EUROGRAPHICS Symposium on Sketch-Based Interfaces and Modeling (2011)
T. Hammond and A. Nealen (Editors)

ClassySeg: A Machine Learning Approach to Automatic Stroke Segmentation

J. Herold[1] and T. F. Stahovich[2]

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

Abstract

We present ClassySeg, a technique for segmenting hand-drawn pen strokes into lines and arcs. ClassySeg employs machine learning techniques to infer the segmentation intended by the drawer. The technique begins by identifying a set of candidate segment points, consisting of all curvature maxima. Features are computed for each candidate point based on speed, curvature, and other geometric properties. These features are adapted from numerous prior segmentation approaches, effectively combining their strengths. These features are used to train a statistical classifier to identify which candidate points are true segment points. A beam search is used to approximate the optimal subset of features to use as input to the classifier. ClassySeg is more accurate than previous techniques for user-independent training conditions, and is as good as the current state-of-the-art algorithm for user-optimized conditions. More importantly, ClassySeg represents a movement away from prior heuristic-based approaches towards a more general and extensible approach.

Categories and Subject Descriptors (according to ACM CCS): I.4.6 [Computer Graphics]: Image Processing and Computer Vision—Segmentation - Edge and feature detection

1. Introduction

Automatic pen stroke segmentation is the process by which a digital pen stroke is segmented into its constituent lines and arcs. For example, stroke segmentation would decompose a hand-drawn triangle into the three straight lines that comprise it. The challenge in this process is determining which bumps and bends in the stroke are intended corners and which are not. It has been shown that curvature information alone is an unreliable indicator of segmentation [SSD06], and thus a more sophisticated approach is required.

Stroke segmentation is an essential first step in shape recognition [LBKS07, PH08] and thus is a crucial part of sketch-based interfaces. Decomposing a stroke into its constituent geometric primitives also facilitates beautification, in which the hand-drawn primitives are replaced by mathematically precise shapes to produce a neater final result [JM92, IMKT97].

Existing segmentation techniques typically rely on heuristic algorithms and empirically determined parameters. ClassySeg provides greater extensibility and generality than these previous methods by employing general machine learning techniques to identify the segment points in a stroke. ClassySeg begins by identifying a set of candidate points consisting of all curvature maxima. Next, speed, curvature, and other geometric features are computed for each candidate point. These features are taken from previous segmentation approaches, effectively combining their strengths. The features are used to train a statistical classifier which determines which candidate points are true segment points and which are not. To optimize performance, a beam search was used to identify the subset of features that produces the most accurate classifier. ClassySeg was evaluated on a large data set of pen strokes from [HS11] and is more accurate than previous techniques for user-independent training conditions. Just as important, ClassySeg can be easily extended to include other features and is highly tunable. For example, it can be optimized for different kinds of shapes and can be tuned for individual users and various drawing hardware.

The next section places our approach in the context of

SBIM 2011, Vancouver, British Columbia, Canada, August 5 – 7, 2011.
© 2011 ACM 978-1-4503-0906-6/11/0008 $10.00

previous work. Next, the dataset used to evaluate ClassySeg and benchmark it against prior techniques is described. This is followed by a discussion of the main components of the ClassySeg approach, including candidate point selection, feature computation, classifier training, and feature subset selection. Finally, ClassySeg's accuracy is compared to that of three previous segmentation approaches, and to a baseline, naıve segmentation approach.

2. Related Work

Pen stroke segmentation is a well researched topic, and numerous methods have been developed. Yu and Cai's [YC03] technique 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 then recursively processed. Segments are merged in a post-processing phase, but the criteria for doing this are not specified.

The technique of Sezgin et al. [SSD06], which we call SSD, uses speed and curvature to segment pen strokes. Segment points are located at points of minimum speed and maximum curvature. This work demonstrated the usefulness of pen speed data for segmentation, and showed that curvature data alone is inadequate. SSD is suitable for segmenting pen strokes into sequences of line segments, but cannot handle arcs.

Wolin et al. [WEH08] developed ShortStraw, which begins by resampling the pen stroke, and then computes the "straw value" for each point, which gives an indication of the local curvature. All points with a straw value below an empirically determined threshold are considered candidate segment points. A top-down phase then examines the segments between each pair of consecutive candidate points to evaluate the quality of the line fit. If the fit is poor, segment points are added.

Xiong and La Viola [XJ10] developed iStraw, which improves upon the ShortStraw approach by including timing information and curvature detection. iStraw achieves better accuracy than ShortStraw and is able to handle curve and arc segments, which ShortStraw cannot.

Wolin et. al's [WPH09] Sort, Merge, Repeat (SMR) technique begins just as SSD does, by locating candidate segment points at speed minima and curvature maxima. The algorithm then finds the shortest segment and merges it with one of its neighboring segments in an attempt to remove false positives. This process is repeated until the line and arc fit errors of the segments are below an empirically determined threshold.

Recently, Herold and Stahovich [HS11] presented SpeedSeg. This approach also identifies the initial candidate segment points at speed minima and curvature maxima. A set of heuristics are then used to both merge and split the initial segmentation to produce a more accurate final result. The heuristics employ several geometric and speed-based features with empirical thresholds. These threshold can be optimized to improve performance.

Nearly all of these approaches rely on heuristics and empirical parameters, which limit their extensibility. In many cases, there is no automated procedure for selecting optimal parameter values. By contrast, ClassySeg uses a general purpose machine learning approach that naturally extends to incorporate any number of features. Here, we use the approach with a collection of features derived from multiple existing segmentation techniques, but other features can be directly added. Furthermore, ClassySeg is highly optimizable. It can both determine the optimal subset of features to use and identify optimal parameter values (via a trained classifier). As a result, ClassySeg can be easily tuned for specific users, specific classes of shapes, and specific drawing hardware.

3. Data Set

We used the pen stroke data from the study described in [HS11] to evaluate ClassySeg's performance and benchmark it against prior segmentation methods. In that study, an HP TC4400 Tablet PC with a digitizer resolution of 1024x768 pixels was used for data collection. The participants were asked to draw each of the ten symbols from Figure 1 six times.

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." Before beginning the exercise, each participant was given a few minutes to practice drawing on the Tablet PC.

Each point from the data set is a triple containing an x-coordinate, y-coordinate, and time value. To facilitate training and testing of algorithms, the true segment points on each pen stroke were manually labeled using an approach analogous to that in [WSA07]. Specifically, the pen stroke data was initially segmented using the segmentation technique described in [Sta04]. Then segment points were manually added, deleted, and moved as necessary to achieve the correct segmentation.

4. Approach

ClassySeg uses a C4.5 decision tree to determine which of the initial candidate segment points are true segment points. The candidates consist of the curvature maxima. The decision tree is trained using 48 features, many of which are taken from existing segmentation techniques. To improve performance, a beam search is used to identify the subset of features that results in the best-performing decision tree.

The sections that follow describe the approach in more detail, including the identification of candidate segment points,

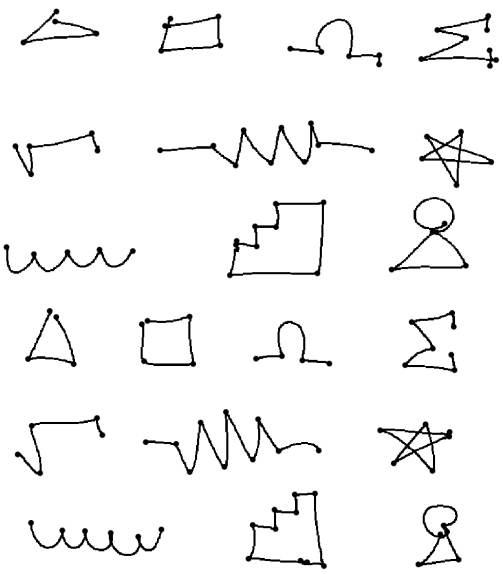

Figure 1: The ten shapes used in the user study: triangle, square, omega, sigma, square root, spring, star, wave, stepped-block, pivot, drawn by two separate users. Large data points denote manually labeled segment points. Note that the number of segments comprising a symbol may differ from one user to the next.

the features used for classification, and the approach used for training.

4.1. Candidate Point Selection

In the dataset described in Section 3, only 2.61% of the data points are segment points. Training a classifier to detect such rare cases can be difficult [Wei04]. There are a number of common ways to address this kind of problem, such as resampling schemes [Jap01] and the use of sophisticated objective functions for training the classifier [KHM98].

However, for the segmentation task, there is a more direct approach to overcome this problem: most segment points are maxima of curvature. In the dataset we consider, only 0.20% of actual segment points are not maxima. Furthermore, 19.91% of the maxima are true segment points, which is a sufficiently high frequency to enable accurate classification.

ClassySeg identifies candidate segment points using the curvature computation in Section 4.2.2. As described there, this computation actually computes the absolute value of curvature. Thus, all curvature extrema, regardless of the direction of concavity are selected as initial candidate segment points.

4.2. Feature Computation

ClassySeg currently uses 48 features. Some describe basic geometric properties of the pen stroke such as arc length and the quality of fit of the line and arc segments. Others are adapted from SSD [SSD06], Kim and Kim's method [KK06], ShortStraw [WEH08], iStraw [XJ10], and SpeedSeg [HS11]. The details of these features are described below.

4.2.1. Arc Length: $f_{aln}, f_{alp}, f_{alnpr}, f_{als}, f_{ale}, f_{alnn}, f_{alpn}, f_{alsn}, f_{alen}$

The ends of a pen stroke often require special handling because of the presence of hooks [HS11]. Thus, we characterize each candidate segment point by a number of arc length features. The arc length, d_i, of a point, p_i, is defined as:

$$d_i = \sum_{j=1}^{i} \left|\left| \vec{P}_j - \vec{P}_{j-1} \right|\right| \quad (1)$$

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

The arc length is computed between each candidate point and both the next candidate point (f_{aln}) and the previous candidate point (f_{alp}). The ratio of f_{aln} to f_{alp} is included as feature f_{alnpr}. A short arc length between consecutive candidate segment points may indicate that one, or both, is not a true segment point. To enable the classifier to properly handle the ends of a stroke, the arc length from the candidate point to both the the start point (f_{als}) and end point of the stroke (f_{ale}) are also computed.

To obtain arc length features that are independent of the size of the pen stroke, a second set of features is obtained by normalizing the above features by the arc length of the stroke. The resulting four features are denoted by the addition of an "n" to the subscript: $f_{alnn}, f_{alpn}, f_{alsn}, f_{alen}$. ($f_{alnpr}$ is already normalized.)

4.2.2. Curvature: $f_c, f_{cp}, f_{cn}, f_{cmm}, f_{cmmp}, f_{cmmn}, f_{ca}, f_{cap}, f_{can}$

We compute curvature using the approach from [HS11]. As all extrema of curvature are potential corners, ClassySeg works with the absolute value of curvature. In this way, all extrema are detected simply as maxima. For convenience, we use term "curvature" to mean the "absolute value of the curvature."

The curvature, c, is computed as the derivative of the tangent angle, θ, with respect to arc length:

$$c = \left| \frac{d\theta}{ds} \right| \quad (2)$$

The feature f_c is defined to be equal to c

To construct the tangent angle at a point, a least squares line is constructed using the point and the five points on each side. This method naturally smooths the often noisy curvature data. If the line fit is inaccurate, i.e., if the average distance from the 11 points to the least squares line is greater than 10% of the arc length of the window of 11 points, a least squares circle is instead used to establish the tangent. The derivative of the tangent angle is also computed using a least squares line fit to the graph of the tangent angle vs. the arc length. The slope of this line gives the curvature in units of radians per pixel.

To provide the classifier with additional context about the curvature near a candidate segment point, the curvature of the points immediately preceding and proceeding the candidate point are used as features f_{cp} and f_{cn}.

The curvature at a corner can be arbitrarily large. Thus, to better characterize the shape, we compute two normalized curvature features. For the first (f_{cmm}), the curvature is normalized by the minimum and maximum curvature along the stroke:

$$f_{cmm} = \frac{(f_c - c_{min})}{c_{max}} \tag{3}$$

where c_{min} is the minimum curvature value over all points in the stroke, and c_{max} is the maximum. This normalized curvature value for the points immediately preceding and proceeding each candidate point are included as features: f_{cmmp} and f_{cmmn}. Likewise, the feature f_{ca} is obtained by normalizing the curvature by the average curvature of the stroke, c_{ave}:

$$f_{ca} = \frac{f_c}{c_{ave}} \tag{4}$$

Again, the average-normalized curvature of the points immediately preceding and proceeding each candidate point are also included as features: f_{cap} and f_{can}.

4.2.3. Straw: f_{sti}, f_{stri}

The ShortStraw algorithm of Wolin et al. [WEH08] uses the "straw value", which is analogous to curvature, to identify corners. The straw value at a point, p_i, is equal to the Euclidean distance between the points p_{i-w} and p_{i+w}:

$$straw_i = |p_{i-w}, p_{i+w}| \tag{5}$$

where w is a constant. Figure 2 shows an example of the straw value of both a segment point and a non-segment point. A small straw value is indicative of a true segment point.

With ShortStraw, the straw value is not well-behaved at the ends of the stroke. The iStraw algorithm [XJ10] uses a more general definition of the straw to remedy this:

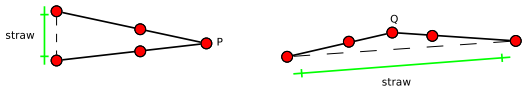

Figure 2: Straw values at a point P which is a true segment point (left) and a point Q which is not (right). (w = 2.) The straw value for a true segment point is much smaller than for a non-segment point.

$$\begin{aligned}
straw_0 &= 0 \\
straw_{N-1} &= 0 \\
straw_1 &= |p_0, p_{1+w}| \times \frac{2w}{(w+1)} \\
straw_2 &= |p_0, p_{2+w}| \times \frac{2w}{(w+2)} \\
straw_{N-2} &= |p_{N-1}, p_{N-2-w}| \times \frac{2w}{(w+1)} \\
straw_{N-3} &= |p_{N-1}, p_{N-3-w}| \times \frac{2w}{(w+2)}
\end{aligned} \tag{6}$$

We compute the straw value using the iStraw approach. Before computing the straw value, both ShortStraw and iStraw resample the ink to produce equally spaced data points. We compute straw values two ways, first using the original data points, then using resampled data points. Additionally, we compute straw values using four different values of w ($w = 1, 2, 3,$ and 4), resulting in four straw features based on the original data points ($f_{st1}, f_{st2}, f_{st3},$ and f_{st4}) and four based on the resampled data points ($f_{str1}, f_{str2}, f_{str3},$ and f_{str4}).

Both ShortStraw and iStraw work with resampled data and do not classify the original data points. By contrast, our goal is to identify which of the original data points are true segment points. Thus, when resampling the ink to compute the straw value, we do a local computation that preserves the data point in question, and computes equally-spaced sample points on either side of it. We use the resampling technique used in [WWL07].

4.2.4. Alpha and Beta: $f_{\square}$, $f_{\square}$, $f_{\square\square}$, $f_{\square r}$, $f_{\square r}$, $f_{\square\square r}$

iStraw [XJ10] uses two angles, $\square$ and $\square$, to help distinguish true corners from smooth curves. Like the straw value, these angles are computed using a pair of resampled points, one on each side of the point in question. Angle $\square$ is computed using data points near the point in question, while angle $\square$ is computed using points that are farther.

Our features $f_{\square}$ and $f_{\square}$ are similar to angles $\square$ and $\square$, except that our features are computed from the actual data

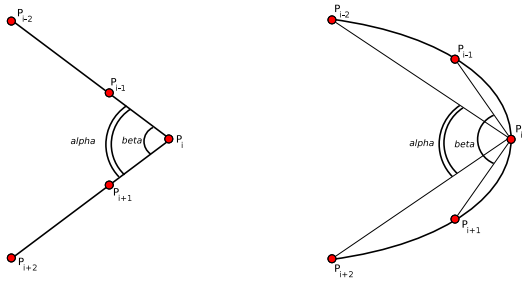

Figure 3: □ and □ values of a point, P_i, in the event of a corner (left) and curve (right). The difference between □ and □ is near zero in the case of a corner and relatively large in the case of a curve.

points, rather than resampled ones. Also, we use a different offset for selecting the points that define the angles. As shown in Figure 3, we compute $f_{\square}$ as the angle between the line segment joining p_i and p_{i-1} and the line segment joining p_i and p_{i+1}. Similarly, we compute $f_{\square}$ as the angle between the line segment joining p_i and p_{i-2} and the line segment joining p_i and p_{i+2}.

In iStraw, the difference between □ and □ is used to distinguish corners. The difference between □ and □ tends to be small in the case of true corners, and large in the case of curves. For this purpose, we use the feature $f_{\square\square}$, which is defined as the difference between $f_{\square}$ and $f_{\square}$.

We compute a second set of angle features ($f_{\square r}$, $f_{\square r}$, and $f_{\square\square r}$) using resampled ink. We resample the ink just as we do to compute the straw features in Section 4.2.3.

4.2.5. Region of Support: f_{asp}, f_{asn}

In their stroke segmentation approach, Kim and Kim [KK06] use a curvature estimation technique which combines the signed curvature values of the k-nearest neighbors of a point — called region of support — to determine the curvature at that point.

We define the region of support for a candidate point as the set of neighboring points that are locally convex to it. More precisely, we compute two region of support features as illustrated in Figure 4: f_{asp} is the arc length from the candidate point to the previous curvature minima, while f_{asn} is the arc length from the candidate point to the next curvature minima. The larger these features, the more likely it is that the candidate point is a segment point.

4.2.6. Speed: f_s, f_{sn}, f_{sp}, f_{smm}, f_{smmn}, f_{smmp}, f_{sa}, f_{san}, f_{sap}

People typically decrease pen speed when making deliberate corners [SSD06]. Thus, low speed at a candidate point is often good evidence that the point is a true segment point.

We compute pen speed using a centered, finite difference

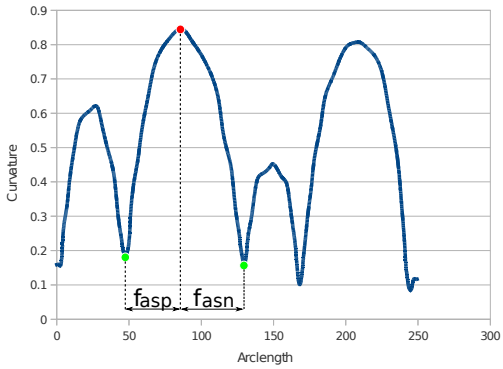

Figure 4: Region of support features for the red candidate point: f_{asp} and f_{asn} are the arc length to the next and previous curvature minima (green points).

approach from [HS11]. The speed at point p_i is initially computed as:

$$s_i = \frac{d_{i+1} - d_{i-1}}{t_{i+1} - t_{i-1}} \tag{7}$$

where, t_i and d_i are the time stamp and arc length, respectively, of point p_i. Because the speed data is noisy, we apply a simple filter: the speed at each point is averaged with that of the two points on either side. The feature f_s is defined as the value of the smoothed pen speed.

Just as with the curvature features, we also compute several normalized speed features. First, speed is normalized by the minimum and maximum pen speed of the stroke: f_{smm}. The minimum/maximum-normalized speed of the points immediately preceding and proceeding each candidate point are also included as features: f_{smmp} and f_{smmn}. Likewise, the feature f_{sa} is obtained by normalizing the speed by the average speed of the stroke. Finally, the average-normalized speed of the points immediately preceding and proceeding each candidate point are included as features: f_{sap} and f_{san}.

4.2.7. Arc and Line Fit: f_{afp}, f_{afn}, f_{lfp}, f_{lfn}, f_{lfw}

One measure of the quality of the segmentation is the degree to which the segments can be fit by geometric primitives. The approach in [YC03], for instance, recursively splits the segmentation until a good a good fit is achieved. Thus, we compute least squares arcs for the segments on either side of a candidate segment point. The features f_{afp} and f_{afn} are defined as the error of fit between these arcs and the actual data points. The features f_{lfp} and f_{lfn} are defined analogously, but represent the error of fit to least squares lines. If the next and previous segments form a single contiguous line, the candidate segment point may be a false segment point. Thus, we compute a single least squares line fit to the ink

spanning from the previous to the next candidate segment points. If the error of fit of this combined segment (f_{lfw}) is small, the candidate is likely a false segment point.

4.3. Classifier Training

The features and manually assigned labels of each candidate point were used to train a C4.5 decision tree classifier [Qui93] implemented in WEKA [HFH*09]. The trained classifier is then used to predict if a given candidate point is a true segment point.

Using the full set of features may not yield the best accuracy. For example, if two features contain redundant information, this can adversely affect the classifier's accuracy. While a decision tree automatically determines which feature values best separate true positives from false positives, a separate process is required to determine the subset of features most useful in training the classifier. To approximate the optimal subset of features, we used a traditional beam search approach [BSP*09, DL97] with a beam width of 10.

Beam search [Zha99] is a modified best-first search used here to explore the large space of possible feature subsets to determine the one that produces the best classification accuracy. The search begins by training and evaluating all possible single-feature classifiers. The 10 single-feature subsets producing the highest accuracy are then expanded to create a set of two-feature classifiers. The 10 best of these are then expanded to create a set of three-feature classifiers, and so on, until the set of all features is reached. The best-performing subset is used as the optimal feature subset.

Two cross-validation schemes were used to evaluate ClassySeg. The first is a *user agnostic* scheme which evaluates ClassySeg's performance under conditions in which the training set excludes data from the subject used for the testing set. The second is a *user optimized* scheme in which the training and testing data are from the same subject, but are still distinct. In both schemes, a beam search is performed in each testing-training fold. The classifiers created in the beam search are both trained and evaluated using the full training set of that fold.

In the user agnostic scheme, a 14-fold, "user-holdout" cross-validation was performed. In each fold, the data from one subject was selected for testing, and the data from the 13 other subjects was used for training. To begin each fold, a beam search was performed to determine the optimal feature set. The optimal features were then used to train the final C4.5 decision tree on the complete training set.

In the user optimized scheme, a 60-fold, "stroke-holdout" cross-validation was performed for each of the 14 subjects. In each fold of "stroke-holdout," one of the subject's strokes was used for testing, and the other 59 were used for training. Here again, a beam search was performed using the training set of each fold to determine the optimal feature set prior to training of the final C4.5 decision tree.

In both schemes, beam search found subsets containing eight to sixteen features. The following features occur in a majority of the feature subsets: f_{lfw}, f_{smmp}, f_{cn}, f_{str1}. The following features appear in none of the feature subsets: f_{lfp}, f_{alnpr}, $f_{\square}$, f_{alsn}, f_{st1}, f_{st2}, f_{st3}, f_{str3}, f_{st4}, f_{str4}, $f_{\square}$. These latter features may be redundant with other features, or may not be useful.

5. Results

We report accuracy using a number of measures common for other segmentation tasks (e.g., speech segmentation [GR00]): precision, recall, f-measure, and all-or-nothing accuracy. Precision, P, is the fraction of the predicted segment points that are true segment points:

$$P = \frac{\text{true positives}}{(\text{true positives} + \text{false positives})} \tag{8}$$

Here, true positives are true segment points that were classified as such by ClassySeg, while false positives are points that were erroneously classified as segment points.

Recall, R, is the fraction of the true segment points that were correctly identified:

$$R = \frac{\text{true positives}}{(\text{true positives} + \text{false negatives})} \tag{9}$$

Here, false negatives are true segment points that were incorrectly classified as not being segment points.

F-Measure, F, combines the precision and recall values; it is the harmonic mean of the two:

$$F = \frac{2PR}{(P+R)} \tag{10}$$

Finally, all-or-nothing accuracy, AoN, is the fraction of the pen strokes the have been perfectly segmented:

$$AoN = \frac{\text{perfect strokes}}{\text{total strokes}} \tag{11}$$

To benchmark ClassySeg's performance, we compared it to five other techniques. Because these techniques may vary in where they place the segment point for a given corner, we used a 20 pixel threshold in evaluating the accuracy of each method – if a technique 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 summarized in Table 1. The first technique is a naıve one in which every candidate segment point is considered to be a true segment point. The other four methods are more sophisticated and include: the SSD implementation from the authors of [WEH08], the iStraw implementation from the authors of [XJ10] (http://www.eecs.ucf.edu/isuelab/), and the SpeedSeg implementation from the authors of [HS11]. (We also implemented the algorithm of Yu and Cai [YC03]. However, it achieved poor performance on our data set: AoN = 4.13%. This may be due to the omission of implementation details in the article.) All techniques were evaluated using the data described in Section 3. SpeedSeg's performance is reported for both default parameter values and parameter values optimized for each individual subject.

Method	P	R	F	AoN
Naıve	19.9%	99.8%	0.33	0%
SSD	84.7%	97.7%	0.91	27.8%
iStraw	93.2%	98.6%	0.96	69.9%
SpeedSeg (Default)	97.5%	96.4%	0.96	78.2%
SpeedSeg (Tuned)				88.6%
ClassySeg (UA)	95.6%	94.7%	0.95	74.6%
ClassySeg (UO)	99.0%	96.3%	0.98	85.7%

Table 1: Segmentation accuracy for various segmentation techniques. SpeedSeg Default and Tuned are the accuracies of SpeedSeg using default parameter values and user-optimized parameter values, respectively. ([HS11] reported only all-or-nothing accuracy for SpeedSeg Tuned.) ClassySeg UA and UO are the accuracies of ClassySeg for user-agnostic and user optimized conditions. P = precision; R = recall; F = f-measure; AoN = all-or-nothing accuracy.

The high recall of the naıve method demonstrates that the set of candidate points contains nearly all of the true segment points. The low precision, on the other hand, reveals that the candidate segment points include many false positives, so many in fact, that the naıve method did not correctly segment even a single stroke (AoN= 0%).

The remaining segmentation methods achieve much higher precision than the naıve approach, with only a small decrease in recall. It is interesting to note the apparent logarithmic relationship between f-measure and all-or-nothing accuracy. For low values of f-measure, a large change in f-measure results in only a small increase in all-or-nothing accuracy: going from an f-measure of 0.33 to 0.91 improves all-or-nothing accuracy by only 27.8%. By contrast, at higher values, a small change in f-measure results in a large change in all-or-nothing accuracy: going from an f-measure of 0.91 to 0.98 improves all-or-nothing accuracy by 57.9%.

We performed an analysis of variance (ANOVA) to determine if there was a significant difference between the all-or-nothing accuracy of ClassySeg and each of the other segmentation techniques. This analysis focuses on all-or-nothing accuracy, as it is the most stringent of the performance metrics. When trained in a user-independent fashion, ClassySeg (UA) achieves an all-or-nothing accuracy that is: significantly higher than that of SSD ($p < 0.001$); not statistically different than that of iStraw and SpeedSeg Default ($p = 0.26$ and $p = 0.31$ respectively); and significantly lower than that of SpeedSeg Tuned ($p < 0.001$). In the latter comparison, ClassySeg is at a disadvantage because SpeedSeg Tuned included user-specific training, while ClassySeg did not.

When trained on user-specific data, ClassySeg (UO) achieves an all-or-nothing accuracy that is significantly higher than that of SSD, iStraw, and SpeedSeg Default ($p < 0.001$, $p = 0.003$, and $p = 0.03$ respectively) and not significantly different than that of SpeedSeg Tuned ($p = 0.28$). Here, only the latter comparison considers comparable user-optimized evaluation conditions.

6. Conclusion

In contrast to previous stroke segmentation approaches which usually rely on empirically determined parameters and heuristics, we present ClassySeg, an automatic stroke segmentation technique that employs general-purpose machine learning techniques. ClassySeg begins by selecting all curvature maxima as candidate segment points. It then computes features for each candidate based on speed, curvature, and other geometric properties. These features are adapted from numerous prior segmentation approaches, effectively combining their strengths. These features are then used as input to a statistical classifier which is trained to distinguish true segment points from false ones. To improve performance, beam search is used to select the optimal subset of features to train the classifier.

When trained in a user-independent fashion, ClassySeg (UA) performed at least as well as existing state-of-the-art algorithms also trained in a user-independent fashion. With user-specific training, ClassySeg (UO) performed better than existing algorithms with user independent training, and as good as the best existing algorithm with user-specific training.

While ClassySeg performs accurately, perhaps its most important property is its generality. The approach can be naturally extended to include any number of features. It is likely that even better performance can be achieved using features beyond the initial set considered here. Additionally, because the approach utilized a statistical classifier, it can be easily trained to optimize performance for specific users, specific classes of shapes, and specific drawing hardware. Finally, this approach is simpler to implement than techniques based on heuristic procedures.

7. Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 0729422.

References

[BSP*09] BISCHEL D., STAHOVICH T., PETERSON E., DAVIS R., ADLER A.: Combining speech and sketch to interpret unconstrained descriptions of mechanical devices. In *IJCAI'09: Proceedings of the 21st international joint conference on Artificial intelligence* (San Francisco, CA, USA, 2009), Morgan Kaufmann Publishers Inc., pp. 1401–1406. 6

[DL97] DASH M., LIU H.: Feature selection for classification. *Intelligent Data Analysis 1* (1997), 131–156. 6

[GR00] GOTOH Y., RENALS S.: Sentence boundary detection in broadcast speech transcripts. In *in Proc. of ISCA Workshop: Automatic Speech Recognition: Challenges for the new Millennium ASR-2000* (2000), pp. 228–235. 6

[HFH*09] HALL M., FRANK E., HOLMES G., PFAHRINGER B., REUTEMANN P., WITTEN I. H.: The weka data mining software: an update. *SIGKDD Explor. Newsl. 11*, 1 (2009), 10–18. 6

[HS11] HEROLD J., STAHOVICH T. F.: Speedseg: A technique for segmenting pen strokes using pen speed. *Computers & Graphics 35*, 2 (2011), 250 – 264. 1, 2, 3, 5, 7

[IMKT97] IGARASHI T., MATSUOKA S., KAWACHIYA S., TANAKA H.: Interactive beautification: A technique for rapid geometric design. In *UIST '97* (1997), pp. 105–114. 1

[Jap01] JAPKOWICZ N.: Concept-learning in the presence of between-class and within-class imbalances. In *Proceedings of the 14th Biennial Conference of the Canadian Society on Computational Studies of Intelligence: Advances in Artificial Intelligence* (London, UK, 2001), AI '01, Springer-Verlag, pp. 67–77. 3

[JM92] JENKINS D. L., MARTIN R. R.: Applying constraints to enforce users' intentions in free-hand 2-D sketches. *Intelligent Systems Engineering 1*, 1 (1992). 1

[KHM98] KUBAT M., HOLTE R., MATWIN S.: Machine learning for the detection of oil spills in satellite radar images. In *Machine Learning* (1998), pp. 195–215. 3

[KK06] KIM D. H., KIM M.-J.: A curvature estimation for pen input segmentation in sketch-based modeling. *Comput. Aided Des. 38* (March 2006), 238–248. 3, 5

[LBKS07] LEE W., BURAK KARA L., STAHOVICH T. F.: An efficient graph-based recognizer for hand-drawn symbols. *Comput. Graph. 31* (August 2007), 554–567. 1

[PH08] PAULSON B., HAMMOND T.: Paleosketch: accurate primitive sketch recognition and beautification. In *Proceedings of the 13th international conference on Intelligent user interfaces* (New York, NY, USA, 2008), IUI '08, ACM, pp. 1–10. 1

[Qui93] QUINLAN J. R.: *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. 6

[SSD06] SEZGIN T. M., STAHOVICH T., DAVIS R.: Sketch based interfaces: early processing for sketch understanding. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses* (New York, NY, USA, 2006), ACM, p. 22. 1, 2, 3, 5

[Sta04] STAHOVICH T.: Segmentation of pen strokes using pen speed. In *AAAI Fall Symposium Series 2004: Making Pen-Based Interaction Intelligent and Natural* (2004). 2

[WEH08] 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). 2, 3, 4, 7

[Wei04] WEISS G. M.: Mining with rarity: a unifying framework. *SIGKDD Explor. Newsl. 6* (June 2004), 7–19. 3

[WPH09] WOLIN A., PAULSON B., HAMMOND T.: Sort, merge, repeat: an algorithm for effectively finding corners in hand-sketched strokes. In *SBIM '09: Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling* (New York, NY, USA, 2009), ACM, pp. 93–99. 2

[WSA07] WOLIN A., SMITH D., ALVARADO C.: A pen-based tool for efficient labeling of 2d sketches. In *Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling* (New York, NY, USA, 2007), SBIM '07, ACM, pp. 67–74. 2

[WWL07] WOBBROCK J. O., WILSON A. D., 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* (New York, NY, USA, 2007), ACM, pp. 159–168. 4

[XJ10] XIONG Y., JR. J. J. L.: A shortstraw-based algorithm for corner finding in sketch-based interfaces. *Computers & Graphics 34*, 5 (2010), 513 – 527. 2, 3, 4, 7

[YC03] YU B., CAI S.: A domain-independent system for sketch recognition. In *GRAPHITE '03: Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia* (2003), ACM, pp. 141–146. 2, 5, 7

[Zha99] ZHANG W.: *State-space search: Algorithms, complexity, extensions, and applications*. Springer, 1999. 6