EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling (2007)
M. van de Panne, E. Saund (Editors)

# A Pen-based Tool for Efficient Labeling of 2D Sketches

Aaron Wolin, Devin Smith and Christine Alvarado

Harvey Mudd College, CA
{awolin,dsmith,alvarado}@cs.hmc.edu

**Abstract**

*High quality labeled data is essential for developing and evaluating sketch recognition algorithms. Unfortunately, labeling freely-drawn sketches is time-consuming and difficult, if not impossible, using current technologies. These difficulties and the resulting lack of labeled data fundamentally limit the development of recognition algorithms. We present an intuitive, direct manipulation pen-based application for labeling sketch data in any two-dimensional domain. Our labeling tool supports the three essential sketch recognition labeling tasks: stroke fragmentation, stroke grouping and label application. Our interface integrates standard and novel interaction techniques to make each task efficient and natural. In a user study, all users felt that labeling data with our tool was quick and efficient.*

Categories and Subject Descriptors (according to ACM CCS): I.5.5 [Pattern Recognition]: Implementation: *Interactive systems*
H.5.2 [Information Interfaces and Presentation]: User Interfaces: *Interaction styles*

## 1. Introduction

Tablet computers have the potential to fundamentally change the way people create and interact with diagrams. Their pencil-and-paper-like interface allows users to sketch directly on the screen, alleviating many of the constraints of the traditional mouse and keyboard interface. However, simply allowing people to draw on the computer is not enough if their diagram is nothing more than an electronic picture. To fully realize the the freedom and power of the Tablet PC, the computer must interpret the user's sketch and then allow the user to interact with it, not simply as a sketch, but as a meaningful system in a particular domain.

In recent years, researchers have improved low-level stroke parsing [Sta04, SD04], isolated symbol recognition [HN04, LKS07] and free-sketch recognition [AD04, GKS05]. Low-level stroke parsing involves breaking a stroke down into primitive components (usually lines and arcs) by identifying the strokes' corners. Isolated symbol recognition involves recognizing a set of strokes known to comprise a single object. Finally, free-sketch recognition involves *stroke grouping*, or partitioning strokes into individual objects, in addition to symbol identification.

Advances in all of these areas require labeled sketch data for training and testing. Unfortunately, few standard datasets exist [OAD04], and many recognition systems require domain and task-specific data.

The lack of standard corpora is due, in part, to the difficulty of data labeling. Although data collection is relatively straightforward, hand-labeling this data is at best tedious and time-consuming and at worst impossible given current technologies. Usually researchers label only isolated shapes by asking users to draw instances of a specific shape and saving each instance in a separate file. This type of labeling suffices for isolated shape recognition but it is not sufficient for free sketch recognition where the goal is to simultaneously group and classify multiple shapes in a large diagram. Simply associating a complete hand-drawn diagram with a list of symbols in the diagram (or even with a domain-specific structural representation of the diagram) is not enough because we need to know which strokes correspond to which objects in the labeled diagram. Automatically creating this association is a difficult recognition problem in itself. To label a freely-drawn sketch, a human must identify and mark corners in each stroke, group strokes into individual objects, and tag each group with a symbol label. This process is not possible without a tool designed to support it. Currently no such tool exists.


Sketch-Based Interfaces and Modeling 2007, Riverside, CA, August 02-03, 2007.
© 2007 ACM 978-1-59593-913-5/07/0008 $5.00

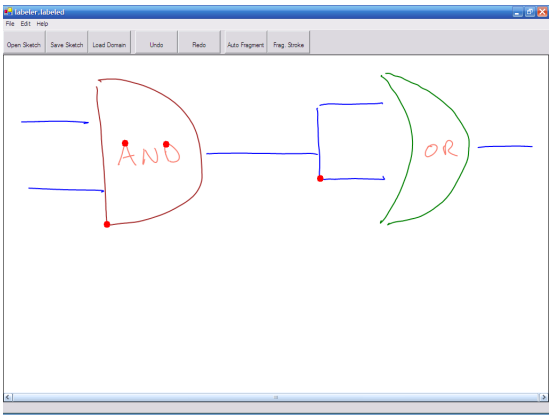



**Figure 1:** *Our labeler application*

We have created an intuitive, pen-based direct manipulation application to support two-dimensional sketch labeling (Figure 1). Using a pen, the user interacts directly with the strokes in the diagram, quickly and easily fragmenting them, grouping them and applying labels. We developed the interface using an iterative design process, and our final user study indicates that the tool is intuitive, fast, and simple for each of the labeling tasks.

This paper makes three central contributions to the field of sketch based interface design. First, it identifies three major sketch recognition and labeling tasks: stroke fragmentation, stroke grouping, and symbol labeling. Second, it presents a tool that provides integrated support for all three labeling tasks in any two-dimensional user-defined domain through:

- an efficient fragmentation interface that combines automatic and manual techniques,
- pen-centric mechanisms for selection and labeling,
- clear visual feedback that helps the user understand the labeling as they work.

Finally, it describes a user study that evaluates the strengths and weaknesses of this tool.

## 2. Data Labeling for Sketch Recognition

Data labeling involves annotating a user's freely-drawn sketch with semantic information needed to guide and evaluate recognition algorithms. We begin by examining briefly the tasks involved in free-sketch recognition: fragmentation, stroke grouping and symbol identification.

Fragmentation, or the task of breaking strokes at their corners into *substrokes*, is the first step in many free-sketch recognition algorithms for two reasons. First, users sometimes draw more that one object with a single stroke, for example, the wire and the bubble in the NOT gate in Figure 2. To recognize these objects correctly, a recognition system must detect the boundary between the wire and the body

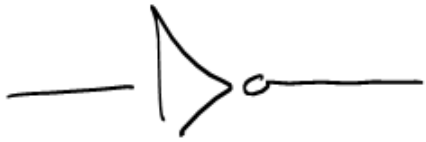


**Figure 2:** *A NOT bubble and a wire can be drawn in a single stroke*

of the not gate, i.e., detect the corner in the stroke. Second, symbol recognizers often work better with a consistent representation for a symbol. Because users' drawing styles can vary, it is useful to break the strokes apart into their primitive components (lines and arcs) to create a consistent representation.

Next, a free-sketch recognition algorithm must group strokes (and substrokes) into individual objects, and identify those objects as symbols in a given domain. In an automatic recognition system, these tasks are inherently intertwined. Simple temporal and spatial grouping techniques are not robust because symbols may overlap and because the user may draw two symbols in parallel. On the other hand, naively matching all templates to all parts of the user's sketch is computationally intractable.

Labeled data is essential to the development and quantitative evaluation of each of the above tasks. Sketches drawn within Tablet PC programs such as Windows® Journal or Microsoft® One Note are, in their most basic form, a series of time-ordered *strokes*. Each stroke is a series of time-ordered points containing position and temporal information sampled from when the user put the pen on the screen until the user lifted the pen. Our goal is to tag the strokes in a sketch with information corresponding to the three recognition subtasks defined above: where the corners of the strokes are, how strokes are grouped into individual objects, and what symbol each group of strokes represents. Figure 1 illustrates the results of all of these tasks. The red circles show stroke corners (some are still missing), and colors indicate the different labels applied to each symbol (e.g., green is an OR gate, blue is a wire, salmon is text, etc.). By default the interface does not indicate stroke groups explicitly, but clicking on any stroke highlights all of the strokes its stroke group (see Section 3.4).

Existing tools provide only limited support for these labeling tasks, and no tool provides support for all three. Automatic fragmentation techniques exist (e.g., [SSD01, Sta04]) but they are not perfect. Indeed, one of the reasons we need to manually label corners is to improve automatic fragmentation algorithms. Similarly, programs such as Windows® Journal and ScanScribe [SFLM03] perform automatic grouping. This grouping is domain-independent and necessarily produces errors. Although these automatic

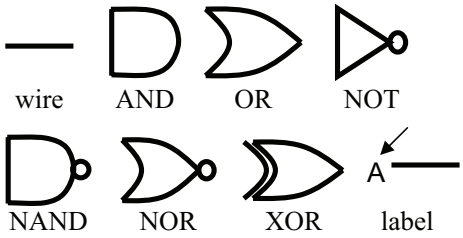



**Figure 3:** *The symbols in the digital circuit domain*

techniques can aid in the labeling process, they are not sufficient without the ability to manually correct and add more groups. Finally, a number of exiting tools support the collection and labeling of isolated symbols (e.g., [OAD04,HN04]), but these tools assume that the grouping is already done.

## 3. Interaction

Using an iterative design methodology, we designed a pen-based tool to support all three of the above tasks. We chose to design a pen-based interface because we wanted to support efficient labeling specifically on a tablet computer. Sketch recognition researchers often work only on a tablet computer, without an external keyboard, mouse and monitor. Compared to a track pad or eraser head mouse, the pen provides a more tangible, precise and fluid way of manipulating stroke data.

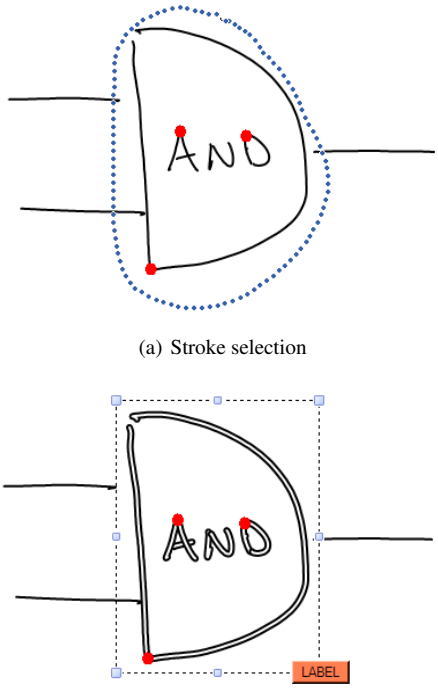
Throughout the design process we worked with three sketch recognition researchers (former members of our research group) who are familiar with the labeling tasks. These early, informal tests helped guide our design. We present a more formal user study in the next section, but throughout this section we refer to the results of this early testing to justify our design decisions where appropriate.

Our tool supports labeling in any two-dimensional domain, but here we focus on digital circuit diagrams for simplicity. Figure 3 illustrates the symbols in this domain. We examine the challenges involved in fragmenting, grouping and labeling the strokes in the sketch in Figure 1, and we describe how our labeler helps mitigate these challenges.

### 3.1. Grouping Strokes

Although recognition systems typically fragment strokes before grouping them, we begin with a discussion of how to group strokes using our tool. Users group and label strokes by selecting them and then applying a label to their selection. We implemented an interface that combines circle and tap motions to make the stroke section process efficient and powerful.

Early user tests revealed that people naturally tapped on




(a) Stroke selection

(b) Unwanted strokes selected

**Figure 4:** *The lasso selection technique necessarily includes unwanted strokes. (This sketch is not fully fragmented.)*

single strokes to select them and circled strokes to select a group. However, a simple lasso selection is not sufficient to create stroke groups. For example, to label the AND gate in Figure 4(a) the user must group the two stroke fragments that make the body of the gate. But when she circles the gate she also selects the text inside the gate (Figure 4(b)).

To address this problem, our selection interface combines lasso selection and stroke tapping. Users can select a group of stroke using a lasso or select a single stroke by tapping on it. Then, tapping on unselected strokes adds them to the current selection while tapping on selected strokes removes them from the selection. Tapping on the background clears the current selection.

Other pen-based applications such as Windows® Journal provide a similar selection mechanism with one important difference: these interfaces require that the user hold down the control key while tapping to add or remove strokes from the selection. People typically do not have access to the keyboard while using a pen-based interface. Our interface provides the same selection power using only the pen.

### 3.2. Labeling Symbols

The next task in the labeling process is to tag a group of selected strokes with one or more labels. Once the user selects

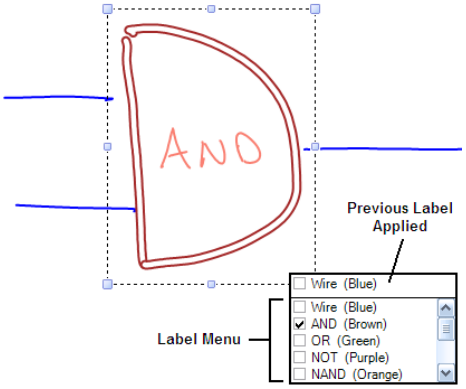



**Figure 5:** *Drop-down label menu. The most recently applied label is displayed in the top box.*

a stroke or group of strokes, a button appears at lower right corner of the selection box. (Figure 4(b)). If the user wishes to label a stroke or group of strokes, she moves her hand a short distance to tap this button and a menu of pre-defined labels appears (Figure 5). The button is small and unobtrusive so users can ignore it and continue to modify the selection.

During a single labeling session, people usually label multiple sketches from a single domain, and each domain tends to have small fixed number of symbols. Instead of forcing the user to type in the name of every symbol, our tool allows the user to load a domain file that specifies the symbol names in the domain and the colors in which to display those symbols when labeled.

The label menu also supports a number of other common labeling behaviors. Our users tended to label all of the symbols of a single type (e.g., all of the AND gates) and then move on to labeling the symbols of another type, etc. We support this process by displaying the most recently applied label at the top of the label menu. Second, sketch recognition researchers sometimes wish to apply multiple labels to a stroke or group of strokes (e.g., "gate" and "AND gate"). Users can select multiple labels by clicking on each label in turn, or they can unlabel a stroke or group of strokes in the same fashion. The menu displays check boxes next to labels to indicate what labels are associated with a stroke or group of strokes.

### 3.3. Fragmentation

Manual fragmentation, though reliable, can be tedious. To make the process more efficient, we integrate automatic and manual fragmentation in our labeler. Clicking on the "Auto Fragment" button at the top of the interface automatically fragments all of the strokes in the sketch using the algorithm described in [SSD01] that splits strokes at points of maximum curvature and minimum speed. The system displays fragment points as red dots on the strokes (Figure 6).

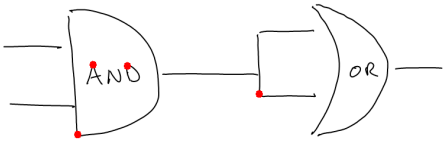



**Figure 6:** *Our digital circuit sketch from Figure 1 with strokes automatically fragmented. Corners are highlighted with red dots. Some corners are missed.*

Of course, automated processes make mistakes, so we allow users to correct false positives and false negatives with a manual fragmentation interface (Figure 7). To manually fragment this stroke the user selects it, clicks the "Hand Fragment" button, and a new window appears displaying the selected stroke. The user then adds fragment points by "slicing" across the stroke with their pen. New fragment points appear where user's pen intersects the stroke. Users can also remove fragment points by clicking the "Clear" button within the manual fragmentation window. This button clears all fragment points in a given stroke. A stroke typically has only one or two true fragment points, so it is easier for the user to clear all the points and then add the correct points back in than it is to select the incorrect points to remove them.

### 3.4. Visualizing the Data

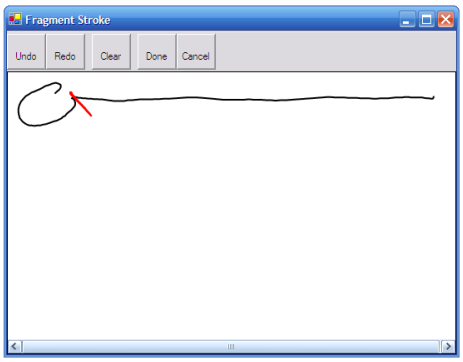
We use colors to communicate labels in our application, for example, red for OR gates and blue for wires (Figure 5). Users can specify their own color scheme in the domain file.

However, as described above, users apply labels to groups of strokes, not just individual strokes, so our tool must communicate not only labels but also stroke groups. Although in some cases stroke grouping is obvious (e.g., two AND gates generally appear far apart in space), in other cases it is not. For example, wire (a) in Figure 8(a) might belong any number of possible groups. To communicate groups within our labeler, we visually thicken all of the strokes involved in a stroke's label when the user selects that stroke 8(b).

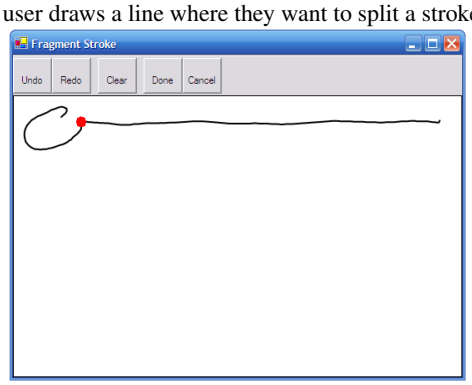
## 4. User Study

We designed our labeler through an iterative process where we received user feedback during each phase. Overall we wanted our labeler to:

- Provide quick and efficient pen-based sketch labeling
- Have a natural, intuitive interface
- Give helpful feedback to convey current labels

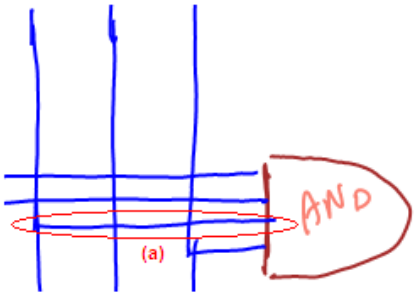



A user draws a line where they want to split a stroke

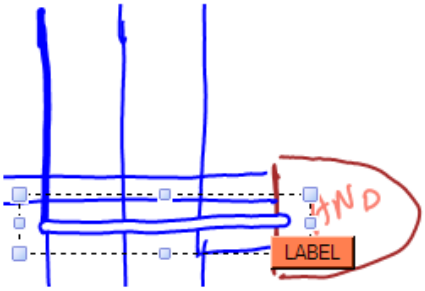



The resulting fragmentation points are displayed

**Figure 7:** *The pop up window supporting the manual fragmentation process*




(a) Stroke (a) could belong to several wire groups




(b) Strokes within a group are thickened when the user selects any of the strokes in the group

**Figure 8:** *Stroke group visual feedback*

It is difficult to compare our final system to others because there are not many other data labeling systems, and no available sketch labeling systems that support the labeling tasks described here. We therefore relied solely on feedback from a final user study to determine how well our labeler meets our criteria.

### 4.1. Participants and Tasks

Our tool is designed to be used by sketch recognition researchers, so for our final user study we recruited members of the Harvey Mudd community with at least basic experience developing sketch recognition programs, but not involved in the design of this tool. Four subjects participated in our final evaluation (three men and one woman, three students and one professor). All subjects had previous experience using a Tablet PC.

Users in our study completed several typical sketch labeling tasks. They were instructed to:

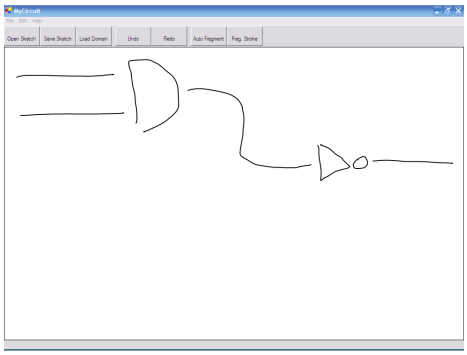
1. Automatically fragment an unlabeled sketch (Figure 9(a))
2. In the same sketch, hand-fragment strokes missed by the automatic fragmenting process (Figure 9(a))
3. Completely label a simple sketch consisting of AND gates, NOT gates and wires (Figure 9(a))
4. Open an intentionally mislabeled sketch and correct any errors (Figure 9(b))

We observed users as they worked and encouraged them to think aloud. After completing all four tasks, users responded to a short survey (Table 1) and provided qualitative feedback about the strengths and weaknesses of the system.

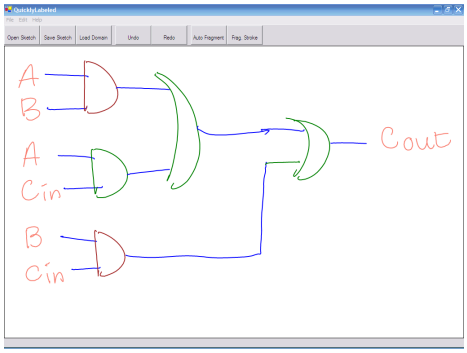
### 4.2. Results

Overall users responded very positively to the labeling tool, with an average response of 6.5 out of 7 to the statement *"Overall, the user interface was excellent."* All users felt our labeler met the initial design criteria we expected to achieve. Table 1 summarizes user survey responses.

All users agreed that labeling data was a quick and efficient process. In their qualitative responses, users indicated that the main interface features that support efficient labeling are the placement of the label button next to the selection and the automatic fragmenter. They commented that the "placement of the label button ... was very nice" and the button "made it quicker to label the [scenario's] diagram." Users



(a) A simple sketch




(b) A sketch with labeling errors (the middle AND gate and one of the wires on the right)

**Figure 9:** *User study sketches taken from a student's digital design notes*

had no trouble invoking the automatic fragmenter or understanding where the fragmenter found corners (points highlighted with red dots).

Users also found the tool easy to learn. No user took longer than 30 minutes to complete the scenarios. Users also mentioned that the interface as a whole "seems very intuitive." Color feedback effectively conveyed which labels users had applied to the strokes. Users responded with an average of 6.75 to the statement *"Types of labels (AND, OR, etc.) were well conveyed"*.

We observed that users relied on both selection techniques (lasso and tapping) but rarely combined the two. Users tended to use lasso selection to select large single strokes or groups of strokes and tap selection to select smaller strokes. Users rarely made a selection using both lassos and taps. In most cases where an initial lasso selection was incorrect, users just tried re-lassoing the group they desired instead of removing or adding strokes to the current selection through taps. Selections involving both lassos and taps mainly occurred when users were correcting intentionally mislabeled data in Figure 9(b). This scenario required more complicated

| Statement | mean | std dev |
|---|---|---|
| This tool was easy to learn to use | 6.5 | 0.58 |
| Labeling data was a quick and efficient process | 6.25 | 0.5 |
| Correcting any errors was simple | 6 | 0.82 |
| Hand-fragmenting strokes was intuitive | 5 | 2.83 |
| The application provided helpful feedback when I performed an action | 5.75 | 0.5 |
| Types of labels (AND, OR, etc.) were well conveyed | 6.75 | 0.5 |
| Stroke groupings were well conveyed | 5.75 | 1.5 |
| Button placement was unobtrusive | 7 | 0 |
| Overall, the user interface was excellent | 6.5 | 0.58 |

**Table 1:** *Summary of user survey responses (1=strongly disagree, 7=strongly agree)*

selections, and probably made using both selection techniques more desirable.

Our user study also revealed some possible areas for improvement. Although no users had trouble actually fragmenting strokes by hand, one user felt that having to open the stroke in a separate window was cumbersome. This user strongly disagreed with the statement *"Hand-fragmenting strokes was intuitive"* because she kept trying to use the "Fragment Stroke" button as if she were switching between selection and fragmentation modes and kept being confused why nothing would happen. She did not notice the tooltip for the button that stated a stroke must be selected before manual fragmentation. A simple solution to this problem is to present a helpful message to the user if she presses the fragment button before she selects a stroke, but this solution does not address the issue that users may prefer a modal interface to a separate window for stroke fragmentation. In future work we plan to compare our current interface to one that uses a separate mode for fragmenting strokes in the main window.

We can also improve our interface by better supporting multiple labels. Multiple labels (e.g., labeling a stroke as a line and as part of an AND gate) are useful in sketch recognition, but it is not yet clear how best to indicate multiple labels through our interface. Our tool allows users to apply more than one label to each stroke, but it colors the stroke according to only the most recently applied label. One participant in our study suggested that we "possibly make the [coloring] stripy" to indicate more than one label.

Another participant disliked having the option to apply multiple labels because it made correcting labeling errors cumbersome: instead of simply selecting the correct label for

a stroke, he had to first unselect the incorrect label, reselect the stroke or group of strokes and then select the correct label. Most of the time users were able to easily select a group of strokes in a single lasso, but a more complicated selection sometimes took a few tries as users were getting used to our lasso and tap methods.

One possible improvement to our system would be the incorporation of a "Previous Selection" button in the toolbar that would reselect the user's previous selection. This alleviates the frustration that users encountered in having to select a group of strokes multiple times, and it safeguards users from accidentally clearing their selection by misclicking.

## 5. Implementation

A central piece of the labeling process is a data format that supports these labeling tasks. We use an XML format developed at MIT that supports fluid, hierarchical labels (`http://rationale.csail.mit.edu/ETCHASketches/format/`). In this format, sketches consist of a series of strokes (points from when the user put the pen down until the user lifted the pen). To support fragmenting and labeling of portions of strokes, each stroke is divided into one more more substrokes. Each substroke holds a time-ordered series of points that contain position $(x,y)$ and, optionally, time information.

This format supports grouping and labeling by allowing for the creation of "shapes", which are simply named groups of strokes, substrokes, or other shapes. Shapes may also contain other meta data such as color, bounding box, creation time, etc.

Sketches in any open format can be converted into the XML format supported by our labeler. To facilitate simple data collection and labeling, we also provide a conversion tool that will convert Windows®Journal files to this XML representation.

We programmed our labeler in C# 1.1, using Microsoft® Visual Studio® .NET 2003. It currently runs on Tablet PCs running Microsoft® Windows® XP Tablet PC Edition 2005. It is available from `http://www.cs.hmc.edu/~alvarado/research/download.html`.

So far, we have used this labeling tool to label hundreds of sketches collected from students in Harvey Mudd College's digital design class. Our labeled data is also available at the above URL.

## 6. Related Work

Labeling sketches is similar to other data labeling tasks. Russell *et al.* present a system for labeling image regions for image recognition [RTMF05]. Their system, LabelMe, tries to solve some of the same problems we do: the software is designed to label portions of an image and they want to clearly indicate any labeled regions. LabelMe works by having users to click to outline the edge of the region in question and then allows the user to type in their own label for the region. Although it solves a similar problem, LabelMe's focus is simply to provide *an* online interface for this type of labeling; the authors do not focus on designing an efficient or natural interface for this task. Their mouse-based interface for creating polygons is somewhat tedious when the region to label is complex with many corners. Applying the pen-based techniques we present here to their task could provide a more natural interface for labeling.

Diakopoulos and Essa present a pen-based system for annotating video [DE06]. Although grouping and tagging video frames over time is somewhat different from grouping and tagging two-dimensional pieces of a single image, we use similar pen-based metaphors where appropriate. For example, both their interface and our interface use the metaphor of the pen as a knife to split pieces of the video or sketch.

Saund *et al.* explore a number of selection and grouping interaction techniques that would make sense to integrate into our tool [SFLM03]. Although users found our manual click selection interaction technique intuitive and easy to use, automatic grouping (like automatic fragmenting) could make the labeling process more efficient still. Furthermore, Saund and Lank's work on selection modality and sloppy selection could further improve our interface by providing a DWIM ("do what I mean"), modeless interface for selection and fragmentation [SL03, LS05].

## 7. Future Work

Automatic stroke fragmentation is one of the strongest aspects of our labeling tool. Following on this success, we would like to incorporate automated techniques for other parts of the labeling process, such as stroke grouping or even, in some cases, symbol labeling. However, these tasks are considerably more difficult than fragmentation and are necessarily prone to errors. We will need provide a powerful error correction interface and to determine whether users prefer correcting system errors over manual grouping and labeling.

We would also like to re-examine our fragmentation interface. Currently, the fragmentation interface opens a seperate window, in part to support stroke resizing. However, some users expressed that they would have preferred an interface in which they toggled between fragmentation and selection modes in the main window. We also would like to allow users to remove fragmentation points with the eraser end fo the digitizing pen, mimicking the process of removing unwanted content with a pencil on paper.

Currently our tool supports basic labeling, but eventually we would like a suite of tools that support all aspects of sketch recognition development. One tool in this suite

would include an interface for comparing and testing different stroke grouping and recognition algorithms. This tool would allow users to plug in different modules for different parts of the recognition process, and run comparative tests on the same dataset and report recognition statistics. Such a tool will help standardize the evaluation of sketch recognition algorithms, a process that is currently somewhat ad hoc.

Finally, we would like to provide a more flexible interface for specifying domain information. Currently users must load text files containing domain information. Although creating these text files is not hard, including the ability to create and modify domain information inside the labeler itself would eliminate the need to switch between our labeler and a text editor whenever a user wants to make an adjustment to the domain she is working with.

## 8. Conclusion

We have presented a pen-based tool designed to support the critical yet often tedious and difficult task of labeling two-dimensional sketch data. Labeled data is critical to the development of sketch recognition algorithms, but currently few labeled datasets exist. Our tool provides a critical step in the creation of these datasets. It allows sketch recognition researchers to quickly and easily label sketch data for their own use, as well as for the use of the community.

## 9. Acknowledgments

We would like to thank our digital engineering students for the sketch data they provided and our anonymous users for useful feedback about our labeling interface. We also thank Jason Fennell and Max Pflueger for help developing our basic software infrastructure. This work is supported in part by an NSF CAREER award (IIS-0546809) and by the Baker Foundation.

## References

[AD04] ALVARADO C., DAVIS R.: Sketchread: A multi-domain sketch recognition engine. In *Proc. UIST* (2004).

[DE06] DIAKOPOULOS N., ESSA I.: Videotater: an approach for pen-based digital video segmentation and tagging. In *UIST '06: Proceedings of the 19th annual ACM symposium on User interface software and technology* (New York, NY, USA, 2006), ACM Press, pp. 221–224.

[GKS05] GENNARI L., KARA L. B., STAHOVICH T. F.: Combining geometry and domain knowledge to interpret hand-drawn diagrams. *Computers and Graphics: Special Issue on Pen-Based User Interfaces* (2005).

[HN04] HSE H., NEWTON A. R.: Sketched symbol recognition using zernike moments. In *Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04)* (2004).

[LKS07] LEE W., KARA L. B., STAHOVICH T. F.: An efficient graph-based symbol recognizer. *Computers and Graphics Special Issue on Sketch-Based Interfaces and Modeling* (2007).

[LS05] LANK E., SAUND E.: Sloppy selection: Providing an accurate interpretation of imprecise stylus selection gestures. *Computers and Graphics 29*, 4 (August 2005), 490–500.

[OAD04] OLTMANS M., ALVARADO C., DAVIS R.: Etcha sketches: Lessons learned from collecting sketch data. In *Making Pen-Based Interaction Intelligent and Natural* (Menlo Park, California, October 21-24 2004), AAAI Fall Symposium, pp. 134–140.

[RTMF05] RUSSELL B. C., TORRALBA A., MURPHY K. P., FREEMAN W. T.: Labelme: a database and web-based tool for image annotation. *MIT AI Lab Memo AIM-2005-025* (2005).

[SD04] SEZGIN T. M., DAVIS R.: Scale-space based feature point detection for digital ink. In *Making Pen-Based Interaction Intelligent and Natural, AAAI Spring Symposium* (2004).

[SFLM03] SAUND E., FLEET D., LARNER D., MAHONEY J.: Perceptually-supported image editing of text and graphics. In *UIST '03: Proceedings of the 16th annual ACM symposium on User interface software and technology* (New York, NY, USA, 2003), ACM Press, pp. 183–192.

[SL03] SAUND E., LANK E.: Stylus input and editing without prior selection of mode. In *UIST '03: Proceedings of the 16th annual ACM symposium on User interface software and technology* (New York, NY, USA, 2003), ACM Press, pp. 213–216.

[SSD01] SEZGIN T. M., STAHOVICH T., DAVIS R.: Sketch based interfaces: early processing for sketch understanding. In *PUI '01: Proceedings of the 2001 workshop on Perceptive user interfaces* (New York, NY, USA, 2001), ACM Press, pp. 1–8.

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