

RingEdit: A Control Point Based Editing Approach in Sketch Recognition Systems

Yuxiang Zhu, Joshua Johnston, and Tracy Hammond

Department of Computer Science and Engineering

Texas A&M University

College Station, TX 77843-3112

giftzyx@gmail.com, {jbjohns,hammond}@cse.tamu.edu

ABSTRACT

Editing a sketch should be one of the essential features provided by sketch recognition systems to allow people to modify what they have drawn, without having to delete and redraw shapes. This paper introduces a control point based editing approach we call RingEdit. RingEdit differs from other sketch editors in that the user actually draws their own control points on the sketch, rather than relying on control points generated by the recognition system. It provides modes that allow moving, rotating, scaling, and bending on both the shape level and stroke level. RingEdit shows great editing capabilities.

Author Keywords

Sketch recognition, sketch editing, control points.

ACM Classification Keywords

H5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

INTRODUCTION

Perhaps the first pen-based interactive computer system was Sketchpad, created by Ivan Sutherland [13]. Since then, during the last few decades, numerous sketch systems have been developed [1, 4, 5, 6, 7, 9, 12]. With the advance of technology, computers have become faster and more powerful than the one Sketchpad ran on, and pen-based input devices such as Tablet PCs, PDAs, and SMART boards are cheap enough to be affordable by more and more people. Thus, the field of sketch recognition and is becoming increasingly popular.

Sketch recognition is essentially different from traditional paper-and-pencil sketching or Computer-Aided Design (CAD) in many ways [3, 4, 6]. On the one hand, people should be able to interact with sketch recognition systems naturally, draw freely as they would usually draw with paper and pencil, and have what they sketch be recognized by the computer without pressing buttons to explicitly invoke recognition. In other words, sketch recognition systems should place no more cognitive burden on a user than a simple piece of paper and pencil would. On the other

hand, sketch recognition is superior to paper and pencil sketching in that strokes and recognized shapes are digitized. This means that modifications of the sketch are supported by a computer, without having to throw away the whole picture and redraw it when some parts of the sketch are to be changed. It is important to support editing so that users may modify their sketches during or after their initial designs as their intentions change. Additionally, sketchers will wish to remove unintentional strokes or rectify ambiguous strokes that could prevent the sketch from being correctly recognized. Editing a stroke can help to avoid the redundant work and wasted effort of redrawing an incorrect sketch or portion of a sketch.

This paper introduces a control point based editing system for sketch recognition called RingEdit. RingEdit is used seamlessly while sketching in the LADDER system [5]. No modality is needed to delineate sketching and editing modes. It supports powerful editing behaviors like moving, scaling, and rotating both entire shapes and individual strokes.

PREVIOUS WORK

Saund et al. constructed an image editing program called ScanScribe [10]. Their system emphasizes easy selection and manipulation of off-line sketches (i.e. material found in casual documents such as sketches, handwritten notes, etc). ScanScribe introduces interface techniques for selecting image objects with a pointing device without resorting to a tool palette of different mode. Image material can be selected by rectangle, lasso, polygon, and clicking on object. Clicking on an object is available any time the image objects have become established within the program. Whether a rectangle or lasso is used is based on what the selection path encloses. Polygon selection is activated by mouse double click. However, this system is only based on the objects found in offline sketches, and the editing behaviors do not involve morphing the shapes of the objects.

Alvarado [2] addresses both HCI and sketch recognition by introducing a recognition-based diagram creation tool that robustly recognizes naturally drawn diagrams and automatically imports them into PowerPoint with a multi-domain sketch engine called SketchREAD [1]. This tool provides editing capabilities including move and delete.

Pen-based editing commands are implemented. Users switch between editing and sketch modes by pressing buttons on the window. Online editing is introduced to allow users to sketch while in edit mode. The user holds down the pen in sketch mode to enter online edit mode, then she can select items to edit. When she lifts the pen, she can draw new items. However, the selected items remain highlighted, indicating that she can move or delete them using the same gestures as in edit mode. The problem in this system is that users need to click buttons to switch between the editing and sketching modes, which is not convenient and distracts people’s attention. Additionally, requiring users to press a button to switch modes can prove to be unwieldy on devices which do not have a button readily available such as a slate, projected Wacom surface, large touch screen, or other such devices without keyboards or mice.

LADDER supports predefined editing behaviors [5]. The possible editing actions include wait, select, deselect, color, delete, translate, rotate, scale, resize, show handle, and set cursor. These actions are triggered by user motions including mouse click, double click, hold, hold drag, draw, draw over shape, scribble over shape, and encircle the shape. The editing behaviors give a wide range of possibility in editing, and the fact that these editing behaviors are user defined offers great flexibility. Nevertheless, users may need to define editing behaviors for each shape in the domain, which could prove tedious and may cause inconsistencies in the editing actions for each shape. Also, the editing is limited to some extent. For example, a stroke can be rotated only at its center or end points. RingEdit’s primitive editing actions—moving, rotating, and scaling—are based on those used in LADDER.

RINGEDIT AND CONTROL POINTS

Editing in the RingEdit system occurs around control points. To edit a stroke or shape, a user creates/defines one or more control points. Our system is unique in that instead of using buttons on toolbars or menus to create control points or relying on a predefined set of control points, control points are created by drawing directly on the sketch. This task does not rely on buttons on the stylus, making it hardware-independent. Additionally, creating control points is less distracting and more natural because the user never leaves sketching mode to switch into an editing modality. The transition is seamless.

A control point is defined as a “ring” (i.e. a small circle on a sketched stroke). Using this definition, we must define what it means for a circle to be *small* and *on the stroke*.

By *small* circle, we mean that the bounding box of the circle has both a width and height below a certain threshold. We empirically set this threshold to 40 pixels. This value is chosen so that the circle is large enough to be recognized by the system (circles drawn too small prove difficult to recognize), and, at the same time, the intention of drawing a circle and setting a control point can be distinguished. We

have found that when users want to set control points, they normally draw much smaller circles (less than 40 pixels in width and height) than they would when they actually intend to draw a circle on their sketch.

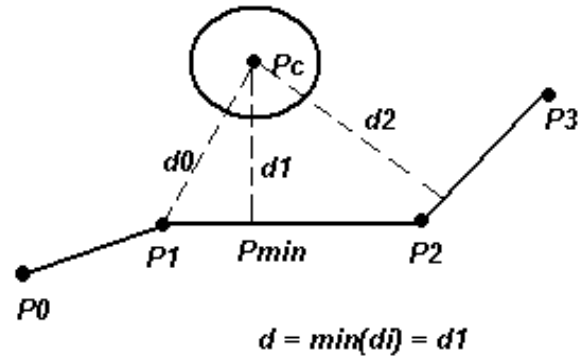


Figure 1. Calculate minimum distance d , if $d \leq 15$, then the circle is “on the stroke”.

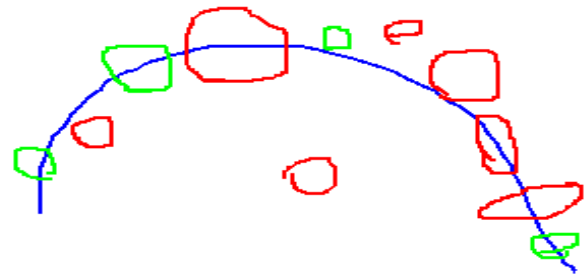


Figure 2. Green circles are “ring”s – control points, red circles are considered as shapes in the sketch.

The second constraint, *on the stroke*, is checked by calculating the distance (d) from the center of the circle (p_c) to the corresponding closest point on the stroke. We designate the closest point as p_{min} . In other words, for a stroke consisting of n points $p_i, 1 \leq i \leq n$,

$$p = p_j \quad \text{where} \quad j = \arg \min_i (p_c, p_i)$$

$$d = \text{dist}(p_c, p_{min})$$

If d falls below the empirically derived threshold of 15 pixels, then the circle is considered to be *on the stroke*. Figure 1 provides an example illustrating this calculation.

If both constraints are met, the circle is both *small* and *on the strokes*, the point p_{min} is set as the control point with respect to the ring that was drawn on the shape to be edited. Figure 2 shows the classification result of a series of circles drawn near or on an arc. The green circles are classified as control points, while the red ones are not.

EDITING MODES

This section introduces the editing modes of RingEdit, including moving, rotating, and scaling (called rubberband

in [5]) on both a shape and stroke basis, as well bending part of a stroke.

Conceptual Model

Many different editing operations can occur with only a single operation that consists of drawing one or more control points and then dragging a point on the stroke. The conceptual model for the interaction is as follows:

- Drawing a control point fixes that point of the stroke to the screen.
- Dragging a point on a stroke modifies the stroke in as simple a way as possible while keeping the control points static.

Moving a Shape

In order to move a shape, the user must draw exactly one control point on any single stroke of the shape (it does not matter which stroke is selected). Then, the user must drag the control point by placing the mouse¹ button *on the control point* (within a range of 15 pixels to the control point, see Figure 3) and move the whole shape by dragging the mouse while the button is depressed.

This maps to the conceptual model as follows:

- A control point is drawn on a shape. When the user drags the control point, because there is no logical point on the shape to remain unmoved, the entire shape translates along with the control point.

Rotating and/or Scaling a Shape

In order to rotate and/or scale a shape, the user must set exactly one control point on the shape to be moved, but instead of dragging the control point, the user must click and drag another point on the stroke (see Figure 3). The control point is set as a fixed point, and when the mouse moves around it, the whole shape rotates and scales accordingly. The scaling factor is computed as the distance from the current mouse point to the control point divided by the distance from the control point to the point where the mouse button was first pressed (ratio of current distance to initial distance). The angle of rotation is the angle from the line between the point where the mouse was first pressed and the control point to the line between the current mouse point and the control point. See Figure 4 for an illustration.

This maps to the conceptual model as follows:

- A control point is drawn on a stroke. The user then attempts to move a different point on the stroke. The stroke rotates and scales to ensure that the control point stays in the same location and the dragged point along the stroke moves to the new location.

¹ All mouse button press or release events mentioned in this paper are equal to pen-down and pen-up using pen-based devices, and vice versa.

Move a Stroke within a Shape

This mode is invoked when a shape is composed of multiple strokes and two control points are set on the shape, one on the stroke to move and the other on any other stroke of the same shape. When one of the control points is dragged, this stroke can be separated from the shape and moved on its own.

This maps to the conceptual model as follows:

- A control point is drawn on one stroke (not to be moved), and another control point is drawn on another stroke (to be moved). The order of drawing these control points is not important.
- If one of the control points is dragged, then that stroke is translated, but the other one remains fixed in its location.

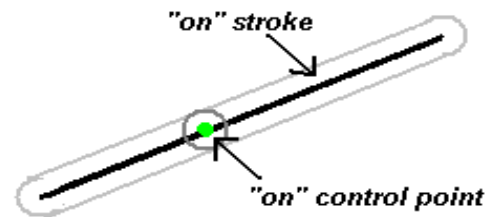


Figure 3. On stroke area and on control point area.

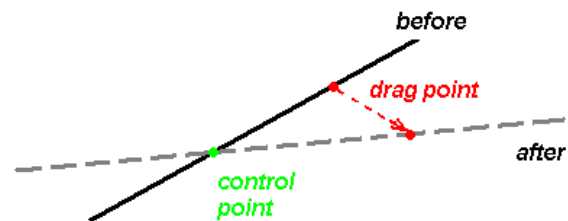


Figure 4. Rotate and scale the stroke.

Rotate and Scale a Stroke within a Shape

In order to rotate and scale a stroke within a shape, the user must set two control points on a multiple-stroke shape, one on the stroke to be edited and the other on any other stroke in the shape. When the stroke to be edited is dragged on a point other than the control point, this stroke can be rotated and scaled independently of the whole shape.

This maps to the conceptual model as follows:

- A control point is drawn on one stroke (not to be moved), and another control point is drawn on another stroke (to be moved). The order of drawing these control points is not important.

- If a point other than a control point is dragged, then the other stroke remains unmoved, as well as the control point of the moving shape. The moving shape is then rotated and scaled on its own as described previously.

Bend a Stroke

To invoke this mode, the user will need to set two control points on the same stroke to anchor the stroke. When the stroke is dragged on the segment between the anchored points, the stroke is bent in the middle (see Figure 5a). When the stroke is dragged on a segment between an anchored point and an endpoint, the stroke will bend at that end (see Figure 5b).

This maps to the conceptual model as follows:

- Two control points are drawn on a single stroke. The two control points must remain fixed.
- The user then moves another point on the stroke. If the moved point is between the two points, the stroke bends in a ‘v’ shape because the control points must remain fixed. (See Figure 5a)
- If the moved point is on one side of each of the control points, then the stroke bends. It does not rotate the whole stroke as above because the second control point must also remain fixed. (See Figure 5b.)

EDITING/SKETCHING SWITCH

Switching between editing and sketching modes is natural with RingEdit. Users do not need to press any buttons to indicate if the next stroke is an editing gesture or a sketching gesture. After control points are set (rings are drawn on the sketch), the next mouse button press or pen-down event in corresponding “hot zones” (see Figure 3) automatically invokes the sketching to editing switch. The next mouse drag gesture that follows the click will be treated as an editing gesture.

The non-modal switch from editing to sketching is automatic. When an editing gesture is completed (signified by the mouse button release or pen-up event) the control points on the shape or stroke that was just edited are removed automatically, allowing the users to continue to sketch or edit other shapes or strokes.

EVALUATION

We performed a preliminary evaluation of RingEdit to assess its usability and functionalities. Users were asked to perform three tasks. After completion of the tasks, we interviewed each user to collect their feedback on the system.

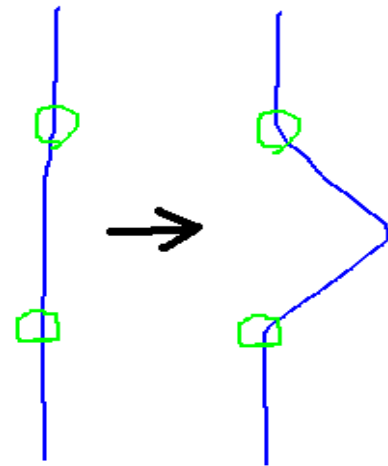


Figure 5a. Bend in the middle.

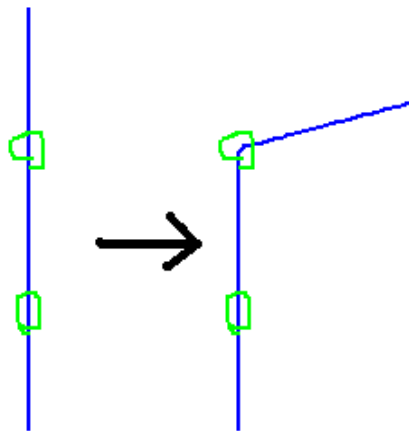


Figure 5b. Bend at one end.

The study was run on a Dell Dimension PC with a 2.4 GHz processor and 1 GiB of RAM. A Wacom Cintiq 21UX Pen-Display Monitor was used as the input/output device. Three graduate students participated in our user study. Two of them were familiar with pen-based sketching, and the other had never used such a system before. All editing modes were demonstrated to them before they started performing the tasks.

Three tasks the users needed to perform were:

- **Task1** Draw four lines of random length and position and edit them to form a rectangle.
- **Task2** Draw a “necklace” – a big circle with small circles of control points on it.
- **Task3** Draw whatever you like, and try whatever editing functions you want.

RESULTS

All users were able to finish all the tasks without much difficulty. They all commented that the editing features were powerful. When asked to score the system on ease of use and satisfaction on a scale of 1 to 10 (10 being the best), the two users with pen-based system experience gave a score of 8 and 6 respectively, while the first-time user gave a score of 8 points. In Task 2, the accuracy of control point recognition was 30/34, 32/40, and 30/33 for the three users. The total accuracy of recognized control points for all three users was 86%. However, when performing Tasks 1 and 3, the accuracy of correctly recognized control points drops. One of the users continuously met problem with setting control points in Task1. But, he quickly mastered it as he tried more times.

DISCUSSION

Overall, users were happy with RingEdit's support to set control points in a natural way through direct sketch. We found that the learning curve for new users was short, suggesting that our system is indeed intuitive, even to a novice user. Shapes and strokes can be edited according to the user's expectations. Editing can be performed both on a shape basis and on a stroke basis. Switching between sketching and editing modes is seamless and the transition between the two is natural. However, there are several problems still to address.

Recognition of drawn rings is sometimes imperfect. This may be due to the fact that small circles are often drawn more sloppily and with less precision. But small circles are difficult to recognize in general. Small circles may contain too few pixels to efficiently differentiate them from a polygon or polyline. We currently use the Sezgin [12] recognizer, but future implementations will use Paulson's PaleoSketch [8] recognizer that has improved circle recognition. Additionally, PaleoSketch is currently being improved to ensure more efficient and accurate recognition of small circles. One user suggested using an "X" or small rectangle to set control points, but these multi-stroke shapes may short circuit with the stroke to be edited. For instance, the first stroke of an "X" may combine with the stroke to be edited to form another shape before drawing the second stroke to set a control point. One solution is to set control points after the mouse is pressed "on" the shapes for a certain amount of time instead of drawing something.

The conceptual model for the editing modes was not explained to the user before the study. And, different editing modes sometimes confuse the users when they first use the system, especially in the stroke-based editing modes where two control points are set. Users tend to forget that if two control points are set on the same stroke, they are meant to be anchored and bent. Instead, the users try to drag one of the control points, expecting the stroke to be moved, rotated, or scaled. To tackle this issue, different colors can be used to show if a control point is anchored or not.

Another problem is that when the strokes are enlarged due to scaling, the number of points in the stroke is not changed

from the original amount of digital ink laid down by the tablet and pen. This makes the points in the stroke become sparse after editing. When the user wants to edit part of the sparsely-sampled stroke, the edited strokes tend to contain less information per unit length which makes the stroke jagged. Strokes should be resampled and points interpolated after scaling to solve this problem.

FUTURE WORK

Since there were only three participants in our preliminary user study, the results may not be representative of a general population. The problems exposed in the user study will need to be mended in future research. More extensive evaluation will need to be conducted to evaluate the system's weaknesses. Whether or not circling is the best way to set control points is yet to be proved, and other gestures can be tried in the future. Also, a choice mediator in the form of a pop-up button ([11]) can be incorporated to ask whether the user intends to draw a circle or a control point when the constraints of a ring are met. It would also be useful to collect more data to determine if our choice of thresholds (for *small* circle and *on the stroke*) is accurate. We would also like to investigate using a dynamic threshold that takes into consideration the context of nearby strokes and shapes.

CONCLUSION

RingEdit allows users to edit without changing modes by drawing control points directly on sketched shapes. This method of creating control points offers a seamless and intuitive switch between sketching and editing modes. RingEdit supports both shape- and stroke-based editing according to the number of control points set on the same shape. It also provides modes to move, rotate, and scale strokes and shapes as well as bend strokes based on the start point of the editing gesture after the control points are set.

Feedback from the user study shows that RingEdit has generally good usability, but problems still exist in control point recognition. Mapping between editing mode actions and users' conceptual models of what should happen when control points/strokes/shapes are manipulated can also be difficult, but we expect that a new description of the mental model will help that for future tests.

ACKNOWLEDGMENTS

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

The authors would like to thank the members of the Sketch Recognition Laboratory for their valuable assistance in this research.

REFERENCES

1. Alvarado, C., Davis, R. SketchREAD: a multi-domain sketch recognition engine, *Proceeding of UIST'04* (2004), 23 - 32.

2. Alvarado, C. Sketch recognition user interfaces: Guidelines for design and development. In *Making Pen-Based Interaction Intelligent and Natural*, AAAI (2004), 8 - 14.
3. Davis, R. Magic Paper: Sketch-Understanding Research, IEEE (2007), 34 - 41.
4. Gross, M. D., Do, E. Y. Ambiguous intentions: a paper-like interface for creative design. *Proceedings of UIST'96*, ACM Press (1996), 183 - 192.
5. Hammond, T., Davis, R. LADDER, a sketching language for user interface developers, *SIGGRAPH'07*, ACM Press (2007), 518 - 532.
6. Herot, C. F. Graphical input through machine recognition of sketches. *SIGGRAPH'76*, ACM Press (1976), 97 - 102.
7. Long, A. C., Landay, J. L. Rowe, and J. Michiels. Visual Similarity of Pen Gestures, In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI 2000)*, ACM (2000), 360 - 367.
8. Paulson, B. and Hammond, T. PaleoSketch: Accurate Primitive Sketch Recognition and Beautification. In *Intelligent User Interfaces (IUI '08)*. New York, USA: ACM Press, 2008.
9. Rubine, D. Specifying gestures by example, In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, ACM (1991), 329 - 337.
10. Saund, E., Fleet, D., Lerner, D., Mahoney, J. Perceptually-supported image editing of text and graphics, *SIGGRAPH'04*, ACM Press (2004), 728 - 728.
11. Saund, E., Lank, E. Stylus input and editing without prior selection of mode. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology (UIST 2003)*, ACM (2003), 213 - 216.
12. Sezgin, T. M., T. Stahovich, and R. Davis. Sketch based interfaces: early processing for sketch understanding. *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, ACM (2006), 22.
13. Sutherland, I. E. Sketchpad a man-machine graphical communication system. In *25 years of DAC: Papers on Twenty-five years of electronic design automation*, ACM Press (1988), 507 - 524.