Using Scribble Gestures to Enhance Editing Behaviors of Sketch Recognition Systems

Wenzhe Li
Sketch Recognition Lab.
Computer Science.
Texas A&M University.
TAMU 3112
College Station, TX 77843
nadalwz1115@gmail.com

Tracy Hammond
Texas A&M University
Sketch Recognition Lab.
Computer Science.
TAMU 3112
College Station, TX 77843
thammond@gmail.com

CHI'12, May 5–10, 2012, Austin, Texas, USA.
ACM 978-1-4503-1016-1/12/05

Abstract
Mechanix is a computer-assisted tutoring system for engineering students. It uses recognition of freehand sketches to provide instant, detailed, and formative feedback as a student progresses through each homework problem. By using recognition algorithms, the system allows students to solve free-body diagrams and truss problems as if they were using a pen and paper. However, the system currently provides little support for students to edit their drawings by using free hand sketches. Specifically, students may wish to delete part or the whole of a line or shape, and the natural response is to scribble that part of shape out. We developed a new method for integrating scribble gestures into a sketch recognition system. The algorithm automatically identifies and distinguishes scribble gestures from regular drawing input using three features. If the stroke is classified as a scribble, then the algorithm further decides which shape or which part of shape to be deleted. Instead of using slower brute-force methods, we use geometric-based linear-time algorithms which efficiently detect a scribble gesture and remove the intended shapes in real-time.

Author Keywords
Sketch recognition; pen-input computing; gesture recognition; editing

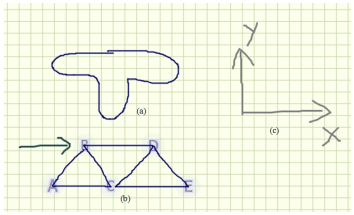

Figure 1: Set of major shapes that are recognized by the Mechanix System. (a) a FBD. (b) a truss with attached force. (c) an axis, which consists of two forces, x and y characters.

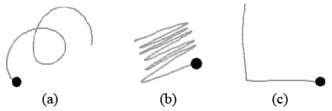

Figure 2: Supported gestures for iCanDraw System. (a) Rollback for undo. (b) Scribble for erase (c) Right angle to mark reference line

ACM Classification Keywords
H.5.2 [User Interfaces]: Interaction styles

Introduction
Mechanix is a sketch based system, which uses recognition of freehand sketches to provide instant, detailed and formative feedback as the student progresses through each homework problem. The system can recognize student diagrams, and compare them to reference solutions provided by the instructors or TAs. Students draw the diagram naturally, with few constraints on their drawing style. Mechniax provides feedback both on the drawn diagrams and the computed answers by comparing the student's submission to a reference solution provided by the instructors or TAs.

A major part of the system is the recognition engine for different shapes including free body diagram (FBD), truss, force, axis and etc. The set of major shapes that are recognized by the Mechanix are shown in figure 1. FBD and truss diagram are the core part of the sketch. The recognition algorithms for both shapes are described in [6]. For other shapes, Mechanix uses a geometric approach similar to [7], where complex shapes are described as combinations of simpler shapes that abide by geometric constraints.

Despite of the advantages provided by Mechanix, the system currently provides little support for editing drawings by using free hand sketches. The students have to explicitly use 'undo', 'redo', 'delete' or 'clear' button to remove the incorrect drawings.

We developed a novel method for integrating scribble gesture into sketch recognition system. The algorithm first classifies an input stroke as scribble or non-scribble gesture by using three different features. If the stroke is classified as a scribble, then the algorithm further decides which shape or which part of shape to be deleted. Instead of using some brute force methods, we use geometric based algorithms which efficiently detects the scribble gesture and remove the intended shapes in real time.

Related Work
Sketch recognition systems typically fall into one of three categories: gesture recognition [10], vision-based recognition [8], and geometric recognition. For non-truss and non-free body shapes like forces and axis, Mechanix uses a geometric approach similar to LADDER [7], where complex shapes are described as combinations of simpler shapes that abide by geometric constraints. We define primitive shapes as shapes that cannot be represented as a combination of simpler shapes. The PaleoSketch recognizer [9] is a purpose-built recognizer for identifying primitive shapes. Mechanix uses PaleoSketch as a low-level stroke processing step before high-level recognition takes place.

Since one important feature of a sketch-based system is to allow people to constantly modify their drawings, it becomes necessary to provide a natural way to interact with them. But unfortunately, it is a non-trivial problem and will complicate the process of designing and recognizing diagrams.

There are two major problems to keep in mind when integrating editing gestures into the sketch-based systems. 1) How to correctly distinguish between editing gestures and normal drawing strokes? 2) How to design editing gestures so that people are comfortable using them?

One simple solution is to explicitly define separate editing and drawing modes [11]. Saund and Moran's PerSketch system focuses on utilizing the principles of Perceptual

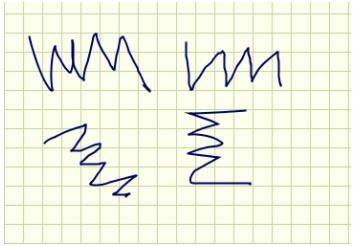

Figure 3: Scribble gestures supported in Mechanix system

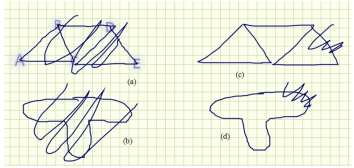

Figure 4: Using a scribble gesture to remove truss or free-body (a) to remove the whole truss (b) to remove the whole free-body (c) to remove the rightmost line only. (d) to remove the right part of free-body only

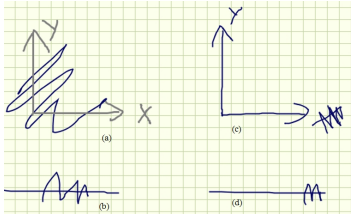

Figure 5: Using a scribble gesture to remove axis and lines (a) to remove the whole axis (b) to remove the x character only (c) to remove the whole line. (d) to remove the right portion of line only

Organization in order to perform complex editing tasks. PerSketch followed a draw\select\modify paradigm and used a modal button press to switch between drawing and editing modes. However, modal switches are often not advantageous, adding additional cognitive load to the user. Saund and Lank would later show that a modeless switch is possible using pen trajectory and context.
In [5], Dixon et al. defines three editing gestures for iCanDraw system. Figure 2 shows how these gestures look like as well as their meanings.

The work by Dahmen is closely related to ours [4]. They recognized scribble gestures using four features that are extracted from a stroke including density, speed, bounding ratio, and number of intersections. They showed that bounding ratio and density can quickly and effectively determine a user's intention. However, the bounding ratio feature is specifically designed for distinguishing between deleting gesture and filling-in gesture, which is not suitable for our context. In addition, the density feature does not work for a system containing over traced strokes. Finally, the intersection algorithm used by Dahmen is a brute-force algorithm that runs in $O(n^2)$. All three of our proposed features are intuitive and computationally efficient.

Defining Problems

First, we define the scribble gesture as in figure 3. The scribble is a common gesture used to remove the drawings or texts.

Next, we define some scenarios to show how we use the scribble gesture to remove the shapes. Figure 4 shows cases when using a scribble gesture should remove either a truss or a free body diagram. In (a) and (b), the scribble gesture covers most parts of the main bodies, with the user intending to remove the whole diagram, whereas in (c) and (d), the sketcher only intends for the gesture to remove part of diagrams. Figure 5 shows similar cases for removing axis and lines. The whole removing process is illustrated in algorithm 1.

Algorithm 1 Overall removing process

Input: an input stroke s, and a $sketch$, which represents the whole strokes/shapes on the screen
$result =$ **classify**(s);
if $result ==$ "scribble" **then**
  try to remove the (part of)shapes.
else
  add s into $sketch$, then recognize it.
end if

Classifying a Stroke as Scribble

The most important step is first classifying an input stroke as a scribble or non-scribble gesture. Here, we propose three intuitive features for classification.

Entropy

We notice that scribble gestures have more curves, angles, or simply motion as compared to non-scribble gestures. From information theory perspective, we can interpret it as scribble gesture contains more uncertainties, this is where our first feature comes from. The idea is also used in [1], where the entropy measurement has been used to distinguish text versus shape. For the entropy calculation, we use the same process that was used in [1].

Input stroke s is defined as $s = (p_1, p_2..., p_n)$, where p_i is the ith point of the stroke s. And each point p_i represents the x-y position in two dimensional space. At the beginning, we select the first three points and calculate the angle among them. Then we move to the next three

Figure 6: High entropy for both scribble gesture and over traced gestures. (a) scribble gestures, entropy = 0.71, LSE = 124.61 (b) over traced gestures entropy = 0.62, LSE = 2.88

points and do the same calculation. The process will continue until we reach the last three points. Finally, we end up with the list of angle values. In order to calculate the entropy for these list of values, we first discretize the values into six different beams. Then Shannon's entropy formula, $H(Y) = -\sum_{y\in\Omega} p(y)\log p(y)$, will be applied.

Least Square Error

The intuition behind using least squared error as a second feature is that over-traced and scribble gestures both have comparatively high entropy values, which is shown in figure 6. It is not hard to notice that over-traced gesture looks much more straight than scribble one. This is where our second feature, least square error (LSE), comes from.

For a stroke $s = \{(x_1, y_1), ...(x_n, y_n)\}$, we try to fit these points to an optimal line $\mathbf{y} = m\mathbf{x} + b$, where m and b can be obtained using the following formulas.

$$m = \frac{n\sum_i x_i y_i - \sum_i x_i \sum_i y_i}{n\sum_i x_i^2 - \sum_i x_i \sum_i y_i}; \quad (1)$$

$$b = \frac{\sum_i y_i - m\sum_i x_i}{n} \quad (2)$$

Then LSE can be calculated as $LSE = \sum_i (mx_i + b - y_i)^2$

Number of Intersections

Since the main purpose of using a scribble gesture is to remove shapes, checking the number of intersections between scribble and intended shapes is another intuitive feature. In [4], they also used this feature, but their brute force approach requires high computational cost.

In our implementation, we fully utilize the efficient geometric algorithms. Instead of modeling each line as list of points, we look at the endpoints only. Using this approximation, we can significantly reduce the time complexity by avoiding comparisons of each pair of points. And the preliminary results also convince us that this relaxed scheme works well in general settings.

Algorithm 2 Locate the shapes to be deleted

Input: scribble stroke $s = (p_0, p_1, ...p_n)$, where p_i is ith point. $sketch$, which consists of set of strokes $s_0, s_1....s_m$ The algorithm takes $O(n \log n + mh)$ time, where h is the number of vertices of the convex hull.
Output: list of line segments $segments$
Initialize Convex hull $CH = \emptyset$
CH = *Graham's scan*$(p_0, p_1, ...p_n)$
for each $stroke\ s_i$ in $sketch$ **do**
  if isIntersect(s_i, CH) == $true$ **then**
    (p_{enter}, p_{leave}) = getIntersectionPoints(s_i, CH)
    $segments$.add(new segment(p_{enter}, p_{leave}));
  end if
end for
return $segments$

Using this assumption, each primitive stroke s_i, which is denoted as "line" in the pseudocode, can be represented by two endpoints (x_{i1}, y_{i1}) and (x_{in}, y_{in}). We first use PaleoSketch [9] to convert input stroke into primitive shapes. PaleoSketch always recognize scribble gesture as "polyline" which consists of set of lines. After that, we compare each line which is part of this polyline with all other strokes in the sketch, to find out how many times they intersect in total. In our algorithm, the subroutine which is used to check whether two lines intersect or not, only takes constant time.

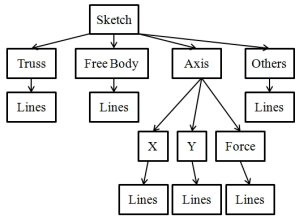

Figure 7: Organization of the whole sketch

Algorithm 3 Remove a line or part of a line

Input: v_1, v_2 are the end points of a line l. p_1, p_2 are the two intersection points between line l and convex hull CH.

$p_{center} = \frac{v_1+v_2}{2}$
if $p_1 < p_{center} < p_2 \;||\; p_2 < p_{center} < p_1$ **then**
  remove the whole line
else
  $p = argmin_{p \in \{p_1, p_2\}}$ dist(p, p_{center})
  if v_1 and p are on the same side **then**
    remove segment $\overline{pv_1}$
  else
    remove segment $\overline{pv_2}$
  end if
end if

Removing Shapes

Once the input stroke is recognized as "scribble", then we try to locate the shape and delete it. For the locating problem, we bring the knowledge from computational geometry field. We first give brief introduction to convex hull and line clipping for completeness purpose.

Convex hull From the definition in [2], the **convex hull** of a set Q of points is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior. We denote the convex hull of Q by CH(Q). There are many algorithms exist to efficiently find the convex hull. In this work, we use Graham's scan, which takes $O(nlgn)$ time, where n is the number of points.

Line clipping The problem is defined as: Given a line l, and convex polygon g, we try to find the intersection points between l and g if they do intersects. Using this algorithm we can determine which portion of line is inside polygon. There are also many algorithms work well for finding such segments. Some of these algorithms can be found in [3]

Locate the shapes to be deleted
Given an input stroke, $s = (p_1, ...p_n)$, we first find the convex hull, CH, of this stroke. Then we find all the segments that reside inside this convex hull. As mentioned before, all the strokes are approximated as "line" in the pseudocode. Thus, the problem becomes that given list of lines and a convex hull, find out all of the line segments that are located inside this convex hull. Algorithm 2 shows the details for implementation.

Remove the selected shapes
Let's first look at the Mechanix system to see how the sketch is actually organized (Figure 7). For each type of shape, we use a slightly different approach.

Truss and Free body In order to decide whether the scribble gesture intends to remove the whole shape or parts of shape only, we can simply comparing the ratio R with predefined threshold T, such that if R is larger than T, we simply remove the whole shape, otherwise only remove the lines that intersect with convex hull. The ratio R is defined as:

$$R = \frac{\#\text{of intersected lines}}{\#\text{of total lines within the body}} \qquad (3)$$

Axis If both of forces are removed, we automatically remove the whole axis. Otherwise, we only remove the part of shape that intersects with convex hull.

Force, X, Y For X and Y, whenever there is a line intersects with convex hull, we will remove the X and Y as a whole. When regarding the force, we only remove the

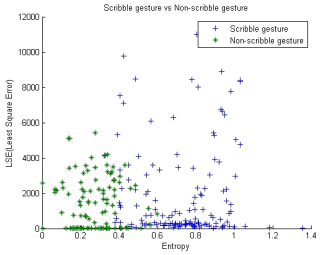

Figure 8: Using entropy and LSE features to distinguish between scribble gesture and non-scribble gesture

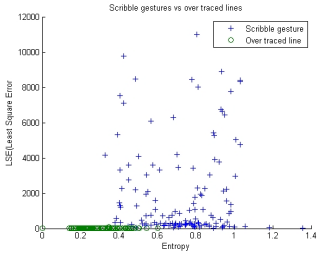

Figure 9: Using entropy and LSE features to distinguish between scribble gesture and over traced lines

lines that intersect with convex hull while leaving other lines remain in the sketch.

Line Here we consider the case that line is represented as individual shape instead of becoming any part of other shapes. Based on our observations, we found that people sometimes prefer to remove only part of a line. We denote v_1, v_2 as the two end points of the line, l. We first find the two points p_1, p_2, which are the intersection points between line l and convex hull. Then we check whether the line segment $\overline{p_1 p_2}$ covers the middle point of the line or not, l. If does, we assume people want to remove the whole line. Otherwise, we only remove the the segment from the point p that satisfies the following condition, to one of the end points of l which is on the same side with point p ('dist' means distance).

$$p = argmin_{p \in \{p_1, p_2\}} \text{dist}(p, p_{center})$$

Result and Discussion

We collected total 140 scribble gestures and 140 non-scribble gestures, we also collected 140 pure over traced lines. Figure 8, 9 show how the first two features can aid to distinguish between scribble gesture and non-scribble gesture (including over traced lines). We can determine the decision boundary between these two classes. As continuing work, we plan to conduct comprehensive user studies to demonstrate its practical usefulness.

Acknowledgements

This material is based upon work supported by Google and the National Science Foundation under grand numbers 0942400, 0935219, and 1129525.

References

[1] A. Bhat and T. Hammond. Using entropy to distinguish shape versus text in hand-drawn diagrams. In *Proc. IJCAI*. AAAI Press, 2009.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press, Cambridge, MA, USA, 2002.

[3] M. Cyrus and J. Beck. Generalized two- and three-dimensional clipping. *Computers and Graphics*, 3(1), 1978.

[4] K. Dahmen and T. Hammond. Distinguishing between sketched scribble look alikes. In *Proc. AAAI Student Abstract*. AAAI Press, 2008.

[5] D. Dixon, M. Prasad, and T. Hammond. icandraw: using sketch recognition and corrective feedback to assist a user in drawing human faces. In *Proc. CHI*, pages 897–906. ACM Press, 2010.

[6] M. Field, S. Valentine, J. Linsey, and T. Hammond. Sketch recognition algorithms for comparing complex and unpredictable shapes. In *Proc. IJCAI*. AAAI Press, 2011.

[7] T. Hammond and R. Davis. Ladder, a sketching language for user interface developers. *Computer and Graphics*, 29(4):518–532, 2005.

[8] L. B. Kara and T. F. Stahovich. An image-based, trainable symbol recognizer for hand-drawn sketches. *Computers and Graphics*, 29(4):501–517, 2005.

[9] B. Paulson and T. Hammond. Paleosketch: accurate primitive sketch recognition and beautification. In *Proc. IUI*, pages 1–10. ACM Press, 2008.

[10] D. Rubine. Specifying gestures by example. In *Proc. SIGGRAPH*, pages 329–337, 1991.

[11] E. Saund and T. P. Moran. Perceptual organization in an interactive sketch editing application. In *Proc. ICCV*, pages 597–607. IEEE, 1995.