Differential Operators on Sketches via Alpha Contours

MARIIA MYRONOVA, WILLIAM NEVEU, and MIKHAIL BESSMELTSEV, Université de Montréal, Canada

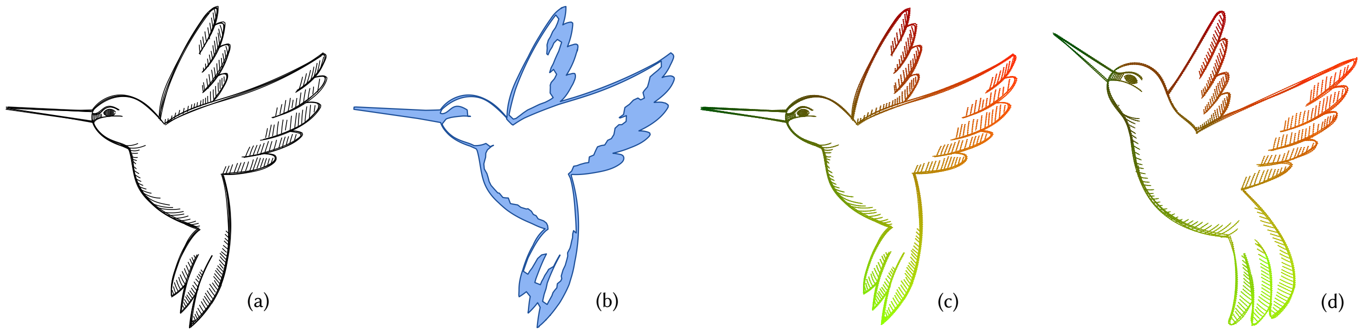

Fig. 1. For a vector sketch (a), our Alpha Contours method produces a 2-manifold shape tightly containing the input strokes, approximating the *positive space* of the sketch (b). The resulting shapes allow us to discretize differential operators, including Laplacian and Steklov operators, enabling standard geometry processing pipelines on vector sketches, such as functional maps (c,d).

A vector sketch is a popular and natural geometry representation depicting a 2D shape. When viewed from afar, the disconnected vector strokes of a sketch and the empty space around them visually merge into *positive space* and *negative space*, respectively. Positive and negative spaces are the key elements in the composition of a sketch and define what we perceive as the shape. Nevertheless, the notion of positive or negative space is mathematically ambiguous: While the strokes unambiguously indicate the interior or boundary of a 2D shape, the empty space may or may not belong to the shape's exterior.

For standard discrete geometry representations, such as meshes or point clouds, some of the most robust pipelines rely on discretizations of differential operators, such as Laplace-Beltrami. Such discretizations are not available for vector sketches; defining them may enable numerous applications of classical methods on vector sketches. However, to do so, one needs to define the positive space of a vector sketch, or the *sketch shape*.

Even though extracting this 2D sketch shape is mathematically ambiguous, we propose a robust algorithm, *Alpha Contours*, constructing its conservative estimate: a 2D shape containing all the input strokes, which lie in its interior or on its boundary, and aligning tightly to a sketch. This allows us to define popular differential operators on vector sketches, such as Laplacian and Steklov operators.

We demonstrate that our construction enables robust tools for vector sketches, such as As-Rigid-As-Possible sketch deformation and functional maps between sketches, as well as solving partial differential equations on a vector sketch.

CCS Concepts: • **Computing methodologies** → **Parametric curve and surface models**; **Shape analysis**.

Authors' address: Mariia Myronova, maria.myronova@umontreal.ca; William Neveu, william.neveu@umontreal.ca; Mikhail Bessmeltsev, Université de Montréal, Canada, bmpix@iro.umontreal.ca.

0730-0301/2023/8-ART1 $15.00
https://doi.org/10.1145/3592420

Additional Key Words and Phrases: vector graphics, sketch processing, differential operators

ACM Reference Format:
Mariia Myronova, William Neveu, and Mikhail Bessmeltsev. 2023. Differential Operators on Sketches via Alpha Contours. *ACM Trans. Graph.* 42, 4, Article 1 (August 2023), 15 pages. https://doi.org/10.1145/3592420

1 INTRODUCTION

Sketching is one of the core instruments of visual expression available to an artist. With the advent of tablets and the increasing accuracy of vectorization methods [Puhachov et al. 2021], many sketches are created or converted into vector format, where they are composed of many disconnected vector strokes. When looked at from afar, the separate strokes visually merge, revealing the depicted 2D geometry and the empty space around it, sometimes known as figure-ground, or *positive* and *negative* space, respectively, in art and perception literature (Fig. 2a) [Wagemans et al. 2012].

The positive and negative space, or dark and light respectively, are the primary elements in the composition of a drawing [Itten 1976]. The drawn strokes together form 'dark', or positive space, and the empty space between them forms 'light', or negative space (Fig. 1b, positive space in blue). Some strokes outline the boundary of that positive space shape, some strokes depict thick curves, and some hatching strokes depict filled or textured regions, forming the interior of the positive space [Dodson 1990; Philbrick and Kaplan 2022]. Note that by this definition, white regions are negative space, regardless of whether they are enclosed in an object (e.g., in Fig. 1b, the white interior of the hummingbird is a negative space).

From a geometric standpoint, sketch strokes are a depiction or a *representation* of 2D geometry of the positive space. Sketch strokes explicitly indicate the interior or boundary of the positive space, while the empty space between the strokes ambiguously suggests the shape's exterior or negative space (Fig. 2b). However, this positive space or *sketch shape* is unknown, and the task of extracting it from a sketch is mathematically ill-posed: it is unclear whether a gap between strokes is intended to belong to the positive or negative space.

For standard discrete geometry representations, such as meshes or point clouds, extensive research has resulted in mature and robust processing pipelines. Some of those most robust algorithms are based on the discretization of differential operators, such as Laplacian [Pinkall and Polthier 1993], Dirac [Liu et al. 2017], and Steklov operators [Wang et al. 2018]. These operators enable numerous exciting applications such as shape correspondence [Ovsjanikov et al. 2012], deformation [Sorkine and Alexa 2007], skinning weights computations [Jacobson et al. 2011], and many others. Such operators, however, are not defined on vector sketches, rendering all these methods largely inapplicable to this popular geometry representation. Defining such operators in a robust way may serve as a core component in future sketch processing methods.

The key issue in defining such a differential operator is that it requires outlining the mathematically ambiguous sketch shape, i.e., separating the interior of a shape from its exterior (Fig. 2). If the 2D shape were known, it would allow distinguishing points that are *intrinsically* close from those that are only *extrinsically* close, a necessary distinction for a shape-aware differential operator.

While extracting the 2D shape of a vector sketch is an ill-posed task, we look for a *conservative* estimate: a 2D shape *tightly containing* all the drawing strokes of the input sketch that excludes only the non-ambiguous empty space. As a conservative estimate, such a shape contains textural elements and small details, where the distinction between inside and outside may be ambiguous but still excludes clear, large enough gaps (Fig. 2c). Due to the *containment*, a 2D shape allows for a definition of a differential operator (e.g., a Laplacian) of the sketch as a restriction of the differential operator of the corresponding shape's manifold onto the sketch strokes. In particular, a function over sketch strokes may be computed as a restriction of a function over the full sketch shape. The *tightness* requirement makes the extracted 2D shape close to the one depicted by a sketch, such that its complement accurately captures the negative space.

Finding such a 2D shape of a sketch, however, is a challenging task. Sketches are, by their very nature, often created quickly and imprecisely. They may contain overlapping, intersecting strokes with unclear connectivity and gaps (Fig. 2b). Most importantly, the task of distinguishing intrinsic proximity from the extrinsic one is ill-posed: Even for a simpler subclass of vector drawings consisting solely of roughly parallel strokes *clusters* (e.g., petals in Fig. 2b), human observers sometimes disagree on whether some strokes are intrinsically close or not [Liu et al. 2018]. This task becomes even more challenging for general sketches (Fig. 1), since those drawings, in addition to clusters, may contain textural elements or small features are often difficult to parse even for human observers (Fig. 12), impeding telling inside from outside.

However, we can observe that the connectivity of strokes is a strong indicator of intrinsic closeness. Precisely, we note that the boundary of the sought shape should mostly follow some of these strokes due to the tightness requirement (Fig. 2). Note that this is true even for high curvature points: Despite different stroke points being close, the shape's boundary follows the stroke without cutting corners. The only locations where the boundary would deviate from the drawn strokes are near stroke endpoints, where otherwise, the boundary would separate intrinsically close strokes (Figs. 2, 6).

We leverage these observations and, inspired by the classical algorithm of Alpha Shapes [Edelsbrunner et al. 1983], we propose a novel robust algorithm to delineate a 2D shape of a vector sketch — Alpha Contours. Controlled by a single parameter α, our algorithm robustly finds conservative estimates of 2D shapes of vector sketches of any kind, enabling numerous novel applications of 2D vector sketches, such as As-Rigid-As-Possible deformation [Sorkine and Alexa 2007], correspondences between sketches via functional maps [Ovsjanikov et al. 2012], computing eigenfunctions of popular differential operators, or solving other partial differential equations on sketches, such as heat equation. We validate our algorithm on a gallery of inputs by demonstrating those applications on various sketches (Sec. 7) and comparing it with the previous work, both qualitatively and quantitatively (Sec. 6). We discuss the choice of the parameter α in Sec. 4; we suggest a simple heuristic to automatically compute α, which we use for all the results in the paper unless otherwise specified.

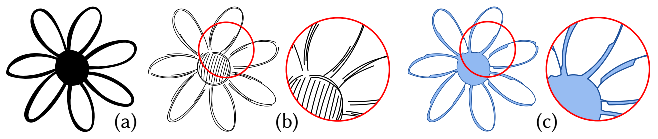

Fig. 2. Positive (black) and negative (white) space are key to a composition of a drawing (a). In sketching, artists often depict positive space via densely sampled strokes (b), where the strokes clearly belong to the positive space, but the empty space between them only ambiguously suggests the negative space. Given an input vector sketch (b), we produce the *sketch shape* (c), which captures the ambiguous concept of positive space, by filling in the gaps and aligning the boundaries of the shape to the input strokes.

2 RELATED WORK

Our work builds upon progress in three areas: differential operators on discrete geometry, envelopes and cages, and vector sketch processing. We only focus on the most relevant works.

2.1 Operators on Discrete Geometry

Geometry processing has a long history of discretizing differential operators on various geometry representations. One of the main operators of interest is the Laplace-Beltrami operator, often discretized on a manifold mesh as the cotangent Laplacian [Pinkall and Polthier 1993]. On point clouds, classical approaches define a Laplacian via a discrete heat kernel [Belkin et al. 2009], or via Gaussian kernels [Liu et al. 2012]. Another practical approach is to use mesh Laplacians on locally meshed neighborhoods [Cao et al. 2010; Clarenz et al. 2004]. Over nonmanifold meshes, including meshed point clouds, the operator can be discretized via a 'tufted cover' [Sharp and Crane 2020]. On codimensional simplices, one can start by discretizing Dirichlet energy, leading to a discretization of a Laplacian [Zhu et al. 2014].

Our Alpha Contours algorithm is designed to produce a 2-manifold 2D shape that we then triangulate and use the standard cotangent Laplacian. We compare our construction with the state-of-the-art

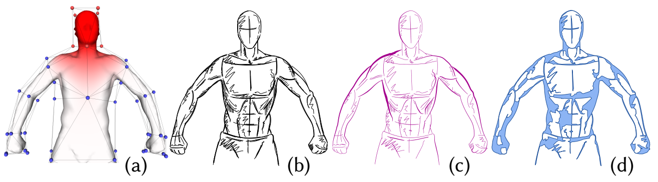

Fig. 3. Automatic cage construction methods find a low-complexity polygon containing the object (a, image from [Casti et al. 2019]) and do not attempt tightly contain it. Having a generic vector sketch (b), stroke cluster parameterization methods [Van Mossel et al. 2021] in conjunction with sketch clustering methods [Liu et al. 2018] can only approximate the shape of certain parts of the drawing (c, thick curves on the arms) and fail on the others (c, rudimentary shading). Our automatic result (d).

point cloud Laplacian discretization [Sharp and Crane 2020] in Sec. 6. As shown there, our construction leverages the connectivity of the input strokes and thus can capture the sketch shape more precisely and, therefore, the shape Laplacian.

In addition to these various discretizations of the Laplace-Beltrami operator, previous work has considered extrinsic differential operators on discrete geometry, such as Dirac [Liu et al. 2017] and Steklov [Wang et al. 2018] operators. We leverage a similar construction to the one of Wang et al. [2018] to demonstrate that Alpha Contours can be used directly to construct the Steklov operator for sketches via the boundary element method (Sec. 5.2).

2.2 Envelopes, Bounding Hulls, and Cages

Envelopes, or Bounding Hulls. Our work is heavily rooted in computational geometry and topology. Among those, we are inspired by generalizations of Convex Hulls, in particular, a classical approach to reconstructing 2D shapes containing a given set of points — Alpha Shapes [Edelsbrunner 1992; Edelsbrunner et al. 1983]. For a given α, Alpha Shape's boundary is composed of segments between points such that there is a circle of radius α through those points that contains no other points from the set. These segments form closed contours, separating the inside of the alpha shape from the outside. Note that alpha shapes are not necessarily convex; convex hull corresponds to $\alpha = \infty$. Other generalizations of a convex hull are concave hulls [Moreira and Santos 2007], χ-hulls [Duckham et al. 2008], α-concave hulls [Asaeedi et al. 2017], or Crust algorithm [Amenta et al. 1998], to name a few.

In order to apply these point cloud approaches to vector sketches, one needs to sample the sketches and ignore all the connectivity. Depending on the parameters, loss of connectivity leads to spurious connections between disconnected strokes or carving out too much of the sketch shape – both unacceptable scenarios (Fig. 4). Changing the sampling does not solve that problem (Fig. 4b,e). In contrast, our method leverages the connectivity of a sketch to robustly separate regions that are intrinsically far, even for a sparse sampling of strokes; this also makes our algorithm robust to the choice of radius α (Fig. 18). We compare our method to Alpha Shapes in detail, as the most popular alternative (Sec. 6, Fig. 14).

Cages. A related area of research is automatic cage generation for shape deformation [Ben-Chen et al. 2009; Casti et al. 2019; Le and Deng 2017]; for a recent review, please refer to Casti et al. [2019] or Nieto and Susin [2013]. In 2D context, given an input mesh, the goal of these methods is to compute a polyline of low complexity, or a *cage*, that contains the input mesh, to serve as the animation proxy (Fig. 3a) [Sederberg and Parry 1986]. Our goals, however, are different: We are aiming to construct a *tightly* aligning shape to the input vector strokes of any complexity (Fig. 3b,d). Automatically constructed cages often poorly align to the input mesh and contain too much extra negative space, affecting the precision of the differential operators.

2.3 Vector Sketch Processing

In recent years, our community has seen progress in a variety of vector sketch processing tasks: sketch colorization [Adobe 2020; Orzan et al. 2008], shading [Finch et al. 2011; Shao et al. 2012], style transfer [Freeman et al. 2003], or 3D reconstruction [Bessmeltsev et al. 2015; Gryaditskaya et al. 2020; Xu et al. 2014]. These methods often assume a clean vector drawing with precise connectivity and junctions. However, artists' vector sketches are typically noisy, have overdrawn strokes, and contain numerous inaccuracies, making these methods inapplicable directly. Imprecise junctions and connectivity can be disambiguated by the recent method of Yin et al. [2022], but its application is limited to clean drawings.

A few methods address the problem of consolidating sketches into clean ones [Barla et al. 2005; Grabli et al. 2004; Liu et al. 2018; Rosin 1994]. These methods typically target a subclass of vector drawings where strokes can be meaningfully grouped into long thin clusters of roughly parallel strokes (e.g., Fig. 15). Such drawings typically contain little to no shading or texture, as compared to general sketches (Figs. 1, 19). Consolidation methods are complementary to ours: Our algorithm accepts any input vector sketch, but for a drawing composed out of clusters, we can leverage the result of consolidation algorithms and produce an even tighter 2D shape, as we show in Figure 15 (Sec. 7).

A related line of work considered gap closing for vector drawings, from the sketching systems [Adobe 2020; Asente et al. 2007; Gangnet et al. 1994], methods detecting intended junctions for polyhedra [Company et al. 2019; Shuxia and Suihuai 2009; Wang et al. 2020], to recent learning-based approaches [Yin et al. 2022]. These methods typically focus on finding closed regions to fill, and do not attempt to find the sketch shape for a differential operator computation.

With a somewhat similar goal, a line of methods simultaneously vectorize and consolidate raster line drawings [Bartolo et al. 2007; Chen et al. 2018; Egiazarian et al. 2020; Favreau et al. 2016; Kim et al. 2018; Mo et al. 2021; Parakkat et al. 2021, 2018; Stanko et al. 2020; Zhang et al. 2009]. These methods work only with raster input and do not directly extract the sketch shape, instead producing only stroke centerlines.

For the vector drawings made out of clusters, a recent sketch parameterization method [Van Mossel et al. 2021] can approximate the shape of these clusters with thin strips (Fig. 3b,c). For the stroke clusters, the union of such strips forms a 2D shape similar to the one we target with our algorithm. However, their algorithm does not find the shape of fills or textural elements, nor their result contains

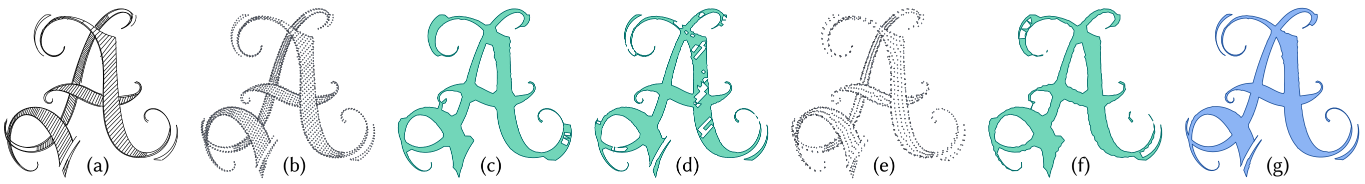

Fig. 4. Alpha Shapes, like other point cloud–based approaches, require converting the input vector sketch (a) into a point cloud (b,e), discarding all the stroke connectivity. This loss of connectivity often results in either spurious connections (c) or holes (d) in the reconstruction, depending on the chosen alpha value (for (c,d,g) we use the same α for a fair comparison, for (d) we took 60% of that value). These issues cannot be easily resolved by resampling the point cloud (e,f). Our approach leverages the connectivity of the input strokes, alleviating these issues.

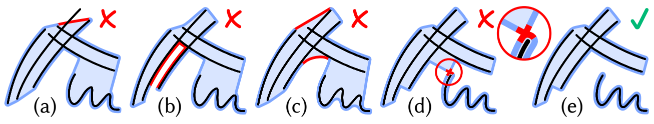

Fig. 5. Requirements for the sketch shape: (a) The shape (blue) should contain the input strokes (black). (b) The shape should not separate strokes that are clearly intrinsically close. (c) The shape should tightly bound the shape, carefully cutting out the negative space. (d) The shape should be manifold. (e) Our sketch shape.

the input strokes, and therefore, cannot be used directly as the geometric domain (Figs. 3c, 16).

3 REQUIREMENTS AND OBSERVATIONS

Given a vector sketch, i.e., a set of vector strokes, our goal is to estimate the depicted positive space, the sketch shape. We first clarify the requirements on the shape, then outline the main observations leading to our method.

3.1 Requirements

Following the principle that the shape must be a *conservative estimate* that *tightly contains* the input strokes (Sec. 1), we now clarify these in the following requirements, guiding all our algorithmic choices:

(1) *Containment.* All the input strokes must be inside or on the boundary of the 2D shape we reconstruct. This guarantees that we can define our differential operators as a restriction of the corresponding differential operator of the 2D shape onto the sketch strokes. As a consequence, the boundary of the shape does not intersect any of the input strokes (Fig. 5a).
(2) *Conservative.* Each input stroke indicates points that unambiguously belong to the shape; consecutive samples along a stroke are, without a doubt, intrinsically close. The gaps between the strokes, however, might or might not belong to the shape (Fig. 5b). Since we are targeting a *conservative* estimate of a 2D shape, its boundary must not separate the strokes that are unambiguously intrinsically close, such as strokes of a stroke cluster, i.e., a group of parallel strokes forming a long narrow region (e.g., each petal in Fig. 2). In other words, we only should exclude the regions from our estimate that are unambiguously outside the unknown 2D shape.
(3) *Tightness.* Since our goal is to envelop the sketch geometry tightly, we require our algorithm, subject to the previous requirements, to produce a shape estimate that leaves as little empty space around the strokes as possible. In other words, the areas that are unambiguously outside the 2D shape must be cut out of it. For some applications of the differential operators we discretize, *tightness* is crucial: For instance, a functional map between two shapes will only provide a meaningful pointwise correspondence if a 2D shape tightly aligns with the input strokes (Fig. 5c, detailed discussion in Sec. 7).
(4) *Manifold with boundary.* We require the 2D shape to be a 2-manifold, i.e., to contain no codimension–0 or -1 features, such as curves or points (Fig. 5d). For convenience, we also assume it contains its boundary. This means, in particular, that the shape's boundary must consist of simple closed curves that never intersect each other of themselves.

3.2 Observations

As we are targeting a 2D shape tightly containing the input strokes, the boundary of the sought 2D shape is necessarily supported by some of the input strokes, only filling in the necessary gaps (Fig. 6b, the solid blue segments). Therefore, in order to find such a shape, we need to perform two tasks: First, complete the boundaries by filling in the gaps between points that are intrinsically close (Fig. 6b, the dashed segments). The input sketch, together with those extra connections, divides the plane into *faces* (Fig. 8c). *Here faces are 2-dimensional cells in the arrangement of curves.* Second, we need to decide which faces of this structure are inside the shape (Fig. 6, the shaded blue shape). In particular, this decision defines for each point of an input stroke whether it lies on the boundary or inside the 2D shape. The union of the faces marked as 'inside' will then form our 2D shape.

Both of these tasks require disambiguation of whether two points are 'intrinsically close' or not. Informally, this defines whether the space between the two points belongs to the shape: For instance, we consider two nearby parallel strokes of a stroke cluster to be intrinsically close, so the space between them belongs to the positive space. Determining this, however, is mathematically ill-posed. To disambiguate this, we leverage the following analogy.

Artistic literature and previous research [Dodson 1990; Van Mossel et al. 2021] suggest that artists draw strokes within a shape they envision, sometimes creating the strokes that outline the boundary of the shape, sometimes drawing textural strokes or adding details within it — a process reminiscent of sampling points on a surface (cf. Fig. 2a,b).

Inspired by this analogy, we can cast the problem of finding the sketch shape as a problem of reconstruction: given the input strokes, reconstruct the full 2D shape they were sampled from. Our approach is to formalize the vague yet intuitive notion of 'intrinsic closeness' by leveraging the connectivity of the sketch and a single parameter $\alpha > 0$. Here α is inversely proportional to the sampling density and defines the upper bound of a gap between two 'intrinsically close' strokes. This implies artists draw the boundary of the sketch shape leaving gaps up to α distance (Fig. 6b), and that any region containing an empty α-ball is assumed to be outside the shape. We refer to this observation as **the sampling assumption**.

If two points on different strokes are within that threshold distance, however, this does not necessarily mean they are intrinsically close, or, equivalently, that the space between them is positive. One example is illustrated in Fig. 4, where the little sliver at the top-left is not intrinsically close to the 'body' of the letter. To determine which points are intrinsically close, we need the following definition:

Definition 3.1. Given a closed set $C \subset \mathbb{R}^2$ and a radius $\alpha > 0$, two points $p, q \in C$ are **α-connected**, if there is a path joining p and q in $C \cap (B_\alpha(p) \cup B_\alpha(q))$, where $B_\alpha(x)$ is a ball of radius α at x.

For a known sketch shape Ω, this definition with $C = \Omega$ captures the intuition of points being intrinsically close (Fig. 7a). Points that are clearly separated by the boundary of Ω will be considered 'far' from each other, or not α-connected, even if the Euclidean distance between them is small. Note that while in that aspect α-connectedness is similar to an upper bound on a geodesic distance, in general, those concepts are different: Two points at a large geodesic distance can be nevertheless α-connected (Fig. 7b).

We can also apply this definition to the input sketch, where we take C to be the union of sketch strokes S, limiting all the paths to only lie on the strokes. We refer to the points that are α-connected for this choice of C as α-connected in S. Then like the sketch is an approximation of the sketch shape, α-connectedness (i.e., the set of all α-connected pairs of stroke points) over S should ideally be the approximation of the α-connectedness over the full shape Ω.

With this definition, we can now formalize the notion of a 'conservative' estimate of the shape. If the true sketch shape Ω is known, then by adding all 'missing' α-connections to the input strokes S, we can guarantee that all the *holes* in this sketch, defined as the faces containing an empty ball of radius α (by the sampling assumption), are either outside or almost completely outside Ω. Simply put, this would mean that the union of holes accurately captures the negative space of a sketch. More formally,

Proposition 3.1. *Let $S \subseteq \Omega \subset \mathbb{R}^2$ be two closed sets. If for all the pairs of points in S that are α-connected in Ω, they are also α-connected in S, then every empty face of S is either (1) completely outside Ω or (2) every boundary segment of Ω contained in that face is within α Hausdorff distance of the boundary of the face.*

The proof scheme is presented in the appendices (App. A). This proposition suggests the following idea of the algorithm. We need to augment the initial sketch S with additional connections to capture all the 'true' (unknown) α-connections between the sketch points. Many points are already α-connected in S: points along the same stroke within the α distance from each other or points on two intersecting strokes connected by a path passing over the intersection. With respect to the (unknown) sketch shape S', however, many more points are α-connected. Therefore, our goal is to add connections to the input sketch such that all the points that are α-connected in S' are also α-connected in our structure — then we would guarantee that we are not missing any holes of significant radius.

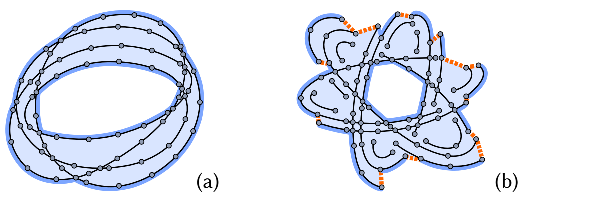

Fig. 6. For a sketch solely composed of closed strokes, its shape's boundary is entirely supported by the input strokes (a). For a typical sketch with open strokes, however, its sketch shape boundary often includes additional connections (b, orange dashed lines).

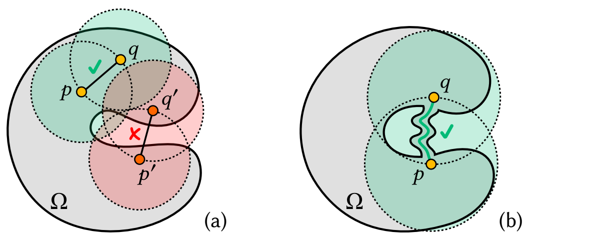

Fig. 7. Examples of α-connected and non-connected points.

However, doing so explicitly would be computationally expensive, as it would require adding numerous connections between many pairs of points. Instead, we only focus on adding such connections when they can contribute to the shape's boundary. Once the latter is known, it defines the shape interior and thus defines the α-connectedness for all the other points.

We first observe that away from endpoints, the boundary is entirely supported by the strokes, following a stroke until an intersection point, then switching to the intersecting stroke. In the simplest case, if a drawing is entirely composed of closed strokes and thus contains no endpoints, its whole boundary is a subset of the strokes (Fig. 6a). Given our assumption that strokes at a distance more than α from each other are not intrinsically close, the boundary of such a shape will simply consist of the unambiguous outer boundary and the boundaries of each empty face containing an empty ball of radius α.

Therefore, the only source of ambiguities is, in fact, endpoints. As the simplest solution, targeting a conservative estimate of a 2D shape, we could connect any two points within the α-radius from each other that are not already α-connected. However, we observe that since some of the strokes are boundary strokes, a segment intersecting a stroke might cross the shape's boundary — thus, adding it would create an incorrect α-connection, as these points are not necessarily α-connected in Ω. Therefore, in order to have a tight estimate of our shape, we only consider adding connections between an endpoint and a point on a different stroke that are (a) within the α distance and (b) if the segment connecting them does not intersect any stroke (Fig. 8b, 9d). Some of those connections complete the boundary of the shape (Fig. 6b).

3.3 Sketch Complex

To capture these observations, we define a *Sketch Complex* from the input set of strokes that decomposes the plane into regions that are *faces* in a 2D arrangement; we then consider all sufficiently large faces, including the unbounded face as the exterior, and define *Sketch Shape* as the complement to that (Fig. 8).

We take inspiration from the classical definition in computational topology [Ghrist 2014]:

Definition 3.2. Given a point cloud Q in $\mathbb{R}^n$ and a length parameter $\alpha > 0$, define the **Čech complex** to be the simplicial complex built on Q as follows. A k-simplex of $\check{C}_\alpha$ is a collection of $k+1$ distinct elements x_i of Q such that the intersection of diameter α balls at the x_i's is nonempty.

Since our goal is to define a decomposition of the plane rather than capturing the discrete topology of the set of samples, we do not aim to answer for every two samples in the sketch whether they are intrinsically connected. Instead, we only consider connecting endpoints to other samples, forming *Sketch Complex*:

Definition 3.3. Given a sketch, i.e., a set of strokes $S = \{s_i\}, i = 1, \ldots, N$, where each s_i is a sequence of sample points $s_i = \{v_i^j\}, j = 1, \ldots, n_i$, a **Sketch Complex** $\check{S}_\alpha$ is a simplicial complex built on S as follows. The complex $\check{S}_\alpha$ is the collection of all the segments connecting an endpoint v_i^1 or $v_i^{n_i}$ with any other sample within α-ball at the endpoint such that (1) the endpoint and the sample are not already α-connected in S and (2) the segment does not intersect the input sketch S.

For a fixed value of α, our complex can be viewed as a subcomplex of the Čech complex where we keep only the segments (1-simplices) that are adjacent to at least one endpoint and satisfy the two conditions above.

The Sketch Complex, together with the input strokes, define a 2D arrangement, or decomposition of the plane into faces, including one unbounded face (Fig. 8c). We now consider any face containing an empty α-ball as the exterior to a sketch and define the complement to this exterior as the **sketch shape**. Thus, the sketch shape includes all smaller bounded faces that do not allow for an in-circle of radius α, as well as some chunks of the input strokes (Fig. 5e, the single stroke in the bottom-right).

Note that the sketch shape satisfies the containment requirement (Sec. 3.2) by construction. We later show in Sec. 6 that our sketch shape, with a particular choice of α (Sec. 4.3), satisfies the requirements of being conservative and tight.

Note that isolated input strokes might lead to the boundary of a shape formally touching itself, creating points where the resulting 2D shape would be non-manifold (Fig. 5e). We allow that in the algorithm in Sec. 4, but later perform a simple post-processing step to guarantee our resulting 2D shape is a 2-manifold with boundary.

Computing Sketch complex, however, can be computationally expensive: Čech complexes, as well as our Sketch Complexes, can have high complexity and thus be prohibitively expensive to store or process. Instead, importantly, we are not interested in the complex itself, only in the boundaries of holes with a diameter more than α, so in our algorithm, we store only the segments that are not clearly inside the sketch shape (see Sec. 4).

4 ALGORITHM

In this section, we introduce the Alpha Contours algorithm, which takes as input a vector sketch and outputs the sketch shape, represented as a set of polygons (Fig. 8).

Sketch Preprocessing. We start by sampling each vector stroke along its arc length parameterization with a fixed step of 0.5% of the sketch bounding box. Our algorithm is robust with respect to sampling size, so this choice is rather arbitrary (Sec. 6). We first assume the single parameter α is given as a part of the input; we later compute it automatically in Sec. 4.3.

To start, we find all intersection points between the input strokes and add those to the set of samples. After, we apply the following preprocessing steps to the input strokes:

(a) We merge strokes if their endpoints are located very close to one another, within the 10^{-10} distance;
(b) We split each stroke at the samples with sharp angles ($< 90°$);
(c) We split strokes at their self-intersection points.

4.1 Algorithm Steps

Leveraging the observations we made (Sec. 3), conceptually, the algorithm is rather straightforward: We create a 2D arrangement of the input strokes with extra line segments closing the gaps. We then find the faces in that arrangement [Chazelle et al. 1993], defining a face as exterior if it contains an empty α-ball (Fig. 9).

The first key step is adding extra line segments to close the gaps, each time connecting only points that are not already α-connected over the sketch. A naïve solution would be to add all such segments into the arrangement. While that would produce the desired Sketch Complex, it can quickly become an efficiency bottleneck. Depending on the sampling of the strokes and the radius α, each endpoint may produce a significant number of extra segments and, therefore, intersection tests, leading to unacceptable times even for simple drawings.

Instead, our key observation is that for a given endpoint (Fig. 10, yellow), only a few of those line segments can affect the final sketch shape boundary (Fig. 10, thick green line segments). Precisely, if some of the other samples within the circle are pairwise α-connected, forming "α-connected component" (Fig. 10, one component of 6 samples on the top, another of 2 samples on the right), then the only extra segments that can affect the boundary of the final shape are the segments within each component forming the minimum and the maximum angle with the stroke tangent (thick green): All the intermediate segments are contained between the minimum and maximum and contained within the α radius of them, bounded from all sides by input strokes or the added segments. These bounded faces (shaded yellow) are inside the α-ball, and, therefore, by our

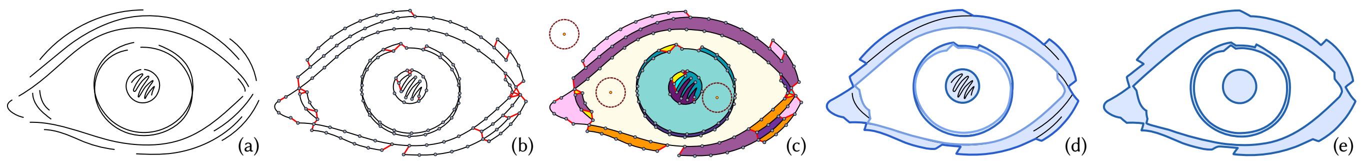

Fig. 8. Algorithm Overview. Starting with an arbitrary input sketch in vector format and provided with a fixed radius α (a), we first add segments completing the boundaries of the sketch shape and forming Sketch Complex (b), then we find faces containing an empty α-ball each (c). We mark the rest as interior (d, shaded blue), and post-process the final region to be 2-manifold, creating the final sketch shape (e).

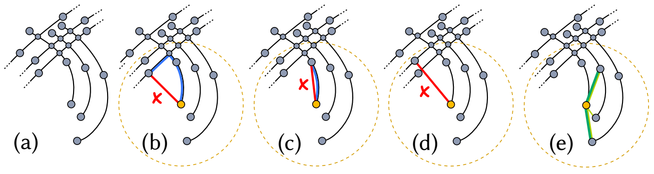

Fig. 9. Given the sampled input strokes (a), the Sketch Complex is composed of all segments between samples within α-distance such that they are not already α-connected (b,c) and do not intersect the sketch strokes (d). For the selected sample (e), the three allowed segments are in green, we only add the ones depicted in bold.

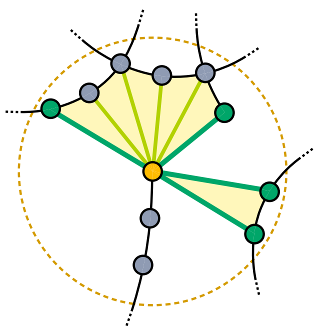

Fig. 10. For a given endpoint (center, yellow), only a few line segments can affect the sketch shape boundary (thick green segments).

definition, must belong to the interior. Hence, the intermediate segments do not affect the boundary of the shape we seek. We thus modify the naïve algorithm: for each endpoint, we only add the segments forming the minimal and the maximal angles with the stroke tangent for each α-connected component. In our implementation, we further optimize it: We skip the α-connectedness test between all the pairs of samples and only keep the segments with the minimal and maximal angles. This gives us a performance boost with only a minor difference in quality.

The second key step is testing whether a face of the arrangement has an empty α-ball. As the largest empty circle within a face must be equidistant from at least two points of the face's boundary, the circle's center belongs to the face's medial axis [Huber 2018]. Therefore, to implement this test, for each face of the arrangement, we find Voronoi vertices of the *curvilinear* Voronoi diagram of the face boundary segments [Karavelas 2004]. These Voronoi vertices are an approximation of the medial axis; we then test whether there is a single Voronoi vertex at a distance α or greater from the face boundary. As a simple optimization, we skip faces that have an area less that $\pi\alpha^2$, because they clearly cannot contain an empty ball of that size.

The full pseudocode, which we dub Alpha Contours, is presented in Alg. 1.

```
**Data:** A set of strokes $S = \{s_i\}, i = 1, \ldots, N$, where each
    $s_i = \{v_i^j\}, j = 1, \ldots, n_i$
**Result:** A set $C \subset \mathbb{R}^2$, the sketch shape
$\check{S} = S$;
**for** *each endpoint p in S* **do**
    $Seg(p) \longleftarrow \{(p, v) | v \in S, ||v - p|| \leq \alpha\}$ ;
    $Seg'(p) \longleftarrow \{(p, v) \in Seg(p) | (p, v)$ are not $\alpha -$
      connected in $S$, $(p, v)$ does not intersect $S\}$;
    **for** *each $\alpha$-connected component D of $Seg'(p)$* **do**
        $\check{S} = \check{S} \cup \{\text{argmin}_{(p,v)\in D} Angle(p, v)\} \cup$
          $\{\text{argmax}_{(p,v)\in D} Angle(p, v)\}$;
    **end**
**end**
$E \longleftarrow$ unbounded face of $\check{S}$;
**for** *each bounded face f in $\check{S}$* **do**
    **if** *Area(f) > $\pi\alpha^2$* **then**
        $V = VoronoiVertices(f)$;
        $R = \max_{v \in V} dist(S, v)$;
        **if** $R \geq \alpha$ **then**
            $E \longleftarrow E \cup f$;
        **end**
    **end**
**end**
$C \longleftarrow \mathbb{R}^2 \setminus E$;
```

Algorithm 1: Alpha Contours algorithm.

The final result, the sketch shape $C \subset \mathbb{R}^2$ is represented as its boundary — a set of closed curves $\Gamma_i, i = 1, \ldots, n$, stored as polylines, each marked as either 'exterior' or 'interior'. Exterior contours can be thought of separating foreground from background, while interior contours cut out holes in the objects. For example, in Fig. 8e, the resulting sketch shape has five boundary curves in total, three exterior and two interior ones.

4.2 Enforcing Manifoldness

Note that the boundary of the extracted sketch shape might touch itself or follow along isolated strokes so that the final domain becomes non-manifold. To alleviate this issue, we then slightly translate the affected boundary samples in the normal direction (by a fixed step of $\varepsilon = 10^{-5}$ of the sketch bounding box in our implementation), thus locally inflating the domain (Figs. 8e, 11).

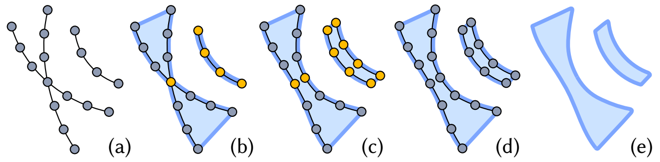

Fig. 11. For some input stroke configurations (a), our algorithm creates points where the sketch shape is non-manifold (b, yellow points). To address that, at those points, we displace the shape boundary in the normal directions (c), yielding a 2-manifold shape (d,e).

4.3 Computing α

For some classes of drawings, the value of α can be defined in a manner consistent with human perception: For instance, for sketches composed solely of clusters, Liu et al. [2018] use a perceptual threshold of 2.1 average distance between strokes, where the average distance is computed by shooting a perpendicular from each stroke sample and computing the distance to its closest intersection. For general sketches, however, there is no known threshold, and we note that such an estimation of the average distance between strokes might be unreliable, giving somewhat larger values. In our experiments, therefore, we set it to simply twice the average distance between strokes. We furthermore show that our method provides meaningful results with different values of α, allowing us to use it in a general context. We note, however, that α must be necessarily larger than the largest distance between the adjacent samples along the same stroke for the result to be meaningful.

5 DIFFERENTIAL OPERATORS

The computed sketch shape follows some samples in the input strokes and forms a manifold shape tightly containing the input geometry. At this point, one can use the geometric domain directly to perform shape-aware computations on a sketch, for instance, geodesic distances. We demonstrate the construction of two popular discrete differential operators on this domain: the Laplace-Beltrami operator via a triangulation of the domain's interior and the Steklov operator that only uses the known discretization of the boundary. Note that those are the operators defined on the sketch shape, but since the shape contains the input strokes entirely, each function on the sketch strokes can be defined as a restriction of a function over the sketch shape.

5.1 Laplacian Operator

To define the Laplace-Beltrami operator for the 2D sketch shape, we start by triangulating the shape. We employ the Constrained Delaunay Triangulation [Shewchuk 1996], specifying all samples of the input strokes as vertices and constraining the edges that belong to the boundary of the extracted shape.

We then compute the standard cotangent Laplacian of that triangulation. We demonstrate some of its eigenfunctions in Fig. 12a. In Fig. 13, we highlight the Fiedler vector, or the eigenvector corresponding to the second smallest eigenvalue, often used in graph and shape partitioning.

5.2 Steklov Operator

We note that triangulating the domain's interior, once we have the boundary provided by the alpha contours, is not necessary to define a differential operator. We can use the boundary only to solve PDEs via Boundary Element Methods. In particular, in a similar spirit to the mesh Steklov operator [Wang et al. 2018], we define the Sketch Steklov operator to capture the shape's extrinsic geometry as follows.

Let us denote the extracted alpha contours as $\Gamma_i, i = 1, \ldots, n$, bounding the sketch shape Ω. We first orient the alpha contours such that Ω is always on their left, meaning counterclockwise for the outer contours, clockwise for the inner ones. Then the complete boundary $\partial\Omega = \bigcup_{i=1}^{n} \Gamma_i = \Gamma$ can be integrated on. Using the fundamental solution of the flat 2D Laplacian $G(x, y) = \frac{1}{4\pi}\frac{1}{|x-y|}$, we can now define the boundary operators [LaForce 2006]:

Definition 5.1. The single layer potential $\mathcal{V} : H^{-1/2}(\Gamma) \to H^{1/2}(\Gamma)$ as

$$[\mathcal{V}\phi](x) := \int_\Gamma G(x, y)\phi(y)d\Gamma(y),$$

where H^a denotes Sobolev space of order a. Analogously, we define

Definition 5.2. The double layer potential $\mathcal{K} : H^{1/2}(\Gamma) \to H^{1/2}(\Gamma)$ as

$$[\mathcal{K}\phi](x) := \int_\Gamma \frac{\partial G(x, y)}{\partial n(y)}\phi(y)d\Gamma(y).$$

Finally, we can define the Steklov operator that can be thought of as a mapping between Dirichlet boundary conditions on Γ for a harmonic function into its Neumann boundary conditions:

$$\mathcal{S} = \mathcal{V}^{-1}(0.5\mathcal{I} + \mathcal{K}).$$

We discretize the single layer and double layer potentials using constant elements [LaForce 2006]. We then directly compute the Steklov operator via explicit matrix inverse. While one can, in principle, use a different expression to compute the Steklov operator to guarantee it is symmetric or compute it in a more efficient manner, as suggested by Wang et al. [2018], we did not find it necessary for our setup. As a demonstration, we compute the 1D Steklov operator of the extracted shape boundaries, compute its eigenfunctions, and interpolate them inside the domain harmonically (Fig. 12b). Precisely, having computed an eigenfunction $\phi(x)\big|_\Gamma$ we interpolate it inside the domain Ω as follows:

$$\phi(x) = [\mathcal{V}\frac{\partial\phi(x)}{\partial n}](x) - [\mathcal{K}\phi](x),$$

where $\frac{\partial\phi(x)}{\partial n}\big|_\Gamma = [S\phi](x)\big|_\Gamma$.

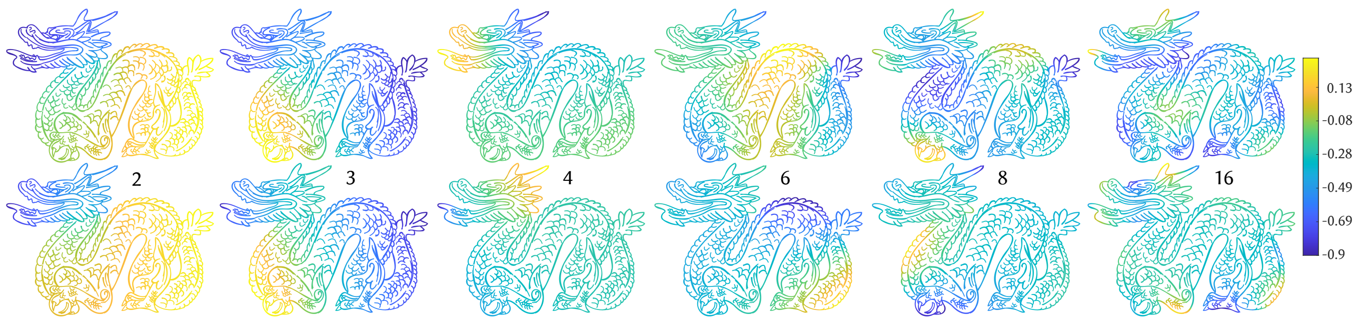

Fig. 12. Some of the eigenfunctions of the Laplacian (top) and 1D Steklov (bottom) operators of the sketch shape, restricted (top) or harmonically interpolated (bottom) to the input sketch strokes. For this figure, we discard all the holes in the sketch shape.

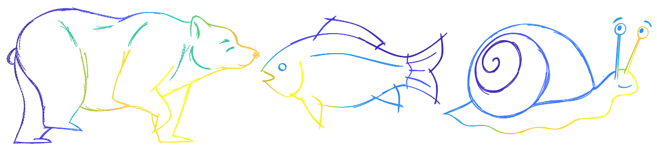

Fig. 13. Fiedler vector, or the eigenvector corresponding to the second smallest eigenvalue of the sketch Laplacian operator. Fish image ©Enrique Rosales.

6 VALIDATION AND DISCUSSION

Up until now, we have demonstrated the results of our algorithm on many input sketches, including the ones mostly consisting of stroke clusters considered in previous work (Figs. 2, 8), as well as general sketches containing many strokes beyond clusters (Figs. 3, 4), such as shading or hatching. We demonstrate that our algorithm works robustly on sketches both directly drawn in the vector format and converted from raster via vectorization methods (see Supplementary materials).

We validate the key aspects of our algorithm in a number of ways.

Conservative Estimate. We first validate that our method indeed produces a conservative estimate of the 2D sketch shape, i.e., it does not separate stroke samples that are perceived as adjacent. While there is no such ground truth data for general vector sketches, we evaluate it on sketches composed of clusters, where the previous methods can robustly estimate such data, excluding some ambiguous regions. On all our input sketches that are composed only of stroke clusters, we run our algorithm with three radii: the automatic value of α, 1.25α and 2α. To measure how much our contours separate intrinsically close samples, we first segment each sketch into clusters using StrokeAggregator [Liu et al. 2018] and parameterize each cluster via StrokeStrip [Van Mossel et al. 2021]. We then consider all pairs of points within a cluster with the same parameter value as intrinsically close and measure the percentage of them that get separated by our sketch shape — in other words, if the isoline segment containing them is not entirely inside the sketch shape. The statistics are presented in Table 1. As seen from the table, our algorithm indeed gives a very conservative estimate of the shape, only separating 0.8% pairs of points on average with the default radius α, or 0.3% with the 1.25α. The only notable exceptions are the bear and the snail, where the strokes are drawn very neatly, and clusters are very tight, leading to an underestimation of the radius.

Comparing with Alpha Shapes. While our method is inspired by Alpha Shapes, our results differ from it in a few important ways (Fig. 14). Alpha Shapes ignore the connectivity of the input strokes, connecting stroke samples only based on their proximity. As a result, alpha shapes typically contain either too much or too little of the positive space of the sketch, often making the drawn shapes unrecognizable (Fig. 14, top and bottom). Our algorithm is designed to avoid all these issues (Fig. 14, right), always producing contours that contain the input strokes yet have an area less than Alpha Shapes. Our sketch shapes are precisely aligned to the input strokes, allowing a more precise approximation of the drawn positive space. An example of why those are important issues is Fig. 19 and its description (Sec. 7).

Finally, compared to Alpha Shapes, our algorithm produces shapes of significantly smaller area for the the same value of α. On our inputs, our extracted sketch shapes have from 8% (fox, Fig. 19) to 41% (dancer, Fig. 15) smaller area than the corresponding alpha shapes (23% on average), making our sketch shapes satisfy the *tightness* requirement (Sec. 3.1). The full statistics on areas are presented in the Supplementary materials.

Comparison with Amenta et al. [1998]. Another classical contour reconstruction algorithm [Amenta et al. 1998] also ignores the stroke connectivity, accepting only the point cloud as the input (Fig. 17). Unfortunately, this algorithm assumes the 2D point cloud is sampled from a 1-manifold, so it is not robust even for rather simple sketches with overdrawn strokes. Overdrawn strokes are often used to depict width; thus, the samples do not come from a 1-manifold but rather a 2-dimensional region that does not have to be a manifold.

Cluster Drawings. Our algorithm can be used on any vector sketches, including cluster drawings, i.e., drawings composed solely of stroke clusters (Fig. 15b). For this subclass of sketches, we are complementary to prior vector consolidation work that aims to segment the input strokes into clusters. In particular, once the clusters are computed (e.g., via StrokeAggregator [Liu et al. 2018]), our algorithm can be trivially used to outline the contour of each, then the set union of those forms a precise shape of the sketch (Fig. 15c). Information about the clusters makes our results more precise on this kind of drawings, albeit at a performance cost: while our algorithm takes only 1.2 seconds to compute the complete result (Fig. 15b), StrokeAggregator takes 145 seconds on the same machine, making the full processing time of Fig. 15c to be 146 seconds. We note, however, that such precision is often not necessary for a variety of applications, see Sec. 7. We compute all other results directly without any pre-consolidation.

Comparison with StrokeStrip. The cluster parameterization method StrokeStrip [Van Mossel et al. 2021] can be used to fit a curve of varying width into a stroke cluster. Theoretically, it can be used to compute the shape of a pre-clustered drawing. While the result is visually comparable to ours (Fig. 16), their strips do not contain the input strokes, making them hard to use in our applications. Their method also further slows down the performance: For the dancer example, in addition to the 145s of StrokeAggregator necessary for clustering the input drawing, StrokeStrip takes extra 10 seconds; our algorithm terminates in 1.2 seconds in total.

More importantly, these algorithms fail on general sketches that are not cluster drawings (Fig. 16d). Our method does not have that limitation (Fig. 16e).

Robustness. Our algorithm is controlled by the single parameter α, which we estimate automatically (Sec. 4.3) for our results, but allows α to be controlled by the user if desired. Changing the α value controls which points are considered intrinsically close; intuitively, the higher the α, the larger gaps are considered unintentional, and the larger the sketch shape is (Fig. 18, top). Note that for any value of α, our sketch shape stays a 2D manifold aligned to the input drawing, containing all the input strokes.

Furthermore, our algorithm is robust to the choice of initial stroke sampling, as changing the sampling density four times leads to only marginal differences in the result (Fig. 18, bottom).

Performance. Our algorithm is easy to implement, robust, and efficient. We implemented it in C++ using CGAL library [Wein et al. 2022]. On our machine (Intel ®i9 9900K 3.6GHz, 128Gb RAM), ten of our 26 results with our alpha parameter take less than 2s each (flower, eye, dancer, koala, hand, bear, daisy, bird, snail, and turtle), the others usually complete in under 20 seconds. The two exceptions are densely drawn sketches, such as the girl with a blanket (Fig. 14, middle, 114s) and the wizard (Fig. 19, 67s). The full performance statistics are presented in Table 2. The bottleneck is curve-segment intersection tests, which can be further optimized. The total time depends on the number of strokes and the overall number of samples.

7 APPLICATIONS

Sketch Correspondences. Algorithmically finding correspondences between two vector sketches is an open problem that is one of the main roadblocks on the way towards automatic cartoon inbetweening, among other applications. Despite significant progress in frame interpolation in the raster domain, the state-of-the-art methods for vector drawing correspondences often rely on the two drawings having the same topology and manually specified correspondences [Whited et al. 2010] or consisting solely of closed regions [Zhu et al. 2016].

Table 1. For the drawings composed solely of clusters, our sketch shapes very rarely separate adjacent strokes. To quantify that, we run our algorithm with three parameter values: α (the automatically computed value), 1.25α and 2α. We then segment these sketches into stroke clusters via StrokeAggregator [Liu et al. 2018] and parameterize each cluster via StrokeStrip [Van Mossel et al. 2021]. Finally, for the same parameter value, we compute the percentage of pairs of points that get separated by our shape contours. As seen from the table, those instances are very rare and diminish or disappear with the increasing radius.

Input	α	1.25α	2α
Muscular	0.1%	0%	0%
Dancer	0.2%	0%	0%
Koala	0.1%	0%	0%
Hand	0.9%	0.6%	0.6%
Toucan	0.1%	0%	0%
Bear	2.8%	0.1%	0%
Sparrow	0.8%	0.7%	0.4%
Snail	1.9%	1.7%	0.9%
Turtle	0.9%	0%	0%
Fish	0%	0%	0%
Average	0.8%	0.3%	0.2%

Table 2. Performance statistics for sketches. All other inputs take 2s or less each.

	n. of strokes	our time
Hummingbird	308	5s
Muscular	544	4s
A	231	4s
Dragon	822	16s
Witch	554	12s
Girl with a blanket	2920	114s
Boy with a dog	491	14s
Kettle	316	8s
Dirigible	816	10s
Human back	329	23s
Toucan	327	3s
Fox	292	7s
Rabbit	251	3s
Wizard	873	67s
Spiderman	692	27s

One of the exciting applications of our method is enabling state-of-the-art methods for functional maps [Ovsjanikov et al. 2012] between vector sketches of arbitrary topology. In our experiments, we use a recent method by [Ren et al. 2021].

We note that the choice of α values computed using our heuristic in Sec. 4.3 is not specifically designed to facilitate the computation of functional maps and thus leads to suboptimal maps. Our heuristic serves more as a lower bound for a reasonable choice of α. Instead, for this application only, we perform a line search procedure, where we sample the interval $I = [\alpha', 2\alpha']$, where α' is the automatically computed value. For each value of α, we compute the two sketch shapes with that value using our algorithm, and then compute the

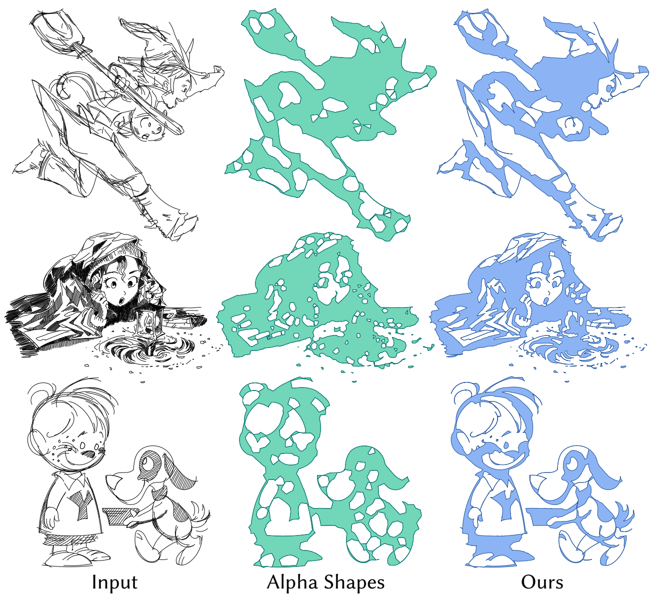

Fig. 14. Compared with Alpha Shapes, our algorithm leverages the connectivity of the input strokes to create tighter (23% less area on average) shapes that better preserve sketch details (compare, e.g., faces of the characters). Input drawings: Witch and girl ©David Revoy CC-BY-4.0, boy with a dog ©Cartoon Animation by Preston Blair (1994), Mission Viejo, CA: Walter Foster Publishing [Yan et al. 2020].

functional map via the method of Ren et al. [2021] with their default parameters on the two triangulated sketch shapes. For each functional map, we then compute the energy measuring the commutativity of the Laplacian, as proposed by Ovsjanikov et al. [2012], and minimize it over the parameter interval, solving

$$\min_{\alpha \in I} E(\alpha) = \min_{\alpha \in I} ||C_{12}\Delta_1 - \Delta_2 C_{12}||^2_{Fro}, \tag{1}$$

where C_{12} is the resulting functional map, and Δ_i is the Laplacian matrix for the corresponding sketch shape, both functions of α (we omit the notation for brevity). We then choose the value of α within that interval with minimal energy.

This line search procedure is expensive, as it uses both our algorithm and the functional map optimization many times, taking from a couple of minutes on simple drawings (hummingbird, daisy, bird, fox, bunny) to a few hours on the complex ones (witch, wizard, spiderman).

Note also that in the energy (Eq. 1) we do not use the full energy presented in Ovsjanikov et al. [2012] that involves a term measuring the preservation of descriptors, as we found the standard descriptors, such as wave kernel signature, unreliable in our context.

The results, where corresponding points are visualized with the same color, are presented in Fig. 1 and Fig. 19. We can observe that despite the two drawings having a different number of strokes and vastly different topology, as well as shapes of the individual strokes, their strategy finds robust correspondences. We hope this will be used in future inbetweening systems for vector drawings.

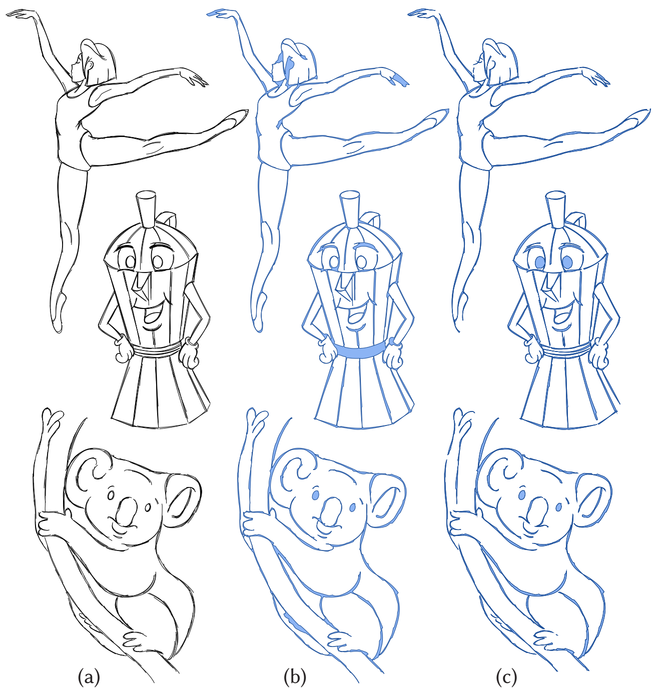

Fig. 15. For sketches composed only of stroke clusters, our algorithm is complementary to the stroke clustering algorithms, such as StrokeAggregator [Liu et al. 2018]. For the input sketches (a), we compare our direct result (b) with the result of our algorithm run on pre-clustered input (c). Pre-clustering indeed makes our algorithm more precise in complex scenarios (cf. fingers of the dancer), which may or may not be significant depending on the application, but at a significant performance cost: For the dancer example, our automatic result (b) takes 1.2s, while together with segmentation (c), the running time increases to 146s. Koala image ©Enrique Rosales.

The tightness requirement (Sec. 3.1) is crucial for the quality of functional maps. As we find functional maps between sketch shapes that contain strokes, the tighter the sketch shape is, the less of the function will be mapped outside the strokes. Simply put, a Dirac delta centered at a point in the first sketch may map to a function with a maximum outside the strokes in the second sketch, albeit within the sketch shape. The tightness requirement effectively minimizes this effect.

Please see Supplementary for the comparison of functional maps on our sketch shapes with the ones computed on the α-shapes using the same procedure. Note that the optimal radii for our algorithm and for the α-shapes can be different.

Solving PDEs on Sketches. The domain our method extracts can be directly used to solve partial differential equations (PDEs) on the sketch shape. As an example, we solve the heat equation modeling heat transfer from a single point source. The discretization of the heat equation depends mainly on the discretization of the Laplacian operator, which we define in Sec. 5. We compare it with the same equation solved using a point cloud Laplacian operator discretization from Sharp and Crane [2020], where we take all the samples of the input strokes as a point cloud. We additionally compare it with the

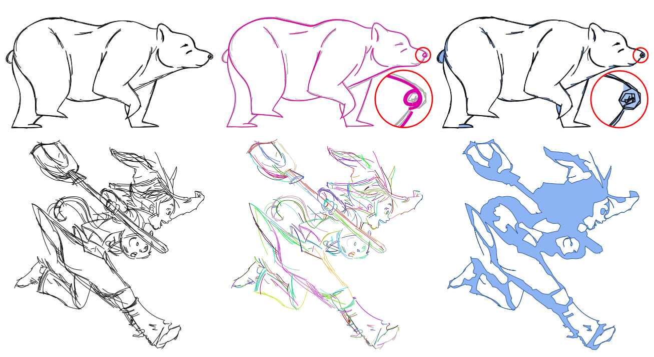

Fig. 16. For an input sketch composed solely of clusters (bear, top), running StrokeStrip [Van Mossel et al. 2021] on the output of StrokeAggregator [Liu et al. 2018] can provide a similar shape (pink) to ours (blue). Unfortunately, their shape does not contain the input drawing (blowup) and thus cannot be used for our purposes. More importantly, their methods do not support general sketches (witch, bottom); here, StrokeStrip fails, we only show the output of StrokeAggregator, which also fails to produce a reasonable segmentation. Our algorithm works on any input vector sketches (right). Witch image ©David Revoy CC-BY-4.0 [Yan et al. 2020].

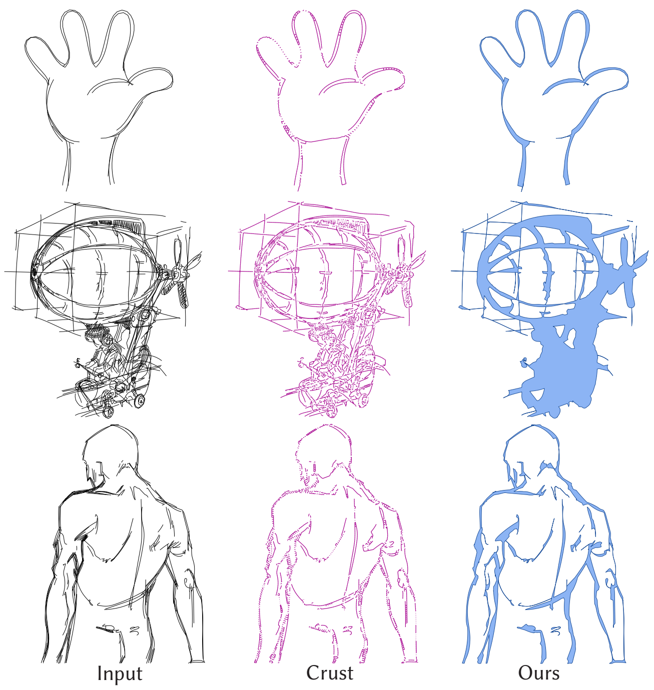

Fig. 17. The classical algorithm by [Amenta et al. 1998] assumes points are sampled from a 1-manifold. Generic sketches, however, do not satisfy this requirement, making their algorithm fail on typical inputs. Our results (right). Input drawings: Human back ©Anton Gulic, dirigible ©David Revoy CC-BY-4.0 [Yan et al. 2020].

heat flow discretized on a pixel image obtained by rasterizing the input vector drawing and dilating it with our α radius. We use an implicit time-stepping integration scheme for all these experiments

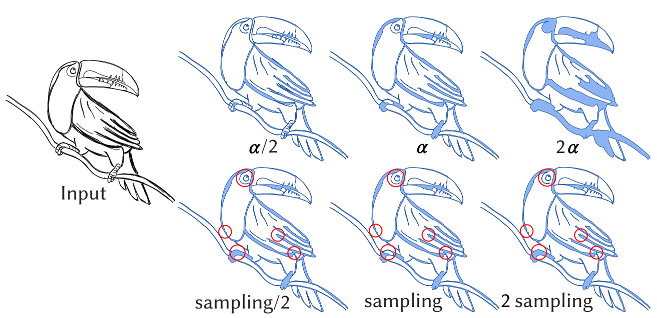

Fig. 18. The only parameter of our algorithm, α, controls the topology and the geometry of the extracted sketch shape (top row, α is the automatically computed value). The algorithm is robust to changing the sampling of the input sketch, only exhibiting minor differences. Toucan image ©Enrique Rosales.

(Fig. 20). Note that our method successfully separates non-adjacent parts of the drawing, allowing the heat to flow along the drawn strokes; point cloud Laplacian and the bitmap (graph) Laplacian ignore the stroke connectivity, leading to heat 'leaking' through the empty space. For instance, point cloud Laplacian relies on k-nearest neighbors instead as a proxy of adjacency. Please see Supplementary materials for the heat flow on the complete domains.

Similarly, we can solve the Laplace equation with the given Dirichlet boundary conditions to perform harmonic interpolation, as we show in Fig. 21.

Sketch Deformation. Our method enables deforming rough vector sketches via the standard deformation methods such as As-Rigid-As-Possible (ARAP) [Sorkine and Alexa 2007] (Fig. 22). Similarly to the previous setup, we apply ARAP to the triangulated domain, then simply read off the transformations of the input samples, as they are a subset of the mesh vertices. Depending on the desired deformation, one may choose to either use the complete sketch shape or only consider the exterior boundaries we extract (i.e., adding all the holes back into the sketch shape), thus simply separating 'background' from 'foreground'. The former strategy would allow deforming, for instance, bunny's nose (Fig. 22, left) separately from the rest of its face; the latter would treat the bunny as a solid mesh without holes and deform nose together with the rest of the body and face. In Fig. 22, we mesh the sketch shape, taking all the extracted α-contours into account.

8 CONCLUSIONS, LIMITATIONS, AND FUTURE WORK

We have presented Alpha Contours, the algorithm to approximate the shape of a vector sketch, or its positive space. We have demonstrated that this shape is essential to discretizing differential operators on a sketch. Our algorithm allows us to use standard geometry processing algorithms based on those operators on vector sketches, paving the way for more robust sketch correspondences, deformation and animation systems, colorization, and many other applications. Our algorithm is robust, efficient, easy to implement, and is controlled by a single parameter α.

We have defined the positive and negative space of a sketch in a purely geometric manner (Sec. 3.2). It remains unclear whether

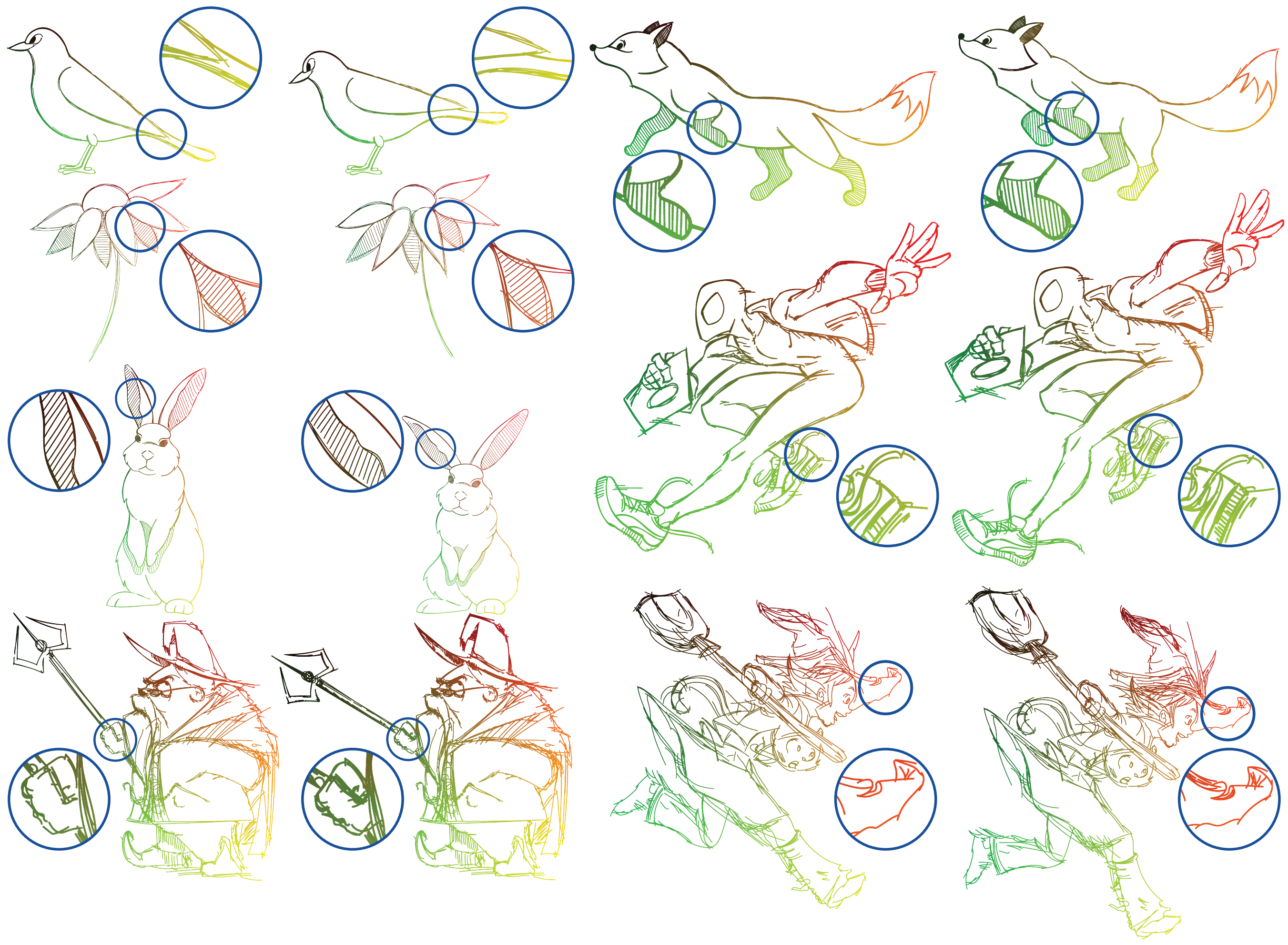
Fig. 19. Our algorithm enables using prior work [Ren et al. 2021] to automatically compute functional maps between vector sketches. Note that despite visual similarity, the sketches have different connectivity and number of strokes (see blow-ups). Input images: Spider-man ©Graham Wilson, witch ©David Revoy CC-BY-4.0, wizard ©AP CC-BY-SA-3.0 [Yan et al. 2020].

our construction agrees with human perception or to which degree human observers are consistent when interpreting positive and negative spaces of a sketch. We leave careful analysis of this phenomenon to future work.

Currently, we do not directly control the topology of our sketch shapes, sometimes generating shapes with multiple connected components (e.g., Fig. 18). While for many applications, that might be the desired behavior, this makes the problem of sketch correspondence (Sec. 7) harder, as it can lead to a discrete matching between disconnected pieces on two sketches. We leave addressing this limitation to future work. Another limitation is that our algorithm may not separate stroke endpoints that visually belong to different clusters (Fig. 15b), in which case one may use our algorithm in conjunction with a stroke consolidation algorithm, improving the results (Fig. 15c).

An exciting future research is investigating how the functional maps can be used for a meaningful inbetweening for two sketches.

ACKNOWLEDGMENTS

We thank Justin Solomon and Edward Chien for insightful conversations. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) under Grant No.: RGPIN-2019-05097 (“Creating Virtual Shapes via Intuitive Input”), the NSERC - Fonds de recherche du Québec - Nature et technologies (FRQNT) NOVA Grant No. 314090, and MITACS Acceleration Grant IT24983.

REFERENCES

Adobe. 2020. *Illustrator*. https://www.adobe.com/ca/products/illustrator.html

Nina Amenta, Marshall Bern, and Manolis Kamvysselis. 1998. A new Voronoi-based surface reconstruction algorithm. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. 415–421.

Saeed Asaeedi, Farzad Didehvar, and Ali Mohades. 2017. α-Concave hull, a generalization of convex hull. *Theoretical Computer Science* 702 (2017), 48–59. https://doi.org/10.1016/j.tcs.2017.08.014

Paul Asente, Mike Schuster, and Teri Pettit. 2007. Dynamic Planar Map Illustration. *ACM Trans. Graph.* 26, 3 (jul 2007), 30–es. https://doi.org/10.1145/1276377.1276415

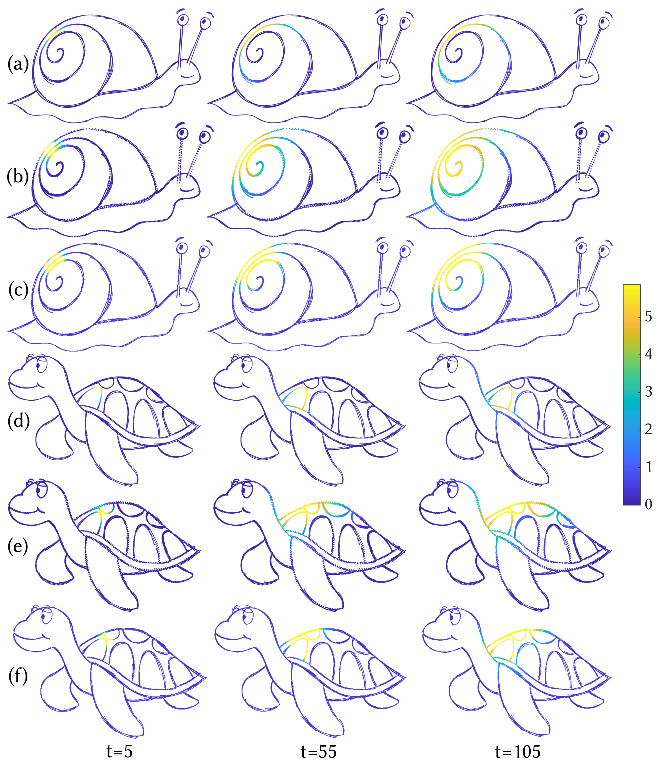

Fig. 20. Heat flow. For each example, we use the Laplacian defined on our sketch shape (a,d) compared to a point cloud Laplacian (b,e) and a raster image dilated with our α radius (c,f). Note how in point cloud and the dilated domain results, heat flows through the empty space; our results do not have that issue.

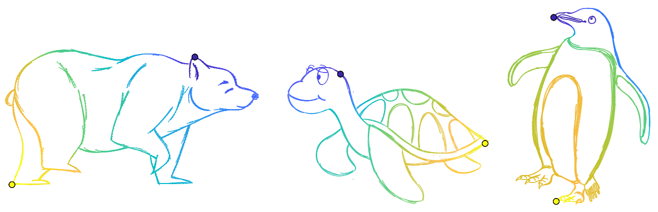

Fig. 21. Harmonic interpolation between two values (blue and yellow circles). Penguin image ©Enrique Rosales.

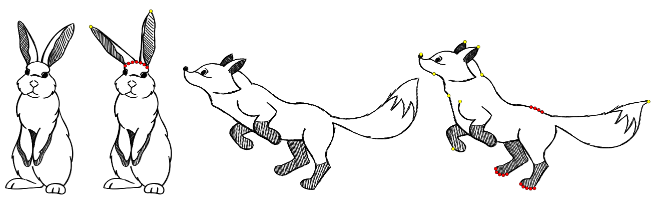

Fig. 22. Deformation of sketches obtained using As-Rigid-As-Possible framework. The yellow and red handles depict the translated and fixed in place vertices, respectively.

Pascal Barla, Joëlle Thollot, and François X. Sillion. 2005. Geometric Clustering for Line Drawing Simplification. In *ACM SIGGRAPH 2005 Sketches* (Los Angeles, California) *(SIGGRAPH '05)*. Association for Computing Machinery, New York, NY, USA, 96–es. https://doi.org/10.1145/1187112.1187227

A. Bartolo, K. P. Camilleri, S. G. Fabri, J. C. Borg, and P. J. Farrugia. 2007. Scribbles to Vectors: Preparation of Scribble Drawings for CAD Interpretation. In *Sketch-Based Interfaces and Modeling*.

Mikhail Belkin, Jian Sun, and Yusu Wang. 2009. Constructing Laplace Operator from Point Clouds in Rd. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms* (New York, New York) *(SODA '09)*. Society for Industrial and Applied Mathematics, USA, 1031–1040.

Mirela Ben-Chen, Ofir Weber, and Craig Gotsman. 2009. Spatial Deformation Transfer. In *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*, Eitan Grinspun and Jessica Hodgins (Eds.). ACM SIGGRAPH / Eurographics Association. https://doi.org/10.1145/1599470.1599479

Mikhail Bessmeltsev, Will Chang, Nicholas Vining, Alla Sheffer, and Karan Singh. 2015. Modeling Character Canvases from Cartoon Drawings. *ACM Trans. Graph.* 34, 5, Article 162 (nov 2015), 16 pages. https://doi.org/10.1145/2801134

Junjie Cao, Andrea Tagliasacchi, Matt Olson, Hao Zhang, and Zhinxun Su. 2010. Point Cloud Skeletons via Laplacian Based Contraction. In *2010 Shape Modeling International Conference*. 187–197. https://doi.org/10.1109/SMI.2010.25

Sara Casti, Marco Livesu, Nicolas Mellado, Nadine Abu Rumman, Riccardo Scateni, Loïc Barthe, and Enrico Puppo. 2019. Skeleton based cage generation guided by harmonic fields. *Computers & Graphics* 81 (2019), 140–151. https://doi.org/10.1016/j.cag.2019.04.004

Bernard Chazelle, Herbert Edelsbrunner, Leonidas Guibas, Micha Sharir, and Jack Snoeyink. 1993. Computing a Face in an Arrangement of Line Segments and Related Problems. *SIAM J. Comput.* 22, 6 (1993), 1286–1302. https://doi.org/10.1137/0222077 arXiv:https://doi.org/10.1137/0222077

Jiazhou Chen, Mengqi Du, Xujia Qin, and Yongwei Miao. 2018. An Improved Topology Extraction Approach for Vectorization of Sketchy Line Drawings. *Vis Comput* 34, 12 (Dec. 2018), 1633–1644.

U. Clarenz, M. Rumpf, and A. Telea. 2004. Finite Elements on Point Based Surfaces. In *SPBG'04 Symposium on Point - Based Graphics 2004*, Markus Gross, Hanspeter Pfister, Marc Alexa, and Szymon Rusinkiewicz (Eds.). The Eurographics Association. https://doi.org/10.2312/SPBG/SPBG04/201-211

Pedro Company, Raquel Plumed, Peter A. C. Varley, and Jorge D. Camba. 2019. Algorithmic Perception of Vertices in Sketched Drawings of Polyhedral Shapes. *ACM Trans. Appl. Percept.* 16, 3, Article 18 (aug 2019), 19 pages. https://doi.org/10.1145/3345507

Bert Dodson. 1990. *Keys to drawing*. Penguin.

Matt Duckham, Lars Kulik, Mike Worboys, and Antony Galton. 2008. Efficient generation of simple polygons for characterizing the shape of a set of points in the plane. *Pattern Recognition* 41, 10 (2008), 3224–3236. https://doi.org/10.1016/j.patcog.2008.03.023

Herbert Edelsbrunner. 1992. Weighted alpha shapes. *Technical Report UIUCDCS-R-92-1760* (1992).

Herbert Edelsbrunner, David G. Kirkpatrick, and Raimund Seidel. 1983. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory* 29, 4 (1983), 551–559. https://doi.org/10.1109/TIT.1983.1056714

Vage Egiazarian, Oleg Voynov, Alexey Artemov, Denis Volkhonskiy, Aleksandr Safin, Maria Taktasheva, Denis Zorin, and Evgeny Burnaev. 2020. Deep vectorization of technical drawings. In *European Conference on Computer Vision*. Springer, 582–598.

Jean-Dominique Favreau, Florent Lafarge, and Adrien Bousseau. 2016. Fidelity vs. Simplicity: A Global Approach to Line Drawing Vectorization. *ACM Trans. Graph.* 35, 4 (July 2016), 120:1–120:10.

Mark Finch, John Snyder, and Hugues Hoppe. 2011. Freeform Vector Graphics with Controlled Thin-plate Splines. *ACM Trans. Graph.* 30, 6 (2011), 166:1–166:10.

William T Freeman, Joshua B Tenenbaum, and Egon C Pasztor. 2003. Learning style translation for the lines of a drawing. *ACM Trans. Graph.* 22, 1 (2003), 33–46.

Michel Gangnet, Van Thong, Avenue Hugo, and Jean-Daniel Fekete. 1994. Automatic Gap Closing for Freehand Drawing.

Robert Ghrist. 2014. *Elementary Applied Topology*. Createspace. https://www2.math.upenn.edu/~ghrist/notes.html

Stéphane Grabli, Frédo Durand, and François X. Sillion. 2004. Density Measure for Line-Drawing Simplification. In *Proceedings of Pacific Graphics*. Seoul, South Korea. https://hal.inria.fr/inria-00510163

Yulia Gryaditskaya, Felix Hähnlein, Chenxi Liu, Alla Sheffer, and Adrien Bousseau. 2020. Lifting Freehand Concept Sketches into 3D. *ACM Trans. Graph.* 39, 6, Article 167 (nov 2020), 16 pages. https://doi.org/10.1145/3414685.3417851

Stefan Huber. 2018. The topology of skeletons and offsets. In *Proc. 34th Europ. Workshop on Comp. Geom.(EuroCG'18)*.

Johannes Itten. 1976. *Design and Form: The Basic Course at the Bauhaus and Later* (paperback ed.). Van Nostrand Reinhold. 136 pages.

Alec Jacobson, Ilya Baran, Jovan Popović, and Olga Sorkine. 2011. Bounded Biharmonic Weights for Real-Time Deformation. *ACM Trans. Graph.* 30, 4, Article 78 (jul 2011), 8 pages. https://doi.org/10.1145/2010324.1964973

Menelaos I. Karavelas. 2004. A robust and efficient implementation for the segment Voronoi diagram. In *In Proc. Internat. Symp. on Voronoi diagrams in Science and Engineering (VD2004)*. 51–62.

Byungsoo Kim, Oliver Wang, A. Cengiz Öztireli, and Markus Gross. 2018. Semantic Segmentation for Line Drawing Vectorization Using Neural Networks. *Comput. Graph. Forum* 37, 2 (2018), 329–338.

Tara LaForce. 2006. PE281 Boundary Element Method Course Notes. Stanford, CA.

Binh Huy Le and Zhigang Deng. 2017. Interactive Cage Generation for Mesh Deformation. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (San Francisco, California) *(I3D '17)*. Association for Computing Machinery, New York, NY, USA, Article 3, 9 pages. https://doi.org/10.1145/3023368.3023369

Chenxi Liu, Enrique Rosales, and Alla Sheffer. 2018. StrokeAggregator: Consolidating Raw Sketches into Artist-Intended Curve Drawings. *ACM Trans. Graph.* 37, 4, Article 97 (jul 2018), 15 pages. https://doi.org/10.1145/3197517.3201314

Hsueh-Ti Derek Liu, Alec Jacobson, and Keenan Crane. 2017. A Dirac Operator for Extrinsic Shape Analysis. *Computer Graphics Forum* (2017).

Yang Liu, Balakrishnan Prabhakaran, and Xiaohu Guo. 2012. Point-Based Manifold Harmonics. *IEEE Transactions on Visualization and Computer Graphics* 18, 10 (2012), 1693–1703. https://doi.org/10.1109/TVCG.2011.152

Haoran Mo, Edgar Simo-Serra, Chengying Gao, Changqing Zou, and Ruomei Wang. 2021. General Virtual Sketching Framework for Vector Line Art. *ACM Trans. Graph.* 40, 4 (July 2021), 51:1–51:14.

Adriano J. C. Moreira and Maribel Yasmina Santos. 2007. Concave hull: A k-nearest neighbours approach for the computation of the region occupied by a set of points. In *International Conference on Computer Graphics Theory and Applications*.

Jesús Nieto and Toni Susin. 2013. *Cage Based Deformations: A Survey*. Vol. 7. 75–99. https://doi.org/10.1007/978-94-007-5446-1_3

Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. 2008. Diffusion Curves: A Vector Representation for Smooth-Shaded Images. In *ACM Trans. Graph.*, Vol. 27.

Maks Ovsjanikov, Mirela Ben-Chen, Justin Solomon, Adrian Butscher, and Leonidas Guibas. 2012. Functional Maps: A Flexible Representation of Maps between Shapes. 31, 4, Article 30 (jul 2012), 11 pages. https://doi.org/10.1145/2185520.2185526

Amal Dev Parakkat, Marie-Paule R Cani, and Karan Singh. 2021. Color by numbers: Interactive structuring and vectorization of sketch imagery. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–11.

Amal Dev Parakkat, Uday Bondi Pundarikaksha, and Ramanathan Muthuganapathy. 2018. A Delaunay triangulation based approach for cleaning rough sketches. *Computers & Graphics* 74 (2018), 171–181.

Greg Philbrick and Craig S. Kaplan. 2022. A Primitive for Manual Hatching. *ACM Trans. Graph.* 41, 2, Article 21 (jan 2022), 17 pages. https://doi.org/10.1145/3503460

Ulrich Pinkall and Konrad Polthier. 1993. Computing Discrete Minimal Surfaces and Their Conjugates.

Ivan Puhachov, William Neveu, Edward Chien, and Mikhail Bessmeltsev. 2021. Keypoint-Driven Line Drawing Vectorization via PolyVector Flow. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 40, 6 (Dec. 2021). https://doi.org/10.1145/3478513.3480529

Jing Ren, Simone Melzi, Peter Wonka, and Maks Ovsjanikov. 2021. Discrete Optimization for Shape Matching. *Computer Graphics Forum* 40, 5 (2021), 81–96. https://doi.org/10.1111/cgf.14359 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14359

Paul L. Rosin. 1994. Grouping Curved Lines. In *Proc. BMVC*. 26.1–26.10. doi:10.5244/C.8.26.

Thomas W. Sederberg and Scott R. Parry. 1986. Free-Form Deformation of Solid Geometric Models. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '86)*. Association for Computing Machinery, New York, NY, USA, 151–160. https://doi.org/10.1145/15922.15903

Cloud Shao, Adrien Bousseau, Alla Sheffer, and Karan Singh. 2012. CrossShade: Shading Concept Sketches Using Cross-Section Curves. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2012)* 31, 4 (2012). http://www.crossshade.com

Nicholas Sharp and Keenan Crane. 2020. A Laplacian for Nonmanifold Triangle Meshes. *Computer Graphics Forum (SGP)* 39, 5 (2020).

Jonathan Richard Shewchuk. 1996. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geormetry, Towards Geometric Engineering, FCRC'96 Workshop, WACG'96, Philadelphia, PA, USA, May 27-28, 1996, Selected Papers (Lecture Notes in Computer Science, Vol. 1148)*, Ming C. Lin and Dinesh Manocha (Eds.). Springer, 203–222. https://doi.org/10.1007/BFb0014497

Wang Shuxia and Yu Suihuai. 2009. Endpoint fusing of freehand 3D object sketch with Hidden-part-draw. In *2009 IEEE 10th International Conference on Computer-Aided Industrial Design & Conceptual Design*. 586–590. https://doi.org/10.1109/CAIDCD.2009.5375407

Olga Sorkine and Marc Alexa. 2007. As-Rigid-As-Possible Surface Modeling. In *Geometry Processing*, Alexander Belyaev and Michael Garland (Eds.). The Eurographics Association. https://doi.org/10.2312/SGP/SGP07/109-116

Tibor Stanko, Mikhail Bessmeltsev, David Bommes, and Adrien Bousseau. 2020. Integer-Grid Sketch Simplification and Vectorization. *Computer Graphics Forum (Proc. SGP)* 39, 5 (7 2020).

Dave Pagurek Van Mossel, Chenxi Liu, Nicholas Vining, Mikhail Bessmeltsev, and Alla Sheffer. 2021. StrokeStrip: Joint Parameterization and Fitting of Stroke Clusters. *ACM Transactions on Graphics* 40, 4 (2021). https://doi.org/10.1145/3450626.3459777

Johan Wagemans, James H Elder, Michael Kubovy, Stephen E Palmer, Mary A Peterson, Manish Singh, and Rüdiger von der Heydt. 2012. A century of Gestalt psychology in visual perception: I. Perceptual grouping and figure–ground organization. *Psychological bulletin* 138, 6 (2012), 1172.

Shuxia Wang, Qian Zhang, Shouxia Wang, Xiaoke Jing, and Mantun Gao. 2020. Endpoint Fusing Method of Online Freehand-Sketched Polyhedrons. *Vis. Comput.* 36, 2 (feb 2020), 291–303. https://doi.org/10.1007/s00371-018-1608-5

Yu Wang, Mirela Ben-Chen, Iosif Polterovich, and Justin Solomon. 2018. Steklov Spectral Geometry for Extrinsic Shape Analysis. *ACM Trans. Graph.* 38, 1, Article 7 (dec 2018), 21 pages. https://doi.org/10.1145/3152156

Ron Wein, Eric Berberich, Efi Fogel, Dan Halperin, Michael Hemmer, Oren Salzman, and Baruch Zukerman. 2022. 2D Arrangements. In *CGAL User and Reference Manual* (5.5.1 ed.). CGAL Editorial Board. https://doc.cgal.org/5.5.1/Manual/packages.html#PkgArrangementOnSurface2

Brian Whited, Gioacchino Noris, Maryann Simmons, Robert W. Sumner, Markus Gross, and Jarek Rossignac. 2010. BetweenIT: An Interactive Tool for Tight Inbetweening. *Computer Graphics Forum* 29, 2 (2010), 605–614. https://doi.org/10.1111/j.1467-8659.2009.01630.x

Baoxuan Xu, William Chang, Alla Sheffer, Adrien Bousseau, James McCrae, and Karan Singh. 2014. True2Form: 3D Curve Networks from 2D Sketches via Selective Regularization. *Transactions on Graphics (Proc. SIGGRAPH 2014)* 33, 4 (2014). https://doi.org/2601097.2601128

Chuan Yan, David Vanderhaeghe, and Yotam Gingold. 2020. A Benchmark for Rough Sketch Cleanup. *ACM Transactions on Graphics (TOG)* 39, 6, Article 163 (Nov. 2020), 14 pages. https://doi.org/10.1145/3414685.3417784

Jerry Yin, Chenxi Liu, Rebecca Lin, Nicholas Vining, Helge Rhodin, and Alla Sheffer. 2022. Detecting Viewer-Perceived Intended Vector Sketch Connectivity. *ACM Transactions on Graphics* 41 (2022). Issue 4.

Song-Hai Zhang, Tao Chen, Yi-Fei Zhang, Shi-Min Hu, and Ralph R. Martin. 2009. Vectorizing Cartoon Animations. *IEEE Trans. Vis. Comput. Graph.* 15, 4 (July 2009), 618–629.

Bo Zhu, Ed Quigley, Matthew Cong, Justin Solomon, and Ronald Fedkiw. 2014. Codimensional Surface Tension Flow on Simplicial Complexes. *ACM Trans. Graph.* 33, 4, Article 111 (jul 2014), 11 pages. https://doi.org/10.1145/2601097.2601201

Haichao Zhu, Xueting Liu, Tien-Tsin Wong, and Pheng-Ann Heng. 2016. Globally Optimal Toon Tracking. *ACM Trans. Graph.* 35, 4, Article 75 (jul 2016), 10 pages. https://doi.org/10.1145/2897824.2925872

A APPENDIX

Proof. (of Proposition 3.1)
By contradiction: Assume there is an empty face R of S that intersects with Ω, i.e., $R \cap \Omega \neq \emptyset$. Then either $R \subseteq \Omega$ or $R \subsetneq \Omega$. By the definition of an empty face in S, there is an empty α-ball in R. If $R \subseteq \Omega$, this α-ball is a subset of Ω, which contradicts the sampling assumption (i.e., the artist left a gap larger than α). Therefore, there is some part of the boundary $\gamma \subseteq \partial\Omega$ that is in R. Clearly, this γ does not contain a single point of the input strokes, since γ belongs to an empty cell R. Therefore, γ must consist solely of the missing parts of the boundary that the artist did not complete, so by the sampling assumption, each connected component of γ must be contained in a α-ball. For some connected components of γ, let's take its endpoints, which will be the points of some input strokes, both outside R. Since these two points are α-connected in Ω by construction, they are also α-connected in S, so there is a path between them in S within an α-ball. The Hausdorff distance between γ and that path inside S is less or equal to α. □