

Towards a Framework for Truly Natural Low-level Sketch Recognition

Brandon Paulson
Sketch Recognition Lab
Texas A&M University
bpaulson@cse.tamu.edu

Tracy Hammond
Sketch Recognition Lab
Texas A&M University
hammond@cse.tamu.edu

ABSTRACT

Although stroke-based systems may be considered the state-of-the-art in low-level sketch recognition, they still contain constraints and intricacies that may be invisible to most novice users. In this paper, we identify some common assumptions and problems of stroke-based systems and propose a plan for the development of a new low-level framework to deal with these issues. The broader impact of this framework will be the development of sketch recognition systems which place fewer (and hopefully no) drawing constraints on users and will allow for more natural sketching, starting at the lowest and most fundamental level.

Author Keywords

sketch recognition, primitive recognition, low-level framework

ACM Classification Keywords

H.5.2 Information Interfaces and Presentation: User Interfaces—*Input devices and strategies*

INTRODUCTION

As computers become an integral part of our lives, it becomes increasingly important to make working with them easier and more natural. Our vision is to make human-computer interaction as easy and natural as human-human interaction. As part of this vision, it is imperative that computers understand forms of human-human interaction, such as sketching. Computers should be able to understand the information encoded in diagrams drawn by scientists and engineers. Sketch recognition is the automatic understanding of free-hand drawing and is a natural modality of human-computer interaction for a variety of tasks; for example, hand-drawn graphical diagrams, which are pervasive throughout education and conceptual design.

One goal of our lab is to develop a new framework for fundamental low-level sketch recognition that allows the complete freedom of natural, unconstrained sketching by the user. By

low-level sketch recognition, we refer to the recognition of simple primitive shapes (such as lines, arcs, circles, etc.) which can be combined by a higher-level sketch recognition system using geometric and spatial constraints to form more complex shapes [2, 3].

PREVIOUS WORK

Algorithms previously used to perform low-level sketch recognition fall into two philosophies: vision-based methods and stroke-based methods. Vision-based methods [5, 6, 10] attempt to look solely at the pixel values drawn to the screen by the user. Most of these methods rely on machine learning methods which require many training examples or templates, and do not allow for freedom in orientation, rotation, or, in some cases, scale without the explicit training of every possible shape configuration (which can be infinite).

On the other hand, stroke-based methods look both at the pixel values and the timestamps of the points that were drawn to the screen. Within the stroke-based philosophy lay two approaches: gesture-based and geometric-based. Gesture-based systems [1, 8, 11] attempt to recognize strokes by analyzing features related to **how** the stroke was drawn. Because gesture-based algorithms analyze the how of a stroke, most of these systems place constraints on how users must draw these shapes (e.g. a user must draw all circles counter-clockwise), or they force the user to specify multiple templates for each possible variation of the gesture.

To avoid these drawing constraints, more recent approaches have focused more on **what** the final stroke looks like [9, 7, 13]. To allow for variance in scale and orientation, these methods use geometric formulas to describe shapes; this is the current state-of-the-art in low-level sketch recognition. And, while these methods claim to allow “natural and unconstrained” sketching [4], there is still a fundamental deficiency with the current methods - *every stroke is assumed to have a significant meaning to recognition*. This deficiency is not present in vision-based systems, which (as mentioned before) have their own faults. We aim to create a better stroke-based system (at the lowest level) that can handle and resolve many of the problems with current stroke-based algorithms.

ISSUES WITH STROKE-BASED METHODS

When a sketch recognition system encounters a new user, he or she is unaware of the constraints and intricacies of stroke-based methods. Issues arise when dealing with the following



Figure 1. Examples of continuation strokes (shown in black; previously recognized shapes shown in color); current systems will recognize the continuation strokes as separate primitives rather than combining the stroke with a previously drawn stroke.



Figure 2. Examples of overtraced strokes (drawn with a single or multiple strokes).

types of strokes:

1. *unintentional strokes* - strokes due to inadvertent contact with the screen.
2. *continuation strokes* - strokes that are meant to either extend an already recognized primitive (such as extending a line) or to form a new primitive (such as two arcs joining to become a circle), seen in Figure 1.
3. *overlapping/overtraced strokes* - a common occurrence with people who like to doodle while sketching (Figure 2).
4. *touch-up strokes* - strokes meant to “clean up” shapes and sketches (Figure 3). Often, new users believe they are helping the sketch recognizer by “cleaning” up their drawings when, in actuality, they are impeding recognition.

With current state-of-the-art frameworks, these four types of strokes are incorrectly added as additional recognized shapes. Furthermore, most primitive recognizers are limited solely to single strokes; drawing primitives using multiple strokes is not an option without further restricting the user, such as with temporal constraints (i.e. the user must wait x seconds before drawing the next shape).

PROPOSAL

Current frameworks typically follow a linear progression, with feedback loops only occurring during the high-level



Figure 3. Example of a “touch-up” stroke (red); in this case the rectangle may have already been recognized without the additional touch-up stroke.

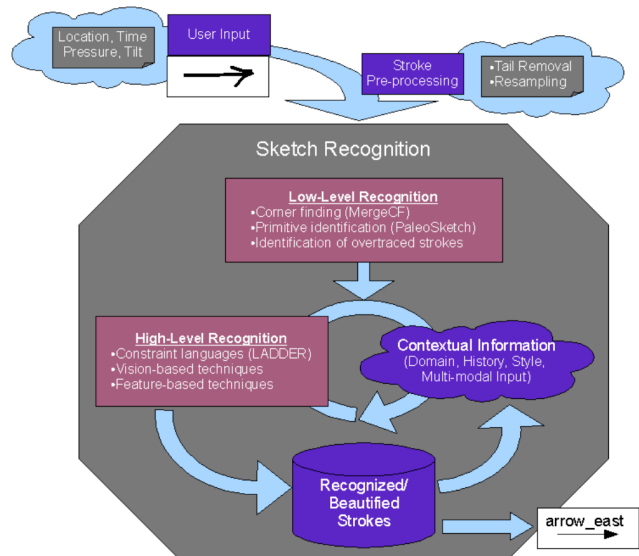


Figure 4. Example of current stroke-based frameworks (taken from [4]).

recognition stage when resolving contextual cues (see Figure 4) [2, 3]. When a stroke is processed by the low-level system, it is sent (as a recognized primitive) to the high-level component, which then attempts to pigeonhole the primitive into one of many shape definitions. The main problem with this approach is that it assumes each stroke has a higher-level meaning and must be used for recognition. However, as seen previously, not all strokes are meant to provide additional recognition information.

Our proposal is to extend current low-level frameworks to be more than simple primitive recognizers. We believe low-level systems should be smart enough to determine occurrences of touch-up strokes, continuation strokes, overlapping strokes, and unintentional strokes. Low-level systems with these capabilities will yield sketch systems that work better for novice users “straight out of the box” without requiring the user to learn the intricacies of stroke-based approaches.

PLAN/FUTURE WORK

Step 1: User Studies

Our first goal will be to conduct user studies (particularly with novice users) to determine their initial assumptions and tendencies when using a sketch recognition system. Some questions we hope to answer from this study are:

1. What is the frequency of unintentional marks made by the user? How does this frequency change over time or with experience?
2. When drawing primitives, do users tend to draw them using single strokes or multiple strokes? Is there a need for a multi-stroke primitive recognizer?
3. Is our initial list of observed (and currently unhandled) stroke types a sufficient categorization? Are there other types of strokes that pose problems for sketch recognition systems and should be handled differently?

- Are novice users initially aware of the intricacies of stroke-based systems? For the example of touch-up strokes, do users realize that cleaning up their sketch may actually be degrading recognition performance?
- What are the effects of real-time beautification on recognized strokes? Will users be more or less likely to draw multi-stroke primitives if beautification takes place after each stroke is drawn?

Step 2: Handling Multi-Stroke Primitives

The next step in our process will be to create algorithms which handle a large number of multi-stroke primitives. Because one of our goals is to allow multiple primitives to be drawn on the screen at the same time, this not only becomes a stroke combination problem, but also a grouping and segmentation problem (to avoid trying every possible combination in the power set of strokes on the screen).

- Stroke grouping and segmentation: Before combining strokes into primitives, we first need to determine which strokes are meant to be combined. Temporal information may not be sufficient in the case of stroke interspersing (finishing up a primitive after drawing other primitives). Likewise, spatial information by itself is not sufficient in the case of overlapping primitives.
- Stroke combination: many recognizers rely heavily on direction and curvature values [9, 7, 13], therefore, simply combining strokes at the endpoints is not sufficient. Issues that need to be addressed include: handling stroke interspersing, smoothing time values between endpoints (to maintain derivative values), and resolving stroke overlap.

Our initial hope is that overlapping and touch-up strokes will simply be sub-cases to the multi-stroke primitive recognition problem. However, special cases and rules may need to be developed to handle these types of strokes.

Step 3: Filtering Unintentional Strokes

Unintentional strokes need to be handled independently of normal strokes which are meant to be recognized. Determining unintentional strokes requires not only simple information like stroke size and position, but also some insight into the intention of the user. We need to determine cases when a marking is unintentional or just meant to be a very small line. We plan to analyze features related to speed, curvature, and (when available) pen pressure to help determine when a user makes inadvertent marks. A large difficulty associated with this problem will be the ability to collect enough data (containing truly unintentional marks) to sufficiently generalize the problem.

Step 4: Analyzing Effects of Beautification

Beautification has already been handled for single stroke primitives in previous work [7]. However, beautification may have different implications for overtraced and multi-stroke primitives. Primitives that were overtraced may have a special meaning, and may possibly (when beautified) need

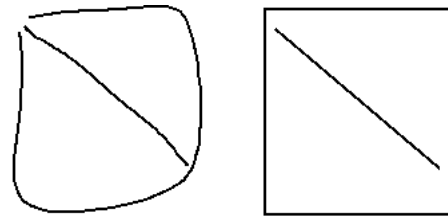


Figure 5. Example of two strokes originally drawn (left) and beautified (right). In this case, a high level system may define this shape with a constraint saying that “the endpoint of the line meets the bottom right of the rectangle.” This case is closer to being met in the original stroke than it is in the beautified version. However, if the rectangle were beautified before the line was drawn, the user may have been more likely to have drawn the line such that it met the constraint better in the beautified version.

to be highlighted or made bold to denote this special meaning. Determining this meaning may require domain-specific context at a lower level; a departure from previous architectures [2]. We also need to analyze the effects of real-time beautification for multi-stroke primitives and determine if those effects hinder high-level recognition. Figure 5 shows an example of how beautification can affect high level recognition.

Step 5: Improving Corner Finding & Primitive Recognition

Although much has been done to improve primitive recognition, it is still in need of advancement. In particular, corner finding (segmenting single strokes into multiple primitives) is still not as accurate as it should be. The current best polyline corner finding algorithm only achieves an all-or-nothing accuracy of 86% [12]. This problem alone already introduces 14% error into primitive recognition.

One of the plans to improve corner finding is to create ensemble techniques which combine interpretations of multiple corner finding algorithms. While no single algorithm has achieved higher than 86% accuracy, we believe that creating an algorithm that combines the interpretations of various techniques could boost accuracy rates to a higher level.

Minus the error due to corner finding, primitive recognition has already achieved high recognition rates (over 98% for 9 shapes [7]). With the idea of multi-stroke primitives comes the question of what makes a primitive; the rectangle is a classic example. A rectangle is a shape that commonly occurs in multiple sketch domains and thus is worthy, in some researchers’ eyes, of being called a primitive. Others argue that a rectangle is composed of four lines and thus not a primitive. However, if an algorithm supports multi-stroke primitives, then the argument of having a rectangle be considered a primitive gains weight.

Our goal will be to increase the number of primitive shapes past the current state-of-the-art. PaleoSketch currently recognizes lines, polylines, circles, ellipses, arcs, curves, spirals, helices, and complex combinations of these [7]. We aim to add other common primitive shapes such as rectangles, diamonds, squares, filled-in dots, arbitrary polygons,

and sinusoidal waves (just to name a few).

Step 6: Development of New Low-Level Framework

To allow multi-stroke primitives means the creation of a new sketch recognition framework. Current frameworks (such as LADDER [3]) treat every incoming stroke as a primitive (or combination of primitives). Multi-stroke primitives must be explicitly (and unnecessarily) re-defined in every domain. Each stroke is recognized by the low-level recognizing and sent to high-level system which performs grouping and recognition of higher-level symbols.

Our new framework aims to remove grouping from the high-level recognizer and insert it into a lower-level system. When a stroke is received it will first be sent to our filtering algorithm which determines if the stroke is intentional. The stroke will then be given an initial primitive label. As more strokes are received, they are grouped together (as appropriate) and given new primitive labels. Our new framework will be much less linear than current frameworks, allowing for a cyclical exchange of information between the grouper, corner finder, and primitive recognizer.

CONCLUSION

Stroke-based methods cannot be considered truly natural until many of the assumptions made by these systems are removed. We have identified an initial set of issues and problems (including unintentional strokes, continuation strokes, overlapping strokes, touch-up strokes, and multi-stroke primitives) that should be handled by stroke-based methods in order to create a sketch system that works for novice users “straight out of the box.” Our goal is to develop a new low-level framework that can easily handle and resolve these issues, and will eventually lead to truly natural low-level sketch recognition.

ACKNOWLEDGMENTS

This research is supported by the NSF IIS Creative IT Grant #0757557 Pilot: Let Your Notes Come Alive: The SkRUI Classroom Sketchbook.

REFERENCES

1. J. A. Chris Long, J. A. Landay, L. A. Rowe, and J. Michiels. Visual similarity of pen gestures. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 360–367, New York, NY, USA, 2000. ACM.
2. C. Alvarado, M. Oltmans, and R. Davis. A framework for multi-domain sketch recognition. In *Sketch Understanding Papers from the 2002 AAI Spring Symposium*, pages 24–31, 2002.
3. T. Hammond and R. Davis. Ladder, a sketching language for user interface developers. *Computers & Graphics*, 29(4):518–532, 2005.
4. T. Hammond, B. Eoff, B. Paulson, A. Wolin, K. Dahmen, J. Johnston, and P. Rajan. Free-sketch recognition: putting the chi in sketching. In *CHI '08: CHI '08 extended abstracts on Human factors in computing systems*, pages 3027–3032, New York, NY, USA, 2008. ACM.
5. L. B. Kara and T. F. Stahovich. Hierarchical parsing and recognition of hand-drawn diagrams. In *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 13–22, New York, NY, USA, 2004. ACM.
6. M. Oltmans. *Envisioning Sketch Recognition: A Local Feature Based Approach to Recognizing Informal Sketches*. PhD thesis, Massachusetts Institute of Technology, 2007.
7. B. Paulson and T. Hammond. Paleosketch: accurate primitive sketch recognition and beautification. In *IUI '08: Proceedings of the 13th international conference on Intelligent user interfaces*, pages 1–10, New York, NY, USA, 2008. ACM.
8. D. Rubine. Specifying gestures by example. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 329–337, New York, NY, USA, 1991. ACM.
9. T. M. Sezgin, T. Stahovich, and R. Davis. Sketch based interfaces: early processing for sketch understanding. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, page 22, New York, NY, USA, 2006. ACM.
10. D. Sharon and M. van de Panne. Constellation models for sketch recognition. In *SBIM '06: Proceedings of the 3rd Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 2006.
11. J. O. Wobbrock, A. D. Wilson, and Y. Li. 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*, pages 159–168, New York, NY, USA, 2007. ACM.
12. A. Wolin, B. Eoff, and T. Hammond. Shortstraw: A simple and effective corner finder for polylines. In *SBIM '08: Proceedings of the 5th Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 2008.
13. B. Yu and S. Cai. 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*, pages 141–146, New York, NY, USA, 2003. ACM.