DOI: 10.1111/cgf.15141
Eurographics Symposium on Geometry Processing 2024
R. Hu and S. Lefebvre
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 43 (2024), Number 5

Winding Number Features for Vector Sketch Colorization

Daniel Scrivener[1], Ellis Coldren[1], Edward Chien[1]

[1]Boston University

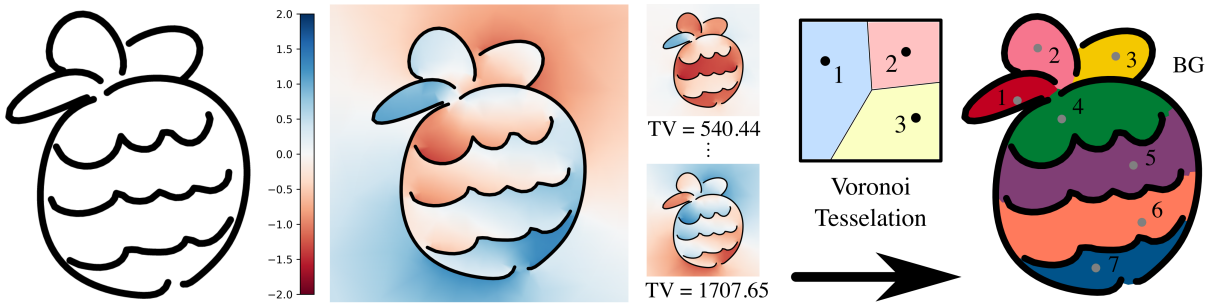


Figure 1: *Lineart vector sketches often feature potential stroke junctions that are hard to disambiguate. We sidestep this challenge by leveraging winding number features, which instead focus on capturing a notion of region closure. Our resulting multi-region segmentation method outperforms existing methods on inputs with a diverse set of gap sizes. (Left) An uncolored vector sketch. (Middle Left) The winding number generated by the sketch's default stroke directions, unsuitable for clustering on its own. (Middle Right) Several winding numbers for random sets of stroke orientations, sorted by total variation, which is used to differentially weight certain features. (Right) The automatically colored sketch after applying k-means clustering.*

Abstract

Vector sketch software (e.g. Adobe Illustrator, Inkscape) and touch-interactive technologies have long aided artists in the creation of resolution-independent digital drawings that mimic the unconstrained nature of freehand sketches. However, artist intent behind stroke topology is often ambiguous, complicating traditional segmentation tasks such as coloring. For inspiration, we turn to the winding number, a classic geometric property of interest for binary segmentation in the presence of boundary data. Its direct application for multi-region segmentation poses two main challenges: (1) strokes may not be consistently oriented to best identify perceptually salient regions; (2) for interior strokes there is no "correct" orientation, as either choice better distinguishes one of two neighboring regions. Thus, we form a harmonic feature space from multiple winding number fields and perform segmentation via Voronoi/power diagrams in this domain. Our perspective allows both for automatic fill region detection and for a semi-automatic framework that naturally incorporates user hints and interactive sculpting of results, unlike competing automatic methods. Our method is agnostic to curve orientation and gracefully handles varying gap sizes in the sketch boundary, outperforming state-of-the-art colorization methods on these "gappy" inputs. Moreover, it inherits the ability of winding numbers to specify "fuzzy" boundaries, leading to simple strategies for color diffusion and single-parameter-driven growing and shrinking of regions.

CCS Concepts

• ***Computing methodologies*** → ***Image manipulation;*** *Shape analysis;*

1. Introduction

Vector representations of freehand sketches are popular among artists, finding use in a variety of commercial software packages. Such formats are particularly appealing as they can be rendered at any scale without loss of detail. In this setting, the lack of explicit stroke connectivity constraints imposed on the user eludes classic segmentation strategies (e.g. flood fill) developed for raster images. For example, artists may produce over- or under-drawn strokes that extend past or fail to meet the intended point of intersection with another stroke [YLL*22]. As such, most state-of-the-art segmentation methods focus on detecting gaps and clas-

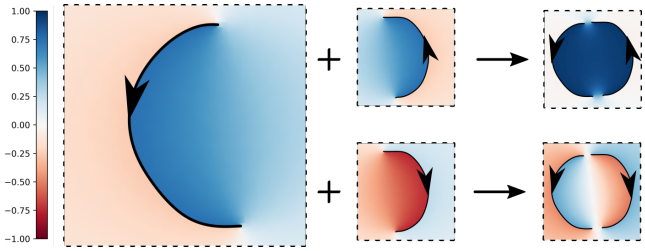


Figure 2: *The winding number is the sum of harmonic functions defined with respect to individual strokes [JKS13]. In this pedagogical example of a two-stroke circle, a naïve use of the winding number is effective only if the artist's stroke orientations are consistent (top), and ineffective if not (bottom). A priori, an artist is unaware of this and has no reason to choose one over the other.*

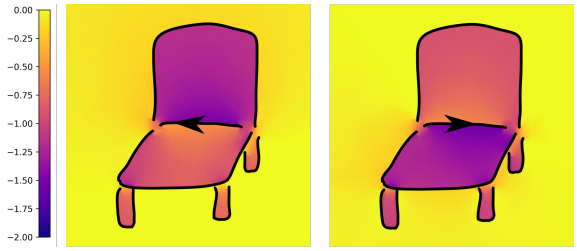


Figure 3: *In this chair sketch, no canonical direction exists for the center stroke. Either choice of direction induces greater variation in the winding number across the lower or upper sections (left and right respectively).*

sifying junctions that form the boundaries between regions in a sketch [ZCZ*09, FTR18, PMC22, YLL*22].

In contrast, we use a collection of winding numbers associated with the strokes/curves of a sketch. First used in computer graphics for solving the point-in-polygon problem [HA01], the *winding number* corresponds to the number of times a closed curve encircles a given query point. This property generalizes well to curves with gaps or over-extensions, providing a sense of *partial* inclusion with confidence imparted by the strength of the winding number at a particular point [JKS13, BDS*18]. The ability of the winding number to predict region closure is analogous to the manner in which humans perceptually distinguish fill regions among a series of disconnected curves [FTR18, YLL*22, Kof35].

Two aspects of the classic winding number impede direct use in segmentation. First, the winding number's orientation-specificity means that one might rely on the artist to orient strokes in a coherent, correct manner to differentiate inside from outside, as illustrated in Fig 2. In practice, there is no apparent reason for artists to make conscious decisions regarding stroke direction while sketching. Furthermore, fixing such orientations in post-processing is a challenging, ongoing problem, tackled by related works in other (e.g., point cloud) contexts [TJKSH14, MHZ*21, XDW*23]. Second, in the setting of multi-region segmentation, there is no "correct" orientation for *interior* strokes of the drawing. Different choices lead to better winding number identification of one of the regions on either side of the stroke, as shown in Fig. 3.

We present a framework for multi-region segmentation of lineart vector sketches that surmounts these difficulties by using multiple stroke orientations and their resulting winding numbers at once, as "features" (axes) in an embedding space. These winding number fields are differentially weighted by their total variation (TV), capturing how consistent they are over broad regions of the sketch, and thus how useful they are as segmentation features. Voronoi/power-diagram-based clustering is then used to determine the desired flat-fill regions of our segmentation. Our framework:

- naturally accommodates both automatic and semi-automatic segmentation methods
- requires no training data
- inherits a notion of region confidence from the classic winding number, demonstrated by a boundary sculpting tool (§4.3.3) and a proof-of-concept diffuse coloring method (§5.3)

Our automatic method achieves superior performance on "gappy" inputs with a diverse range of gap sizes, and performs comparably otherwise. Inputs of this kind are representative of doodles or sketches from users with a broader range of skill sets, and our approach makes effective flat-fill tools available to this wider audience. Additionally, we found that *all* automatic methods are prone to undesirable errors, so we find it to be a particular strength of our framework that it allows for simple, intuitive modification of the results, in contrast to competing automatic methods. The semi-automatic method allows user input in the form of both color hints and scribbles as well as easy modification of region boundaries via power diagram weightings. We achieve colorings comparable to those of other frameworks with minimal input.

2. Related Work

2.1. Sketch segmentation and colorization

We give a brief overview of the many works aimed at sketch segmentation and colorization with both vector and raster input. Our method stands out among existing works for its "boundary-focused" approach that explicitly considers the global effect of each stroke element.

Levin et al. develop a landmark method for image colorization using least-squares optimization to propagate user-provided color hints across regions of similar intensity in black and white raster images [LLW04]. Color hints are an intuitive mode of user interaction, and inspire our work as well as several others [QWH06, SBv05, SDC09, PCS21, PMC22, ZLW*18, ZLSS*21, FTR18].

LazyBrush is a semi-automatic, user-guided coloring tool that generalizes to a wider variety of artistic styles than the works described above [SDC09]. They solve for an optimal raster coloring via a modified Potts energy that aims to align region boundaries with areas of low intensity and softly satisfies color hints. Several limitations are noted by the authors: most notably, their method penalizes boundaries that trace long, highly concave strokes, preventing effective filling of these regions (see Fig. 8B of [SDC09]). In contrast, our method explicitly retains stroke geometry in computation of the winding number features, and is effective in such scenarios. In supplementary §12.1.3, we recreate Fig. 8B of [SDC09]

and show that LazyBrush does not handle this input appropriately, whereas our method succeeds.

Zhang et al. propose a variation on flood-fill known as the "trapped ball method" [ZCZ*09]. A radius r ball fills a region from an initial start point by moving freely in all directions, but cannot pass through gaps smaller than $2r$. The method relies on effective tuning of an initial r value and the rate at which it decreases between iterations, both of which are sensitive to any large diversity in gap sizes over the image. Conversely, our fill regions evolve gradually relative to local gap size as a result of the "confidence" imparted by the winding number.

Fourey et al. interactively segment raster sketches by proposing a low-complexity gap closure method [FTR18]. Curvature and stroke-normal-based quality measures are used to rank candidate gap closure curves, which are drawn above a certain quality threshold. Accurate estimation of stroke endpoint normals is crucial to the success of this method, which may be confounded by the presence of "hooklike" artifacts at the end of strokes [YLL*22]. Such imperfections are also reflected in the winding number, but they do not have a pronounced *global* effect on the manner in which the winding number identifies fill regions.

Yin and Liu et al. develop a learning-based method for identifying junctions between strokes, broadly applicable to various segmentation tasks [YLL*22]. They construct a family of sketch features from local/global geometric properties such as stroke thickness, inter-stroke distance, and tangency at stroke endpoints. Random forest classifiers are trained on human-annotated sketches to differentiate between end-to-end and T-junctions. The sketch is segmented by completing the gaps around each detected junction. We also note the existence of deep learning methods for sketch coloring [ZLW*18, ZLSS*21], but these are typically aimed at *diffuse* coloring and do not produce strict multi-region segmentations. Unlike these methods, our approach is not learning-based and is not dependent on training data.

Finally, in [PCS21, PMC22], Parakkat et al. use a constrained Delaunay triangulation, with all vertices lying either on strokes or a rectangular "frame" surrounding the image, and partition its triangles to determine flat-fill regions. Building upon their vectorization work [PBM18], they propose a dual-graph-based method where triangles are partitioned via max flow from color hints. Flow capacity of a dual edge is defined by the shared edge length between two triangles. Automatic color hints may be generated by placing pairs of hints on either side of candidate region boundaries. Their choice of triangulation strongly restricts the possible gap closures (see Fig. 23 of their work) and does not allow for natural shifting of region boundaries via some notion of confidence. In our method, we use a constrained *conforming* Delaunay triangulation (see §4.1), which samples the domain in a locally-adaptive manner and gives a broader choice of gap closures. Region boundaries are inferred via winding numbers, which have a natural notion of region confidence as leveraged in the boundary refinement of §4.3.

2.2. Orienting boundary elements

Many recent works have focused on orientation of boundary elements in the setting of binary inside-outside segmentation, as a precursor to application of methods such as Poisson surface reconstruction [KBH06]. We cover several below, while also noting that none have considered orientation in the setting of *multi-region segmentation*.

Following the groundbreaking use of the winding number in inside-outside segmentation of triangle meshes [JKS13], Takayama et al. explore boundary element configurations that minimize Dirichlet energy for binary segmentation [TJKSH14]. It was noted, however, that this objective fails to penalize sign changes between objects that are "attached" along some boundary. Metzer et al. obtain a set of surface patches from an input point cloud whose orientations are first determined with respect to *local* geometry by a neural network [MHZ*21]. Then, the *global* orientation of patches is propagated throughout the point cloud according to an electric-dipole-based energy. Equivalently, Barill et al. formulate the winding number for point clouds as the sum of these dipole contributions per oriented point [BDS*18]. Intuitive properties of the winding number are leveraged by Xu et al. in their approach to normal assignment [XDW*23]. Guided by the insights that the ideal winding number should be (1) sharply differentiated between 0 and 1 and (2) balanced evenly across the vertices of each Voronoi cell, they regularize the winding number field according to these objectives.

2.3. Sketch vectorization

The problem of sketch vectorization: that of extracting and refining a curve network that best represents a raster sketch, has a vast literature, e.g. [NHS*13, ZCZ*09, FLB16, SSII18, PBM18, BS18, PNCB21, GHJB*23]. Our method or another [YLL*22] may be applied to the vector output of such methods for segmentation of raster input. However, such a pipeline subjects the latter methods to potential imperfections in the vectorizations, so we focus on the problem of vector input and do not perform such experiments in this article. For the interested reader, Yan et al. conduct a comprehensive survey and analysis of state-of-the-art sketch vectorization methods [YVG20].

3. Background

We briefly review winding numbers, total variation (TV), and power diagrams below. For further reading, we recommend [JKS13, BDS*18] (winding numbers), [AFP00] (TV), and [Aur87] (power diagrams).

3.1. Winding number

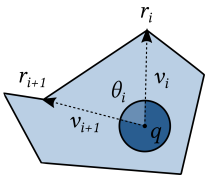
Given a curve $\gamma : [0,1] \to \mathbb{R}^2$ in the plane, the winding number w of a point $p \in \mathbb{R}^2 \setminus \gamma([0,1])$ specifies how many times the curve winds around p in the counterclockwise direction. In particular, it is defined as the integral of the signed angle rotated by an observer located at p, tracking the curve γ:

$$w(p) = \frac{1}{2\pi} \int_\gamma d\theta$$

If γ is discretized as a polyline given by a sequence of points $r_0, r_1, \ldots, r_n$, then the natural discretization for this integral is:

$$w(p) = \frac{1}{2\pi} \sum_{i=0}^{n-1} \theta_i, \tag{1}$$

where $\tan(\theta_i) = \frac{\|v_i \times v_{i+1}\|}{v_i \cdot v_{i+1}}$ is the angle subtended by each segment of the curve, and $v_i = r_i - p$ for all i. This is illustrated in the following inset figure.



As a function itself, the winding number w is harmonic on its domain $\Omega := \mathbb{R}^2 \setminus \gamma([0,1])$. Some examples with single curves are shown in Fig. 2. It (and its 3D/surface generalizations) have found much use as a way of robustly partitioning a space into an inside and outside given a boundary representation [JKS13, BDS*18, XDW*23, FGC23], and this fundamental task forms the basis for many higher-level geometry processing operations. When the boundary curve is closed, the winding number takes on integer values, and will also detect "overlaps" in the region bounded by the curve. In the case of open or noisy curves, the gradual transition in values from interior to exterior gives a natural confidence measure on inclusion of a point. We leverage a similar notion of confidence measure in our framework (see §4.3.3, §5.3).

When γ is discretized, w is the *sum* of the harmonic winding numbers associated with each polyline segment $\overrightarrow{r_i r_{i+1}}$, whereby double-sided boundary conditions of $\pm 1/2$ are placed on the left/right, respectively. This may be seen as an instance of the boundary element method (BEM) for solving the Laplace equation with double-sided Dirichlet boundary conditions on Ω. Note that the *resultant* boundary conditions are not $\pm 1/2$, but are instead the limiting winding number values as one approaches γ (Fig. 2).

3.2. Total variation (TV)

We use total variation (TV) of a winding number to measure its utility for our segmentation framework. For $w : \Omega \subset \mathbb{R}^2 \to \mathbb{R}$

$$\mathsf{TV}(w) := \sup_{\phi \in C_c^1(\Omega,\mathbb{R}^2), ||\phi||_{L^\infty(\Omega)} \le 1} \left\{ \int_\Omega w(x) \operatorname{div}\phi(x)\, dx \right\}. \quad (2)$$

In the above, ϕ is a compactly-supported differentiable vector field, which has norm no greater than 1 everywhere.

In our setting (see §4.1), we discretize our winding numbers as piecewise-linear over a triangulation of the sketch complement Ω. Following [ROF92], one applies integration-by-parts and the divergence theorem to Eq. (2), to arrive at the more interpretable:

$$\mathsf{TV}(w) := \sum_{T \in M} \text{Area}(T) \|\nabla w\|. \quad (3)$$

This energy rewards winding numbers that are relatively constant in large open spaces and only change quickly in small gaps (see Fig. 4). Such winding numbers only have significant gradient in these gaps, and thus have relatively low TV.

3.3. Power diagrams

To allow fill regions to grow and shrink in a natural fashion (§4.3.3), we leverage *power diagrams*, a generalization of Voronoi diagrams. One assigns a *power radius* ρ_i to each seed point μ_i and considers the *power distance* (squared) to each seed point $d_\rho^2(p,\mu_i) := \|p - \mu_i\|^2 - \rho_i^2$. The *power cells* are determined by minimal power distance:

$$B_\rho(\mu_i) := \{p \in \mathbb{R}^n \mid d_\rho^2(p,\mu_i) \le d_\rho^2(p,\mu_j)\ \forall j \ne i\} \quad (4)$$

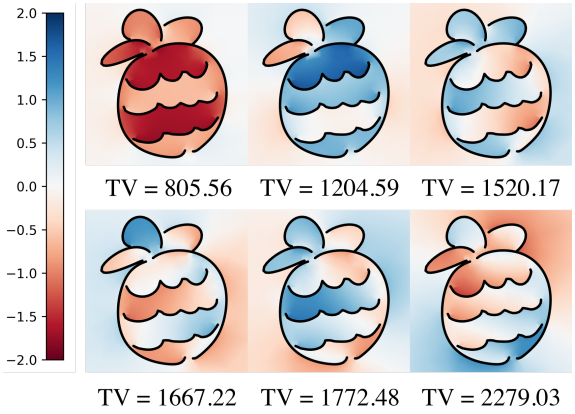


Figure 4: *Sampled winding numbers sorted by TV. Six randomly sampled stroke configurations (with varying stroke orientations) and their resulting winding numbers, sorted by total variation (TV) are shown. As desired, the winding numbers with lower TV are more informative for clustering, as they are relatively constant in large, open areas, and change quickly at gaps, between regions.*

As one increases/decreases the power radius ρ_i, the power cell $B_\rho(\mu_i)$ grows/shrinks, with cell boundaries remaining orthogonal to the line between the appropriate seed points. If $\rho = [\rho_1, \ldots, \rho_k]$ is a constant vector (ρ_i equal for all i), we recover the standard Voronoi diagram.

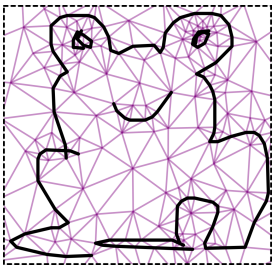
4. Method

The input to our algorithm is a vector sketch encoded by a set of strokes $\mathcal{S}$, with optional point or scribble color hints specified by the user $H = H_P \cup H_S$. For ease of processing, we frame each sketch with an axis-aligned bounding box $B \subset \mathbb{R}^2$ extended in all directions by $\min(\Delta x, \Delta y)$, where Δx and Δy are the sketch width and height respectively. This margin is large enough to allow the winding number to approach zero on the exterior, as desired, and can easily be obscured from the user. The output of our method is a set of filled polygons corresponding to the regions of the sketch.

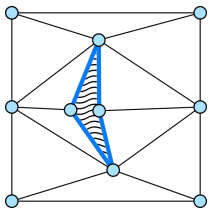
In our implementation, each stroke $s \in \mathcal{S}$ is a polyline, a set of points $r_0, r_1, \ldots, r_n \in B$ with inferred edges between each pair of adjacent points. We regard this particular representation as interoperable with alternatives such as Bézier curves (via sampling). We assign a set of orientation labels $d \in \{-1,+1\}^{|\mathcal{S}|}$ to $\mathcal{S}$, distinguishing the original direction in which a given stroke was traced (+1) and the opposite direction (-1). Under this scheme, the artist's original strokes correspond to the labeling $\{+1\}^{|\mathcal{S}|}$. We later explore the space of stroke orientations on which to perform clustering in §4.2.1. Each *point* color hint $h_p = (q_p, c_p) \in H_P$ consists of a point $q_p \in B$ and a color label $c_p \in \mathbb{N}$. Similarly, each *stroke/"scribble"* color hint $h_s = (\{q_0^s, q_1^s, \ldots, q_n^s\}, c_s) \in H_S$ consists of a polyline, or sequence of points, $\{q_0^s, q_1^s, \ldots, q_n^s\} \in B$ and a color label $c_s \in \mathbb{N}$.

In order to approximate the continuous winding number field over the sketch, we compute the constrained *conforming* Delaunay triangulation of the complement of the sketch strokes $\Omega := B \setminus \mathcal{S}$ using Triangle [She96]. This yields a

2D mesh $M = (V,E,T)$ of vertices V, edges E, and triangles T. Triangle inserts additional points not present in the original sketch, both on stroke boundaries and in the interior [She02], in order to satisfy the Delaunay property for all triangles [Che87] while maintaining a guaranteed minimum angle of 20 degrees. This default angle is a conservative choice based on a theoretically achievable guarantee of 25.65 degrees ([She97], §3.4.2). Empirically, the resulting point sampling is denser near the stroke boundary, mirroring greater variation in the winding number.



To capture the jump discontinuity of the winding number across strokes, we further modify $M = (V,E,T)$ and "cut" the mesh along the path traced by each stroke, duplicating vertices as they are encountered. See the inset figure for a schematic of this process. The split vertices along the stroke are not actually moved apart; we simply illustrate the cut with a gap.



4.1. Winding number calculation

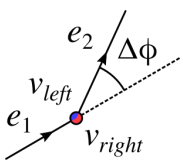
Given a set of stroke orientations d, we compute the winding number as the sum of per-stroke contributions:

$$w^d(v) = \sum_{i=0}^{|\mathcal{S}|-1} d_i w_i(v) \tag{5}$$

where w_i denotes the winding number of stroke i's *default* orientation, considered independently of other sketch strokes. Decomposing the calculation in this manner allows for quick generation of new winding numbers for different sets of stroke orientations.

For interior vertices, the winding number is computed as the sum of signed angles subtended by each polyline edge (Eq. (1)). Piecewise linear interpolation is performed to approximate winding numbers on the interior of triangles. For query points outside the axis-aligned bounding box of a particular stroke, this stroke's contribution to the winding number is identical to that of the line segment between its endpoints [JKS13]. We use this approximation of the sketch geometry where appropriate to improve performance.

For points v that lie on the interior of strokes, our calculation must be adjusted. See inset for a schematic. The contribution to the winding number from the two edges e_1, e_2 that share v as an endpoint is:



$$\tilde{w}_i(v) = \frac{\Delta\phi \pm \pi}{2\pi} \quad \text{for } v_{left/right} \tag{6}$$

where $\Delta\phi$ is the turning angle between e_1, e_2. Contributions from other edges remain the same.

Furthermore, the true winding number at stroke endpoints is undefined, and limits to different values depending on the direction in which it is approached (see e.g. §2.3.2 of [FGC23]). To ensure finite total variation, we simply use a weighted average of the neighboring winding numbers at stroke endpoints. This approximation is sufficient for our purposes and leads to reasonable region boundaries in our results.

$$w_i(v) = \frac{\sum_{n\in\mathcal{N}(v)} \frac{w_i(n)}{\|v-n\|}}{\sum_{n\in\mathcal{N}(v)} \|v-n\|^{-1}} \quad \text{if } v \text{ is an endpoint of stroke } i \tag{7}$$

Our simple implementation of the winding number calculation does not introduce a performance bottleneck and facilitates quickly recomputing the winding number for many arbitrary sets of stroke orientations. A fast multipole method might be used to accelerate this calculation [BDS*18], but it is less apparent that its gains would be worthwhile with input of a single stroke. Similarly, simpler hierarchical strategies with less overhead may be considered (§4.3 of [JKS13]). We refer readers to §5.4 for a detailed breakdown on the runtime of our method for sketches of varying complexity.

4.2. Clustering

To segment Ω into a set of color regions, we perform Voronoi/power-diagram-based clustering in a feature space comprising multiple winding numbers. The winding number fields are denoted $\{F_0,\ldots,F_{n-1}\}$, each of which corresponds to a set of stroke orientations generated according to §4.2.1. Ω is embedded into the space via its triangulation M, with vertex embeddings:

$$\mathcal{E}(v) \mapsto \left[\alpha_0 F_0(v),\ldots,\alpha_{n-1}F_{n-1}(v)\right]^T, \tag{8}$$

where $\alpha_i \in \mathbb{R}^+$ denote feature weights described below in Eq. (9). The vertex embeddings are piecewise-linearly interpolated to obtain an embedding of M.

4.2.1. Feature generation and weighting

We first uniformly sample $m = \min(100, |\mathcal{S}|)$ *random* stroke configurations $\{d_0,\ldots,d_{m-1}\}$ and calculate a winding number field for each: $F_i(v) = w^{d_i}(v)$. Each such feature F_i is then scaled by a weight α_i representing the normalized relative total variation:

$$\tilde{\alpha}_i = \max_{F_j}\{\mathsf{TV}(F_j)\} - \mathsf{TV}(F_i) \quad \& \quad \alpha_i = \frac{\tilde{\alpha}_i}{\sum_{j=0}^{m-1}\tilde{\alpha}_j}. \tag{9}$$

This weighting ensures that winding numbers with lower TV have greater influence on the segmentation. As shown in Fig. 4, these are fields with variation concentrated in small gaps rather than in large, open areas. In supp. §7, we present an ablation study where TV weighting is removed, showing clearly that it improves the quality of clusters.

As for m, we established a lower bound of 100 as a conservative choice, based on experiments (supp. §4) that showed minimal dependence on this parameter beyond a certain threshold. To accommodate larger sketches, we also asked that m at least scale with the number of strokes. There are also additional supplementary experiments that show the method is relatively invariant to the random sampling of features (supp. §9).

4.2.2. Semi-automatic method

Our semi-automatic method colors a sketch according to any user-provided hints $H = H_P \cup H_S$. We initialize seed points μ_i for a Voronoi diagram from these hints in the following manner:

- Point hints $h_p = (q_p, c_p) \in H_P$ result in a single seed point $\mu_p = \mathcal{E}(q_p)$ with color label c_p.
- Scribble hints $h_s = (\{q_0^s, \ldots, q_n^s\}, c_s) \in H_S$ result in multiple seed points whose Voronoi cells are later merged. In particular, there is a seed point for each triangle t containing at least one point in h_s. For each such triangle, we take the average of all $q_i^s \in t$, denoted $\bar{q}_t$, and initialize a seed point $\mu_t^s = \mathcal{E}(\bar{q}_t)$ with color label c_s.

As multiple point and scribble hints may have the same color labels, we then merge all clusters that share a label. This basic approach may form the first part of a pipeline that continues with a connectivity-preserving post-processing step (§4.2.4) and simple editing/sculpting operations (§4.3). As such post-processing and editing/sculpting steps may also be applied to our automatic output, we describe our automatic pipeline first as follows.

4.2.3. Automatic coloring and cluster initialization

As an alternative to manual specification of hints, we also propose a fully automatic coloring method based upon the efficient k-means clustering algorithm. Supplementary section §1 provides a brief overview of Lloyd's algorithm [Llo82] and k-means. One minor change from the base algorithm is that, per iteration, we compute centroids as an *area-weighted* average of the vertices associated with a cluster, as befits our particular application.

User selection of k is important to the overall success of the automatic method, but we feel this parameter is both flexible and intuitive in our setting and framework. First, k is simple to estimate, constituting a rough guess at the number of desired color regions. Color regions themselves are perceptually defined, and there is no objectively correct number; a user's estimate captures their perceptual reasoning. Second, owing to the speed of k-means, rapid iteration across different values of k is convenient and could be realized with a simple "slider" UI element (see §5.4 for timing information). Lastly, if the user does not wish to modify k, over/under-segmentation are easily fixed via the editing/sculpting operations of §4.3. To support these points, we provide experiments demonstrating behavior with respect to varying k (Fig. 15, supp. §8) and a supplementary video demonstrating the editing/sculpting operations.

In addition, we pair k-means with a custom strategy for initializing seed points. We utilize properties unique to our constrained conforming Delaunay triangulation: namely, that triangles become larger as distance to the nearest stroke increases (i.e. within fill regions) and that the *largest* triangles appear on the rectangular frame boundary. Our strategy is inspired by k-means++ [AV07], a randomized strategy for seed initialization, and by the graph-based approach in [PCS21, PMC22]. The following strategy is deterministic and initializes a set of seed points $U = \{\mu_0, \ldots, \mu_{k-1}\}$:

1. In order to avoid over-segmentation of the background, we first generate a mask of background triangles $T_{BG} \subseteq T$; full details are provided in supp. §2.

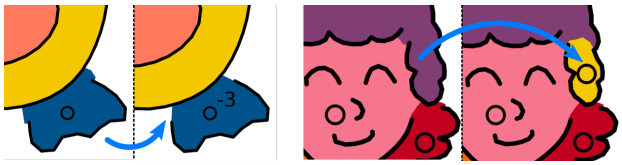


Figure 5: *Examples of user refinements to an automatically generated coloring. (Left) The user adjusts the power radius of the centroid associated with a color hint, eliminating slight color spill. (Right) The user places an additional color hint to segment a finer detail of the sketch.*

2. The *first* seed point μ_0 is initialized on the barycenter of the sketch's largest triangle, which will be in T_{BG}.
3. For following seed points $\mu_1, \ldots, \mu_{k-1}$, we score all triangles $t \in T \backslash T_{BG}$ as follows:

$$score_i(t) = \min_{j \in \{0,\ldots,i-1\}} \left\{ \|\mu_j - \mathcal{E}(\text{Bary}(t))\|^2 \right\} \times \sqrt{\text{Area}(t)},$$

where $\text{Bary}(t)$ and $\text{Area}(t)$ denote the triangle barycenter and area, respectively. Then, we select the triangle t_s with the highest score and initialize $\mu_i = \mathcal{E}(t_s)$.

Our strategy balances two aims: (1) selection of triangles within the *interior* of fill regions, where triangles are largest and (2) initialization of seed points that distinguish *new* clusters apart from those already identified by previous selections, akin to k-means++.

4.2.4. Post-processing and boundary extraction

The Voronoi cells in our feature space partition the vertices of M into clusters that are used to determine the flat-fill regions. We first perform a few intuitive post-processing steps to ensure that (1) trivial (fully-closed) fill regions are always detected and (2) each cluster of vertices is connected as a subset of the edge graph of M. Full details on these procedures are described in supp.§6.1. A characteristic example is shown in Fig. 10.

In order to determine the boundaries between flat fill regions, we pull back Voronoi/power cell boundaries onto M via the embedding $\mathcal{E}$, giving us piecewise-linear boundaries. These boundaries are linear over triangles with two different vertex labels, and form a barycentric subdivision of triangles with three different labels. Full details are given in supp. §6.2. We do *not* perform any boundary beautification via curve optimization, sharp corner detection (both done in [PMC22]), or junction type classification [YLL*22]. Our method would be compatible with such post-processing, if desired.

4.3. User editing/sculpting of fill regions

Our framework easily allows three types of natural editing and sculpting operations to be applied to the draft output from previous steps. These are explained below, and showcased in Fig. 5 and in our supplementary video.

4.3.1. Merging clusters

Each cluster seed point is projected from the feature space to its closest vertex on the mesh as an interactive marker. Using these markers, the user may select clusters and merge them at will.

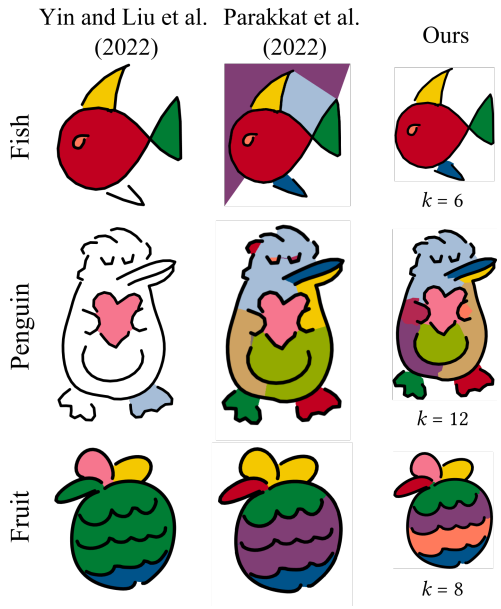


Figure 6: *Automatic coloring comparison. We compare to [YLL*22] and [PMC22] on three representative examples. Our method appropriately handles larger gaps, and benefits from better seed initialization relative to [PMC22]. "Fish" sketch from [Goo17], licensed under CC 4.0 BY.*

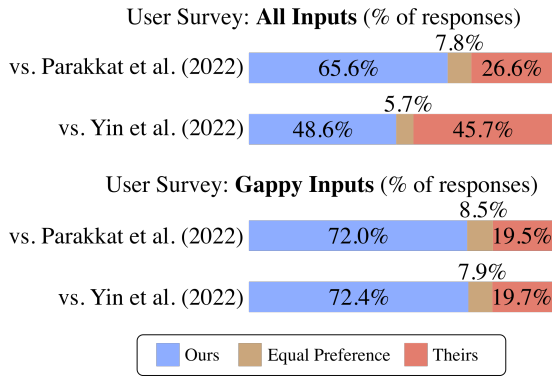


Figure 7: *Informal user survey on automatic method comparisons to [YLL*22] and [PMC22]. 20 participants took part in a 16-question survey that asked them to pick a preference (or neither) among paired, unlabelled results presented to them. All sketches, the survey, and detailed survey results may be found in our supplementary materials archive. The subset of "gappy" sketches was subjectively determined without knowledge of the survey results: Abstract, Fruit, Penguin, Television, Bee, Elephant, Fish, and Frog.*

4.3.2. Splitting/creating new clusters

The user may place additional color hints on the sketch. These hints can correspond to existing color groups, or represent a new group, thereby splitting existing fill regions. Each added hint creates additional seed points as described in §4.2.2. We then recompute the identification of points to clusters *without* running k-means an additional time in order to precisely capture the user's selection.

4.3.3. Cluster refinement via power radius adjustment

For hints that under- or over-fill a desired region, the user may adjust the power radius ρ_i in order to refine the cluster boundary. Note that the user need not specify the power radius of each cluster in granular detail. Instead, we implement this feature as a "slider" for which the increment size is calculated automatically (see supp. §5 for details).

5. Results

Our algorithm was run on a sample of 15 lineart sketches gathered from the Quick, Draw! [Goo17], [EHA12], and [YLL*22] datasets and from the authors. In supp. §3.1 and supp. §12, we include 25 additional results from Quick, Draw!, [EHA12], and Blender Cloud [Ble21]. Please see our dataset for detailed license information.

5.1. Comparison to automatic works

We compare our automatic method to those of [YLL*22] and [PMC22] via an informal user survey (Fig. 7) and some representative examples (Fig. 6). All results are available in supp. §12.2.

To generate the outputs of [YLL*22], we first run sketches through their preprocessing script, which performs smoothing, stroke consolidation, and de-hooking with the default user-specified parameters. We then apply the algorithm of [YLL*22]. The outputs of [PMC22] were graciously generated by the authors upon request. For our method, k was estimated by the user (§4.2.3). Other parameters are set via the formulae described elsewhere in the text: see §4.2.1 for the number of features m and TV weights; see §2 for background thresholding; see §5 for power radius incrementation.

Overall, the survey shows that our method outperforms [PMC22] and is comparable to [YLL*22]. Furthermore, we clearly outperform both methods on "gappy" sketches that have a diverse range of gap sizes. For such sketches, the opposing methods miss large gaps [YLL*22] or over/under-segment due to poor automatic seed placement [PMC22] as illustrated in Fig. 6.

In such sketches, humans tend to close gaps in a perceptual manner without conscious effort [Kan79], rather than relying on a precise geometric criterion like gap width. Based on the survey results, we argue that the winding number better mirrors this intuitive notion of region closure. This contrasts with the gap/junction-based approaches of the competing methods. In [PMC22], edges of the constrained Delaunay triangulation explicitly close gaps via shortest line segments and explicit edge length bottlenecks are used. In [YLL*22], a learning-based junction classifier is trained, and an important *closure factor* parameter (§7 of [YLL*22]) must be set. This parameter is correlated with a maximum allowable gap size and we used the default value in our experiments, not having encountered a strategy for setting it otherwise. Further results on 10 additional inputs from [Goo17] are provided in supp. §12.2.1, highlighting our relative superiority on gappy inputs. Additionally, we include results on 5 additional inputs from [Ble21] that were included in the dataset of [YLL*22] in supp. §12.2.2.

Finally, we highlight that all automatic methods (ours included)

were not entirely robust and prone to undesirable errors. Of the three automatic methods being compared, only ours accommodates a simple transition to manual editing, allowing additional user hints **and** natural shifting of region boundaries via power radii. Moreover, the user may specify k to fit their desired workflow: an artist can encourage under-segmentation in the automatic output and specify new clusters manually, or encourage over-segmentation and perform manual merging. Operations in an example coloring workflow are demonstrated in Fig. 5 and in our supplementary video.

5.1.1. Quantitative Assessment

Quantitative comparisons on colorized vector sketches are challenging due to the need for a "ground truth" segmentation. This is almost never available for *vector* sketches "in the wild", and retroactive segmentation may not preserve the artist's intent. Furthermore, as argued in the sketch vectorization literature [PNCB21, GHJB*23], purely geometric measurements ignore small, but perceptually important differences.

For completeness, we consider the benchmark metrics of [YVG20]: symmetric Hausdorff/Chamfer distances between *raster* curves. See §5.1 of [YVG20] for specific details. To this end, a small quantitative experiment was performed on 10 sketches from [EHA12] with *near*-perfect segmentation. After completing any small gaps manually, the boundaries of the closed sketches were destroyed at random by introducing gaps: see supp. §3 for our procedure. We ran our method and that of [YLL*22] on the destroyed inputs, then rasterized the results for use with the code of [YVG20], comparing to the ground truth in each case.

The table in Fig. 13 shows the benchmark metrics and Fig. 14 shows selected examples (rest in supp. §3.1). Our method performs favorably compared to that of [YLL*22] on a majority of inputs due to the varying gap sizes. [YLL*22] misses some junctions and thus fails to create certain desired regions at all, penalizing their performance to an inordinate degree.

5.2. Comparison to manual works

We selected several state-of-the-art coloring tools based on works described in §2.1 ([SDC09], [FTR18], [PMC22]) and compare to their output (Figs. 8, 9). Our sketches were colored using public implementations of these methods: the "Colorize Mask" tool of Krita 5.2.2 [Kri23] for [SDC09] and the "Colorize Lineart [Smart Coloring]" tool of G'MIC [TF24] for [FTR18]. For [PMC22], the authors colored our sketches independently upon request.

For all manual methods, the colorings can be made near-perfect with enough input, so we have fixed a minimal set of hints for comparison: strokes for [SDC09] and [FTR18] in Fig. 8, and points for [PMC22] in Fig. 9. For our method, we have also included a small number of strength adjustments constituting minimal additional input, easily implemented with a scroll wheel, for example. In the stroke comparisons, we find that our method qualitatively outperforms competing methods, successfully avoiding over-segmentation of the boundary, and better captures a sense of region closure (e.g. Penguin's beak). In the point comparisons, we perform similarly, and note that our method allows for simple adjustment of boundary regions via power radii, for which [PMC22] lacks a direct

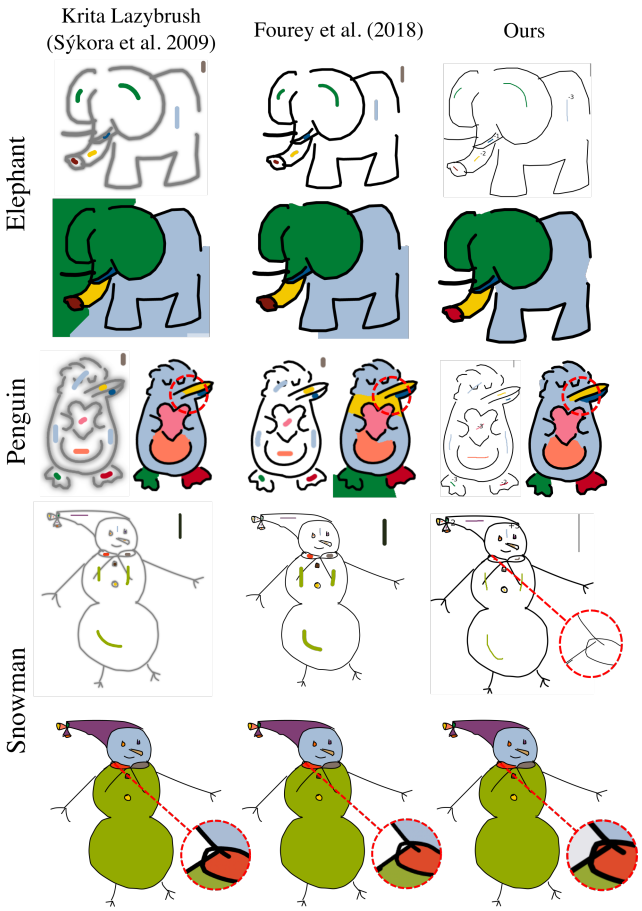


Figure 8: *Comparison to stroke hint methods [SDC09, FTR18]. Fixed "natural" stroke hints were provided, along with a few strength adjustments for our method. We avoid over-segmentation of the boundary, and better capture region closure, as highlighted in the beak of the penguin. We also segment the snowman's scarf correctly due to operating on vector input, which maintains connectivity at any resolution. "Elephant" sketch from [Goo17]; "Snowman" sketch from [EHA12], both licensed under CC 4.0 BY.*

mechanism. In both settings, we benefit from operating strictly on *vector* input in correctly coloring the snowman's scarf. Full results for all 15 sketches are present in supplementary §12.1.

5.3. Proof-of-concept diffuse coloring

To further illustrate the usefulness of the winding number as a confidence measure, we devise a rudimentary scheme for diffuse sketch coloring (Fig. 11). We construct the same feature space $\mathcal{E}$ as above. Then, we compute the color $c(v)$ at each $v \in V$ as follows:

$$\tilde{c}(v) = \sum_{i=0}^{k-1} \frac{C(\mu_i)}{\|\mu_i - \mathcal{E}(v)\|^4} \quad \& \quad c(v) = \frac{\tilde{c}(v)}{\sum_{j=0}^{|V|-1} \tilde{c}(j)} \tag{10}$$

where $C(\mu_i)$ is the color associated with seed point μ_i.

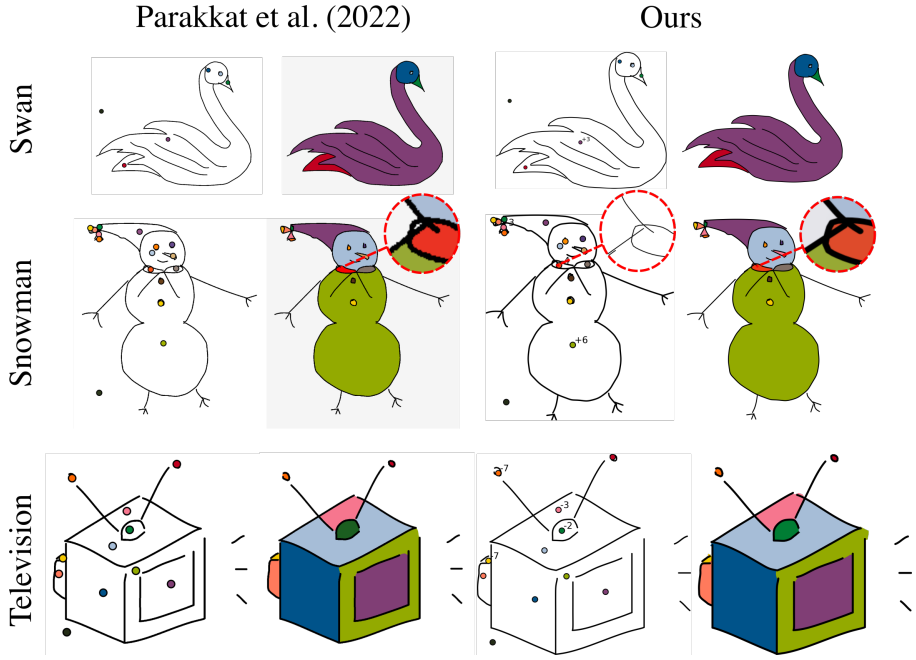


Figure 9: *Comparison to point hint methods [PMC22]. Fixed "natural" point hints were provided, along with a few strength adjustments for our method. We perform similarly, and segment the snowman's scarf correctly due to operating on vector input, which maintains connectivity at any resolution. We also highlight our ability to adjust region boundaries via power radii, which is not available in the competing framework. "Swan" and "Snowman" sketches from [EHA12], licensed under CC 4.0 BY.*

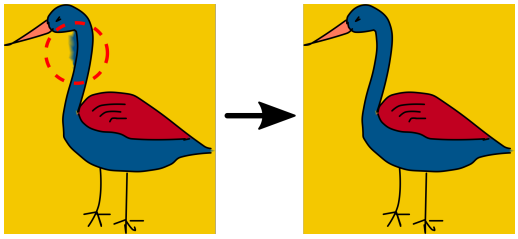


Figure 10: *An example of cluster connectivity postprocessing. A few erroneously classified points appear on the left-hand side of the crane's neck; we trivially detect these points as disconnected from the interior and merge them with their neighboring cluster. "Crane" sketch from [EHA12], licensed under CC 4.0 BY.*

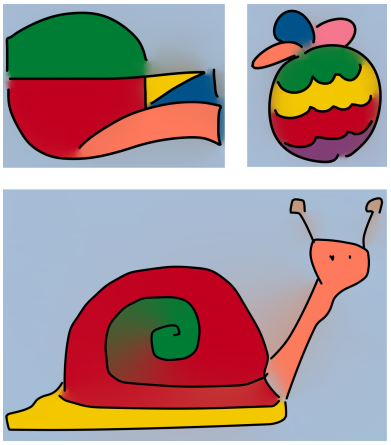


Figure 11: *Sketches colored using our prototype color diffusion method. "Abstract" (top left) was colored using seed points. "Fruit" (top right) and "Snail" (bottom) were colored automatically with $k = 8$ and $k = 5$ respectively. "Snail" sketch from [EHA12], licensed under CC 4.0 BY.*

5.4. Timing of method

The table in Fig. 12 provides a breakdown of the amount of time (in seconds) taken at each stage in our automatic algorithm on several sketches. These experiments were run on a 2021 MacBook Pro (Apple M1 Pro, 16 GB LPDDR5). $k = 20$ is used in clustering, which we consider to be a high degree of segmentation.

We note that the implementation of our mesh cutting (which occurs after Triangle [She96] has produced an initial constrained conforming Delaunay triangulation) is rather inefficient with considerable room for optimization. This is by far the largest contributor to the first measured "Triangulation" stage, with Triangle itself terminating in a matter of milliseconds. Similarly, our implementation of k-means does not make use of parallelization strategies and includes only the most basic distance caching.

Nonetheless, our method is quite performant, especially in terms of the minimal time required to re-run the segmentation once a

Sketch	No. edges	Triangulation	WN	TV	k-means	Cluster Refinement	Boundary Extraction	Total
penguin	181	0.099	0.006	0.044	0.054	0.008	0.016	0.226
noir	850	0.688	0.027	0.275	0.223	0.054	0.046	1.313
cat	1362	1.139	0.113	0.425	0.277	0.082	0.074	2.111
hero_bane	11203	20.092	1.623	3.817	2.405	0.939	3.999	32.876

Figure 12: *Time (in seconds) taken at each stage of our method's pipeline. ("WN" = winding number, "TV" = total variation)*

sketch has been loaded into the program. See "Triangulation," "WN," and "TV" for the upfront, one-time cost of loading a sketch; see "k-means" for the cost of running the automatic segmentation (§4.2.3); see "Cluster Refinement," "Boundary Extraction" for the stages involved in adding/merging/refining clusters. In particular, updates to user-provided hints can be performed in a real-time fashion: see our supplementary video for a demonstration.

5.5. Parameter & other experiments

We perform several parameter and ablation experiments to verify design decisions in our framework. Results of these are presented on three representative sketches each in the supplementary.

- We vary m, showing our choice is conservatively high (supp. §4).
- We vary the set of m randomly sampled winding number features, showing stability of results (supp. §9).
- We consider the feature space without TV weighting, showing degradation of the results (supp. §7).
- We vary k, and show suitable over/under-segmentation that is easily improved with simple editing operations (Fig. 15, supp. §8).
- We vary hint positions, showing stability of results (supp. §10)

Lastly, in supplementary §11, we consider an application of basic spectral clustering to the problem at hand, and see that it suffers from a bias toward balanced clusters and poor seed initialization.

6. Conclusion

We have proposed a winding-number-based method for multi-region segmentation of lineart vector sketches that outperforms existing methods on inputs with a diverse range of gap sizes. Such input is typical of less professional sketches and doodles, broadening the availability of flat-fill tools to a wider range of users. This challenging problem is solved by considering a feature space of winding numbers weighted by relative total variation. As proof of our method's utility, we design two intertwined paradigms by which users may color sketches: automatic rough coloring and interactive region sculpting, and show that we perform comparably to state-of-the-art methods. Lastly, our perspective inherits the notion of region confidence imparted by winding numbers; we demonstrate its use in designing sculpting tools and a proof-of-concept diffuse coloring scheme.

6.1. Limitations & future work

There are several aspects of our framework that invite further improvement or investigation. First, we did not experiment extensively with some of the precise formulae in our pipeline, e.g., feature weights (Eq. (9)) and feature value at stroke endpoints (Eq. (4.1)). Tuning these may boost performance further. Second, we anticipate that applying existing (or developing novel) aesthetic gap closure methods would also significantly boost the quality of our output. Third, the postprocessing of §4.2.4 to ensure connected components may complicate the boundary refinement of §4.3.3; strategies to avoid it could be explored. Fourth, the current implementation of our method does not consider how to "split" strokes, and may fail when a single long stroke is not split along sharp corners or self-intersections (Fig. 16).

Lastly, we highlight three future directions that arise from the simplicity and intuitiveness of our approach. First, it is certainly possible to combine our space of winding number features with other methods that are more focused on gap/junction detection and classification. We feel that the ability of winding numbers to more intuitively capture region closure could have a symbiotic relationship with such methods. Second, multi-region segmentation with a winding number feature space generalizes naturally to a 3D setting, where strokes are replaced by triangle meshes/soups, parametric surfaces, or oriented point clouds. Third, we believe that we have not fully leveraged the inherited notion of region *confidence*. We look forward to improving and extending our power-based tool for boundary refinement and proof-of-concept diffuse coloring, and to finding additional applications for our theoretical framework.

	Hausdorff		Chamfer	
	YLL (2022)	Ours	YLL (2022)	Ours
bell	0.0182	0.0255	0.0022	0.0011
bottle	0.8302	0.0665	0.1928	0.0064
butterfly	0.2463	0.1314	0.0280	0.0032
candle	0.0077	0.0090	0.0007	0.0018
dog	0.0921	0.1632	0.0049	0.0078
pizza	0.2221	0.0679	0.0147	0.0040
seagull	-	0.0467	-	0.0049
teddy	0.2444	0.2153	0.0181	0.0193
tree	0.5140	0.0463	0.0946	0.0036
umbrella	0.4942	0.0496	0.0774	0.0035
Average	0.2966	0.0821	0.0482	0.0056
Std. dev.	0.2703	0.0661	0.0638	0.0052

Figure 13: *Results of the quantitative study (§5.1.1). A set of 10 well-segmented images from [EHA12] serve as our ground truth. Then, we randomly destroy strokes by introducing gaps, and run our method and that of [YLL*22] on the destroyed inputs. We compare the output of each method to the ground truth segmentation using the metrics of [YVG20], ranging from 0 to $\sqrt{2}$ (with 0 being a perfect match between region boundaries). [YLL*22] does not generate fill regions for the "seagull" input.*

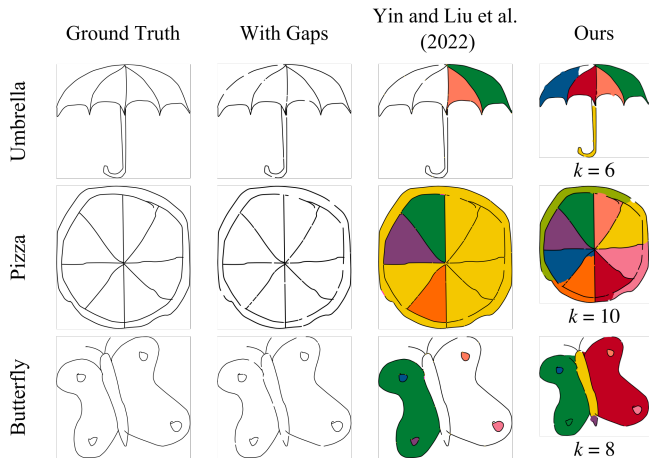


Figure 14: *Comparison to [YLL*22] on selected inputs from the quantitative study (§5.1.1). Results from our method were generated by the automatic variant. k was specified by the user for each sketch. "Umbrella," "Pizza," "Butterfly" sketches from [EHA12], licensed under CC 4.0 BY.*

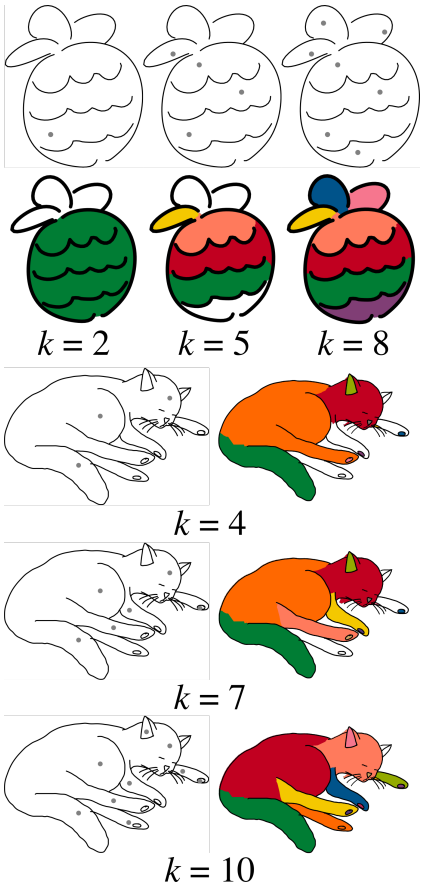


Figure 15: *Results of varying k in our automatic method for two sketches. We show the seed points alongside each output. "Cat" sketch by Dave Pagurek [YLL*22], licensed under CC 4.0 BY.*

Acknowledgements

We thank Amal Dev Parakkat for kindly running the method of [PMC22] on our set of inputs and providing the results shown in this work. Additionally, we thank Andrew Wood for lending the authors equipment in order to run experiments.

References

[AFP00] AMBROSIO L., FUSCO N., PALLARA D.: *Functions of bounded variation and free discontinuity problems*. Oxford Mathematical Monographs, 2000. 3

[Aur87] AURENHAMMER F.: Power diagrams: Properties, algorithms and applications. *SIAM Journal on Computing 16*, 1 (1987), 78–96. 3

[AV07] ARTHUR D., VASSILVITSKII S.: k-means++: the advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (USA, 2007), SODA '07, Society for Industrial and Applied Mathematics, p. 1027–1035. 6

[BDS*18] BARILL G., DICKSON N., SCHMIDT R., LEVIN D. I., JACOBSON A.: Fast winding numbers for soups and clouds. *ACM Transactions on Graphics* (2018). 2, 3, 4, 5

[Ble21] BLENDER: Blender cloud. https://cloud.blender.org/p/gallery/5b642e25bf419c1042056fc6, 2021. 7

[BS18] BESSMELTSEV M., SOLOMON J.: Vectorization of line drawings via polyvector fields, 2018. 3

[Che87] CHEW L. P.: Constrained delaunay triangulations. In *Proceedings of the Third Annual Symposium on Computational Geometry* (New York, NY, USA, 1987), SCG '87, Association for Computing Machinery, p. 215–222. 5

[EHA12] EITZ M., HAYS J., ALEXA M.: How do humans sketch objects? *ACM Trans. Graph. (Proc. SIGGRAPH) 31*, 4 (2012), 44:1–44:10. 7, 8, 9, 10, 11

[FGC23] FENG N., GILLESPIE M., CRANE K.: Winding numbers on discrete surfaces. *ACM Trans. Graph. 42*, 4 (jul 2023). 4, 5

[FLB16] FAVREAU J.-D., LAFARGE F., BOUSSEAU A.: Fidelity vs. simplicity: A global approach to line drawing vectorization. *ACM Trans. Graph. 35*, 4 (jul 2016). 3

[FTR18] FOUREY S., TSCHUMPERLÉ D., REVOY D.: A Fast and Efficient Semi-guided Algorithm for Flat Coloring Line-arts. In *Vision, Modeling and Visualization* (2018), Beck F., Dachsbacher C., Sadlo F., (Eds.), The Eurographics Association. 2, 3, 8

[GHJB*23] GUȚAN O., HEGDE S., JIMENEZ BERUMEN E., BESSMELTSEV M., CHIEN E.: Singularity-free frame fields for line drawing vectorization. *Computer Graphics Forum 42*, 5 (2023), e14901. 3, 8

[Goo17] GOOGLE: Quick, draw! the data. https://quickdraw.withgoogle.com/data, 2017. 7, 8

[HA01] HORMANN K., AGATHOS A.: The point in polygon problem for arbitrary polygons. *Computational Geometry 20*, 3 (2001), 131–144. 2

[JKS13] JACOBSON A., KAVAN L., SORKINE O.: Robust inside-outside segmentation using generalized winding numbers. *ACM Trans. Graph. 32*, 4 (2013). 2, 3, 4, 5

[Kan79] KANIZSA G.: *Organization in Vision: Essays on Gestalt Perception*. Bloomsbury Academic, 1979. 7

[KBH06] KAZHDAN M., BOLITHO M., HOPPE H.: Poisson surface reconstruction. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing* (Goslar, DEU, 2006), SGP '06, Eurographics Association, p. 61–70. 3

[Kof35] KOFFKA K.: *Principles of Gestalt Psychology*. Harcourt, Brace, Oxford, England, 1935. 2

[Kri23] KRITA: Krita, 2023. URL: https://krita.org/en/. 8

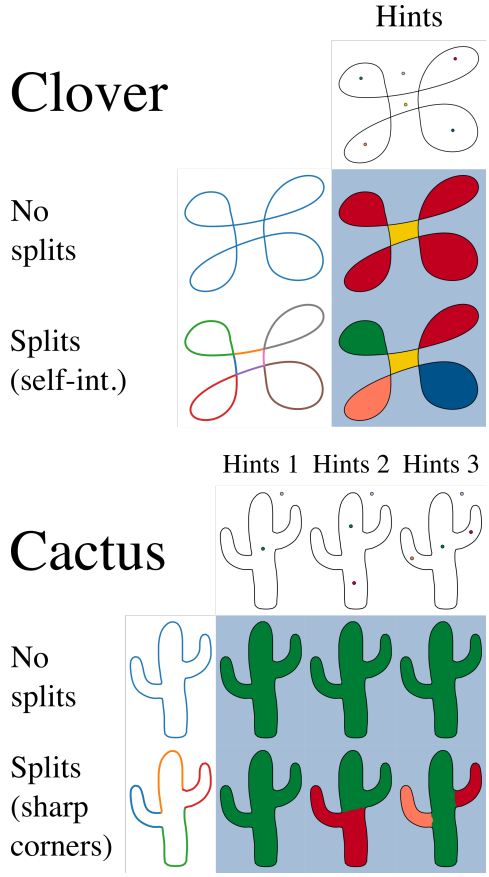


Figure 16: *A single stroke that captures multiple intended regions must be split in order to evince segmentation cues in the winding number. To illustrate this point, we construct single-stroke examples and show the output of our method* ***without*** *the connectivity-based postprocessing described in §4.2.4. (Top, clover) The single-stroke winding number is identical between each "leaf" of the clover due to the orientation of the curve. (Bottom, cactus) The winding number is constant in closed regions and does not differ between "arms" of the cactus. Only by splitting the stroke at sharp corners do we recover these regions, if desired by the user.*

[Llo82] LLOYD S.: Least squares quantization in pcm. *IEEE Transactions on Information Theory 28*, 2 (1982), 129–137. 6

[LLW04] LEVIN A., LISCHINSKI D., WEISS Y.: Colorization using optimization. *ACM Trans. Graph. 23*, 3 (aug 2004), 689–694. 2

[MHZ*21] METZER G., HANOCKA R., ZORIN D., GIRYES R., PANOZZO D., COHEN-OR D.: Orienting point clouds with dipole propagation. *ACM Trans. Graph. 40*, 4 (jul 2021). 2, 3

[NHS*13] NORIS G., HORNUNG A., SUMNER R. W., SIMMONS M., GROSS M.: Topology-driven vectorization of clean line drawings. *ACM Trans. Graph. 32*, 1 (feb 2013). 3

[PBM18] PARAKKAT A. D., BONDI PUNDARIKAKSHA U., MUTHUGANAPATHY R.: A delaunay triangulation based approach for cleaning rough sketches. *Computers & Graphics 74* (2018), 171–181. 3

[PCS21] PARAKKAT A. D., CANI M.-P. R., SINGH K.: Color by numbers: Interactive structuring and vectorization of sketch imagery. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2021), CHI '21, Association for Computing Machinery. 2, 3, 6

[PMC22] PARAKKAT A. D., MEMARI P., CANI M.-P.: Delaunay painting: Perceptual image colouring from raster contours with gaps. *Computer Graphics Forum 41*, 6 (2022), 166–181. 2, 3, 6, 7, 8, 9, 11

[PNCB21] PUHACHOV I., NEVEU W., CHIEN E., BESSMELTSEV M.: Keypoint-driven line drawing vectorization via polyvector flow. *ACM Trans. Graph. 40*, 6 (dec 2021). 3, 8

[QWH06] QU Y., WONG T.-T., HENG P.-A.: Manga colorization. *ACM Transactions on Graphics (SIGGRAPH 2006 issue) 25*, 3 (July 2006), 1214–1220. 2

[ROF92] RUDIN L. I., OSHER S., FATEMI E.: Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena 60*, 1 (1992), 259–268. 4

[SBv05] SÝKORA D., BURIÁNEK J., ŽÁRA J.: Colorization of black-and-white cartoons. *Image and Vision Computing 23*, 9 (2005), 767–782. 2

[SDC09] SÝKORA D., DINGLIANA J., COLLINS S.: LazyBrush: Flexible painting tool for hand-drawn cartoons. *Computer Graphics Forum 28*, 2 (2009), 599–608. 2, 8

[She96] SHEWCHUK J. R.: Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, Lin M. C., Manocha D., (Eds.), vol. 1148 of *Lecture Notes in Computer Science*. Springer-Verlag, May 1996, pp. 203–222. From the First ACM Workshop on Applied Computational Geometry. 4, 9

[She97] SHEWCHUK J. R.: *Delaunay refinment mesh generation*. PhD thesis, Carnegie Mellon University, 1997. 5

[She02] SHEWCHUK J. R.: Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry 22*, 1 (2002), 21–74. 16th ACM Symposium on Computational Geometry. 5

[SSII18] SIMO-SERRA E., IIZUKA S., ISHIKAWA H.: Mastering sketching: Adversarial augmentation for structured prediction. *ACM Trans. Graph. 37*, 1 (jan 2018). 3

[TF24] TSCHUMPERLÉ D., FOUREY S.: G'mic (greyc's magic for image computing): A full-featured open-source framework for image processing, 2024. URL: https://gmic.eu/. 8

[TJKSH14] TAKAYAMA K., JACOBSON A., KAVAN L., SORKINE-HORNUNG O.: Consistently orienting facets in polygon meshes by minimizing the dirichlet energy of generalized winding numbers, 2014. 2, 3

[XDW*23] XU R., DOU Z., WANG N., XIN S., CHEN S., JIANG M., GUO X., WANG W., TU C.: Globally consistent normal orientation for point clouds by regularizing the winding-number field. *ACM Trans. Graph. 42*, 4 (jul 2023). 2, 3, 4

[YLL*22] YIN J., LIU C., LIN R., VINING N., RHODIN H., SHEFFER A.: Detecting viewer-perceived intended vector sketch connectivity. *ACM Transactions on Graphics 41* (2022). 1, 2, 3, 6, 7, 8, 10, 11

[YVG20] YAN C., VANDERHAEGHE D., GINGOLD Y.: A benchmark for rough sketch cleanup. *ACM Transactions on Graphics (TOG) 39*, 6 (Nov. 2020). 3, 8, 10

[ZCZ*09] ZHANG S.-H., CHEN T., ZHANG Y.-F., HU S.-M., MARTIN R. R.: Vectorizing cartoon animations. *IEEE Transactions on Visualization and Computer Graphics 15*, 4 (2009), 618–629. 2, 3

[ZLSS*21] ZHANG L., LI C., SIMO-SERRA E., JI Y., WONG T.-T., LIU C.: User-guided line art flat filling with split filling mechanism. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021). 2, 3

[ZLW*18] ZHANG L., LI C., WONG T.-T., JI Y., LIU C.: Two-stage sketch colorization. *ACM Transactions on Graphics (SIGGRAPH Asia 2018 issue) 37*, 6 (November 2018), 261:1–261:14. 2, 3