Beautification of Design Sketches Using Trainable Stroke Clustering and Curve Fitting

Günay Orbay and Levent Burak Kara

Abstract—We propose a new sketch parsing and beautification method that converts digitally created design sketches into beautified line drawings. Our system uses a trainable, sequential bottom-up and top-down stroke clustering method that learns how to parse input pen strokes into groups of strokes each representing a single curve, followed by point-cloud ordering that facilitates curve fitting and smoothing. This approach enables greater conceptual freedom during visual ideation activities by allowing designers to develop their sketches using multiple, casually drawn strokes without requiring them to indicate the separation between different stroke groups. With the proposed method, raw sketches are seamlessly converted into vectorized geometric models, thus, facilitating downstream assessment and editing activities.

Index Terms—Sketch-based design, conceptual design, sketch parsing, sketch beautification, supervised stroke clustering, Laplacian Eigenmaps, curve fitting.

1 Introduction

Early design ideation and product styling activities often require intensive visual exploration, assessment, and iteration through which the product form eventually evolves [1]. During these activities, designers typically externalize and communicate their ideas by generating a multitude of conceptual sketches, which has been recognized as a critically important stage for product design and development [2], [3]. In support of these early activities, recent technological advances have led to a variety of pen-based digital design tools, which are becoming increasingly more accessible. However, a key issue that hinders a wide adoption of such tools for computer-aided design and modeling is the difficulty in transforming input pen strokes into geometric content that accurately represents designers' intentions. Due to this challenge, the majority of the current sketch-based design interfaces either leave the raw sketches unprocessed (thus, simply serving as digital drawing tools) or otherwise require users' obtrusive intervention to transform input strokes into usable geometric forms. The latter often forces designers to abandon their natural drawing styles and attend to peripheral details, which severely inhibits their conceptual freedom.

In this study, we describe a new computational method that automatically transforms conceptual design sketches into vectorized drawings using a trainable stroke clustering, point ordering, and curve fitting algorithms. From an input sketch containing an arbitrary number of strokes drawn in arbitrary directions and order, our method identifies the salient stroke groups and beautifies each into a single geometric curve. This transformation allows raw design sketches to be seamlessly converted into drawings consisting of parametric curves, thus, facilitating downstream computer-aided modeling operations. Although this task is rather trivial for humans, it is often not the case for computational systems due to the presence of many strokes exhibiting large variations in their intrinsic and extrinsic attributes such as curvature, angular orientation, intersections, and spatial proximity. The key advantage of the proposed approach is that it can learn how to group the strokes in a given sketch by studying a previously created sketch (or sketches). Hence, the system can be readily adapted for different designers and drawing styles without requiring external calibration or retuning.

At the heart of our method is a set of stroke-level geometric features that encodes and exploits the relationships between input strokes. From a manually clustered training sketch, our system trains a neural network that encodes a discriminative mapping between the features extracted from a pair of strokes, and a decision regarding whether those strokes should be grouped together. This process results in a set of raw *stroke groups* each of which may contain a single geometric curve, or a set of curves that forms locally difficult-to-detect bifurcations. These bifurcations are then detected using a set of global attributes of each group, resulting in distilled clusters each representing a single salient curve. Next, the focus shifts from the strokes to the coordinate points comprising those strokes. From the aggregate set of unorganized points extracted from each stroke cluster, our system computes a spatial ordering of those points using a spectral dimensionality reduction method. It then uses this point ordering to either generate a natural parameterization for curve fitting, or to directly synthesize a geometric curve. To enhance robustness against noninformative, exploratory strokes, our system additionally exploits the variations in pen pressure during digital sketching, thereby, generating curve fits that are most commensurate with the designer's intentions.

With the proposed approach, drawings with sketchy, overlapping strokes forming open curves, closed curves,

• *The authors are with the Department of Mechanical Engineering, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213. E-mail: gorbay@andrew.cmu.edu, lkara@cmu.edu.*

*Manuscript received 26 Aug. 2009; revised 19 Mar. 2010; accepted 12 June 2010; published online 29 July 2010.
Recommended for acceptance by J. Stam.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-2009-08-0188.
Digital Object Identifier no. 10.1109/TVCG.2010.105.*

intersecting and self-intersecting curves, and curves exhibiting subtle bifurcations can be beautified into singular geometric curves. In this work, we limit our approach to sketches that contain the *primary* drawing strokes forming the shape communicated in the scene. Hence, input sketches are assumed to be devoid of auxiliary strokes such as scaffolding lines, shading strokes, or crosshatches which are common in many professional design sketches. When desirable, however, such set of strokes can be suitably separated into different sets using the digital layering capabilities of existing drawing software.

2 Related Work

Advances in sketch-based design interfaces have led to a variety of novel technologies for 2D and 3D geometric modeling. However, techniques necessary for transforming input pen strokes into forms suitable for digital modeling operations have not been well developed. Igarashi et al. [4] introduced *beautification* in which each input stroke is converted into straight line segments using a number of visual attributes such as connections to existing segments, parallelism, perpendicularity, and other similar geometric constraints. This approach has been designed to produce purely straight line segments, thus, restricting its use to rectilinear designs. This pioneering work has coined the term *beautification* of strokes, which has since become the de facto term for converting pen strokes into usable geometric entities.

Studies [5], [6], [7], [8] proposed starting with an initial stroke and progressively improving it using modifier strokes. Baudel [5] suggested a stroke-based spline editing method, which can take as input multiple strokes to modify a single curve. Avro and Novins [6] proposed an active sketch assistance system to "classify the sketch as one of several predefined basic shapes" and to gradually morph the strokes to the identified basic shape on the fly. Bae et al. [7] suggested developing curves by converting each stroke into a cubic Bézier curve and averaging all while favoring recently drawn strokes over previously drawn ones. Kara et al. [8] used active contours to update existing curves under elastic forces exerted by the modifier strokes. Although these methods are useful in curve modification, the immediate beautification of the strokes requires users' guidance and confirmation of the results, which can disrupt the natural flow of sketching.

In studies [9], [10], researchers suggested using multiple strokes to define a single smooth curve via curve fitting. Schmidt et al. [9] used variational curves to convert the input strokes into geometric curves via curve fairing, gap filling, and smoothing operations. Kara and Shimada [10] utilized Principal Component Analysis (PCA) to determine a spatial ordering for points comprising the input strokes to facilitate B-spline curve fitting. However, similar to the previous methods, these approaches require untimely attention to curve beautification, thus disrupting fluidity during sketching.

2.1 Techniques for Drawing Simplification

Studies [11], [12], [13], [14], [15] propose extracting geometric curves from completed sketches. Saund [11] describes a drawing simplification algorithm to extract arcs in a drawing at different scales using *curve element tokens*. The method is concerned with identifying the curvilinear structures within drawings at various scales controlled by the user. Our work, on the other hand, aims to simplify hand-drawn sketches into free-form curves. Shesh and Chen [12] describe a sketch simplification method that replaces strokes sharing similar slopes and proximate end points, with an average straight line. This approach is designed for sketches composed exclusively of straight line segments. Pusch et al. [14] describe a sketch beautification method that produces the desired curve elements by first subdividing the sketch into rectangular regions until individual strokes are attained, and later using this information to construct intended curves. The subdivision-based approach makes the method more suitable for nonintersecting curves. Shesh and Chen [15] propose a drawing simplification method, which uses efficient proximity calculations to facilitate stroke grouping. The amount of simplification is determined through parameters that control stroke proximity and overlap. This approach is concerned with reducing the number of geometric elements to be processed during dynamic rendering at a local level, and thus, is not designed to identify the underlying geometric structures in a drawing. Similarly, Barla et al. [13] propose a geometric clustering and simplification method that sequentially merges curves representable by a single curve at a prescribed scale. The merging decision is based on an ε-line that has a user-specified width enveloping the candidate curves. As their method uses a greedy clustering and simplification approach, the resulting drawing is usually one of many admissible solutions. Moreover, the ε-line approach favors mergers that exhibit sharp corners, which, in fact, may be cues for different geometric structures. Likewise, it fails to identify curves that may be self-intersecting as often times such curves do not lie within an ε-line. Our approach is designed to alleviate these difficulties through a trainable stroke clustering algorithm that learns the rules for stroke grouping from the users' sketches. Additionally, our approach is designed to handle self-intersecting and bifurcating curves that are often difficult to distinguish using a purely local analysis.

2.2 Techniques for Curve Fitting to Point Sets

There exist a number of model-based methods for curve fitting to unorganized point sets [16], [17], [18]. Wang et al. [16] describe a curvature-based squared distance minimization method to fit nonself-intersecting, open/closed curves to unorganized point sets. The performance of their method is sensitive to the initialization of the model curve. Additionally, the user has to specify whether an open or closed curve is to be fit. The method proposed by Yang et al. [17] can fit self-intersecting closed curves to unorganized point sets. However, their work is tailored toward closed curves, which facilitate the use of encircling curves as the initial curve model. Both methods require a sufficiently reasonable starting curve to make them computationally affordable. Moving least squares (MLS) [19], [20] have also been used for curve and surface fitting to unorganized point sets. Although MLS methods can represent a wide variety of shapes, they are limited to manifold geometries devoid of self-intersections. In addition to curve fitting methods, methods based on direct point set parameterization have

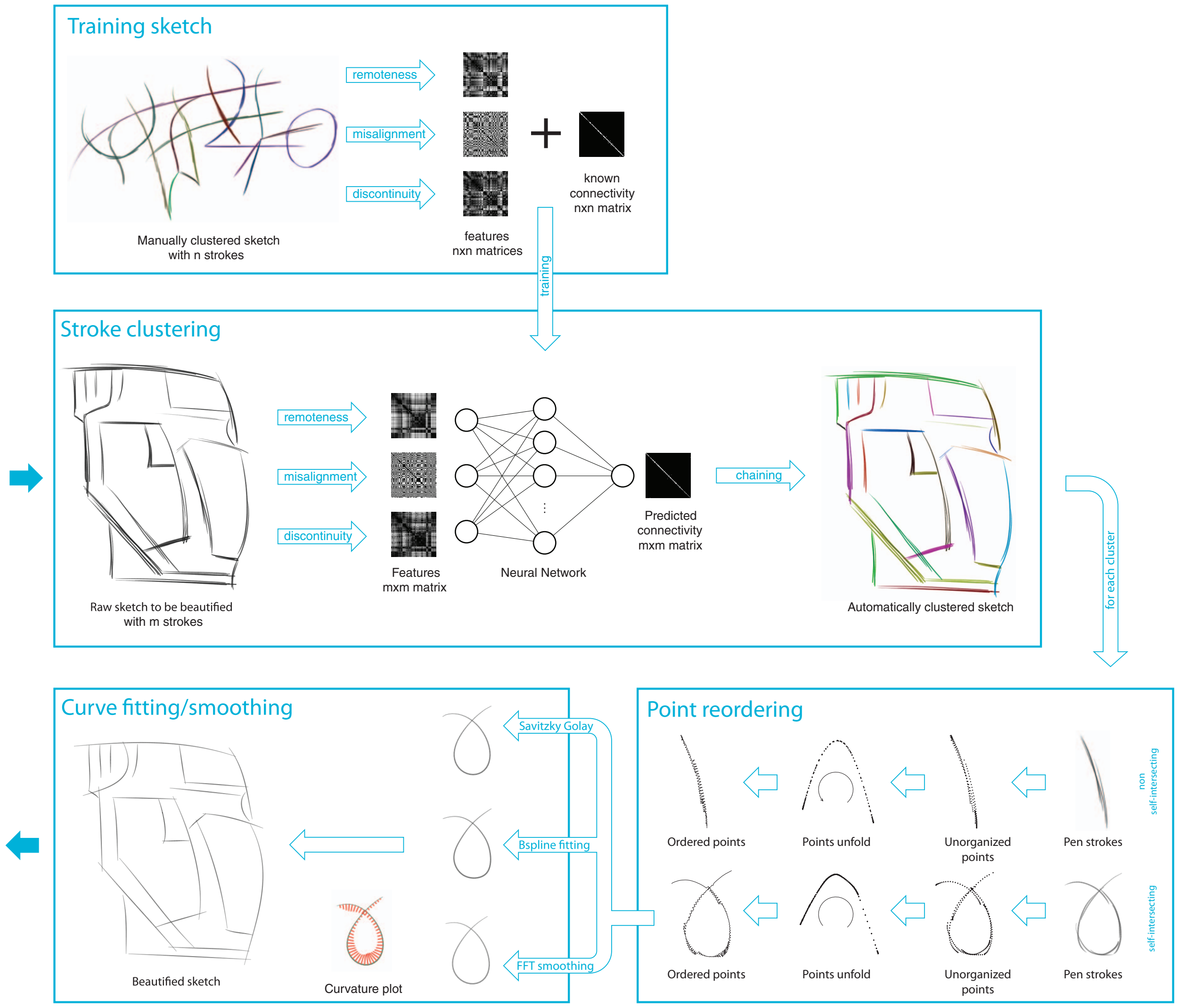


Fig. 1. Automatic beautification of design sketches: raw sketches are grouped using a neural-network-based stroke clustering method, which is trained by manually clustered sketches. Resulting stroke clusters are converted into beautified curves via point reordering and curve fitting/smoothing.

also been proposed. Goshtasby [21] proposes an image-based method to parameterize point sets while automatically determining whether the resulting geometry should represent an open or closed curve. However, it is not suitable for self-intersecting curves. In this paper, we propose a new parametrization and curve fitting method tailored toward sketch-based applications. Our approach does not require an initial curve model and automatically determines whether the final curve should be open or closed. It determines the parametrization on the point set prior to curve fitting, and uses classical fitting and smoothing methods designed for organized point sets. We also describe a method that exploits pen pressure to produce final curves that are aligned well with the users' intentions.

3 User Interaction and Overview

In this work, we propose a new sketch parsing and beautification method that converts digitally created conceptual design sketches into sketches consisting of beautified curve elements. Our system is designed to operate on a complete sketch rather than requiring the user to explicitly demarcate the separation between different curves during construction. We use a pressure-sensitive graphics tablet for recording the stylus position and tip pressure as a function of time. While the temporal order of the strokes is readily available using this setup, by design, our system does not rely on this information. This enables strokes to be drawn in an arbitrary order and allows the user to readily return to a previously developed portion of a sketch to add new strokes. Our method consists of three main steps: 1) stroke clustering, 2) point reordering, and 3) curve fitting and smoothing, and is illustrated in Fig. 1.

4 Trainable Stroke Clustering

In the first step, our system uses a trainable clustering method to group input strokes into clusters each forming a single, unique curve. At the heart of our approach is a bottom-up stroke fusion method followed by a top-down curve fission. In the initial bottom-up phase, our approach uses a trainable, neural network method that takes as input a set of geometric features extracted from each stroke pair and decides whether the two strokes should reside in the same group. Once pairwise decisions are made, our system consolidates the connected stroke pairs using a greedy linking algorithm to synthesize compound stroke groups. A compound stroke group may contain perceptually dominant stroke branches that require a global analysis beyond the pairwise decisions. Such distinct branches are often the cumulative result of small and gradual bifurcations along the raw strokes, and thus, are undetectable using a purely local analysis. For each initial compound group, the subsequent top-down analysis temporarily reverts to a point-based representation and uses a spanning tree analysis to split the group at candidate bifurcation points. A key advance in this work is the ability to detect such bifurcations, which are identified as the nodes that split the tree into similarly weighted branches. Finally, the branches identified in the point-based representation are transformed back to the stroke-based representation, thereby resulting in stroke clusters each corresponding to a branch-free, unique curve that can be subsequently beautified.

4.1 Preprocessing Input Strokes

Strokes to our system are collected through a digitizing tablet, which provides a series of data points sampled along the trajectory of the stylus. Stylus slippage on the drawing surface often results in unintended hooks at the beginning of the strokes. To alleviate the difficulties in feature extraction caused by this phenomenon, our system initially converts each incoming stroke into a cubic Bézier curve similar to [7]. A fixed number of points are then extracted along the Bézier curve to produce a densely spaced set of coordinate points approximating the original stroke. This intermediate transformation, however, is only used to expedite feature extraction in stroke clustering as described in the next section. In the subsequent curve beautification stage, our system reverts back to the original points sampled along the stylus in order to produce final curves that accurately match the designer's raw strokes. For scale independence, our system normalizes incoming sketches to a unit box while preserving their aspect ratios.

4.2 Feature Extraction

We define three geometric features to inform stroke clustering. The first feature, *remoteness*, is a measure of spatial distance between a pair of strokes. The second feature, *misalignment*, is a measure of angular distance revealing how well two strokes are aligned with one another. The third feature, *discontinuity*, measures the likelihood of one stroke forming a natural continuation of another stroke. While we have found these features to be sufficiently informative for stroke clustering, our system can be readily expanded to incorporate other geometric features if desired.

4.2.1 Remoteness

We define remoteness as the shortest distance between two strokes:

$$d_{AB} = \|\mathbf{x}_i - \mathbf{x}_j\|, \tag{1}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are the position vectors of the points resulting in the shortest distance between strokes A and B. Here, $\|\cdot\|$ is the euclidean norm operator. Fig. 2a illustrates this measure. Note that since incoming sketches are normalized to the unit box, this distance will always lie

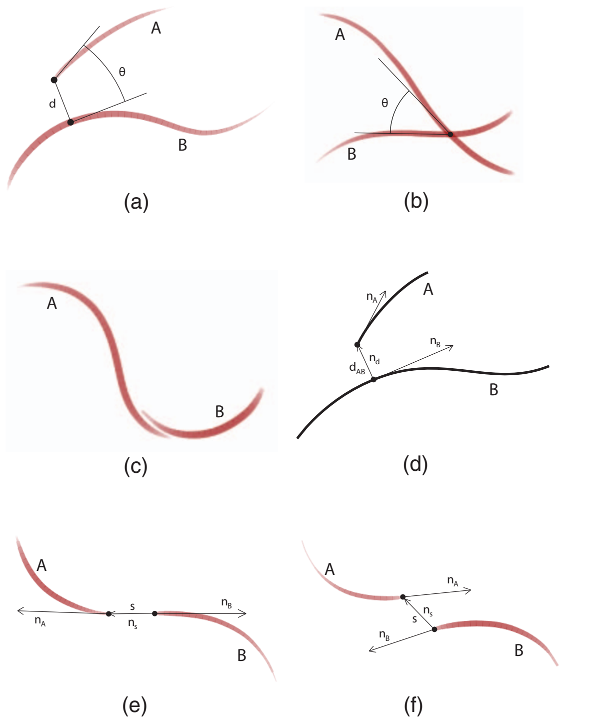


Fig. 2. Geometric features defined on a pair of strokes A and B. (a) Illustration of remoteness and misalignment features. (b) Two strokes exhibiting zero remoteness but a large angular misalignment. (c) Two well-aligned strokes exhibiting a small angular misalignment. (d) Illustration of the tangent vectors at the nearest points. (e) Two strokes exhibiting strong continuity as well as alignment (small discontinuity and misalignment). (f) Two strokes exhibiting strong alignment but weak continuity.

between $[0, 1]$. For a given pair of strokes, the remoteness will be expectedly large if this distance is large relative to the size of the scene. However, the same distance will appear smaller if the constituent strokes are made longer relative to their gap, or the sketch contains other strokes that enlarge the size of the active scene. This phenomenon is desirably commensurate with human vision, as our perception of proximity between two geometric entities will vary relative to the size of the observed scene.

4.2.2 Misalignment

Remoteness, while central to clustering, is not a sufficient measure of grouping alone as illustrated in Fig. 2b. Here, while remoteness vanishes, a significant angular misalignment suggests a separation between the strokes. Our examination of a large set of sketches and user reactions have revealed that humans tend to view two strokes to belong to the same group if the strokes are well aligned at their nearest points. Fig. 2c illustrates such a case. We quantify this observation as the misalignment between two strokes, which we formulate as the angular difference between the two strokes' tangent vectors at the points of minimum distance (Fig. 2d):

$$a_{AB} = \frac{|\angle(\mathbf{n}_A, \mathbf{n}_B)|}{\pi/2}, \tag{2}$$

where $\mathbf{n}_A$ and $\mathbf{n}_B$ are the unit tangent vectors on strokes A and B, respectively, and $\angle(\cdot,\cdot)$ maps the angle between two vectors to $[-\pi/2, +\pi/2]$.

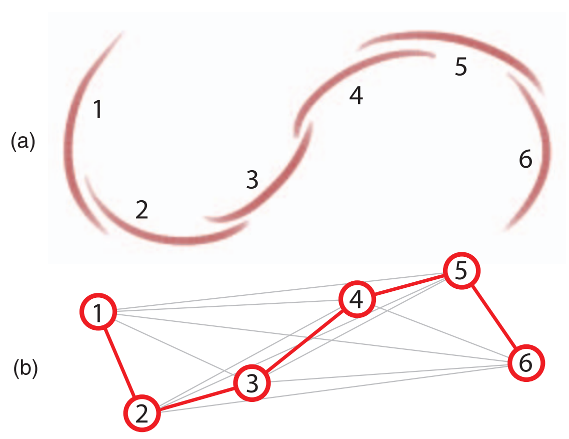


Fig. 3. (a) Six strokes belonging to the same cluster. A direct computation of the training features between distant strokes such as 1 and 6 may give rise to large feature distances. (b) Shortest path geodesic distances help adjust the original feature distances. For a given stroke pair, the geodesic path is computed separately for each type of feature.

4.2.3 Discontinuity

Small values of this feature indicate the likelihood of one stroke naturally continuing another one. Figs. 2e and 2f exhibit cases of strong continuity and weak continuity, respectively. Note that in both cases, the strokes are well aligned, with small misalignment feature values. We define discontinuity in relation to the unit tangent vectors $\mathbf{n}_A$ and $\mathbf{n}_B$ at the end points, and the distance vector s between the end points:

$$c_{AB} = \frac{\|(\mathbf{n}_A \times \mathbf{n}_B)\| + \|(\mathbf{n}_A \times \mathbf{n}_s)\| + \|(\mathbf{n}_B \times \mathbf{n}_s)\|}{3} |s|, \qquad (3)$$

where $\mathbf{n}_s$ is the normalized unit vector along distance vector s. The cross-product operator $\times$ results in vectors perpendicular to the drawing plane. When all three vectors $\mathbf{n}_A$, $\mathbf{n}_B$, and $\mathbf{n}_s$ are aligned with one another, this measure will attain a small value indicating strong continuity. Discontinuity will increase with higher misalignment between any two of the constituent vectors.

4.3 Supervised Bottom-Up Stroke Fusion

4.3.1 Training

From a manually clustered training sketch, a neural network is trained that predicts the pairwise stroke groupings. For an arbitrary pair of strokes marked to be in the same group during training, the neural network must take as input the distances produced by the three geometric features described above. As shown in Fig. 3, however, spatially separated strokes may indeed be perceptually *proximate* in the presence of neighboring strokes that bridge the two strokes. To reliably identify the distance between two arbitrary strokes, we thus take into account the circuitry that exists between the strokes in the form of a principal distance. For each of the three features, we define this distance as the total distance traveled along the shortest path connecting the two strokes (i.e., the geodesic distance between the strokes) averaged by the number of strokes visited. This distance is calculated for all three features separately. Mathematically, the training distance between the ith and jth strokes in the kth feature thus becomes:

$$df^k(i,j) = \begin{cases} \dfrac{d_G^k(i,j)}{n_G^k(i,j)} & \text{if } i,j \text{ are in same group,} \\ d^k(i,j), & \text{otherwise,} \end{cases} \qquad (4)$$

where $d_G^k(i,j)$ is the geodesic distance, $n_G^k(i,j)$ is the number of strokes traveled along the shortest path, $df^k(i,j)$ is the training distance in kth feature, and $d^k(i,j)$ is the actual distance. We calculate the geodesic distance using the Dijkstra's algorithm. Note that with this formulation, feature distances between the strokes belonging to the same cluster are attenuated, while features for the strokes in different clusters remain unchanged. Once calculated, the training distances in each feature are collected into a single vector as

$$\mathbf{df}(i,j) = [df^1(i,j) \quad df^2(i,j) \quad df^3(i,j)]^T, \qquad (5)$$

where $\mathbf{df}(i,j)$ is a feature vector composed of the training distances in features, $df^k(i,j)$s. Each feature vector is then associated with a binary class decision as

$$c(i,j) = \begin{cases} 1, & \text{if } i,j \text{ are in same group,} \\ 0, & \text{otherwise.} \end{cases} \qquad (6)$$

This adjustment of the pairwise stroke similarities through average geodesic distances is analogous to the *average with-in cluster point distances* presented in [22], which exploit the bridge points to adjust the nominal distances between point pairs.

During training, the neural network takes as input the three training distances corresponding to each of the three features described above. All stroke pairs in the sketch are utilized as training samples, thus, providing both positive and negative samples. We use a feed-forward network with a single hidden layer with 15 units. We have designed the activation functions at the hidden and output layers of the network to be hyperbolic tangent and sigmoid functions, respectively. The network is trained using a batch steepest gradient descent algorithm. The training continues until the absolute training error is reduced to less than 0.1 percent, or a prescribed number of iterations are reached. Note that for a sketch with n strokes, the number of unique training pairs is $O(n^2)$.[1] Hence, a large set of training data can often be obtained from a single sketch. If desired, multiple sketches can be used for training, thus, resulting in a large corpus of training data.

The decision boundary obtained from the NN depends on the training sketch and the users' style. As shown in Fig. 4, the decision regions can be linearly separable or inseparable. To accommodate such variations in the user style and training sketches, the NN described above is used as opposed to alternatives such as logistic regression or support vector machines. Although these classifiers can be designed to produce nonlinear decision boundaries with proper kernels, such kernels must often times be manually respecified based on the boundary, thus, requiring user intervention.

4.3.2 Stroke Group Identification

Given an unclustered input sketch, our system first calculates the feature vectors with actual distances instead, and then uses the trained neural network to determine the pairwise groupings between all stroke pairs. Example plots

1. $n(n+1)/2$ to be exact.

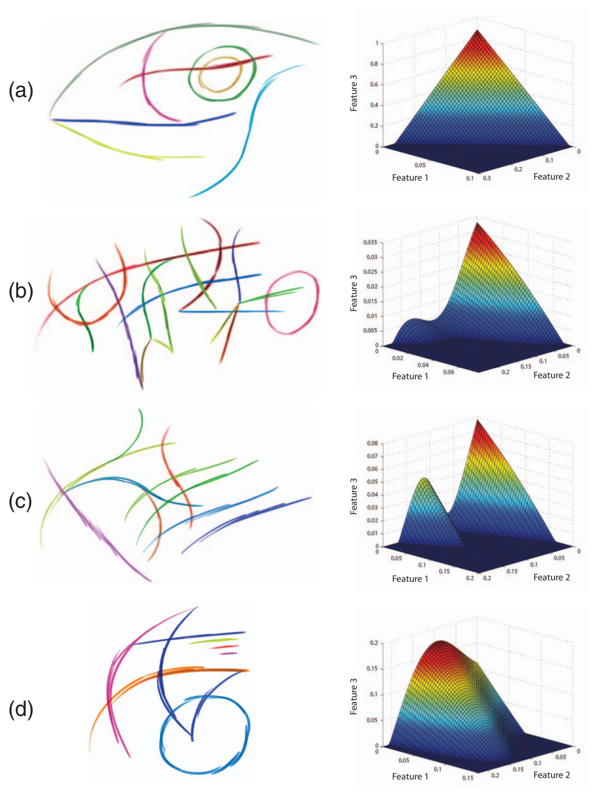


Fig. 4. Manually clustered training sketches and the resulting decision boundaries. In all cases, regions in front of the surface toward the reader belong to decision "0" (ungrouped strokes) while decision regions of "1" (grouped strokes) are behind the surface. (a) Forty-two strokes in eight clusters. (b) Forty-four strokes in 10 clusters. (c) Ninety-five strokes in 17 clusters. (d) Thirty-five strokes in nine clusters.

of the computed features and the stroke connectivities are given in Fig. 5. Given the pairwise groupings, our system uses a greedy chaining algorithm to progressively merge the strokes into the final compound stroke groups.

4.4 Top-Down Stroke Group Fission

At times, the strokes in an identified group may exhibit bifurcating branches that form more than one unique curve (e.g., Y-junctions). Such situations are often difficult to detect at a local level, and thus, necessitate a global postanalysis of each cluster. For this, our approach temporarily switches to a point-based representation of the strokes. Processing each cluster as a point cloud, our approach identifies the candidate bifurcation points and associated branches, thereby producing distilled subclusters each corresponding to a single curve.

4.4.1 Branch Splitting

This step aims to detect and split such bifurcations, resulting in the final stroke clusters each corresponding to a unique curve. We sample a fixed number of equally spaced representative points from each stroke in a stroke group.[2] This representation breaks the strokes into an unorganized point set, which facilitates a graph-theoretic analysis within the group. Specifically, we compute the minimum spanning tree (MST) [23], [24] of the point set that helps reveal the bifurcations and associated branches of the set. Note that any candidate bifurcation point must have at least three neighboring points. Additionally, bifurcations are characterized by the significance of the branches they carry relative to the rest of the MST. To reliably detect such points, we encode the forward and backward cumulative path lengths on each edge of the MST. Fig. 6 illustrates the idea. Note that for points connecting four or more edges, it suffices to have only three significant branches.

Salient bifurcation points are identified as those having a $\frac{BL_{min}}{BL_{max}} > \varepsilon$, where BL_{min} and BL_{max} are the minimum and maximum cumulative branch lengths emanating from the point. For points that join more than three branches, the above ratio is computed among the largest three branches. ε is derived from an analysis of a large corpus of training sketches that contain both branching and nonbranching stroke groups. The threshold, which is 0.05 in our implementation, is determined as the lowest possible of the above fraction that does not result in an undesired splitting of nonbranching groups. Once the bifurcation points are identified, the remaining points in the MST are grouped into distinct point groups. For n bifurcation points, this results in $1 + 2n$ such groups.

4.4.2 Selective Branch Merging

In this step, we revert to the stroke representation by assigning each of the original strokes to one of the newly identified point groups. Note that each point group may contain points belonging to different strokes. For each point group, we first identify the stroke that contributes the most number of points to that group (or multiple strokes in the case of ties). This process may leave a number of strokes that are not assigned to any particular point group. This step typically helps reveal the core of each branch far from the bifurcation points in the form of seed curves. These curves are encoded in the form of cubic Bézier curves, fit to the underlying strokes (dashed curves in Fig. 6c).

Next, the strokes forming the seed curves are further processed to identify the natural *skeleton* associated with each bifurcation point. For a bifurcation, there are two such skeletons, each formed by the trunk of the Y-junction, attached to one of the remaining two branches. Our method uses the seed curves' end point tangents to determine the natural continuations among the branches to identify the trunk, and subsequently, the two skeletons for each bifurcation point as shown in Fig. 6c.

Finally, the newly formed skeletons are used to facilitate the assignment of the remaining strokes, resulting in the final stroke clusters. To facilitate this assignment, we measure the maximum deviation of the end points of an unassigned stroke from a skeleton. Fig. 6d illustrates an example. For a given stroke, this deviation is calculated with respect to each of the two skeletons. The stroke is assigned to the skeleton that minimizes this deviation. Finally, the skeleton that possesses the most number of strokes is chosen as the salient skeleton, and the associated strokes are grouped into a single cluster. Note that, as branch merging decisions are based solely on the number of strokes in each candidate skeleton, the final dominant branch suggested by our system may not always be aligned with the user's *perceived* dominant branch. Likewise, the remaining strokes of the second skeleton are grouped into

2. In our current implementation, we have found five sampling points to provide a good compromise between stroke resolution and the speed of subsequent computations.

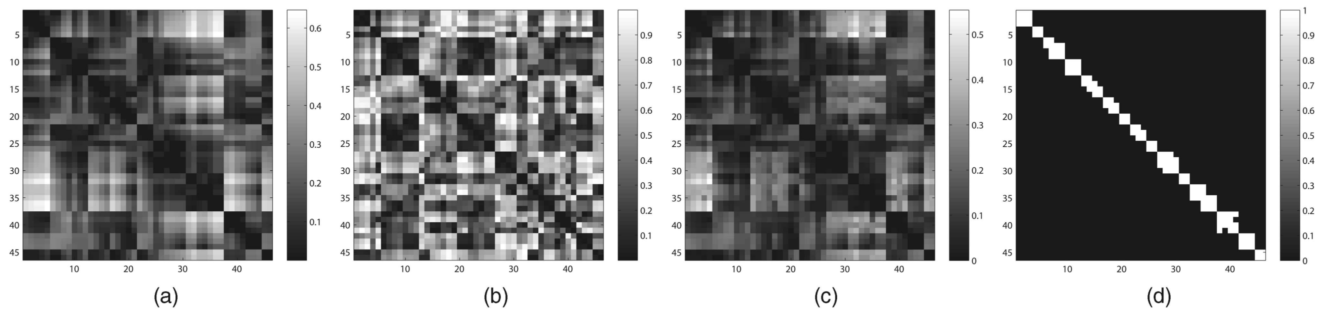


Fig. 5. Feature distances and the resulting connectivity information for a clustered sketch. (a), (b), and (c) The rows and columns correspond to stroke indexes forming square feature matrices. Color-coded matrix entries indicate distances between stroke pairs. Brighter colors in these matrices correspond to higher feature distances. (d) Resulting pairwise connectivity decisions. Brighter pixels correspond to the network outputs closer to 1, thus, revealing strokes to be grouped together. The banded nature of the final map suggests that in this particular example, proximate strokes were drawn mostly consecutively. (a) Remoteness feature. (b) Misalignment feature. (c) Discontinuity feature. (d) Connectivity.

a separate cluster, thereby, forming the perceived branching group. The same method is applied to all such bifurcation points (Fig. 6e). In the end, for n number of bifurcation points, $1+n$ final stroke clusters are generated.

Fig. 4 shows example training sketches and associated NN decision boundaries. To facilitate the stroke-pair labeling necessary for preparing the training sketches (i.e., to demarcate whether two strokes belong to the same or different groups), the user presses a key button when transitioning from one group to another. After training the system with the sketch in Fig. 4a, a testing sketch was grouped into the initial compound stroke groups through the bottom-up stroke grouping step as shown in Fig. 7a. Note that some of the stroke groups exhibit branching behavior. Next, these stroke groups are further processed during the top-down stroke group fission step forming the final stroke clusters as illustrated in Fig. 7b.

5 Point Reordering

At the end of clustering, the sketch is parsed into unique clusters of strokes each representing a single curve. In its initial form, each cluster is merely a set of unorganized coordinate points originating from the unorganized set of strokes. The next challenge is thus to prepare each cluster for curve fitting by spatially ordering the strokes' constituent points. At this point, the focus shifts to the coordinate points making up the strokes rather than the strokes themselves, as a spatial ordering of the strokes only is not sufficient to facilitate curve fitting.

Point ordering is not trivial due to a variety of challenges such as the presence of closed or folding curves, resulting in a loss of unique projection axes. Prior attempts at this problem such as Principal Component Analysis [10], active contours [25] are either limited by the complexity of the curves or are computationally demanding. In this study, we propose the use of Laplacian Eigenmaps [26] as a way to unfold input clusters in the spectral domain, and use this to inform the ordering in the canvas coordinates. Laplacian Eigenmaps is a spectral-based dimensionality reduction method used for mapping high-dimensional data onto lower dimensions similar to other dimensionality reduction methods ([27], [28]). Our approach exploits the unfolding capability of this method, without the need for any dimensionality reduction.

Before we explain the technical derivation of point ordering via Laplacian Eigenmaps, we will first draw an analogy to mechanical systems to explain the intuition behind its use. In this context, Laplacian Eigenmaps formulation is analogous to a modal vibration analysis in which the input points correspond to unit masses. Each point is connected to all other points with springs whose stiffness is inversely proportional to the distance between the point pairs. When the rigid body motion of this point set is removed, the mode shapes of the system reveal the motions of the points relative to one another at different natural frequencies. The fundamental mode, i.e., the mode with the lowest natural frequency, corresponds to the primary *flexing* motion similar to the way a simply loaded cantilever beam would deform. Thus, the monotonically increasing beam deflection helps establish the spatial ordering of the points along the beam, which is originally unknown. In our case, this idea readily applies to point sets representing a nonintersecting open curve. By considering the second mode shape, we also extend this approach to nonintersecting closed curves. Additionally, by introducing a cosine similarity factor into the pairwise affinity calculations between points, our approach resolves the ambiguity arising from self-intersecting open or closed curves as described below.

Our reordering approach is based on an affinity calculation between each point pair in the cluster. To this end, our system begins by encoding the euclidean distances between each point pair in the cluster:

$$d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|, \tag{7}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are the position vectors of the ith and jth points, and d_{ij} is the distance between the two. It then computes an estimated tangent vector (Fig. 8a) at each point using the temporal order of points within their respective strokes. Next, to help resolve self-intersecting curves, it calculates a cosine similarity factor for each point pair as:

$$\eta_{ij} = |\mathbf{t}_i \cdot \mathbf{t}_j|, \tag{8}$$

where $\mathbf{t}_i$ and $\mathbf{t}_j$ are the normalized tangents drawn at the ith and jth points, respectively. Finally, it constructs an affinity matrix by transforming the raw distances through a Gaussian kernel, and adjusting the result by the cosine similarity factor as

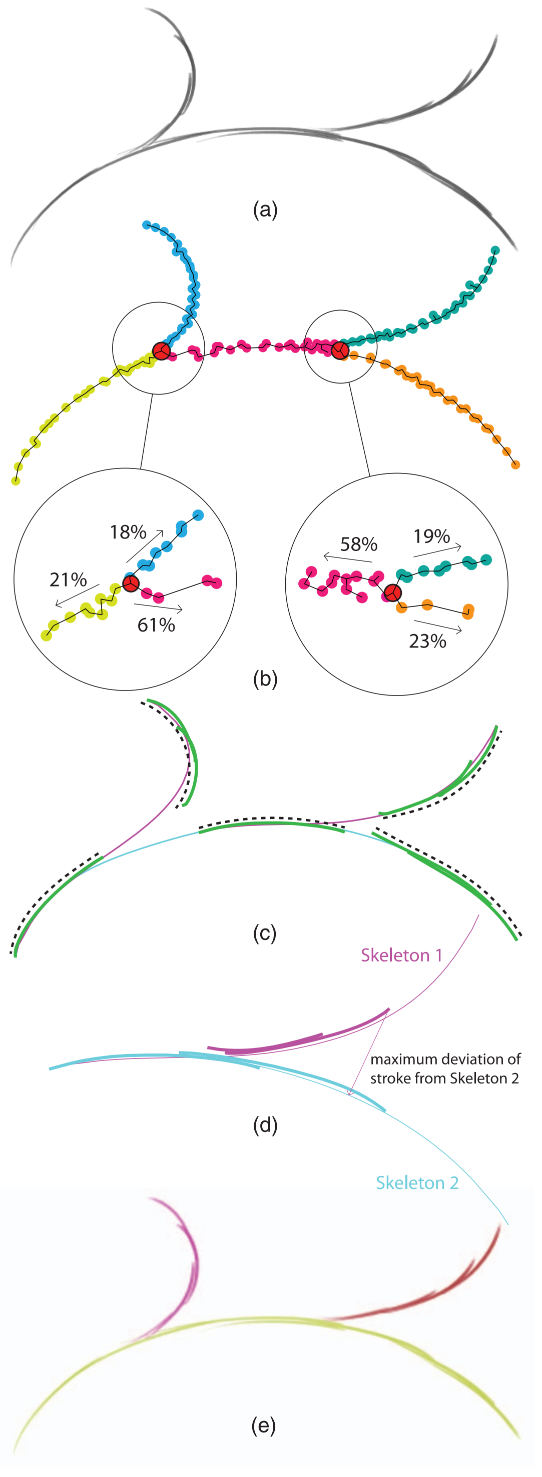


Fig. 6. (a) Strokes forming Y-junctions at two separate locations. (b) The MST identifies the candidate split points by calculating tree weights around high valence points. The point set is then split at those points. (c) The strokes that contribute most to each point branch are converted into seed curves (dashed lines). The seed curves are used to form the skeletons. (d) The remaining strokes are distributed to skeletons from which they deviate the least (i.e., purple stroke to the purple skeleton. (e) Final stroke clusters.

$$A_{ij} = \exp\left(-\frac{d_{ij}^2}{\alpha\sigma}\right)\left((1-\lambda)+\lambda\eta_{ij}\right)^{\beta}, \tag{9}$$

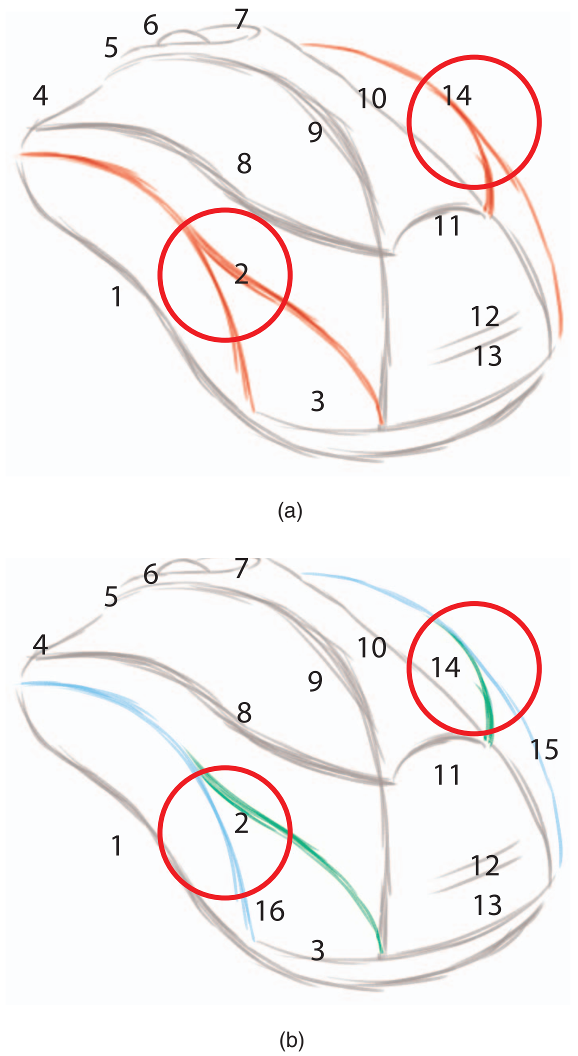


Fig. 7. (a) Stroke fusion initially generates 14 stroke groups. The sketch shown in Fig. 4a is used for training. Groups forming Y-junctions are depicted in red. (b) The final stroke clusters computed after stroke group fission. (a) A simple test case. The bottom-up step initially forms groups. These groups may exhibit branching behavior. (b) The top-down step identifies branches and splits the stroke group if needed.

where A_{ij} is the affinity between the ith and jth points, σ is the standard deviation of all the distances computed for the point set, α is a scaling factor, and λ and β are the parameters that control the amount of modification on the affinity values. We experimentally determined $\alpha = 0.5$ to result in a robust ordering for a large variety of curve shapes. This kernel helps strengthen the bonds among proximate and aligned (i.e., sharing similar tangent vectors) points while weakening those among spatially distant, or tangentially unaligned points. Fig. 8 demonstrates an example of self-intersecting curve along with the initial affinities, cosine factors, and adjusted final affinities.

Next, the affinity values under a threshold are cut off resulting in the final affinity matrix $\mathbf{W}$:

$$W_{ij} = \begin{cases} A_{ij}, & \text{if } A_{ij} > 0.75, \\ 0, & \textit{otherwise}. \end{cases} \tag{10}$$

Here, the cutoff value is also experimentally chosen to be 0.75. This final affinity matrix sets up the stage for the following eigenvalue problem:

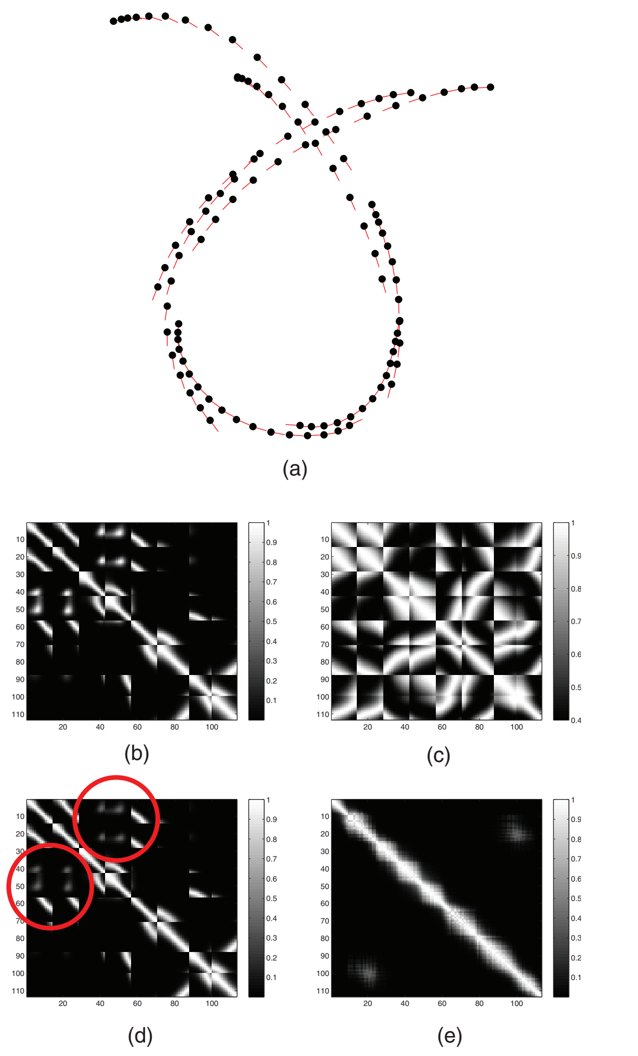


Fig. 8. (a) The constituent points of strokes forming a self-intersecting manifold. The red lines show the tangents calculated at each point using the temporal information of the host strokes. (b) The affinities using the raw euclidean distances. (c) The cosine factors. (d) The final affinities that have weaker bonds between points on intersecting branches. Note that the values demarcated with the red circles are less than the values on the initial affinities. (e) Once the bonds between intersecting branches are weaker, Laplacian Eigenmaps yields the expected point ordering.

$$\mathbf{L}f = \lambda \mathbf{D} f, \tag{11}$$

where $\mathbf{D}$ is a diagonal matrix whose diagonal elements are $D_{ii} = \sum_j w_{ij}$, and $\mathbf{L} = \mathbf{D} - \mathbf{W}$. The resulting eigenpairs, λ_ks and f_ks, are sorted using the eigenvalues in increasing order, where the first eigenpair corresponds to the pair with the smallest eigenvalue. The coordinates of the ith point in this new spectral domain become

$$\mathbf{x}_i = (f_2(i) \quad f_3(i) \quad \dots \quad f_{N+1}(i)), \tag{12}$$

where N is the dimension of the spectral domain. Note that the first eigenvector f_1 corresponding to the rigid body motion has been omitted. For dimensionality reduction, N is chosen to be less than the dimensions of the original space. In our case, we keep $\mathrm{N} = 2$ to be able to handle nonintersecting closed loops.

One straightforward approach to organize a point set is to project all points in a point cloud onto a single dimension in the spectral domain and to order them along this new dimension. With this approach, point sets representing open curves (first row of Fig. 9) can be successfully ordered along a single dimension (the horizontal axis in Fig. 9e). However, the same strategy fails when ordering point sets representing closed curves (second row of Fig. 9). In such situations, what remains informative in the spectral domain, however, are the unique polar angles of the points making up the closed curve. We exploit this property to reliably sort the point cloud, and later transform this information to the canvas coordinates.

Point sets that form self-intersecting curves require different amounts of modification on the affinity values, while point sets that define nonself-intersecting curves can be readily ordered without modification. The amount of required modification for self-intersecting curves is strongly influenced by the intersection angles between overlapping branches. To achieve a proper separation between intersecting branches, a limited range of different λ and β values are used to order a point set. For each pair of λ and β values, the resulting ordering is analyzed and a measure of disparity in chord lengths between the consecutive points is calculated as $\sigma_{chord}/median_{chord}$. Here, σ_{chord} is the standard deviation and $median_{chord}$ is the median of chord lengths between consecutive points. The set of parameters that minimizes the disparity is picked to be the set that results in the final ordering. Fig. 9 shows the final point ordering of open and closed self-intersecting curves both drawn with multiple strokes.

In cases where low sampling rate input devices are used, significantly large gaps between consecutively sampled points can make the spatial ordering appear ambiguous even to the human eye. As illustrated in Figs. 10a and 10b, the original point set is not sufficient to reveal whether the curve is u shaped or gamma shaped. However, as shown in Fig. 10a, the strokes leading to the point set help our perception by resolving the ambiguity. To mimic a similar visual cue, we artificially upsample input strokes (using a linear operator) to a level where the gaps between consecutive data points are below a certain range. As the gaps between consecutive points decrease, the system becomes increasingly more accurate in its ordering of the original points as the newly introduced points serve as connecting bridges. To this end, our system first introduces the artificial points that eliminate large gaps, then orders the cumulative set of points in the spectral domain, and finally, dismisses the artificially introduced points.

6 Curve Beautification

In the last step, the reordered point sets are beautified by curve fitting and smoothing. In curve fitting, our system uses the previously determined parameterization to replace the point set with a B-spline curve that minimizes the sum of squared distance errors. Unlike conventional curve fitting methods, our system additionally monitors the differences in pen pressure along the strokes to improve the computed curve fits. This information helps enhance robustness to noninformative strokes, thereby producing results commensurate with the users' expectations. Once generated, the user can further improve the resulting curves using Fourier Transform smoothing [29] or Savitzky-Golay smoothing

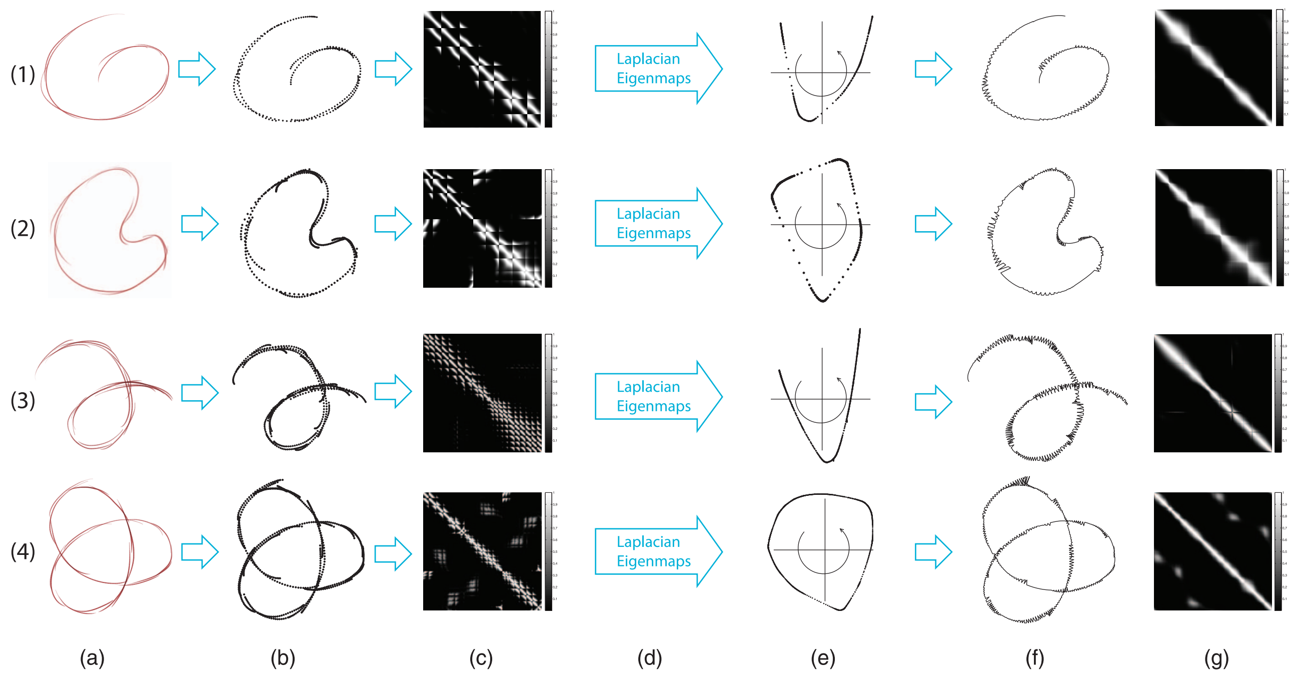


Fig. 9. Point reordering in stroke clusters. Rows illustrate (1) nonself-intersecting open, (2) nonself-intersecting closed, (3) self-intersecting open, and (4) self-intersecting closed stroke groups. (a) Original strokes. (b) Constituent points forming an unorganized point set. (c) Affinity matrices. (d) Laplacian Eigenmaps transforms the points into spectral domain. (e) Points in the spectral domain are reordered using their unique polar angles. (f) The resulting ordering is then used to order the original coordinate points. (g) Note that the resulting ordering produces a banded structure on the affinity matrices. Examples on rows 1, 2, 3, and 4 have 7, 10, 19, and 21 strokes, respectively.

[30]. Alternatively, both types of smoothing operations can be used directly on the set of ordered points to generate the final curves, without the need for parametric curve fitting.

To this end, our system first uses a technique similar to moving least squares [31] to correct any ordering defects and to determine a natural parameterization to be used for curve fitting. Starting from one end of the point set, a locally quadratic curve is fit to a fixed number of points as shown in Fig. 11a. Each original point (x_p, y_p) is then projected vertically onto the curve along the local coordinate frame resulting in $(x_p, P(x_p))$. Next, the projection point is slid along the local tangent until the distance to the original point is minimized resulting (x_e, y_e). This provides a first order estimation of the normal projection of (x_p, y_p) onto the quadratic curve. Finally, a parameterization is calculated from the distances between the projected points as shown in Fig. 11b. The point set is rectified if the new parameterization suggests a point ordering different from the original one. The same process is repeated multiple times over the unprocessed coordinate points until the end of the point set is reached.

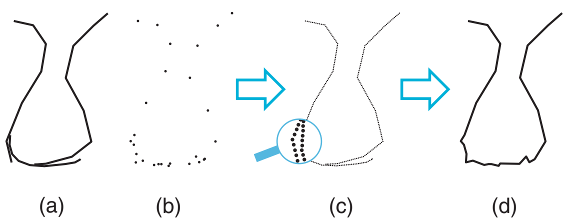


Fig. 10. (a) Resampling strokes in case of strokes with low sampling rate. (b) The sampled points are highly sparse resulting in ambiguous point ordering. (c) The upsampled point set. (d) Resulting ordering depicted in the form a polyline.

Given the above parameterization, the user may subsequently elect to fit a cubic B-spline to the entire point set when desired. The number of spline control points can be chosen by the user. For B-spline fitting, a commonly used approach is to minimize an error function defined between the original points and the curve points:

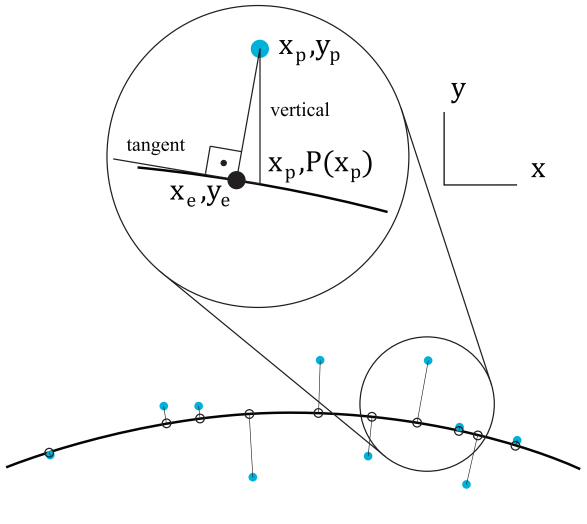
$$E = \frac{1}{2}\sum_{k=1}^{K} \|\mathbf{P}(u_k) - \mathbf{X}_k\|^2, \qquad (13)$$

where $\mathbf{X}_k$ is the original position vector of the kth point and $\mathbf{P}(u_k)$ is the position vector of the corresponding point on the curve with parameter u_k.

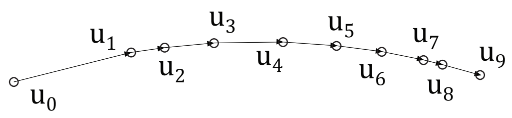
We have found that a direct application of this approach often produces fits that are insensitive to the user's drawing mechanics. Specifically, we have identified that the stylus pressure is strongly correlated to the regions of high emphasis, and thus, needs to be accounted for during curve fitting. To this end, we employ a modified error function that explicitly incorporates the stylus pressure:

$$E = \frac{1}{2}\sum_{k=1}^{K} p_k \|\mathbf{P}(u_k) - \mathbf{X}_k\|^2, \qquad (14)$$

where p_k is the pen pressure of kth point. In the above error expression, high pressure points have a greater influence on the cumulative error. Fig. 12 illustrates the idea. Fig. 12a shows the input strokes and the resulting point ordering obtained from Laplacian Eigenmaps. Consecutive points are linked by line segments. Fig. 12b shows the initial cubic B-spline fitting that uses the trivial chord length parameterization [32]. Fig. 12c shows a similar fit when the initial ordering and the subsequent parameterization is updated using the



(a)



(b)

Fig. 11. Quadratic curve fitting for order correction and chord length estimation. (a) An original point x_p, y_p is projected parallel to the y-axis of the local coordinate frame. The new updated point location x_e, y_e is found by projecting the original point x_p, y_p vertically onto the tangent direction defined by the initial projection. (b) Curve parameterization is computed after the point placements. This often leads to a better parameterization than the original chord length parameterization applied to the original point set.

local quadratic curve fitting and point projection algorithm described in the previous section. Note that while the fit has improved, it remains too sensitive to overstroking and divergent stroke extensions. Fig. 12d shows the final fit when the curve fit is informed by the pen pressure.

Alternatively, the user may elect to retain the ordered set of points in each stroke cluster as a connected polyline similar to the way it appears in Fig. 12a. In such situations, the user may enhance the originally jagged curves using Fourier Transform smoothing [29] or Savitzky-Golay smoothing [30].

7 Complexity Analysis

The computational cost of our approach is primarily dictated by the following two steps: Let $n_s = \text{total number}$ of strokes in the sketch, $n_p = \text{maximum number}$ of original sampled points for each stroke, and $n_r = \text{number}$ of resampled points along the cubic Bézier curve fitted to each stroke.

7.1 Stroke Clustering

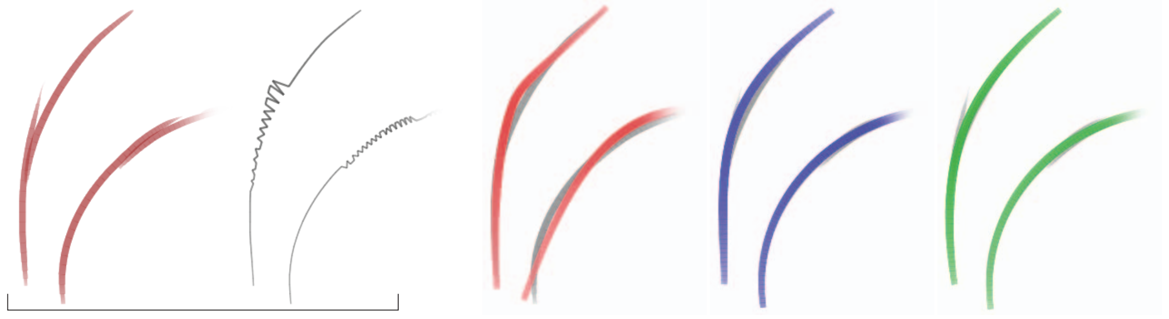
The geometric feature extraction between each stroke pair requires $O(n_r^2)$ distance calculations applied to a total of $O(n_s^2)$ stroke pairs. Once pairwise features are determined, the neural network takes as input a total of $O(n_s^2)$ feature vectors to identify the binary stroke connectivity decisions. Note that depending on the size of the sketch, either n_s or n_r may dominate these computations. Other auxiliary operations in this step such as stroke resampling $(O(n_s \cdot (n_r + n_p)))$ and stroke linking $(O(n_s)$ logic operations) cost significantly less.

The top-down stroke group fission step solves the MST problem over a reduced number of points. The computational complexity of the MST problem is $O(m)$, where m is the number of edges. Since we extract a reduced number of points from each stroke, with a fully connected tree (i.e., for n_s number of strokes in a group, the number of edges is $m = n_s^2$), the computational complexity is $O(n_s^2)$.

7.2 Point Set Ordering

Once unique stroke clusters are obtained, we determine a point ordering within each cluster to facilitate curve fitting. For point reordering, we define $n_c = \text{total number}$ of identified clusters, and $n_v = \text{number}$ of points within each cluster. Note that n_v will be such that $n_s \cdot n_p \equiv n_c \cdot n_v$, where each side represents the total number of sampled points in the sketch.

For each cluster, $O(n_v^2)$ computations are performed to construct the affinity matrix A. Next, the eigenvalue problem is solved that requires an additional $O(n_v^2)$ operations. Finally, a worst-case $O(n_v^2)$ logic operations are required for point sorting. This results in a total point ordering cost of $O(n_c \cdot n_v^2)$. In cases involving self-intersecting curves, the



(a) (b) (c) (d)

Fig. 12. Curve fitting examples. (a) The original strokes and the initial ordering. (b) The result of B-spline fitting with chord length parameterization. Note that the jagged regions cause a weak curve fit. (c) The rectified parameterization suppresses the influence of the jagged regions but the curve is insensitive to pressure variations. (d) The pressure-based weighted error expression results in improved curve fits.

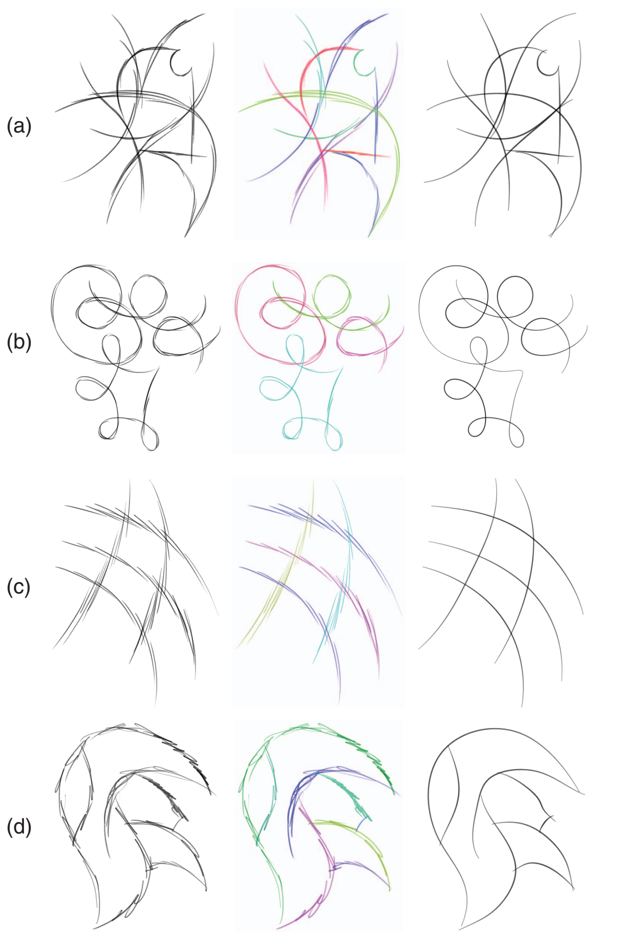


Fig. 13. Sketches illustrating the key capabilities of our approach. (a) Y-junctions and overlapping strokes. Fifty-seven strokes in 11 clusters. (b) Self-intersecting curves. Eighty-seven strokes in four clusters. (c) Overlapping and parallel strokes. Forty-seven strokes in five clusters. (d) Wiggly input strokes. Forty-four strokes in eight clusters.

computational cost increases to a constant multiple of the previous cost.

Note that the worst-case scenario for point set ordering occurs when the sketch consists of a single stroke cluster. In such situations, while n_c becomes unity, n_v attains its maximum, thus, significantly impacting the cost. The next section presents typical runtimes on example sketches.

8 Examples and Discussions

In this section, we first demonstrate the key capabilities of our approach through several exemplar sketches. Second, we discuss the effect of the variability in the sketching style (as learned by the training algorithm) on the performance of the method. Finally, the overall performance of the method is evaluated using several design sketches.

The following examples are best viewed and studied electronically using appropriate magnification functions of the document viewers.

8.1 Clustering and Beautification

Fig. 13 shows successfully clustered and beautified sketches, each emphasizing a key aspect of our approach. In all examples, the sketch shown in Fig. 1 is used for training. Fig. 13a shows the identification of salient curves

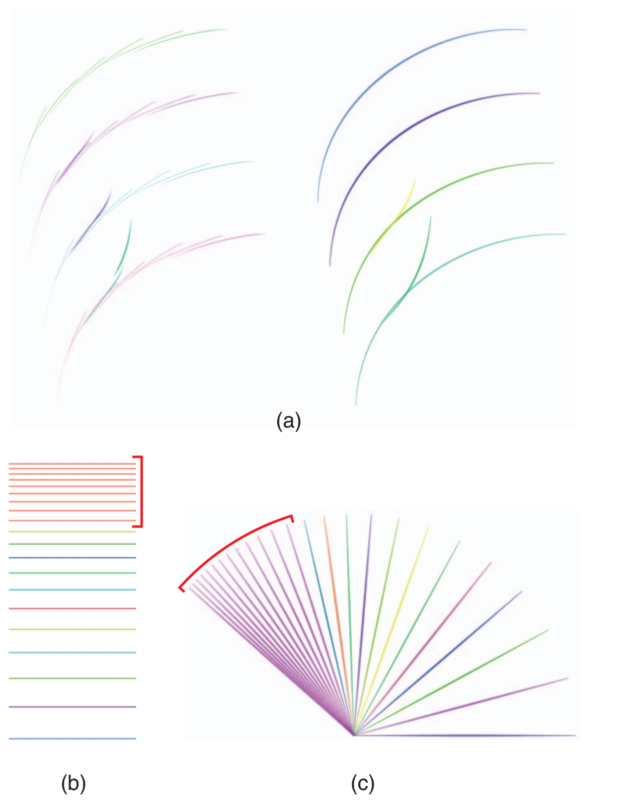


Fig. 14. Illustration of the clustering decisions for (a) Y-junctions, (b) parallel configurations with varying spatial distances, (c) coaxial configurations with varying angular distances. In all cases, the sketch shown in Fig. 1 is used for training.

in the presence of numerous overlapping strokes exhibiting Y-junctions. Fig. 13b illustrates sketch beautification in the presence of self-intersecting and overlapping curves. Fig. 13c illustrates the beautification of a sketch containing moderately disjoint and sketchy strokes. Finally, Fig. 13d shows the clustering and beautification of highly wiggly strokes. Note that since the final beautified curves are fit to the original stroke points following point ordering, the resulting curves are not undesirably influenced by the temporary Bézier curve fits to the individual strokes during the early stages of stroke clustering.

Fig. 14a demonstrates the Y-junction identifications using the configuration presented in Section 4.4. The sketches on the left exhibit a gradually more prominent branching. The first two sketches are deemed to form singular curves, whereas bifurcations are detected for the latter two.

Figs. 14b and 14c illustrate the spatial and angular separation bounds of our clustering algorithm using synthetically generated line segments sampled at 25 points. The red lines show the strokes clustered together. Since the separations increase monotonically, the widest gaps in the clustered strokes correspond to the largest admissible separation for the with-in cluster strokes. Note that these bounds implicitly emerge during training.

8.2 User Style Variability

The nature of the training sketch has a direct influence on the resulting clustering decisions. This allows our approach to be customizable to individual users or particular drawing styles. Figs. 15a and 15b show training sketches

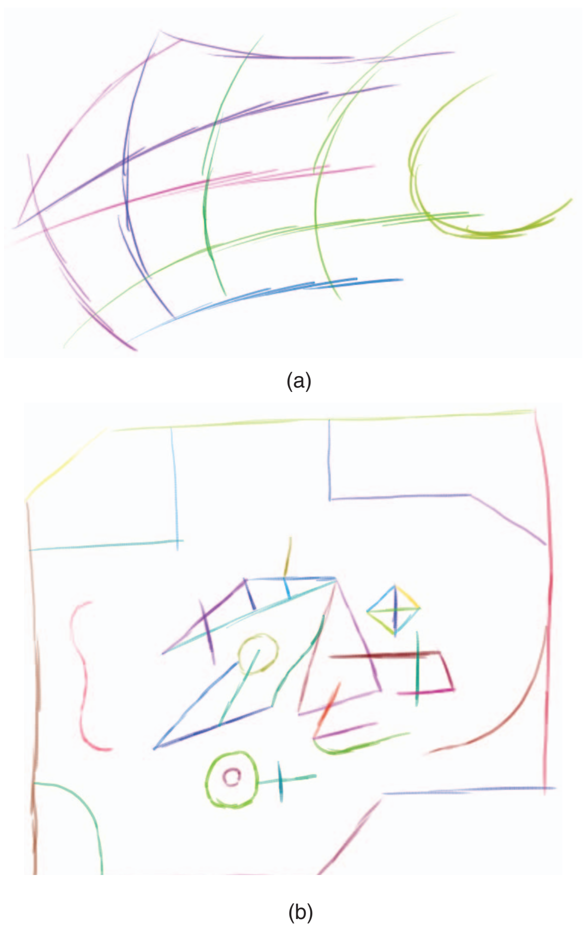


Fig. 15. Training sketches drawn by two different users. (a) Training sketch drawn by User 1. Fifty-one strokes in 11 clusters. (b) Training sketch drawn by User 2. Two hundred twenty-five strokes in 47 clusters.

drawn by two different users. The first user has a sketchier and more casual drawing style, while the second user's strokes are cleaner and drawn more carefully. The NNs trained with these sketches are used to cluster the sketches drawn by the two users shown in Fig. 16. Erroneous clusterings are demarcated in red circles. As shown, most errors occur when the training and test sketches belong to different users. While some errors produce acceptable final results, certain errors significantly impact the resulting beautification. The same figure shows the clustering performance of our approach on each test sketch, measured in terms of the percentage of all misclustered strokes within the sketch. The ground truth clustering has been determined by the authors. Misclusterings are identified in relation to the ground truth data, and consist of the strokes in all the erroneous splits and mergers.

Note that the first user's training sketch results in larger admissible gaps among the strokes within a cluster, thus, resulting in a more aggressive clustering. The second user's training sketch enables a finer separation between different clusters, thus, allowing various details to be successfully beautified. Our observations suggest that while sketchier training examples help accommodate a wider variety of drawing styles, it may nonetheless fail to capture the details in carefully drawn sketches. Regardless of the drawing style, however, the overall performance of our approach expectedly increases when the training sketches contain a sufficiently rich set of stroke configurations that are also observed in the test sketches.

Our analysis of the individual processes that produce the results in Fig. 16 has shown that the errors are primarily due to either 1) the bottom-up feature distance calculations erroneously grouping strokes that were not supposed to be

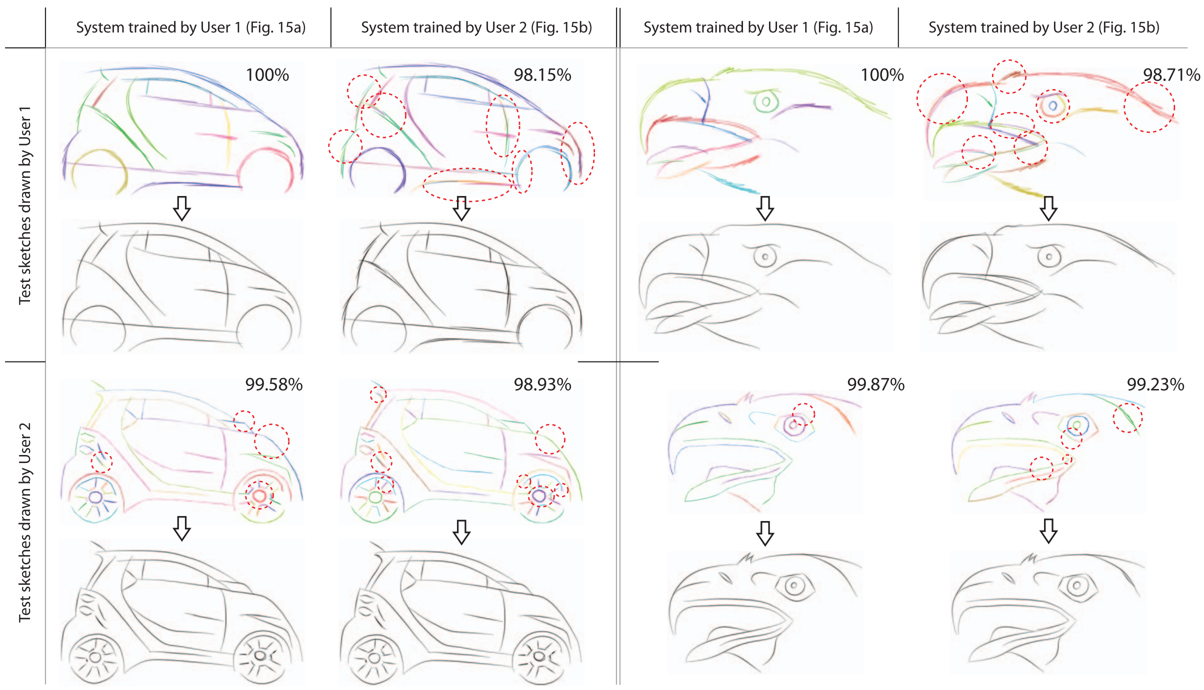

System trained by User 1 (Fig. 15a)
System trained by User 2 (Fig. 15b)
System trained by User 1 (Fig. 15a)
System trained by User 2 (Fig. 15b)
Test sketches drawn by User 1
Test sketches drawn by User 2
100%
98.15%
100%
98.71%
99.58%
98.93%
99.87%
99.23%

Fig. 16. Clustering (colored) and beautification (black and white) results of the test sketches drawn by the two users. The sketches in the first and third columns and the sketches in the second and fourth columns are clustered using the training sketches from User 1 and User 2, respectively (sketches shown in Figs. 15a and 15b). Upper and lower rows are sketches drawn by User 1 and User 2, respectively. Stroke-based clustering performance is also shown for each test sketch. Erroneous splits and mergers are counted as misclusterings, which are demarcated in dashed red circles.

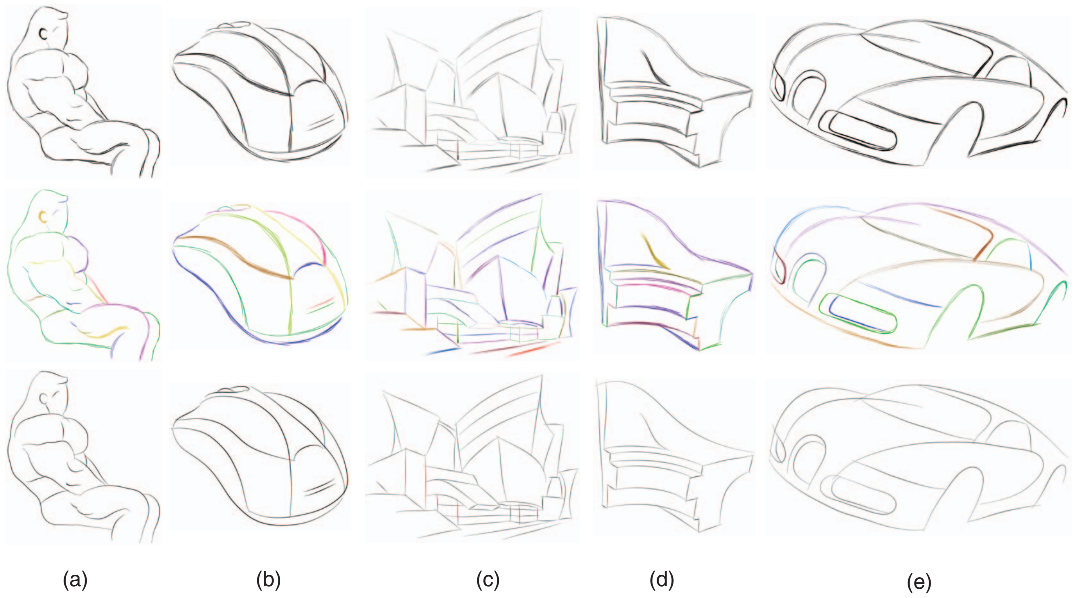


Fig. 17. Successfully clustered and beautified sketches. (a) Figure sketch. (b) PC mouse sketch. (c) Architectural sketch. (d) Mechanical part sketch. (e) Car sketch. The first row shows the raw sketches, the second row shows the clustered strokes, and the third row shows the beautified final sketch.

grouped or 2) the top-down fission process splitting originally valid stroke groups. The first type of error occurs when the feature distances between a pair of strokes remain below the decision boundary learned by our Neural Network classifier. Such errors expectedly diminish as the stroke styles in the test sketches match closer to those in the training sketches. In the latter case, superfluous bifurcations typically result in such splitting. We have found this issue to be more prominent with stroke groups that are smaller in size (and consist of shorter strokes) relative to the rest of the sketches, as the otherwise discernible bifurcations in such groups are suppressed by their absolute sizes.

8.3 Case Studies

Figs. 17a, 17b, 17c, 17d, and 17e show various design sketches. For all sketches, the sketch shown in Fig. 1 is used for training. The number of strokes, the number of clusters, and the computational performance statistics are listed in Table 1. In all cases, a 2.5 GHz computer with 3 GB of RAM is used. The major components of our system are deployed in Matlab, which is integrated with a sketch-based interface written in C++ and OpenGL that facilitate user interaction and stylus input.

9 Conclusion

We describe a new computational method to convert digitally created conceptual sketches into vector drawings consisting of beautified curve segments. We propose a supervised stroke clustering algorithm that learns meaningful stroke clusters by studying the geometric relationships between the strokes of a training sketch. Our approach uses a bottom-up stroke fusion combined with a top-down stroke fission method to reliably identify the intended unique curves. A key advantage of the proposed system is its ability to adapt to different drawing styles, as well as its ability to beautify Y-junctions, closed loops, and self-intersections. This work also introduces a new point set ordering algorithm based on Laplacian Eigenmaps that is applicable to both self-intersecting, as well as open or closed curves. Finally, we describe a pressure-sensitive curve fitting method capable of suppressing the exploratory pen strokes and stroke artifacts on the final curve fits. Our tests have shown the validity of the proposed approach on a variety of design sketches. Currently, our approach aims to isolate and beautify the individual stroke clusters, while remaining oblivious to higher level geometric relationships that may exist among these clusters. Our future goals include a more comprehensive analysis of these clusters that will leverage a richer set of modeling operations such as automated trimming, constraint identification, curve blending, and curve modification.

TABLE 1
Sketch Specifications and Computational Performance Statistics for the Sketches in Fig. 17

	a	b	c	d	e
# of strokes	184	137	345	178	337
# of clusters	28	15	66	29	19
$t_{fusion}(s)$	13.8	7.34	47.0	12.5	44.5
$t_{fission}(s)$	0.97	0.77	1.27	0.45	1.77
$t_{ordering}(s)$	1.42	1.03	2.31	0.98	1.07

Acknowledgments

The authors created all the sketches in the paper. The design examples are courtesy of their owners.

References

[1] K.T. Ulrich and S.D. Eppinger, *Product Design and Development*, fourth ed. McGraw-Hill, 2008.

[2] D.G. Ullman, S. Wood, and D. Craig, "The Importance of Drawing in the Mechanical Design Process," *Computers & Graphics*, vol. 14, no. 2, pp. 263-274, 1990.

[3] I. Verstijnen, C. Van Leeuwen, G. Goldschmidt, R. Hamel, and J. Hennessey, "Sketching and Creative Discovery," *Design Studies*, vol. 19, no. 4, pp. 519-546, 1998.

[4] T. Igarashi, S. Matsuoka, S. Kawachiya, and H. Tanaka, "Interactive Beautification: A Technique for Rapid Geometric Design," *Proc. 10th Ann. ACM Symp. User Interface Software and Technology (UIST '97),* pp. 105-114, 1997.

[5] T. Baudel, "A Mark-Based Interaction Paradigm for Free-Hand Drawing," *Proc. Seventh Ann. ACM Symp. User Interface Software and Technology,* pp. 185-192, 1994.

[6] J. Arvo and K. Novins, "Fluid Sketches: Continuous Recognition and Morphing of Simple Hand-Drawn Shapes," *Proc. 13th Ann. ACM Symp. User Interface Software and Technology,* pp. 73-80, 2000.

[7] S.-H. Bae, W.-S. Kim, and E.-S. Kwon, "Digital Styling for Designers: Sketch Emulation in Computer Environment," *Proc. Int'l Conf. Computational Science and Its Applications (ICCSA '03),* p. 987, 2003.

[8] L.B. Kara, C.M. D'Eramo, and K. Shimada, "Pen-Based Styling Design of 3D Geometry Using Concept Sketches and Template Models," *Proc. 2006 ACM Symp. Solid and Physical Modeling (SPM '06),* pp. 149-160, 2006.

[9] R. Schmidt, B. Wyvill, M.C. Sousa, and J.A. Jorge, "Shapeshop: Sketch-Based Solid Modeling with Blobtrees," *Proc. Second Eurographics Workshop Sketch-Based Interfaces and Modeling,* pp. 53-62, 2005.

[10] L.B. Kara and K. Shimada, "Sketch-Based 3D-Shape Creation for Industrial Styling Design," *IEEE Computer Graphics and Applications,* vol. 27, no. 1, pp. 60-71, Jan. 2007.

[11] E. Saund, "Labeling of Curvilinear Structure across Scales by Token Grouping," *Proc. IEEE CS Conf. Computer Vision and Pattern Recognition,* pp. 257-263, 1992.

[12] A. Shesh and B. Chen, "Smartpaper: An Interactive and User Friendly Sketching System," *Proc. Conf. Eurographics,* vol. 23, no. 3, 2004.

[13] P. Barla, J. Thollot, and F. Sillion, "Geometric Clustering for Line Drawing Simplification," *Proc. Eurographics Symp. Rendering,* 2005.

[14] R. Pusch, F. Samavati, A. Nasri, and B. Wyvill, "Improving the Sketch-Based Interface: Forming Curves from Many Small Strokes," *The Visual Computer,* vol. 23, no. 9, pp. 955-962, 2007.

[15] A. Shesh and B. Chen, "Efficient and Dynamic Simplification of Line Drawings," *Computer Graphics Forum,* vol. 27, no. 2, pp. 537-545, 2008.

[16] W. Wang, H. Pottmann, and Y. Liu, "Fitting B-Spline Curves to Point Clouds by Curvature-Based Squared Distance Minimization," *ACM Trans. Graphics,* vol. 25, no. 2, pp. 214-238, 2006.

[17] Z. Yang, J. Deng, and F. Chen, "Fitting Unorganized Point Clouds with Active Implicit B-Spline Curves," *The Visual Computer,* vol. 21, no. 8, pp. 831-839, 2005.

[18] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active Contour Models," *Int'l J. Computer Vision,* vol. 1, no. 4, pp. 321-331, 1988.

[19] D. Shepard, "A Two-Dimensional Interpolation Function for Irregularly-Spaced Data," *Proc. 1968 23rd ACM Nat'l Conf.,* pp. 517-524, 1968.

[20] P. Lancaster and K. Salkauskas, "Surfaces Generated by Moving Least Squares Methods," *Math. Computation,* vol. 37, no. 155, pp. 141-158, 1981.

[21] A. Goshtasby, "Grouping and Parameterizing Irregularly Spaced Points for Curve Fitting," *ACM Trans. Graphics,* vol. 19, no. 3, pp. 185-203, 2000.

[22] C. Zahn, "Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters," *IEEE Trans. Computers,* vol. 20, no. 1, pp. 68-86, Jan. 1971.

[23] J. Kruskal, Jr., "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem," *Proc. Am. Math. Soc.,* vol. 7, no. 1, pp. 48-50, 1956.

[24] R. Prim, "Shortest Connection Networks and Some Generalizations," *Bell System Technical J.,* vol. 36, no. 6, pp. 1389-1401, 1957.

[25] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active Contour Models," *Int'l J. Computer Vision,* vol. 1, no. 4, pp. 321-331, 1988.

[26] M. Belkin and P. Niyogi, "Laplacian Eigenmaps for Dimensionality Reduction and Data Representation," *Neural Computing,* vol. 15, no. 6, pp. 1373-1396, 2003.

[27] S. Roweis and L. Saul, "Nonlinear Dimensionality Reduction by Locally Linear Embedding," *Science,* vol. 290, no. 5500, pp. 2323-2326, 2000.

[28] J. Tenenbaum, V. Silva, and J. Langford, "A Global Geometric Framework for Nonlinear Dimensionality Reduction," *Science,* vol. 290, no. 5500, pp. 2319-2323, 2000.

[29] C. Zahn and R. Roskies, "Fourier Descriptors for Plane Closed Curves," *IEEE Trans. Computers,* vol. 21, no. 3, pp. 269-281, Mar. 1972.

[30] A. Savitzky and M.J.E. Golay, "Smoothing and Differentiation of Data by Simplified Least Squares Procedures," *Analytical Chemistry,* vol. 36, pp. 1627-1639, 1964.

[31] I. Lee, "Curve Reconstruction from Unorganized Points," *Computer Aided Geometric Design,* vol. 17, no. 2, pp. 161-177, 2000.

[32] L. Peigl and W. Tiller, *The NURBS Book.* Springer, 1995.

Günay Orbay received the BS and MS degrees in mechanical engineering from the Middle East Technical University. He is currently working toward the PhD degree in the Department of Mechanical Engineering at Carnegie Mellon University. His research interests include pen-based user interfaces, intelligent assistance in conceptual design, human-computer interaction, curve and surface design, and computational aesthetics and styling.

Levent Burak Kara received the BS degree from the Middle East Technical University and the MS and PhD degrees from Carnegie Mellon University, all in mechanical engineering. He is an assistant professor in the Department of Mechanical Engineering at Carnegie Mellon University. His research interests include computational design, geometric modeling, sketch-based interfaces, human-computer interaction, and artificial intelligence.