Computers & Graphics 35 (2011) 250–264

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

SpeedSeg: A technique for segmenting pen strokes using pen speed

James Herold [a], Thomas F. Stahovich [b,*]

[a] Department of Computer Science and Engineering, University of California, Riverside, CA 92521, United States
[b] Department of Mechanical Engineering, University of California, Riverside, CA 92521, United States

ARTICLE INFO

Article history:
Received 31 July 2009
Received in revised form
12 November 2010
Accepted 10 December 2010
Available online 16 December 2010

Keywords:
Pen stroke segmentation
Sketch understanding
Pen-based user interfaces
Pen speed

ABSTRACT

We present SpeedSeg, a technique for segmenting pen strokes into lines and arcs. The technique uses pen speed information to help infer the segmentation intended by the drawer. To begin, an initial set of candidate segment points is identified. This set includes speed minima below a threshold, and curvature maxima at which the pen speed is also below a threshold. The ink between each pair of consecutive segment points is then classified as either a line or an arc, depending on which fits best. Next, a feedback process is employed, and segments are judiciously merged and split as necessary to improve the quality of the segmentation. In user studies, SpeedSeg performed accurately for new users. The studies also demonstrated that SpeedSeg's accuracy is surprisingly insensitive to the values of many of the empirical parameters used by the technique. However, it is still possible to quickly tune the system to optimize performance for a given user. Finally, SpeedSeg outperformed a state-of-the-art segmentation algorithm.

1. Introduction

Despite the power and sophistication of modern engineering software, engineers often delay using it until late in the design process. Instead, they do much of their early design work on paper, using sketches extensively. After essential design issues have been resolved, the sketched designs are then recreated on the computer, so that design software can be used. This problem is due, in part, to the cumbersomeness of traditional user interfaces. When designs are in flux, the inconvenience of traditional user interfaces places too much overhead on the creative process.

In our research, we are working to change this by creating interfaces that allow users to operate software by means of familiar sketching skills. The ultimate goal is to create software that is as easy to use as paper and pencil, yet is as powerful as traditional software. Rather than the user having to learn how to use software, software should be able to read, understand, and use the kinds of sketches people ordinarily draw. For example, an engineer should be able to operate an analysis tool by drawing the same kinds of simple sketches that he or she would draw when solving problems by hand [18,19,9].

When implementing sketch-based computer interfaces, care must be taken to avoid placing constraints on the drawing process. For example, some existing sketch-based systems require that each pen stroke represent a single shape, such as a single line or arc segment [44,15,6,7,35]. Other systems allow pen strokes to have more complicated shapes, but each stroke must constitute a single symbol or gesture [31,8,22,39]. While these kinds of drawing constraints facilitate shape recognition, they can result in a less than natural drawing environment.

The work presented here concerns the low level processing of pen strokes necessary to overcome some of these kinds of constraints. In particular, we present SpeedSeg, a technique for automatically segmenting pen strokes into the intended geometric primitives. SpeedSeg enables one to draw a shape with as few or as many stokes as desired. For example, one can draw a triangle with one, two, or three pen strokes. Likewise, SpeedSeg enables one to include parts of different shapes or symbols in the same pen stroke. Shape recognizers, such as those described in [2,10,9,24] are required for assembling the segments into larger shapes. In Section 6 we present results obtained using SpeedSeg in combination with the recognizer in [24].

The challenge in segmenting a pen stroke is deciding which bumps and bends are intended, and which are accidents. We have found it difficult to determine this by considering shape alone. The size of the deviation from an ideal line or arc is not a reliable indicator of what was intended.

Our approach to segmentation relies on examining the motion of the pen tip as the pen strokes are created. We have observed that it is natural to slow the pen when making many kinds of intentional discontinuities in a shape. Consider, for example, the square shown in Fig. 1. Although it is not drawn as four precise lines, the intended corners can be easily identified as points at which the pen speed is a local minimum.

SpeedSeg's first task is to examine the pen stroke to identify the *segment points*, the points that divide the stroke into different primitives. The initial set of candidate segment points includes

* Corresponding author. Tel.: +1 951 827 7719; fax: +1 951 827 2899.
E-mail addresses: jhero001@ucr.edu (J. Herold), stahov@engr.ucr.edu (T.F. Stahovich).

0097-8493/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2010.12.003

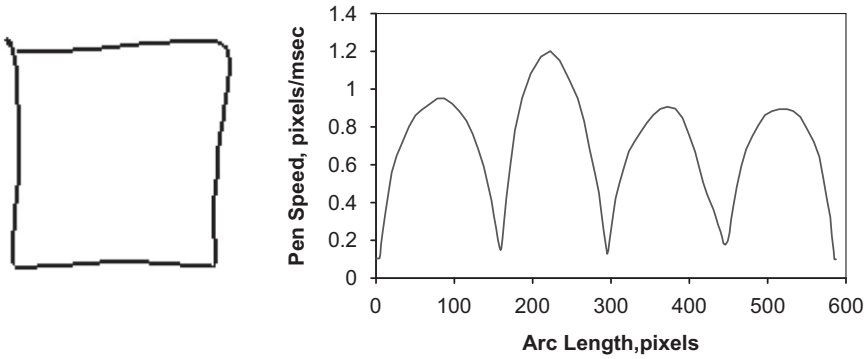

Fig. 1. A square drawn with one pen stroke, and the corresponding pen speed profile. Intended corners correspond to points of minimum speed.

speed minima below a threshold, where the threshold is computed from the average pen speed. Points at which the absolute value of the curvature is a maximum are also included, but only if there is corroborating pen speed information. (For convenience, we will use the term "curvature" to refer to the "absolute value of the curvature." In cases where the sign matters, we use the term "curvature sign.") The ink between each pair of consecutive segment points is called a segment, and is classified as a line or an arc, depending upon which best fits the ink. Although the initial segmentation is reasonably accurate, feedback can often be used to improve the accuracy. During the feedback process, the initial segmentation is examined, and segments are judiciously merged and split as necessary to correct any detected problems.

The SpeedSeg algorithm employs 12 tunable parameters, such as the speed threshold for identifying speed-based segment points, and an error tolerance for merging neighboring line segments that are nearly collinear. Our user studies have demonstrated that our default parameter values work well for a variety of users, even those with no experience using the system. The studies also demonstrated that SpeedSeg's accuracy is relatively insensitive to the values of many of the parameters. Nevertheless, it is still possible to quickly tune the system to optimize performance for a given user.

While speed and curvature have long been used for pen stroke segmentation (e.g., [11,1,43,32]), our work makes several important contributions. First, we provide a systematic study of the influence of the processing parameters on segmentation accuracy. Second, we report the first segmentation technique that can automatically tune itself for individual users. Third, we demonstrate that our approach achieves better accuracy than a state-of-the-art algorithm.

The next section places this work in context by discussing related work. This is followed in Section 3 by a detailed description of the SpeedSeg technique. Section 4 describes details of SpeedSeg's implementation, including the gesture-based interface for manually correcting segmentation errors, and an automated tuning technique to optimize performance for individual users. Section 5 presents the results of three user studies evaluating SpeedSeg's accuracy under a variety of conditions. Section 6 provides a discussion of the results, including an analysis of the influence of the processing parameters on segmentation accuracy. This section also includes performance results obtained with two sketch-based applications built using SpeedSeg. Finally, Section 7 presents conclusions.

2. Background

Segmentation of pen strokes is similar to the problem of corner detection in digital curves, a field that has attracted the efforts of numerous researchers. Algorithms for this problem typically locate corners by searching for points at which curvature is a maximum. To suppress noise and false corners, the data must be smoothed. The main difficulty is selecting a reliable "observation scale" or amount of smoothing. Too little smoothing leads to superfluous corners, whereas excessive smoothing causes the disappearance of true corners. Early approaches (see [38] for an overview) relied on a single scale, which created difficulties for curves containing both large and small features.

Later work addressed the problem of individual curves containing features at various scales. For example, Rattarangsi and Chin [30] developed a scale-space approach for corner detection. A digital Gaussian filter is repeatedly applied to the curvature data, and the maxima of curvature are identified for each scale. Curvature maxima that persist across multiple scales indicate corner points. Although the method can find features at multiple scales, it is still necessary to define the range of scales to be considered. Also, the approach produces false corners when there is quantization error. For example, corner points may be found on accurate digital circles. Lee et al. [23] developed a multi-scale corner detection algorithm based on the wavelet transform. This approach produces fewer false corners than Rattarangsi and Chin's, and is less computationally expensive. Sezgin has applied a multi-scale approach to sketches [33]. As described below, his work suggests that curvature data alone is inadequate for segmenting hand-drawn pen strokes.

Yu has applied a curvature-based method to the problem of segmenting hand-drawn pen strokes [42]. The method is based on a "mean shift" technique in which the curvature and tangent angle are simultaneously smoothed. The resulting segmentation is compared to the original ink, and if the fit is not precise, the stroke is recursively subdivided until a precise fit is achieved. In a post-processing phase, lines and arcs are merged, but the criteria for this are not defined. Yu and Cai [43] present a related method that first attempts to fit a stroke with a single primitive. If the fit is poor, the stroke is segmented at the point of highest curvature, and the two pieces are recursively processed. Again, segments are merged in a post-processing phase, but the criteria for doing this are not specified. As described in Section 6, our approach is substantially more accurate than this one.

Wolin and Hammond [40] present ShortStraw, a segmentation technique based on the notion of "straw" distance. The straw distance at a point is defined at the distance between the endpoints of a window of points centered on that point. Corners are identified as points with minimum straw distances. The technique is limited to pen strokes comprised solely of line segments. In more recent work, Xiong and LaViola [41] have built upon this technique to improve its accuracy and extend it to strokes with curves and arcs. As described in Section 6, our approach is more accurate than both of these state-of-the-art approaches.

The earliest report of using pen speed for segmenting that we have been able to find is the work of Herot [11]. His system found corners by identifying points at which pen speed was a minimum.

The author reported that the system did not work well for all users and he concluded that the program contained a "model of human sketching behavior that fit some users more closely than others."

Agar and Novins [1] have developed a segmenter for polygons. The system identifies segment points as a polygon is drawn, and provides immediate feedback to the user. The approach is based on examining the time intervals between mouse movement events. If the mouse is stationary for more than 30 ms, the location is taken to be a segment point. This approach is analogous to our pen speed approach. However, because it requires that the mouse be paused at each corner, the approach is likely to work well only at very sharp corners. Additionally, the approach can handle only line segments and not arcs.

Sezgin et al. [32] presented a technique that used speed and curvature to segment hand-drawn pen strokes. Segment points were located at points of minimum speed and maximum curvature. This work demonstrated the usefulness of speed data for segmenting, and demonstrated that curvature data alone is inadequate. The technique is suitable for segmenting pen strokes into sequences of line segments, but cannot handle arcs. Curved regions of the pen stroke are not segmented, but rather are represented by b-splines. The approach presented here can handle pen strokes consisting of both lines and arcs. Also, the technique in [32] iteratively adds segment points until the error of fit between the line segments and the raw ink is less than a threshold. Our approach is far less concerned with the error of fit, as a tight fit to the ink may not be what was intended. Finally, the method in [32] lacks the ability to refine the segmentation through segment merging. As described in Section 6, our approach is substantially more accurate than this one.

As a variant of the approach in [32], Sezgin explored the use of multi-scale methods for selecting speed minima and curvature maxima [33]. However, he found that unless the pen strokes were exceptionally noisy, there was little benefit in doing so.

Qin et al. [29] have also developed a curvature-based approach that makes use of pen speed information. Segment points are identified as points of maximum discontinuity of the smoothed tangent angle. Very large discontinuities are always considered segment points. Smaller discontinuities must exceed an adaptive threshold that is inversely proportional to pen speed. Thus, the threshold is higher for low pen speed. With our approach, by contrast, slower pen speed provides increased evidence of a segment point. While their method identifies segment points as points of maximum curvature, ours identifies them as points of minimum pen speed. Their segment points are thus a subset of the dominant-points produced by methods such as [38], while ours are not. Also, our method uses merging and splitting to refine the segmentation, while their method has only limited refinement capabilities: segment points are eliminated if they are too close together.

Kim and Kim [21] present a more recent algorithm that uses curvature and pen speed information to segment pen strokes. Pen strokes are resampled to produce uniformly spaced data points. Curvature is initially computed in terms of the angle between consecutive data points. An examination of the local geometry is then used to determine a larger region of support for the curvature at a point. Segment points are identified as maxima (in absolute value) of curvature above a threshold determined by the pen speed. As described in Section 6, our approach is substantially more accurate than this one.

Simhon and Dudek [36] use a hidden Markov model to refine the shape of a pen stroke to match geometric constraints learned from training examples. The approach does not compute explicit segmentation, but instead refines a hand-drawn shape to make it "look" like one used to train the system.

Hse et al. [12] use two kinds of templates to segment pen strokes. One kind specifies the number of line and elliptical segments that comprise a shape, the other also includes the order of those segments. Dynamic programming is used to optimally segment a stroke to match a specified template. Hse and Newton [13] combine this approach with a shape recognizer to automatically select the correct template. The approach is accurate, but is limited to shapes with predefined templates and requires that pen strokes can be recognized prior to segmentation. Deufemia and Risi [4] present a related approach in which a grammar and LR parser are used to dynamically select templates to segment the strokes in a sketch.

Most of the above approaches operate by first locating segment points, and then defining the segments between them. Dudek and Tsotsos [5] have turned the problem around by first looking for the segments. Their approach is called "curvature-tuned smoothing." The method uses energy minimization to compute an approximation curve that best matches the input curve while at the same time attempting to maintain a desired curvature. If an approximation with sufficiently low energy cannot be found, the approximation curve is subdivided and the process is iterated. This process can be performed with different values of the desired curvature to find regions of the input curve that have various curvatures. Each such region constitutes a segment. A given data point in the input curve may belong to different segments having different values of the curvature, resulting in overlapping segments. This approach may not be suitable for sketch understanding, as most shape recognition techniques assume that segments do not overlap.

SpeedSeg is intended to support shape recognizers that rely on structural shape descriptions such as those in [2,10,9,24]. Such recognizers are particularly useful in applications in which it is necessary to identify the individual parts of a shape. For other types of applications, there are recognizers that do not require segmentation. For example, some recognizers use feature-based representations of shape (e.g., [31,8]). However, because classification relies on aggregate features of the pen strokes, these approaches sometimes have difficultly differentiating between similar shapes. Image-based recognizers (e.g., [39,20]) also do not require segmentation, but they have difficulty when the relative sizes of the different parts of a shape vary.

Because SpeedSeg is intended for use in constructing structural shape descriptions to support recognition, we segment into line and arc segments. This is appropriate for many shapes and symbols that are commonly used in science and engineering. For applications with more complex geometric primitives, the technique in [27] can be used to recognize a variety of primitive shapes such as spirals and helixes. This approach does not perform segmentation, but instead assumes that each pen stroke comprises a single primitive.

There is a growing body of work in the area of sketch beautification [25,16,15]. For example, Igarashi et al. [15] have developed a system that can infer geometric constraints, such as perpendicularity and congruency, and can automatically adjust the sketch so that the constraints are satisfied. SpeedSeg can be viewed as having limited beautification capabilities because it transforms raw ink into straight lines and round arcs. Furthermore, it can serve as a foundation for tools that do focus on beautification. For example, Igarashi's system assumes that each pen stroke is a single line segment, but in principle could be extended to operate on the segments computed by SpeedSeg.

3. Segmenting technique

Segmentation is the process of decomposing a pen stroke into the constituent geometric primitives. For the domains of interest to us, the primitives consist of lines and arcs. Our SpeedSeg technique relies extensively on pen speed information for identifying the locations of intended segment points. It also considers the final shape of the ink, by using curvature information to find other segment

points. To achieve high accuracy, SpeedSeg monitors its own performance and improves the segmentation when necessary.

To begin the segmentation process, an initial set of candidate segment points is identified. This set includes the points on the pen stroke at which speed is a minimum or curvature is a maximum (the complete criteria are described below). The ink between each pair of consecutive segment points is called a segment, and is classified as a line or an arc, depending on which best fits the ink.

Although the initial segmentation is often reasonably accurate, feedback can be used to improve the accuracy. If the initial segmentation does not accurately match the original ink, segments are either merged or split to improve the fit. For example, if two adjacent segments form pieces of the same arc, it is likely that they were intended to be so. In this case, the two are merged into a single arc segment. Conversely, if a particular line or arc is a poor fit for the ink, additional segment points are considered. This situation often occurs when there is a smooth change in the sign of curvature, for example, when moving from one lobe of an "S" shape to the other as shown in Fig. 6. This kind of transition can be made without slowing the pen, and thus is not detected as a speed minimum. Consequently, if a segment is a poor fit for the ink, points at which the curvature changes sign are considered as additional candidate segment points. Curvature sign is an unreliable indication of a segment point, thus such points are considered only when there is evidence that extra points are needed.

The sections that follow describe the various steps of the segmentation process including: initial processing of the ink, identification of segment points, fitting of the segments, and merging and splitting.

3.1. Initial processing of the ink

SpeedSeg is designed to work with a digitizing tablet and stylus or other similar device that provides time-stamped coordinates. For example, we have used Wacom Cintiq and Intous II tablets, and Tablet PCs. During the initial processing phase, we use the time-stamped coordinates to compute pen speed and curvature. The first step is to construct the arc length coordinate of each point. Arc length is measured along the path of the pen stroke, and is computed in the obvious way by summing straight line distances:

$$d_i = \sum_{j=1}^{i} \|\vec{P}_j - \vec{P}_{j-1}\| \tag{1}$$

where $\vec{P}_j$ is the coordinates of the jth data point. The first data point has index $j=0$ and $d_0 = 0$.

We then use a centered finite difference approach to compute pen speed:

$$s_i = \frac{d_{i+1} - d_{i-1}}{t_{i+1} - t_{i-1}} \tag{2}$$

where t_i is the time-stamp of the ith point. The speed at the first and last point of a pen stroke is taken to be equal to the speed at the second and penultimate points, respectively. Often, there is noise in the pen speed signal. To correct this, we apply a simple smoothing filter. The speed at each point is averaged with that of the two points on either side. After averaging, the first two and last two points in the pen stroke are assigned speeds equal to those of the third and third to last points, respectively.

There are various ways of computing signed curvature. (Note that SpeedSeg uses the *absolute value* of the curvature to locate candidate segment points.) For example, one could use the standard formula from analytic geometry [26]:

$$C = \frac{\dot{x}\ddot{y} - \ddot{x}\dot{y}}{[\dot{x}^2 + \dot{y}^2]^{3/2}} \tag{3}$$

where the dot indicates differentiation with respect to the arc length, s. For digital data, the derivatives are typically evaluated using a finite difference technique. For the purposes of identifying segment points, however, the resulting curvature data would require a significant amount of smoothing, for example, by means of a Gaussian filter [30].

As an alternative approach, we compute curvature as the derivative of the tangent angle, θ, with respect to arc length:

$$C = \frac{\partial \theta}{\partial s} \tag{4}$$

We use this approach for several reasons. First, our system already computes an accurate tangent, which is used for other purposes. Second, this method naturally smoothes the data, so that no additional smoothing is needed.

To construct the tangent at a given point, we first construct a least squares line fit to a window of data points centered on that point. Using a window of points has the effect of smoothing noise. Some of the noise comes from minor fluctuations in the drawing, while other noise comes from the digitizing error of the input device. The larger the window, the greater the smoothing effect. We have found that a window of 11 points (five on other side of the point in question) provides adequate smoothing without loss of essential information about the shape.

If the least squares line fit is an accurate fit for the window of points, the line is used as an approximation of the tangent. Accuracy is defined as the average distance from the points to the line. If this is less than 10% of the arc length of the window of points, the line fit is deemed acceptable. Otherwise, a least squares circle fit is constructed, and the tangent is taken from the circle. In either case, the tangent direction is selected so as to align with the local direction of the pen motion. (The line fit is attempted first to avoid numerical issues that occur when computing a least squares circle fit to data forming a nearly straight line.)

To compute the rate of change of the tangent angle, we could numerically differentiate the tangent angle data, but this would again require smoothing. Thus, we again use a least squares line fit. In this case, we consider the graph of the tangent angle vs. the arc length.[1] The slope of the least squares line gives the curvature in units of radians per pixel. Here again, when computing the least squares line, we use a window of 11 points as a means of smoothing the data.

Fig. 2 shows signed curvature data computed with our technique and the traditional approach from Eq. (3). (The data is for the square in Fig. 1.) Comparison of traces **A** and **B** shows that our approach produces significantly smoother curvature data. To estimate the amount of smoothing our approach achieves, we repeatedly applied a Gaussian filter to the data from Eq. (3) (trace **B**) until the smoothing was comparable to that of our data (trace **A**). In each application of the filter, the new value at a data point was taken to be 0.5477 times the current value plus 0.2236 times the current value on either side [3]. We found that between 5 (trace **C**) and 10 (trace **D**) applications of the filter resulted in an equivalent amount of smoothing.

We have found that our approach for calculating curvature works well in practice. In fact, this approach is similar in spirit to the way draftspersons used to compute graphical derivatives in the era before computers [37]. In some sense, we are smoothing the way a draftsperson would by eye. As Fig. 2 shows, however, our approach is comparable to the traditional calculation (Eq. (3)) combined with Gaussian smoothing. Thus, if desired, one could directly implement our segmentation approach using the more traditional technique.

[1] Care is taken to avoid false discontinuities in the tangent angle. For each point, we adjust the angle by adding or subtracting multiples of 2π until it differs in absolute value by less than 2π from the angle of the previous point.

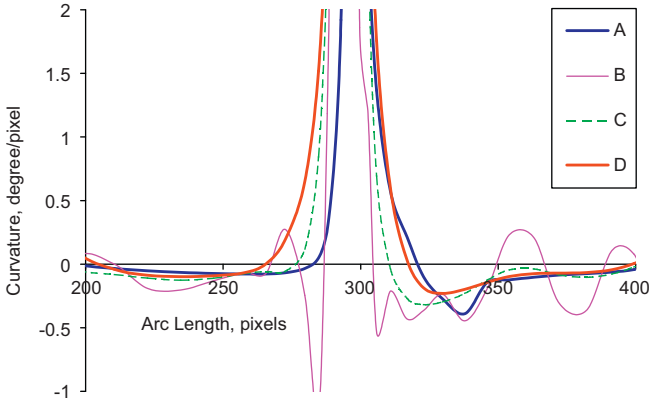

Fig. 2. Signed curvature data from a square. **A**: Curvature-based on tangent angle. **B**: Curvature computed with Eq. (3). **C**: Data from **B** after five applications of a Gaussian filter. **D**: Data from **B** after 10 applications of a Gaussian filter.

3.1.1. Least squares line and arc fitting

Least squares line and arc fitting is used for multiple purposes in our system. As described above, it is used for computing both tangents and curvature. It is also used for fitting lines and arcs to the segmented ink. For completeness, this section provides a review of the least squares techniques we use.

For sake of efficiency and simplicity, we use a linear least squares fit. A line is defined as

$$y = Ax + B \tag{5}$$

Minimizing $\sum_{i=1}^{n}(Ax_i+B-y_i)^2$ results in the usual regression equation:

$$\begin{bmatrix} n & \sum x_i \\ \sum x_i & \sum x_i^2 \end{bmatrix}\begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \end{bmatrix} \tag{6}$$

where n is the number of data points and the (x_i,y_i) are the coordinates of the data points. The linear least squares technique fails if the line is nearly vertical, because error is defined as the vertical distance from a data point to the line. To avoid this, if the data points have little variation in x, we instead fit the data to the line $x = Ay + B$. We could have used a non-linear least squares fit in which the error is defined as the minimum (perpendicular) distance from a data point to the line. Such an approach would be more accurate and would not require special treatment of vertical lines, but would be more expensive computationally.

For fitting circles, we again use a linear least squares approach. A circle is defined as

$$x^2 + y^2 + 2ax + 2by + c = 0 \tag{7}$$

where $(-a,-b)$ is the center of the circle, and the radius is $r = \sqrt{a^2+b^2-c}$. Minimizing the total squared error computed as $\sum_{i=1}^{n}(x_i^2+y_i^2+2ax_i+2by_i+c)^2$ results in the regression equation:

$$\begin{bmatrix} 2\sum x_i^2 & 2\sum x_i y_i & \sum x_i \\ 2\sum x_i y_i & 2\sum y_i^2 & \sum y_i \\ 2\sum x_i & 2\sum y_i & n \end{bmatrix}\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \sum -(x_i^2+y_i^2)x_i \\ \sum -(x_i^2+y_i^2)y_i \\ \sum -(x_i^2+y_i^2) \end{bmatrix} \tag{8}$$

If the ink is nearly straight, the matrix becomes ill conditioned. As a remedy, we solve Eq. (8) using singular value decomposition [28]. An alternative approach would be to use a more sophisticated least squares circle fitting technique as described in [34].

When evaluating the quality of fit, we use an average error. For non-vertical lines (those described by Eq. (5)), the *error of fit* is

$$e = \frac{1}{n}\sum_{i=1}^{n}|Ax_i+B-y_i| \tag{9}$$

For vertical lines, or those that are nearly so, the absolute value term becomes: Ay_i+B-x_i. For circles, the error of fit is

$$e = \frac{1}{n}\sum_{i=1}^{n}\left|\sqrt{(x_i+a)^2+(y_i+b)^2}-r\right| \tag{10}$$

3.2. Candidate segment points

Once the initial processing of the ink is completed, the next step is to compute the set of initial candidate segment points. The first and last points on a pen stroke are always included in the initial set. The remaining segment points are identified by examining speed and curvature data. This process depends on the five parameters listed in Table 1. As discussed here, and described in detail in Section 6, the accuracy of SpeedSeg is relatively insensitive to three of these five parameters (P_{ST}, P_{CT}, and P_{CZT}). The table includes the default values of the parameters, which were obtained empirically.

Pen speed has proven to be an effective indicator of intended segmentation, as segment points often occur at locations at which pen speed is a local minimum. Consider, for example, the sketch of a pivot in Fig. 3(a). This sketch, which was drawn as a single pen stroke, was intended to be three lines and an arc (Fig. 3(b)). Fig. 4 shows the speed profile for the pen stroke. The intended segment points correspond to local speed minima as indicated by circles. There are, of course, other speed minima that do not correspond to

Table 1
Parameters controlling the generation of segment points.

Parameter	Symbol	Default value
Speed threshold	P_{ST}	25%
Curvature threshold	P_{CT}	0.75°/pixel
Curvature-speed threshold	P_{CST}	80%
Curvature-sign (zero) threshold	P_{CZT}	0.1°/pixel
Point merge trigger	P_{PMR}	7

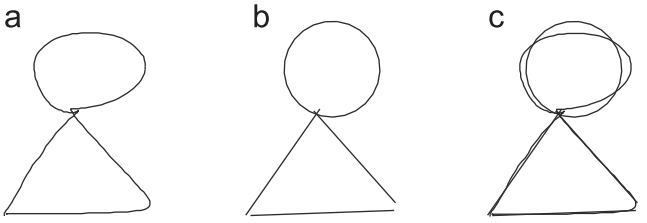

Fig. 3. A hand-drawn pivot symbol. (a) Raw ink. (b) Segmented ink. (c) Raw and segmented ink overlaid.

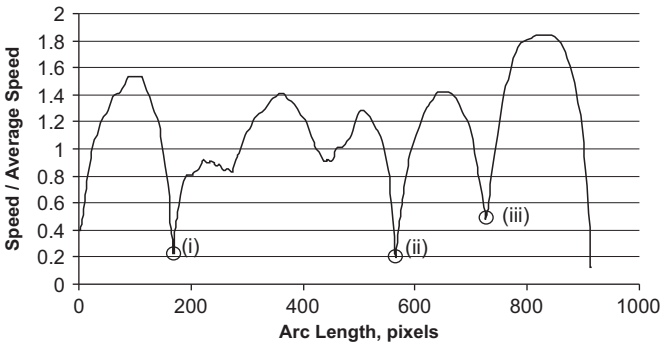

Fig. 4. Pen speed normalized by the average speed for the pivot in Fig. 3. Intended segment points are indicated by circles.

intended segment points, but these are distinguishable by their higher speed.

Our approach, therefore, is to locate segment points at speed minima that are slower than a threshold, P_{ST}. We select the threshold as a fraction of the average speed along the pen stroke. Average speed is computed in the usual way as the total arc length divided by the total elapsed time. In practice, a threshold between 25% and 100% of the average speed works well. In fact, in User Study 1 presented in Section 5.1 there was little variation in the overall accuracy when the threshold was varied over this range. Larger values of the threshold decrease the number of intended segment points that are missed, while smaller values decrease the number of unintended segment points that are selected. (The ordinate in Fig. 4 directly corresponds to possible values of P_{ST}.)

We typically, use a small threshold (25%) because very low pen speed is a clear indication of an intended segment point. If a speed minimum is above the threshold, the point may still be a segment point, but additional information is required to be certain. In this case, we rely on the curvature of the ink. In Fig. 4, for example, segment points (i) and (ii) are detected with a threshold of 25%. Segment point (iii) is above this threshold, but as shown in Fig. 5, this point corresponds to a maximum of curvature, which provides additional evidence about the existence of a segment point.

One approach for locating segment points would be to identify points that are both a minimum of speed and maximum of curvature. In practice, we have found it adequate to simply select points that: (a) are a maximum of curvature, (b) have speed less than threshold P_{CST}, which is defined as a fraction of the average speed along the pen stroke, and (c) have curvature greater than threshold P_{CT}. The last condition eliminates false segment points resulting from small curvature peaks. For example, the curvature of a nearly straight line often fluctuates in sign, resulting in many small curvature peaks. The third condition prevents such points from being selected as segment points. We have found that a value of 80% for P_{CST} and a value of 0.75°/pixel for P_{CT} work well. As described in Section 6, P_{CST} does have a strong influence on accuracy but P_{CT} does not.

The speed-based and curvature-based segment points are always included in the initial set of segment points. There is a third class of segment points that is not considered initially. These are the points at which the curvature changes sign. We define three qualitative "signs" for curvature: positive if the magnitude is greater than P_{CZT}, negative if the magnitude is less than $-P_{CZT}$, and zero otherwise. The default value for P_{CZT} is 0.1°/pixel.

A change in curvature sign is an unreliable indication of an intended segment point, thus such points are considered only when the other segment points do not result in a good fit for the ink. For example, it is common for there to be a change in curvature sign on each side of a 90° corner as shown in Fig. 7. It is clear that such changes in curvature sign do not correspond to intended segment points. Consequently, segment points based on curvature sign are not part of the set of initial candidate segment points. Instead, they are considered during the splitting process described in Section 3.4. In essence, a change in the sign of curvature is insufficient evidence to decide that a segment point was intended. Instead, additional information about the gross shape of the ink is needed. This information comes from examining how well the initial segmentation fits the ink.

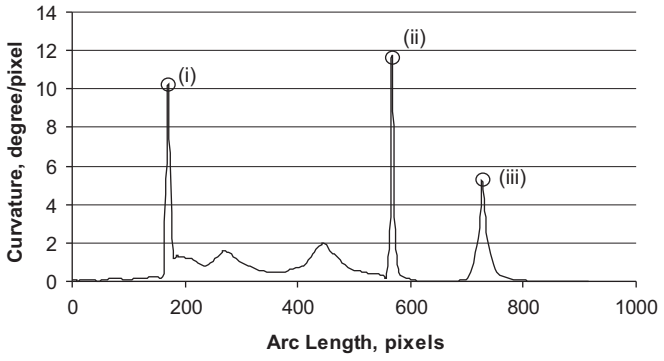

Fig. 5. Ink curvature for the pivot in Fig. 3.

Due to noise, it is possible for there to be small clusters of closely located segment points. For example, there may be two speed minima that are separated by only a few data points, or there may be a speed minimum near a curvature maximum. Thus, once the speed and curvature segment points are identified, the data is filtered to eliminate nearly coincident segment points. If a point is within P_{PMR} (default value 7) data points of a subsequent segment point, it is eliminated.

3.3. Fitting segments

Once the initial set of candidate segment points has been identified, the next step is to fit primitives to the segments. Least squares line and circle fits are constructed for the segment between each pair of consecutive segment points. The segment is typically classified by whichever shape fits it with the smallest error of fit (Section 3.1.1). In practice, it is common for nearly straight lines to be accurately fit by an arc with a large radius. In fact, even a straight line can be perfectly fit by an arc with infinite radius. Thus, even if a segment is best fit by an arc, it is classified as such only if it would represent at least one tenth of a circle (36°).

If a segment is classified as a line segment, the end points of that line segment are determined by constructing perpendiculars from the first and last data points to the least squares line. Similarly, for arcs, the end points are determined by constructing radial lines through the first and last data points. This approach may result in gaps between segments where no gaps existed in the original ink. For the purposes of recognition, however, this typically does not pose a problem because tolerances are often used when evaluating topology. (Section 6 presents recognition accuracy results for a shape recognizer that uses SpeedSeg.) For beautification, by contrast, it would be necessary to adjust the end points so as to preserve the original connectivity of the segments.

3.4. Merging and splitting

Once the initial segments have been computed, a quality control process is initiated. The segments are compared to the original ink, and are then merged, split, and deleted as necessary. In this fashion, feedback is used to improve the accuracy. This post-processing depends on seven parameters listed in Table 2. The table includes the default values of these parameters, which were obtained empirically. As discussed in Section 6, the accuracy of SpeedSeg is relatively insensitive to the specific values of most of these parameters.

If there is a very short segment adjacent to a long one, frequently the short one was unintended. Thus, if a segment is shorter than

Table 2
Parameters controlling splitting and merging of segments.

Parameter	Symbol	Default value
Segment merge length trigger	P_{SMLR}	20%
Segment merge error threshold	P_{SMET}	110%
Parallelism threshold	P_{PT}	0.75
Relative end length trigger	P_{RELR}	10%
Absolute end length trigger	P_{AELR}	15 pixel
Segment split error trigger	P_{SSER}	7 pixel
Segment split error threshold	P_{SSET}	65%

a fraction P_{SMLR} (default value 20%) of the length of an adjacent segment, SpeedSeg attempts to merge them. The program constructs a new segment containing all of the data points of the two original segments. The type of this new segment is forced to be the same as that of the longer of the original two. For example, if a short line segment is adjacent to a long arc, the program attempts to join them into a single arc segment. If the error of fit (Section 3.1.1) of the new segment is no greater than P_{SMET} (default value 110%) times the sum of the fit errors of the original two segments, they are discarded and replaced with the new one. Otherwise, the new segment is discarded.

A special case of this procedure is applied to the ends of each pen stroke to eliminate "hooks," discontinuities that occur as the stylus gains or loses contact with the drawing surface. If the first or last segment in a stroke is shorter than P_{AELR} (default value 15) pixels, it is discarded. Similarly, if the first or last segment is shorter than a fraction P_{RELR} (default value 10%) of the average length of the first or last three segments, respectively, it is discarded. An alternative approach would be to "dehook" strokes prior to segmentation using an algorithm such as the one in [17].

If adjacent segments are of the same type, SpeedSeg examines if they might reasonably be interpreted as the same segment. For example, if two arcs are adjacent, the program computes a new arc containing the data points from the two original arcs. Here again, the merge is accepted only if the new error of fit is no greater than P_{SMET} times the sum of the original errors of fit. (Note that the same parameter P_{SMET} governs all segment merging computations.) The program considers merging two segments only if their drawing directions are consistent. For arcs, the requirement is that they both be drawn in the same sense, i.e., both clockwise or both counterclockwise. Similarly, for line segments, the program constructs unit vectors from the lines, and attempts a merge only if the dot product of these unit vectors is greater than P_{PT} (default value 0.75).

If a particular least squares line or arc does not fit the ink well, SpeedSeg attempts to improve the fit by introducing a segment point based on a change in the sign of the curvature. The program splits a segment in this fashion if the fit error is greater than P_{SSER} pixels. In other words, if on average the data points are at least P_{SSER} pixels from the least squares line or arc, the program attempts to split the segment. The default value of P_{SSER}, which is seven pixels, was empirically determined to work with our hardware for digitizer resolutions of 1024×768 and 2048×1536, and would likely require tuning for different hardware.

Typically there are only a few curvature-sign segment points in any given segment. Thus, it is feasible to exhaustively consider each of them. The program considers splitting the segment with each of the curvature-sign segment points, one at a time. The best choice is the one in which the sum of the fit errors for the two new segments is minimum. If this minimum is less than a fraction P_{SSET} of the original fit error, the new segmentation is retained, otherwise it is rejected. The default value of P_{SSET} is 65%, which is designed to require significant improvement in the fit before a new segment point is added.

Fig. 6 shows an example of how curvature-sign points are used. In the initial segmentation, curvature-sign points are excluded, and the stroke is segmented into a single arc segment. Because the fit is poor, the program tests both curvature-sign points in the middle of the curve and finds that an improved segmentation can be achieved. The result is shown in Fig. 6b.

Fig. 7 shows why curvature-sign points are not considered unless the initial segmentation is poor. Fluctuations in approximately straight portions of the pen stroke result in a large number of curvature-sign changes. These are clearly not intended to be segment points.

Post-processing begins with an application of the merging routine that fixes hooks at the start and end of each stroke. The general routine for merging segments is then applied, followed by an application of the splitting routine. A small gain in accuracy is typically achieved with a second application of the splitting routine followed by the merging routine.

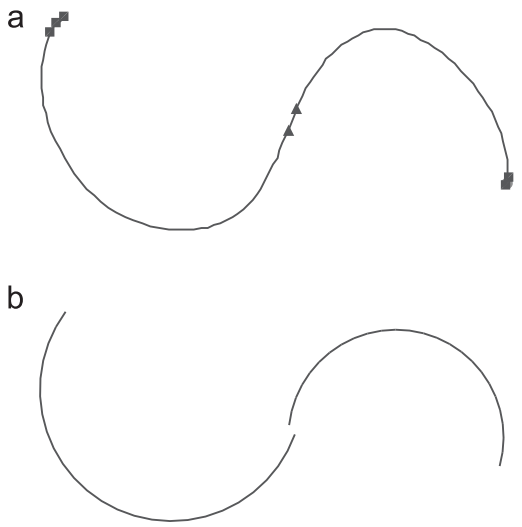

Fig. 6. (a) Candidate segment points for an "s-curve". Circle = speed segment point, square = curvature (magnitude) segment point, triangle = curvature-sign segment point. (b) Final segmentation—two arc segments.

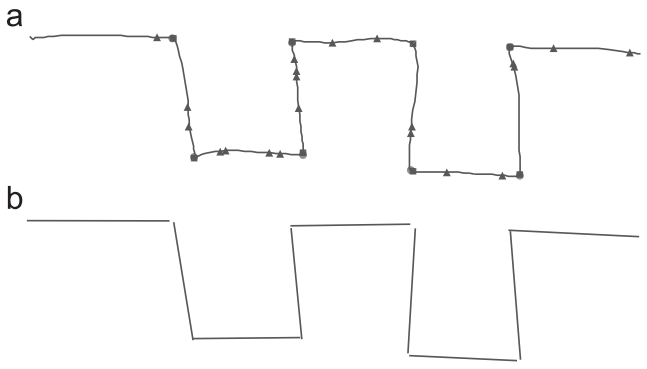

Fig. 7. (a) Candidate segment points for a square wave. Circle = speed segment point, square = curvature (magnitude) segment point, triangle = curvature-sign segment point. (b) Final segmentation—nine line segments.

4. Using the system

We have used SpeedSeg with a Wacom Intous II tablet, a Wacom Cintiq LCD tablet, and a Wacom-based Tablet PC. (SpeedSeg, which is implemented in C++, directly communicates with the Wacom driver to acquire data at the maximum possible sampling rate.) With the latter two devices, the user draws directly on the display, and virtual ink is rendered directly under the stylus tip. With the Intous II, the user draws on the tablet, and virtual ink is rendered on the display. As a means of providing better feedback to the user, the Intous II can also be used with an "inking" stylus. In this case, paper is placed over the tablet and the stylus tip leaves physical ink. SpeedSeg provides an option for displaying the raw ink, the segmented ink, or both. For the latter two cases, the current pen stroke remains in its raw form until the stylus is lifted, and then the segmented ink is instantly displayed.

SpeedSeg provides editing gestures enabling the user to correct segmentation errors by adding and removing segment points. Segment points are added by pressing the button on the barrel of the stylus and drawing a line across the ink at the desired location of the new segment point. Similarly, with the button pressed, drawing a circle around a set of segments will merge them together. Circling the

beginning or ending of a stroke will restore ink removed by dehooking. Using the eraser end of the stylus, the user can also erase individual segments or entire pen strokes. A few strokes of the eraser remove a segment, while many strokes remove the entire pen stroke.

Although the default parameter values produce acceptable results for many users, it is possible to tune the system to achieve optimal performance for a specific user. (Except were explicitly indicated, all accuracy results reported in this article were obtained without tuning.) To do this, the user first draws a set of shapes and uses the editing gestures to correct any segmentation errors. Because the default parameters usually achieve good accuracy, little editing is typically needed. SpeedSeg then uses hill climbing to adjust the parameter values to maximize the number of perfectly segmented shapes. When comparing the computed segmentation to the manually edited correct segmentation, segment points are said to match if the distance between them is 20 pixels or less. This threshold is needed because different parameter values may result in slightly different locations for the same segment point. This may occur, for example, at rounded corners where the segment point can move slightly along the corner depending on the particular values used.

Optimization over a 12-dimensional search space is computationally expensive. Fortunately, SpeedSeg's accuracy is only weakly dependent on most of the processing parameters. To reduce computation, we optimize over only five parameters, which include: the curvature-speed threshold (P_{CST}), the segment merge error threshold (P_{SMET}), the point merge trigger (P_{PMR}), the absolute end length trigger (P_{AELR}), and the speed threshold (P_{ST}). As demonstrated in Section 6, the first four parameters are the ones that have the most influence on accuracy. The speed threshold affects accuracy only weakly, but was included to provide additional opportunities for tuning.

To construct the starting point for hill climbing, SpeedSeg computes the locally optimal value for each parameter. This is accomplished by evaluating the accuracy as each parameter is varied between one fourth and four times its default value, with the other 11 parameters held at their default values. In this fashion, SpeedSeg samples the accuracy for 11 values of each parameter. The set of locally optimal values constitutes the starting point. On each iteration of hill climbing, SpeedSeg randomly selects one of the five optimized parameters to change. A computational coin flip then determines if the parameter should be increased or decreased by 2.5%.

In our experiments, working from the locally optimal starting point, the hill climbing typically converged in 20 or fewer iterations. Thus, a total of 152 iterations are needed to tune SpeedSeg for a given user: 132 iterations to obtain the starting point, followed by 20 iterations of hill climbing. On a training set of 50 shapes, the tuning process takes about 75 s (excluding file I/O) on an HP TC4400 Tablet PC with a 2.0 GHz Intel T7200 Core 2 Duo Processor and 2 GB of memory. (On average, it takes about 10 ms to process a pen stroke once.)

5. User studies

We conducted three user studies to test SpeedSeg's accuracy. In each study, participants were asked to draw the set of shapes shown in Fig. 8. The pivot and omega symbols were designed to test SpeedSeg's accuracy on shapes that include both line and arc segments, while the wave symbol tests accuracy on sequences of arcs. The remaining symbols evaluate accuracy on shapes containing line segments of various relative lengths and vertex angles.

The first user study examined the sources of segmentation errors as the speed threshold is varied. The second study evaluated the effect of symbol size and input device resolution on accuracy. The third study evaluated the algorithm as an interactive sketch input technique. In particular, this study examined how visual feedback of the pen stroke segmentation influences accuracy. In all studies, the participants had no prior experience with SpeedSeg, and thus these studies represent a worst-case test of the system's accuracy.

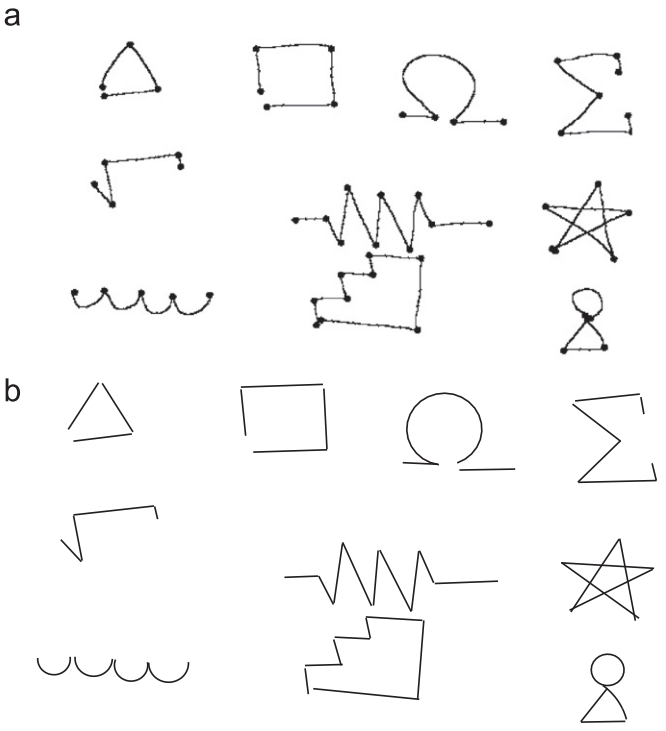

Fig. 8. The 10 shapes used in the user studies: triangle, square, omega, sigma, square root, spring, star, wave, stepped-block, pivot. (a) Raw ink from one of the user study participants. Dots indicate the locations of the computed segment points. (b) Segmented ink.

5.1. Study 1

The first study evaluated the types of segmentation errors that occur as the speed threshold (P_{ST}) is varied. The study included five participants who had no experience with SpeedSeg, but at least some experience using a PDA or digitizing tablet. The participants were asked to draw the 10 symbols in Fig. 8 four times each at a size of approximately 3 cm. (User Study 2, described below, explored accuracy as a function of symbol size.) The participants were instructed to draw accurately but naturally, and were informed that the experiment was intended to evaluate the accuracy of our segmenter. Because the participants were allowed to draw naturally, some drew some of the shapes (mostly pivots, but other shapes as well) using multiple strokes. Overall, 15.5% of the shapes were multi-stroke.

Data was collected using an Intous II tablet with a paper overlay and an inking stylus (see Section 4). The digitizer resolution was set to 1024×768 pixels. To avoid biasing the participants, they were given no feedback about how well SpeedSeg performed. They could view the physical ink on the paper overlay and the virtual ink on the computer display, but the segmentation results were not displayed. Participants were given a few minutes to become familiar drawing with the digitizing tablet and stylus before providing data for the study.

There were variations in the way participant drew the various shapes. For example, the number of intended "wiggles" in the spring symbol varied from one participant to the next. Table 3 tabulates the number of intended segment points ("num segment points") for each participant, which was typically about 233 for the complete set of 40 examples.

Table 3 summarizes the types of segmentation errors that occurred using SpeedSeg's default parameter values. Errors are

Table 3
Study 1—accuracy for a speed threshold of 25%. False negatives = missing candidates + mistakenly merged + mistakenly dehooked.

Participant	1	2	3	4	5	Average
Num segment points	230	229	244	229	233	233
Missing candidates	11	5	17	0	2	7
Mistakenly merged	2	1	2	0	0	1
Mistakenly dehooked	0	1	3	0	0	0.8
False negatives	13	7	22	0	2	8.8
False positives	0	3	0	0	2	1
True positives	217	222	222	229	231	224.2
Precision (%)	100.0	98.7	100.0	100.0	99.1	99.6
Recall (%)	94.4	96.9	91.0	100.0	99.1	96.3
F-measure	0.971	0.978	0.953	1.0	0.991	0.97.9
All-or-nothing (%)	82.5	82.5	67.5	100.0	90.0	84.5

Table 4
Study 1—accuracy for a speed threshold of 100%. False negatives = missing candidates + mistakenly merged + mistakenly dehooked.

Participant	1	2	3	4	5	Average
Num segment points	230	229	244	229	233	233
Missing candidates	0	0	0	0	0	0
Mistakenly merged	2	0	1	0	0	0.6
Mistakenly dehooked	0	1	4	0	0	1
False negatives	2	1	5	0	0	1.6
False positives	12	13	3	1	5	6.8
True positives	228	228	239	229	233	231.4
Precision (%)	95.0	94.6	98.8	99.6	97.9	97.1
Recall (%)	99.1	99.6	98.0	100.0	100.0	99.3
F-measure	0.970	0.970	0.984	0.998	0.989	0.982
All-or-nothing (%)	72.5	75.0	87.5	97.5	90.0	84.5

expressed in terms of the number of false negatives and false positives. False negatives can occur for one of three reasons: (1) no candidate segment point was found, (2) a candidate was found but was later eliminated by merging of the two adjacent segments, or (3) a candidate was found but was later eliminated during the removal of hooks at the ends of the pen stroke. False positives are points that were not intended as segment points, but which were labeled as such by the program.

We characterize overall segmentation accuracy in terms of precision, recall, and f-measure.[2] Precision is the fraction of the hypothesized segment points that are true segment points:

$$\text{Precision} = \frac{\text{True positives}}{\text{True positives} + \text{False positives}} \tag{11}$$

Recall is the fraction of the actual (intended) segment points that were correctly located:

$$\text{Recall} = \frac{\text{True positives}}{\text{True positives} + \text{False negatives}} \tag{12}$$

F-measure is the simple harmonic mean of precision and recall:

$$\text{F-measure} = \frac{\text{Precision} * \text{Recall}}{\frac{1}{2}(\text{Precision} + \text{Recall})} \tag{13}$$

Most of the segmentation errors occurred because no candidate segment point was identified. On average, there were seven such errors for each set of 40 examples. Far fewer points were missed because of segment merging or dehooking; there was on average 1.8 such errors for each set of 40 examples. The dehooking errors typically occurred when participants drew the square root and sigma symbols with very small serifs.

We also characterize performance in terms of "all-or-nothing" accuracy, which is the fraction of the symbols that had no segmentation errors of any kind. As indicated in Table 3, SpeedSeg achieved an average all-or-nothing accuracy of 84.5%. Symbols contained an average of about six segment points, thus there are multiple ways for there to be an error in a given symbol. Consequently the all-or-nothing accuracy is lower than the f-measure, which is based on the number of correct segment points. When the algorithm did err, there was often more than one error per symbol. In the data in Table 3, for example, there were typically 1.6 segmentation errors (false negatives and false positives) per incorrectly segmented symbol.

To evaluate the influence of the speed threshold on accuracy, we recomputed the segmentation using a larger value of the speed threshold, with all other parameters held to their default values. In Table 3, a threshold value of 25% of the average speed was used, but in Table 4 the threshold was increased to 100%. With the lower threshold, there were on average 8.8 false negatives and one false positive for each set of 40 examples. With the higher threshold, there were on average 1.6 false negatives and 6.8 false positives. As one would expect, as the threshold increased, the number of false negatives decreased and the number of false positives increased.

For four of the participants, there was at most a small decrease in the all-or-nothing accuracy for the larger speed threshold. For the third participant, however, there was a large increase in all-or-nothing accuracy. During the debriefing after the data collection, that participant revealed that he was a calligrapher and was skilled at maintaining uniform pen speed so as to avoid ink blotches. Because the increase in all-or-nothing accuracy for the third participant offset the decreases for the other participants, the average all-or-nothing accuracy was the same for both speed thresholds ($p = 1.0$).[3] Likewise, the difference in average f-measure for the two cases was not statistically significant at $p < 0.05$ ($p = 0.6$).

5.2. Study 2

The second user study was designed to evaluate segmentation accuracy as a function of the size of the symbols and the digitizer resolution. This study employed five participants, one of whom was a participant in the first study. Each participant was asked to draw each of the 10 symbols from Fig. 8 at sizes of approximately 1, 2, and 4 cm. As in the first study, data was collected using an Intous II tablet with a paper overlay and an inking stylus, but the digitizer resolution was doubled to 2048×1536 pixels. Again, to avoid biasing the participants, they were given no feedback about how well SpeedSeg performed. To explore the impact of digitizer resolution, each participant also drew a second set of 4 cm symbols with the digitizer set to a resolution of 1024×768 pixels. Because the participants were allowed to draw naturally, some drew some of the shapes (mostly pivots) using multiple strokes. Overall, 9.5% of the shapes were multi-stroke shapes.

Table 5 summarizes the results of this study. For a digitizer resolution of 2048×1536, the average f-measure and all-or-nothing accuracy increased only slightly with symbol size. However, these differences were not significant at $p < 0.05$ (f-measure: $p = 0.2$; all-or-nothing: $p = 0.3$). Similarly, the average accuracies for the 4 cm symbols drawn with the high resolution setting were statistically the same as with the low-resolution setting (f-measure: $p = 0.9$; all-or-nothing: $p = 1.0$). Thus, there were no statistically significant variations in accuracy with variations in symbol size or digitizer resolution. However, because of the small sample size — there were only five participants in the study — additional data is needed to confirm this conclusion.

[2] While these accuracy measures have not been commonly used for pen stroke segmentation, they are widely used for other segmentation tasks, such as speech segmentation [14].

[3] All of our statistical analyses were computed with the SPSS (PASW) repeated measures analysis of variance.

Table 5
Study 2—accuracy for a speed threshold of 85%. Resolution: high = 2048 × 1536, low = 1024 × 768. False negatives = missing candidates + mistakenly merged + mistakenly dehooked.

Symbol size	1 cm	2 cm	4 cm	4 cm
Tablet resolution	High	High	High	Low
Num seg points, ave.	58	57.8	58.4	57
Missing candidates, ave.	2	0.2	0	0.4
Mistakenly merged, ave.	0.2	0.2	0.4	0
Mistakenly dehooked, ave.	1.2	0.4	0.2	1
False negatives, ave.	3.4	0.8	0.6	1.4
False positives, ave.	1	2	2	1
True positives, ave.	54.6	57.0	57.8	55.6
Precision, ave. (%)	98.2	96.6	96.7	98.2
Recall, ave. (%)	94.1	98.6	99.0	97.5
F-measure, ave.	0.961	0.976	0.978	0.979
All-or-nothing, ave. (%)	68.0	76.0	80.0	80.0

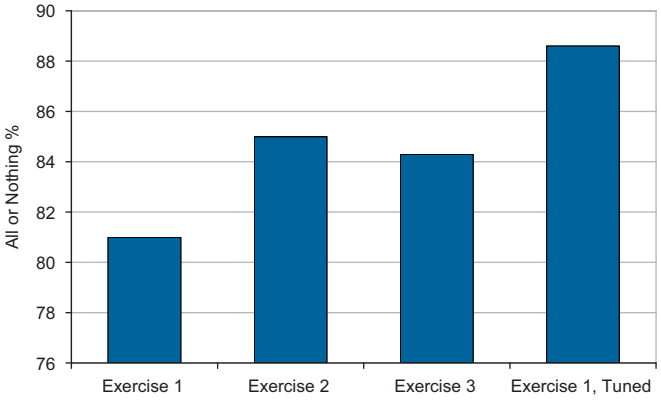

Fig. 9. All-or-nothing accuracy for Study 3. Left three columns: accuracy computed using default parameter values for Exercises 1–3. Rightmost column: accuracy for Exercise 1 using parameter values tuned for each individual user.

5.3. Study 3

The third study involved three exercises, designed to evaluate how accuracy varies as users are provided with increasing amounts of feedback about the segmentation results. In all exercises, an HP TC4400 Tablet PC with a resolution of 1024 × 768 pixels was used for data collection. In each exercise, the participants were asked to draw each of the ten symbols from Fig. 8 six times, using a single pen stroke for each shape.

In the first exercise, the participants were informed that the purpose of the study was to collect data to evaluate the performance of an algorithm. Participants were instructed to "draw naturally with ordinary accuracy," and to not attempt to "trick or break the system." The participants received no feedback about the system's performance; only the raw ink was displayed. Prior to beginning the exercise, each participant was given a few minutes to practice drawing on the Tablet PC.

In the second exercise, participants were informed that the purpose of the software is to segment pen strokes into lines and arcs. Each participant was then instructed as follows: "Guided by the program's feedback, we would like you to draw so as to achieve the intended segmentation." After each stroke was drawn, the ink was immediately replaced with the computed segmentation. Participants could choose to display both the raw ink and the segmented ink, but most chose not to. Again, prior to beginning the exercise, each participant was given a few minutes to practice drawing.

In the third exercise, participants were informed that: "One of the ways that the program finds the corners in a pen stroke is by monitoring the speed of the pen, because the hand naturally slows down at corners." The participants were then asked to repeat the second exercise.

The study involved a total of 14 participants, including nine undergraduates and five graduate students. Four participants were left-handed. The participants comprised five Computer Science majors, three Electrical Engineering majors, and six Mechanical Engineering majors. Two participants reported using pen-based devices (e.g., smart phones, gaming devices, and Tablet PCs) often, while the rest reported using such devices rarely if ever. Only one participant reported drawing frequently, while the rest reported drawing rarely if ever.

The results from the three exercises are summarized in Fig. 9. The detailed results from Exercise 1 are summarized in Table 6. The average f-measure across all 14 participants was 0.968. Likewise, the average all-or-nothing accuracy was 78.2%. Because dehooking errors are common and do not significantly affect the interpretation of a shape, Table 6 includes the symbol accuracy ignoring the first and last 20 pixels of each stroke. This accuracy is reported under the title "AoNI." On average, this accuracy was 81.0%.

Table 6
Study 3, Exercise 1—accuracy with default parameter values. Pts = number of segment points; F_n = false negatives; F_p = false positives; T_p = true positives; P = precision (%); R = recall (%); F = f-measure; AoN = all-or-nothing accuracy (%); AoNI = all-or-nothing accuracy ignoring hooks (%).

Participant	Pts	F_n	F_p	T_p	P	R	F	AoN	AoNI
1	336	19	1	317	99.7	94.3	0.969	78.3	81.7
2	347	23	3	324	99.1	93.4	0.961	78.3	80.0
3	370	3	19	367	95.1	99.2	0.971	75.0	81.7
4	350	16	11	334	96.8	95.4	0.961	70.0	70.0
5	358	3	9	355	97.5	99.2	0.983	80.0	85.0
6	359	0	4	359	98.9	100.0	0.994	93.3	95.0
7	347	9	15	338	95.8	97.4	0.966	65.0	68.3
8	341	18	1	323	99.7	94.7	0.971	83.3	86.7
9	359	1	1	358	99.7	99.7	0.997	98.3	98.3
10	327	28	7	299	97.7	91.4	0.945	68.3	75.0
11	326	43	20	283	93.4	86.8	0.900	56.7	58.3
12	373	2	18	371	95.4	99.5	0.974	80.0	80.0
13	369	3	6	366	98.4	99.2	0.988	88.3	91.7
14	349	11	8	338	97.7	96.8	0.973	80.0	81.7
Average	350.8	12.8	8.8	338.0	97.5	96.2	0.968	78.2	81.0

Table 7
Study 3, Exercise 2—accuracy with default parameter values. Pts = number of segment points; F_n = false negatives; F_p = false positives; T_p = true positives; P = precision (%); R = recall (%); F = f-measure; AoN = all-or-nothing accuracy (%); AoNI = all-or-nothing accuracy ignoring hooks (%).

Participant	Pts	F_n	F_p	T_p	P	R	F	AoN	AoNI
1	352	7	4	345	98.9	98.0	0.984	86.7	86.7
2	355	11	6	344	98.3	96.9	0.976	86.7	86.7
3	365	1	10	364	97.3	99.7	0.985	85.0	86.7
4	351	11	6	340	98.3	96.9	0.976	78.3	80.0
5	357	1	5	356	98.6	99.7	0.992	90.0	93.3
6	357	3	9	354	97.5	99.2	0.983	86.7	90.0
7	356	5	15	351	95.9	98.6	0.972	78.3	83.3
8	343	19	7	324	97.9	94.5	0.961	73.3	80.0
9	360	1	1	359	99.7	99.7	0.997	96.7	100.0
10	338	19	1	319	99.7	94.4	0.970	78.3	90.0
11	328	37	10	291	96.7	88.7	0.925	53.3	55.0
12	361	7	24	354	93.7	98.1	0.958	68.3	70.0
13	358	3	2	355	99.4	99.2	0.993	96.7	98.3
14	356	6	7	350	98.0	98.3	0.982	80.0	90.0
Average	352.6	9.4	7.6	343.3	97.8	97.3	0.975	81.3	85.0

The results from Exercise 2 are summarized in Table 7. Compared to Exercise 1, the average f-measure increased by 0.0072 to 0.975

($p=0.05$). Similarly, the average all-or-nothing accuracy increased by 3.1 percentage points to 81.3% ($p=0.2$). Likewise, the average all-or-nothing accuracy ignoring hooks increased by 4.0 percentage points to 85.0% ($p=0.07$). Enabling participants to view the segmenter's output resulted in a statistically significant increase in accuracy ($p<0.05$) only for f-measure.

Finally, the results from Exercise 3 are summarized in Table 8. Compared to Exercise 1, the average f-measure increased by 0.0094 to 0.977 ($p=0.07$). Similarly, the average all-or-nothing accuracy increased by 4.1 percentage points to 82.3% ($p=0.2$). Likewise, the average all-or-nothing accuracy ignoring hooks increased by 3.3 percentage points to 84.3% ($p=0.2$). The information provided to the participants about how the segmenter works resulted in no statistically significant increase in accuracy at $p<0.05$.

We also evaluated SpeedSeg's accuracy for Exercise 1 using parameter values tuned for each individual participant. The results are presented in Table 9. Tuning was performed using the hill climbing technique described in Section 4. Sixfold cross-validation was used to obtain an average tuned accuracy for each participant. In each fold of cross-validation, one of the participant's sets of 10 symbols was used for testing, and the other five sets were used for training. The all-or-nothing accuracy, averaged across all participants, increased from 78.2% with the default parameter values, to 88.6% with the tuned values, which is statistically significant at $p<0.001$.

The affects of tuning were most dramatic for participant 11, whose accuracy increased from 56.7% to 78.3%. This participant drew substantially slower and with less variation in speed than nearly all of the other participants. (Drawing at this participant's average speed, it would take about seven seconds to make a line across the width of the tablet display.) However, tuning enabled SpeedSeg to accommodate this particular drawing style.

Table 8
Study 3, Exercise 3—accuracy with default parameter values. Pts=number of segment points; F_n=false negatives; F_p=false positives; T_p=true positives; P=precision (%); R=recall (%); F=f-measure; AoN=all-or-nothing accuracy (%); AoNI= all-or-nothing accuracy ignoring hooks (%).

Participant	Pts	F_n	F_p	T_p	P	R	F	AoN	AoNI
1	349	9	7	340	98.0	97.4	0.977	86.7	90.0
2	355	9	4	346	98.9	97.5	0.982	85.0	85.0
3	364	0	5	364	98.6	100.0	0.993	91.7	95.0
4	350	14	19	336	94.6	96.0	0.953	63.3	63.3
5	358	0	3	358	99.2	100.0	0.996	95.0	95.0
6	353	6	2	347	99.4	98.3	0.989	90.0	90.0
7	356	4	4	352	98.9	98.9	0.989	88.3	90.0
8	342	17	8	325	97.6	95.0	0.963	70.0	71.7
9	356	2	1	354	99.7	99.4	0.996	95.0	96.7
10	344	12	4	332	98.8	96.5	0.976	76.7	83.3
11	327	32	3	295	99.0	90.2	0.944	68.3	71.7
12	375	2	21	373	94.7	99.5	0.970	70.0	76.7
13	363	7	15	356	96.0	98.1	0.970	86.7	86.7
14	356	4	6	352	98.3	98.9	0.986	85.0	85.0
Average	353.4	8.4	7.3	345.0	98.0	97.5	0.977	82.3	84.3

Table 9
All-or-nothing accuracy for Study 3, Exercise 1. "With tuning"=accuracy with parameters tuned for each participant (the per-user accuracy is averaged over sixfolds of cross-validation). "Without tuning"=accuracy with default parameter values.

Participant	With tuning (%)	Without tuning (%)
1	88.3	78.3
2	91.7	78.3
3	83.3	75.0
4	70.0	70.0
5	93.3	80.0
6	98.3	93.3
7	85.0	65.0
8	88.3	83.3
9	98.3	98.3
10	86.7	68.3
11	78.3	56.7
12	95.0	80.0
13	96.7	88.3
14	86.7	80.0
Average	88.6	78.2

6. Discussion

To compare SpeedSeg's performance to that of existing algorithms, we evaluated its accuracy on the ShortStraw data set from [40] consisting of 244 polyline shapes (there are no arcs) drawn by six different users. These shapes include 12 acute angles, 37 right angles, and 16 obtuse angles. Testing on the ShortStraw data set provides additional evidence of SpeedSeg's generality, as nine of the 11 shapes are not in our data sets. SpeedSeg's performance on this data set was evaluated using default parameter values; SpeedSeg was not tuned for the ShortStraw shapes.

SpeedSeg's accuracy on the ShortStraw data set is summarized in Table 10. The table also includes accuracy results reported in [40] for the ShortStraw algorithm [40], the algorithm of Sezgin et al. [32] which we call "SSD", and the algorithm of Kim and Kim [21] which we call "KK". SpeedSeg's f-measure is comparable to ShortStraw's and much better than that of SSD and KK.[4] More importantly, however, SpeedSeg's all-or-nothing accuracy of 86.0% is substantially better than that of all three other algorithms — ShortStraw achieved 74.1%, SSD 27.8%, and KK 29.7%. Thus, while ShortStraw and SpeedSeg achieved comparable performance measured in terms of the number of correct segment points, SpeedSeg performed substantially better in terms of the number of completely correct symbols, which is the most stringent measure of accuracy. Furthermore, SpeedSeg has an additional advantage in that it can handle strokes with arc segments, while ShortStraw cannot.

To provide another benchmark, we used the data from Exercise 1 of study 3, which contains shapes with both lines and arcs, to compare SpeedSeg's performance to that of other algorithms. Specifically, we compared SpeedSeg's accuracy to that of SSD, which is an early approach based on speed and curvature, and iStraw [41], which is a state-of-the-art algorithm. We used the SSD implementation from the authors of [40] and the iStraw implementation available from the website[5] of the authors of [41].[6] Because the algorithms vary in where they place the segment point for a given corner, we used a 20 pixel threshold in evaluating the accuracy of SSD and iStraw — if the algorithm identified a segment point within 20 pixels of the correct location, as determined by manual labeling of the data, the segment point was considered to be correct. Using a threshold was particularly important for iStraw, which segments a resampled version of the stroke, rather than the original stroke data points.

The results are shown in Table 11. SSD, which is the oldest of the three algorithms, performed the worst on all accuracy measures, achieving an f-measure of 0.882 and all-or-nothing accuracy of only 14.4%. SpeedSeg's all-or-nothing accuracy is significantly better than SSD's ($p<0.001$). Even using default parameter values, SpeedSeg outperformed iStraw on both f-measure (0.969 vs. 0.958)

[4] In [40] it is reported that the data set has 1842 segment points, while our interpretation suggests that there are 1851. We use the latter value as the basis for computing SpeedSeg's accuracy.

[5] http://www.eecs.ucf.edu/isuelab/

[6] We also implemented the algorithm of Yu and Cai [43]. However, it achieved poor performance on our data set: recall=91.9%, precision=51.2%, f-measure=0.658, and all-or-nothing=4.1%. This may be due to the omission of implementation details in the article.

Table 10
Comparison of segmentation accuracy of various algorithms on the data set from [40]. The results for ShortStraw [40], SSD [32], and KK [21] are taken from [40].

Algorithm	ShortStraw	SSD	KK	SpeedSeg
Num seg pts	1842	1842	1842	1851
False positives	38	324	387	58
False negatives	32	42	76	21
True positives	1810	1800	1766	1830
Precision (%)	97.9	84.7	82.0	96.9
Recall (%)	98.3	97.7	95.9	98.9%
F-measure	0.981	0.908	0.884	0.979
All-or-nothing (%)	74.1	27.8	29.7	86.0

Table 11
Comparison of segmentation accuracy of various algorithms on the data set from User Study 3, Exercise 1. "SpeedSeg default" = results using default parameter values. "SpeedSeg tuned" = results using parameter values tuned for each individual participant. During the tuning process, only all-or-nothing accuracy was saved, thus only this accuracy measure is reported for the tuned version of SpeedSeg.

Participant	SSD	iStraw	SpeedSeg default	SpeedSeg tuned
All-or-nothing				
1	18.90%	71.10%	78.30%	88.3%
2	20.60%	62.80%	78.30%	91.7%
3	11.70%	80.00%	75.00%	83.3%
4	15.00%	40.60%	70.00%	70.0%
5	15.00%	92.20%	80.00%	93.3%
6	15.60%	83.90%	93.30%	98.3%
7	17.80%	77.20%	65.00%	85.0%
8	7.20%	66.10%	83.30%	88.3%
9	22.20%	67.20%	98.30%	98.3%
10	19.40%	58.90%	68.30%	86.7%
11	5.60%	67.20%	56.70%	78.3%
12	7.20%	56.10%	80.00%	95.0%
13	13.30%	98.40%	88.30%	96.7%
14	11.70%	66.10%	80.00%	86.7%
Overall				
False positives	833	355	123	
False negatives	380	69	179	
True positives	4530	4841	4735	
Precision	84.5%	93.2%	97.5%	
Recall	92.3%	98.6%	96.4%	
F-measure	0.882	0.958	0.969	
All-or-nothing	14.4%	69.9%	78.2%	88.6%

and all-or-nothing accuracy (78.2% vs. 69.9%). The latter difference is significant at $p < 0.1$ but not at $p < 0.05$ ($p = 0.09$). Using tuned parameter values, SpeedSeg's all-or-nothing accuracy is significantly better than iStraw's ($p < 0.001$).[7]

SpeedSeg is intended to segment pen strokes into line and arc segments. To provide insight about its performance on these two types of segments, we tabulated the all-or-nothing accuracy ignoring hooks for each type of shape from Study 3. The results, listed in Table 12, indicate that SpeedSeg performed accurately for all 10 shapes, including those comprised solely of line segments, and those that included arcs. SpeedSeg's performance was best for the star and worst for the square root symbol.

To provide a more detailed evaluation of performance on the two types of segments, we computed the accuracy with which the program located line and arc segments in Study 3. To provide the benchmark, we manually labeled the segment points for each stroke and then labeled the resulting segments as lines or arcs based on a least squares fit as described in Section 3.3. We assigned segment type in this fashion, rather than considering what might be expected for a given kind of shape, because of the variations in how individual participants drew particular shapes. For example, while stars are often drawn solely with line segments, the star in Fig. 10 contains one segment that is best described as an arc. We measured accuracy by determining how many of the benchmark segments were actually located by the program. For Study 3, 83.3% of the segments were in fact lines, while 16.7% were arcs. SpeedSeg correctly located 93.6% of the lines and 87.6% of the arcs. Any segment that is correctly located is automatically assigned the correct segment type. Thus, the program performed well for both types of segments.

[7] As the goal of tuning is to optimize all-or-nothing accuracy, we did not save the f-measure data for the tuned algorithm. Thus, we did not perform ANOVA for f-measure.

Table 12
All-or-nothing accuracy ignoring hooks for User Study 3, in percent. Tri = triangle, Sqr = square, Om = omega, Sig = sigma, Sqrt = square root, Star = star, Wave = wave, SBlk = stepped-block, Piv = pivot.

Exercise	Tri	Sqr	Om	Sig	Sqrt	Spr	Star	Wave	SBlk	Piv
1	92.9	91.7	91.7	84.5	69.0	59.5	88.1	85.7	71.4	75.0
2	94.0	91.7	92.9	88.1	73.8	71.4	96.4	90.5	69.0	82.1
3	92.9	85.7	90.5	85.7	72.6	77.4	97.6	82.1	78.6	79.8
Ave	93.3	89.7	91.7	86.1	71.8	69.4	94.0	86.1	73.0	79.0

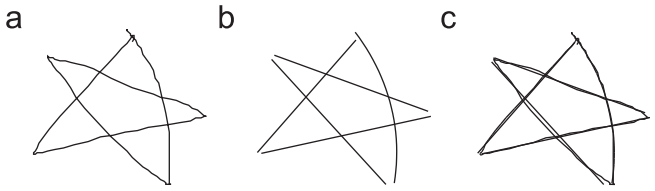

Fig. 10. Although stars are typically drawn with only line segments, this example includes an arc. (a) Raw ink. (b) Segmented ink. (c) Raw and segmented ink overlaid.

Comparison of the all-or-nothing accuracy with and without hooks in Study 3 suggests that about 3% of symbols are incorrectly segmented solely due to errors in the processing of the hooks. Because hooks are by definition small features, these errors may not significantly affect downstream applications, such as shape recognizers. Nevertheless, dehooking methods such as the one in [17] may provide a means of preventing this kind of error.

The three user studies were designed to explore different aspects of SpeedSeg's performance. However, all studies included a similar experimental protocol. Specifically, Study 1, the low-resolution portion of Study 2, and Exercise 1 of Study 3 all used 1024×768 digitizer resolution for unbiased data collection in which study participants could not view the computed segmentation. Combining this data provides a data set comprised of 1090 shapes drawn by 23 distinct participants — studies 1 and 2 had one participant in common. On this combined set, SpeedSeg achieved a precision of 97.8%, a recall of 96.3%, an f-measure of 0.970, and an all-or-nothing accuracy of 80.0%.

We also used this combined data set to investigate the influence of SpeedSeg's 12 tunable parameters (Tables 1 and 2) on accuracy. Fig. 11 shows the variation in all-or-nothing accuracy as parameters are varied over a range from 1/2 to two times their default values. Each trace on the graph represents variation of a single parameter, with the other parameters held at their default values. The abscissa of the graph represents the $\log_2$ of a parameter's value normalized by its default value, and thus ranges from -1 to 1. From inspection of Fig. 11, it is apparent that accuracy is relatively insensitive to the speed threshold (P_{ST}), the curvature threshold (P_{CT}), the curvature-sign (zero) threshold (P_{CZT}), the segment merge length trigger (P_{SMLR}), the parallelism threshold (P_{PT}), the relative end length trigger (P_{RELR}), the segment split error trigger (P_{SSER}), and

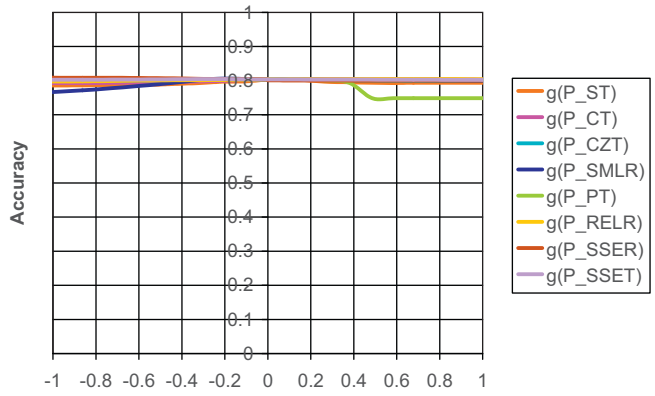

Fig. 11. All-or-nothing accuracy vs. parameter value for parameters that have relatively little influence on SpeedSeg's accuracy. $g(P_X) \equiv \log_2(P_X/P_X^0)$, where P_X is a parameter and P_X^0 is its default value. The abscissa represents a range of 1/2 to two times the default value of each parameter.

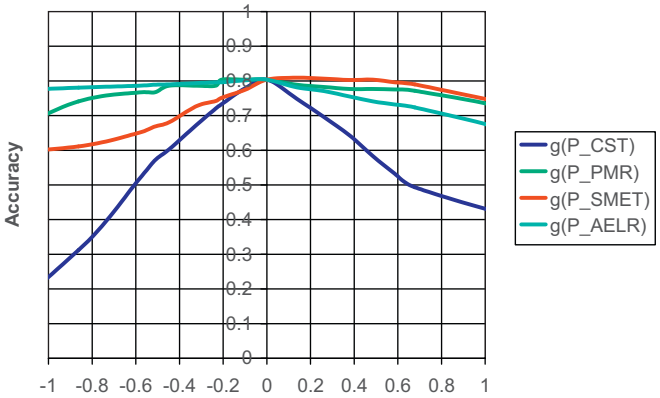

Fig. 12. All-or-nothing accuracy vs. parameter value for the parameters that have the most influence on SpeedSeg's accuracy. $g(P_X) \equiv \log_2(P_X/P_X^0)$, where P_X is a parameter and P_X^0 is its default value. The abscissa represents a range of 1/2 to two times the default value of each parameter.

the segment split error threshold (P_{SSET}). Note that the "bend" in the graph of the parallelism threshold occurs because this parameter has only a small range of meaningful values. This threshold represents the dot product of two unit vectors, which ranges from zero to one. Note that the variations in accuracy in Fig. 11 are statistically significant at $p < 0.05$ only for P_{ST} and P_{PT}.

Fig. 12 is similar to Fig. 11 but describes the remaining parameters, the ones that have the most influence on SpeedSeg's accuracy. These parameters include: the curvature-speed threshold (P_{CST}), the point merge trigger (P_{PMR}), the segment merge error threshold (P_{SMET}), and the absolute end length trigger (P_{AELR}). The variations in accuracy for these parameters are highly statistically significant ($p < 0.001$). The curvature-speed threshold is the most influential. For this data, the default value of this parameter is in fact a local optimum. The segment merge error threshold is the next most influential parameter, but even this parameter can be doubled in value with a decrease in accuracy of only 20 percentage points. The point merge trigger and the absolute end length trigger have smaller influences in that both can be halved or doubled without reducing the accuracy by more than 13 percentage points.

SpeedSeg's robustness to variations in the parameter values is due, in part, to the system's feedback loop. Errors in the initial enumeration of candidate segment points are detected and corrected by the subsequent merging and splitting processes.

The sensitivity analysis in Figs. 11 and 12 is intended to demonstrate the overall robustness of the SpeedSeg approach: the algorithm achieves good performance over a wide range of parameter values. However, within this large range, there may be particular combinations of parameter values that are optimal for a given user. This is evidenced by the results in Table 9, which demonstrate that the average all-or-nothing accuracy in Study 3, Exercise 1 could be increased by more than 10 percentage points by tuning the parameters for each user. Our tuning process performs very limited search: only 152 search states are considered. It is possible that additional search would result in even better accuracy. (This might be facilitated by using a more effective search strategy such as simulated annealing or a genetic algorithm.)

Our goal in segmenting ink is not to match the ink precisely, but rather to match the drawer's intent. Fig. 3 shows a typical example in which the intended segmentation is actually a poor fit for the ink. We believe that this necessitates the use of empirical parameter values, like those SpeedSeg uses. These parameters are, in essence, a model of what a person would perceive as important in a hand-drawn sketch. We do not believe that intrinsic properties of a curve alone are adequate to indicate the drawer's intent. Empirically determined parameter values are essential. We take an engineering perspective on this issue: if our program can provide a natural drawing environment, then we have achieved our goals.

Study 2 suggests that SpeedSeg is accurate for a range of digitizer resolutions and symbol sizes. Because this result is based on data from only five participants, it is inconclusive. In Study 3, participants were allowed to draw at whatever size they found comfortable. Thus, Study 3 confirms SpeedSeg's accuracy for conditions in which there are no constraints on symbol size. However, using SpeedSeg with drawing hardware substantially different from the types of hardware we have used may require parameter tuning, which can be done with the tuning technique described in Section 4.

All of the studies evaluated segmentation accuracy under conditions in which the participants did not observe the segmentation results. The fact that high accuracy was achieved under these conditions suggests that the SpeedSeg approach is consistent with natural drawing. Study 3 also examined accuracy when the segmentation was displayed to the participants. This did result in a small, but statistically significant increase in f-measure, suggesting that even after only a brief exposure to the system, users can intuitively control the segmentation process, if only to a small extent. When participants were provided with minimal information about how SpeedSeg computes segmentation, there was no statistically significant gain in accuracy. This may suggest that being informed of how the system works does not matter because the system naturally matches the way most users draw. It is also possible that our explanation to the participants did not provide enough information. At the completion of the study, one participant did realize that he had misunderstood the explanation and applied the reverse strategy of speeding up at the intended corners. As our goal is to create a system that is natural to use, we crafted our explanation so as to provide guidance without explicitly telling participants how to use the system. It may be useful to conduct a future study evaluating performance under conditions in which participants are explicitly told how to use the system most effectively.

Because SpeedSeg is intended to be used in interactive sketching applications, runtime performance is critical. SpeedSeg has proven to be quite fast. Segmentation of a typical pen stroke from the user study takes only about 10 ms on an HP TC4400 Tablet PC with a 2.0 GHz Intel T7200 Core 2 Duo Processor and 2 GB of memory. (Pen strokes were recorded at the maximum sampling rate of the hardware by directly addressing the Wacom tablet driver.) Likewise, tuning the system on a training set of 50 shapes takes only about 75 s of processor time, excluding file I/O to load and save the training data.

The purpose of a segmenter is to support shape recognizers. Thus one measure of the performance of a segmenter is the accuracy of a recognizer built upon it. We have used SpeedSeg as part of a graph-based shape recognizer [24]. In user studies reported in [24],

this recognizer achieved a top-one accuracy (correct class is the top choice) of 93.6% and a top-three accuracy (correct class is in the top-three choices) of 99.0%. This suggests that a shape recognizer can overcome the occasional false negative and false positive segment points. This, in fact, provides an opportunity for error correction. Once a recognizer has identified a shape, knowledge of the shape can then be used to correct segmentation errors, should that be desirable.

We have also used SpeedSeg to build AC-SPARC [9], a sketch-based interface for the SPICE electric circuit simulator. In user studies with this system, SpeedSeg correctly segmented 91% of the symbols. The particular version of SpeedSeg used in AC-SPARC had some small differences from the version described above. Thus, the 91% accuracy cannot be directly compared to the results of the three user studies reported in Section 5.

The two speed thresholds (P_{ST} and P_{CST}) are computed from the average pen speed along the entire pen stroke. While this works well, we expect it may be possible to obtain even better results by considering only a portion of the pen stroke. For example, the speed thresholds could be based on the average speed within a window of points that slides along the stroke. In this way, the thresholds at any given point would be more strongly influenced by the local properties of the pen stroke. This may be particularly useful for long strokes.

7. Conclusion

The challenge in segmenting a pen stroke is to identify the geometric primitives intended by the drawer. Frequently, the intent is not a literal interpretation of the stroke. Thus, segmentation techniques that aim for a precise match to the ink are likely to produce poor results. As an alternative, our approach uses pen speed to help infer intent. The approach is based on the observation that it is common for the drawer to slow the pen tip at points of intended discontinuities.

Based on this observation, we have developed a technique for segmenting hand-drawn pen strokes into lines and arcs. To begin, an initial set of candidate segment points is identified. This set includes speed minima below a threshold, where the threshold is computed from the average pen speed along the stroke. The set also includes curvature maxima at which the pen speed is again below a threshold. Once the initial set of candidates has been generated, the ink between each pair of consecutive segment points is classified as either a line or an arc, depending on which fits best. A feedback process is then employed, and segments are judiciously merged and split as necessary to improve the quality of the segmentation.

We conducted three formal user studies to evaluate SpeedSeg's performance. These studies considered the unbiased case in which the participants could not view the computed segmentation. We found the system to be accurate even for new users: on the combined data from the three studies, SpeedSeg achieved an f-measure of 0.970 and an all-or-nothing accuracy of 80.0% using default parameter values. Allowing participants to view the computed segmentation resulted in a small but statistically significant increase in f-measure, suggesting that even after only a brief exposure to the system, users can begin to intuitively control the segmentation process.

Our studies have also shown that SpeedSeg performs well for both small (1 cm) and large (4 cm) shapes, and that the system can be used with a range of digitizer resolutions. The system has proven to be fast, requiring only about 10 ms to segment a typical pen stroke. SpeedSeg uses several empirical parameters, but accuracy is strongly sensitive to only one of them. Nevertheless, it is still possible to quickly tune the system to optimize performance for a given user. In our experiments, 75 s of optimization on a set of 50 shapes resulted in an average increase in all-or-nothing accuracy of 10 percentage points.

In summary, this work has demonstrated the utility of pen speed for inferring the intended segmentation of pen strokes. While there are still opportunities to improve SpeedSeg's performance, our studies have demonstrated that the technique is sufficiently reliable for use in practical systems.

Acknowledgments

The authors gratefully acknowledge support for this work provided by the National Science Foundation via award no. 0729422. They also thank Tracy Hammond and her Sketch Recognition Lab for providing their implementation of the SSD algorithm, and Robert Calfee for providing assistance with the statistical analyses.

References

[1] Agar P, Novins K. Polygon recognition in sketch-based interfaces with immediate and continuous feedback. In: Proceedings of the first international conference on computer graphics and interactive techniques in Australasia and South East Asia. ACM Press; 2003. p. 147–50.
[2] Alvarado C, Davis R. Sketchread: a multi-domain sketch recognition engine. In: Proceedings of UIST, 2004.
[3] Burt PJ. Fast filter transforms for image processing. Computer Graphics and Image Processing 1981;16:20–51.
[4] Deufemia V, Risi M. A dynamic stroke segmentation technique for sketched symbol recognition. In: Pattern recognition and image analysis. Berlin/Heidelberg: Springer; 2005. p. 328–35.
[5] Dudek G, Tsotsos J. Shape representation and recognition from multiscale curvature. Computer Vision and Image Understanding 1997;68(2):170–89.
[6] Eggli L. Sketching with constraints. Master's thesis, University of Utah; 1994.
[7] Zeleznik R, et al. SKETCH: an interface for sketching 3D scenes. In: Proceedings of SIGGRAPH'96, 1996. p. 163–70.
[8] Fonseca MJ, Jorge JA. Using fuzzy logic to recognize geometric shapes interactively. In: Proceedings of the ninth international conference on fuzzy systems (FUZZ-IEEE 2000), San Antonio, USA, May 2000.
[9] Gennari L, Kara LB, Stahovich TF, Shimada K. Combining geometry and domain knowledge to interpret hand-drawn diagrams. Computers & Graphics 2005;29(4):547–62.
[10] Hammond T, Davis R. Ladder, a sketching language for user interface developers. Computers & Graphics 2005;28:518–32.
[11] Herot CF. Graphical input through machine recognition of sketches. In: Proceedings of the third annual conference on computer graphics and interactive techniques. ACM Press; 1976. p. 97–102.
[12] Hse H, Shilman M, Richard Newton A. Robust sketched symbol fragmentation using templates. In: IUI '04: proceedings of the ninth international conference on intelligent user interfaces, 2004. p. 156–60.
[13] Hse HH, Richard Newton A. Recognition and beautification of multi-stroke symbols in digital ink. Computers & Graphics 2005;29(4):533–46.
[14] Kim JH, Woodland PC. The use of prosody in a combined system for punctuation generation and speech recognition. In: Proceedings of EUROSPEECH, 2001. p. 2757–60.
[15] Igarashi T, Matsuoka S, Kawachiya S, Tanaka H. Interactive beautification: a technique for rapid geometric design. In: UIST '97, 1997. p. 105–14.
[16] Jenkins DL, Martin RR. Applying constraints to enforce users' intentions in free-hand 2-D sketches. Intelligent Systems Engineering 1992;1(1).
[17] LaViola Jr JJ. Mathematical sketching: a new approach to creating and exploring dynamic illustrations. PhD thesis, Brown University; 2005.
[18] Kara LB, Gennari L, Stahovich TF. A sketch-based tool for analyzing vibratory mechanical systems. Journal of Mechanical Design 2008;130(10).
[19] Kara LB, Stahovich TF. Hierarchical parsing and recognition of hand-sketched diagrams. In: UIST, 2004. p. 13–22.
[20] Kara LB, Stahovich TF. An image-based, trainable symbol recognizer for hand-drawn sketches. Computers & Graphics 2005;29(4):501–17.
[21] Kim DH, Kim M-J. A curvature estimation for pen input segmentation in sketch-based modeling. Computer-Aided Design 2006;38(3):238–48.
[22] Landay JA, Myers BA. Sketching interfaces: toward more human interface design. IEEE Computer 2001;34(3):56–64.
[23] Lee J-S, Sun Y-N, Chen C-H. Multiscale corner detection by using wavelet transform. IEEE Transactions on Image Processing 1995;4(1):100–4.
[24] Lee W, Kara LB, Stahovich TF. An efficient graph-based recognizer for hand-drawn symbols. Computers & Graphics 2007;31(August):554–67.
[25] Lin VC, Gossani DC, Light RA. Variational geometry in computer-aided design. Computer & Graphics 1981;15(3).
[26] Mortenson ME. Geometric modeling. John Wiley & Sons, Inc; 1985.
[27] Paulson B, Hammond T. Paleosketch: accurate primitive sketch recognition and beautification. In: IUI '08: proceedings of the 13th international conference on intelligent user interfaces. New York, NY, USA: ACM; 2008. p. 1–10.
[28] Press WH, Vetterling WT, Teukolsky SA, Flannery BP. In: Numerical recipes in C++: the art of scientific computing. 2nd ed.. New York, NY, USA: Cambridge University Press; 2002.
[29] Qin SF, Wright DK, Jordanov IN. On-line segmentation of freehand sketches by knowledge-based nonlinear thresholding operations. Pattern Recognition 2001;34(10):1885–93.

[30] Rattarangsi A, Chin RT. Scale-based detection of corners of planar curves. IEEE Transactions on Pattern Analysis and Machine Intelligence 1992;14(4):339–430.
[31] Rubine D. Specifying gestures by example. Computer & Graphics 1991;25(July): 329–37.
[32] Sezgin T, Stahovich T, Davis R. Sketch based interfaces: early processing for sketch understanding. In: Proceedings of the 2001 perceptive user interfaces workshop (PUI'01), 2001.
[33] Sezgin TM. Feature point detection and curve approximation for early processing of free-hand sketches. Master's thesis, Massachusetts Institute of Technology; 2001.
[34] Shakarji CM. Least-squares fitting algorithms of the NIST algorithm testing system. Journal of Research of the National Institute of Standards and Technology 1998;103(6).
[35] Shpitalni M, Lipson H. Classification of sketch strokes and corner detection using conic sections and adaptive clustering. ASME Journal of Mechanical Design 1996;119(2):131–5.
[36] Simhon S, Dudek G. Pen stroke extraction and refinement using learned models. In: Eurographics workshop on sketch-based interfaces and modeling (SBIM'04), 2004.
[37] Steidel RF, Henderson JM. The graphic language of engineering. John Wiley & Sons; 1983.
[38] Teh CH, Chin RT. On the detection of dominant points on digital curves. IEEE Transactions on Pattern Analysis and Machine Intelligence 1989;11(8): 859–72.
[39] Wobbrock JO, Wilson AD, Li Y. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In: UIST '07: proceedings of the 20th annual ACM symposium on user interface software and technology, 2007. p. 159–68.
[40] Wolin A, Eoff B, Hammond T. Shortstraw: a simple and effective corner finder for polylines. In: Eurographics workshop on sketch-based interfaces and modeling (SBIM'08), 2008.
[41] Xiong Y, LaViola Jr JJ. A shortstraw-based algorithm for corner finding in sketch-based interfaces. Computers & Graphics 2010;34(5):513–27.
[42] Yu B. Recognition of freehand sketches using mean shift. In: International conference on intelligent user interfaces, IUI03, Miami, Florida, January 2003.
[43] Yu B, Cai S. A domain-independent system for sketch recognition. In: GRAPHITE '03: proceedings of the first international conference on computer graphics and interactive techniques in Australasia and South East Asia. ACM; 2003. p. 141–6.
[44] Zhao R. Incremental recognition in gesture-based and syntax directed diagram editor. In: Proceedings of interCHI'93, 1993. p. 95–100.