrature:

$$w_S(\mathbf{q}) = \int_S \frac{(\mathbf{x}-\mathbf{q})\cdot\hat{\mathbf{n}}}{4\pi\|\mathbf{x}-\mathbf{q}\|^3}\,dx \approx \sum_{i=1}^{m} a_i \frac{(\mathbf{p}_i-\mathbf{q})\cdot\hat{\mathbf{n}}_i}{4\pi\|\mathbf{p}_i-\mathbf{q}\|^3} =: w(\mathbf{q}), \tag{5}$$

where a_i is the geodesic Voronoi area of the point $\mathbf{p}_i$ on the surface S. We have thus arrived at a definition of the winding number function $w:\mathbb{R}^3\to\mathbb{R}$ for a cloud of m oriented points as the area-weighted sum of dipole contributions located at each point in the cloud. At first glance, the a_i coefficients appear to be tunable "weighting parameters", however they actually have definitive meaning and physical units of surface area. They are not arbitrary parameters that might need tuning (in contrast to parameters of [Carr et al. 2001; Fuhrmann and Goesele 2014; Kazhdan et al. 2006; Kazhdan and Hoppe 2013; Shen et al. 2004]). Assuming sufficient regularity during refinement, this approximation will converge to the exact winding number for the smooth surface.

The area a_i associated to each point at $\mathbf{p}_i$ may be known due to the particular point-cloud acquisition process [Fuhrmann and Goesele 2014]. Otherwise, they can be approximated using any number of methods in the literature (e.g., [Belkin et al. 2009]). In our examples, we project each point and its k-nearest neighbors (filtering out those with opposite normals) onto the best-fit plane, and compute that point's associated 2D Voronoi area using Triangle [Shewchuk 1996]. Our results are insensitive to the exact value of k. We use $k = 20$, matching the general range used in point cloud Laplace/mass matrix literature. Accounting for Gaussian curvature using a quadratic fit [Cazals and Pouget 2003] should improve accuracy, but we leave this as an incremental improvement as our method is not too sensitive to area accuracy (see Fig. 5).

For *unoriented* point clouds we can also use the plane to define normals n_i (up to sign, [Berger et al. 2017]).

3.2 Fast approximation

Most applications require evaluating the winding number at a large number of query points. Evaluating w via direct summation over the m dipoles for n queries would result in $O(nm)$ complexity. We now describe a fast approximation based on the observation that while dipoles have large local influence they flatten out quickly ($1/r^2$) far away.

Consider the winding number field of a cluster of m points (see Fig. 6). The field is locally complex, but *zooming out* the field looks like a single, more heavily weighted, dipole. For a far away query point, rather than computing the contribution of all dipoles ($O(m)$ computation), we can compute the contribution of a single pre-constructed *representative* dipole with appropriate orientation and area ($O(1)$ computation).

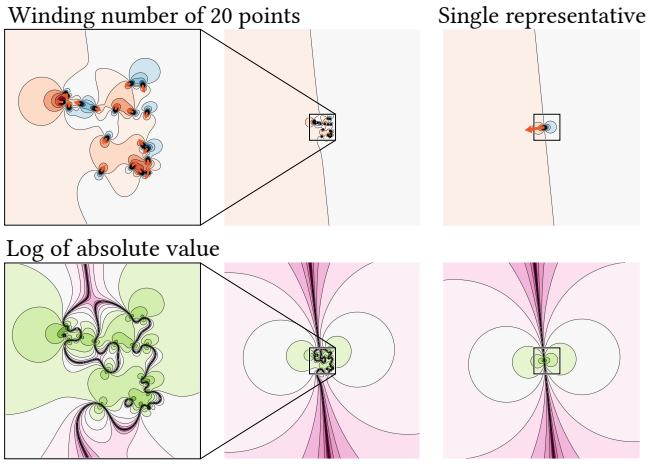

Fig. 6. A cluster of 20 dipoles has an intricate winding number field nearby (left), but far away their function is quite tame (middle) and well approximated by a single, *stronger* dipole (right).

$$w(\mathbf{q}) = \sum_{i=1}^{m} a_i \frac{(\mathbf{p}_i-\mathbf{q})\cdot\hat{\mathbf{n}}_i}{4\pi\|\mathbf{p}_i-\mathbf{q}\|^3} \approx \frac{(\tilde{\mathbf{p}}-\mathbf{q})\cdot\tilde{\mathbf{n}}}{4\pi\|\tilde{\mathbf{p}}-\mathbf{q}\|^3} =: \tilde{w}(\mathbf{q}) \tag{6}$$

$$\tilde{\mathbf{n}} = \sum_{i=1}^{m} a_i\hat{\mathbf{n}}_i, \quad \tilde{\mathbf{p}} = \frac{\sum_{i=1}^{m} a_i\mathbf{p}_i}{\sum_{i=1}^{m} a_i}, \tag{7}$$

where $\tilde{\mathbf{n}}$, and $\tilde{\mathbf{p}}$ are the (non-unit) total area-weighted normal and area-weighted average position, respectively. The formula for approximation function $\tilde{w}$ for the entire cluster mimics the formula for a single dipole. The only modification is that length of the normal vector $\tilde{\mathbf{n}}$ acts as the collective area or strength.

This works for query points far from all of the input points, but, presumably, most query points will lie near some part of the input point cloud.

We can apply this approximation in a divide and conquer fashion using a tree data structure to separate points into smaller and smaller clusters with more and more accurate single-dipole approximations [Barnes and Hut 1986] (see Fig. 7). We provide a pseudocode implementation in Algorithm 1.

To achieve $O(\log m)$ complexity for a single evaluation, we build a bounding volume hierarchy (e.g., using an octree or axis-aligned bounding-boxes).

In Algorithm 1, we assume that each node of the tree contains an approximation of the winding number $\tilde{w}$ for all dipoles contained in the node's bounding cell. We now turn to the construction of this approximation, known in the n-body literature as the "far field expansion."

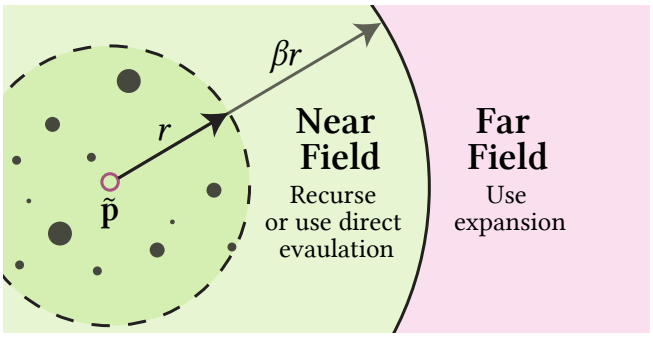

Fig. 7. Our spatial partitioning separates near and far fields, recursively.

Algorithm 1: Fast Approximation of Winding Number FASTWN(**q**,tree)

Inputs:
- **q** Query point in $\mathbb{R}^3$
- tree Root of bounding volume hierarchy for points/triangles
- β accuracy parameter

Outputs: scalar winding number of all elements in tree at **q**

```
// tree.p: center of tree's winding number approximation, tree.w̃
// tree.r: maximum distance from tree.p to any of its elements
if ||q − tree.p|| > β ∗ tree.r then
    // q is sufficiently far from all elements in tree
    return tree.w̃(q)
else
    val ← 0
    if tree has no children then
        // q is nearby; use direct sum for tree's elements
        for each point/triangle e in tree do
            // w_e: area-weighted dipole or solid angle
            val += w_e(q)
    else
        for each child of tree do
            // Recursive call
            val +=FASTWN(q, child)
    return val
```

3.2.1 Higher-order approximation. The accuracy of our approximation for a cluster of dipoles can be further improved, allowing an even tighter definition of "nearby" in the algorithm.

The representation of a sum of dipoles as a single representative dipole in Equation (6) can be understood as the first term in a Taylor series expansion about the center of mass $\tilde{\mathbf{p}}$. We can improve the accuracy by adding more terms to the sum.

Recall: The Taylor expansion for a function $f(\mathbf{x}) : \mathbb{R}^3 \to \mathbb{R}$ at a point $\tilde{\mathbf{p}}$ is given as

$$f(\mathbf{x}) \approx f(\tilde{\mathbf{p}}) + (\mathbf{x}-\mathbf{p}) \cdot \nabla f(\tilde{\mathbf{p}}) + \tfrac{1}{2}((\mathbf{x}-\mathbf{p}) \otimes (\mathbf{x}-\mathbf{p})) \cdot \nabla^2 f(\tilde{\mathbf{p}}) + \ldots,$$

where $\nabla f : \mathbb{R}^3 \to \mathbb{R}^3$ and $\nabla^2 f : \mathbb{R}^3 \to \mathbb{R}^{3\times 3}$ are the gradient and the Hessian of f, producing vectors and symmetric two-tensors respectively. We use $\otimes$ and $\cdot$ to mean tensor outer and inner products, respectively.

We start by considering the Taylor expansion for a single unit-area dipole function, taking the expansion *with respect to the dipole position* **x** at what will be the center point $\tilde{\mathbf{p}}$:

$$\begin{aligned} G_{\hat{\mathbf{n}}}(\mathbf{q},\mathbf{x}) &= \hat{\mathbf{n}} \cdot \nabla G(\mathbf{q},\mathbf{x}) && (8)\\ &= \hat{\mathbf{n}} \cdot \nabla G(\mathbf{q},\tilde{\mathbf{p}}) && (9)\\ &\quad + ((\mathbf{x}-\tilde{\mathbf{p}}) \otimes \hat{\mathbf{n}}) \cdot \nabla^2 G(\mathbf{q},\tilde{\mathbf{p}}) && (10)\\ &\quad + \tfrac{1}{2}((\mathbf{x}-\tilde{\mathbf{p}}) \otimes (\mathbf{x}-\tilde{\mathbf{p}}) \otimes \hat{\mathbf{n}}) \cdot \nabla^3 G(\mathbf{q},\tilde{\mathbf{p}}) && (11)\\ &\quad + \ \textit{higher order terms}, && (12) \end{aligned}$$

where ∇G, $\nabla^2 G$, and $\nabla^3 G$ are the gradient, the Hessian and the three-tensor of third derivatives of the Green's function for the Laplace equation. Formulae for each derivative are located in Appendix A, for reference.

The derivatives themselves do not depend on the dipole's actual center **x**, so we can apply the Taylor expansion to the sum of m area-weighted dipoles to get an approximation of their collective winding number. We replace **x** and **n** in Equations (8-12) with summations over each point $\mathbf{p}_i$ and its normal $\hat{\mathbf{n}}_i$ to obtain our fast winding number approximation $\tilde{w}(\mathbf{q})$:

$$\begin{aligned} w(\mathbf{q}) &\approx \left(\sum_{i=1}^{m} a_i \hat{\mathbf{n}}_i\right) \cdot \nabla G(\mathbf{q},\tilde{\mathbf{p}}) && (13)\\ &\quad + \left(\sum_{i=1}^{m} a_i (\mathbf{p}_i - \tilde{\mathbf{p}}) \otimes \hat{\mathbf{n}}_i\right) \cdot \nabla^2 G(\mathbf{q},\tilde{\mathbf{p}}) && (14)\\ &\quad + \tfrac{1}{2}\left(\sum_{i=1}^{m} a_i (\mathbf{p}_i - \tilde{\mathbf{p}}) \otimes (\mathbf{p}_i - \tilde{\mathbf{p}}) \otimes \hat{\mathbf{n}}_i\right) \cdot \nabla^3 G(\mathbf{q},\tilde{\mathbf{p}}) && (15)\\ &=: \tilde{w}(\mathbf{q}). && (16) \end{aligned}$$

The coefficients in front of ∇G, $\nabla^2 G$ and $\nabla^3 G$ terms are vectors, two-tensors, and three-tensors respectively. These are *precomputed* during bounding volume tree construction and stored for each non-empty node of the tree. The expansion could continue with higher order terms indefinitely, in practice we saw diminishing returns beyond the second-order expansion above.

If we use pth order expansion, we add $O(3^p)$ complexity to our evaluation. The overall, amortized complexity – assuming a well formed tree – for pre-computation and evaluation at n queries is $O(3^p m \log m + 3^p n \log m)$.

Our method falls into the Barnes-Hut, tree-algorithm family. The Fast Multipole Method (FMM) is closely related and achieves $O(n)$ performance, but precomputation is significantly more involved and complexity results rely on very strict assumptions about the input point distribution and the query distribution. We implemented FMM and found overhead to be excessively costly and that the typical point set surface extraction does not appear to meet the input assumptions for good performance.

In comparison to many Fast Multiple Method (FMM) libraries (e.g., FMMLIB3D [Greengard and Gimbutas 2017]), we *must* (and do) separate precomputation from evaluation for applications such as iso-surface polygonization using a continuation method [Wyvill et al. 1986] (see Fig. 9). This is in contrast to the prototypical application of FMM to n-body gravitational systems, where the evaluation points are the same as the sources and evaluation is done only once.

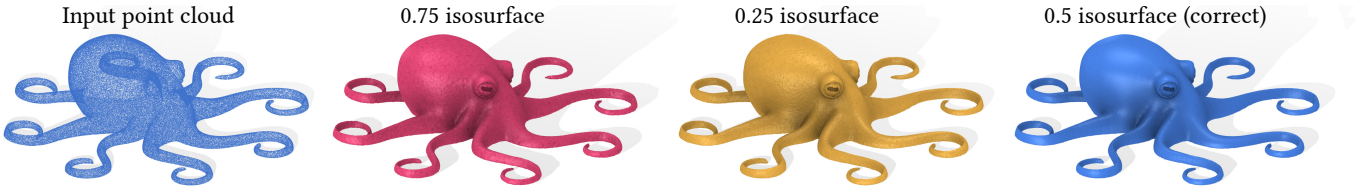

Fig. 8. Input point cloud (left), wrong isovalues produce close surfaces (middle). The correct isovalue from the smooth theory is ½ and the polygonized isosurface of our fast approximation agrees (right).

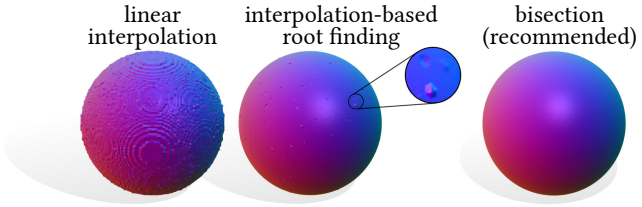

Fig. 9. Many implicit surface polygonizers use linear interpolation or linear-interpolation based root-finding. However, the winding number for points has asymptotic shape near each dipole, leading to poor surface quality. Iterative bisection, instead, extracts clean isosurfaces.

3.3 Triangle Soups

Along with the introduction of the generalized winding number for triangle meshes, Jacobson et al. [2013] propose a divide and conquer algorithm for efficient evaluation. Their method is also based on a bounding volume hierarchy, but differs from our design in two major ways: 1) it is *exact* while ours is approximate, and 2) its computational complexity strongly coupled to the connectivity of the input mesh. In the worst case, for m triangles and n query points their method reduces to the direct sum and performs with $O(nm)$ complexity. Instead, we now describe how to leverage our fast winding number approximation for triangle soups.

Elevating our fast approximation algorithm to input triangle soups turns out to be straightforward. Referring to Algorithm 1, we will use a triangle's solid angle (à la [Jacobson et al. 2013]) for evaluations of w_e in the direct sum, but need to define our approximations $\tilde{w}$ for a cluster of triangles.

Like any surface, the solid angle of a single flat triangle t is the integral of the dipole over its area:

$$\Omega_t(\mathbf{q}) = \int_t \nabla G_{\hat{\mathbf{n}}_t}(\mathbf{q}, \mathbf{x}) \cdot \hat{\mathbf{n}} \; dA, \tag{17}$$

thus, the contribution of a triangle can be interpreted as a sum of point contributions.

Differentiation and integration associate, so the summations over points in the coefficients of the Taylor expansion in Equation (13) are replaced with summations over triangles, each summand expanding into an integral over the corresponding triangle:

$$w(\mathbf{q}) \approx \left(\sum_{t=1}^{m} \int_t \hat{\mathbf{n}}_t \; dA \right) \cdot \nabla G(\mathbf{q}, \tilde{\mathbf{p}}) \tag{18}$$

$$+ \left(\sum_{t=1}^{m} \int_t (\mathbf{x} - \tilde{\mathbf{p}}) \otimes \hat{\mathbf{n}}_t \; dA \right) \cdot \nabla^2 G(\mathbf{q}, \tilde{\mathbf{p}}) \tag{19}$$

$$+ \tfrac{1}{2} \left(\sum_{t=1}^{m} \int_t (\mathbf{x} - \tilde{\mathbf{p}}) \otimes ((\mathbf{x} - \tilde{\mathbf{p}}) \otimes \hat{\mathbf{n}}_t) \; dA \right) \cdot \nabla^3 G(\mathbf{q}, \tilde{\mathbf{p}}) \tag{20}$$

$$=: \tilde{w}(\mathbf{q}). \tag{21}$$

The first coefficient $\sum_{t=1}^{m} \int_t \hat{\mathbf{n}}_t \; dA$ is simply the total area-weighted normal over the triangles, mimicking the intuition that a cluster of triangles are being replaced with a single, larger triangle. All terms have closed form expressions found in Appendix B.

Implementation details. The broad implementation of our algorithm is agnostic to the bounding volume hierarchy used. In practice, we use an octree as a bounding volume for point clouds and an axis-aligned bounding box tree for triangle soups. The axis-aligned bounding box tree allows us to avoid clipping triangles. In the case of points, our algorithm was fastest when we set no limit to the depth of our octree – any cell containing more than two points has children.

For point clouds, we use a continuation method [Wyvill et al. 1986] for voxelization and isosurface extraction. The winding number function is smooth and very flat away from the input points, but each point introduces a dipole singularity. Computing values at each grid corner and relying on linear interpolation to find the surface (as many "marching cubes" [Lorensen and Cline 1987]) will produce visible pockmarks, isolated small dents and bumps like the eyes on a potato (see Fig. 9). Root finding [Wyvill et al. 1986] avoids this and finds a more accurate surface. This method begins with a series of "seed cubes" on the surface then incrementally expands to neighbouring cubes which contain the isosurface. A standard problem in continuation polygonizers is finding initial seed cubes. However, since our surface is defined by point samples which lie on the surface, we use them as the set of seed points.

For closed surfaces in the smooth setting, the winding number of the interior is exactly one and the exterior is zero: the value of $\frac{1}{2}$ neatly follows the surface. For an area-weighted point set, the $\frac{1}{2}$-level-set converges to the underlying surface in the limit. As such, we use an isovalue of $\frac{1}{2}$ for point set surface polygonizations in our examples (see Fig. 8).

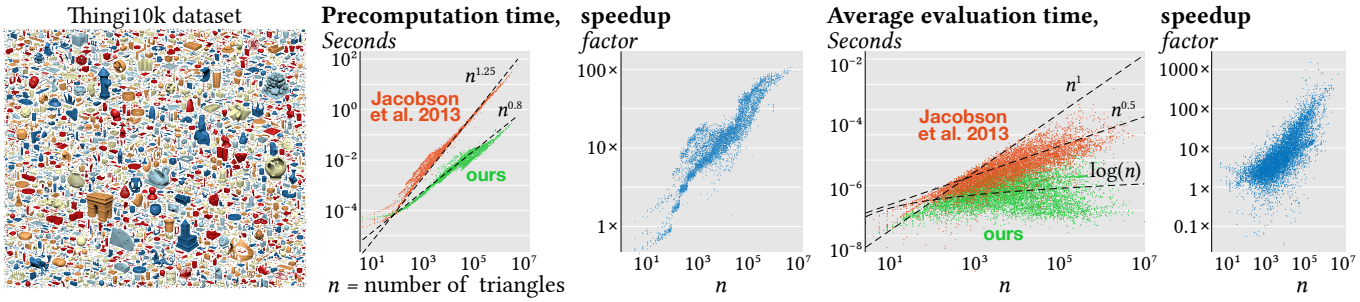

Fig. 10. We exhaustively test the performance of our method on ten thousand models [Zhou and Jacobson 2016]. Compared to the divide and conquer method introduced by Jacobson et al. [2013] our method is significantly faster in both precomputation time (up to 100× faster) and average evaluation time (up to 1000× faster). Both stages empirically demonstrate asymptotically better performance.

4 EXPERIMENTS & RESULTS

We implemented our area estimation and fast evaluation in C++ using `std::thread` and Intel TBB for parallelization, utilizing libigl [Jacobson et al. 2017] and Eigen [Guennebaud et al. 2010] for geometry processing and linear algebra routines, respectively. Performance timings are conducted on a Intel i7-6950X CPU with 128GB of DDR4 memory.

In Fig. 1, we demonstrate the versatility of our method both in terms of output and inputs. We use our fast winding number evaluation on clean meshes, triangle soups and point clouds of varying regularity. Using this evaluation to determine what is inside or outside the given shape regardless of representation, we may output a clean isosurface, a voxelization, 3D printer path instructions or signed distance fields.

We demonstrate simply signing an unsigned distance field with the winding number, as a drop-in robust replacement for pseudonormal testing or ray stabbing (see Fig. 4). This preserves the absolute value of the distance at the cost of creating a discontinuity along the winding number threshold level-set. In contrast, Xu et al. [Xu and Barbič 2014] create a continuous (discretized) signed distance field by effectively morphologically *closing* the input. This comes at the cost of introducing a parameter. We view our methods as complementary.

In Fig. 10, we compute winding number values for each of the ten thousand models in the Thingi10k dataset [Zhou and Jacobson 2016]. We record precomputation and average evaluation time (over a 50^3 lattice of queries in the bounding box of the shape) for our fast approximation and the divide and conquer method of Jacobson et al. [2013]. Their method has a considerably more involved precomputation stage; consequently, their method is up to 100× slower for precomputation alone. While our method's precomputation empirically follows a slightly sublinear trend, the precomputation of Jacobson et al. [2013] is superlinear. This is explainable because their performance is bound by the number of "boundaries" created while slicing up the input model into a bounding volume hierarchy (under regularity assumptions, curve boundaries on a surface should have $O(m^{\frac{1}{2}})$ complexity). While their method appears to achieve sublinear average evaluation for a subset of the dataset, there is also a consistent linear $O(n)$ subset. In constrast, our average evaluation time more neatly follows a logarithmic trend, as expected.

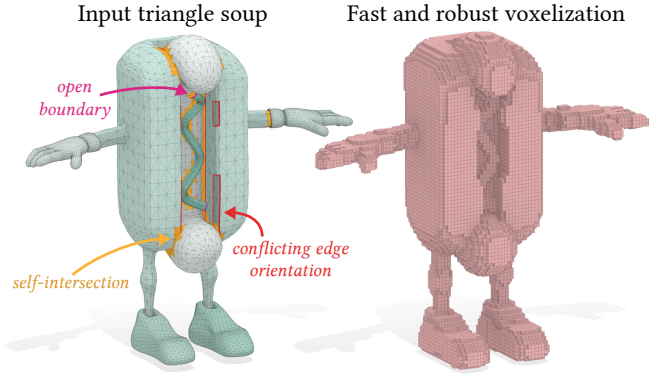

Fig. 11. Voxelization requires robust inside-outside segmentation; our fast evaluation takes 0.021 secs for precomputation and 0.184 secs to evaluate the winding number at 100^3 voxel centers. In contrast, the divide and conquer method of Jacobson et al. [2013] takes 0.129 secs of precomputation and 9.13 secs for the million queries.

For our timings and all other experiments (unless specified), we set our accuracy parameter to $\beta = 2$ for triangles and $\beta = 2.3$ for points. We chose these values to achieve 10^{-3} and 10^{-2} root mean squared error for triangles and points respectively. These values provide a good trade-off between accuracy (see Fig. 4) and speed (see Fig. 10). Even a small value of $\beta = 2$ produces the identical voxelization as the exact winding number (i.e., when thresholding at $1/2$).

In Fig. 11, we compute a voxelization of a typical triangle soup from virtual 3D modeling. The surface is composed of multiple connected components that intersect each other with spurious inconsistent orientations and open boundaries. In the inset figure we show how the max and average error follow the β parameter on this hotdog. As β is

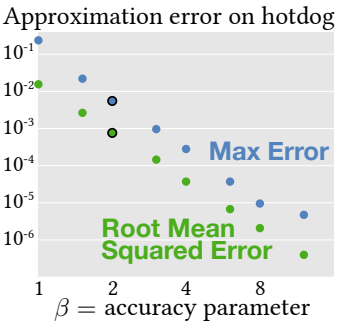

Fig. 12. Holey, non-manifold triangle meshes $\mathcal{A}$ and $\mathcal{B}$ each induce a generalized winding number function on the other's faces, useful for boolean operations such as $\mathcal{A} \setminus \mathcal{B}$.

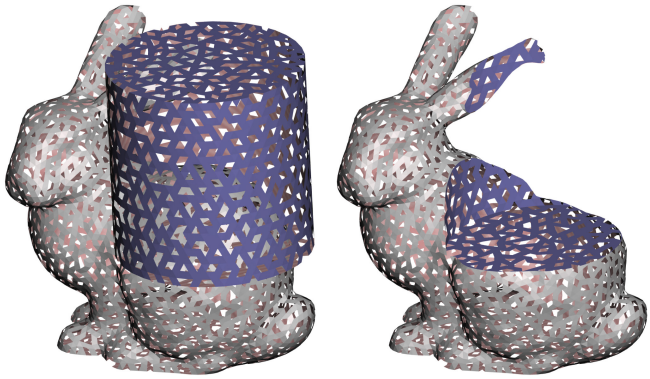

Fig. 13. A holey cylinder is subtracted from a holey bunny.

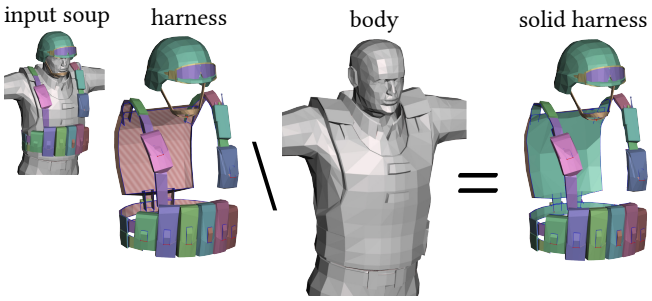

Fig. 14. The harness and accessories of this army man were modeled as mere façades: all have missing backs and are simply placed atop the man's body (backfaces shown with red stripes). The body itself also has mesh defects, but despite these problems, we subtract it from the harness to create effectively solid pieces.

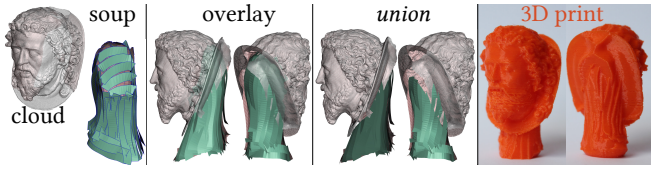

Fig. 15. A point cloud of St. John the Baptist's severed head and a "pedestal" created out of a soup of ribbons are unioned each induce a continuous winding number field. We quickly approximate this during their Boolean union and send the result to be 3D printed. Credit for the head model goes to Geoffrey Marchal [2015].

increased, more and more of the computation moves from far field expansions to near field recursion and direct summation, increasing runtime. However, increasing β rapidly decreases the error. For example, if we choose β so large that our runtimes are equivalent to the divide and conquer method of Jacobson et al. [2013], then our approximation error is already below machine epsilon for single-precision floats (10^{-7}).

For a measure of quality, we evaluated our fast winding number on the Thingi10K dataset, using a 100^3 voxel grid. Over 50% of shapes had exactly zero misclassified voxels using our approximation. The median number of misclassifications is therefore also zero, and the average is 363 (or, 0.0363%). The average root mean squared error of the fast winding number value is $8e{-}3$.

Boolean composition operations are a standard tool in the shape modeling toolbox, and widely used with mesh representations. Geometric mesh Boolean techniques generally operate by computing intersections of the mesh elements. If there are holes in the mesh in the intersection regions, then the Boolean operation is mathematically ill-posed. However, given two "holey" surfaces such as those in Fig. 12, one can easily imagine what the Boolean result might be. Our fast winding numbers provide a way to implement such an operation. Rather than intersect the triangles, we trim each mesh against the winding field of the other [Jacobson 2016]. The result follows our intuition – where triangles exist, the cut lines follow them, and where they do not, something sensible happens.

In Fig. 13, we take the difference of two very holey meshes. Triangle edges are not stitched in the few areas where they do touch. While this is possible, it is insignificant given the large number of adjacent holes. This notion of surfaces being closed "within a tolerance" is widely accepted in all industry-standard Computer-Aided Design (CAD) tools, where B-Rep "solids" are formed from trimmed NURBS patches. In the general case it is mathematically impossible for two adjacent trimmed patches to exactly meet along a 3D curve. Despite these gaps, the geometry can be rendered, simulated, and manufactured.

Fig. 14 illustrates another example representative of the kinds of models commonly found in the film and games industries, where many shape details are open shells that simply intersect. Visually this assembly forms an apparent solid suitable for display, but geometrically it is a disaster for many downstream applications. We can close off the large open regions of the equipment harness by subtracting the body of the underlying character. The geometry in areas away from the intersections remains unmodified. We can now, for example, render this harness independently in an equipment-selection screen, apply deformable-body simulation while colliding it with the character's clothing, or 3D-print a separate removable plastic harness during a figurine-design workflow.

The examples above illustrate an interesting approach to mesh modeling, where we no longer depend on having closed, well-defined volumes. Each mesh "part" has an associated winding field that interpolates the existing geometry, while doing something sensible in any soupy areas. Operations can be formulated using whichever of these representations is most relevant. We can also incorporate our point-based winding field into this framework, allowing for hybrid operations between point-set surfaces and meshes (see Fig. 15).

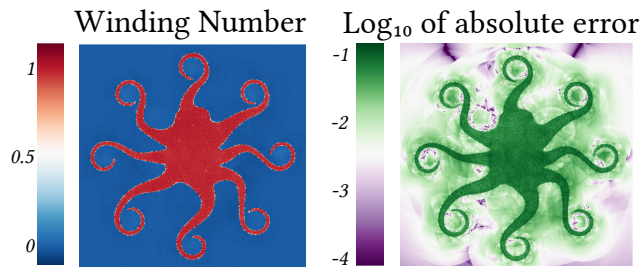

Fig. 16. Here we take a 2D slice through the 3D winding number field of the octopus model found in Fig. 8

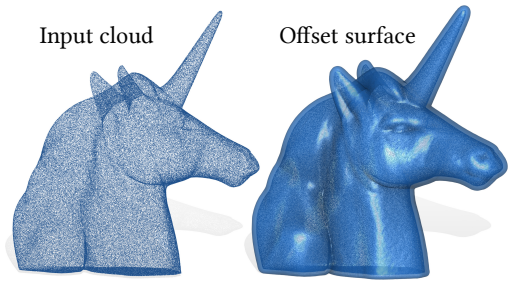

Fig. 17. By signing the distance field, we can take an offset surface that strictly contains the point cloud.

In Fig. 5 we compare our point-based definition of the winding number using approximated areas and ground truth, demonstrating robustness. From our area estimation we are able to extract the point surface from a fixed isovalue of $\frac{1}{2}$.

For point cloud timings on a laptop, refer to the example in Fig. 16. On a point cloud of 300,000 dipoles, with 250,000 queries, the fast winding number took 0.578 seconds for the precomputation and an average evaluation of 3.01 micoseconds per query. In contrast, the direct evaluation took an average of 4,080 microseconds per query. The maximum approximation error was less than 0.1, with a root mean squared error of 0.0191, which was sufficient to produce the isosurface in Fig. 8.

Using the inside-outside testing of the winding number, we are able to bestow sign upon an unsigned distance field. This can be done for both triangles and points. In Fig. 1 we sign a distance field of a soupy triangle mesh, and in Fig. 17 we use signed distance to create an offest surface for a point cloud.

With our fast winding numbers for points, we are able to 3D print directly from point clouds and soupy meshes – no meshing or remeshing required (see Fig. 3). This allows us to save on both memory and computation. We note that this slicing step is a small fraction – less than 10% – of the print preparation time. These slices are then passed to the toolpathing software. At the level of accuracy of 3D print heads, determining the print paths from both mesh and point cloud winding number fields is essentially lossless – the printing of sharp corners and small features will be indistinguishable from those generated by slicing triangles.

5 LIMITATIONS & CONCLUSION

The classic and generalized winding numbers require orientation. This applies to the soups and clouds we consider, too. For unoriented soups, automatic methods for determining a consistent orientation exist (e.g., [Borodin et al. 2004; Takayama et al. 2014]). For unoriented point clouds, off-the-shelf normal estimation methods (e.g., MeshLab [Cignoni et al. 2008]) complement our method well: a few incorrectly oriented points will only degrade the overall winding number field locally.

As mentioned in the introduction, our method does not directly fit the requirements for surface reconstruction of noisy points. Our point set surfaces are interpolating: dipoles are $\pm\infty$ on either side, so any isovalue will pass through them (see Fig. 8). Early experiments show that treating noisy points as random variables drawn from Gaussian distributions is a promising approach (e.g., using Monte Carlo sampling to generate more points creates a *smoothed* winding number). Similarly, we show results for point samplings of complete surfaces *without boundaries*. For "partial surfaces", the level-set isovalue should be adjusted according to total Gaussian curvature (analog of a 2D curve's *turning number*). Fortunately, curvature estimation for point clouds is well studied [Belkin et al. 2009; Cazals and Pouget 2003]. It would also be interesting to consider using interval trees to improve isosurface extraction [Cignoni et al. 1997].

Our fast evaluation for triangle soups and point clouds will not only improve the performance of all applications already relying on generalized winding numbers, but opens the door to new opportunities. We foresee a rich topic of research exploring how to push raw, unstructured geometric data like soups and clouds *farther* along the geometry processing pipeline.

ACKNOWLEDGMENTS

This work is funded in part by NSERC Discovery Grants (RGPIN2017-05235 & RGPAS-2017-507938 & RGPIN-2017-05524), NSERC DAS (RGPAS-2017-507909), Connaught Funds (NR2016-17), the Canada Research Chairs Program, and a gift by Adobe Systems Inc. We thank Michael Tao for the derivations found in Appendix A and B, as well as Sarah Kushner for help with figure creation.

REFERENCES

M Alexa, Johannes Behr, D Cohen-Or, Shachar Fleishman, David Levin, and Cláudio T Silva. 2001. Point Set Surfaces. *IEEE Visualization* (2001), 21–537.

Nina Amenta and Yong Joo Kil. 2004. Defining point-set surfaces. *ACM Trans. Graph.* (2004).

James L. Andrews. 2013. *User-Guided Inverse 3D Modeling.* Ph.D. Dissertation. UC Berkeley.

J. Andreas Baerentzen and Henrik Aanaes. 2005. Signed Distance Computation Using the Angle Weighted Pseudonormal. *IEEE TVCG* (2005).

J. Barnes and P. Hut. 1986. A hierarchical O(N log N) force-calculation algorithm. *Nature* (1986).

Mikhail Belkin, Jian Sun, and Yusu Wang. 2009. Constructing Laplace Operator from Point Clouds in Rd. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '09)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1031–1040. http://dl.acm.org/citation.cfm?id=1496770.1496882

Matthew Berger, Andrea Tagliasacchi, Lee M. Seversky, Pierre Alliez, Gaël Guennebaud, Joshua A. Levine, Andrei Sharf, and Claudio T. Silva. 2017. A Survey of Surface Reconstruction from Point Clouds. *Comput. Graph. Forum* (2017).

Guy E Blelloch and Girija J Narlikar. 1994. A practical comparison of N-body algorithms. *Parallel Algorithms* (1994).

Dobrina Boltcheva and Bruno Lévy. 2017. Surface reconstruction by computing restricted Voronoi cells in parallel. *Computer-Aided Design* (2017).

Pavel Borodin, Gabriel Zachmann, and Reinhard Klein. 2004. Consistent Normal Orientation for Polygonal Meshes. In *Computer Graphics International 2004 (CGI 2004)*. 18–25.
Fatih Calakli and Gabriel Taubin. 2011. SSD - Smooth Signed Distance Surface Reconstruction. *Comput. Graph. Forum* 30, 7 (2011), 1993–2002.
Stéphane Calderon and Tamy Boubekeur. 2014. Point morphology. *ACM Transactions on Graphics* (2014).
Jonathan C Carr, Richard K Beatson, Jon B Cherrie, Tim J Mitchell, W Richard Fright, Bruce C McCallum, and Tim R Evans. 2001. Reconstruction and representation of 3D objects with radial basis functions. *Siggraph* (2001), 67–76.
F. Cazals and M. Pouget. 2003. Estimating differential quantities using polynomial fitting of osculating jets. In *Proc. SGP*. 177–187.
Angel X. Chang, Thomas A. Funkhouser, Leonidas J. Guibas, Pat Hanrahan, Qi-Xing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. 2015. ShapeNet: An Information-Rich 3D Model Repository. *CoRR* (2015). arXiv:1512.03012
Paolo Cignoni, Massimiliano Corsini, and Guido Ranzuglia. 2008. MeshLab: an Open-Source 3D Mesh Processing System. *ERCIM News* 73 (April 2008), 45–46. http://vcg.isti.cnr.it/Publications/2008/CCR08
Paolo Cignoni, Paola Marino, Claudio Montani, Enrico Puppo, and Roberto Scopigno. 1997. Speeding Up Isosurface Extraction Using Interval Trees. *IEEE Trans. Vis. Comput. Graph.* (1997).
Fang Da, David Hahn, Christopher Batty, Chris Wojtan, and Eitan Grinspun. 2016. Surface-only liquids. *ACM Transactions on Graphics* (2016).
Martin de Lasa. 2018. Autodesk. personal communication. (2018).
Olivier Dionne and Martin de Lasa. 2013. Voxelization using generalized winding numbers. (2013). https://vimeo.com/72468198
Olivier Dionne and Martin de Lasa. 2014. Geodesic Binding for Degenerate Character Geometry Using Sparse Voxelization. *IEEE Trans. Vis. Comput. Graph.* (2014).
Lawrence C. Evans. 1997. *Partial Differential Equations* (1 ed.). 37 pages.
Oleg Fryazinov, Alexander A Pasko, and Valery Adzhiev. 2011. BSP-fields - An exact representation of polygonal objects by differentiable scalar fields based on binary space partitioning. *Computer-Aided Design* (2011).
Simon Fuhrmann and Michael Goesele. 2014. Floating Scale Surface Reconstruction. *ACM Trans. Graph.* 33, 4, Article 46 (July 2014), 11 pages. https://doi.org/10.1145/2601097.2601163
Leslie Greengard and Zydrunas Gimbutas. 2017. fmmlib3d: Fast Multipole Method (FMM) library in R^3. (2017). https://github.com/zgimbutas/fmmlib3d Version 1.2.1.
L. Greengard and V. Rokhlin. 1997. A Fast Algorithm for Particle Simulations. *J. Comput. Phys.* 135, 2 (1997), 280 – 292. https://doi.org/10.1006/jcph.1997.5706
Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. (2010). http://eigen.tuxfamily.org.
A Ichim, P Kadlecek, L Kavan, and M Pauly. 2017. Phace: Physics-based Face Modeling and Animation. *ACM Trans. Graph.* (2017).
Peter Ilbery, Luke Kendall, Cyril Concolato, and Michael McCosker. 2013. Biharmonic Diffusion Curve Images from Boundary Elements. *ACM Trans. Graph.* (2013).
Alec Jacobson. 2016. Boolean Operations using Generalized Winding Numbers. *CoRR* abs/1601.07953 (2016). arXiv:1601.07953 http://arxiv.org/abs/1601.07953
Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. 2013. Robust Inside-Outside Segmentation using Generalized Winding Numbers. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)* 32, 4 (2013), 33:1–33:12.
Alec Jacobson, Daniele Panozzo, et al. 2017. libigl: A simple C++ geometry processing library. (2017). http://libigl.github.io/libigl/.
Chiyu Jian and Philip Marcus. 2017. Hierarchical Detail Enhancing Mesh-Based Shape Generation with 3D Generative Adversarial Network. *CoRR* (2017). arXiv:1709.07581
Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. 2006. Poisson surface reconstruction. In *Proc. SGP*. 61–70.
Michael Kazhdan and Hugues Hoppe. 2013. Screened poisson surface reconstruction. *ACM Transactions on Graphics (TOG)* 32, 3 (2013), 29.
Binh Huy Le and Zhigang Deng. 2017. Interactive cage generation for mesh deformation. *I3D* (2017).
William E Lorensen and Harvey E Cline. 1987. Marching cubes - A high resolution 3D surface construction algorithm. *Siggraph* (1987), 163–169.
Wenjia Lu, Zuoqiang Shi, Bin Wang, and Jian Sun. 2017. Gauss Surface Reconstruction. In *Geometric Modeling and Processing*.
Geoffrey Marchal. 2015. Head St Batist-decimated. (May 2015). https://sketchfab.com/models/3d3115c7db08466abbbb0dcaac2c962e
A.L.F Meister. 1769. Generalia de genesi figurarum planarum et inde pendentibus earum ajfectionibus. *Novi Comm. Soc. Reg. Scient. Gotting.* (1769), 144–180+9 plates.
Christian Mostegel, Rudolf Prettenthaler, Friedrich Fraundorfer, and Horst Bischof. 2017. Scalable Surface Reconstruction from Point Clouds with Extreme Scale and Density Diversity. *CVPR* (2017).
T. M. Murali and Thomas A. Funkhouser. 1997. Consistent Solid and Boundary Representations from Arbitrary Polygonal Data. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics (I3D '97)*. ACM, New York, NY, USA, 155–ff. https://doi.org/10.1145/253284.253326
Rahul Narain. 2013. The harmonicity of the generalized winding number. (2013). http://r-to-the-n.tumblr.com/post/52752267384/
Fakir S. Nooruddin and Greg Turk. 2003. Simplification and Repair of Polygonal Models Using Volumetric Techniques. *IEEE Transactions on Visualization and Computer Graphics* 9, 2 (April 2003), 191–205. https://doi.org/10.1109/TVCG.2003.1196006
Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. 2003. Multi-Level Partition of Unity Implicits. *ACM Trans. Graph.* (2003).
Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. 2004. Ridge-Valley Lines on Meshes via Implicit Surface Fitting. *ACM Trans. Graph.* (2004).
Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. 2008. Diffusion curves - a vector representation for smooth-shaded images. *ACM Trans. Graph.* (2008).
Mathieu Sanchez, Oleg Fryazinov, and Alexander Pasko. 2012. Efficient Evaluation of Continuous Signed Distance to a Polygonal Mesh. In *Proc. SCCG*.
L. M. Seversky and L. Yin. 2012. A Global Parity Measure for Incomplete Point Cloud Data. *Comput. Graph. Forum* 31, 7pt1 (Sept. 2012), 2097–2106. https://doi.org/10.1111/j.1467-8659.2012.03202.x
Shy Shalom, Ariel Shamir, Hao Zhang 0002, and D Cohen-Or. 2010. Cone carving for surface reconstruction. *ACM Trans. Graph.* (2010).
C Shen, J.F. O'Brien, and J.R. Shewchuk. 2004. Interpolating and approximating implicit surfaces from polygon soup. 23, 3 (2004), 896–904.
Jonathan Richard Shewchuk. 1996. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*. Lecture Notes in Computer Science, Vol. 1148. Springer-Verlag, 203–222.
Timothy Sun, Papoj Thamjaroenporn, and Changxi Zheng. 2014. Fast multipole representation of diffusion curves and points. *ACM Transactions on Graphics* (2014).
Xin Sun, Guofu Xie, Yue Dong, Stephen Lin, W Xu, Wencheng Wang, Xin Tong, and Baining Guo. 2012. Diffusion curve textures for resolution independent texture mapping. *ACM Transactions on Graphics (TOG)* (2012).
Kenshi Takayama, Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. 2014. A Simple Method for Correcting Facet Orientations in Polygon Meshes Based on Ray Casting. *Journal of Computer Graphics Techniques* (2014).
Jasper van de Gronde. 2010. *A High Quality Solver for Diffusion Curves*. Ph.D. Dissertation. University of Groningen.
A van Oosterom and J Strackee. 1983. The Solid Angle of a Plane Triangle. *IEEE Trans. Biomedical Engineering* 30, 2 (1983).
He Wang, Kirill A. Sidorov, Peter Sandilands, and Taku Komura. 2013. Harmonic Parameterization by Electrostatics. *ACM Trans. Graph.* (2013).
G Wyvill, C McPheeters, and B Wyvill. 1986. Data structure for soft objects. *The Visual Computer* (1986).
Hongyi Xu and Jernej Barbič. 2014. Signed Distance Fields for Polygon Soup Meshes. *Graphics Interface 2014* (2014).
Lyubomir Zagorchev and A Ardeshir Goshtasby. 2012. A Curvature-Adaptive Implicit Surface Reconstruction for Irregularly Spaced Points. *IEEE Trans. Vis. Comput. Graph.* (2012).
Xinxin Zhang and Robert Bridson. 2014. A PPPM fast summation method for fluids and beyond. *ACM Transactions on Graphics* (2014).
Kaichi Zhou, Eugene Zhang, Jirí Bittner, and Peter Wonka. 2008. Visibility-driven Mesh Analysis and Visualization through Graph Cuts. *IEEE Transactions on Visualization and Computer Graphics* 14 (2008).
Qingnan Zhou and Alec Jacobson. 2016. Thingi10K: A Dataset of 10, 000 3D-Printing Models. *CoRR* abs/1605.04797 (2016). arXiv:1605.04797 http://arxiv.org/abs/1605.04797

A DIPOLE DERIVATIVES

For convenience, we list the gradient and Hessian of a dipole with center position $\mathbf{x} \in \mathbb{R}^3$ and normal vector $\hat{\mathbf{n}} \in \mathbb{R}^3$ at a query point $\mathbf{q} \in \mathbb{R}^3$. Let $\mathbf{I} \in \mathbb{R}^{3\times3}$ be the identity matrix, $\mathbf{e}_i$ be its ith column, and $\mathbf{r} = \mathbf{x} - \mathbf{q}$. Then

$$\nabla G(\mathbf{q}, \mathbf{x}) = \frac{\mathbf{r}}{4\pi\|\mathbf{r}\|^3}, \tag{22}$$

$$\nabla^2 G(\mathbf{q}, \mathbf{x}) = \frac{\mathbf{I}}{4\pi\|\mathbf{r}\|^3} - \frac{3\mathbf{r}\otimes\mathbf{r}}{4\pi\|\mathbf{r}\|^5}, \tag{23}$$

$$\nabla^3 G(\mathbf{q}, \mathbf{x}) = -\frac{\sum_{i=1}^3 \mathbf{r}\otimes\mathbf{e}_i\otimes\mathbf{e}_i + \mathbf{e}_i\otimes\mathbf{r}\otimes\mathbf{e}_i + \mathbf{e}_i\otimes\mathbf{e}_i\otimes\mathbf{r}}{4\pi\|\mathbf{r}\|^5} + \frac{15\mathbf{r}\otimes\mathbf{r}\otimes\mathbf{r}}{4\pi\|\mathbf{r}\|^7}. \tag{24}$$

B TRIANGLE INTEGRALS

We also list the closed form expressions for the coefficients of our dipole integrated over a triangle t with corner positions $\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k$ and area a_t expanded about a cluster center $\tilde{\mathbf{p}}$. These functions are constant, linear and quadratic with respect to the position in the triangle. Consequently, we use single point quadrature for the first two integrals and edge-midpoint quadrature for that last. All are exact.

$$\int_t \hat{\mathbf{n}}_t \, dA = a_t \hat{\mathbf{n}}_t \tag{25}$$

$$\int_t (\mathbf{x} - \tilde{\mathbf{p}}) \otimes \hat{\mathbf{n}}_t \, dA = a_t \left(\frac{1}{3}(\mathbf{x}_i + \mathbf{x}_j + \mathbf{x}_k) - \tilde{\mathbf{p}} \right) \otimes \hat{\mathbf{n}}_t \tag{26}$$

$$\int_t (\mathbf{x} - \tilde{\mathbf{p}}) \otimes (\mathbf{x} - \tilde{\mathbf{p}}) \otimes \hat{\mathbf{n}}_t \, dA = a_t \mathbf{C}_t \otimes \hat{\mathbf{n}}_t, \tag{27}$$

(28)

where

$$\begin{aligned} \mathbf{C}_t = & \frac{1}{3} \left(\frac{1}{2}(\mathbf{x}_i + \mathbf{x}_j) - \tilde{\mathbf{p}} \right) \otimes \left(\frac{1}{2}(\mathbf{x}_i + \mathbf{x}_j) - \tilde{\mathbf{p}} \right) + \\ & \frac{1}{3} \left(\frac{1}{2}(\mathbf{x}_j + \mathbf{x}_k) - \tilde{\mathbf{p}} \right) \otimes \left(\frac{1}{2}(\mathbf{x}_j + \mathbf{x}_k) - \tilde{\mathbf{p}} \right) + \\ & \frac{1}{3} \left(\frac{1}{2}(\mathbf{x}_k + \mathbf{x}_i) - \tilde{\mathbf{p}} \right) \otimes \left(\frac{1}{2}(\mathbf{x}_k + \mathbf{x}_i) - \tilde{\mathbf{p}} \right). \end{aligned} \tag{29}$$