EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling (2004)
John F. Hughes and Joaquim A. Jorge (Editors)

Pen Stroke Extraction and Refinement using Learned Models

Saul Simhon[†] and Gregory Dudek[‡]

McGill University
Quebec, Canada

Abstract

This paper presents a smart interface that automatically extracts and refines pen strokes from images of hand drawn sketches. The interface allows users to digitize hand-drawn material such sketches of flowcharts, cartoons or other pen based drawings and automatically isolate and refine the individual strokes making up the sketch. First, we present a method for extracting pen strokes based on learned constraints on curves. The approach consists of using a training set that shows good examples of curves and how a user would draw them. Given an image of a hand-drawn sketch, the system selects the pen stroke that is most statistically consistent with the examples in the training set and ranks the others based on their likelihood. Users can keep the selected candidate or they may scroll through the other top candidates to select a preferred solution. Second, we present an overview of our refinement procedure and its application on the extracted pen strokes. Using the same database of examples, the extracted pen stroke is refined to make it 'look' more like those in the database.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Line and Curve Generation

1. Introduction

Many people find that the easiest and most natural way to describe a concept is by drawing a rough sketch. Despite this, the creation of high-quality good-looking sketches remains time consuming and skill dependent. In this work, we develop an application that allows users to easily edit, clean up and refine a hand drawn sketch. The emphasis is on sketches that consist of pen strokes that are either disproportionate, displaced, spurious or lacking the detail of the objects they attempt to depict. Users can quickly extract pen strokes from a sketch and perform transformations on them such as a split, merge, resize and reposition. Users can also apply an automated refinement process on the extracted pen strokes [SD04].

While most modern image editing applications provide standard tools for transforming objects, they always rely on the assumption that a pen stroke (or any other object) is already isolated and selected. Manually extracting individual pen strokes from images that are noisy or include occlusions is a tedious task. Further, once the pen strokes are extracted, the existing methods for detailed editing require either good artistic talent or strong knowledge of specialized tools. To address these issues, the main focus of our work is on the automated extraction of pen strokes from images with a supplementary refinement process.

Sketches are often used as a first order presentation of concepts for artistic drawings, animations (story board), comics, prototype designs and diagrams. In all these, the goal is to quickly construct a coarse visualization of a domain specific end-result. While this coarse visualization lacks the quantitative detail, within a given a context, the sketch is meant to provide sufficient information for a human observer to *hallucinate* the original intention. For example, in one context, a rough circle may be a coarse representation of a gear, while in another, it may represent the head of a stick figure. In order to simulate this, our system must understand the sketch and resolve ambiguities under various contexts.

Given the a wide array of drawing styles and contexts, providing a system that can accommodate for all is a difficult task. Further, how to computationally characterize different drawing types in accordance to human perception remains

[†] saul@cim.mcgill.ca
[‡] dudek@cim.mcgill.ca

an open problem. In our approach, the system learns from examples the types of pen strokes it expects to see. This includes learning both the coarse shapes a user will draw and the refined shapes the user actually intends to draw. As such, the user can collect different training sets, classifying each in a different context and applying them in various domains of application.

Each training set is used to train a Hidden Markov Model, a modeling formalism used to encode constraints on the sequences of hidden states (the refined shape) and the observations they produce (the shape the user would actually draw when their intent is the refined one). The user would then use one of these models to both extract and refine a pen stroke. If the sketch has been originally entered using a digital medium (such as a tablet or PDA), then the pen strokes are readily available and the system skips directly to the refinement step. Otherwise, given an image of a sketch, where the pen strokes are not defined and may be occluded, the user can simply click near an end-point of the desired pen stroke and the system would automatically extract it.

2. Related Work

There is an abundance of literature concerning the extraction of curves from images. This literature typically deals with several distinct processes: edge detection, curve grouping, and segmentation. The latter two (grouping and segmentation) refer to the process of extracting meaningful connected curves from data that may be confusing, cluttered or incomplete.

Curvature information is a key heuristic for building curves from noisy data. A standard approach in the presence of ambiguous data is to select the curve that minimizes a "goodness measure" based on minimum curvature, minimum absolute curvature, or minimum variation in curvature. Such goodness measures can be posed as energy functionals, procedural rules, or decision trees. Work by Ullman and Sha'ashua [US98] use locally connected networks to determine saliency for smoothness, continuity, and curve length. Similarly, Jacobs [Jac93] developed a method for extracting curve segments based on a convex saliency measure. Earlier work by Low showed how curves can be extracted by applying perceptual grouping rules with proprieties such as proximity, collinearity and parallelism [Low85]. Estrada and Jepson [EJ04] use predefined geometry-based affinity measures to evaluate the quality of line segment junctions. All of these approaches have proven very powerful, but they are based almost universally on an attempt to obtain generic domain-independent grouping strategies. In most cases, these strategies are based on rules inspired by visual psychophysics [Kof22].

Another approach is to use probabilistic methods to maintain likelihoods over multiple solutions. These likelihoods are typically biased using both hard-coded conditionals (such as a preference on curvature) and learned conditionals (computed on the fly using a single example or multiple examples). Williams and Jacobs [WJ97] developed a stochastic method for contour extraction. In their work, a prior probability on the shape of a boundary is computed using paths of particles that undergo a random walk in the image. Recently, August and Zuker [WJ03] defined the notion of *curve indicator fields* as generic models for producing edge likelihoods. In particular, their experiments employ a Markov random field model for contour enhancements. Similar in spirit to these methods, our approach consists of applying probabilistic constraints on the candidate pen strokes found in images. Our system is geared toward understanding images of sketches where, using a Hidden Markov Model, we learn not only the constraints on shapes, but also the user's intent. Further, unlike many of the previous methods, our method captures features over multiple scales using a wavelet representation.

While the problem of extracting curves from images has been a long standing research topic in the domain of computer vision, there is an abundance of work specific to sketching systems that employ techniques similar in principle for recognizing and grouping sketch components. Saund [Sau03] developed a method to rank candidate paths for perceptually closed contours. The approach used local preferences for tightly closed paths and smooth paths. Later, Saund *et al.* [SFLM03] developed a sketch editing application that includes image analysis techniques for the separation of foreground from background and the perceptual grouping of sub-regions of a sketch. Other approaches to sketch understanding typically use predefined primitive fitting for recognizing curve strokes and include high-level context based rules. Alvarado and Davis [AD01] develop a smart sketching interface geared to recognize a mechanical engineer's sketch. In their work, ambiguities on shapes are resolved using prior contextual knowledge and preferred criteria. Landay and Myers [LM01] develop a sketching system that can recognize predefined curve strokes and widgets (for designing UI), grouping them based on spacial relationships. The sketch is then refined to produce a *cleaner* version of the original.

3. Pen Stroke Extraction

3.1. Overview

Our system allows users to take an image of a sketch and extract the pen strokes. Once a curve is extracted, the user may edit it using traditional curve transformations (split, merge, move, scale, rotate, filter, etc.). The user may also automatically refine the curve (discussed later). Figure 1 show an example of extracting a curve from an image. The system identifies the pen stroke that best fits a selected model (in this case, a leaf). This same model is further used to automatically augment the extracted curve.

The key issue we address is: how do we identify what pen

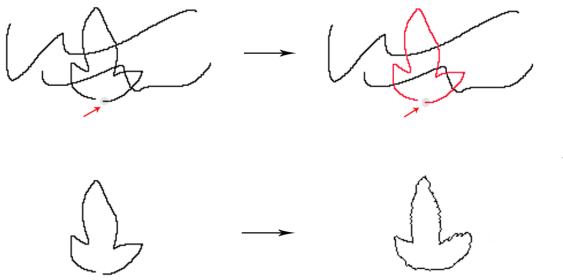

Figure 1: *Example extraction and refinement. Top left shows original image and the user pointer, top right shows the automatically selected curve (in red). Bottom left shows the curve isolated by dragging it and bottom right shows result of the automated refinement process.*

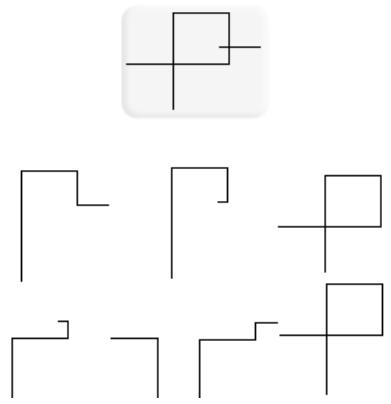

Figure 2: *The pen stroke paths that can be produced form an image. Top figure show the original image and the figures below show the possible paths.*

Figure 3: *Some examples from a leaf training set. The figures on the right show the refined examples and the figures on the left show the strokes a user will typically draw when their intended shape is the associated refined one. These coarse curves can be manually drawn or can be produced by filtering the refined curves.*

stroke path the user really intends to select? Figure 2 shows some of the possible candidate paths that can be extracted from an example sketch. If we can automatically identify the path that *stands out* from the rest, in some context, then we can include it as a potential candidate for the user's selection. Our approach to this consists of using a Hidden Markov Model (HMM) to model local constraints from a set of examples that are pre-classified by the user under some semantic context. For example, if the user presents the system with examples of flow-chart shapes, the system would train a HMM then search for the path that best fits the model constraints. As such, if there is a shape that stand out significantly more than others, it can be extracted with high confidence, even if the shape is not an exact instance from the training set. Figure 3 shows an example training set for leaves. The set includes examples of both the refined shapes and the shapes the user would actually draw when their intent is for the refined ones.

3.2. Generating Paths

We assume that we are given an image of a sketch with a well defined foreground consisting of thin edges that are two pixels thick. (In practice, there are well established methods that can extract foreground and thin edges, most prevalent in common edge detectors or skeletal graphs [SBTZ02, Can86, SFLM03].) When the user clicks on the image near the desired pen stroke, we wish to generate all possible paths starting at the nearest edge. Our approach consists of first finding a starting point, then recursively iterating over the neighbors of pixels to create a *path segment tree*. This tree encodes all possible paths, from the starting point, that a pen stroke can take in the image. The nodes represent path segments and edges represent junctions (Figure 4).

To find the starting point, we first search for the nearest pixel within some distance d to the mouse click. This distance should span enough pixels such that the user does not have to deal with the accuracy in clicking exactly on the sketch. Once we find this pixel, we examine its neighbors in four directions (up, down, left and right). If there is only one neighbor, then the current pixel is considered as the starting point, otherwise, we recursively check for the staring point at each neighbor. On successive iterations, we only consider a neighbor if it is not the same pixel from the previous iteration (so that we do not go back the way we came). This is performed for l steps and if an end-point is not found, then the original starting point is chosen (this avoid issues with closed paths).

Once we found the starting point, we perform a similar recursion and construct the path segment tree. We start with the starting point p_0 and create a path segment c_0^0 with one point $(p_0(x), p_0(y))$. Then, we examine the neighbors. If there is only one neighbor p_1, then we add that point to c_0^0, otherwise, for each neighbor p_i we create a new path c_i^1 and add the point. This is performed recursively over all the neighbors in order to complete the tree for all junctions. Figure 4 shows an example path segment tree.

To avoid issues with loops, we only consider a pixel as a neighbor if that pixel does not already exist in the current segment or any of the parent segments up to the root of the tree. For each node of the tree, we maintain an image mask in order to have random access to this information. Further, in order to allow for self-intersection, we always consider points that have three unmasked neighbors, even if they exist in the parent segments, they are never marked in the image mask.

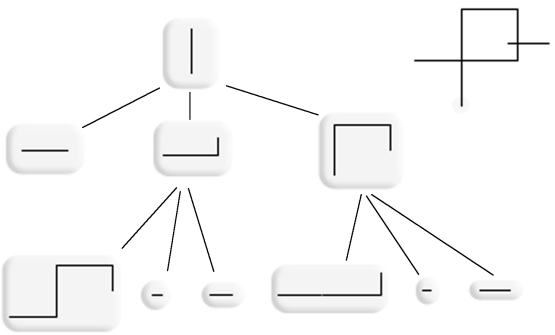

Figure 4: *Path segment tree. Top right shows the original image with the starting point highlighted*

In both the search for the starting point and for the paths, we perform a first order look-ahead that ignores neighbors falling under certain criteria. In particular, because we assume the width of a pen stroke to be two pixels (and in bad cases, maybe more), we must prune the neighbors of neighbors as follows: if p_0, p_1 and p_2 are neighbors of p and if all of p_0's neighbors are neighbors of either p_1 or p_2, then we do not consider p_0. For consistency, we always follow the same ordering when pruning neighbors.

Once we have finished the search, we can traverse the tree and construct N paths capturing every possible pen stroke. Each path is sampled with the same sampling rate as that used in training, filtered to reduce aliasing effects and normalized if the training examples are also normalized. The tangent angles along the paths are also computed from the Cartesian points.

3.3. Ranking Paths

Our goal is to rank the candidate paths generated by the method in Section 3.2. Our approach consists of examining what path can best be explained by our training data. Because our approach considers local shape similarity, novel paths, never seen in the training set, can also be classified as belonging to the set. This is accomplished by using a Hidden Markov Model that encodes constraints on the sequence of tangent angles.

3.3.1. Pen Stroke Representation

We represent a pen stroke path by a curve over 2D space parametrized by the arc-length. Let α represent a parametric curve $(x(t), y(t))$ where t is the arc-length of the curve from $0 <= t <= T$. We assume that our curves are suitably normalized and uniformly sampled. We encode them by their starting point, starting direction, and the sequence of all edge-lengths and exterior angles (i.e., a planar polyline). The shape is characterized up to rigid orientation-preserving transforms by just the latter data (i.e., ignoring start-point and start-direction).

3.3.2. Markov Model

We assume that a stochastic process Δ is the source for a family of curves. Each curve is considered to be a random signal with characteristics described by the probability density function of the process. Let α denote a curve and $\theta(t)$ denote the tangent angles of that curve parametrized over the arc-length t. We assume that the sequence of samples $\theta(t)$ from $0 <= t <= T$ for all curves exhibit an n^{th} order Markov property, i.e. Δ is a Markov Process:

$$p\{\theta(t+1) \mid \theta(t), \theta(t-1), \ldots, \theta(t-n+1)\} =$$

$$p\{\theta(t+1) \mid \theta(t), \theta(t-1), \ldots, \theta(0)\}$$

This *locality* condition states that information from recent samples is sufficient to predict the next sample point. Further, this dependency can also be position-variant, where statistical relationships between successive points may be *non-stationary* with respect to the arc-length. Thus, the locality condition suggests that we only consider local constraints. For many training sets, these local constrains may be applicable anywhere on a curve (e.g. curves that exhibit a regular properties such as a straight line or zig-zag pattern). For other training sets, one can manually specify that the constraints should change at different regions. (For example, if the training set consist of examples of leaves, when we are extracting the left side of the leaf we may not need to consider the constraints from the right side.)

3.3.3. Hidden Markov Model

A Hidden Markov Model encodes the dependencies of successive elements from a set of *hidden* states along with their relationship to *observable* states. It is typically used in cases where a set of states, that exhibit the Markov property, are not directly measurable but only their effect is visible through other observable states. Formally, a Hidden Markov Model Λ is defined as follows:

$$\Lambda = \{M, B, \pi\} \quad (1)$$

where M is the transition matrix with transition probabilities of the hidden states, $p\{h_i(t) \mid h_j(t-1)\}$, B is the confusion

matrix containing the probability that a hidden state h_j generates an observation o_i, $p\{o_i(t) \mid h_j(t)\}$, and π is the initial distribution of the hidden states.

In our work, the hidden states play the role of sample points from the refined curves while the observation states play the role of samples points of the coarse curves (Figure 5). These samples points are the tangent angles along the training examples. Given an ensemble of training examples, each example i in the set is an associated pair of refined/coarse curves $\{\theta_i(t), \phi_i(t)\}$. We first estimate the transition probabilities by the statistics of successive elements from the user defined *stationary regions* and construct the transition matrix M where:

$$\Psi_\theta(t+1) = M(t)\,\Psi_\theta(t) \tag{2}$$

The transition matrix propagates the information in the probability distribution $\Psi_\theta(t)$ to predict the next distribution $\Psi_\theta(t+1)$. For sets that exhibit stationarity $M(0) = M(1) = \ldots = M(T) = M$, measured over all sample points of every training example. Otherwise, the transition matrix is calculated over fixed local regions of the curves. The ability to specify the local regions of stationarity (hence global non-stationarity) allows us to accommodate for styles that inherently posses some global constraints (i.e., the constraints at one area of the curve are not used in other areas of the curve). We assume a uniform initial probability distribution $\pi = \Psi_\theta(0)$, providing equal likelihoods to all examples at arc-length position zero.

We estimate the probabilities of the confusion matrix B from the statistics of associated sample points $(\theta_i(t), \phi_i(t))$ over all examples i. This allows us to predict what the user may draw given a belief of the intended shape:

$$\Psi_\phi(t) = B\,\Psi_\theta(t) \tag{3}$$

where the elements of the confusion matrix are the conditional probabilities $p(\phi_i|\theta_j)$ for all states i and j.

Because we cannot expect our inputs to match exactly with training, we blur our distribution Ψ using a Gaussian noise model. Further, only encoding first order constraints on the tangent angles may not sufficiently capture the local structure of the examples. Thus, we capture the shape at multiple scales by first using a wavelet transform [FS94] and then sampling over the different scales, storing this information in the states. The dimensionality of our state space is augmented to accommodate for this mutli-scale representation. That is, $\theta(t)$ now becomes $\theta(t,s)$ where a sample point at a larger scale represents a summary for the region (simulating a higher order Markov assumption). Because this increases the state space exponentially, we no longer store the entire transition and confusion matrices for every possible states but only keep those that are non-zero (labeling the matrices' index dynamically).

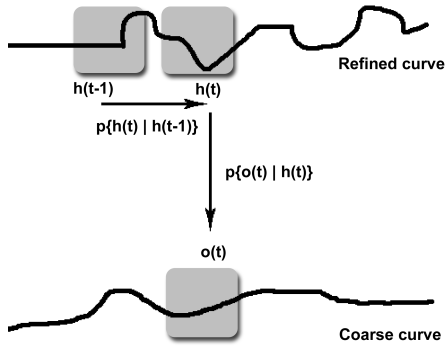

Figure 5: *For all curve segments (in gray), the transition matrix and confusion matrix store the above likelihoods. This is computed over every example in a given set. The states store the tangent angles at multiple scales.*

3.3.4. Evaluating Curve Candidates

Given a candidate curve $C_k = \{\phi_k(0), \phi_k(1), \ldots, \phi_k(T)\}$, and a HMM Λ, we wish to compute the likelihood that the observation sequence $\phi_k(0), \phi_k(1), \ldots, \phi_k(T)$ is generated by the model Λ. This is accomplished by evaluating the HMM over the sequence of observations using the *Forward* algorithm. First, using the confusion matrix B, we take our initial distribution π over the hidden states and condition it by the observation $\phi_k(0)$. Then we propagate this forward using the transition matrix M. We iterate this process over the observation sequence, up to and including the last sample point $\phi_k(T)$:

$$\begin{aligned} \psi\{\theta_i(t)\} &= p\{\phi(t) \mid \theta_i(t)\}\psi\{\theta_i(t)\} \\ \psi\{\theta_i(t+1)\} &= \sum_j [p\{\theta_i(t+1) \mid \theta_j(t)\}\psi\{\theta_j(t)\}] \end{aligned} \tag{4}$$

Once we have reached the last observation, we sum up the probabilities in the vector $\Psi(T)$ (accounting for all possible ways the model can generate the observation sequence). This value is then used to rank the candidate curve.

Once all candidate curves from our segment tree have been ranked, we sort them from best to worst and present to the user the best one. The user can then scroll through the list to see if there is a preferred solution. Figure 6 shows the top candidate path selected. This was computed using a training set consisting of zig-zag patterns at four orientations (Figure 7).

4. Pen Stroke Refinement

We present an overview of our refinement process applied to the extracted pen strokes. A more detailed description can

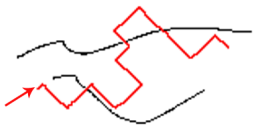

Figure 6: *Extraction of a zig-zag pattern (shown in red).*

Figure 7: *A sample from the Zig-Zag training set. The full set consisted of this pattern at four orientations.*

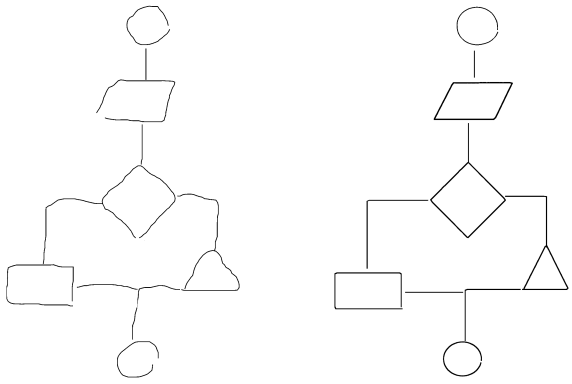

Figure 8: *Left shows a hand draw sketch (given the user pen stroke) and the right shows the refined version.*

be found in [SD04]. Our method consists of producing the most statistically consistent mixture of segments from the training set that best explains the observations. That is, when we extract a coarse curve Φ from the image, the system looks at the most likely hidden state sequence that would generate the observations:

$$\max_{\theta_i \ldots \theta_N} p\{\theta(0), \theta(1), \ldots, \theta(T) \mid \phi(0), \phi(1), \ldots, \phi(T), \Lambda^0\} \tag{5}$$

We solve for this problem by decoding the HMM using the *Viterbi* algorithm.

Figure 8 shows an example where the user draws a sketch consisting of simple shapes and then automatically refines the sketch. The training set used for this refinement is shown in figure 9. Note that the resulting shapes in figure 8 are not exact instances of the training set but are made to exhibit the same local features. Figure 10 shows an example where the user extracted a square shape, refined it and then resized it. This was also done using the latter training set (Figure 9).

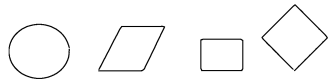

Figure 9: *Simple shapes training set. The associated coarse curve (expected observations) consist of blurred version of the above.*

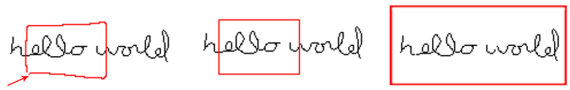

Figure 10: *Left shows extraction (in red), middle shows refinement, right shows a resize.*

5. Implementation

The core of our application resides in a module that implements the automated extraction (Section 3) and refinement (Section 4) algorithms. This module has been written in $C++$ and can be linked in either a Unix or Windows environment. There is also a Web based API that allows for submissions and returns of curve strokes using HTTP post.

We have implemented two interfaces; one in a Linux environment which directly links to the core $C++$ module, the other in a web-based environment consisting of a java applet using HTTP post to the back-end API. The Linux implementation provides a good developing, testing and debugging environment while the web-based interface helps gather user feedback for ongoing studies on the usability of the system.

6. Future Work

One of the difficulties that our curve extraction system can run into is where there is an excessive number of candidate paths. A direction for future work can be to examine method that dynamically prune the graph in parallel to the evaluation process. While its always best to evaluate a path in its entirety, one can suggest that an iteratively deepening look-ahead can help prune candidate path that will never rank high enough.

7. Conclusion

We have presented a method for extracting and refining pen strokes from images of hand drawn sketches. Users can easily select a pen-stroke, even when occluded by other strokes,

and perform editing operations. The system learns from examples and can adapt to a wide array of domains.

References

[AD01] ALVARADO C., DAVIS R.: Resolving ambiguities to create a natural computer-based sketching environment. In *International Joint Conference on Artificial Intelligence* (2001).

[Can86] CANNY J.: A computational approach to edge detection. In *IEEE Transactions on Pattern Analysis and Machine Intelligence* (1986), vol. 8.

[EJ04] ESTRADA F. J., JEPSON A. D.: Controlling the search for convex groups. In *Technical Report CSRG-482* (January 2004).

[FS94] FINKELSTEIN A., SALESIN D. H.: Multiresolution curves. In *Proceedings of ACM SIGGRAPH* (July 1994), pp. 261–268.

[Jac93] JACOBS D.: Robust and efficient detection of convex groups. In *Computer Vision and Pattern Recognition* (1993), pp. 770–771.

[Kof22] KOFFKA K.: Perception: and introduction to the gestalt-theory. In *Psychological Bulletin* (1922), vol. 19, pp. 531–585.

[LM01] LANDAY J. A., MYERS B. A.: Sketching interfaces: Toward more human interface design. In *IEEE Computer* (2001), vol. 34, pp. 56–64.

[Low85] LOWE D. G.: Perceptual organization and visual recognition. In *Kluwer Academic Publisher* (1985).

[Sau03] SAUND E.: Finding perceptually closed paths in sketches and drawings. In *IEEE Transactions on Pattern Analysis and Machine Intelligence* (April 2003), vol. 25, pp. 475–491.

[SBTZ02] SIDDIQI K., BOUIX S., TANNENBAUM A., ZUCKER S. W.: Hamilton-jacobi skeletons. In *International Journal of Computer Vision* (2002), vol. 48, pp. 215–231.

[SD04] SIMHON S., DUDEK G.: Sketch interpretation and refinement using statistical models. In *Eurographics Symposium on Rendering* (2004).

[SFLM03] SAUND E., FLEET D., LARNER D., MAHONEY J.: Perceptually-supported image editing of text and graphics. In *ACM Symposium on User Interface Software and Technology (UIST)* (2003), pp. 183–192.

[US98] ULLMAN S., SHA'ASHUA A.: Structural saliency: The detection of globally salient structures using a locally connected network. In *International Conference on Computer Vision* (1998), pp. 321–327.

[WJ97] WILLIAMS L., JACOBS D.: Stochastic completion fields: A neural model of illusory contour shape and salience. In *Neural Computation* (1997), vol. 9, pp. 837–858.

[WJ03] WILLIAMS L., JACOBS D.: Sketches with curvature: The curve indicator random field and markov processes. In *IEEE Trans. on Pattern Analysis and Machine Intelligence* (April 2003), vol. 25, pp. 387–400.