

Robust Inside-Outside Segmentation using Generalized Winding Numbers

Alec Jacobson¹

Ladislav Kavan^{2,1}

Olga Sorkine-Hornung¹

¹ETH Zurich ²University of Pennsylvania

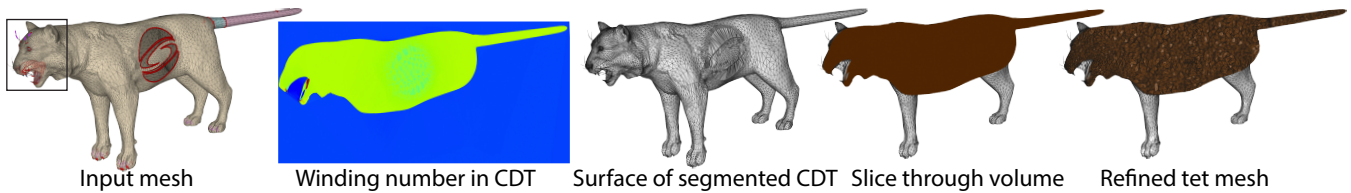


Figure 1: The Big SigCat input mesh has 3442 pairs of intersecting triangles (bright red), 1020 edges on open boundaries (dark red), 344 non-manifold edges (purple) and 67 connected components (randomly colored). On top of those problems, a SIGGRAPH logo shaped hole is carved from her side.

Abstract

Solid shapes in computer graphics are often represented with boundary descriptions, e.g. triangle meshes, but animation, physically-based simulation, and geometry processing are more realistic and accurate when explicit volume representations are available. Tetrahedral meshes which exactly contain (interpolate) the input boundary description are desirable but difficult to construct for a large class of input meshes. Character meshes and CAD models are often composed of many connected components with numerous self-intersections, non-manifold pieces, and open boundaries, precluding existing meshing algorithms. We propose an automatic algorithm handling all of these issues, resulting in a compact discretization of the input’s inner volume. We only require reasonably consistent orientation of the input triangle mesh. By generalizing the winding number for arbitrary triangle meshes, we define a function that is a perfect segmentation for watertight input and is well-behaved otherwise. This function guides a graphcut segmentation of a constrained Delaunay tessellation (CDT), providing a minimal description that meets the boundary exactly and may be fed as input to existing tools to achieve element quality. We highlight our robustness on a number of examples and show applications of solving PDEs, volumetric texturing and elastic simulation.

Keywords: winding number, tetrahedral meshing, inside-outside segmentation

Links: [DL](#) [PDF](#) [WEB](#) [VIDEO](#) [DATA](#)

1 Introduction

A large class of surface meshes used in computer graphics represent solid 3D objects. Accordingly, many applications need to treat such models as volumetric objects: for example, the animation or

physically-based simulation of a hippopotamus would look quite different (and unrealistic) if handled as a thin shell, rather than a solid. Since many operations in animation, simulation and geometry processing require an explicit representation of an object’s volume, for example for finite element analysis and solving PDEs, a conforming¹ tetrahedral meshing of the surface is highly desired, as it enables volumetric computation with direct access to and assignment of boundary surface values. However, a wide range of “real-life” models, although they *appear* to describe the boundary of a solid object, are in fact *unmeshable* with current tools, due to the presence of geometric and topological artifacts such as self-intersections, open boundaries and non-manifold edges. As a consequence, processing is often limited to the surface, bounding volumetric grids [McAdams et al. 2011] or approximations with volume-like scaffolding [Zhou et al. 2005; Baran and Popović 2007; Zhang et al. 2010].

The aforementioned artifacts are common in man-made meshes, as these are the direct output of human creativity expressed through modeling tools, which very easily allow such artifacts to appear. Sometimes they are even purposefully introduced by the designer: for example, character meshes will typically contain many overlapping components representing clothing, accessories or small features, many of which have open boundaries (see Figure 2). Modelers

¹Contrary to some authors’ use of “conforming” to mean that every mesh edge is locally Delaunay, we use it simply to mean that the volume mesh interpolates to the boundary description.

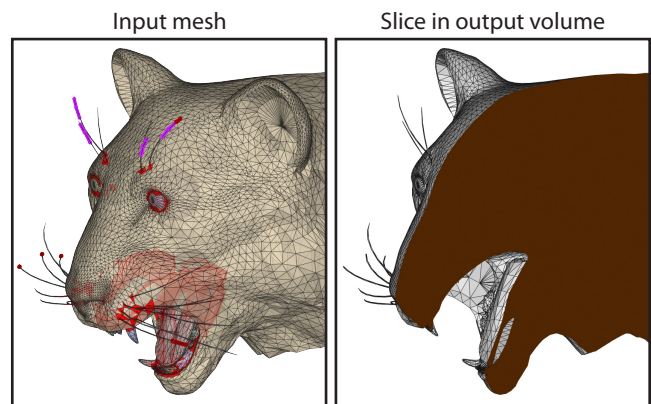


Figure 2: Each whisker, tooth and eye of the Big SigCat is a separate component that self-intersects the body.

choose very specific boundary mesh layout and vertex density, necessary for articulation or faithful representation of important features while staying on a tight vertex budget. It is therefore highly desirable to avoid remeshing and subsequent interpolation, and at the same time obtain a valid and precise representation of the object’s inner volume.

In this paper, we propose an automatic algorithm for producing volumetric meshes that fully contain the geometry of the input surface model. Our method robustly handles artifacts common in man-made meshes while still supporting the full set of quality assurances, as do existing conforming meshing tools.

Past years have seen many advances in algorithms for generating high quality simplicial (triangles in \mathbb{R}^2 , tetrahedra in \mathbb{R}^3) volume meshes. The popular tools TRIANGLE [Shewchuk 1996] and TET-GEN [Si 2003] are examples of methods that exactly conform to a given piecewise-linear boundary description. Such tools support a wide range of features, in particular concerning element quality, but they fail when the input boundary descriptions contain geometric ambiguities or flaws which make the inner volume even remotely ambiguous. The number of these issues in a common man-made model may range in the hundreds or thousands (see Figure 1), so manual clean up is time consuming and deadeningly tedious. An alternative could be to treat this as a surface repair problem, but this precludes exactly maintaining the original boundary vertices and facets during local fixup operations [Attene 2010]. Surface reconstruction techniques are not quite suitable in our setting either, because they focus on recovering surfaces from scans of real solids, where the artifacts should only arise from scanning errors, and hence partial or complete remeshing and loss of input features may occur.

Our method generally follows the steps of reconstruction based on constrained Delaunay tessellation (CDT): we compute the (tetrahedral) CDT of the convex hull of the input vertices and facets. We rely on state-of-the-art CDT tools, which currently require certain pre-processing of the input, such as subdivision of self-intersecting facets and discarding degeneracies. The goal is then to segment the CDT volume into “inside” and “outside” elements, such that the set of inside elements comprises a valid conforming tet mesh. Our main contribution is the introduction of a new inside-outside confidence function by generalizing the *winding number*. Though similar at a high level, signed distance functions do not encode segmentation confidence. They smoothly pass through zero at the surface, whereas our function has a sharp jump there. Away from the surface our function is smooth (in fact, harmonic!). It defines a perfect, piecewise-constant segmentation of space when the input is perfect (i.e. a watertight surface). When the input contains ambiguities and missing information, the well-behaved nature of our function makes it suitable for guiding an energy-minimizing segmentation of the CDT, which can be efficiently computed using graphcut. We can constrain the segmentation to exactly contain all input vertices and facets, as well as ensure surface manifoldness. The final output is a minimal tetrahedral mesh carved from the CDT which may be post-processed using existing tools to achieve high-quality elements or heterogeneous sizing.

We evaluate our algorithm on a wide range of inputs, which are otherwise unmeshable with existing tools. We demonstrate the usefulness of the method via applications such as physically-based elasticity simulation, skinning weight computation for real-time animation, geometric modeling and volumetric texturing. Our algorithm offers a step towards a new level of robustness of unstructured volumetric meshing, which will potentially have a large impact on the standard computer graphics pipeline, especially as geometry processing turns toward treating solids as solids rather than operating (often just out of convenience or obligation) on merely the surface.

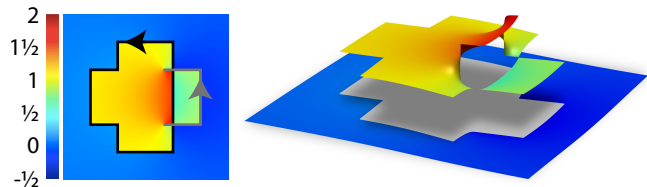


Figure 3: The winding number intuitively captures self-intersections, maintaining boundary exactly (cf. Figure (5) in [Shen et al. 2004]).

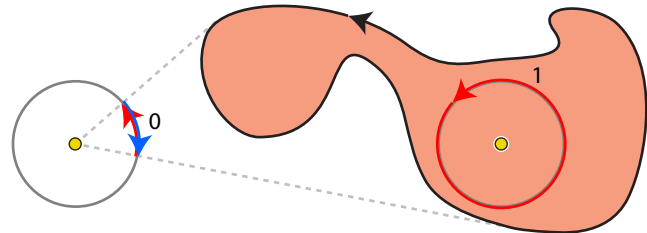


Figure 4: Winding number is the signed length of the projection of a curve onto a circle at a given a point divided by 2π . Outside the curve, the projection cancels itself out. Inside, it measures one.

2 Related work

Surface repair. Artifacts of surface meshes, such as violations of the connected 2-manifoldness, consistent orientation or watertightness properties, not only disturb conforming volumetric meshing but also surface-based processing, because the majority of geometric algorithms assume clean input. Although the problem of mesh repair has been extensively studied, it remains elusive in practice [Ju 2009; Campen et al. 2012]. Most methods for meshing of polygon soups into surfaces do not robustly deal with self-intersecting input facets [Hoppe et al. 1993; Kraevoy et al. 2003; Guéziec et al. 2001; Podolak and Rusinkiewicz 2005] insofar as promising a volume-meshable, *watertight* surface. Some methods do offer guarantees; they work by globally remeshing the output [Ju 2004; Bischoff et al. 2005] or by making local modifications at the cost of not maintaining the original mesh (geometry and/or connectivity) in troublesome areas [Yamakawa and Shimada 2009; Attene 2010]. Bischoff and Kobbelt [2005] repair CAD models while trying to preserve the original meshing, but they assume the input is divided into manifold patches that do not self-intersect, and their method requires a spatially-varying threshold for gap filling. [Attene 2010] provide volume meshing as an application of their watertight output. However, because their algorithm iterates between removing troublesome patches and hole filling, large portions of the original mesh may be deleted (see Figure 18). Holes are filled by locally modifying the mesh and become hard boundary constraints for volume meshing. Conversely, our winding number function incorporates global information to intelligently resolve missing information ambiguities. A volumetric tool for general surface repair exists [Nooruddin and Turk 2003], but its voxel-based nature does not scale well for large, detailed models and complicates interpolation of the input mesh. Unlike our method, the work of [Murali and Funkhouser 1997] is not restricted to consistently oriented input. However, their voting-based approach is prone to mis-assignment in overlap regions and loss of small details [Attene 2010].

Surface reconstruction can be seen as an alternative way to obtain a clean, watertight surface mesh. However, most reconstruction algorithms are tuned to noisy point cloud inputs and hence do not strive to preserve the input mesh structure. Algorithms like the Zipper of [Turk and Levoy 1994] stitch range images by generally

only modifying the mesh along the overlap, but this approach is only suitable for well-aligned range images. A host of reconstruction methods, starting with [Hoppe et al. 1992], fit an implicit function to the input surface geometry and extract a level set, which is guaranteed to be watertight for well-behaved functions; recent methods are quite robust to noisy data [Kazhdan et al. 2006; Mullen et al. 2010] and even unoriented data [Alliez et al. 2007]. However, the original input mesh is generally lost during contouring. Shen et al. [2004] design level-sets using moving least squares to perfectly interpolate input facets, but contouring loses any premeditated discretization distribution. Due to the oscillatory nature of their function, the exact interpolation constraint may need to be relaxed when components overlap (see our Figure 3 and their Figure 5).

Surface reconstruction of point clouds has been achieved with graphcut segmentation on voxel-grids [Hornung and Kobbelt 2006] and on Delaunay meshes [Wan et al. 2011]. Wan et al. [2012] tackled open surfaces via graphcut on a level-set of an intersection of approximating “crusts”. Our method, as many previous ones, segments a volume from a constrained Delaunay tessellation of the input convex hull. The peeling procedure of [Dey and Goswami 2003] fills surface holes, ensuring a watertight result, though possibly non-manifold. It requires a fine enough initial discretization to prevent a degenerate solution. The spectral method of [Kolluri et al. 2004] improves upon this. They provide similar post-processing heuristics to ours for ensuring manifoldness. However, extending their spectral analysis to interpolate input facets is not obvious.

Unstructured tetrahedral mesh generation. Efficient creation of Delaunay tessellations is well studied; [Shewchuk 2012] is an excellent survey. Methods can be subdivided into those that exactly conform to input vertices and faces [George et al. 1990; Shewchuk 1996; Joshi and Ourselin 2003; Si 2003; Geuzaine and Remacle 2009] and those that approximate watertight input surfaces [Shimada and Gossard 1995; Alliez et al. 2005; Bridson et al. 2005; Labelle and Shewchuk 2007]. We heavily rely on the former to mesh the convex hull of our input. Additionally, as our method outputs a minimal tetrahedral mesh, we may post-process with mesh refinement tools [Schöberl 1997; Si 2003; Klingner and Shewchuk 2007; Geuzaine and Remacle 2009] to achieve element quality.

Winding number, inside-outside tests. The winding number of closed curves is an old concept [Meister 1769/70]. To the best of our knowledge, no previous work has generalized “winding numbers” computed as integrals on open curves or surfaces, but many related functions exist. Mean value coordinates use a similar projection integral [Floater 2003; Ju et al. 2005], but lack the jump discontinuity across the boundary that gives the winding number its unique segmentation property. They are also notably not harmonic, and may oscillate and not satisfy the maximum principle. In the terminology presented by [Zhou et al. 2008], our winding number adheres to an “object-based” definition of inside-outside. Thus we are a complement to their “view-based” definition. Their method uses ray-shooting combined with graphcut to achieve a different set of applications, more suitable to computer vision.

3 Method

Our goal is a tet mesh conforming to an input shape. We achieve this by computing a constrained Delaunay tessellation (CDT) containing the input vertices and facets; by evaluating a generalization of the winding number for each element, we segment inside and outside elements of the CDT, resulting in the final tet mesh.

Let the input shape in \mathbb{R}^d be described by a list of n vertices $\mathcal{V} = \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n\}$, $\mathbf{v}_i \in \mathbb{R}^d$ and a list of m simplicial facets $\mathcal{F} = \{f_1, \dots, f_m\}$ where $f_i \in \{1, 2, \dots, n\}^d$ (we only consider $d =$

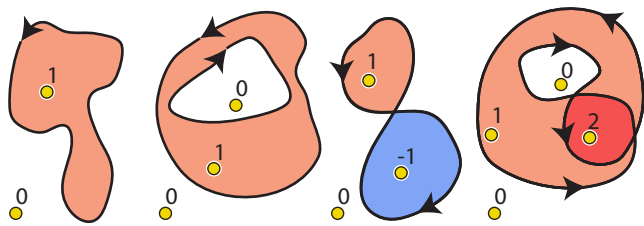


Figure 5: Winding number exactly segments inside and outside for concave, high-genus, inverted and overlapping curves. Multiple components are also naturally handled: consider this entire figure.

2 and $d = 3$). The goal is then to find a set of elements $\mathcal{E} \subset \{1, \dots, k\}^{d+1}$ defined over a set of vertices $\mathcal{V}_{\mathcal{E}}$ which represent the area (if $d = 2$) or volume (if $d = 3$) of $(\mathcal{V}, \mathcal{F})$. In the ideal case, we achieve *exact interpolation*: $\mathcal{V}_{\mathcal{E}} = \mathcal{V}$ and all facets in \mathcal{F} appear as subfacets of elements in \mathcal{E} . Note, facets and elements correspond to triangles and tetrahedra in \mathbb{R}^3 and edges and triangles in \mathbb{R}^2 .

Although \mathcal{F} forms a graph or *mesh* over \mathcal{V} , the input is not assumed to be $(d - 1)$ -manifold, orientable or closed. We do assume the mesh *intuitively represents* or loosely approximates the surface of some solid and has reasonably consistent orientation. This is motivated by the observation that most practical input meshes were created in such a way that they *appear* to be the surface of some solid when rendered with single-sided lighting.

We first construct an inside-outside confidence function which generalizes the winding number. We then evaluate the integral average of this function at each element in a CDT containing $(\mathcal{V}, \mathcal{F})$. Finally, we select a subset \mathcal{E} of the CDT elements via graphcut energy optimization with optional constraints to enforce strict facet interpolation and manifoldness.

4 Winding number

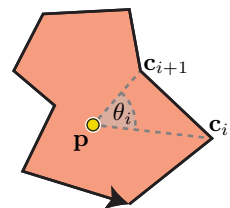
The traditional *winding number* $w(\mathbf{p})$ is a signed, integer-valued property of a point \mathbf{p} with respect to a closed Lipschitz curve \mathcal{C} in \mathbb{R}^2 . Intuitively, if we imagine there is an observer located at \mathbf{p} tracking a moving point along \mathcal{C} , the winding number tells us the number of full revolutions the observer took. Full counter-clockwise revolutions increase the count by one, while clockwise turns subtract one. In other words, $w(\mathbf{p})$ is the number of times \mathcal{C} wraps around \mathbf{p} in the counter-clockwise direction. Without loss of generality let $\mathbf{p} = \mathbf{0}$, parameterize \mathcal{C} using polar coordinates and define

$$w(\mathbf{p}) = \frac{1}{2\pi} \oint_{\mathcal{C}} d\theta. \quad (1)$$

It is the signed length of the projection of \mathcal{C} onto the unit circle around \mathbf{p} divided by 2π (see Figure 4). A value of 0 or 1 means \mathbf{p} lies outside or inside \mathcal{C} , respectively. The winding number distinguishes outside and inside for curves enclosing regions of arbitrary genus, and also identifies regions of overlap (see Figure 5).

The integral in Equation (1) provides an immediate and exact discretization if \mathcal{C} is piecewise linear:

$$w(\mathbf{p}) = \frac{1}{2\pi} \sum_{i=1}^n \theta_i, \quad (2)$$



where θ_i is the **signed** angle between vectors from two consecutive vertices \mathbf{c}_i and \mathbf{c}_{i+1} on \mathcal{C} to \mathbf{p} .

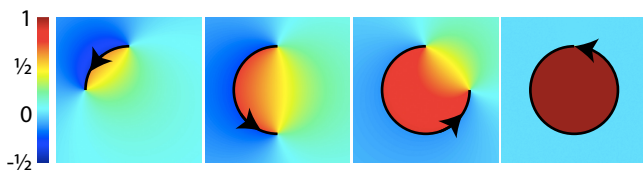


Figure 6: Left to right: winding number field with respect to an open, partial circle converging to a closed circle. Note the ± 1 jump discontinuity across the curve. Otherwise the function is harmonic: smooth with minimal oscillation.

Let $\mathbf{a} = \mathbf{c}_i - \mathbf{p}$ and $\mathbf{b} = \mathbf{c}_{i+1} - \mathbf{p}$, then:

$$\tan(\theta_i(\mathbf{p})) = \frac{\det([\mathbf{a} \ \mathbf{b}])}{\mathbf{a} \cdot \mathbf{b}} = \frac{a_x b_y - a_y b_x}{a_x b_x + a_y b_y} \quad (3)$$

4.1 Generalization to \mathbb{R}^3

The winding number immediately generalizes to \mathbb{R}^3 by replacing angle with *solid angle*. The solid angle Ω of a Lipschitz surface \mathcal{S} with respect to a point $\mathbf{p} \in \mathbb{R}^3$ (w.l.o.g. let $\mathbf{p} = \mathbf{0}$) is defined using spherical coordinates to be:

$$\Omega(\mathbf{p}) = \iint_{\mathcal{S}} \sin(\phi) \, d\theta d\phi. \quad (4)$$

It is the signed surface area of the projection of \mathcal{S} onto the unit sphere centered at \mathbf{p} .

Let the winding number of a closed surface \mathcal{S} at point \mathbf{p} be defined as $w(\mathbf{p}) := \Omega(\mathbf{p})/4\pi$. The same classification properties apply as in \mathbb{R}^2 . The notion of “winding”, now counts the (signed) total number of times the surface wraps around a point.

And again, if we have a triangulated, piecewise-linear surface, there is an immediate and exact discretization of Equation (4):

$$w(\mathbf{p}) = \sum_{f=1}^m \frac{1}{4\pi} \Omega_f(\mathbf{p}), \quad (5)$$

where Ω_f is the solid angle of the **oriented** triangle $\{\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_k\}$ with respect to \mathbf{p} . Let $\mathbf{a} = \mathbf{v}_i - \mathbf{p}$, $\mathbf{b} = \mathbf{v}_j - \mathbf{p}$, $\mathbf{c} = \mathbf{v}_k - \mathbf{p}$ and $a = \|\mathbf{a}\|$, $b = \|\mathbf{b}\|$, $c = \|\mathbf{c}\|$; then following [van Oosterom and Strackee 1983]:

$$\tan\left(\frac{\Omega(\mathbf{p})}{2}\right) = \frac{\det([\mathbf{a} \ \mathbf{b} \ \mathbf{c}])}{abc + (\mathbf{a} \cdot \mathbf{b})c + (\mathbf{b} \cdot \mathbf{c})a + (\mathbf{c} \cdot \mathbf{a})b} \quad (6)$$

4.2 Open, non-manifold and beyond

The simplicity of the discrete formulae in Equations (2) and (5) begs the question, what will happen if the input is open? Or non-manifold? Or otherwise ambiguous?

We first consider open curves in \mathbb{R}^2 . Instead of an indicator, step function, Equation (2) is now an otherwise smooth function that jumps by ± 1 across the curve (see Figure 6). In fact, the smoothness and fairness of this *generalized winding number* may be well

```
# define Laplacian operator in 2d
Laplacian2 := (f,x,y) -> diff(f,x,x) + diff(f,y,y);
# arbitrary position for vi, a := vi - p
a_x := v1_x-px; a_y := v1_y-py;
# arbitrary position for vj, b := vj - p
b_x := vj_x-px; b_y := vj_y-py;
# determinant of (a,b)
detab := a_x*b_y - b_x*a_y;
# a dot b
adotb := a_x*b_x + a_y*b_y;
quotient := detab / adotb;
sab := 2*arctan(simplify(quotient));
simplify(Laplacian2(sab,px,py),symbolic);
# result is 0
```

```
# define Laplacian operator in 3d
Laplacian3 := (f,x,y,z) -> diff(f,x,x) + diff(f,y,y) + diff(f,z,z);
# vi := (0,0,0), a := vi - p
a_x := 0-px; a_y := 0-py; a_z := 0-pz;
# arbitrary position for vj, b := vj - p
b_x := vj_x-px; b_y := vj_y-py; b_z := vj_z-pz;
# arbitrary position for vk, c := vk - p
c_x := vk_x-px; c_y := vk_y-py; c_z := vk_z-pz;
# determinant of (a,b,c)
detabc := a_x*b_y*c_z + b_x*c_y*a_z + c_x*a_y*b_z -
a_x*c_y*b_z - b_x*a_y*c_z - c_x*b_y*a_z;
a := sqrt(a_x*a_x+a_y*a_y+a_z*a_z);
b := sqrt(b_x*b_x+b_y*b_y+b_z*b_z);
c := sqrt(c_x*c_x+c_y*c_y+c_z*c_z);
# divisor in atan
divisor := a*b*c + (a_x*b_x+a_y*b_y+a_z*b_z)*c +
(b_x*c_x+b_y*c_y+b_z*c_z)*a + (c_x*a_x+c_y*a_y+c_z*a_z)*b;
sabc := 2*arctan(detabc / divisor);
simplify(Laplacian3(sabc,px,py,pz),symbolic);
# result is 0
```

Figure 7: MAPLE code proving that signed angle in \mathbb{R}^2 , solid angle \mathbb{R}^3 , and, by extension, the winding number are harmonic.

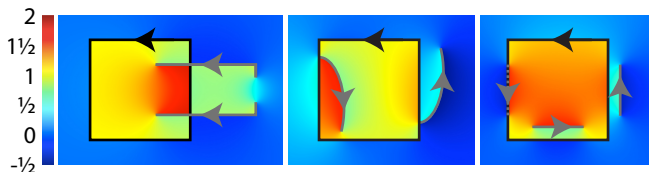


Figure 8: Winding number gracefully handles holes (in grey curve, left), non-manifold attachments (middle), and exactly or nearly duplicate facets (right).

understood. Except on the curve, it is harmonic! This implies C^∞ smoothness and minimal oscillations – highly desirable properties.

The sum of harmonic functions is harmonic, so it suffices to show that all θ_i and Ω_i are harmonic. This is easy to do using symbolic differentiation and simplification using Maple [Char et al. 1983] (see Figure 7). In \mathbb{R}^3 treating all triangle vertices $\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_k$ as symbolic variables makes Maple run out of memory, therefore we take advantage of invariance to translation and fix $\mathbf{v}_i = (0, 0, 0)$.

The winding number is not *simply* the unique harmonic function determined by setting one side of the boundary to 0 and the other to 1, as if by a diffusion curve of [Orzan et al. 2008] (also cf. [Davis et al. 2002]). This is true if and only if the input is watertight. Rather, the winding number is the sum of harmonic functions corresponding to each input facet, setting one side to $-1/2$ and the other to $1/2$ (see Figure 9). We do not explicitly control the boundary conditions — they are implicitly defined by the boundary winding number itself. This allows graceful shift from a perfect segmentation function to a smooth confidence measure as artifacts appear in the boundary. Unlike [Orzan et al. 2008] who solve a variational problem, we have a closed-form expression to evaluate the winding number.

Equation (5) may be interpreted as an instance of the boundary element method (BEM) for evaluating the solution to the Laplace equation. If we define Dirichlet boundary conditions on each side

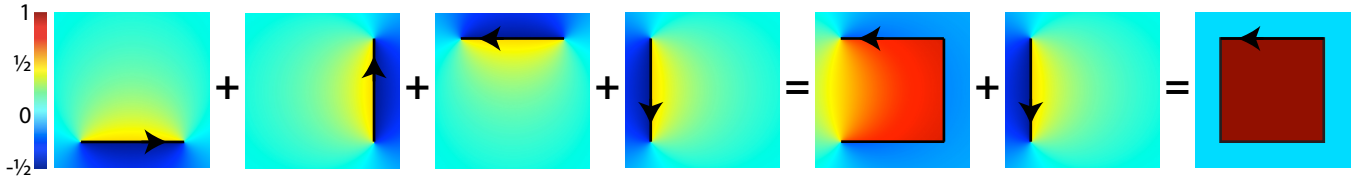


Figure 9: The winding number is the sum of harmonic functions defined for each facet.

of our facets using the winding number, the solution of the Laplace equation on the entire space is equivalent to $w(\mathbf{p})$ for $\mathbf{p} \in \mathbb{R}^d$. This follows from the uniqueness property of harmonic functions.

An alternative understanding of the winding number is to shoot rays in every direction from \mathbf{p} . For each ray sum ± 1 for each signed intersection with the input. The traditional *and our generalized* winding number is the average of these values. This understanding is useful conceptually, but difficult to realize as an algorithm. While casting a few rays is possible [Nooruddin and Turk 2003; Houston et al. 2003], this approximation will be noisy in the presence of open boundaries and non-manifold edges. By considering the input’s projection on the unit ball around \mathbf{p} instead, our algorithm is tantamount to shooting all possible rays.

The jump discontinuity across the input facets provides the winding number a unique advantage as a confidence measure in contrast to other methods (e.g. signed distance fields). Such measures continuously approach a zero level-set, where the difference between the measure at a clearly inside point (just to the inside of a facet) and a clearly outside point (just to the outside) diminishes. In contrast, the winding number instead becomes ever more confident and the measure approaches the discontinuous boundary conditions at that facet, regardless of whether the facet is part of a watertight component (see Figure 3).

Non-manifold edges appear often in 3D character meshes to describe thin clothing or accessories. It is convenient to conceptually treat each manifold patch of $(\mathcal{V}, \mathcal{F})$ as an appropriately open or closed surface. Each patch then contributes independently to the total winding number. Thus non-manifold edges affect the winding number in a similarly predictable manner to open boundaries (see Figure 8 middle).

In character meshes and CAD models, there may be entirely duplicated or nearly duplicated patches of the input mesh. These shift the winding number range locally (see Figure 8 right). This disqualifies simply thresholding the winding number for final segmentation, hence our use of a carefully crafted graphcut energy.

4.3 Hierarchical evaluation

The discrete formulae in Equations (2) and (5) give a direct route to a naive implementation to compute $w(\mathbf{p})$: simply sum the contribution of θ_i or Ω_i for each input facet. This is embarrassingly parallel and the geometric definition invites the possibility of a shader-style parallel implementation. However, the asymptotic runtime would still grow linearly with the number of input facets. A facet’s effect on $w(\mathbf{p})$ diminishes with respect to its distance to \mathbf{p} . We could asymptotically speed up our evaluation with an adaptation of the Fast Multipole Method, however this would only be an approximation. Instead, we achieve exact evaluation and asymptotic performance gains by noticing that the winding number obeys the following simple property.

Consider a possibly open surface \mathcal{S} and an arbitrary *closing surface* $\bar{\mathcal{S}}$ such that $\partial\bar{\mathcal{S}} = \partial\mathcal{S}$ and $\bar{\mathcal{S}} \cup \mathcal{S}$ is some closed, oriented surface \mathcal{T} . Then if \mathbf{p} is outside the convex hull of \mathcal{T} , we know

Algorithm 1: `construct_hierarchy($T, \mathcal{V}, \mathcal{F}$)`

Inputs:

T root of subtree in hierarchy
 \mathcal{V} mesh vertex positions
 \mathcal{F} list of facets in `bbox(T)`

begin

```

 $E \leftarrow \text{exterior\_edges}(\mathcal{F})$  // Compute list of exterior edges of  $\mathcal{F}$ 
 $T.\bar{\mathcal{S}} \leftarrow \text{closure}(E)$  // Compute closure of  $\mathcal{F}$ 
if  $|\mathcal{F}| < 100$  or  $|T.\bar{\mathcal{S}}| \geq |\mathcal{F}|$  then
   $T.\mathcal{F} \leftarrow \mathcal{F}$  // mark as leaf and save  $\mathcal{F}$ 
  return
end
 $\mathcal{F}_{\text{left}} \leftarrow \text{restrict}(\mathcal{V}, \mathcal{F}, \text{bbox}(T.\text{left}))$  // Restriction of  $\mathcal{F}$ , left
 $\mathcal{F}_{\text{right}} \leftarrow \text{restrict}(\mathcal{V}, \mathcal{F}, \text{bbox}(T.\text{right}))$  // Restriction of  $\mathcal{F}$ , right
construct_hierarchy( $T.\text{left}, \mathcal{V}, \mathcal{F}_{\text{left}}$ ) // recurse
construct_hierarchy( $T.\text{right}, \mathcal{V}, \mathcal{F}_{\text{right}}$ ) // recurse

```

end

Algorithm 2: `hier_winding(p, T, \mathcal{V}) $\rightarrow w$`

Inputs:

\mathbf{p} evaluation point
 T root of subtree in hierarchy
 \mathcal{V} mesh vertex positions

Outputs:

w exact generalized winding number at p

begin

```

if  $T$  is a leaf then
   $w \leftarrow \text{naive\_winding}(\mathbf{p}, T.\mathcal{F}, \mathcal{V})$  // use all faces  $T.\mathcal{F}$ 
else if  $\mathbf{p}$  is outside bbox( $T$ ) then
   $w \leftarrow -\text{naive\_winding}(\mathbf{p}, T.\bar{\mathcal{S}}, \mathcal{V})$  // use closure  $T.\bar{\mathcal{S}}$ 
else
   $w_{\text{left}} \leftarrow \text{hier\_winding}(\mathbf{p}, T.\text{left}, \mathcal{V})$  // recurse left
   $w_{\text{right}} \leftarrow \text{hier\_winding}(\mathbf{p}, T.\text{right}, \mathcal{V})$  // recurse right
   $w \leftarrow w_{\text{left}} + w_{\text{right}}$  // sum
end
return  $w$ 

```

end

that $w_{\mathcal{S}}(\mathbf{p}) + w_{\bar{\mathcal{S}}}(\mathbf{p}) = w_{\mathcal{T}}(\mathbf{p}) = 0$. Interestingly this means $w_{\mathcal{S}}(\mathbf{p}) = -w_{\bar{\mathcal{S}}}(\mathbf{p})$, regardless of how $\bar{\mathcal{S}}$ is constructed. Notice this result is trivial if \mathcal{S} is closed, as $w_{\mathcal{S}}(\mathbf{p}) = 0$.

We can conceptually express our mesh as a union of manifold patches. We define *exterior edges* as boundary edges of such a segmentation. In \mathbb{R}^3 , if \mathbf{p} lies outside the convex hull of $(\mathcal{V}, \mathcal{F})$, then we collect all exterior edges and trivially triangulate each with an arbitrary vertex. Though ugly from a surface repair point of view, these triangles indeed represent a valid closing of $(\mathcal{V}, \mathcal{F})$ and will only be used for winding number evaluation. Note that the segmentation into manifold patches is never explicitly computed. Rather we traverse around each facet in order, and for each directed edge i, j we increment `count(i, j)` if $i < j$ and decrement `count(j, i)` if $j < i$. In this way we keep track of how many *extra* times each edge

is seen in the forward or backward direction. Finally all edges with $\text{count}(i, j) \neq 0$ are declared exterior and triangulated with some arbitrary vertex k with orientation $\{i, j, k\}$ if $\text{count}(i, j) = c > 0$ and $\{j, i, k\}$ if $\text{count}(i, j) = -c < 0$. These triangles are repeated $|c|$ times to account for possible multiple coverage of the same exterior edge. In \mathbb{R}^2 , we analogously find *exterior vertices* and connect them to an arbitrary vertex using appropriately oriented line segments.

For reasonably tessellated meshes, the number of exterior edges and thus the number of closing triangles will be $\mathcal{O}(\sqrt{m})$. We exploit this by evaluating the winding number using a bounding volume hierarchy partitioning \mathcal{F} . Though there is an art to optimizing bounding volume hierarchies, we opt for a simple axis-aligned-bounding-box hierarchy. We initialize the root with the bounding box of \mathcal{V} . We precompute the exterior edges and closure of \mathcal{F} , then we simply bisect the box, splitting its longest side. Each facet of \mathcal{F} is distributed to the child whose box contains the facet’s barycenter. We recurse on each child. Splitting stops when the number of a box’s exterior edges approximately equals the number of its facets or when the number of its facets slips below a threshold (≈ 100). This stopping criterion ensures that worst case performance stays the same. See Algorithm 1. To evaluate the winding number, we traverse this hierarchy recursively. When we reach a box of which the evaluation point is outside, we evaluate using the closure. See Algorithm 2. In general we see large speed ups (see Figures 10 & 11).

5 Segmentation

We segment according to the winding number by selecting a subset of the elements in a constrained Delaunay tessellation of the convex hull of $(\mathcal{V}, \mathcal{F})$. We may then refine this mesh to meet quality criterion using [Si 2003] or [Klingner and Shewchuk 2007].

Theoretically the only problems when computing a CDT on our input mesh $(\mathcal{V}, \mathcal{F})$ are self-intersections. In \mathbb{R}^2 , the TRIANGLE program [Shewchuk 1996] automatically adds Steiner points at line segment intersections. To our knowledge there is no equivalent in \mathbb{R}^3 . So, we first remove any duplicate or degenerate facets. Then we compute all triangle-triangle intersections using the exact construction kernel in [CGAL]. This kernel is exact even for difficult cases like coplanar, overlapping triangles. It specifies the locations for Steiner points and constraint segments on each offending triangle. We solve a separate 2D CDT problem to meet each set of constraints. Alternatively, employing [Campen and Kobbelt 2010] promises performance gains.

Unfortunately, efficient CDT algorithms are prone to numerical issues and fail when input constraints are too close together. Thus additional clean up is occasionally required. Rather than remesh the entire input, we notice that in practice a CDT is possible when no facets are constrained. Thus we enforce as many facets as permitted by our choice of CDT meshing software [Si 2003]. Troublesome facets are removed or subdivided according to a small area and small angle threshold. Subdivision helps ensure minimal disturbance of the facet interpolation.

By using an imperfect CDT, we are relaxing our strict interpolation constraint. However, surface repair methods like [Attene et al. 2007] are much more aggressive (see Figure 18). Further, our pre-processing is solely to facilitate construction of the CDT, which is orthogonal to our volume segmentation problem. All original facets are still used to compute the winding number. When improved CDT methods appear, our method will immediately see benefits.

Winding number computation time (subdivided Dino)

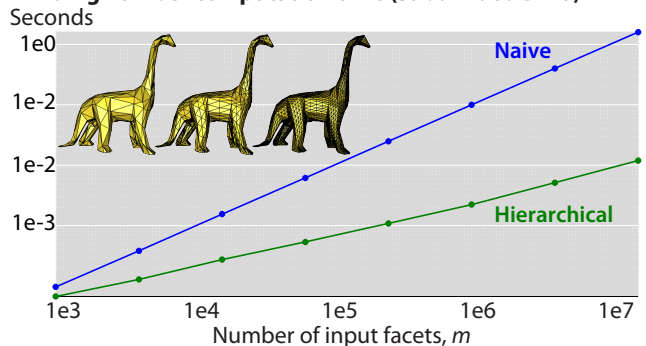


Figure 10: Hierarchical evaluation performs asymptotically better than the naive implementation on the subdivided Dino. Naive (blue) fits neatly to $m^{0.94}$, hierarchical (green) fits neatly to $m^{0.43}$.

Winding number computation time (SHREC Dataset)

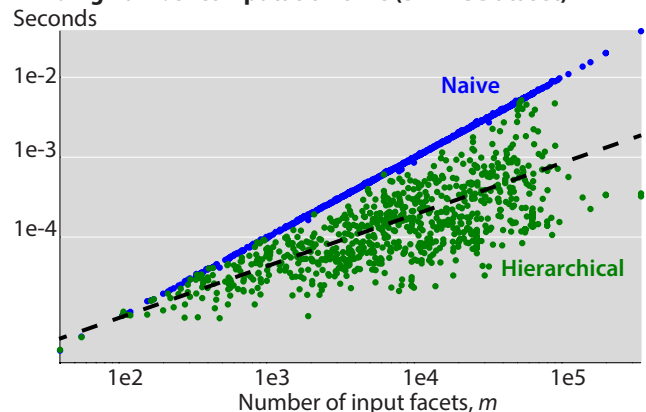


Figure 11: Hierarchical evaluation performs asymptotically better than the naive implementation on a large set of different meshes. Naive (blue) fits neatly to $m^{1.00}$, hierarchical (green) fits in least squares sense to $m^{0.55}$ (black line).

5.1 Energy minimization with graphcut

We now have a standard segmentation problem. If the input is perfectly free of ambiguities then the winding number already acts as an exact segmentation. If the input is not perfectly clean then we need a more sophisticated segmentation. An obvious first approach is to apply a simple threshold:

$$\text{is_outside}(e_i) = \begin{cases} \text{true} & \text{if } w(e_i) < 0.5 \\ \text{false} & \text{otherwise} \end{cases}, \quad (7)$$

where by abuse of notation, let $w(e_i) = \frac{1}{V} \int_{e_i} w(\mathbf{p}) dV$ be the integral average of w in element e_i . However, this does not incorporate coherency between neighboring elements (see Figure 12).

Instead we propose an energy functional, consisting of a data term and smoothness term, whose minimum respects the winding number, but behaves better due to enforced smoothness. The energy is written:

$$E = \sum_{i=1}^m \left[u(x_i) + \gamma \frac{1}{2} \sum_{j \in N(i)} v(x_i, x_j) \right] \quad (8)$$

where x_i is the unknown binary segmentation function at element e_i , $N(i)$ is the set of elements sharing a facet with e_i and γ is a

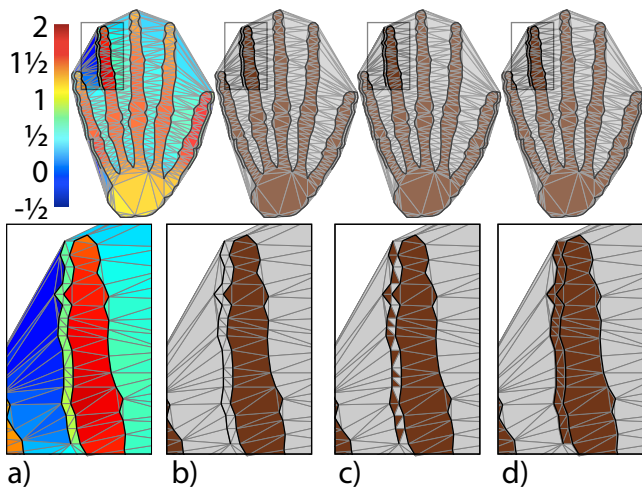


Figure 12: The winding number inside a hand with thin accessories (a). Without constraints the accessories may be lost (b). They are recovered by adding the incident element with highest winding number (c). Local improvement of the graphcut energy encourages smoothness (d).

parameter balancing the data and smoothness terms. We define the data term as:

$$u(x_i) = \begin{cases} \max(w(e_i) - 0, 0) & \text{if } x_i = \text{outside} \\ \max(1 - w(e_i), 0) & \text{otherwise} \end{cases} \quad (9)$$

These terms will become edge weights in a graphcut optimization and thus must be non-negative. If $\gamma = 0$ then the optimal solution coincides with constant thresholding [Chen et al. 2011].

We use an exponential function to achieve a discontinuity-aware smoothness term [Boykov and Funka-Lea 2006]:

$$v(x_i, x_j) = \begin{cases} 0 & \text{if } x_i = x_j \\ \frac{a_{ij} \exp(-|w(e_i) - w(e_j)|^2)}{2\sigma^2} & \text{otherwise} \end{cases} \quad (10)$$

where a_{ij} is the length/area (for $d = 2/3$) of the facet shared between e_i and e_j , and σ is a “noise”-estimation parameter.

A graph with appropriate edge-weights is constructed according to [Kolmogorov and Zabini 2004], and the optimal segmentation is found by running a max-flow algorithm.

One last question remains: how to evaluate the integral average of the winding number per element? A simple solution is to evaluate w at the barycenter of each element. This works well for inputs without major issues and when the CDT contains reasonably well-shaped elements. For extremely difficult cases we can increase the accuracy of this integral by using more quadrature points. We use a simple symmetric scheme of [Zhang et al. 2009] and see diminishing returns on the number of points.

5.2 Optional hard constraints

Our generalized winding number combined with graphcut can be seen as an outlier detector if some of the input facets \mathcal{F} do not appear as subfacets of the segmented elements \mathcal{E} , as this only happens when the input is ambiguous (see Figure 13). Unfortunately, we cannot efficiently and *optimally* enforce facet interpolation as hard constraints. Enforcing these constraints as infinite penalty terms in Equation (8) results in a *nonregular* function in the parlance of

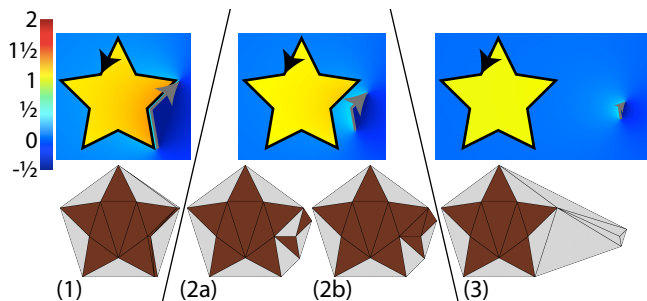


Figure 13: Thresholding winding number finds unambiguous attachments (1). Harder cases require facet constraints. Splinters (2a) are avoided by local improvement with a smoothness energy fixes this (2b). Finally, the winding number can detect outliers (3).

[Kolmogorov and Zabini 2004]. They prove that such functions, and thus our constraints, can not be optimized using graphcut.

For completeness we implement a simple heuristic approach to ensuring facet constraints are met automatically. We march over unsatisfied constraints and satisfy them by adding the incident element with largest winding number. After each update we greedily optimize the energy in Equation (8) by recursively testing whether to flip the assignment of elements neighboring any just-flipped elements. During improvement we do not allow flips that violate any already satisfied constraints. We converge to a local and feasible minimum with respect to the energy and the facet constraints.

We may similarly enforce a surface manifoldness constraint by marching over edges and vertices in the CDT. When a non-manifold issue is found we simply sort incident elements in descending order according to their winding number and flip them to the inside until local manifoldness is achieved. Again we greedily improve after each step to a local minimum. Note that [Attene et al. 2007] proposes a method for converting sets of tetrahedra (e.g. our output) into manifold volumetric meshes, and alternatively we could use it to post-process our output without manifoldness constraints.

6 Experiments and Results

We evaluated our algorithm on a large number of input shapes. Figure 21 and high resolution images in supplemental material show the input mesh, highlighting artifacts, a slice through the bounding box, showing the winding number computed for each element of the CDT, and our resulting tet mesh with cut-away slices. We show success on a variety of man-made meshes: CAD models (e.g. *Phone*, *Alien Space-object*) and character meshes (e.g. *Skeleton*, *SWAT Man*, *Ballet Woman*, *Crocodile*). Our input and output meshes are publicly available as supplemental material.

Meshes like the *Skeleton* contain many slightly overlapping connected components. These could be meshed independently and combined using boolean operations, but this complicates implementation and will not work for inputs like *SWAT Man*, whose overlapping components have open boundaries and non-manifold edges. For *SWAT Man*, we activate our optional constraints ensuring that all input facets are contained in the final tet mesh. This is necessary for such applications as physically-based simulation requiring safe contact detection.

The *Ant* has minimal triangulation for the thin legs and antennae, which our method preserves. This not only allows direct access to and assignment of boundary values, but enables efficient storage as the input mesh and our output tet mesh share the same vertex set.

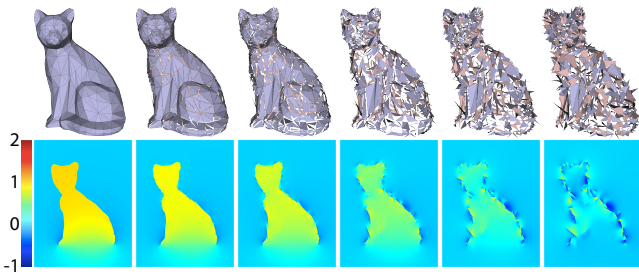


Figure 14: Each triangle of the Cat (originally with open bottom) is ripped off and slowly rotated in a random direction. The winding number gracefully degrades.

The *Ballet Woman* contains a very detailed mouth (see also *Ballet Woman's Head* in the supplemental video). Our meshing preserves these features while still correctly segmenting out the mouth cavity.

We report statistics in Table 1. Our timings were obtained on an iMac Intel Core i7 3.4GHz computer with 16GB memory. Our implementation is serial except for computing the winding number, which uses an OPENMP parallel for loop over the evaluation points. We tested the performance of our hierarchical evaluation versus a naive one with two experiments. First, we measured average computation time of a single evaluation in the bounding box of the *Dino* mesh under increasing subdivision levels (see Figure 10). Next we considered 700 (target) models of the SHREC dataset [Bronstein et al. 2010] (see Figure 11). For both experiments we average the computation time of 1000 random samples in the test shape's bounding box. Both experiments show that in general our hierarchical evaluation performs asymptotically better.

We stress tested our generalization of the winding number by considering how the function responds to degenerating input. The *Cat* in Figure 14 has an open base, and its winding number is a smooth (harmonic) field in $\approx[0, 1]$. We separate each triangle of the mesh and slowly rotate it in an arbitrary direction, evaluating the effect on the winding number. The winding number field maintains the image of cat until the triangles have rotated by π , when the mesh as a whole clearly breaks our consistent orientation assumption.

We compare our method to first repairing the input as a surface using [Attene 2010] and meshing the result (see Figure 15). The *Elephant's* ear flips inside-outside making volume determination badly ill-posed there: our method deletes the region creating a topological handle. Attene's MESHFIX deletes the region and then fills the hole with a different topology, but other parts of the mesh suffer: the tusks and eyes are also deleted. In Figure 18, [Attene 2010] fills the holes in the *Holey Cow* with the same topology as our method, but deletes the entire tail, which self-intersects its udder. Because our method avoids such drastic surface changes, we may compute a volumetric texturing using [Takayama et al. 2008] that meets the original surface (see Figure 16). One may then simply render the original surface and only show the inner texture when the *Tree* is cut.

In lieu of computing a volume discretization, many geometry processing tasks may be instead conducted on the surface. For example, the self-intersections in the *Beast* might have previously discouraged the use of a volumetric deformation technique due to the manual cleanup involved in preparing the model for tet meshing. Bending with surface-based technique reveals shell-like collapses when compared to a volumetric technique using a our volume discretization (see Figure 17). Some techniques like computing skinning weights automatically with [Jacobson et al. 2011] are designed specifically for volumes (see Figure 20). Without our method, this algorithm

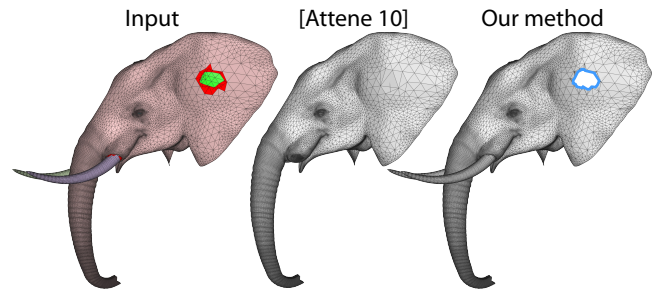


Figure 15: The ears of the Elephant Head overlap and flip inside-out (bright green) creating a negative volume. The result of [Attene 2010] creates a watertight surface, but the tusks and eyes are conspicuously missing. Our winding number identifies this region ($w < 0$), but our segmentation removes the region creating a hole (actually topological handle, blue).



Figure 16: The Tree contains many intersections and open boundaries (left). Our method is robust to these, producing a compatible mesh for applying volumetric texturing (right).

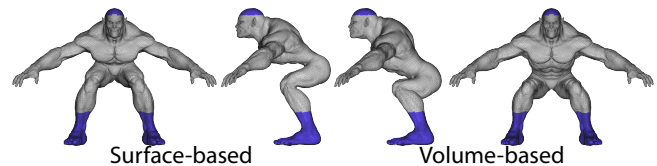


Figure 17: Self-intersections in the otherwise clean Beast prevent volume-meshing with previous methods. Surface-based deformation is one option, but bending causes shell-like collapses not present in a volume deformation enabled by our method.

has a limited set of inputs or requires tedious user preparation of input (defeating its automation gains). State of the art physically-based elasticity simulation techniques also require tetrahedral volume meshes. Our method accordingly expands the domain of inputs for these methods (see *Rubber Hippo* and *Rubber Chihuahua* in supplemental video).

7 Limitations and future work

The winding number and our generalization rely heavily on the orientation of input facets. Triangle soups with unknown or erroneous orientations would need further preprocessing (e.g. with [Borodin et al. 2004]). Since a single facet has a drastically different effect on the total winding number when its orientation agrees with its neighbors, it would be interesting to use the notion of our generalized winding number to verify or correct triangle orientations.

The number of connected components in our output is not controlled even when manifoldness is constrained. It would be interesting to extend the work of [Chen et al. 2011] to 3D, enabling such topological constraints in our graphcut segmentation.

Model name	Input model						Computation time				Output
	$ \mathcal{V} $	$ \mathcal{F} $	$ \partial\mathcal{F} $	#self-int.	#CC	#nme.	pre.	CDT	w	cut	$ \mathcal{E} $
Tree	2599	4067	1097	386	32	0	1.99	0.48	1.06	0.06	11643
Holey Cow	2632	5080	206	83	1	0	0.74	0.02	0.05	0.05	9232
Bikini Woman	2827	5204	477	472	11	24	2.29	0.22	0.60	0.06	13057
Ant	2859	5258	152	1578	62	1	7.14	0.48	0.59	0.06	18466
SWAT Man	5277	9820	551	2806	51	24	12.12	1.14	2.47	0.08	31317
Frog	6614	13216	0	316	3	0	1.75	0.39	1.57	0.06	21909
Dog	7953	15848	56	0	1	0	0.51	0.67	2.45	0.07	27707
Rhino	8071	16031	23	2150	26	0	10.29	0.73	4.37	0.10	74446
Alien Space-object	8762	17692	0	1686	32	0	13.41	0.74	7.86	0.10	57293
Skeleton	11963	21551	0	4095	206	0	25.17	4.19	31.87	0.28	217517
Flying Bug	12603	23932	1200	1731	25	0	9.12	1.77	8.69	0.10	62285
Crocodile	17332	34404	0	5236	65	0	22.33	0.20	6.88	0.13	98719
Bear	24936	23530	320	5572	37	0	24.62	0.15	5.38	0.15	56605
Beast	32311	64618	0	969	1	0	7.84	2.98	40.10	0.36	192613
Ballet Woman's Head	39068	76618	1146	8660	44	0	33.19	4.72	92.39	0.10	201991
Big SigCat	40224	60502	1020	3442	65	344	18.88	1.67	9.22	0.12	95896
Phone	42003	83998	0	1597	11	3	15.50	2.76	17.76	0.20	150159
Elephant Head	52740	105056	416	613	5	0	11.26	2.94	19.47	0.27	186025
Ballet Woman	70488	139324	1714	9734	44	0	45.95	7.07	153.23	0.83	615313

Table 1: Statistics for the various examples. $|\mathcal{V}|$ and $|\mathcal{F}|$ are the number of vertices and facets in the input 3D model. $|\partial\mathcal{F}|$ is the number of boundary edges, #self-int. the number of intersecting pairs of facets, #CC the number of connected components, and #nme. the number of non-manifold edges. We report timings for each stage of our algorithm in seconds: (pre.) pre-processing (dominated by self-intersection meshing), constructing a CDT with TETGEN (CDT), hierarchically evaluating the winding number w , and final graphcut segmentation (cut). The number of elements in the output tet mesh is $|\mathcal{E}|$.

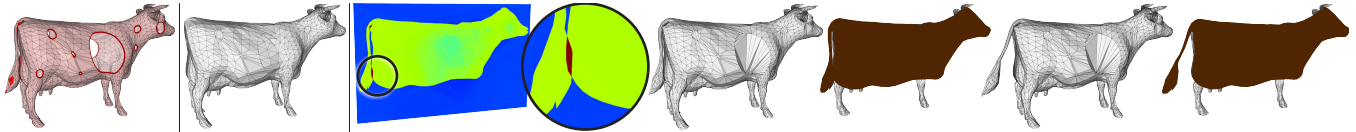


Figure 18: Left to right: Holey Cow with its tail intersecting its udder. [Attene 2010] fills the holes, but deletes the tail. A slice through the winding number shows correct assignment of 0 outside, 1 inside the main part, and 2 inside the overlapping tail (red), inset. This may be meshed as usual gluing the tail to the body. Or we may duplicate this doubly covered region and glue it to either side. This allows the tail and its volume to be pulled out.

Many meshes contain sheet-like features not part of the main solid body, such as leaves on a tree or cape on an action hero. Such features are typically *two-sided* and would require special treatment to consider the thin solids they represent. Conversely the accessories or nearly duplicated regions we do handle may also cause ambiguities. When duplicated surfaces nearly enclose a concavity, the winding number increases and may cause the region to be marked as inside (see Figure 19). The difference between inside and outside in these cases is a matter of semantics. To alleviate this, such accessories could be tagged as non-participatory for the winding number computation, but still constrained during our segmentation. Achieving such tagging automatically is an interesting direction for future work in the accelerating field of retrieving semantics from 3D shapes.

Our generalized winding number correctly identifies regions of overlap even in the presence of surface artifacts such as holes (see Figure 18). This suggests the ability to construct volume discretization that *respect* self-intersections of the original surface (rather than “correct” them). We show a proof-of-concept of this idea, by duplicating the meshing inside the *Holey Cow*’s overlapping tail (where $w \approx 2$) and gluing separately to the tail and body. The tail and its

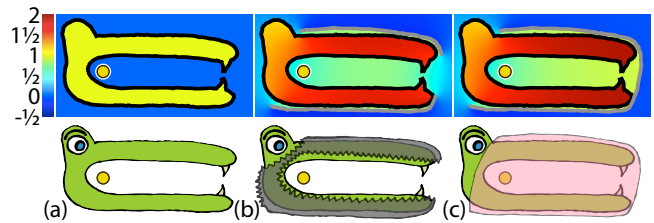


Figure 19: Inside-outside of the Snake (a) becomes ambiguous when thin sheets are used to represent accessories such as a ski-mask (b) or a muzzle (c). In (b) and (c) the winding number at the yellow points are similar, but the semantic inside-outside classification is opposite.

volume may then be deformed in and out of the body. More complicated overlaps are far from trivial to untangle and we continue to investigate this problem in our future work.

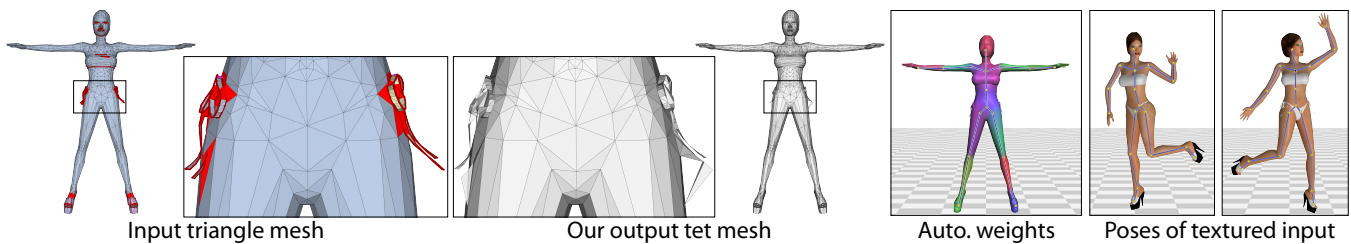


Figure 20: Left to right: the Bikini Woman has many artifacts, as well as thin sheet-like accessories. Their volumes are ambiguous, but our facet constraints ensure some trivial connection. This enables automatic, volumetric skinning weight computation [Jacobson et al. 2011] on a refinement of our output. Only the weights on the original vertices are needed to deform the original textured input mesh.

8 Conclusion

Generalizing the winding number to arbitrary triangle meshes proves to be a powerful and mathematically beautiful tool. The core of our method is simple to implement and our hierarchical acceleration structure enables efficient evaluation on large models. The winding number’s harmonic nature and implicitly defined, discontinuous boundary conditions make it ideal for guiding our graphcut segmentation when input meshes contain self-intersections, open boundaries, and non-manifold pieces. We hope that our algorithm’s success on previously *unmeshable* models will encourage volumetric processing of solid shapes throughout computer graphics.

Acknowledgements

We are indebted to Ilya Baran, Leo Guibas, Pierre Alliez, Alexander Sorkine-Hornung and Daniele Panozzo for illuminating conversations. We are grateful to Leonardo Koller Sacht, Fabian Hahn and Kaan Yücer for helping create results. Thanks to Josef Pelikan for recognizing the practical significance of winding numbers and including them in introductory computer graphics classes. Also, thanks to Marco Attene for making his MESHFIX program open source. This work was supported in part by the ERC grant iModel (StG-2012-306877), by an SNF award 200021_137879 and the Intel Doctoral Fellowship.

References

- ALLIEZ, P., COHEN-STEINER, D., YVINEC, M., AND DESBRUN, M. 2005. Variational tetrahedral meshing. *ACM Trans. Graph.* 24, 3.
- ALLIEZ, P., COHEN-STEINER, D., TONG, Y., AND DESBRUN, M. 2007. Voronoi-based variational reconstruction of unoriented point sets. In *Proc. SGP*.
- ATTENE, M., FERRI, M., AND GIORGI, D. 2007. Combinatorial 3-manifolds from sets of tetrahedra. In *Proc. CW*.
- ATTENE, M. 2010. A lightweight approach to repairing digitized polygon meshes. *The Visual Computer* 26, 11, 1393–1406.
- BARAN, I., AND POPOVIĆ, J. 2007. Automatic rigging and animation of 3D characters. *ACM Trans. Graph.* 26, 3, 72:1–72:8.
- BISCHOFF, S., AND KOBBELT, L. 2005. Structure preserving CAD model repair. *Comput. Graph. Forum* 24, 3, 527–536.
- BISCHOFF, S., PAVIC, D., AND KOBBELT, L. 2005. Automatic restoration of polygon models. *ACM Trans. Graph.* 24, 4.
- BORODIN, P., ZACHMANN, G., AND KLEIN, R. 2004. Consistent normal orientation for polygonal meshes. In *Proc. CGI*.
- BOYKOV, Y., AND FUNKA-LEA, G. 2006. Graph cuts and efficient ND image segmentation. *IJCV* 70, 2.
- BRIDSON, R., TERAN, J., MOLINO, N., AND FEDKIW, R. 2005. Adaptive physics based tetrahedral mesh generation using level sets. *Engineering with Computers* 21, 1, 2–18.
- BRONSTEIN, A. M., BRONSTEIN, M. M., CASTELLANI, U., FALCIDIENO, B., FUSIELLO, A., GODIL, A., GUIBAS, L. J., KOKKINOS, I., LIAN, Z., OVSJANIKOV, M., PATANÉ, G., SPAGNUOLO, M., AND TOLDO, R. 2010. SHREC 2010: robust large-scale shape retrieval benchmark. In *Proc. 3DOR*, 71–78.
- CAMPEN, M., AND KOBBELT, L. 2010. Exact and robust (self-)intersections for polygonal meshes. *Comput. Graph. Forum* 29, 2.
- CAMPEN, M., ATTENE, M., AND KOBBELT, L., 2012. A practical guide to polygon mesh repairing. Eurographics Tutorial.
- CGAL, Computational Geometry Algorithms Library. <http://www.cgal.org>.
- CHAR, B., GEDDES, K., AND GONNET, G. 1983. The Maple symbolic computation system. *SIGSAM Bull.* 17, 3–4, 31–42.
- CHEN, C., FREEDMAN, D., AND LAMPERT, C. 2011. Enforcing topological constraints in random field image segmentation. In *Proc. CVPR*.
- DAVIS, J., MARSCHNER, S. R., GARR, M., AND LEVOY, M. 2002. Filling holes in complex surfaces using volumetric diffusion. In *Proc. 3DPVT*, 428–438.
- DEY, T. K., AND GOSWAMI, S. 2003. Tight cocone: a water-tight surface reconstructor. In *Proc. SM*.
- FLOATER, M. S. 2003. Mean value coordinates. *Computer-Aided Geometric Design* 20, 1, 19–27.
- GEORGE, P. L., HECHT, F., AND SALTEL, E. 1990. Automatic 3D mesh generation with prescribed meshed boundaries. *IEEE Transactions on Magnetics* 26, 2, 771–774.
- GEUZAINÉ, C., AND REMACLE, J. F. 2009. GMSH: A 3-D finite element mesh generator with built-in pre- and post-processing. *Numerical Methods in Engineering*.
- GUÉZIEC, A., TAUBIN, G., LAZARUS, F., AND HOM, B. 2001. Cutting and stitching: Converting sets of polygons to manifold surfaces. *IEEE TVCG* 7, 2.
- HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. 1992. Surface reconstruction from unorganized points. In *Proc. ACM SIGGRAPH*.

- HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. 1993. Mesh optimization. In *Proc. ACM SIGGRAPH*.
- HORNUNG, A., AND KOBBELT, L. 2006. Robust reconstruction of watertight 3D models from non-uniformly sampled point clouds without normal information. In *Proc. SGP*.
- HOUSTON, B., BOND, C., AND WIEBE, M. 2003. A unified approach for modeling complex occlusions in fluid simulations. In *ACM SIGGRAPH 2003 Sketches & Applications*.
- JACOBSON, A., BARAN, I., POPOVIĆ, J., AND SORKINE, O. 2011. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.* 30, 4.
- JOSHI, B., AND OURSELIN, S. 2003. BSP-assisted constrained tetrahedralization. In *Proc. IMR*, 251–260.
- JU, T., SCHAEFER, S., AND WARREN, J. 2005. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph.* 24, 3.
- JU, T. 2004. Robust repair of polygonal models. *ACM Trans. Graph.* 23, 3.
- JU, T. 2009. Fixing geometric errors on polygonal models: a survey. *Journal of Computer Science and Technology* 24, 1.
- KAZHDAN, M., BOLITHO, M., AND HOPPE, H. 2006. Poisson surface reconstruction. In *Proc. SGP*.
- KLINGNER, B. M., AND SHEWCHUK, J. R. 2007. Agressive tetrahedral mesh improvement. In *Proc. IMR*.
- KOLLURI, R., SHEWCHUK, J., AND O'BRIEN, J. 2004. Spectral surface reconstruction from noisy point clouds. In *Proc. SGP*.
- KOLMOGOROV, V., AND ZABIN, R. 2004. What energy functions can be minimized via graph cuts? *IEEE PAMI* 26, 2.
- KRAEVOY, V., SHEFFER, A., AND GOTSMAN, C. 2003. Matchmaker: constructing constrained texture maps. *ACM Trans. Graph.* 22, 3.
- LABELLE, F., AND SHEWCHUK, J. R. 2007. Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. *ACM Trans. Graph.* 26, 3.
- MCADAMS, A., ZHU, Y., SELLE, A., EMPEY, M., TAMSTORF, R., TERAN, J., AND SIFAKIS, E. 2011. Efficient elasticity for character skinning with contact and collisions. *ACM Trans. Graph.* 30, 37:1–37:12.
- MEISTER, A. 1769/70. Generalia de genesi figurarum planarum et inde pendentibus earum affectionibus. *Novi Comm. Soc. Reg. Scient. Gotting.*, 144–180+9 plates.
- MULLEN, P., DE GOES, F., DESBRUN, M., COHEN-STEINER, D., AND ALLIEZ, P. 2010. Signing the unsigned: Robust surface reconstruction from raw pointsets. *Comput. Graph. Forum* 29, 5.
- MURALI, T. M., AND FUNKHOUSER, T. A. 1997. Consistent solid and boundary representations from arbitrary polygonal data. In *Proc. 13D*.
- NOORUDDIN, F. S., AND TURK, G. 2003. Simplification and repair of polygonal models using volumetric techniques. *IEEE TVCG* 9, 2.
- ORZAN, A., BOUSSEAU, A., WINNEMÖLLER, H., BARLA, P., THOLLOT, J., AND SALESIN, D. 2008. Diffusion curves: a vector representation for smooth-shaded images. *ACM Trans. Graph.* 27, 3, 92:1–92:8.
- PODOLAK, J., AND RUSINKIEWICZ, S. 2005. Atomic volumes for mesh completion. In *Proc. SGP*.
- SCHÖBERL, J. 1997. NETGEN: An advancing front 2D/3D-mesh generator based on abstract rules. *Computing and Visualization in Science*.
- SHEN, C., O'BRIEN, J. F., AND SHEWCHUK, J. R. 2004. Interpolating and approximating implicit surfaces from polygon soup. *ACM Trans. Graph.* 23, 3, 896–904.
- SHEWCHUK, J. R. 1996. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, vol. 1148 of *Lecture Notes in Computer Science*. 203–222.
- SHEWCHUK, J. 2012. Unstructured mesh generation. *Combinatorial Scientific Computing* 12, 257.
- SHIMADA, K., AND GOSSARD, D. C. 1995. Bubble mesh: automated triangular meshing of non-manifold geometry by sphere packing. In *Proc. SMA*, 409–419.
- SI, H., 2003. TETGEN: A 3D Delaunay tetrahedral mesh generator. <http://tetgen.berlios.de>.
- TAKAYAMA, K., OKABE, M., IJIRI, T., AND IGARASHI, T. 2008. Lapped solid textures: Filling a model with anisotropic textures. *ACM Trans. Graph.* 27, 3.
- TURK, G., AND LEVOY, M. 1994. Zippered polygon meshes from range images. In *Proc. ACM SIGGRAPH*.
- VAN OOSTEROM, A., AND STRACKEE, J. 1983. The solid angle of a plane triangle. *IEEE Trans. Biomedical Engineering* 30, 2.
- WAN, M., WANG, Y., AND WANG, D. 2011. Variational surface reconstruction based on Delaunay triangulation and graph cut. *Int. Journal of Numerical Engineering*.
- WAN, M., WANG, Y., BAE, E., TAI, X., AND WANG, D. 2012. Reconstructing open surfaces via graph-cuts. *IEEE TVCG* 19, 2.
- YAMAKAWA, S., AND SHIMADA, K. 2009. Removing self intersections of a triangular mesh by edge swapping, edge hammering, and face lifting. In *Proc. IMR*.
- ZHANG, L., CUI, T., AND LIU, H. 2009. A set of symmetric quadrature rules on triangles and tetrahedra. *J. Comput. Math* 27, 1, 89–96.
- ZHANG, S., NEALEN, A., AND METAXAS, D. 2010. Skeleton based as-rigid-as-possible volume modeling. In *Proc. Eurographics, short papers volume*, 21–24.
- ZHOU, K., HUANG, J., SNYDER, J., LIU, X., BAO, H., GUO, B., AND SHUM, H.-Y. 2005. Large mesh deformation using the volumetric graph Laplacian. *ACM Trans. Graph.* 24, 3, 496–503.
- ZHOU, K., ZHANG, E., BITTNER, J., AND WONKA, P. 2008. Visibility-driven mesh analysis and visualization through graph cuts. *IEEE TVCG* 14, 6.

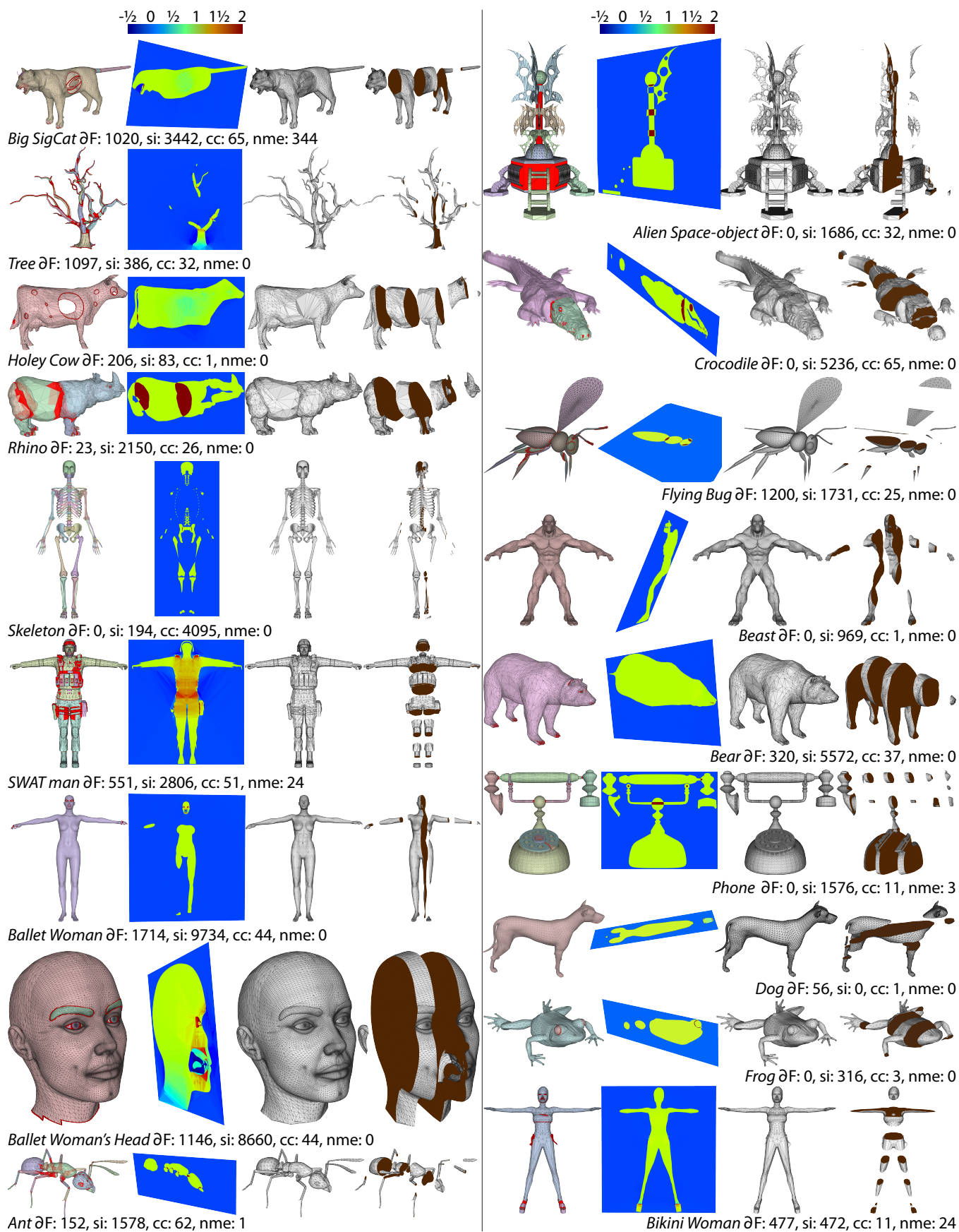


Figure 21: Each row shows left to right: input model with connected components randomly colored, self-intersections facets marked in red, open boundaries in dark red and non-manifold edges in purple; slice through CDT visualizing winding number; surface of output mesh; hot-dog slice view of output mesh.