Polygon Recognition in Sketch-Based Interfaces with Immediate and Continuous Feedback

Peter Agar
Department of Computer Science, University of Auckland
paga001@ec.auckland.ac.nz

Kevin Novins
Department of Computer Science, University of Auckland
novins@cs.auckland.ac.nz

Abstract

Sketch-based drawing interfaces attempt to combine the ease of freehand drawing with the advantages of computer processing by analysing hand-drawn pen strokes in real time and replacing them with perfect geometric representations. This paper describes the development of a sketch-based interface for drawing polygons. An investigation into how people draw polygons revealed that it is possible to detect the corners of a polygon while it is being drawn by examining the time stamps of the sample points on the user's stroke. Results of a pilot user study suggest that this method is more pleasant and convenient to use than conventional polygon drawing tools and would be useful for composing drawings quickly. However, users still prefer the traditional method of clicking in the location of each corner for drawing accurately.

CR Categories: I.3.6 [Computer Graphics] Methodology and Techniques – Interaction Techniques. I.5.5 [Pattern Recognition] Implemention – Interactive Systems.

Keywords: Sketch-based user interfaces, corner detection.

1 Introduction

Drawing using pen and paper comes naturally to nearly all of us, yet in the majority of drawing programs, users have to draw in a manner that is unnatural and at times quite frustrating. Users frequently have to search through menus or an array of buttons to choose a specific tool to draw the particular shape that they require, and often the method to draw this shape is awkward or confusing. In an effort to avoid such a cumbersome method of drawing, what are known as sketch-based interfaces [Sezgin et al. 2001] are being developed to allow users to draw as they would with pen and paper, but with the added benefits of a computer drawing package. These interfaces use on-line graphics recognition [Wenyin et al. 2001] to interpret the pen strokes of hand-drawn sketches as geometric figures, usually replacing the imperfect stroke with its perfect representation.

There are many different methods of determining what this perfect representation is. The majority of these algorithms are run on the data only after the user has finished drawing. This means that users cannot be sure of how their sketch will be interpreted as they go along. They cannot make any corrections to their sketch to ensure that they achieve the shape they intended to draw.

On the other hand, the Fluid Sketching paradigm [Arvo and Novins 2000] avoids this problem by providing users with immediate feedback on how the user's pen stroke is being understood, as it is drawn. By continuously analysing the user's sketch, attempting to identify what the intended ideal shape is, and morphing the stroke to fit this ideal shape, users can see exactly how their drawing is being interpreted and make adjustments accordingly.

We have adopted this method of continuous and immediate feedback in a sketch-based interface to develop a tool for recognising and drawing open polylines or closed polygons. The essential problem in identifying what polyline the user intended to draw is the identification of its corners.

Most of the corner detection algorithms, such as least squares fitting [Shpitalni and Lipson 1997] or detection of high curvature points [Chetverikov and Szabo 1999], require that either all the data, or at least sufficient data after the last intended corner, be available for the algorithm to work. As such, these algorithms are not well suited to an interface where the corners must be detected as soon as they are drawn. Sezgin et al. [2001], however, acknowledge the intuitive notion that users slow down when they approach and arrive at each corner of their polygon. Although their method requires the drawing to be complete before it could identify these corner points, the concept of using time as a tool in analysing pen strokes proved to be extremely useful in a real-time application.

2 Analysis of User-Drawn Corners

The crucial aspect in recognising a user drawn stroke as a polyline is the identification of the shape's corner points. Before developing a corner detecting algorithm, we decided to study how people draw polygons freehand using a computer system.

In order to gather data, we asked the user to perform a sequence of simple tasks. Each task requires the users to draw one of the shapes indicated in Figure 1 on a virtual drawing canvas, then to indicate on their drawing where they intended their corners to appear. The application provides users with basic undo capabilities to correct any errors they may have made, such as indicating a corner in the wrong position.

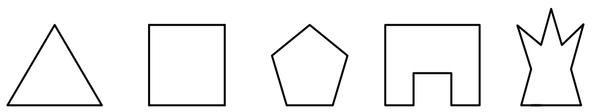
Figure 1: In the data gathering application, users were required to draw the shapes above.

As the user completes each shape and indicates on it where each of the corners are (see Figure 5(a)), the data is automatically saved to a file that stores all the sampled pen positions, along with their time stamps, as well as with the location of all the user's intended corners. This data could then be accessed and analysed to try and identify features of how people draw polygons that may be useful in a real-time corner detection algorithm. The subjects were eight colleagues who had differing experience with drawing programs.

To aid analysis of the data, we developed a function that allowed saved sketches to be replayed on screen as if the original user was drawing them. This provided us with the ability to closely examine the movements users make when attempting to draw polygons within a computer interface. By studying all the data it became apparent that although users attempted to draw straight lines between each corner, this was frequently not the case as can be seen in Figure 2(a). However, it also appeared that the speed of the user's stroke varied as the user drew, slowing down as it approached an intended corner.

In Figure 2(b), we show a visualisation in which the intensity of each line segment indicates the speed of the stroke at each section. If the time difference between a pair of consecutive sample points is relatively small (< 30ms) the line is drawn bright white. As this time difference gets greater, the colour gradually fades to black, which represents a time difference of about 250ms. This intensity coding gave clear evidence that users slow their stroke down significantly at the corner points of their polygon.

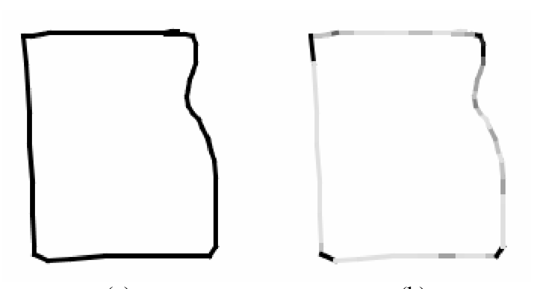

Figure 2: (a): A rectangle drawn by a user where an error typical of hand-drawn shapes can be seen on the right side of the shape. (b): The same shape, colour-coded so that the slower the stroke was drawn, the darker the line appears. This clearly shows that users slow down when they approach and draw corners.

3 Corner Detection Algorithm

The results of the investigation provided good evidence to suggest that corners on a user drawn polyline can be uniquely identified using the time stamps on each sample point rather than using geometric methods of corner detection. From this perspective two different directions presented themselves for further exploration: the algorithm could either use the time difference between samples, or it could use the velocity of the stroke by examining the time difference between each pair of samples and their distance apart, as suggested by Sezgin et al. [2001].

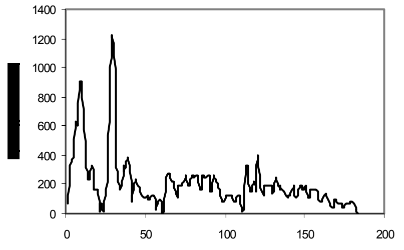

Figure 3(a): Speed graph for a stroke drawing a square.

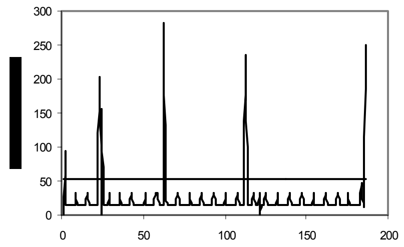

Figure 3 (b): Time difference graph for the same stroke with a threshold to identify maxima.

Although using the velocity of the stroke seemed more sensible, further inspection revealed that this data is excessively noisy, as in Figure 3(a), and although speed minima are visible, establishing a universal threshold is difficult, as observed by Sezgin et al. [2001]. On the other hand, time differences between samples provide extremely distinct maxima, as can be seen in Figure 3(b). Such well-defined maxima make identifying a threshold to classify a certain point as a corner entirely feasible.

The reason such a vast difference between velocity data and time data is visible is due to the way in which the points are sampled. The points are sampled using Java's built in MouseMotionListener, which can listen for mouse drag events. If the mouse or pointing device is stationary, no events are generated. When the mouse or pointing device is dragged (moved with the mouse button held down) events are fired at regular intervals. As such the time difference between each sample points is relatively constant, except at the corners. When the user's stroke arrives at a corner, the user pauses slightly, and since the mouse is not moving, no points are sampled until the mouse begins moving again. Hence, at the corners of the polygon there is a higher than usual time difference between two sample points. On the other hand, regular intervals between samples do not greatly affect the velocity measures of the stroke, and since users draw with a continually varying velocity, we see a considerable amount of noise in the data.

Once this essential feature about polygon sketching had been identified, writing a corner detection algorithm based on time data was relatively straightforward. The algorithm runs every 30ms. When a user begins to draw, a straight line is drawn from where they started drawing to the cursor's current position. As soon as a corner is detected, the line anchors at that corner, and then to the cursor. This continues with each additional corner. Figure 4 shows a new corner being recognised with a straight line always joining each corner, and the last corner with the current cursor position.

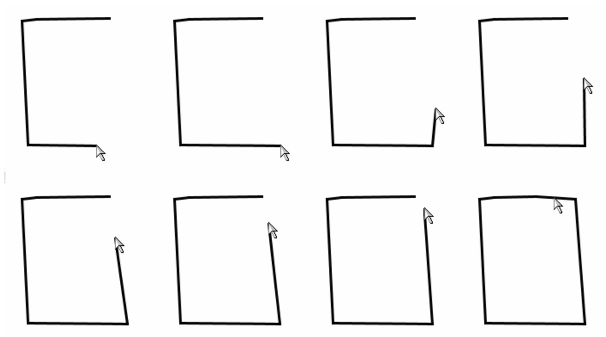

Figure 4: The stroke drawn in Figure 2 is continuously analysed to identify new corners. Straight lines are drawn between each corner and from the last corner to the current cursor position.

The algorithm loops through each of the sample points, from the last sample point identified as a corner to the most recently sampled point, finding the time difference between each consecutive pair of samples. If the time difference is greater than the average time difference between samples (excluding corners) multiplied by a predetermined value (empirically ascertained to be 3.0), then the first of the pair of points is classified as a corner. This successfully identifies the corner points of a sketch, but because users usually slow down before they arrive at the actual corner, this method can sometimes pick up false positives right beside a correctly identified corner. Avoiding this is simply a matter of detecting whether a newly identified corner is less than a certain optimal distance (empirically determined as 14 pixels) from the last detected corner. If it is, the last detected corner is replaced by the new one.

As final touches, the algorithm allows corners to snap together and if the two end points of the polyline are near each other, the polygon is automatically closed. As each new corner is found, the algorithm searches through each of the existing detected corners. If it finds a corner within 10 pixels from the current corner, this new corner is "snapped" to the location of the other corner. When the user releases the mouse button, indicating that they have finished drawing, the algorithm checks to see if the two end points are within 30 pixels of each other. If they are, both points are relocated to the point where the first and last lines of the polyline would intersect. This has the result of closing the polygon if the first and last lines do not cross each other, or culling the ends of the lines if they overlap.

4 Analysis

During the earlier investigation phase (described in Section 2), drawings and intended corner locations were gathered from eight users and stored in files. We developed a program to run the algorithm and calculate statistics about the algorithm's accuracy on each user's file. By doing so, the program can, on completion, provide a set of measures indicating how well the algorithm performed on the whole data set, as well as individual measures to identify which files the algorithm performed particularly poorly on.

In order to produce a measure of how well the algorithm performed on a particular data file it is necessary to determine how many detected corners were not supposed to be identified (false positives), how many user corners were not identified (false negatives) and the average distance correctly identified corners were from their associated user corners. To do this we need to match detected corners to user corners. However finding an appropriate matching raises some difficulties.

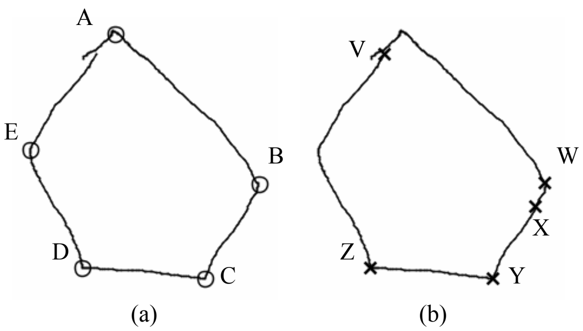

Figure 5: (a): A user drawn stroke showing the locations of the user's intended corners (o). (b): The same stroke showing corners detected by a corner detection algorithm (x).

In Figure 5 we see that the user had indicated points A, B, C, D and E as the corners of their sketch and the algorithm identified the points V, W, X, Y and Z. To estimate how well the algorithm performed on this data, we need to find a matching between user corners and detected corners that is 1-1 and minimises the distance between each pair of corners. A 1-1 matching is where each user corner is matched to only one detected corner and vice versa. If we were to match detected corners with their nearest user corners, then user corner B would be matched with both W and X, whereas X should be declared as a false positive. Conversely, if user corners were matched with their nearest detected corner, E would be matched with V when it should be declared as a false negative.

Matching corners 1-1 while minimising the total distance between pairs of corners is an instance of the assignment problem [Brassard and Bratley 1996] (our implementation used an assignment problem solver by Che [1997]). However, in its pure form, the assignment problem assumes that every corner must be matched with another corner. We need a system that allows for false positives and false negatives, thus we need to allow a corner to not be matched with anything. Also, the assignment problem requires equal numbers of user and detected corners.

These problems can be solved by having dummy corners [Belongie et al. 2001] that act as though they are 30 pixels from every other corner. By having as many dummy detected corners as there are user corners, we can ensure that each user corner is matched to something. This also guarantees that no user corner will be matched with a detected corner that is more than 30 pixels away, as it will always be more optimal for it to be matched with a dummy corner. Conversely, there are as many dummy user corners as there are detected corners to ensure that each detected corner has a dummy to match to. If a user corner is matched with a dummy, then it is declared as a false negative, similarly, if a detected corner is matched with a dummy, it is declared as a false positive.

A further enhancement to the testing program gives the user the ability to inspect individual files, providing them with a visual representation of the accuracy measurements. The program displays on screen the original user stroke, together with the corners identified by the user and the corners identified by the corner detection algorithm. It adds lines joining associated user and detected corners, and circles the false positives and false negatives. A sample output is shown in Figure 6.

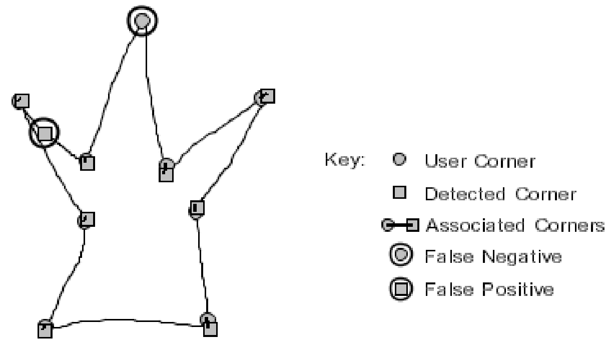

Figure 6: Output of the testing program showing how well a corner detection algorithm performed on a sample user stroke.

Using this program it became possible to write a corner detection algorithm and test it automatically on a large data set, with the program assessing how well the algorithm performed on the data. The program allowed us to confidently make alterations to the parameters of the algorithm, and to immediately see whether or not they provided better results. It also gave us the ability to further investigate specific anomalies to see exactly what was going wrong in those cases and provide insights into writing a more robust algorithm. The thresholds of spacing and pause times described in Section 3 were also obtained using this program.

However, it is necessary to acknowledge that this method of assessing accuracy is not foolproof. Such measures as a maximum spacing of 30 pixels are simply heuristics and cannot guarantee that false positives and negatives will be correctly identified. Nevertheless, in every one of the sampled strokes, the associations between user and detected corners and the identification of false positives and negatives corresponded perfectly with what we would intuitively identify, and the only situations where it seems likely that the method would fail (such as identifying corners that were consistent translations of user corners) are extremely unlikely to arise within this context.

5 User Evaluation

We designed an informal evaluation in which six colleagues participated. Participants were presented with a simple application that had only two drawing tools, labelled A and B. One of these was the real-time polygon detection tool described in this paper, and the other was a polygon drawing tool similar to that found in typical drawing packages, whereby a user manually indicates where they want their corners to be by clicking the mouse button in each location. With this method, a line always joins the last corner with the cursor's current position, until the time that the user indicates they have stopped drawing by double-clicking the mouse. To prevent order biases we alternated which tool was labelled A and which was labelled B. For each method, we explained how to use the tool, gave the participant a demonstration of how to use it by drawing a square, a pentagon and a star, and asked them to try it out for themselves, letting them draw whatever shapes they wished. Once they had tried both of the methods, we asked them to tell us what they felt about each method, which they preferred and why, and under what circumstances, if any, they would use the other method of drawing polygons.

This study provided useful feedback on the benefits and drawbacks of the real-time polygon recognition tool. Participants stated that the sketching style of drawing is more pleasant to use than the traditional method. They also said it was more convenient having the tool automatically detect the corners as they drew, and that they found that it was actually faster to use than with the traditional method, since users actually pause for longer at each corner if they have to click the mouse button. However, the participants also identified disadvantages in using this method of drawing polygons. They noted that the tool was less forgiving of errors made while drawing, such as overshooting your target: if you miss where you were aiming for, stop and move the cursor back to where it should have been, a corner is usually added at the point where you stopped. Similarly, participants said that it didn't allow them to change their mind about what they wanted to draw. With the traditional polygon drawing tool, they could stop for a while before deciding where they wanted the next corner to be, whereas the automatic polygon tool did not allow for this. In summary, they stated that although the automatic method was great for assembling a quick diagram, the point and click method was better for drawing something very precisely. Overall three of the participants preferred the automatic polygon recognition tool, one had no preference, and the other two said they still preferred the traditional method.

6 Future Work

An immediate enhancement that could be made to the polygon-drawing tool was identified during user testing. The system needs to allow the user to correct mistakes made either by the user or by the recognition algorithm. One possible solution to overshooting a mark, could be to provide a form of backtracking, allowing the user to reverse their stroke over what they have just drawn in order to erase that part of the sketch and letting them redraw the shape correctly. Additionally the tool could have its own automatic error correction techniques, perhaps combining other forms of corner detection, as suggested by Sezgin et al. [2001], to see if a detected corner should really have been detected.

Another interesting feature would be the regularisation of polygons as they are being drawn. This would allow users to easily sketch any regular polygon. The system might also assist the user in aligning polygons with the axes. For these cases, the recogniser would almost certainly work on thresholds. For example, if a certain quadrilateral were almost a square, then a perfect square would be drawn instead; if it were almost axis aligned then it would be drawn axis-aligned, otherwise it would remain unchanged. Incorporating these features would result in a more complete and intelligent drawing tool.

This paper represents a small step in the direction of a new style of interface design. As user tests have demonstrated, sketch-based interfaces are easier and more enjoyable to use. Further research and development of tools such as the one discussed in this paper, combining error checking and editing abilities would lead to a natural and user-friendly style of drawing that could surpass current techniques to become the preferable way to draw.

References

ARVO, J., AND NOVINS, K. 2000. Fluid Sketches: Continuous Recognition and Morphing of Simple Hand-Drawn Shapes. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, ACM Press, 73-80.

BELONGIE, S., MALIK, J., AND PUZICHA, J. 2001. Matching Shapes. In *Proceedings of the ICCV2001*, Vancouver , vol. 1, 454-461.

BRASSARD, G., AND BRATLEY, P. 1996. *Fundamentals of Algorithmics*. New Jersey: Prentice-Hall.

CHE, Y. 1997. *Assignment Problem Solver*, Available: http://www-fp.mcs.anl.gov/otc/Guide/CaseStudies/assign/.

CHETVERIKOV, D., AND SZABO, Z. 1999. A Simple and Efficient Algorithm for Detection of High Curvature Points in Planar Curves. In *Proceedings of the 23rd Workshop of the Austrian Pattern Recognition Group*, 175-184.

SEZGIN, T. M., STAHOVICH, T., AND DAVIS, R. 2001. Sketch Based Interfaces: Early Processing for Sketch Understanding. In *Proceedings of the Perceptive User Interfaces Workshop (PUI2001)*.

SHPITALNI, M., AND LIPSON, H. 1997. Classification of Sketch Strokes and Corner Detection Using Conic Sections and Adaptive Clustering. *ASME Journal of Mechanical Design*, 119(2):131-135.

WENYIN, L., QIAN, W., XIAO, R., AND JIN, X. 2001. Smart Sketchpad-An On-line Graphics Recognition System. In *Proceedings of the ICDAR2001*, Seattle, 1050-1054.

The Least-Squares Line. 2002. Wolfram Research, Available: http://www.efunda.com/math/leastsquares/lstsqr1dcurve.cfm.