SketchREAD: A Multi-Domain Sketch Recognition Engine

Christine Alvarado
MIT CSAIL
Cambridge, MA 02139 USA
calvarad@csail.mit.edu

Randall Davis
MIT CSAIL
Cambridge, MA 02139 USA
davis@csail.mit.edu

ABSTRACT

We present SketchREAD, a multi-domain sketch recognition engine capable of recognizing freely hand-drawn diagrammatic sketches. Current computer sketch recognition systems are difficult to construct, and either are fragile or accomplish robustness by severely limiting the designer's drawing freedom. Our system can be applied to a variety of domains by providing structural descriptions of the shapes in that domain; no training data or programming is necessary. Robustness to the ambiguity and uncertainty inherent in complex, freely-drawn sketches is achieved through the use of context. The system uses context to guide the search for possible interpretations and uses a novel form of dynamically constructed Bayesian networks to evaluate these interpretations. This process allows the system to recover from low-level recognition errors (e.g., a line misclassified as an arc) that would otherwise result in domain level recognition errors. We evaluated SketchREAD on real sketches in two domains—family trees and circuit diagrams—and found that in both domains the use of context to reclassify low-level shapes significantly reduced recognition error over a baseline system that did not reinterpret low-level classifications. We also discuss the system's potential role in sketch-based user interfaces.

Categories and Subject Descriptors: I.5.4 [**Pattern Recognition**]: Applications; H.5.2 [**User Interfaces**]: Interaction Styles

Additional Keywords and Phrases: Pen-based UIs, input and interaction technology, sketch recognition, intelligent UIs, Bayesian networks

1 INTRODUCTION

While in recent years there has been an increasing interest in sketch-based user interfaces [9, 13, 14], the problem of robust free-sketch recognition remains largely unsolved. Because existing sketch recognition techniques are difficult to implement, and are error-prone or severely limit the user's drawing style, many previous systems that support sketching perform only limited recognition. ScanScribe, for example, uses perceptual guidelines to support image and text editing, but does not attempt to recognize the user's drawing [14]. Similarly, the sketch-based DENIM system supports the design of web pages but recognizes very little of the user's sketch [13]. Finally, NuSketch reasons about spatial relationships in military diagrams, but does not recognize sketched symbols [5]. Systems of this sort involve the computer in the early design, making it easy to record the design process, but they do not always facilitate automatic transition from the early stage design tool to a more powerful design system.

To enable the construction of sketch-based interfaces for a number of domains, we have created SketchREAD (Sketch Recognition Engine for mAny Domains), a system capable of understanding freely-drawn, messy, two-dimensional diagrammatic sketches. SketchREAD "understands" a user's sketch in that it parses a user's strokes as they are drawn and interprets them as objects in a domain of interest. SketchREAD operates in the background while the user sketches; recognition results may be displayed after the user completes the sketch or at any time during the recognition process. Our engine does not assume it will receive user feedback for its recognition, because having to give feedback can distract the user during the design process. It may be applied to any domain in which sketches may be described in terms of diagrammatic symbols (e.g., circuit diagrams, military course of action diagrams). Although SketchREAD is not designed to recognize other types of sketches (e.g., three-dimensional sketches and free-form sketches common in domains such as architecture) the class of sketches it is designed to recognize is important for designers in many domains. This system both helps solve a challenging problem in sketch understanding and enables more natural interaction with design software.

One of the most difficult problems in creating a sketch recognition system is handling the tradeoff between ease of recognition and drawing freedom. The more we constrain the user's drawing style, the easier recognition becomes. For example, if we enforce the constraint that each component in the domain must be a carefully drawn symbol that can be created with a single stroke, it is relatively easy to build recognizers capable of distinguishing between the symbols, as was done with Palm Pilot Graffiti™. The advantage of using restricted recognizers is accuracy; the disadvantage is the designer is constrained to a specific style of sketching.

Previous recognition-intensive systems have focused on tasks where drawing style assumptions can greatly reduce recognition complexity. Long *et al.* focus on designing special graphical symbols that will not be confused easily by the computer [10]. This approach improves recognition, but it limits the designer to a specific set of single-stroke symbols that may be natural only for certain tasks. The Quickset

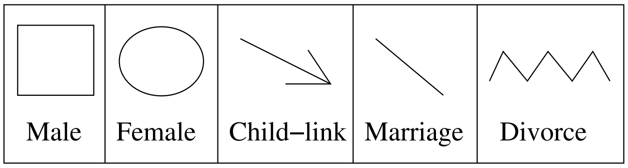


Figure 1: The symbols in the family tree domain.

system for recognizing military course of action (COA) diagrams uses multi-modal information to improve recognition of sketched symbols [20], but assumes that each symbol will be drawn independently, and that the user will likely speak the name of the symbol when drawing it. These assumptions aid recognition, but may fail for design tasks in other domains. In electrical engineering, for example, designers draw several symbols without pausing and probably do not speak the name of the symbols they draw. Other previous systems have similar constraints on drawing style or do not provide the level of recognition robustness we seek here [17, 6, 3].

While the previous systems have proven useful for their respective tasks, we aim to create a general sketch recognition system that does not rely on the drawing style assumptions of any one domain. To be usable, a sketch recognition-based system must make few enough mistakes that sketching is less work than using a more traditional (i.e., menu-based) interface. To be broadly effective the system's architecture should be easily applied across a variety of domains, without having to reengineer the system. Our system is a significant step toward achieving these goals.

Our approach makes four contributions to the field of sketch-based UIs. First, our engine separates information about basic shapes from their interpretation in a particular domain so that our engine can more easily be extended to multiple domains without having to recreate the shape recognizers. Second, we have developed a novel form of dynamically constructed Bayesian networks to allow both stroke data and higher-level shape information to influence the system's interpretation of the user's strokes. Third, our system uses this novel Bayesian network technique to guide its search for possible interpretations of the user's sketch, allowing it to recover from low-level interpretation errors (e.g., a line misclassified as an arc) that would otherwise prevent recognition of the sketch. Fourth, we gathered and tested our recognition engine on unconstrained freely-drawn data in two domains—family trees and circuits. We show that SketchREAD consistently reduced recognition errors over a baseline system that did not reinterpret low-level classifications. The results of these tests make concrete the strengths of our approach and the remaining challenges we face in building a recognition engine that can better handle real-world data.

We begin by exploring the challenges of recognizing real-world sketches. Next, we present our new approach to recognition, then analyze our system's performance on real data. We conclude with a discussion of how to extend our system's power and how it can be used in sketch recognition user interfaces (SkRUIs).

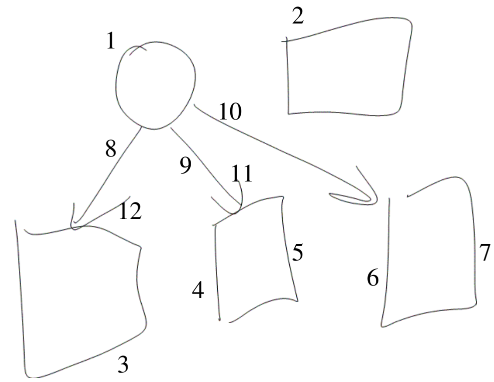


Figure 2: A partial sketch of a family tree.

2 THE CHALLENGES OF SKETCH UNDERSTANDING

Figure 2 shows the beginning of a sketch of a family tree, with the strokes labelled in the order in which they were drawn. The symbols in this domain are given in Figure 1. The user started by drawing a mother and a father, then drew three sons. He linked the mother to the sons by first drawing the shafts of each arrow and then drawing the arrowheads. (In our family tree diagrams, each parent is linked to each child with an arrow.) He will likely continue the drawing by linking the father to the children with arrows and linking the two parents with a line.

Although relatively simple, this drawing presents many challenges for sketch recognition. While previous recognition systems have addressed some of these challenges, SketchREAD is the first to address all of them using a general framework that can be extended to multiple domains.

The first challenge illustrated in Figure 2 is the incremental nature of the sketch process. Incremental sketch recognition allows the computer to seamlessly interpret a sketch as it is drawn and keeps the user from having to specify when the sketch is complete. To recognize a potentially incomplete sketch, a computer system must know when to recognize a piece of that sketch and when to wait for more information. For example, Stroke 1 can immediately be recognized as a female, but Stroke 6 cannot be recognized without Stroke 7.

The second challenge is that many of the shapes in Figure 2 are visually messy. For example, the center arrowhead (Stroke 11) looks more like an arc than two lines. Next, the stroke used to draw the leftmost quadrilateral (Stroke 3) looks like it is composed of five lines—the top of the quadrilateral is bent and could be reasonably divided into two lines by a stroke parser. Finally, the lines in the rightmost quadrilateral (Strokes 6 and 7) do not touch in the top left corner.

The third issue is segmentation: It is difficult to know which strokes are part of which shapes. For example, if the computer knew that Strokes 9 and 11 were part of one shape, the system would likely be able to match an arrow pattern to these strokes using a standard algorithm, such as a neural network. Unfortunately, segmentation is not an easy task. The shapes in this drawing are not clearly spatially segmented, and naively trying different combinations of strokes is prohibitively time consuming. To simplify segmentation, many previous systems (e.g., [13, 20]) assume each shape will be drawn with temporally contiguous strokes. This assumption does not hold here.

There are also some inherent ambiguities in how to segment the strokes. For example, lines in our domain indicate marriage, but not every line is a marriage-link. The shaft of the leftmost arrow (Stroke 8) might also have been interpreted as a marriage link between the female (Stroke 1) and the leftmost male (Stroke 3). In this case, the head of that arrow (Stroke 12) could have been interpreted as a part of the drawing that is not yet complete (e.g., the beginning of an arrow from the leftmost quadrilateral (Stroke 3) to the top quadrilateral (Stroke 2)).

Finally, how shapes are drawn can also present challenges to interpretation. The head of the rightmost arrow (part of Stroke 10) is actually made of three lines, two of which overlap to form one side of the arrowhead. In order to recognize the arrow, the system must know to collapse those two lines into one, even though they do not actually overlap. Another challenge arises because the same shape may not always be drawn in the same way. For example, the arrows on the left (Strokes 8 and 12, and Strokes 9 and 11) were drawn differently from the one on the right (Stroke 10) in that the user first drew the shaft with one stroke and then drew the head with another. This variation in drawing style presents a challenge for segmentation and recognition because a system cannot know how many strokes will be used to draw each object, nor the order in which the parts of a shape will appear.

Many of the difficulties described in the example above arise from the messy input and visual ambiguity in the sketch. It is the context surrounding the messy or ambiguous parts of the drawing that allows humans to interpret these parts correctly. We found that context also can be used to help our system recover from low-level interpretation errors and correctly identify ambiguous pieces of the sketch. Context has been used to aid recognition in speech recognition systems; it has been the subject of recent research in computer vision [18, 19] and has been used to a limited extent in previous sketch understanding systems [3, 6, 13]. We formalize the notion of context suggested by previous sketch recognition systems. This formalization improves recognition of freely drawn sketches using a general engine that can be applied to a variety of domains.

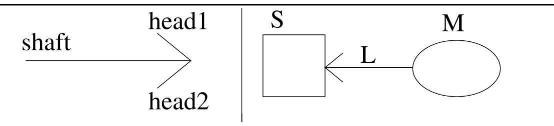
3 TECHNICAL APPROACH

We have developed and implemented a general framework for sketch recognition that handles the challenges presented in the previous section and that can be applied to a variety of domains by supplying domain specific pattern descriptions.

3.1 Knowledge Representation

We use a hierarchical shape description language to describe the shapes in a domain. A hierarchical representation is useful because it enables re-use of geometric shapes (e.g., arrows) in a variety of domains, and because many sketched symbols are compositional. Here we describe the language only briefly. For a more complete description, see [7].

Figure 3 shows a simple use of the language. The arrow is an example of a *shape*, which we use to mean a pattern recognizable in a given domain. The arrow is a *compound shape*, i.e., one composed of non-recursive *subshapes* fit together according to *constraints*. A line is a *primitive shape*—one that cannot be decomposed into subshapes. Although primitive shapes cannot be decomposed into subshapes, they may have named *subcomponents* that can be used when describing other shapes, e.g., the endpoints of a line, “p1” and “p2”, used in Figure 3. *Domain shapes* are shapes that have semantic meaning in a particular domain. For example, `line`, `arrow` and `child-link` are all shapes that may be recognized, but only a child-link has meaning in the family tree domain. *Domain patterns* are combinations of domain shapes that are likely to occur, for example a child-link pointing from a female to a male, indicating a relationship between mother and son. Recognition information for primitive shapes and constraints are built in to SketchREAD; compound shapes, including domain shapes and patterns, are recognized from primitive shapes and vary depending on the domain to which SketchREAD is applied.



DEFINE ARROW
(**Subshapes**
L1,L2,L3: (Line *shaft head1 head2*))
(**Constraints**
C1: (coincident *shaft.p1 head1.p1*)
C2: (coincident *shaft.p1 head2.p2*)
C3: (equal-length *head1 head2*)
C4: (shorter *head1 shaft*)
C5: (acute-angle *head1 shaft*)
C6: (acute-angle *head2 shaft*)

DEFINE MOTHER-SON
(**Subshapes**
M,S,L: (Female *M*) (Male *S*) (Child-link *L*)
(**Constraints**
C1: (touches *L.head S*)
C2: (touches *L.tail M*))

Figure 3: The description of the shape “arrow” and the domain pattern “mother-son.” Child-links are defined from arrows, males from quadrilaterals, and females from ellipses. Labels for the subshapes and constraints are used in Figure 4.

The context in which a shape is likely to occur is given by the higher-level shapes and domain patterns in which it appears. For example, the domain pattern `mother-son` provides a context in which child-links (and in turn arrows), males and females are likely to occur. This representation allows our system to incorporate both domain knowledge and shape information in its interpretation of the user’s sketch. The separation between domain shapes and their geometric subshapes facilitates the re-use of geometric shapes in other domains (e.g., the arrow in electrical engineering symbols). The shapes and domain patterns for the family tree domain are listed in Table 1; constraints are omitted to save space.

3.2 Recognition Overview

Recognizing the sketch is a matter of parsing a user’s strokes according to the specified visual language. Visual language parsing has been studied [12], but most previous approaches

Shape/Pattern (Abbr.)	Subshapes
Line (L)	–
Ellipse (E)	–
Polyline (PL)	–
Arrow (A)	Line *h1, h2, shaft*
Quadrilateral (Q)	Line *l1, l2, l3, l4*
Marriage-link (ML)	Line *l*
Divorce-link (DL)	Polyline *pl*
Female (F)	Ellipse *e*
Male (M)	Quadrilateral *m*
Child-link (CL)	Arrow *a*
Divorce (Div)	Male *h*; Female *w*; DL *l*
Marriage (Mar)	Male *h*; Female *w*; ML *l*
Partnership-F (PartF)	Female *w1*, *w2*; ML *l*
Partnership-M (PartM)	Male *h1*, *h2*; ML *l*
Father-daughter (FD)	Male *f*; Female *d*; CL *l*
Mother-daughter (MD)	Female *m, d*; CL *l*
Father-son (FS)	Male *f, s*; CL *l*
Mother-son (MS)	Female *m*; Male *s*; CL *l*

Table 1: A complete list of the shapes and domain patterns in the family tree domain.

assume diagrammatic input free from low-level recognition errors and cannot handle realistic, messy, stroke-based input. Mahoney and Fromherz use mathematical constraints to cope with the complexities of parsing sketches of curvilinear configurations such as stick figures [11]. Shilman *et al.* [16] present a parsing method similar to our approach, with two differences. First, their work employs a spatially-bounded search for interpretations that quickly becomes prohibitively expensive. Second, their parsing method builds and scores a parse tree for each interpretation independently; we allow competing interpretations to influence each other.

As the user draws, our system uses a two-stage generate and test recognition process to parse the strokes into possible interpretations. This two-dimensional parsing problem presents a challenge for a real-time system. Noise in the input makes it impossible for the system to recognize low-level shapes with certainty or to be sure whether or not constraints hold. Low-level misinterpretations cause higher-level interpretations to fail as well. On the other hand, trying all possible interpretations of the user's strokes guarantees that an interpretation will not be missed, but is infeasible due to the exponential number of possible interpretations.

To solve this problem we use a combined bottom-up and top-down recognition algorithm that generates the most likely interpretations first (bottom-up), then actively seeks out parts of those interpretations that are still missing (top-down). Our approach uses a novel application of dynamically constructed Bayesian networks to evaluate partial interpretation hypotheses and then expands the hypothesis space by exploring the most likely interpretations first. The system does not have to try all combinations of all interpretations, but can focus on those interpretations that contain at least a subset of easily-recognizable subshapes and can recover any low-level subshapes that may have been mis-recognized.

3.3 Hypothesis Evaluation

Our method of exploring the space of possible interpretations depends on our ability to assess partial hypotheses. We

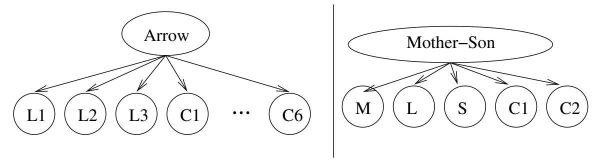


Figure 4: The Bayesian network fragment constructed from the description of an arrow and the domain pattern "mother-son" given in Figure 3.

use a dynamically constructed Bayesian network to evaluate the current set of hypothesized interpretations. (We discuss below how these hypotheses are generated.) We give an overview and illustration of this method here; more details are presented in [1].

Bayesian networks provide a framework for combining multiple sources of evidence to reason about uncertainty in the world. They consist of two parts: a directed acyclic graph that encodes *which* factors in the world influence each other, and a set of conditional probability distributions that specify *how* these factors influence one another. Each node in the graph represents something to be measured, and a link between two nodes indicates that there is a causal relationship from one node to another. Each node contains a conditional probability table specifying how it is influenced by its parents. For more information, see [4], which provides an intuitive overview of Bayesian networks.

Bayesian networks are traditionally used to model static domains in which the variables and relationships between those variables are known in advance. Static networks are not suitable for the task of sketch recognition because we cannot predict *a priori* the number of strokes or symbols the user will draw. Therefore, our network structure must be changed to reflect each new stroke. To allow the network to grow as new data arrives, we specify a library of Bayesian network fragments that describe shapes and domain patterns. This framework is similar to the Object-Oriented Bayesian Networks proposed in [8] but has been developed specifically for use in a real-time recognition system. As an example of our representation, the fragments for the descriptions from Figure 3 are given in Figure 4. As the recognition system proposes interpretations for the user's strokes, it makes copies of the corresponding fragments from the library and links them together to form a complete Bayesian network (as in Figure 5). Each node, n, in the network has two values, `true` and `false`, and represents a possible interpretation for a subset of the user's strokes or a constraint between interpretations. $P(n = \texttt{true})$ reflects the strength of that interpretation. The complete network contains the set of all the interpretations the system is considering.

Recall that links in the Bayesian network indicate causal relationships, so the arrow fragment in Figure 4 represents the hypothesis that the user intended to draw an arrow, which in turn "caused" her to produce three lines that obey a corresponding set of constraints (i.e. they are connected, two of them are the same length, etc.). Similarly, the user's intent to draw a mother-son relationship caused her to draw a male, a female and a child-link, which in turn caused her to draw an ellipse, an arrow and a quadrilateral, and so forth.

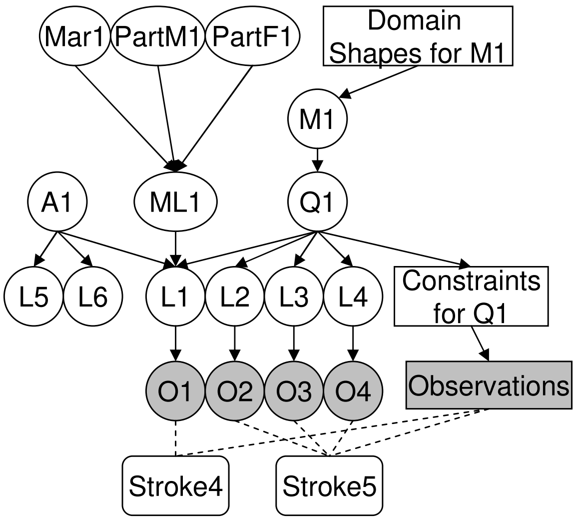


Figure 5: A portion of the interpretation network generated while recognizing the sketch in Figure 2. Abbreviations are given in Table 1

To explain the semantics of the Bayesian network further, we consider a piece of the network that is generated in response to Strokes 4 and 5 in the example given in Figure 2. Figure 5 shows the part of the Bayesian network representing the possible interpretations that the system generated for these strokes (which we call the *interpretation network*). Each node represents a hypothesized interpretation for some piece of the sketch. For example, Q1 represents the system's hypothesis that the user intended to draw a quadrilateral. A higher level hypothesis is compatible with the lower level hypotheses it points to. For example, if M1 (the hypothesis that the user intended to draw a male) is correct, Q1 (the hypothesis that the user intended to draw a quadrilateral) and L1-L4 (the hypotheses that the user intended to draw 4 lines) will also be correct. Two hypotheses that both point to the same lower level hypothesis represent competing interpretations for the lower level shape and are incompatible. For example, A1, ML1 and Q1 are three possible higher level interpretations for line L1, only one of which may be true.

Hypotheses are linked to stroke data with observation nodes that represent measurements taken from one or more strokes. For example, L1 is linked to O1, which is measured from Stroke 4. Consequently, L1 represents the hypothesis that Stroke 4 is a line, not simply that there is a line somewhere on the page. A1 is a partial hypothesis–it represents the hypothesis that L1 (and hence Stroke 4) is part of an arrow whose other two lines have not yet been drawn. Line nodes representing lines that have not been drawn (L5 and L6) are not linked to observation nodes because there is no stroke from which to measure these observations. We refer to these nodes (and their corresponding interpretations) as *virtual*.

The probability of each interpretation is influenced both by stroke data (through its children) and by the context in which it appears (through its parents), allowing the system to handle noise in the drawing. For example, the bottom edge of quadrilateral Q1 is slightly curved (see Figure 2); stroke data (O4) only weakly supports the corresponding line hypothesis (L4). However, the other three edges of Q1 are fairly straight, and O1-O3 raise the probabilities of L1-L3, respectively, which in turn raise the probability of Q1. Q1 provides a context in which to evaluate L4, and because Q1 is well supported by L1-L3 (and by the constraint nodes), it raises the probability of L4.

The fact that partial interpretations have probabilities allows the system to assess the likelihood of incomplete interpretations based on the evidence it has seen so far. In fact, even virtual nodes have probabilities, corresponding to the probability that the user (eventually) intends to draw these shapes but either has not yet drawn this part of the diagram or the correct low-level hypotheses have not yet been proposed because of low-level recognition errors. As we describe below, a partial interpretation with a high probability cues the system to examine the sketch for possible missed low-level interpretations.

3.4 Hypothesis Generation

The major challenge in hypothesis generation is to generate the correct interpretation as a candidate hypothesis without generating too many to consider in real-time. A naive approach to hypothesis generation simply would attempt to match all shapes to all possible combinations of strokes, but this would produce an exponential number of interpretations. Our method of evaluating partial interpretations allows us to use a bottom-up/top-down generation strategy that greatly reduces the number of hypotheses considered but still generates the correct interpretation for most shapes in the sketch.

Our hypothesis generation algorithm has three steps:

1. **Bottom-up step**: As the user draws, the system parses the strokes into primitive objects using a domain-independent recognition toolkit developed in previous work [15]. Compound interpretations are hypothesized for each compound object that includes these low-level shapes, even if not all the subshapes of the pattern have been found.
2. **Top-down step**: The system attempts to find subshapes that are missing from the partial interpretations generated in step 1, often by reinterpreting strokes that are temporally and spatially proximal to the proposed shape.
3. **Pruning step**: The system removes unlikely interpretations.

This algorithm, together with the Bayesian network representation presented above, deals successfully with the challenges presented in Section 2. Using the example in Figure 2, we illustrate how the system generates hypotheses that allow the Bayesian network mechanism to resolve noise and inherent ambiguity in the sketch, how the system manages the number of potential interpretations for the sketch, how the system recovers from low-level recognition errors, and how the system allows for variation in drawing style.

Based on low-level interpretations of a stroke, the bottom-up step generates a set of hypotheses to be evaluated using the Bayesian network mechanism presented in the previous section. In the sketch in Figure 2, the user's first stroke is correctly identified as an ellipse by the low-level recognizer, and from that ellipse the system generates the interpretation `ellipse`, and in turn, partial interpretations for mother-son,

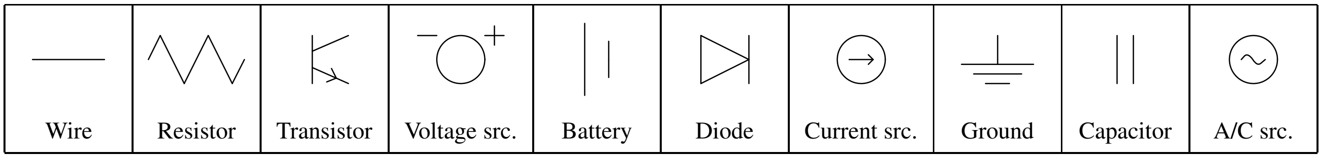


Figure 6: The symbols in the circuit domain.

mother-daughter, father-daughter, marriage, partner-female, and divorce. These proposed interpretations are called *templates* that have a *slot* for each subshape. Future interpretations will be filled into empty slots.

Naive bottom-up interpretation easily can generate too many hypotheses to consider in real-time. We employ three strategies to control the number of hypotheses generated in the bottom-up step. First, when an interpretation can be fit into more than one slot in a higher-level template (e.g., in Figure 5, L1 could be the shaft or either of the lines in the head of A1), the system arbitrarily chooses one of the valid slots rather than generating one hypothesis for each potential fit. Later, the system can shuffle the shapes in the template when it attempts to fit more subshapes.

Second, the system does not generate higher-level interpretations for interpretations that are only partially filled. The lines generated from Strokes 4 and 5 result in one partial hypothesis—`arrow` (A1)—and two complete hypotheses—`quadrilateral` (Q1) and `marriage-link` (ML1) (Figure 5). Continuing to generate higher-level templates from partial hypotheses would yield a large number of hypotheses (one hypothesis for each higher level domain pattern involving each existing partial hypothesis). To avoid this explosion, the system continues to generate templates using only the complete hypotheses (in this case, ML1 and Q1).

Third, when the system processes polylines, it assumes that all the lines in a single polyline will be used in one interpretation. While this assumption does not always hold, in practice we find that it is often true and greatly reduces the number of possible interpretations. The system recognizes Stroke 2 as a 4-line polyline. The bottom-up step generates only a quadrilateral because that is the only shape in the domain that requires four lines.

The top-down step allows our system to recover from low-level recognition errors. Stroke 3 is incorrectly, but reasonably, parsed into 5 lines by the low-level recognizer. Because the system does not know about any 5-line objects, but does know about things that contain fewer than 5 lines, it attempts to re-segment the stroke into 2 lines, 3 lines and 4 lines (with a threshold on acceptable error). It succeeds in re-segmenting the stroke into 4 lines and successfully recognizes the lines as a quadrilateral. Although the 4 line fit is not perfect, the network allows the context of the quadrilateral in addition to the stroke data to influence the system's belief in the 4-line interpretation. Also note that the 5 lines from the original segmentation remain in the interpretation network.

The system controls the number of interpretations in the network through pruning, which occasionally causes it to prune a correct hypothesis before it is complete. The top-down step regenerates previously-pruned hypotheses, allowing the system to correctly interpret a symbol despite variations in drawing order. The leftmost arrow in Figure 2 was drawn with two non-consecutive strokes (Strokes 8 and 12). In response to Stroke 8, the system generates both an arrow partial hypothesis and a marriage-link hypothesis (using the line hypothesis generated for this stroke). Because the user does not immediately complete the arrow, and because the competing marriage-link hypothesis is complete and has a high probability, the system prunes the arrow hypothesis after Stroke 9 is drawn. Later, Stroke 12 is interpreted as a 2-line polyline and a new arrow partial hypothesis is generated. The top-down step then completes this arrow interpretation using the line generated previously from Stroke 8, effectively regenerating a previously pruned interpretation.

3.5 Selecting an Interpretation

As each stroke is drawn, the sketch system uses a greedy algorithm to select the best interpretation for the sketch. It queries the Bayesian network for the strongest complete interpretation, sets aside all the interpretations inconsistent with this choice, chooses the next most likely remaining domain interpretation, and so forth. It leaves strokes that are part of partial hypotheses uninterpreted. Although the system selects the most likely interpretation at every stroke, it does not eliminate other interpretations. Partial interpretations remain and can be completed with the user's subsequent strokes. Additionally, the system can change its interpretation of a stroke when more context is added.

4 APPLICATION AND RESULTS

Applying SketchREAD to a particular domain involves two steps: specifying the structural descriptions for the shapes in the domain and specifying the prior probabilities for the domain patterns and any top-level shapes (i.e., those not used in domain patterns, which, consequently, will not have parents in the generated Bayesian network. See [1] for details on how probabilities are assigned to other shapes). We applied SketchREAD to two domains: family trees and circuits. For each domain, we wrote a description for each shape and pattern in that domain (Figures 1 and 6) and estimated the necessary prior probabilities by hand. Through experimentation, we found the recognition performance to be insensitive to the exact values of these priors.

4.1 Data Collection

SketchREAD can recognize simple sketches nearly perfectly in both the family tree and circuit domains, but we wanted to test its performance on more complex, real-world data. Our goal was to collect sketches that were as natural and unconstrained as the types of sketches people produce on paper to test the limits of our system's recognition performance. To collect these sketches, we used a data collection program for the Tablet PC developed by others in our group that allows the user to sketch freely and displays the user's strokes

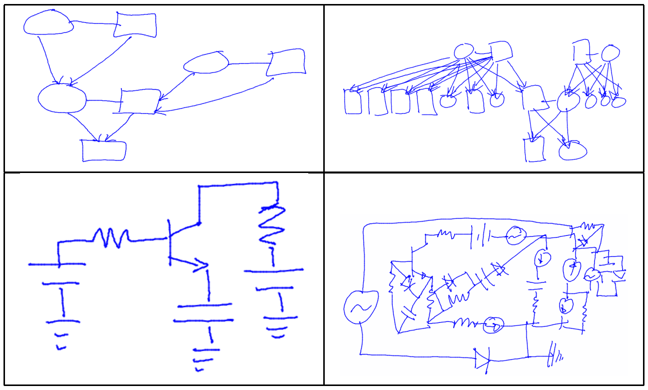


Figure 7: Examples that illustrate the range of complexity of the sketches collected.

exactly as she draws them, without performing any type of recognition. Most of our users had played with a Tablet PC before they performed our data collection task but had never used one for an extended period of time. None used any type of pen-based computer interface on a regular basis. The users first performed a few warm-up tasks, at the end of which all users expressed comfort drawing on the Tablet PC.

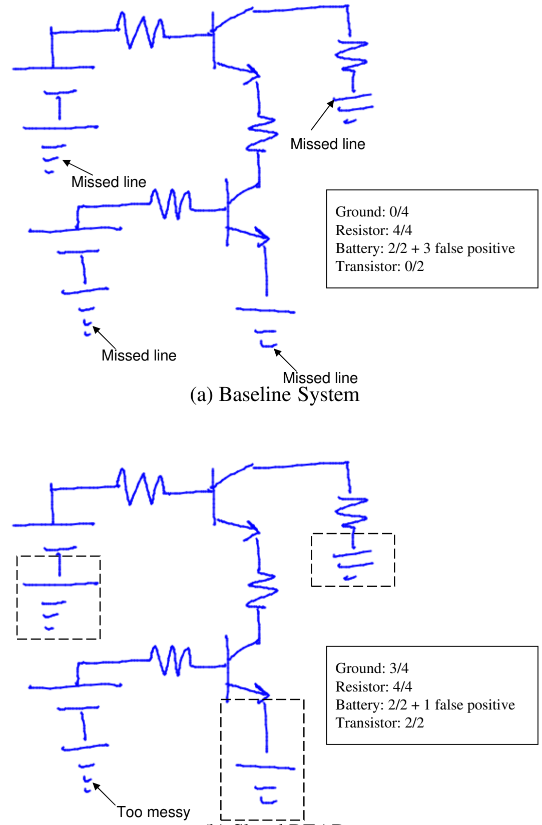
To collect the family tree sketches, we asked each user to draw her family tree using the symbols presented in Figure 1. Users were told to draw as much or as little of the tree as they wanted and that they could draw the shapes however felt natural to them. Because erasing strokes introduces subtleties into the recognition process that our system is not yet designed to deal with, users were told that they could not erase, and that the exact accuracy of their family tree diagram was not critical. We collected ten sketches of varying complexity.

We then recruited subjects with basic knowledge of circuit diagram construction and showed them examples of the types of circuits we were looking for. After a warm-up task, subjects were instructed to draw several circuits. We specified the number and types of components to be included in the circuit and then and asked them to design any circuit using those components. Subjects were instructed not to worry about the functionality of their circuit, only that they should try to produce realistic circuits. We collected 80 diagrams in all.

The circuit diagrams were considerably more complicated than the family tree diagrams. One limiting assumption that SketchREAD currently makes is that the user will not draw more than one symbol with a single stroke. Unfortunately, in drawing circuit diagrams, users often draw many symbols with a single stroke. To allow SketchREAD to handle the circuit diagrams, we broke apart strokes containing multiple objects by hand. This is clearly a limitation of our current system; we discuss below how it might be handled.

5 Performance Results

We ran SketchREAD on each family tree sketch and each circuit sketch. We present qualitative results, as well as aggregate recognition and running time results for each domain. Our results illustrate the complexity our system can currently handle, as well as the system's current limitations. We discuss those limitations below, describing how best to use the system in its current state and highlighting what needs to be done to make the system more powerful. Note that to apply the system to each domain, we simply loaded the domain's shape information; we did not modify the recognition system. Although SketchREAD does not perform perfectly on every sketch, its generality and performance on complex sketches illustrates its promise over previous approaches.



(a) Baseline System

(b) SketchREAD

Figure 8: Recognition performance example. Overall recognition results (# correct / total) are shown in the boxes.

Figure 8 illustrates how our system is capable of handling noise in the sketch and recovering from missed low-level interpretations. In the baseline case, one line from each ground symbol was incorrectly interpreted at the low-level, causing the ground interpretations to fail. SketchREAD was able to reinterpret those lines using the context of the ground symbol in three of the four cases to correctly identify the symbol. In the fourth case, one of the lines was simply too messy, and SketchREAD preferred to (incorrectly) recognize the top two lines of the ground symbol as a battery.

In evaluating our system's performance, direct comparisons with previous work are difficult, as there are few (if any) published results for this type of recognition task. The closest published sketch recognition results are for the Quickset system, which also uses top-down information (through multi-modal input) to recognize sketched symbols, but this system assumes object segmentation, making its recognition task different from ours [20]. We compared SketchREAD's recognition performance with the performance of a strictly bottom-up approach of the sort used in previous systems

	Size	#Shapes	% Correct	
			BL	SR
Mean	**50**	**34**	**50**	**77**
S1	24	16	75	100
S2	28	16	75	87
S3	29	23	57	78
S4	32	22	31	81
S5	38	31	54	87
S6	48	36	58	78
S7	51	43	26	72
S8	64	43	49	74
S9	84	49	42	61
S10	102	60	57	80

Table 2: Recognition rates for the baseline system (BL) and SketchREAD (SR) for each sketch for the family tree domain. The size column indicates the number of strokes in each sketch.

	Total	% Correct		# False Pos	
		BL	SR	BL	SR
AC Source	4	100	100	35	29
Battery	96	60	89	56	71
Capacitor	39	56	69	27	14
Wire	1182	62	67	478	372
Ground	98	18	55	0	5
Resisitor	330	51	53	7	8
Voltage Src.	43	2	47	1	8
Diode	77	22	17	0	0
Current Src.	44	7	16	0	0
Transistor	43	0	7	0	14

Table 3: Aggregate recognition rates for the baseline system (BL) and SketchREAD (SR) for the circuit diagrams by shape.

[3, 13]. This strictly bottom-up approach combined low-level shapes into higher-level patterns without top-down reinterpretation. Even though our baseline system did not reinterpret low-level interpretations, it was not trivial. It could handle some ambiguities in the drawing (e.g., whether a line should be interpreted as a marriage-link or the side of a quadrilateral) using contextual information in the bottom-up direction. To encourage others to compare their results with those presented here we have made our test set publicly available at http://rationale.csail.mit.edu/ETCHASketches.

We measured recognition performance for each system by determining the number of correctly identified objects in each sketch (Table 2 and Table 3). For the family tree diagrams SketchREAD performed consistently and notably better than our baseline system. On average, the baseline system correctly identified 50% of the symbols while SketchREAD correctly identified 77%, a 54% reduction in the number of recognition errors. Due to inaccurate low-level recognition, the baseline system performed quite poorly on some sketches. Improving low-level recognition would improve recognition results for both systems; however, SketchREAD reduced the error rate by approximately 50% independent of the performance of the baseline system. Because it is impossible to build a perfect low-level recognizer, SketchREAD's ability to correct low-level errors will always be important.

Circuit diagrams present SketchREAD with more of a challenge for several reasons. First, there are more shapes in the circuit diagram domain and these shapes are more complex. Second, there is a stronger degree of overlap between shapes in the circuit diagrams. For example, it can be difficult to distinguish between a capacitor and a battery. As another example, a ground symbol contains within it (at least one) battery symbol. Finally, there is more variation in the way people draw circuit diagrams, and their sketches are messier causing the low-level recognizer to fail more often. They tend to include more spurious lines and over-tracings.

Overall, SketchREAD correctly identified 62% of the shapes in the circuit diagrams, a 17% reduction in error over the baseline system. It was unable to handle more complex shapes, such as transistors, because it often failed to generate the correct mapping between strokes and pieces of the template. Although the system attempts to shuffle subshapes in a template in response to new input, for the sake of time it cannot consider all possible mappings of strokes to templates. We discuss below how we might extend SketchREAD to improve its performance on complex domains such as circuit diagrams.

We measured SketchREAD's running time to determine how it scales with the number of strokes in the sketch. Figure 9 graphs the median time to process each stroke for each domain. The vertical bars in the graph show the standard deviation in processing time over the sketches in each domain. (One family tree diagram took a particularly long time to process because of the complexity of its interpretation network, discussed below. This sketch affected the median processing time only slightly but dominated the standard deviation. It has been omitted from the graph for clarity.) Three things about these graphs are important. First, although SketchREAD does not yet run in real-time, the time to process each stroke in general increased only slightly as the sketch got larger. Second, not every stroke was processed by the system in the same amount of time. Finally, the processing time for the circuit diagrams is longer than the processing time for the family trees.

By instrumenting the system, we determined that the processing time is dominated by the inference in the Bayesian network, and all of the above phenomena can be explained by examining the size and complexity of the interpretation network. The number of nodes in the interpretation network grows approximately linearly as the number of strokes increases. This result is encouraging, as the network would grow exponentially using a naive approach to hypothesis generation. The increase in graph size accounts for the slight increase in processing time in both graphs. The spikes in the graphs can be explained by the fact that some strokes not only increased the size of the network, but had more higher-level interpretations, creating more fully connected graph structures, which causes an exponential increase in inference time. After being evaluated, most of these high-level hypotheses were immediately pruned, accounting for the sharp drop in processing time on the next stroke. Finally, the fact that circuits take longer to process than family trees is related to the relative complexity of the shapes in the domain. There are more shapes in the circuit diagram domain and they are more complex, so the system must consider more interpretations for the user's strokes, resulting in larger and more connected Bayesian networks.

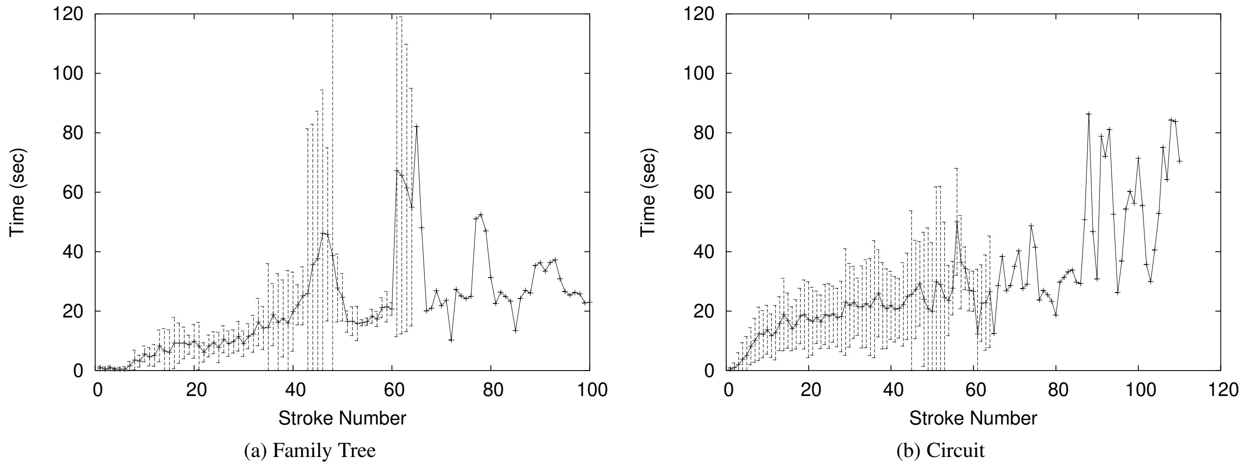


Figure 9: The median incremental time it took the system to process each stroke in the family tree and circuit diagrams. Vertical bars show the standard deviation across the sketches in each domain.

6 SKETCH RECOGNITION USER INTERFACES (SKRUIS)

Our goal is to use SketchREAD in sketch recognition user interfaces (SkRUIs). In this section we discuss SketchREAD's strengths and limitations with two purposes. First, we discuss the type of interface in which SketchREAD can be used currently. Second, we discuss the system's limitations and how they can be addressed so that its overall performance can be improved to better handle more complicated domains.

6.1 Building Design Tools with SketchREAD

We presented results on sketches of family trees and circuits, but SketchREAD can be applied to any domain that has a clear language of symbols. Currently SketchREAD is best used in simple domains where the shapes have little overlap and are drawn spatially separated. As an example, SketchREAD was able to process the simpler family trees in near real-time. It ran into difficulty when encountering many overlapping shapes, such as the arrows the top right drawing in Figure 7. Based on these guidelines, we used SketchREAD to build a sketch-based system for constructing box and arrow diagrams in Power Point and conducted an informal user study on this system [2]. People sketched their diagrams freely while SketchREAD ran in the background. The system produced few errors in this simple domain, allowing us to investigate issues such as how and when to display recognition feedback, and how to allow people to edit their sketches.

While interpretations are selected after every stroke, determining when to display these interpretations to the user is by itself an interesting and difficult question of human computer interaction. One approach we are exploring is to use the system's determination of the completeness of an interpretation to inform it when to display an interpretation and when to wait for more strokes.

Even though SketchREAD reduces recognition error, it will never eliminate it. SketchREAD's interpretation-graph architecture could itself be used to reduce the burden placed on the user in correcting the system's errors. Often in complicated diagrams, errors are interconnected. If there are many competing interpretations, choosing the wrong interpretation for one part of the diagram often will lead the system to choose the wrong interpretation for the surrounding strokes. SketchREAD could display its recognition results on top of the user's original strokes, so that the user can see the context of these results. Then, the user could help the system by tracing over the strokes for one of the symbols that was misrecognized. This retracing would help the system recover from the error because the user's strokes would be cleaner and because the system would know that they were all part of a single symbol. Then, based on this new interpretation, the system could reevaluate the surrounding strokes and (hopefully) recover some of the missed interpretations that might still exist but simply were not currently chosen as the best interpretation.

Finally, one of the main causes of recognition error might be dealt with through UI design. The system had trouble with the fact that the users varied the structure of their symbols even though they were explicitly shown the desired structure for each symbol. For example, although we instructed people to draw a voltage source using a circle with a plus and a minus next to it, some people put the plus and minus inside the circle. SketchREAD is designed to handle variations in the way people draw, but cannot handle such unexpected changes to the basic shape of the symbol. Although ideally we would like to support all methods people have for drawing each object, this might never be possible. Instead, a simple interactive training step before a new user uses the interface could help eliminate this type of variation without imposing too many limitations on the user's drawing style.

6.2 Performance Improvement

SketchREAD significantly improves the recognition performance of unconstrained sketches. However, its accuracy, especially for complicated sketches and domains, is still too low to be practical in most cases. Here we consider how to improve the system's performance.

First, while SketchREAD always corrected some low-level interpretation errors, its overall performance still depended on the quality of the low-level recognition. Our low-level recognizer was highly variable and could not cope with some users' drawing styles. In particular, it often missed corners of polylines, particularly for symbols such as resistors. Other members of our group are working on a low-level recognizer that adapts to different users' drawing styles.

Second, although in general SketchREAD's processing time scaled well as the number of strokes increased, it occasionally ran for a long period. The system had particular trouble with areas of the sketch that involved many strokes drawn close together in time and space and with domains that involve more complicated or overlapping symbols. This increase in processing time was due almost entirely to in increase in Bayesian network complexity.

We suggest two possible solutions. First, part of the complexity arises because the system tries to combine new strokes with low-level interpretations for correct high-level interpretations (e.g., the four lines that make a quadrilateral). These new interpretations were pruned immediately, but they increased the size and complexity of the network temporarily, causing the bottlenecks noted above. In response, we are testing methods for "confirming" older interpretations and removing their subparts from consideration other higher-level interpretations as well as confirming their values in the Bayesian network so that their posterior probabilities do not have to be constantly re-computed. Second, we can modify the belief propagation algorithm we are using. We currently use Loopy Belief Propagation, which repeatedly sends messages between the nodes until each node has reached a stable value. Each time the system evaluates the graph, it resets the initial messages to one, essentially erasing the work that was done the last time inference was performed, even though most of the graph remains largely unchanged. Instead, this algorithm should begin by passing the messages it passed at the end of the previous inference step.

Finally, because our recognition algorithm is stroke-based, spurious lines and over-tracing hindered the system's performance in both accuracy and running time. A preprocessing step to merge strokes into single lines would likely greatly improve the system's performance. Also, in the circuit diagram domain, users often drew more than one object with a single stroke. A preprocessing step could help the system segment strokes into individual objects.

7 CONCLUSION

We have shown how to use context to improve online sketch interpretation and demonstrated its performance in SketchREAD, an implemented sketch recognition system that can be applied to multiple domains. We have shown that SketchREAD is more robust and powerful than previous systems at recognizing unconstrained sketch input in a domain. The capabilities of this system have applications both in human computer interaction and artificial intelligence. Using our system we will be able to further explore the nature of usable intelligent computer-based sketch systems and gain a better understanding of what people would like from a drawing system that is capable of understanding their freely-drawn sketches as more than just strokes. This work provides a necessary step in uniting artificial intelligence technology with novel interaction technology to make interacting with computers more like interacting with humans.

8 ACKNOWLEDGEMENTS

This work is supported by the Microsoft iCampus project and by MIT's Project Oxygen. We would like to thank our users for their time and the members of the MIT Design Rationale Group and Alex Snoeren for helpful editing comments. We would also like to thank Alex Snoeren and Aaron Adler for help with compiling recognition results.

REFERENCES

1. C. Alvarado. *Multi-Domain Sketch Understanding*. PhD thesis, Massachusetts Institute of Technology, 2004.
2. C. Alvarado. Sketch recognition and usability: Guidelines for design and development. In *AAAI Fall Symposium on Pen-Based Interaction*, 2004.
3. C. Alvarado and R. Davis. Resolving ambiguities to create a natural sketch based interface. In *Proc. of IJCAI*, 2001.
4. E. Charniak. Bayesian networks without tears: making bayesian networks more accessible to the probabilistically unsophisticated. *Artificial Intelligence*, 12(4):50–63, 1991.
5. K. D. Forbus, J. Usher, and V. Chapman. Sketching for military course of action diagrams. In *Proc. of IUI*, 2003.
6. M. Gross and E. Y.-L. Do. Ambiguous intentions: a paper-like interface for creative design. In *Proc. of UIST*, 1996.
7. T. Hammond and R. Davis. LADDER: A language to describe drawing, display, and editing in sketch recognition. In *Proc. of IJCAI*, 2003.
8. D. Koller and A. Pfeffer. Object-oriented bayesian networks. In *Proc. of UAI*, 1997.
9. J. A. Landay and B. A. Myers. Interactive sketching for the early stages of user interface design. In *Proc. of CHI*, 1995.
10. A. C. Long, Jr., J. A. Landay, L. A. Rowe, and J. Michiels. Visual similarities of pen gestures. In *Proc. of CHI*, 2000.
11. J. V. Mahoney and M. P. J. Fromherz. Three main concerns in sketch recognition and an approach to addressing them. In *AAAI Spring Symposium on Sketch Understanding*, 2002.
12. K. Marriott, B. Meyer, and K. Wittenburg. A survey of visual language specification and recognition. In K. Marriott and B. Meyer, editors, *Visual Language Theory*, pages 5–85. Springer-Verlag, 1998.
13. M. W. Newman, J. Lin, J. I. Hong, and J. A. Landay. DENIM: An informal web site design tool inspired by observations of practice. *Human-Computer Interaction*, 18(3):259–324, 2003.
14. E. Saund, D. Fleet, D. Larner, and J. Mahoney. Perceptually supported image editing of text and graphics. In *Proc. of UIST*, 2003.
15. T. M. Sezgin, T. Stahovich, and R. Davis. Sketch based interfaces: Early processing for sketch understanding. In *Proc. of PUI*, 2001.
16. M. Shilman, H. Pasula, S. Russell, and R. Newton. Statistical visual language models for ink parsing. In *AAAI Spring Symposium on Sketch Understanding*, 2002.
17. T. Stahovich, R. Davis, and H. Shrobe. Generating multiple new designs from a sketch. *Artificial Intelligence*, 104(1-2):211–264, 1998.
18. T. M. Strat and M. A. Fischler. Context-based vision: Recognizing objects using information from both 2-d and 3-d imagery. *IEEE Trans. on PAMI*, 13(10):1050–1065, 1991.
19. A. Torralba and P. Sinha. Statistical context priming for object detection. In *Proc. of ICCV*, 2001.
20. L. Wu, S. L. Oviatt, and P. R. Cohen. Multimodal integration—a statistical view. *IEEE Trans. on Multimedia*, 1(4):334–341, 1999.