Topology-Driven Vectorization of Clean Line Drawings

Gioacchino Noris[1,2] Alexander Hornung[2] Robert W. Sumner[2] Maryann Simmons[3] Markus Gross[1,2]
[1]ETH Zurich [2]Disney Research Zurich [3]Walt Disney Animation Studios

Abstract

Vectorization provides a link between raster scans of pencil-and-paper drawings and modern digital processing algorithms that require accurate vector representations. Even when input drawings are comprised of clean, crisp lines, inherent ambiguities near junctions make vectorization deceptively difficult. As a consequence, current vectorization approaches often fail to faithfully capture the junctions of drawn strokes. We propose a vectorization algorithm specialized for clean line drawings that analyzes the drawing's topology in order to overcome junction ambiguities. A gradient-based pixel clustering technique facilitates topology computation. This topological information is exploited during centerline extraction by a new "reverse drawing" procedure that reconstructs all possible drawing states prior to the creation of a junction and then selects the most likely stroke configuration. For cases where the automatic result does not match the artist's interpretation, our drawing analysis enables an efficient user interface to easily adjust the junction location. We demonstrate results on professional examples and evaluate the vectorization quality with quantitative comparison to hand-traced centerlines as well as the results of leading commercial algorithms.

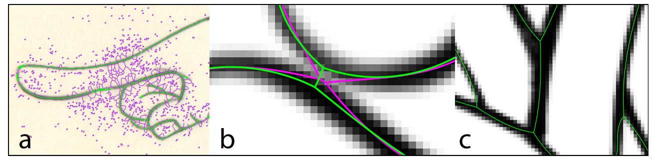

Figure 1: *Vectorization Challenges. Noise (a) and spatially adjacent strokes (b) require fine tuning of threshold parameters in existing approaches. Even for clean, high resolution input images, the superposition of strokes near junctions and sharp corners results in inaccurate centerline placement (c). (Results of Adobe Live Trace for different threshold settings shown in purple and green.)*

1 Introduction

Raster and vector representations form the foundation upon which nearly all two-dimensional graphics is built. Raster images can represent extremely rich detail but do not encode the kind of semantic information that promotes editing. Vector images, on the other hand, abstract image content as mathematical primitives such as lines and arcs that facilitate editing but can limit detail. While converting from a vector to a raster representation is a straightforward sampling operation, the complementary procedure of vectorization is significantly more difficult since it involves inferring high-level abstractions from low-level pixel content.

Hand-drawn 2D animation represents one particularly important application area of vectorization. The expressiveness, efficiency, and tactile control afforded by real pencil and paper are yet to be matched by digital drawing tools, and therefore 2D animations are often still hand-drawn on paper and then scanned. The content in the resulting raster images cannot be easily edited or used with higher level algorithms that require a stroke-based vector representation, such as computer-assisted inbetweening [Whited et al. 2010]. A similar problem exists even when digital drawing tools are used, since artists often build up lines with many short strokes, leading to an unorganized collection of tiny unconnected segments that are not amenable to further high-level processing. In this domain, an automated vectorization approach is essential as a feature animation can contain hundreds of thousands of individual drawings.

Loose and sketchy drawings contain a great deal of ambiguity which makes automatic vectorization extremely difficult. At the other end of the spectrum, "clean" drawings are defined by crisp, distinct lines and thus present the ideal input for vectorization. However, even in this case, ambiguities at stroke junctions make vectorization deceptively difficult. Due to its fixed structure and limited resolution, the regular grid of a raster image is ill-suited to represent regions where many strokes come together, overlap, cross, or join. As a consequence, current vectorization algorithms often fail at accurately representing junctions.

The poor quality of junction reconstruction is due in part to the local nature of existing vectorization algorithms: extracted lines result solely from the information contained in a fixed-size pixel neighborhood. In practice, a larger scope is often necessary to understand the inherent structure of the strokes defining a junction. In this sense, one can consider these stroke relationships as *non-local*. Motivated by this observation, we propose a non-local vectorization algorithm that employs the analysis of a drawing's topology in order to extract high-quality centerlines and junctions from clean drawings.

The first step of our approach analyzes the input image to derive the stroke topology. Here a gradient-based pixel clustering technique is employed that facilitates the extraction of the correct topology in under-sampled regions of the drawing. This topological information is exploited during centerline extraction by a "reverse drawing" procedure that reconstructs all possible drawing states prior to the creation of a junction and selects the most likely stroke configuration. If the automatic result does not match the artist's interpretation, our drawing analysis enables an efficient user interface to easily adjust the junction location. We demonstrate results on professional examples and evaluate the vectorization quality with quantitative comparison to centerlines hand-traced by an expert artist as well as with side-by-side comparisons to output from leading commercial methods.

Our system fits naturally into current pipelines to enable vector processing of scanned drawings. We make the technical contributions of the gradient-based pixel clustering procedure for accurate topological analysis as well as the reverse drawing procedure for producing the most plausible junction configurations. For either hand- or digitally-drawn input, our work provides a bridging technology that converts drawings into a format designed for further editing, automatic inbetweening, or other advanced vector-based processing algorithms.

2 Related Work

Existing vectorization methods can be roughly classified into two groups based on whether they are designed to process image or line data. Techniques for the vectorization of general images make the assumption that the image content can be represented by a collection of boundary curves, together with smooth interpolating functions between the curves. In one family of approaches, the image is first segmented into regions by, for example, triangulation or using quad-dominant gradient meshes, and then the region interiors are filled with smooth gradients [Lecot and Lévy 2006; Sun et al. 2007; Xia et al. 2009]. Alternatively, using diffusion curves, the smooth interior can be computed by solving a Poisson equation with the curves as boundary constraints [Orzan et al. 2008]. Zhang and colleagues [2009] present an approach specifically tailored for temporally coherent cartoon animations, while Sýkora and coworkers [2005] vectorize regions of cartoon frames for the purpose of compression. Both address final cartoon frames with all colored foreground and background layers.

A related problem is the extraction of curve skeletons from 2D shape boundaries using variational methods [Cornea et al. 2007]. Additional hybrid methods seek to find centerlines in images. In the field of medical imaging, blood vessel extraction requires identifying and reconstructing the tubular structures from images and scans. A range of techniques has been developed [Kirbas and Quek 2000], from pattern recognition, to model-based and tracking-based methods. In this domain, Whited and colleagues [2009] present a semi-automatic centerline extraction from networks of strokes that also works in more general images (e.g. river networks from satellite imagery). While effective for vectorizing general image content, none of the above methods are designed to work with line drawings and do not sufficiently address the accurate extraction of centerlines and junctions.

The second group of methods is primarily concerned with vectorization of line drawings such as technical layouts. Prominent approaches are based on tracing [Freeman 1974], thinning [Lam et al. 1992], or methods utilizing contours or projections such as the Hough transform [Liu and Dori 1998]. Due to the focus on technical images, many of these methods are restricted to fitting straight line segments to input drawings [Janssen and Vossepoel 1997]. Exceptions include the method by Chang and Yan [1998], which fits Bezier curves, and the method by Zou and Yan [2001], which focuses on issues such as jaggy line boundaries and junction points. Hilaire and Tombre [2006] also address robustness and describe fitting of higher order primitives such as arcs in addition to line segments. Their method mainly addresses issues found in binary technical drawings and cannot be easily generalized to freehand sketches. Bartolo et al. [2007] describe an approach based on Gabor and Kalman filtering in order to convert rough scribbles into a vectorized representation. When boundaries are well defined, skeleton methods [Lakshmi and Punithavalli 2009] produce good vector centerlines that could be used to represent line drawings. However distorted skeleton centerlines appear at junctions.

Finally, commercial tools for vectorization of line art include Toon Boom Harmony, Adobe Live Trace, CorelDRAW, VectorEye, VectorMagic, and AutoTrace.

In many situations, existing methods and techniques provide high-quality results. However, strokes drawn very close together and junctions areas are usually poorly reconstructed. In most cases, this limitation arises from an algorithm that employs local operators without considering the overall structure of the drawing. Such information is needed to accurately reconstruct stroke centerlines and junction points, and to perform more sophisticated editing operations such as morphing and inbetweening. In our work we specifically address these open challenges.

3 Overview of Approach

The goal of vectorization is to extract stroke centerlines and a network of vector curves and junctions from an input raster image of a line drawing.

Current vectorization techniques face two major challenges. The first problem, illustrated in Figure 1a,b, is insufficient *local* discrimination of individual strokes due to noise and spatial proximity of strokes. The second problem, which is of a *global* nature, is the difficulty of obtaining accurate estimation of centerlines at junctions. It is a global problem because it requires information about the drawing topology and stroke configuration (see Figure 1b,c and also Figure 5). Both problems compromise centerline estimates and result in bad vectorization quality using existing techniques.

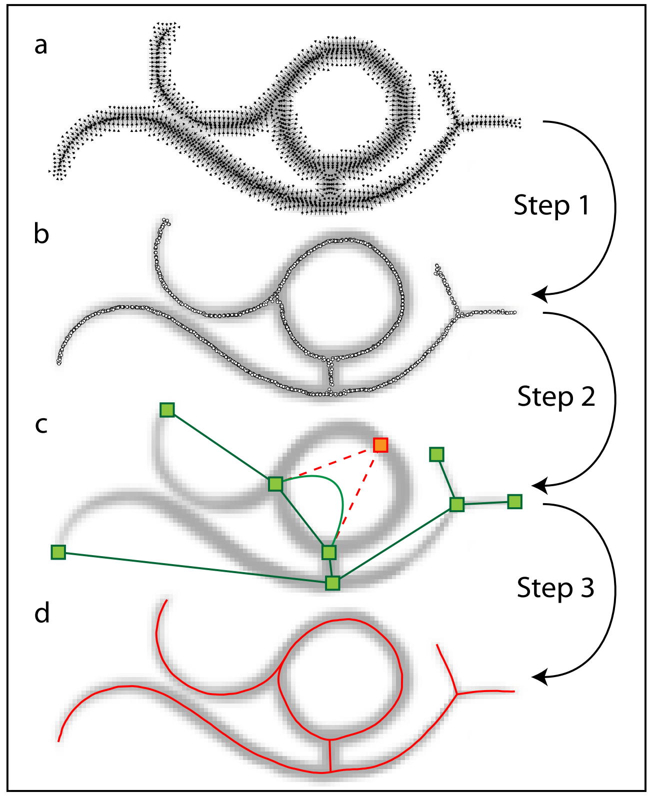

Figure 2: *Method Overview. Step 1: First, our algorithm disambiguates the input pixels using a gradient-based clustering process. Step 2: From the clusters, the topological skeleton of the drawing is extracted. Step 3: By utilizing the topological information, our reverse drawing procedure extracts accurate centerline estimates and junction positions.*

The algorithm we propose for vectorization of line drawings addresses these problems with a novel bottom-up analysis, which translates into three successive processing phases; each step of the algorithm increases the level of abstraction of the representation, until accurate centerlines can be reconstructed.

Step 1: Stroke Disambiguation by Clustering: Our first observation is that, in line drawings, the *color gradient* at each input pixel often provides a good local estimate of the center of a nearby stroke centerline (see Figure 2a). We show that a clustering approach, which moves pixels along the gradient field based on the notion of gradient “confidence”, enables effective local disambiguation of strokes (Figure 2b) and compares favorably to existing skeletonization techniques.

Step 2: Topology Extraction: After clustering, the pixels are connected to form a *cluster graph* (Figure 2b). The second phase of our algorithm then analyzes this cluster graph to compute the underlying topological skeleton of the drawing (Figure 2c). This skeleton represents the individual stroke segments, stroke endpoints, and junctions between stroke segments. The proposed procedure is

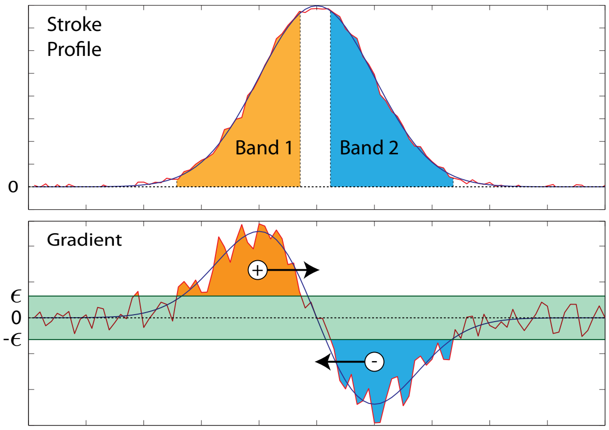

Figure 3: *The gradient threshold ϵ defines two bands of pixels with opposite gradient directions.*

based on the computation of minimum spanning trees as an efficient solution for global topology extraction even on large drawings with complex cluster graphs.

Step 3: Centerline Reconstruction and Reverse Drawing: Using the topology of the cluster graph, the centerlines of the drawing can be extracted (Figure 2d). Particular care is taken in inherently ambiguous regions like junctions. The novelty of this approach is a topology-driven identification of such ambiguous regions, followed by an exploration of all the possible stroke configurations in a process we refer to as *reverse drawing*. We first score pairs of incident stroke segments at a junction and then select the most likely configurations, based on the assumption that smoothly joining stroke segments are generally more likely to belong to a continuously drawn stroke than segments joining at sharp angles. Notice that this step can be applied independently of the previous steps to improve results obtained for instance with robust skeletonization or thinning algorithms.

The algorithmic details of these three phases, from local stroke disambiguation to topology-aware reconstruction of centerline configurations, are described in Section 4. Our results and evaluation in Section 5 demonstrate that our approach resolves the limitations of existing techniques and results in high-quality vectorization.

4 Algorithm

4.1 Clustering for Stroke Disambiguation

Pixels $\mathbf{p}_i$ of a raster input image can be roughly classified into two categories, depending on their respective image gradient ∇_i: small gradients do not carry sufficient information about the stroke center, while large gradients provide a more confident guess about the centerline location. Accordingly, we classify each pixel $\mathbf{p}_i$ as either stationary: $\mathcal{S} = \{\mathbf{p}_i | \|\nabla_i\| < \epsilon\}$ or moving: $\mathcal{M} = \{\mathbf{p}_i | \|\nabla_i\| \geq \epsilon\}$ by thresholding the gradient norm. The norm threshold value ϵ should be set to be above the gradient levels of the image noise. All results presented in this paper have been produced with ϵ equal to 10% of the maximal gradient length.

The basic idea of our stroke clustering is that confident pixels $\mathbf{p}_i \in \mathcal{M}$ move towards the centerline by following the direction ∇_i[1]. Although local noise may influence the trajectory of individual pixels, as long as the gradient noise level is below ϵ, the pixels converge and cluster towards the centerline (see Figure 3).

[1]Gradient directions are kept constant throughout the clustering.

For all pixels $\mathbf{p}_i \in \mathcal{M}$, the motion vector is set to $m_i = \delta t \nabla_i$, where δt is a constant speed factor, in our implementation equal to 10% of the width of a pixel. This has two consequences: first, the pixels move in compact bands, and second, centerlines are located where the two opposing bands meet. The stopping condition for each moving pixel is then naturally given by the motion coherence in its local neighborhood; for each pixel $\mathbf{p}_i \in \mathcal{M}$ the nearest neighbors $\mathcal{N}_i = \{\mathbf{p}_j | \|\mathbf{p}_j - \mathbf{p}_i\| \leq 1\}$ are collected. By looking at the sign of the dot product of the gradients $\nabla_i \cdot \nabla_j$, neighboring pixels $\mathbf{p}_j \in N_i$ are classified either as belonging to the same band (positive dot product), or the opposing band (negative dot product). The stopping condition is then defined as having one or more pixels of the opposing band that traveled past the location of pixel $\mathbf{p}_i$, which can be expressed as $(\mathbf{p}_j - \mathbf{p}_i) \cdot \nabla_i < 0$.

The clustering process terminates when the number of moving pixels drops below 1% of the initial set $\mathcal{M}$. Outliers, such as remaining background and isolated pixels, can be eliminated by removing those pixels that remained stationary through the whole clustering process or that have less then 2 neighbors within a 1-pixel radius. This clustering procedure results in a *contraction* of the input pixels around the approximate location of the stroke centerline.

Notice that at this stage it is possible to get an estimate of the stroke thickness by considering the distance that boundary pixels traveled. In our implementation we do not explicitly mark pixels as belonging to the boundary, but rather we store the traveled distance of each pixel as the approximate local stroke radius r_i and take a conservative estimate by setting it to be the maximum r_j of all $\mathbf{p}_j \in \mathcal{N}_i$. This estimate of the local stroke thickness will be utilized in the subsequent steps of our algorithm.

4.2 Topology Extraction

We are interested in extracting the topology of the drawing. After convergence, the cluster is a point set that densely samples the proximity of the drawing stroke centerlines. Instead of applying techniques from geometry reconstruction, we treat this point set as a graph, and rely on well-known efficient graph algorithms to extract a skeleton that explicitly contains the topology of the drawing.

The procedure is illustrated in Figure 4. A graph structure (the *cluster graph*) is constructed by connecting each clustered pixel $\mathbf{p}_i$ to each neighbor $\mathbf{p}_j$ within the local stroke thickness (see Figure 4b). A weighted edge e_{ij} is added for each pair $(\mathbf{p}_i, \mathbf{p}_j)$. The weight $\omega(e_{ij})$ of an edge is simply the Euclidean distance $\omega(e_{ij}) = D(\mathbf{p}_i, \mathbf{p}_j)$.

The topological skeleton (endpoints, junctions, connectivity) of the drawing is then computed by topology-preserving coarsening of the cluster graph (Figure 4b-e). First, a minimum spanning tree (MST) of the graph is computed [Kleinberg and Tardos 2005]. Due to the dense pixel clustering, the MST is characterized by a number of main branches with many very short branches ("twigs") which contribute to the stroke width/detail, but not ultimately to the topological structure we are seeking. In order to isolate the main branches, the leaves of the MST are iteratively pruned (Figure 4c). To avoid pruning the entire graph, we keep track of the length of the branches being removed, and terminate the iteration if deleting an additional node will make the total length of the removed branch greater than the local stroke thickness.

By definition, any loop in the drawing will be cut by the global MST. Figure 4f-k illustrates a procedure to reliably detect and close these cuts through the construction of a local MST around each leaf node. Consider the cut produced by the global MST (4f). The leaf pruning will erode both sides, widening the cut up to approximately $2r_i$ (4g). For each leaf node (4h), we compute local MST (4i) and then apply leaf pruning (4j). Loop connectivity is restored by taking

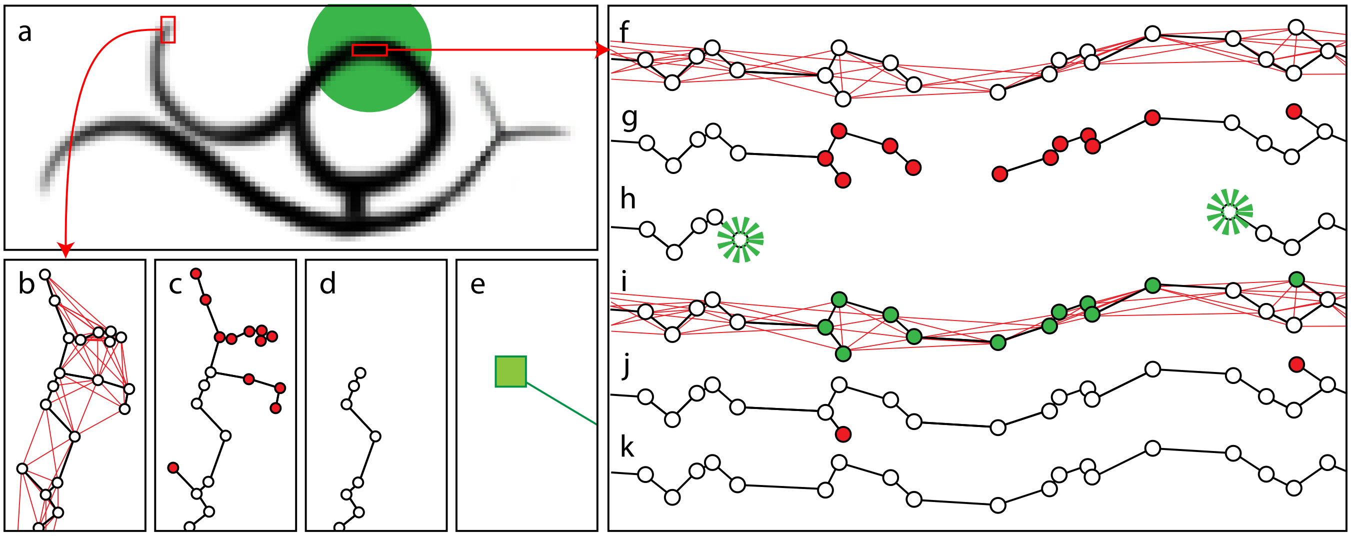

Figure 4: *Topology Extraction and Loop fixing (a).* ***Topology Extraction*** *(b-e). An minimum Spanning Tree (MST) is computed (b). Branches of length smaller than the stroke thickness are removed (c), resulting in a skeletonized version of the cluster (d) from which the final topology of the skeleton is extracted (e).* ***Loop Fixing*** *(f-k). For drawings containing loops (a), the MST computation breaks the loop in at least one location (f). The leaf pruning (g) widens the gap. To restore the link, for each of the remaining leaves (h) a local MST is computed (local scope: the green circle in a). (i) The local and global MSTs are merged, and remaining leaves pruned (j), ultimately restoring the loop (k).*

the union of the global MST and the local ones (4k). Notice that this procedure will not affect actual end points, as the local MSTs followed by the leaf pruning will produce the same initial leaf nodes as the pruned global MST.

We now mention a few implementation details. First, an MST cut generates two leaf nodes, but it is sufficient to apply the above procedure to one of the two. Checking if a leaf is still a leaf after each iteration can save computation time. Second, prior the computation of the local MST, we zero the weights of the edges that are in the global MST. This will force the local MST to pick the same edges and only expand within the gap, avoiding the introduction of triangular structures or undesired loops. Third, in order to account for variation in our stroke thickness estimate, we set a conservative range for the local MST of $4r_i$ (green circle in Figure 4a).

The final topological skeleton of the drawing can then be obtained by collapsing all nodes of valence 2 in the graph. Nodes of valence 1 then correspond to stroke endpoints, nodes of valence ≥ 3 to stroke junctions, and the graph edges represent the topological stroke segments in the drawing (see Figure 4e).

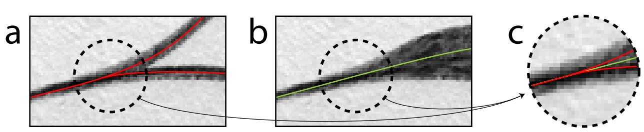

Figure 5: *Local Ambiguity. A junction (a) and a stroke with varying thickness (b) cannot be distinguished by considering the local appearance only (c).*

4.3 Centerline Extraction and Reverse Drawing

The main challenge for reconstructing accurate centerline estimates is ambiguities which cannot be resolved by purely local methods and hence lead to reconstruction artifacts in existing approaches. Figure 5 illustrates such an ambiguity where two strokes converge and cannot be distinguished from a stroke with varying thickness by considering only a local window. Moreover, even when it is clear that multiple strokes meet at a junction, one has to choose among a number of possible configurations (see Figure 6).

In order to address these challenges, our algorithm performs two steps. First, a set of base centerlines is traced, connecting all end points and junctions according to the drawing topology. We refer to the processing up to this point as the *base method*, as it produces centerlines which in nature are similar to the results obtained with prior art (see Figure 18). Second, our reverse drawing procedure (see Figure 7) utilizes the drawing topology to identify ambiguous regions (e.g., junctions and sharp corners) and then corrects the centerline estimates by choosing the most likely centerline configuration among all possible ones.

4.3.1 Base centerlines

Base centerlines are constructed by computing the source to destination shortest path on the full cluster graph. As source and destination points we pick the junctions and endpoints defined by the drawing topology. Hence, each topology edge generates a base centerline. The stroke thickness is derived locally from the selected path nodes.

Two special cases have to be considered: an edge connecting a junction to itself (loop) and two junctions connected by more than one edge. For both special cases, in our implementation we split the edges adding dummy valence-2 junctions (see red node in Figure 2c), forcing the shortest path to take the individual necessary routes.

The extracted shortest paths are then smoothed by applying a data-driven smoothing operator which moves the path along the local curve normal towards the center of mass of the clustered stroke pixels. For each point $\mathbf{p}_i$ of a centerline path, a Gaussian weighting function is used to assign weights to the cluster pixels $\mathbf{p}_j$ in the neighborhood. The refined position $\tilde{\mathbf{p}}_\mathbf{i}$ is then given as

$$\tilde{\mathbf{p}}_\mathbf{i} \leftarrow \mathbf{p}_i + ((\mathbf{c}_i - \mathbf{p}_i) \cdot \mathbf{n}_i)^T \mathbf{n}_i, \text{ with} \quad (1)$$

$$\mathbf{c}_i = \frac{\sum_j w_j \mathbf{p}_j}{\sum_j w_j}, \text{ and } w_j = e^{-\frac{D(\mathbf{p}_i, \mathbf{p}_j)}{2\sigma^2}}, \quad (2)$$

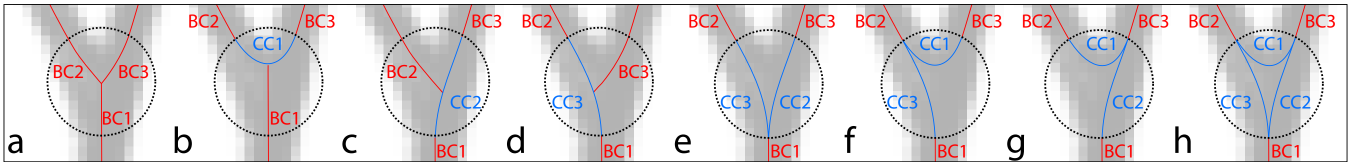

Figure 6: *Junction Configurations. This image shows all possible configurations combining the base centerlines (BCs) and continuous centerlines (CCs) inside the ambiguous region (black dotted circles). Case 'h' is included for completeness, but occurs rarely in practice.*

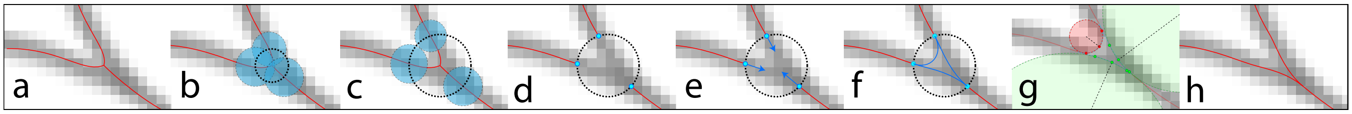

Figure 7: *Reverse Drawing. For correct centerline estimation in the proximity of overlapping strokes (e.g., at junctions) (a), our algorithm first identifies the ambiguous region (b,c) and removes the corresponding centerline estimates (d). From the intersections of the ambiguous region with the base centerlines (e), continuous candidate centerlines (CCs) are computed (e, f). Then the stroke-curvatures of the CCs are evaluated (g), and the final centerline configuration is selected (h).*

where $\mathbf{n}_i$ is the normalized approximation of the local curve normal and $\sigma = r_i$ to adapt the weighting scheme to the local stroke thickness (Figure 8). We compute approximate vertex normals by averaging the normals of the adjacent line segments, and then interpolate these normals linearly over the segments.

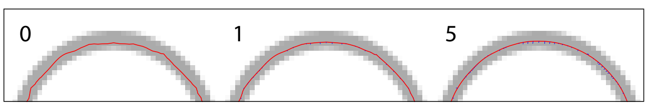

Figure 8: *Smoothing. This image shows the result of the smoothing of centerline paths for 0, 1, and 5 iterations. The movement of points is marked in blue.*

4.3.2 Reverse drawing

An overview of the reverse drawing procedure is illustrated in Figure 7. The first step consists of identifying ambiguous regions where strokes overlap.

Ambiguous region estimation: For each junction, we iteratively grow a circle AR at the junction position until the strokes no longer overlap. The process is summarized by the steps:

1. Create a circle AR centered at the junction.
2. Intersect AR with each adjacent base centerline BC_j.
3. At each intersection, generate a circle S_j of radius equal to the local stroke thickness (Figure 7b, blue circles).
4. If any pair of S_j intersects, increase the radius of AR and repeat from 2. Otherwise stop. (Figure 7c).

Continuous centerline construction: Given the ambiguous region at a junction, the base centerlines inside this region are removed and replaced by all possible configurations of continuous centerline candidates (CCs) at that junction (Figure 6b-h). Given the position and curve tangent of the intersections between the ambiguous region and a pair of base centerlines, a CC is computed as a cubic Hermite spline (Figure 7e,f). The normalized curve tangents are scaled to one third of the distance between the two points.

Stroke-Curvature: When considering the curvature of strokes, we observe that different stroke thicknesses (Figure 9a,b) result in different perceived curvatures even when the centerline is the same. As shown in Figure 9c, we define the *stroke-curvature* α by sampling the stroke centerline at three points with distance r (the local stroke thickness) and fitting a circle, computing $\alpha = 2\arcsin\left(\frac{r}{2c}\right)$.

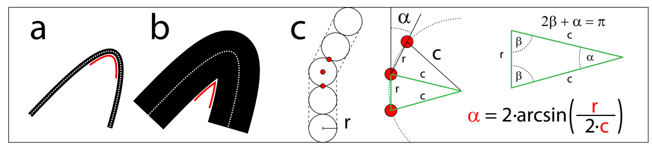

Figure 9: *Stroke-Curvature. (a) and (b) illustrate the apparent difference in visual smoothness of strokes with same centerline but different radii. (c) shows the geometric reconstruction of the stroke-curvature α from the local stroke radius r and curvature radius c.*

To compute the stroke-curvature α_i of a continuous centerline CC_i, we sample CC_i uniformly with the sample distance r according to the local stroke thickness, and then set α_i as the maximum stroke-curvature over the whole curve (see Figure 7g).

Centerline selection: Our goal is to connect the base centerlines (BCs) around an ambiguous region by either picking continuous centerlines (CCs) only (e.g., Figure 6f-h), or a combination of BCs and CCs (e.g., Figure 6b-e).

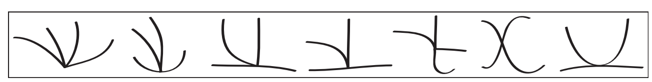

Figure 10: *Examples of valence-4 junctions.*

In application contexts where fixed alphabets or specific drawing patterns are defined, junction classes (T, Y, X, etc) can be associated with predefined centerline configurations. However, in line drawings, the number of such classes is virtually infinite. Figure 10 shows just a few of the many valence-4 junctions that can be found in line drawings, each one with its own nuances. This makes it impractical to build a comprehensive classification.

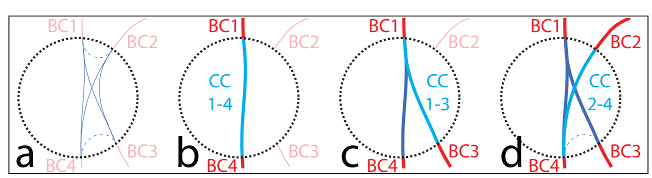

Figure 11: *Centerline Selection. CCs are generated (a), and for each, the stroke-curvature is computed. Any CC (dashed) with curvature exceeding the threshold is excluded. Sequentially (b,c,d), the CCs with the smallest curvature are selected until all BCs are connected (d).*

Instead, we opt for a fixed selection scheme that can operate on any kind of junction (see Algorithm 1). Figure 11 illustrates the major steps for a valence-4 junction.

Data: A junction with BCs and CCs
Result: A set of accepted BCs and CCs
foreach CC_i **do**
    Compute α_i;
    if $\alpha_i > t$ **then** reject CC_i as *sharp turn*;
end
Put the remaining CCs in a list;
Sort list by ascending α_i;
while *the list is not empty* **do**
    Extract the straightest CC_i from the list and accept it;
    if *all BCs are connected* **then** Terminate;
end
foreach *disconnected BC_i* **do**
    Extend BC_i linearly until it crosses any accepted centerline (either BC_i or CC_i);
    Accept BC_i;
end

Algorithm 1: Centerline Selection Scheme.

This selection scheme favors straight centerlines over curved ones. The stroke-curvature threshold t discriminates acceptable continuous centerlines from undesirable sharp turning connections, which are usually associated with cases where base centerlines stop at the junction (Figure 6a-d). In our implementation t is equal to $50°$, an optimal value according to our empirical validation (see Section 5 and Figure 13).

Spikes: A spike is generally formed by two strokes, drawn in approximately opposite directions, that overlap in the region of the tip. Spikes are a special case of valence-3 junctions where one branch is relatively short. In terms of topology, a junction exists where the overlap starts, and the tip forms an end point, as illustrated by Figure 12a. After such a structure is detected in the topology extraction, the reverse drawing procedure treats it as any other junction by selecting the straighter CCs, which results in the appropriate representation of the spike.

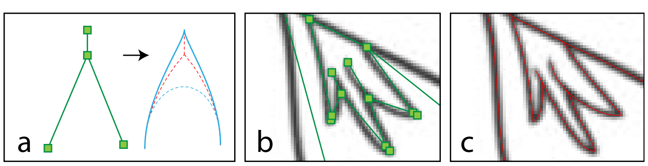

Figure 12: *Spikes can be considered a special case of a junction, where one of the branches is very short (a). (b-c) show respectively the topology and the resulting centerlines for several spikes.*

5 Validation and Results

We evaluate our approach on a variety of clean line drawings, including production drawings from 2D short and feature animations (Figure 17). Of the paper examples, we include recently created drawings, as well as an archival piece which has degraded in quality over time and is therefore difficult to vectorize using existing techniques due to the age-related artifacts in the paper texture. We provide numerical and visual comparisons to two standard commercial tools, Adobe Live Trace [Adobe 2010] and Harmony [Toon-Boom 2010], as well as the Stentiford and Zhang-Suen thinning algorithms (implemented in Wintopo [Sisoft.net 2010]).

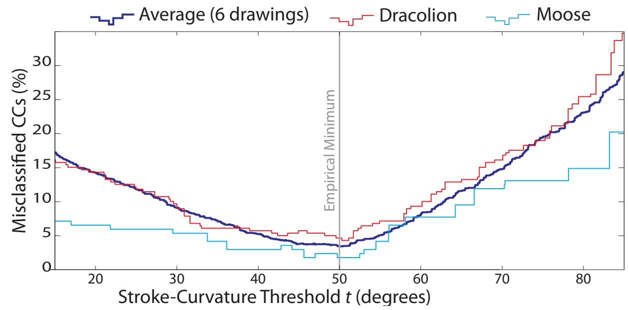

Figure 13: *Stroke-Curvature Thresholding. To establish an empirically optimal value of t, we built a data set containing more than 2000 CCs, manually labeled as either smooth or sharp turns. This image displays the classification obtained with Algorithm 1, plotting percentage of misclassified CCs versus the input threshold parameter t. For cross-validation, we show the average misclassification for 6 out of the 8 drawings, which suggests an optimal threshold value t, and finally we assess the quality of the prediction on the two remaining 2 drawings.*

5.1 Evaluation

In order to evaluate the accuracy of our centerline reconstruction and to compare it to existing methods, we used a data set consisting of eight drawings (four scanned, four digitally drawn and then rasterized). In total, the drawings contain more than a thousand topologically relevant points (59.8% junctions, 40.2% endpoints), and about two thousand centerlines. For the digitally created images, accurate ground truth centerlines are available. For the scanned drawings, we asked an artist to manually trace the centerlines and highlight the correct topological configuration at junctions. The results of the evaluation are summarized in Table 1. We evaluate the following criteria:

Selection Scheme Accuracy (SSA): Algorithm 1 requires as input a stroke-curvature threshold parameter t. To pick an appropriate value, we proceeded as follows. For all images in our data set, an artist manually labeled all candidate centerlines as either *smooth* or *sharp turns*, obtaining a data set with more than 2000 classified CCs. Then, as illustrated in Figure 13, we evaluated the percentage of misclassified CCs against different thresholds t and picked the empirical optimum at $t = 50°$. To validate this choice, we then performed a leave-one-out cross-validation (repeated for all drawings). On average, our choice produces an optimum prediction error of $1.99°$, which corresponds to an error of 0.25% in terms of misclassified CCs. Overall, with the chosen threshold value $t = 50°$, our algorithm produces results that match the artist's classification in 95.5% of the cases. Furthermore, we broke down the values based on valence. Junctions with valence greater than 3 occurred less frequently (5.7% of total), and were more difficult to classify (accuracy 93.4%).

Centerline Error (CE): We evaluate centerline quality by computing the average minimum distance of dense sample points on the extracted centerlines to the ground truth centerlines (see Figure 14). The error with our approach has an average of only 4.13% of the average stroke thickness. We perform the same computation for Adobe Live Trace, resulting in an average of 7.97%, and for the base version of our algorithm (where no reverse drawing is applied) which shows an average error of 5.38%. For the result in the specific drawings, refer to the second to last column in Table 1. Our method shows consistent improvements of the centerline quality for all examples.

Salient Points Error (SPE): Junctions and endpoints are critical elements in the vectorization of line drawings. The quality of a re-

Name	Input Type	Image Res.	Process Time	SSA Valence 3	> 3	% Valence > 3	CE vs. ALT	vs. Base	SPE vs. ALT	vs. Base
Alligator	Scan Clean	2048^2	3m 10s	95.3%	88.8%	1.6%	176 %	125 %	159 %	138 %
Dracolion	Digital	1024^2	29s	95.5%	100%	2.2%	279 %	177 %	145 %	144 %
Dr Facilier	Scan Clean	1024^2	46s	96.0%	75.0%	4.8%	168 %	120 %	181 %	157 %
Father	Scan Rough	2048^2	3m 50s	97.4%	97.4%	9.5%	186 %	140 %	270 %	156 %
Moose	Scan Digital	2048^2	3m 27s	98.6%	98.6%	8.2%	140 %	106 %	168 %	134 %
Mouse	Digital	1024^2	1m 1s	95.8%	90.9%	8.0%	355 %	158 %	136 %	113 %
Muten	Digital	1024^2	24s	96.0%	N/A	0%	212 %	125 %	379 %	288 %
Sheriff	Digital	1024^2	55s	94.5%	94.5%	19.8%	187 %	123 %	145 %	115 %

Table 1: *Numerical results and ground truth evaluation for different input drawings. See Section 5 for a detailed discussion (ALT: Adobe Live Trace, Base: base version of our algorithm, where no reverse drawing is applied).*

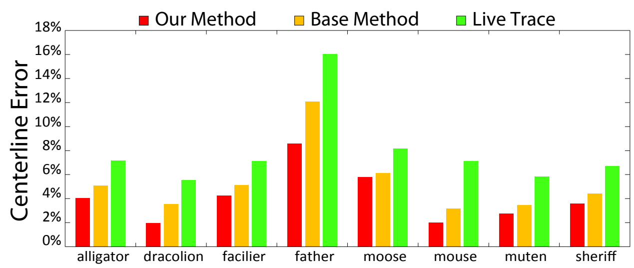

Figure 14: *Centerline Error. This image shows a comparison of the centerline error for the results obtained with Adobe Live Trace, our method, and the base version of our method, where no reverse drawing is applied. The values are expressed as percentage of the average stroke thickness in each drawing. Our method consistently produces a better score.*

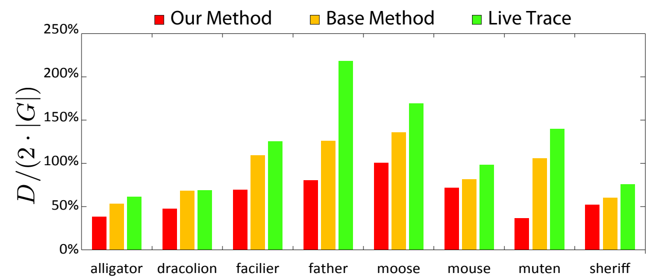

Figure 15: *Salient Points Error. This image shows a comparison of the salient points error for the results obtained with Adobe Live Trace, our method, and the base version of our method, where no reverse drawing is applied. The values are expressed as percentage of the average stroke thickness in each drawing. Our method consistently produces a better score.*

construction can be assessed by considering both the correctness in the placement of these points as well as the correctness in the overall topology. Given a drawing, consider two sets of salient points G and M, from the ground truth and the method to be evaluated respectively. We then compute:

$$D = \sum_{i \in M} \min_{j \in G} dist(i,j) + \sum_{j \in G} \min_{i \in M} dist(i,j) \tag{3}$$

A good reconstruction should produce a set M that is as similar as possible to G. In the case of correct topology, the contribution of a pair $(i \in M, j \in G)$ will express the quality of the salient point placement. However, if a method does not capture the correct topology (by either missing a junction or detecting more junctions than there actually are) mismatched pairs will penalize the score with additional accumulated distance. Figure 15 shows the comparison of our method with Adobe Live Trace and the base version of our method. To assess the performance across our data set, we normalized the values D considering twice the number of salient points $2 \cdot |G|$ in each drawing.

5.2 Input Resolution and Clustering Robustness

The clustering step described in Section 4.1 samples the input image to generate bands of moving pixels that stop when they meet. We experimented with super-sampling as a way to cope with very low input resolutions. Even 1-pixel wide strokes, if super-sampled appropriately with a smooth filter (in our tests: bi-cubic filter, 4 samples per pixel), can be reconstructed accurately, leading to robustness comparable to competing approaches.

As shown in Figure 16, the centerline error relative to the ground truth (logarithmic scale) drops exponentially with increasing resolutions. Differences between our method and Adobe Live Trace become more evident at higher resolutions.

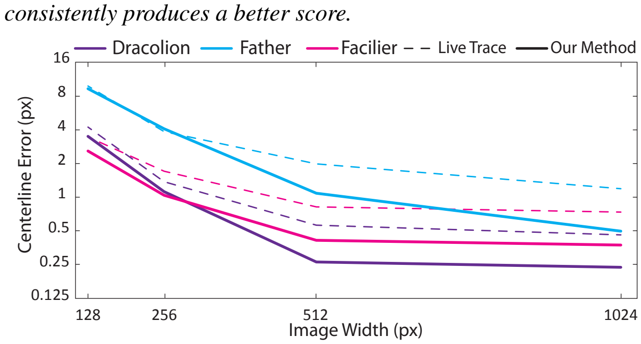

Figure 16: *Effect of increasing the input resolution. Both our method and Adobe Live Trace (dashed) exhibit an exponential reduction of the error. Differences between the two methods become more evident at higher resolutions.*

5.3 Result Images

For a general overview of the capabilities of our method, we present a selection of vectorization results in Figure 17, taken from the eight drawings used in the previous section for error evaluation. Notice how several ambiguous regions are properly handled, and in most cases the proper junction configuration is selected. These results combine the advantages of both the pixel clustering and the reverse drawing.

Figure 18 illustrates the benefits of pixel clustering on its own. Figure 18a,b show the vectorization results obtained with Adobe Live Trace and Sisoft Wintopo. Figure 18c shows results from only the base portion of our algorithm, where centerlines are traced from the pixel clusters, but no reverse drawing is applied. Notice how both medial axis and thinning methods (a,b) rely on the creation of boundaries, usually obtained through color thresholding. However,

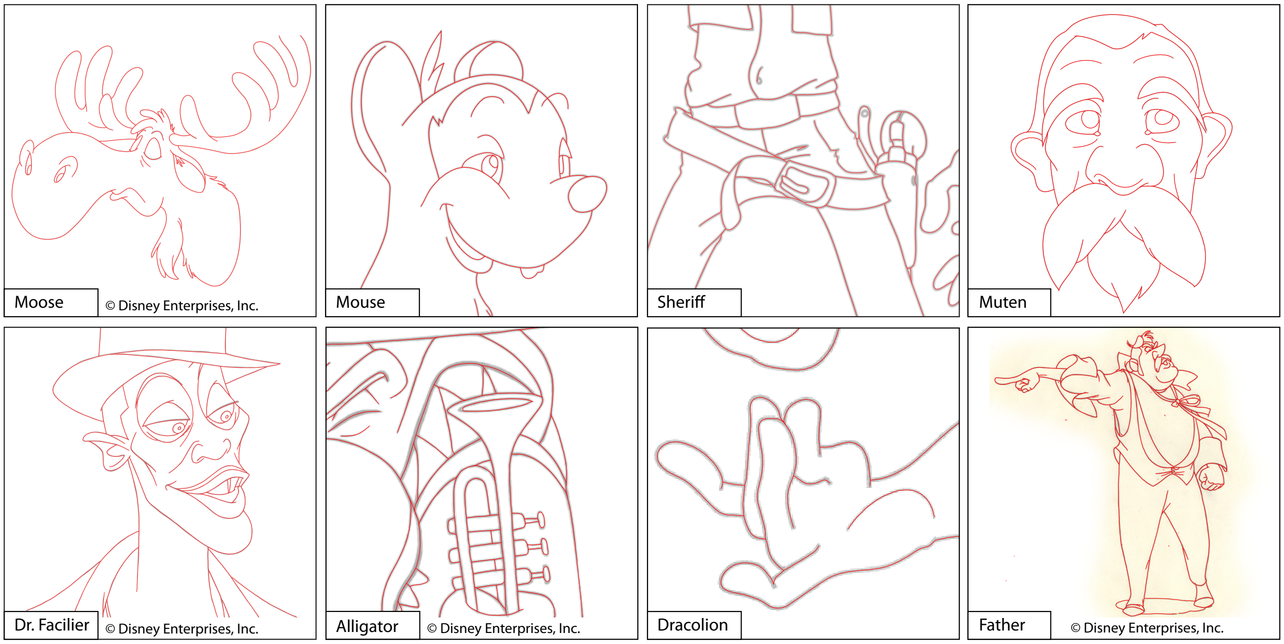

Figure 17: *Collection of vectorization results generated with our method. The Father image is not fully clean (the paper quality is degraded and lines not sharp), and thus represents a borderline case.*

with low thresholds, nearby strokes are not distinguished properly, and with high values parts of the drawings are lost. Pre-sharpening the images can alleviate these problems, but the proper kernel size has to be used; in our experiments, this approach required manual tuning to achieve good results. In contrast, our method successfully separates nearby strokes, while avoiding stroke losses.

Finally, Figure 19 provides visual comparisons of our complete method to leading commercial vectorization implementations, Toon Boom Harmony [ToonBoom 2010], Adobe Live Trace [Adobe 2010], and SoftSoft Wintopo, respectively. Notice how the existing techniques have difficulties in discriminating nearby strokes, which results in merged centerlines for separate strokes and an incorrect drawing topology. Moreover, junction points are placed at inaccurate positions. Our method finds a more natural placement. For these comparisons, we attempted to tune the parameters of the software packages to obtain the best possible results. Our method uses the standard parameter values described in the previous sections and requires no per-drawing tuning.

5.4 Processing Time

The processing time of our method for each of the input images is provided in Table 1, measured on a desktop PC[2]. The time necessary to process an image depends both on its resolution as well as the number of strokes and junctions. Due to the more complex topological analysis of a drawing, our algorithm requires more processing time than tools such as Live Trace or Harmony. However, the timings are still in the range of only a few seconds to minutes, and the necessary time spent in post-production to correct the centerline estimates and junctions is significantly reduced compared to the previously available solutions (see Figure 20).

For the Alligator example (Figure 17), we compared the timing of manual post-processing of the output to fix erroneous junctions for Live Trace and our method: it took an artist 12 times longer to produce comparable results for the Live Trace example.

[2]Mac Pro, Quad-Core 2.66 GHz, 4GB RAM

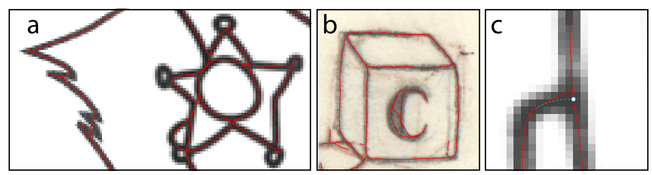

Figure 21: *Limitations. With the current method, small details (a) may not be captured by the topology extraction. The method is also not designed to work with solid areas (b). (c) The linear extension of the base centerlines does not always produce the best junction location (higher order extrapolation shown in blue).*

5.5 Limitations and Future Work

An important application for this system is 2D animation. As is, our system can be used in tandem with inbetweening techniques, such as [Whited et al. 2010]. However, the vectorization quality could be improved by considering the information contained in subsequent frames, improving the decision making (Algorithm 1) and recovering from errors in the topology. The difficulty however is that distinct elements moving independently may drastically change both the topology and the junction configurations, making these extensions very challenging.

As for any raster based method, image quantization and noise play a crucial role. In the case of low resolution input, details are hard to extract (Figure 21a). In our tests with variable resolution (see Figure 16) we observed a similar response for our method and Adobe Live Trace, but in order to not break - i.e. have enough moving pixels - the proposed pixel clustering requires super-sampling. Noise may lead to strokes being torn apart. Additionally, we observe in Figure 16 the decay of the improvement rate as the resolution increases. This is partly due to the error getting closer to zero, but also to the difficulty in exploiting the additional information.

While Algorithm 1 in most cases produces good results, there may be cases of technical drawings, with grid structures or specific texture-like patterns, where a failure of the algorithm appears in many junctions, making manual fixing very time consuming. Here, a possible approach would be to apply machine learning to update the guiding criteria. The current spike detection relies on the presence of particular topological structures, and might also be improved by using machine learning in conjunction with the proposed stroke-curvature measure to be able to explicitly extract sharp corners.

Finally there are some minor limitations. As shown in Figure 21b, our method is not designed to work with solid areas. Additionally, as explained in Section 4.3, base centerlines that are selected by Algorithm 1 for the final configuration are extended *linearly* inside the ambiguous regions. Figure 21c shows a case where this linear method results in a less accurate junction location than the one obtained with higher order extrapolation.

6 Conclusion

We have described a novel vectorization technique for clean line drawings which produces a high-quality representation suitable for vector processing. Our approach consists of two techniques: a gradient based pixel clustering that helps disambiguate difficult cases, and a reverse drawing procedure which exploits the drawing topology to make educated choices when dealing with junctions. Since these techniques are independent, the reverse drawing can be applied to improve the result of existing techniques that provide the drawing topology.

We have demonstrated the application of our method to a variety of professional examples. Our results show how our approach improves the vectorization of junctions and nearby strokes which represent the major shortcomings of state-of-the-art solutions when it comes to clean line drawings. Such accurate junction and centerline recovery makes stroke-based editing operations such as automatic inbetweening more feasible in a digital pipeline.

Possible future directions include addressing the limitations of the current method by exploring the vectorization of more sketchy and noisy drawings, considering image pre-filtering using LoG [Chen et al. 1987]. Moreover, we are interested in studying the semantic information present in sketchy drawings, considering strokes not only as sets of pixels, but as objects with mathematical properties (e.g., trajectory, direction) with the goal of exploiting such semantics in a clustering approach.

References

ADOBE, 2010. Illustrator. http://www.adobe.com/.

BARTOLO, A., CAMILLERI, K. P., FABRI, S. G., BORG, J. C., AND FARRUGIA, P. J. 2007. Scribbles to vectors: preparation of scribble drawings for CAD interpretation. In *SBIM*, 123–130.

CHANG, H.-H., AND YAN, H. 1998. Vectorization of hand-drawn image using piecewise cubic bézier curves fitting. *Pattern Recognition 31*, 11, 1747–1755.

CHEN, J. S., HUERTAS, A., AND MEDIONI, G. 1987. Fast convolution with laplacian-of-gaussian masks. *IEEE Trans. Pattern Anal. Mach. Intell. 9* (July), 584–590.

CORNEA, N. D., SILVER, D., AND MIN, P. 2007. Curve-skeleton properties, applications, and algorithms. *IEEE Trans. Vis. Comput. Graph. 13*, 3, 530–548.

FREEMAN, H. 1974. Computer processing of line-drawing images. *ACM Comput. Surv. 6*, 1, 57–97.

HILAIRE, X., AND TOMBRE, K. 2006. Robust and accurate vectorization of line drawings. *IEEE Trans. Pattern Anal. Mach. Intell. 28*, 6, 890–904.

JANSSEN, R. D. T., AND VOSSEPOEL, A. M. 1997. Adaptive vectorization of line drawing images. *Computer Vision and Image Understanding 65*, 1, 38–56.

KIRBAS, C., AND QUEK, F. K. H. 2000. A review of vessel extraction techniques and algorithms. *ACM Computing Surveys 36*, 81–121.

KLEINBERG, J., AND TARDOS, E. 2005. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc.

LAKSHMI, J. K., AND PUNITHAVALLI, M. 2009. A survey on skeletons in digital image processing. In *Proceedings of the International Conference on Digital Image Processing*, IEEE Computer Society, Washington, DC, USA, 260–269.

LAM, L., LEE, S.-W., AND SUEN, C. Y. 1992. Thinning methodologies - a comprehensive survey. *IEEE Trans. Pattern Anal. Mach. Intell. 14*, 9, 869–885.

LECOT, G., AND LÉVY, B. 2006. ARDECO: Automatic Region DEtection and COnversion. In *EGSR'06*, 349–360.

LIU, W., AND DORI, D. 1998. A survey of non-thinning based vectorization methods. In *SSPR/SPR*, 230–241.

ORZAN, A., BOUSSEAU, A., WINNEMÖLLER, H., BARLA, P., THOLLOT, J., AND SALESIN, D. 2008. Diffusion curves: a vector representation for smooth-shaded images. *ACM Trans. Graph. 27*, 3.

SISOFT.NET, 2010. Wintopo. http://wintopo.com/.

SUN, J., LIANG, L., WEN, F., AND SHUM, H.-Y. 2007. Image vectorization using optimized gradient meshes. *ACM Trans. Graph. 26*, 3, 11.

SÝKORA, D., BURIÁNEK, J., AND ŽÁRA, J. 2005. Video codec for classical cartoon animations with hardware accelerated playback. In *Proceedings of International Symposium on Visual Computing*, 43–50.

TOONBOOM, 2010. Harmony. http://www.toonboom.com/.

WHITED, B., ROSSIGNAC, J., SLABAUGH, G., FANG, T., AND UNAL, G. 2009. Pearling: Stroke segmentation with crusted pearl strings. *Pattern Recognition and Image Analysis 19*, 2 (06), 277–283.

WHITED, B., NORIS, G., SIMMONS, M., SUMNER, R. W., GROSS, M., AND ROSSIGNAC, J. 2010. BetweenIT: An interactive tool for tight inbetweening. *Computer Graphics Forum (Eurographics 2010 Proceedings) 29*, 2.

XIA, T., LIAO, B., AND YU, Y. 2009. Patch-based image vectorization with automatic curvilinear feature alignment. *ACM Trans. Graph. 28*, 5, 1–10.

ZHANG, S.-H., CHEN, T., ZHANG, Y.-F., HU, S.-M., AND MARTIN, R. R. 2009. Vectorizing cartoon animations. *IEEE Trans. Vis. Comput. Graph. 15*, 4, 618–629.

ZOU, J. J., AND YAN, H. 2001. Cartoon image vectorization based on shape subdivision. In *Computer Graphics International*, 225–231.

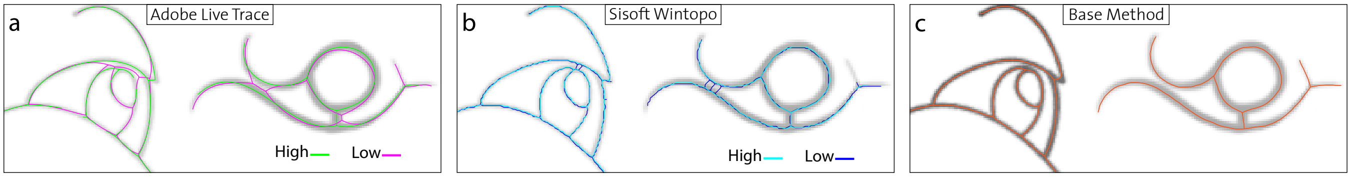

Figure 18: *Pixel Clustering Comparison. This image shows a comparison between Adobe Live Trace (a), Sisoft Wintopo (b), and the result of our base algorithm (c), where only the base centerlines have been extracted, and no reverse drawing has been applied. In (a) and (b) different color threshold parameters have been used. Notice how in order to get the nearby strokes separated, the color threshold must be set to high values, losing other parts of the drawing. Our method successfully separates the strokes without such losses.*

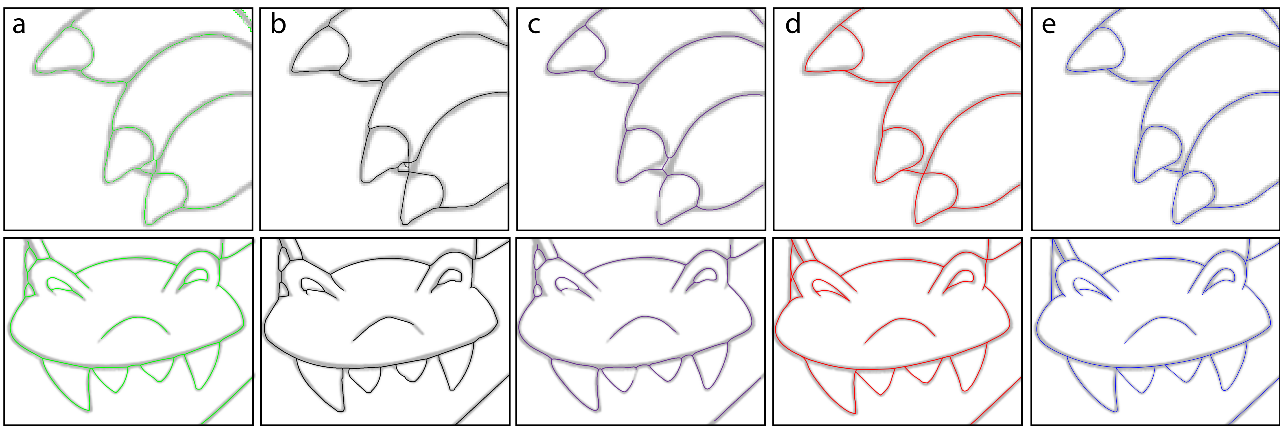

Figure 19: *General Comparison. This image shows close-up comparison between Adobe Live Trace (a), SiftSoft Wintopo (b), Toon Boom Harmony (c), our method (d), and the ground truth (e).*

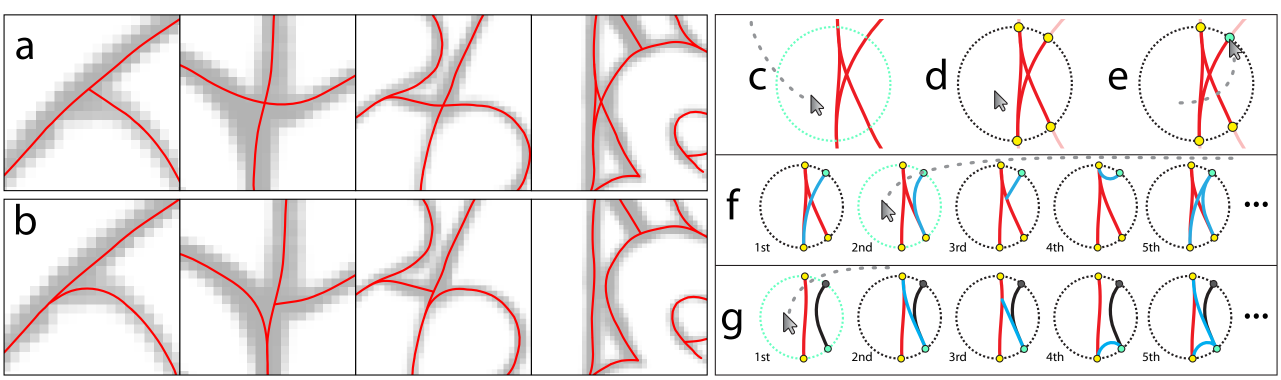

Figure 20: *System output (a). With a user interaction of only a few seconds per junction, different configurations can be obtained (b). The interaction steps are shown in (c-g). First, the user selects the junction (c), and activates the editing mode (d). BCs entry points (yellow dots) are used by the user to make the desired changes. Once an entry point is clicked (e), the system shows a set of valid configurations to choose from (f). The first criterion is that the entry point must be connected: 1 connection is favored over 2, 2 over 3, etc. Straighter CCs are the first choice, then linearly prolonged BBs, and finally, rejected CCs. If a choice influences the connectivity of another entry point, that point is automatically selected, and possible configurations displayed (g). Previous choices are marked in black. This process requires at most one choice per entry point.*