Assignment 2 -- Corner Finding
CAP6105

Due: 10/2/09 11:59pm

This purpose of this assignment is twofold. First, it is designed to give you a feel for the intricacies and complexities of corner finding, an important tool used in gesture and sketch-based interfaces. Second, it is to give you experience in implementing a state of the art algorithm from a research paper and to study its performance.

Requirements

There are two main requirements for this assignment. First, you will implement IStraw, a corner finding technique that was published in the Eurographics Symposium on Sketch-Based Interfaces and Modeling in August 2009. The paper is attached to this document. Second, you will compare this algorithm to the cusp finding algorithm found in starPad.

Finally, as part of your deliverables, you are to provide a 1 page report on how IStraw compares to starPad's cusp finder.

Testing

To test the corner finding algorithms, create a set of polyines based on the ones found in the IStraw paper. You should create 5 to 10 samples of each polyline you find in the paper.

Deliverables

You must submit a zip file containing your source and any relevant files needed to compile and run your application. Also include your report and a README file describing what works and what does not, any known bugs, and any problems you encountered. To submit, you can email me your zip file.

Grading

Grading will be loosely based on the following:

70% correct implementation of IStraw
20% analysis of IStraw compared to starPad's cusp finder
10% documentation

EUROGRAPHICS Symposium on Sketch-Based Interfaces and Modeling (2009)
C. Grimm and J. J. LaViola Jr. (Editors)

Revisiting ShortStraw – Improving Corner Finding in Sketch-Based Interfaces

Yiyan Xiong† and Joseph J. LaViola Jr.‡

University of Central Florida, School of EECS, Orlando, FL USA

Abstract

We present IStraw, a new corner finding technique based on an analysis of the ShortStraw algorithm. Our analysis reveals several limitations in ShortStraw and we develop techniques to overcome them. We also present an extension to our corner finding approach for dealing with ink strokes that contain curves and arcs. An evaluation of our approach shows significant accuracy improvements over ShortStraw for polyline ink strokes with and without curves using an all-or-nothing accuracy metric while still maintaining ShortStraw's computational complexity.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation H.5.2 [Computer Graphics]: User Interfaces—Graphics user interfaces

1. Introduction

Corner finding is a fundamental component in creating pen-based interfaces. Since it is often used in the segmentation of ink strokes into lower level primitives, it is one of the most important steps in the process of free-form sketch recognition and understanding [KS05, AD05, HD04, HR07, PH08]. Corner finding is also used in the heuristic-based recognition of gestures, such as erasing ink using a pen scribble, circling a handwritten mathematical expression to invoke a recognizer [LZ04], or as part of a feature set in a machine learning algorithm [LJZ07]. Other uses of corner finding include pen-based word entry on virtual keyboards [ZKS05] and in sketching simple animations for 2D characters [TBP04].

Given corner finding's utility in building pen-based interfaces and the fact that finding corners accurately will, in many cases, help to determine the overall accuracy of a pen- or sketch-based recognizer, an accurate corner finding technique is essential. In 2008, Wolin et al. introduced ShortStraw, a simple and efficient corner finding algorithm that was shown to be highly accurate in both total correct corners and all-or-nothing corner accuracy benchmarks [WEH08]. In this paper, we revisit the ShortStraw algorithm by examining its components. We uncover several limitations with ShortStraw and present a new corner finding algorithm, IStraw, thats attempts to alleviate ShortStraw's shortcomings while maintaining ShortStraw's computational complexity. We also extend IStraw to deal with ink strokes with curves. An evaluation of our algorithm shows significant improvements in all-or-nothing corner accuracy compared to ShortStraw for polyline ink strokes. The evaluation also shows IStraw, with our curve finding extension, has significantly higher all-or-nothing corner accuracy than using ShortStraw alone and in combination with our curve finding extension.

In the next section we examine work related to corner finding followed by a discussion of the ShortStraw algorithm and its limitations. Section 4 presents IStraw, which handles ink strokes with both arcs and polylines in addition to just polyline drawings. Section 5 discusses the computational complexity of our improved approach and presents a series of experiments comparing IStraw to ShortStraw. Section 6 discusses our findings and Sections 7 and 8 present areas for future work and conclusions.

2. Related Work

There has been several algorithms developed to find corners in sketch-based interfaces. One approach looks for extrema in the portions of the curvature and speed data that lie beyond a given threshold, taking these points as stroke corners [SSD01, QWJ01, Sta04]. Sezgin et al. [SSD01] look for maxima of curvature where it is already high and minima of speed only when it is already low. After the system combines the set of candidate corners from both curvature

† e-mail: lucy@cs.ucf.edu
‡ email: jjl@eecs.ucf.edu

and speed data, a set of hybrid fits is found to detect the real corners. Other approaches to detect corners by estimating curvature directly from input data have also been developed [RW75, FYH97].

Kim and Kim created a new curvature measurement in their corner finding algorithm [KK06]. They avoid the need for arc length calculations because they resample the raw input such that adjacent resampled points have constant distance. This simplification allows for the curvature calculation to be defined as the direction change at a given point.

Another technique for finding corners is with a scale-based approach. Rattarangi and Chin smooth a stroke's points with a varying Gaussian scale in their system to eliminate noise to improve the corner detection process [RC92]. Sezgin improved upon this algorithm by using scale-space feature point detection [SD06], and Lee et al. developed a multi-scale corner finder by using the wavelet transform [LSC95].

Combining segmentation and primitive recognition together to find dividing points has also been utilized in corner detection [Yu03, HSN04]. For example, Yu recursively selects points that are farthest from the line passing through the first and last stroke (sub-stroke) points to split the original stroke (sub-stroke) into two sub-strokes until the segment can be approximated by one of the primitive shapes.

All the algorithms mentioned above rely on more advanced mathematical knowledge. However, Wolin et al. built an accurate and simple polyline corner finder, ShortStraw [WEH08], introducing the concept of "straws", which relies on a window of constant size to examine contiguous pieces of an ink stroke. This approach is in contrast to Teh and Chin's corner finder [TC89], which uses a variable window for each point during corner finding. Our algorithm, like ShortStraw, only uses "straws" of constant size to find possible corners and, in order to get higher accuracy, we set corner detection thresholds dynamically based on shape instead of changing window size during post-processing.

3. The ShortStraw Algorithm

ShortStraw is an accurate polyline corner finder that is easy to understand and implement [WEH08]. After resampling the input data, ShortStraw finds corners using both a bottom-up and top-down approach. In this system, users can draw polylines free-form while achieving a high total corners and all-or-nothing accuracy. Furthermore, the algorithm can be quickly integrated into sketch-based interfaces. However, there is still room to improve its accuracy and to extend the technique to deal with polyline ink strokes containing arcs and curves. In this section, we will discuss the implementation of ShortStraw and its shortcomings.

3.1. ShortStraw Implementation

The first pass of ShortStraw involves resampling the input data, which is an important component necessary for achieving high corner finding accuracy using Wolin et al.'s approach. The resampling algorithm used by ShortStraw is based upon [WWL04], but uses a different interspacing distance between points.

ShortStraw then finds corners with two steps: a bottom-up and top-down approach. First, ShortStraw defines the concept of "straws" from primitive information. A straw for a point at resampled point p_i is computed as:

$$straw_i = |p_{i-W}, p_{i+W}|$$

where W is a constant window equal to 3 and $|p_{i-W}, p_{i+W}|$ is the Euclidean distance between the resampled points p_{i-W} and p_{i+W}. The shorter the straw, the more likely the point will be a corner. The initial corner set is taken from the resampled stroke points that are a local minimum below a threshold t, based on the median of the computed straw list.

After the bottom-up approach, some higher-level processing is used to find missed corners and remove false positives. ShortStraw checks to see whether two adjacent corners pass a line test. If not, then there must be additional corners between these two corners, and the point with the minimum straw value will be added to the possible corner set. The process is repeated until all of the stroke segments between pairs of consecutive corners are lines. A collinear check is then run on subsets of triplet, consecutive corners. If the three corners are collinear, the middle one is not a real corner and is removed from the corner set.

3.2. ShortStraw Limitations

Although ShortStraw achieves outstanding accuracy compared to other corner finding algorithms [KK06, SSD01], there are still some issues ignored by Wolin et al. [WEH08]. During the bottom-up approach of ShortStraw, the first three and last three resampled points do not have straw values, given the window size $W = 3$ is constant. In addition, timing information can be useful for corner finding, since users prefer to slow down on the corner, but ShortStraw does not take advantage of the speed change.

In the top-down step, the triplet collinear check will be unreliable if some corners are missed between these points and may lead to a false deletion of the correct corner. Another issue with the ShortStraw approach is the best way to set the threshold for a collinear check. The constant threshold used by ShortStraw is not robust to all kinds of shapes. In addition, noise caused by resampling is also an issue.

Finally, ShortStraw only works well for polyline ink strokes (see Section 5.2) but not for ink strokes with curves and arcs. However, free-hand shapes with both lines, curves, and arcs are necessary in most sketch-based system. Thus, dealing with the removal of false positive corners on a curve is important to address.

4. IStraw - A New Corner Finding Approach

To improve the accuracy and extend the scope of ShortStraw we analyzed the issues listed above and developed techniques to alleviate them.

4.1. Straws

The original ShortStraw algorithm uses a window size W of 3, resulting in the straws of the first three and last three resampled points to not be computed, but remain a default number, 0. Thus, these resampled points might be selected as corners during a post processing step. We can set values for the straws for these points. Given the indices of the resampled points p_i where i goes from 0 to $N-1$, the computations are shown as follows

$$straws_1 \leftarrow DISTANCE(p_0, p_{1+w}) \times \frac{2w}{(w+1)}$$

$$straws_2 \leftarrow DISTANCE(p_0, p_{2+w}) \times \frac{2w}{(w+2)}$$

$$straws_{N-2} \leftarrow DISTANCE(p_{N-1}, p_{N-2-w}) \times \frac{2w}{(w+1)}$$

$$straws_{N-3} \leftarrow DISTANCE(p_{N-1}, p_{N-3-w}) \times \frac{2w}{(w+2)}.$$

Note that having the straws for the start point p_0 and end point p_{N-1} be zero is acceptable, since these two points will always be chosen as corners. Also note we found changing the threshold t from $MEDIAN(straws) \times 0.95$ to $MEAN(straws) \times 0.95$ tends to give higher corner detection accuracy based on our early experimentation.

4.2. Timing Information

By using timing information we can get missed possible corner candidates, due to the typical case that users are more likely to slow down while coming to a corner [SSD01]. So we introduced timing data into our corner finding algorithm to assist "straws". When resampling the ink stroke, we compute the timing information for each resampled point by calculating the mean time of all the raw points between the current resampled point and the previous one. Then during the bottom-up step, we look for extrema in those portions of the mean time that is beyond a threshold. Those points have the minimum speed in the local area.

4.3. Consecutive False Corners Avoidance

Consecutive false corners is a special case defined as missing a correct corner, caused by failing to detect a corner or falsely removing one, bringing about the false deletion of subsequent corners. This phenomena occurs because the missing corner decreases the reliability of the collinear test on the triplet in the top-down component of ShortStraw.

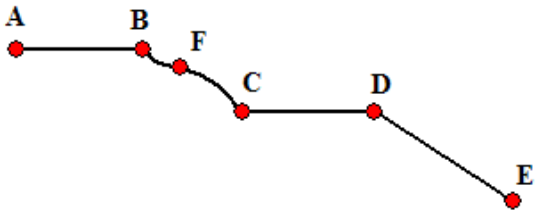

Figure 1: *An example of consecutive false corner deletion.*

Consider Figure 1. Points A to E are all the correct corners and point F is a unwanted corner. Since F is close to B, the Euclidean distance and the path distance from A to F have so little difference that B is defined as a wrong corner. The existence of F can lead to the false deletion of point B, then the triplet collinear check of point F will be with A, F, and C instead of B, F, and C. In this case, the system will leave F as a correct corner and go on to the next corner candidate C. Without deleting F, corner C will face the same problem as B and be identified as a unwanted corner.

To avoid this situation, it is necessary to delete F before the triplet collinear check of point B. We solve this problem by asking all the candidate corners go through the triplet collinear pass twice. The first pass has a higher threshold, and then we relax the threshold for the second pass. This dual pass approach will remove a false corner who distances between the previous and next corner are small (e.g., removing F in Figure 1).

4.4. Dynamic Threshold

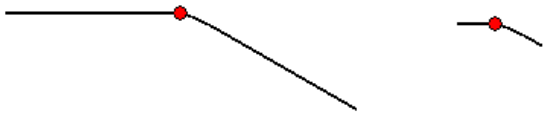

Figure 2: *Length of segment will affect corner decision: a longer segment (left) and a shorter segment (right)*

During our initial exploration of ShortStraw, we found, for example, that the point on the left stroke in Figure 2 is more likely to be a corner than the point on the right stroke in Figure 2, even if the angle between the two line segments for both strokes are equal. Thus, the threshold for calculating the initial corner set should change based on line segment length.

During the second collinear pass on any three consecutive corners, we set the threshold based on the length of the segment, if the difference between the first and third corner indices is larger than ten, then we increase the threshold from 0.974 to 0.98. In this case, length not only means the Euclidean distance between the first and the third points, but

also the difference between the resampled point indices of the two corners.

4.5. Sharp Noise Avoidance

Figure 3: *Two example of sharp noise: caused by hook (left) and caused by sharp angle (right).*

Sharp noise manifests itself in two situations. The first situation occurs in the start or end of the stroke (e.g. Figure 3 (left)). As we take the beginning and the end resampled points as corners, the hooks in a stroke normally will cause unwanted corners close to these points. The second situation exists with corners with sharp angles (e.g. Figure 3 (right)). This case is induced by the distortion of the stroke after resampling, which might change the shape from one sharp angle to two angles (e.g. Figure 4). Both situations can result in correct corners.

Figure 4: *Properly resampled sharp angle(left) and improperly resampled sharp angle (right).*

Often, two adjacent resampled points are both treated as corners. However, it is impossible for a user a draw a stroke with two corners so close together. Therefore, we can take one of the two adjacent resampled points as the correct corner to avoid sharp noise. In our approach, we choose the one with the smaller straw value, which has been empirically shown to achieve good results.

4.6. Curve Detection

Thus far, we have focused on strategies for improving the ShortStraw algorithm that works well for polyline-based ink strokes. However, these methods do not work well when strokes contain curves and arcs, finding many unnecessary corners on the curve. Therefore, we need an approach to decrease the false positives caused by the curves and arcs.

To remove unwanted corners, it is necessary to be aware of the difference between a real corner and a fake one. Ideally, a candidate corner c_i is the vertex of an angle defined by two rays generated from c_i and a resampled point on each side of the vertex. If it is a real corner, this angle will not

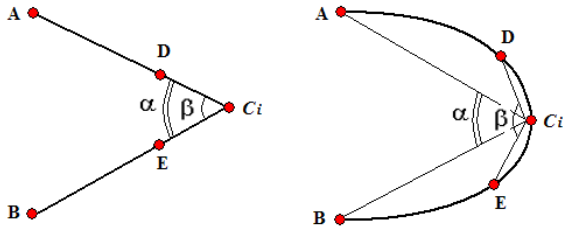

Figure 5: *Difference between the corner and the curve: the angle does not change with a real corner as the vertex (left) and the angle will increase with a false corner on a curve (right).*

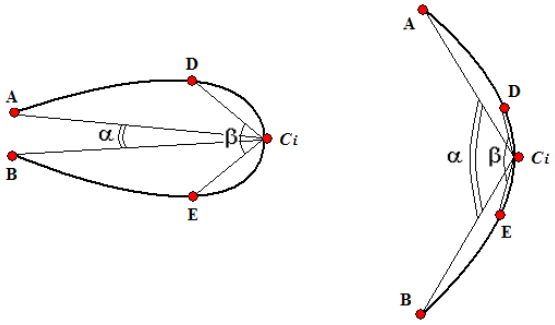

Figure 6: *Difference between* α *and* β *based upon the value of* α*:* α *is small (left) and* α *is large (right).*

significantly increase by choosing rays using other resampled points closer to the vertex. However, if it is a curve, this angle will get larger. This approach requires finding all possible angles from the resampled point data.

Instead of comparing all the possible angles, we can pick two representative angles for comparison to enhance efficiency. As in Figure 5, the further angle α is formed by c_i with the two resampled points A and B, whose indices are equal to the index of c_i plus/minus a shift value. The two points, D and E, for the closer angle β have the indices equal to the index of c_i plus/minus the shift value divided by 3. If $\beta - \alpha$ is below the threshold t_a, then c_i is a correct corner, otherwise it is a point on the curve. In our approach, the t_a is set dynamically based on the value of α. From Figure 6, we can see the value $\beta - \alpha$ will increase if α decreases, requiring t_a to be increased. t_a was derived empirically with to be between 14 and 33 degrees, calculated using $10 + 800/(\alpha + 35)$.

Setting the $shift$ value is crucial to the reliability of this approach. Figure 7 shows one possible case that can lead to a wrong decision if $shift$ is chosen unwisely. The best $shift$ value is the local minimum angle closest to the candidate corner, but this will sacrifice the simplicity of the algorithm. Thus, we chose $shift = 15$, which we determined empirically. However, if the previous corner, c_{i-1}, is too close to

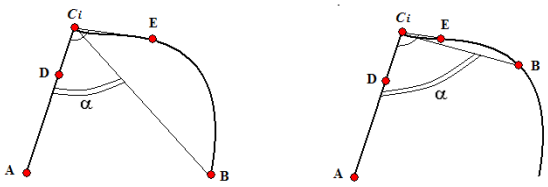

Figure 7: *Example 1 of unwise shift value: $\beta - \alpha$ is large even though c_i is a correct corner (left) and a smaller shift value which moves point B closer to c_i is needed to make the right decision (right).*

c_i, we change the value of $shift$ to the difference between the indices of these two corners. This approach also works for the corner c_{i+1}.

As with handling polylines, falsely deleting a correct corner may cause sequential problems, and we use the approach discussed in Section 4.3 by having all possible corners to go through the curve detection pipeline twice to avoid consecutive false corner deletion.

5. Evaluation

To evaluate IStraw, we focused on computational complexity of the algorithm and its accuracy compared to ShortStraw.

5.1. Analysis of Computational Complexity

In order to get higher corner finding accuracy, we developed IStraw by making several changes and adding new components to the ShortStraw algorithm. The question arises as to how these changes will increase the computational complexity of our approach compared with ShortStraw. To do this, we examine each change made and compare the computational complexity between our algorithm and ShortStraw. First, we set the number of raw points to M and number of resampled points to N.

During resampling, we use the same algorithm as the one in [WEH08] and the runtime is $O(M+N)$. For the polyline corner finding component, we did not modify the bottom-up component of the algorithm, $O(N)$, but use the speed data to add more potential corners. We have many enhancements in the top-down approach, but all these will not affect the computational complexity, so this part runs in time $O(CN)$ and the running time for the worst scenario will be $O(N^2)$. To avoid consecutive false corners, we need one more loop, whose iteration time is C, the number of corners. However, $O(2C)$ is equal to $O(C)$.

The last part of our algorithm is curve detection, and the algorithm contains two loops to remove the unnecessary corners. These two loops are similar. The iteration number of each one is C, so the computational complexity is $O(C)$. In conclusion, the computational complexity of our algorithm is $O(M+N^2+C)$, exactly the same as ShortStraw.

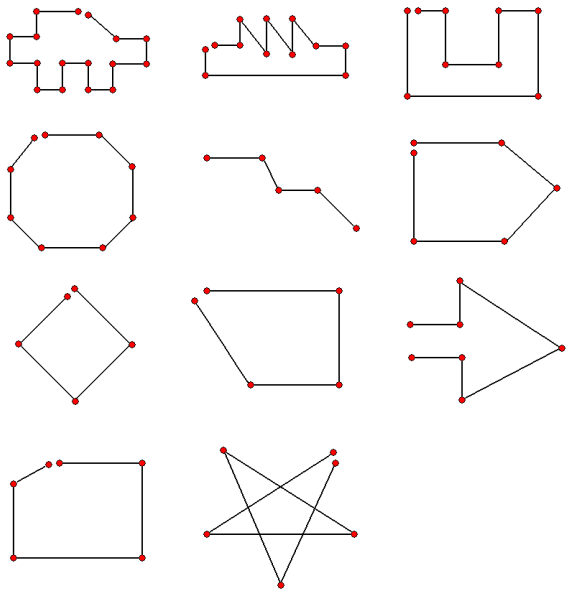

Figure 8: *The 11 polyline shapes used for corner finding testing from ShortStraw. There are 87 corners including the start and end points in total, which are marked with red points.*

5.2. Evaluation Tests

As in [WEH08], we use two different measures to determine the accuracy of IStraw. The first one, "Correct Corners Accuracy", described in [SSD01], is equal to the number of correct corners found divided by the total number of correct corners a human would perceive. The second one, "All-or-Nothing Accuracy", defined in [WEH08], take false corners into account, which means a correct stroke should have no false positives or negatives. This accuracy metric is calculated by dividing the number of correctly segmented strokes by the total number of strokes.

We used the test data, 244 polyline strokes in [WEH08] to configure the polyline ink stroke part of our algorithm. This set of data, which consists of the 11 shapes in Figure 8, was drawn by six users. The configuration data, 120 strokes in total, for curve detection was gathered from six students with the shapes from Figure 9.

To test IStraw, we collected data from 15 different users (6 female and 9 male) from computer science, electrical engineering or mechanical engineering fields. Nine out of the fifteen users had tablet PC experience. User wrote samples for 21 shapes including the 11 found in Figure 8 and the 10 in Figure 9. After getting familiar with the system, each user was asked to draw each shape four times. 1260 strokes were collected, but 14 were removed because they were very poorly written. Thus, our test set contained 1246 strokes, 656 from polyline ink strokes and 590 from curve ink strokes.

To make a thorough comparison, we tested two other algo-

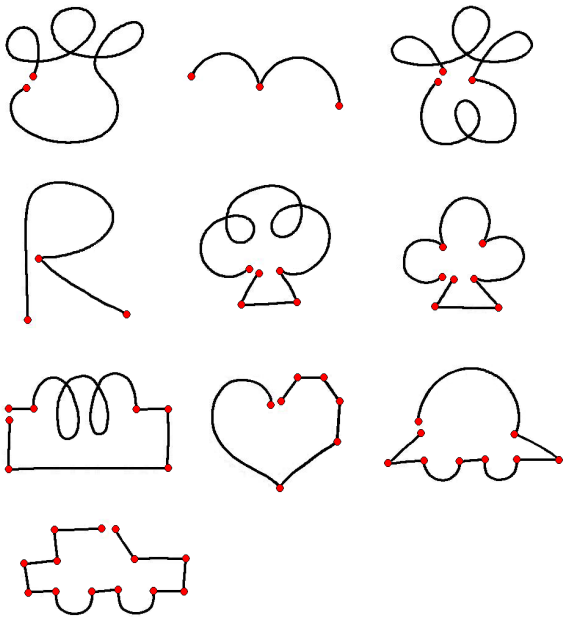

Figure 9: *The 10 new shapes with curves used for corner finding testing. There are 59 corners including the start and end points in total, which are marked with red points.*

rithm variations, in addition to ShortStraw and IStraw. The first one is ShortStraw+C, ShortStraw combined with our curve detection approach and the second is IStraw-C, our basic algorithm without curve detection.

5.2.1. Original ShortStraw Data

The results in Table 1 are based on the test set use in the ShortStraw paper, and we used it to help set the thresholds for IStraw-C. We also chose to use this data to ensure our implementation of the original ShortStraw algorithm had the same results as Wolin et al. [WEH08]. The results show that our ShortStraw implementation did indeed give us the same results as [WEH08] and that IStraw-C obtains very high correct corners and all-or-nothing accuracy with optimized parameters.

	ShortStraw	IStraw-C
False Positives	32	2
False Negatives	38	1
Correct Corners	1804	1840
Total Corners	1842	1842
Correct Corners Accuracy	0.979	0.999
All-or-Nothing Accuracy	0.741	0.998

Table 1: *Accuracy results for ShortStraw and IStraw-C, our algorithm without curve detection. The results are for the data used in the original ShortStraw paper.*

5.2.2. Polyline Ink Stroke Test

For the polyline ink stroke test, we wanted to examine IStraw with and without our curve detection extension to determine if it would cause any accuracy degradation for polyline ink strokes. Thus, we ran two separate tests on the polyline stroke part of our dataset. The results from Table 2 show accuracy values for ShortStraw compared to IStraw-C. Table 3 shows the accuracy results for both ShortStraw+C and IStraw.

	ShortStraw	IStraw-C
False Positives	32	2
False Negatives	93	12
Correct Corners	5059	5140
Total Corners	5152	5152
Correct Corners Accuracy	0.983	0.997
All-or-Nothing Accuracy	0.838	0.979

Table 2: *Accuracy results for ShortStraw and IStraw-C, our algorithm without curve detection (656 polyline strokes).*

	ShortStraw+C	IStraw
False Positives	30	1
False Negatives	168	21
Correct Corners	4984	5131
Total Corners	5152	5152
Correct Corners Accuracy	0.967	0.996
All-or-Nothing Accuracy	0.777	0.968

Table 3: *Accuracy results for ShortStraw+C, ShortStraw with our curve detection approach and IStraw, our complete corner finding algorithm (656 polyline strokes).*

5.2.3. Curve Detection Tests

To test whether IStraw works for strokes containing curves, we conducted experiments with the stroke data with curves dataset (see Figure 9). Table 4 show the results of testing ShortStraw and IStraw-C while Table 5 shows the test results of testing both ShortStraw+C and IStraw.

	ShortStraw	IStraw-C
False Positives	8297	8613
False Negatives	34	10
Correct Corners	3438	3462
Total Corners	3472	3472
Correct Corners Accuracy	0.990	0.997
All-or-Nothing Accuracy	0	0

Table 4: *Accuracy results for ShortStraw and IStraw-C, our algorithm without curve detection (590 strokes with curves).*

	ShortStraw+C	IStraw
False Positives	172	28
False Negatives	130	37
Correct Corners	3342	3435
Total Corners	3472	3472
Correct Corners Accuracy	0.963	0.989
All-or-Nothing Accuracy	0.634	0.908

Table 5: *Accuracy results for ShortStraw+C, ShortStraw with our curve detection extension and IStraw (590 strokes with curves).*

	ShortStraw+C	IStraw
False Positives	202	29
False Negatives	298	58
Correct Corners	8326	8566
Total Corners	8624	8624
Correct Corners Accuracy	0.965	0.993
All-or-Nothing Accuracy	0.709	0.940

Table 7: *Accuracy results for ShortStraw+C, ShortStraw with our curve detection extension and IStraw (1246 strokes with and without curves).*

6. Discussion

Our test results show that IStraw has significantly higher all-or-nothing accuracy over ShortStraw. For polyline shapes, IStraw-C, our algorithm without curve detection, improves the all-or-nothing corner finding accuracy from 83.8% to 97.9%. Adding the curve part to the algorithms will induce a lower accuracy, but IStraw still achieves an all-or-nothing accuracy of 96.8% compared to ShortStraw's 77.7%. In addition, our algorithm shows correct corners accuracy greater than 99.5

Both ShortStraw and IStraw-C did not perform well on the strokes with curves data (0.0% all-or-nothing accuracy) due to too many false positives. Although, the strokes cannot be properly segmented without curve detection, it is interesting to note that the correct corners accuracy is high, which means there are usually only a few corners that are false positives or negatives for each stroke. Once the curve detection component was added to our algorithm, the all-or-nothing accuracy improved to 90.8% compared to 63.4% with ShortStraw+C.

Tables 6 and 7 show the results of our algorithm and ShortStraw with and without curve detection on the complete dataset. In both cases, the results show IStraw has higher accuracy than ShortStraw, especially the all-or-nothing accuracy of our complete algorithm (94.0% versus 70.9%). Finally, Figures 10 and 11 show some examples of strokes from our dataset where corners were found correctly and incorrectly with the IStraw algorithm.

	ShortStraw	IStraw-C
False Positives	8326	8615
False Negatives	127	22
Correct Corners	8497	8602
Total Corners	8624	8624
Correct Corners Accuracy	0.985	0.997
All-or-Nothing Accuracy	0.441	0.515

Table 6: *Accuracy results for ShortStraw and IStraw-C, our algorithm without curve detection (1246 strokes with and without curves).*

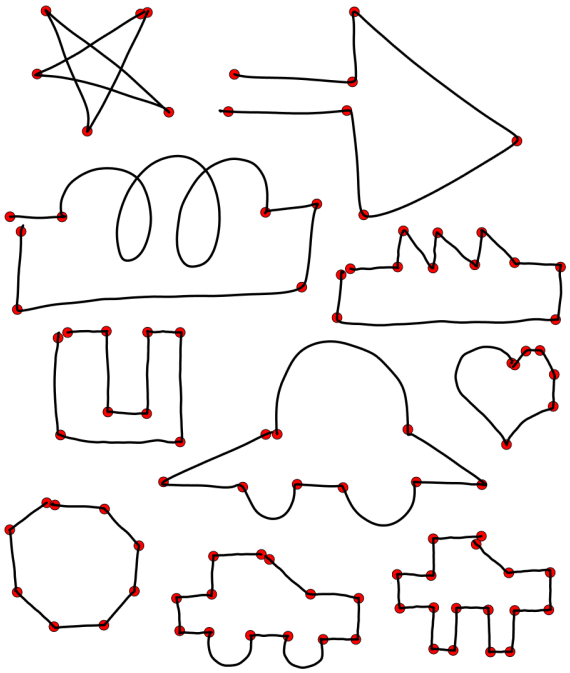

Figure 10: *Examples of correctly segmented strokes by IStraw. These strokes come from the set of 1246 symbols drawn by fifteen test users.*

7. Future Work

We see two potential areas for future work. First the question of how to optimize the parameters is important to our approach, since we need to set many parameters in deciding the thresholds. So far, we have chosen these values empirically. One way to fix this problem is to use numerical optimization techniques or machine learning algorithms to algorithmically find these parameters. Second, the distortion between the resampled points and the original stroke can cause corner detection problems so we would like to determine how to best utilize the original stroke data to reduce this distortion.

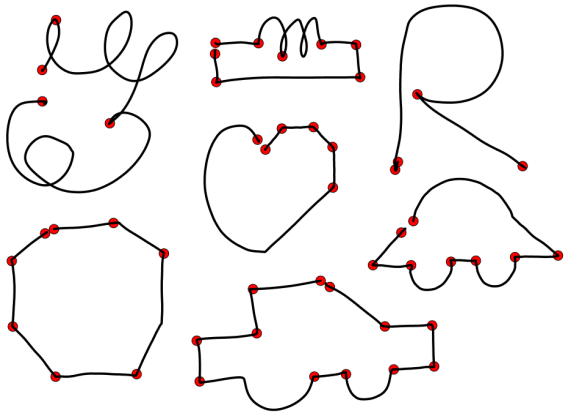

Figure 11: *Examples of incorrectly segmented strokes by IStraw.*

8. Conclusion

We have presented, IStraw, a new corner finding algorithm that handles both polyline ink strokes and ink strokes with curves. By analyzing ShortStraw, we have developed several new methods to overcome ShortStraw's shortcomings and have created a curve detection method for dealing with a large class of ink strokes. Our algorithm improves upon the state of the art in terms of all-or-nothing corner finding accuracy without increasing ShortStraw's computational complexity. Our approach is a fundamental step in sketch recognition, which will enable humans to express their ideas through sketch-based interface more efficiently and naturally.

Acknowledgements

This work is supported in part by IARPA, SAIC, and NSF CAREER award IIS-0845921. We also thank the anonymous reviewers for their valuable feedback.

References

[AD05] ALLVARADO C., DAVIS R.: Sketchread: A multi-domain sketch recognition engine. In *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology* (2005), vol. 29, pp. 518–532. 1

[FYH97] FU A. M. N., YAN H., HUANG K.: A curve bend function based method to characterize contour shapes. *Pattern Recognition 30*, 30 (1997), 1661–1671. 2

[HD04] HAMMOND T., DAVIS R.: Ladder, a sketching language for user interface developers. *Elsevier, Computers and Graphics* (2004), 35. 1

[HR07] HOU S., RAMANI K.: Classifier combination for sketch-based 3d part retrieval. In *Computers and Graphics* (2007), vol. 31, pp. 598–609. 1

[HSN04] HSE H., SHILMAN M., NEWTON A. R.: Robust sketched symbol fragmentation using templates. In *IUI'04: Proceedings of the 9th international conference on Intelligent user interfaces* (2004), pp. 156–160. 2

[KK06] KIM D., KIM M.-J.: A curvature estimation for pen input segmentation in sketch-based modeling. In *Computer-Aided Design* (2006), vol. 38, pp. 238–248. 2

[KS05] KARA L., STAHOVICH T.: An image-based trainable symbol recognizer for sketch-based interfaces. In *Computers and Graphics* (2005), vol. 29, pp. 501–517. 1

[LJZ07] LAVIOLA JR. J. J., ZELEZNIK R. C.: A practical approach for writer-dependent symbol recognition using a writer-independent symbol recognizer. *IEEE Trans. Pattern Anal. Mach. Intell. 29*, 11 (2007), 1917–1926. 1

[LSC95] LEE J.-S., SUN Y.-N., CHEN C.-H.: Multiscale corner detection by using wavelet transform. *Image Processing, IEEE Transactions on 4* (1995), 100–104. 2

[LZ04] LAVIOLA J., ZELEZNIK R.: Mathpad^2: A system for the creation and exploration of mathematical sketches. *ACM Transactions on Graphics 23*, 3 (Aug. 2004), 432–440. (Proceedings of SIGGRAPH 2004). 1

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

[QWJ01] QIN S. F., WRIGHT D. K., JORDANOV I. N.: Online segmentation of freehand sketches by knowledge-based nonlinear thresholding operations. *Pattern Recognition 34* (2001), 1885–1893. 1

[RC92] RATTARANGSI A., CHIN R.: Scale-based detection of corners of planar curves. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 14* (1992), 430–449. 2

[RW75] ROSENFELD A., WESZKA J. S.: An improved method of angle detection on digital curves. *IEEE Trans. Comput. 24*, 9 (1975), 940–941. 2

[SD06] SEZGIN T., DAVIS R.: Scale-space based feature point detection for digital ink. In *SIGGRAPH '06: ACM SIGRRAPH 2006 Courses* (New York, NY, USA, 2006), ACM, p. 29. 2

[SSD01] SEZGIN T., STAHOVICH T., DAVIS R.: Sketch based interfaces: Early processing for sketch understanding. In *Workshop on Perceptive User Interfaces* (2001). 1, 2, 3, 5

[Sta04] STAHOVICH T.: Segmentation of pen strokes using pen speed. In *Proceedings 2004 AAAI Fall Symposium on Making Pen-Based Interaction Intelligent and Natural* (2004). 1

[TBP04] THORNE M., BURKE D., PANNE M.: Motion doodles: an interface for sketching character motion. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers* (2004), pp. 424–431. 1

[TC89] TEH C., CHIN R.: On the detection of dominant points on digital curves. *IEEE Trans. Pattern Anal. Mach. Intell 17* (1989), 859–872. 2

[WEH08] WOLIN A., EOFF B., HAMMOND T.: Shortstraw: A simple and effective corner finder for polylines. In *EUROGRAPHICS 5th Annual Workshop on Sketch-Based Interfaces and Modeling* (2008), pp. 33–40. 1, 2, 5, 6

[WWL04] WOBBROCK J., WILSON A., LI Y.: Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *Proceedings of the Nineteeth National Conference on Artificial Intelligence (AAAI-04)* (San Jose, California, USA, 2004), pp. 159–168. 2

[Yu03] YU B.: Recognition of freehand sketches using mean shift. In *IUI '03: Proceedings of the 8th international conference on Intelligent user interfaces* (2003), ACM, pp. 204–210. 2

[ZKS05] ZHAI S., KRISTENSSON P.-O., SMITH B. A.: In search of effective text input interfaces for off the desktop computing. In *Interacting with Computers* (2005), vol. 17, pp. 229–250. 1