Magic Paper: Sketch-Understanding Research

Randall Davis
Massachusetts Institute of Technology

Sketches are hand-drawn informal figures often created as a way of thinking about or working through a problem. Sketch-understanding systems let users interact with computers by drawing naturally, offering a freedom not available with traditional CAD systems.

Sketching is ubiquitous: We draw as a way of thinking, solving problems, and communicating in a wide variety of fields, for both design (such as sketches of conceptual designs) and analysis (such as sketches drawn to help puzzle through problems in physics or electronic circuits).

Unfortunately in today's technology, sketches are dead—they're either graphite on slices of dead trees, or, if captured on a PDA or tablet computer, simply pixels of digitized ink. The Sketch Understanding Group at MIT has been working toward a kind of "magic paper"—that is, a surface that's as natural and easy to draw on as paper, yet that understands what you draw.

What does it mean for the paper to "understand"? One example, in Figure 1, shows some of our earliest work. We use Assist (A Shrewd Sketch Interpretation and Simulation Tool) to sketch simple 2D physical devices, then watch them behave.[1] Assist understands the raw sketch shown in Figure 1a in the sense that it interprets the ink the same way we do (Figure 1b), that is, as an inclined plane with a wheeled cart. As Figure 1c shows, it hands this interpretation to a physics simulator, which animates the device, giving the user the experience of drawing on intelligent paper.

One detail helps illustrate the sense in which the system understands the sketch in a manner similar to a human observer. The wheels (blue circles) are attached to the car's body with pin joints (pink circles), yet the user draws both the wheels and pin joints with the same geometric shape—a circle. The system interprets a circle differently based on context—a circle can be a pin joint only if it's drawn over two bodies that are already overlapping (in this case the car body and the wheel).

We've built sketch-understanding systems for a variety of domains, including the simple physics sketcher shown in Figure 1, Unified Modeling Language diagrams (Figure 2), analog circuits (Figure 3), and chemical structure sketches[2] (Figure 4).

TERMINOLOGY AND FOCUS

Sketches differ from diagrams. By "diagrams" we mean the more formal, at times draftsman-like figures that CAD systems produce, while sketches are the hand-drawn informal figures people create on paper, whiteboards, napkins, and, more recently, tablet computers. There is a significant body of work on understanding diagrams such as the International Conferences on Document Analysis and Recognition, but the task of understanding a sketch is significantly different.

Research on sketch understanding also differs from the sizable body of work on handwriting understanding. Sketch-understanding work proceeds largely from an attempt to recognize the shape of the objects drawn, using the same notion of shape that people use. Handwriting recognition, on the other hand, doesn't attempt to recognize each letter by its shape, and has successfully used machine-learning techniques that derive distinguishing features that might or might not correspond to what people attend to.

Published by the IEEE Computer Society

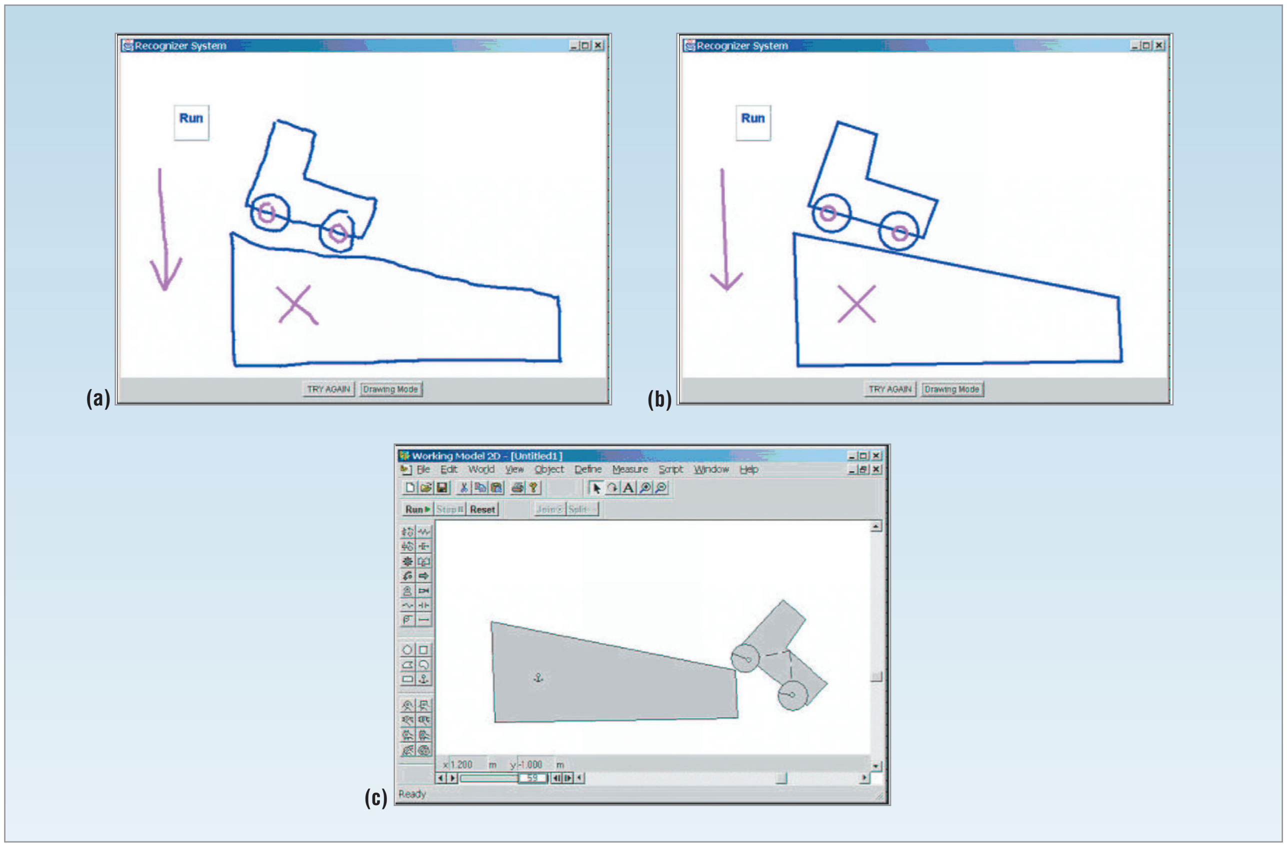
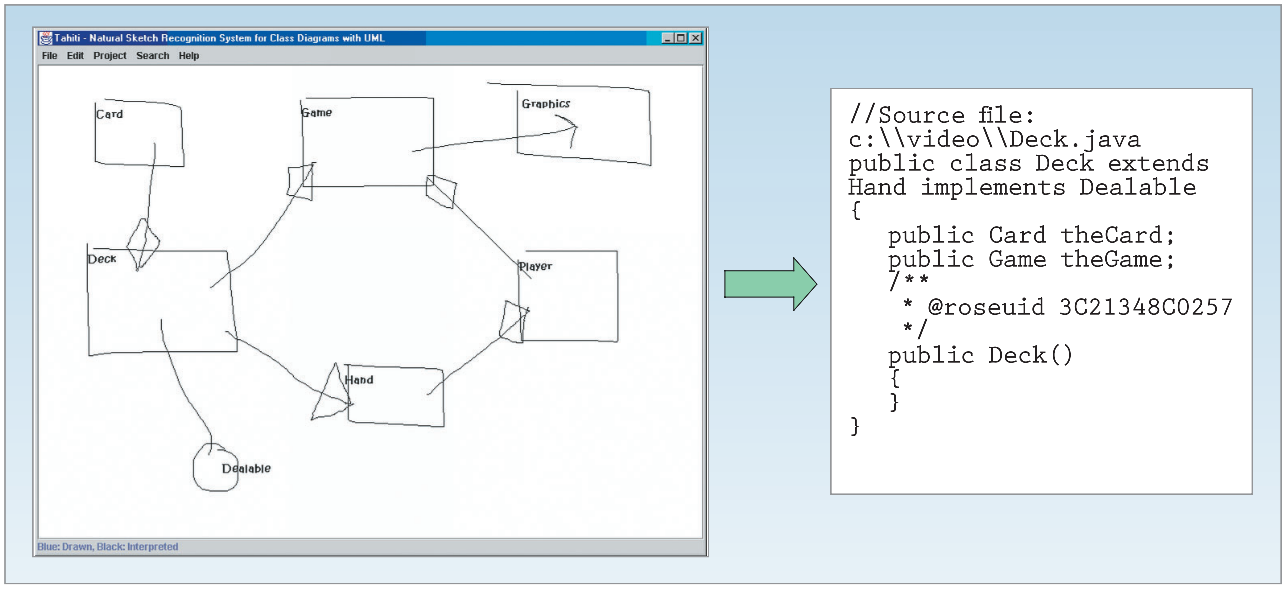


Figure 1. Assist system. (a) The initial sketch, (b) the sketch as cleaned up by Assist, and (c) the simulation, showing the consequences.


Tahiti - Natural Sketch Recognition System for Class Diagrams with UML
File Edit Project Search Help
Card
Game
Graphics
Deck
Player
Hand
Dealable
Blue: Drawn, Black: Interpreted

```
//Source file:
c:\\video\\Deck.java
public class Deck extends
Hand implements Dealable
{
   public Card theCard;
   public Game theGame;
   /**
    * @roseuid 3C21348C0257
    */
   public Deck()
   {
   }
}
```

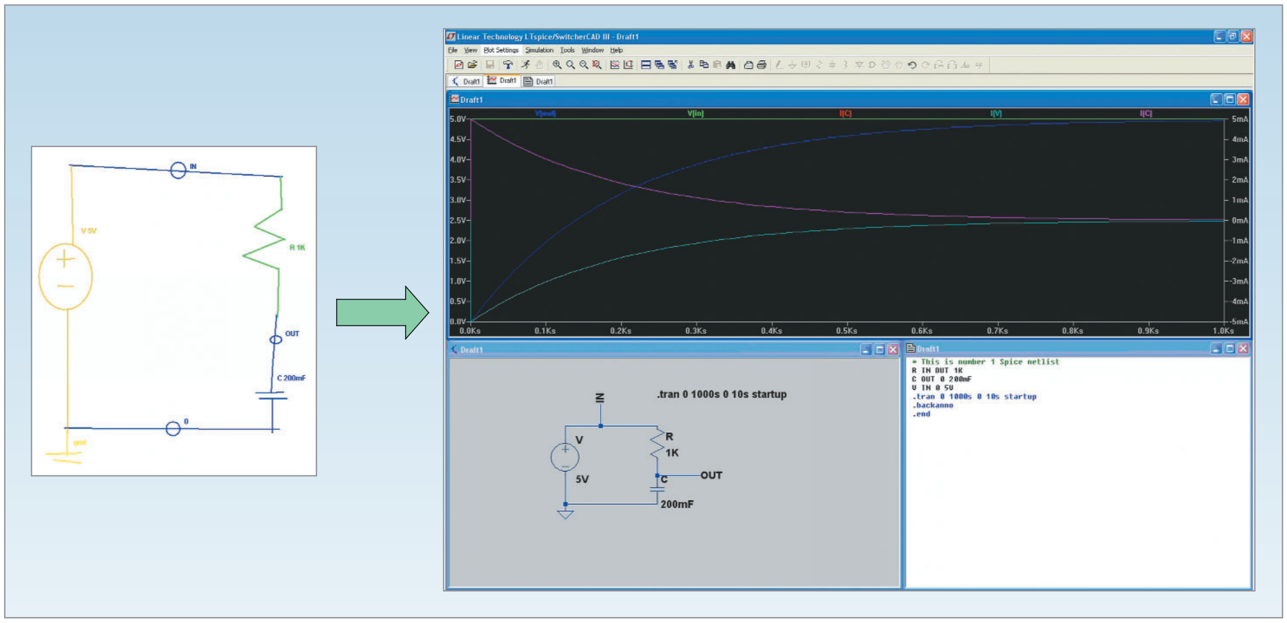
Figure 2. A UML sketch recognized then turned into code using RationalRose.

WHY SKETCH?

Given the prevalence and power of design-automation tools, it's reasonable to ask, why sketch when we have tools to create far more precise and polished drawings?

Two things motivate the desire to enable sketching.

First, picking up a pen is still more natural than using a keyboard and mouse, and free-form sketched figures seem more intuitive than the formal shapes a CAD program generates. There's a freedom of thought that seems to go with the ease of sketching, an intuition supported by evi-



```
* This is number 1 Spice netlist
R IN OUT 1K
C OUT 0 200mF
V IN 0 5V
.tran 0 1000s 0 10s startup
.backanno
.end
```

Figure 3. An analog circuit sketch recognized, then analyzed by Spice.

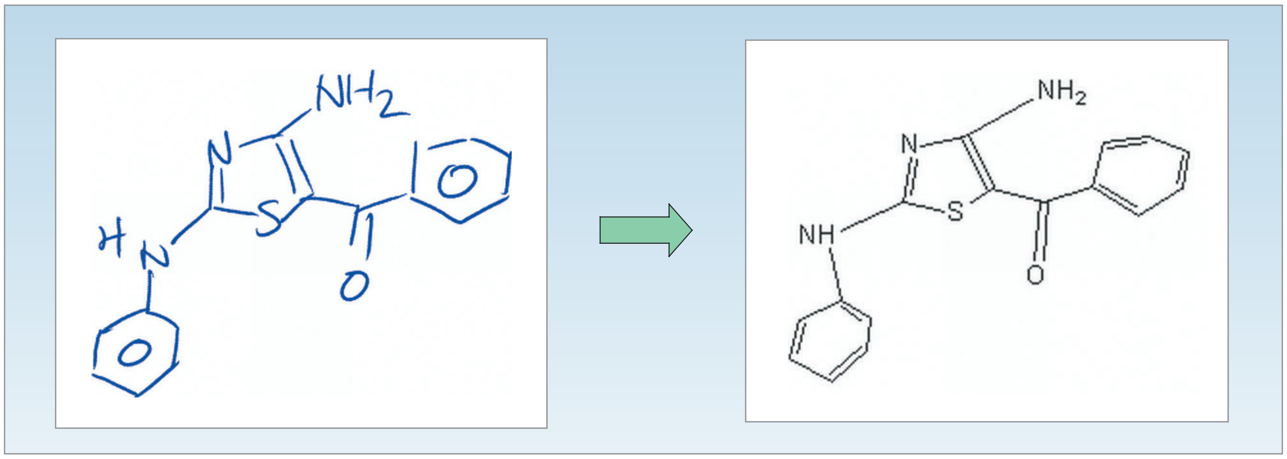


Figure 4. A chemical structure sketch recognized, then redrawn by ChemDraw.

dence from cognitive science research, which showed that designers who sketched produced more design alternatives than did those who were using a drafting tool.[3]

Second, and perhaps more important, CAD systems typically require commitments that the sketcher might not want to make, particularly at the early conceptual design stage, or when sketching a rough picture for an analysis problem. It is, for example, impossible simply to draw a line in, say, a mechanical CAD system: Every line has a precise dimension, angle, and so on. Yet, when you want to dash off an idea as to how a device might work, the need to specify precise sizes, angles, radii, and so on gets in the way.

DIFFICULTIES WITH SKETCH INTERPRETATION

Sketch understanding seems so quick and intuitive when we do it that we may wonder why creating software to do the same thing is so difficult. One way to see the difficulties in the task is to view the problem as one of signal interpretation (interpreting a time-stamped sequence of points). Many of the standard signal interpretation issues arise, along with some novel ones.

Sketch interpretation in general is difficult in direct proportion to the user's allowed degree of freedom—the less constrained the drawing style, the more difficult the interpretation task. At one end of the spectrum lie techniques such as Graffiti, which prescribes a specific set of gestures to be drawn in a specified manner. In these circumstances, recognition is considerably easier, and the computational demand relatively modest. The tradeoff is the necessity of learning and using a particular drawing style. An intermediate position is taken by work like that in Silk,[4] which suggests doing minimal interpretation of the strokes, in part because the system is intended as a design platform, and in part to avoid interfering with the creative process.

Our work lies closer to the unrestricted end of the spectrum. We want people to be able to draw as they would normally, without specifying the number, order, or direction of strokes in a symbol, yet still have the system understand. Figure 5 illustrates several difficulties that arise from this.

First, our task is incremental. That is, we want the system to interpret users' strokes as they're drawn. This lets the system provide continuous feedback about its understanding of the sketch, so the user can make corrections when a misunderstanding arises. But it also means that interpretation must be continually revised. In Figure 5, for example, the user has not yet drawn the line connecting the circle and rectangle in the upper part of the sketch that will indicate a married couple.

In addition, as in any signal interpretation problem, there is noise. In this case, noise arises from our inability to draw with mechanical precision: Lines wiggle, curves intended to be closed might have a gap or an overlap, and so on.

Next, the drawing conventions in many domains permit variations. In analog circuits, for example, you can draw capacitors as two straight parallel lines or as a straight line and a curved line.

Individual styles also vary, across users and even within a sketch. In Figure 5, the user drew the three successive rectangles in the lower part of the figure, yet drew them in three different ways (using a single stroke, a squared C-shape closed by a line, and two L-shaped strokes).

Another issue is the difficulty of segmentation. Consider the arrow pointing toward the rectangle at the bottom left in Figure 5. The arrowhead touches the rectangle, but isn't connected to the shaft. Determining which strokes might belong together requires understanding the domain.

Next, difficulties arise when the user lays down extensive amounts of ink, as when strokes are overtraced or when figures are filled in—for example, the filled-in triangles used in drawing some arrowheads. The problem here is that the fundamental view of the sketch as a set of strokes or time-stamped points breaks down. A filled-in shape is a particularly clear example: It's of little use to represent the ink in the interior of the shape as a sequence of strokes. It might have been laid down that way, but the intent was to create a 2D area of continuous ink; the particular set or sequence of strokes used is irrelevant. Overtracing is similar. It's the ink's appearance that is intended to create the line. The numerous (possibly zigzagging) strokes used to lay down the ink are irrelevant. This calls for a change of representation, from the sketch as a sequence of strokes to an area of ink.

Finally, and perhaps most interesting, the signal is both 2D and nonchronological. It's 2D in the obvious sense that, unlike normal handwriting, it spreads across the page. In this it's similar to but more unrestricted than handwritten mathematical formulas.

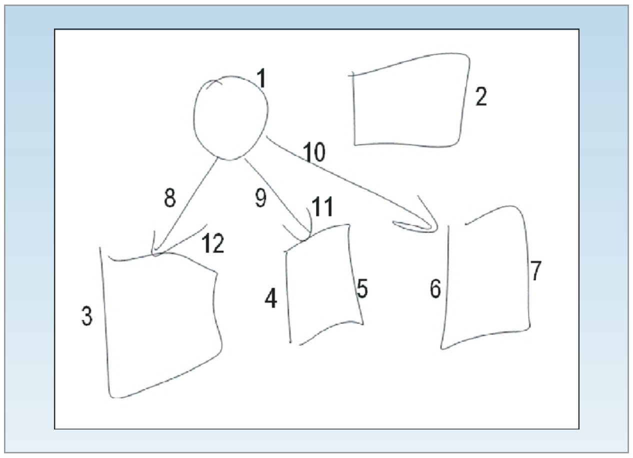


Figure 5. Snapshot of a family tree as a user is drawing it. Squares indicate males, circles indicate females, arrows indicate offspring, and a straight line (not yet drawn) indicates a marriage. Numbers have been added to indicate stroke order.

The signal is nonchronological in the sense that we don't require each object to be finished before the next is started, so a user might add strokes to a sketch to complete something started earlier. For example, as the numbers in Figure 5 indicate, the user drew the arrows' shafts in sequence, and only later added the heads.

This differs from other signal interpretation problems, such as speech recognition. When talking you might restate something, but you can't go back in time and change the sounds you made earlier. Yet in sketching, newly added strokes can change the interpretation of strokes made earlier.

SKETCH UNDERSTANDING

Two basic assumptions ground most work in sketch understanding.

First, the work is done in domains where there's a reasonably well-established graphical lexicon and grammar. The lexicon is the set of shapes used in a domain—for example, the standard graphical notation for digital circuit components. The grammar describes interrelations among shapes, indicating, for example, that a transistor symbol should be connected to three other components. Many scientific and engineering domains have such a graphical lexicon and syntax, but this is clearly not universal. The sort of impressionistic sketches drawn by architects early in the design process, for example, are not describable in these terms.

Second, much like work in speech understanding, sketch-understanding systems are built for a specific domain. Unrestricted sketch understanding is currently out of reach, for many of the same reasons that unrestricted speech understanding isn't particularly successful.

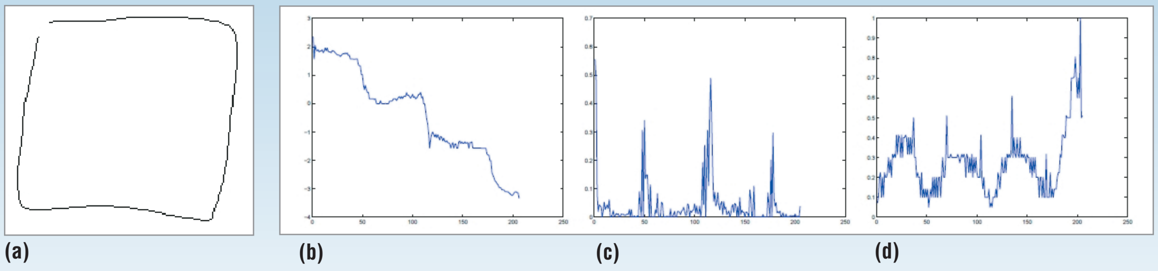


Figure 6. Characteristics of a drawing. (a) A hand-drawn square with corresponding graphs showing the drawing's (b) direction, (c) curvature, and (d) speed.

Representing a sketch

At the most basic level, a sketch is a collection of time-stamped coordinate points, grouped into individual strokes—that is, from pen down to pen up. As with handwriting recognition, online and offline sketching differ. Offline sketches must be scanned and lines located, introducing additional noise, while the raw data of online sketches is composed of strokes whose locations and widths are known precisely.

Finding primitives

One basic task common to much sketch-recognition research involves reinterpreting the raw data as primitive shapes—lines and arcs. This low-level processing is an interesting task by itself, given the noisiness of the data.

One common approach to finding primitives uses the data's temporal character, based on the observation that, when drawing by hand, people routinely slow down at corners, without consciously attempting to do so.[5] This lets the system locate corners more precisely by combining information about curvature and speed, looking for points that combine high curvature and low speed.

Although combining information from these two sources helps, the problem is still difficult, due to the fine-grained noisiness of the data (Figure 6). In response, we have explored the use of scale-space filtering to remove the noise, relying on the basic intuition (common to all scale-space approaches) that the signal and noise have different spatial frequencies.[6] As those frequencies are not generally determinable a priori, that work uses dynamic selection of the appropriate scale from the data itself.

After locating the corners, we represent strokes as piecewise linear approximations, with additional processing merging nearly collinear segments, and using arcs (for example, Bezier curves) where an arc is a better fit than a line or line sequence.[5]

We can then use the resulting lines and arcs as the raw material in any of the representation and recognition approaches described next.

Recognizing shapes

The variety of information available in online sketches enables three distinctly different representations, with three correspondingly different approaches to recognition:

- how the shape is defined—the set of geometric constraints the shape must obey to be an instance of a particular object;
- how the shape is drawn—the sequence of strokes used; and
- what the shape looks like—the traditional concept of image appearance.

We've built recognition systems to explore each of these representations individually and are working to understand how to combine them.

How it's defined. We represent a shape's definition in a language called Ladder.[7] As Figure 7 shows, Ladder descriptions list the basic components making up the shape and the geometric constraints that must hold between them. Recognizing individual shapes is then a process of matching the strokes drawn against a set of shape descriptions for a domain.[8]

The description is significant both for what it constrains—for example, to be an arrow the shaft must be longer than the heads—and for what it doesn't constrain—for example, it doesn't specify the lines' orientation or length, letting the system recognize arrows of any size and in any orientation.

How it's drawn. When drawing the standard symbol for an AND-gate (Figure 8a), most people start with the vertical stroke, add the arc, then draw the connecting wires. This sequence, along with just a few others, accounts for most of the routinely encountered stroke orders. This is true for a wide variety of shapes, not just those as simple as in Figure 8.

This observation provides a foundation for recognition by observing stroke sequence, using a variation on a hidden Markov model (HMM) called a dynamic Bayes' net. We train the DBN to recognize shapes based on the sequence of primitive geometric elements such as lines and arcs found in the strokes.[9]

```
components
 Line shaft
 Line head1
 Line head2
constraints
 longer shaft head1
 equal head1 head2
 coincident head1.p1 head2.p1
 coincident head1.p1 shaft.p1
 acuteMeet head1 shaft
 acuteMeet shaft head2
```

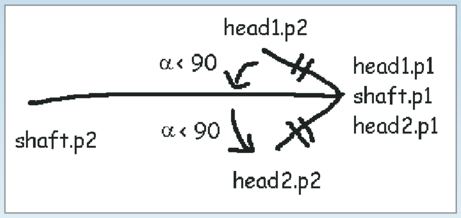


Figure 7. The Ladder description of an arrow. The definition includes the shape's basic components and their geometric constraints.

One standard simplifying assumption in an HMM-based approach is that each shape is completed before the next is started. This assumption puts a tractable limit on the number of states the HMM must have. If you can arbitrarily intersperse shapes, the number of states needed in the HMM increases exponentially. Although people don't typically intersperse shapes arbitrarily, neither do they reliably finish one object before drawing another. When drawing the transistor in Figure 8b, for example, people routinely draw the vertical connecting wire strokes before the arrowhead strokes that complete the transistor itself. We have extended the DBN approach to learn and then deal with the sort of interspersing encountered in real-world practice.[9]

How it looks. The long history of image-understanding research provides various approaches to recognizing a sketch by its appearance. Several of these techniques have been used in sketch understanding, typically by matching the user's strokes (viewed as bitmaps) against templates, using some combination of traditional distance metrics—for example, the Hausdorff distance, Tanimoto coefficient, and so on.[10]

One difficulty with sketches is that, unlike images, we can't assume that the template and image are related by an affine transform. Hand-drawn sketches of symbols routinely exhibit variations that are nonuniform over the symbol, making the matching task more challenging, inspiring efforts to develop variations on the traditional metrics.

Recognizing a sketch by how it looks is useful in dealing with situations such as overtracing and filled-in shapes, where the stroke-based representation breaks down. Here we need to determine whether the ink looks like a line, no matter what sequence or collection of strokes produced the ink.

One recent approach inspired by work in image processing attempts to capture local visual features, rather than match an entire template.[11] It builds on the notion of shape contexts, measuring the ink's placement and orientation at various points in a shape using a small circular bull's-eye pattern. A collection of these patterns forms a characterization of the entire shape, which we can then compare to similar characterizations made for template figures.

SKETCH-ENABLED INTERFACES

Given the ability to translate strokes into object descriptions, we can connect sketch understanding to various back-end programs. We produced the simulation in Figure 1, for example, by sending our sketch understander's output to a physics simulator and having it compute and animate the resulting behavior.

We've experimented with several other back-end programs as well. We have, for example, connected a UML sketch-understanding system to RationalRose. Users sketch a class hierarchy and RationalRose generates class declaration code for that hierarchy. In another application, we linked a program that understands sketches of chemical structures to ChemDraw, which redraws the structure neatly so we can check our interpretation. We also plan to link our chemical sketch understander to one or more databases of information about chemical compounds. These databases contain massive amounts of useful information, but currently must be searched by providing one of the text-based structural encodings (such as Smiles or the International Chemical Identifier), which are nonintuitive, to say the

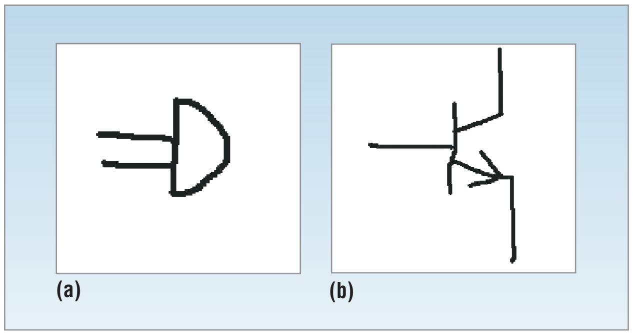


Figure 8. Hand-drawn shapes. (a) A hand-sketched AND-gate. Most people start this drawing with the vertical line, then add the arc, and the connecting wires. (b) A hand-drawn transistor.

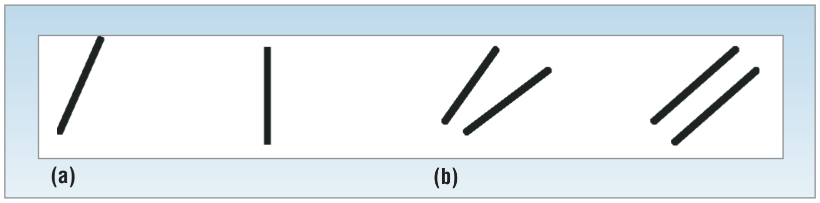


Figure 9. Line orientations. (a) A slanted and a vertical line. Rotate a slanted line slightly and it's still a slanted line, but rotating a vertical line changes its verticality property. (b) Two slanted lines and two parallel lines. Rotating one of a pair of parallel lines can change the parallelism property.

least. For example, the InChI language encodes the elementary compound L-ascorbic acid as `InChI=1/C6H8O6/c7-1-2(8)5-3(9)4(10)6(11)12-5/h2,5,7-10H,1H2/t2-,5+/m0/s1`. The complexity of this notation arises from the need to capture in linear text information about connectivity, 3D orientation, and so on. It will be far easier and more natural simply to draw a molecule and use that as the query.

Linking sketch-understanding systems to back-end applications allows us to sketch-enable those applications, expanding the modalities used to interact with them, and consequently vastly simplifying their use.

LEARNING A NEW DOMAIN

When sketch understanding is done by matching descriptions against strokes, creating a sketch understander for a new domain can be as straightforward as writing a new set of shape definitions—much as some speech recognizers let users write lexicons and grammars for a new domain. This is considerably easier than writing custom code for recognizers and lets nonprogrammers develop sketching interfaces.

Writing shape descriptions for a new domain, however, isn't always easy. It is challenging at times because even some simple shapes might involve multiple lines of text (for example, the 11 lines needed to describe an arrow). Writing descriptions is also challenging because shape definitions have inevitable subtleties, in the form of constraints that are typically not obvious until they're violated (for example, the barbs of an arrow need to meet the end of the shaft at an acute angle). In practice, people find it difficult to think of all the constraints a shape must meet, leading to definitions that are underconstrained. Other times, they write a description while thinking of a particular instance that isn't general enough, leading to overconstrained definitions.

We've explored a variety of techniques for learning shape definitions by example, attempting in particular to learn from a single example. The problem is intriguing because going from a specific drawing to an appropriately generalized description is a classic problem in learning. The fundamental question is, what about the specific example is essential and what is accidental?

This is a particularly challenging question where drawings are concerned because they contain so much information. When viewed geometrically, every line has a length and orientation, intersection points with other lines, numerous pairwise relations—*parallel*, *perpendicular*, *near*, and *far*—as well as higher arity relations, such as *between*. Separating the essential from the accidental thus means selecting the important properties from numerous irrelevancies.

Consider the drawing in Figure 8a. If we're told that this is an example of an AND-gate, precisely which geometric properties are essential to its being an AND-gate? Does it matter that the arc is connected to both ends of the vertical line? Does it matter that the ratio of the lengths of the two parallel lines is 1.034? Even someone unfamiliar with the domain would likely guess that the answers are yes and no, respectively. But how might a program even begin to arrive at these answers?

One useful insight comes from work by some early 20th-century psychologists, who suggested that our perceptual systems are drawn toward detecting what they termed *singularities*—geometric properties that are in a sense fragile and hence unlikely to have been accidental. Figure 9 illustrates the concept: Most people would describe the first line in Figure 9a as slanted—and the next line as vertical—its verticality jumps out at us. The verticality property is "fragile" in the sense that the slanted line, if rotated slightly, is still a slanted line, while a vertical line rotated slightly loses its verticality property. The situation is similar for properties like horizontal, parallel (Figure 9b), and perpendicular.

We've used this insight to select from among the profusion of properties present in a sketch those relations that people are likely to attend to, using them as a reasonable first-pass guess at the essential properties.[12] The intuition here is that a graphical notation is more likely to be usable if it depends on the geometric properties that our eyes and brain tend to pick out. Hence, those properties are more likely to be the sketch's essential properties.

We've built on this work by engaging the user in a description-refinement process, using an interesting variation on near-misses as a learning vehicle.[13] Where past work on near-misses has relied on the user to offer the system near-miss examples, our research has shown that the program can generate its own near misses, querying the user about their status as examples or nonexamples of the concept. This has the well-known advantage of speeding learning, while having the system take on the difficult task of determining what example would be most informative to consider next.

We've also explored learning from multiple examples. This has the familiar problem of requiring numerous

such examples, as well as the less familiar difficulty of getting examples that explore a sufficiently wide range of possibilities. As one trivial example, when asked to draw examples of arrows and near-miss arrows, people rarely think of the near-miss arrow that has its shaft shorter than the arrowheads.

SKETCHING AND MULTIMODALITY

Watch anyone draw in a routine environment—for example, on a whiteboard during a meeting—and you'll almost invariably observe that they speak and gesture as they draw. Not infrequently, their gestures and words are essential to making sense of the sketch, adding information not representable in the drawing. As a consequence, researchers have shown considerable interest in the ability to use all these modalities, leading to several efforts that have explored various modality combinations, producing proof-of-concept demonstrations of their value and plausibility.[14, 15]

Sketch recognition today is at a stage roughly analogous to the early years of speech recognition, when isolated words could routinely be recognized, but continuous speech was still a research goal. Several sketch-understanding systems and approaches deal with individual symbols reasonably well, even those of moderate complexity, with a reasonable amount of noise, overtracing, and so on. No correlate yet exists to speech-recognition system grammars—in sketching, the inter-symbol relations—that approaches the complexity manageable in speech work, but this is one important direction of effort and progress in the field.[7]

Future work also seeks to extend recognition capabilities beyond line drawings to the more impressionistic sketching of engineers, where the profusion of strokes and the depiction of 3D structure present a set of difficult challenges.

Efforts at sketch understanding are in some ways an attempt to get back to the future—we want to return to the world where we can pick up a drawing implement and freely sketch out an idea, inspiration, or depiction of a problem to be solved, but at the same time, have a sketching surface that is intelligent enough to understand what's being drawn and thus can facilitate the design and analysis process. It's an intriguing undertaking, one with rapid current progress and the promise of considerable payoff. ■

Acknowledgments

I'd like to acknowledge the work of numerous collaborators whose impressive contributions are summarized briefly here, as well as the organizations that have supported this work, including Ford Motor Co., the MIT/Oxygen Project, Microsoft, Intel, and the Pfizer Corporation.

References

1. C. Alvarado and R. Davis, "Resolving Ambiguities to Create a Natural Sketch-Based Interface," *Proc. Int'l Joint Conf. Artificial Intelligence* (IJCAI), AAAI Press, 2001, pp. 1365-1371.
2. T.Y. Ouyang and R. Davis, "Recognition of Hand-Drawn Chemical Diagrams," *Proc. AAAI 2007*, CD-ROM, AAAI Press, 2007.
3. V. Goel, *Sketches of Thought*, MIT Press, 1995.
4. J. Landay and B. Myers, "Sketching Interfaces: Toward More Human Interface Design," *Computer*, Mar. 2001, pp. 56-64.
5. T.M. Sezgin, T. Stahovich, and R. Davis, "Sketch-Based Interfaces: Early Processing for Sketch Understanding," *Proc. Workshop Perceptive User Interfaces*, ACM Press, 2001, pp. 1-8.
6. T.M. Sezgin and R. Davis, "Scale Space-Based Feature Point Detection for Digital Ink," *Proc. AAAI Fall Symp. Series 2004: Making Pen-Based Interaction Intelligent and Natural*, AAAI Press, pp. 145-151.
7. T. Hammond and R. Davis, "Ladder: A Language to Describe Drawing, Display, and Editing in Sketch Recognition," *Proc. Int'l Joint Conf. Artificial Intelligence* (IJCAI), AAAI Press, 2003, pp. 461-467.
8. T. Hammond and R. Davis, "Automatically Transforming Symbolic Shape Descriptions for Use in Sketch Recognition," *Proc. AAAI*, AAAI Press, 2004, pp. 450-456.
9. T.M. Sezgin, "Sketch Interpretation Using Multiscale Stochastic Models of Temporal Patterns," PhD thesis, Dept. of Electrical Eng., MIT, 2006.
10. L.B. Kara and T.F. Stahovich, "An Image-Based, Trainable Symbol Recognizer for Hand-Drawn Sketches," *Computers & Graphics*, vol. 29, no. 4, 2005, pp. 501-517.
11. M. Oltmans, "Envisioning Sketch Recognition: A Local Feature-Based Approach to Recognizing Informal Sketches," PhD thesis, Dept. of Electrical Eng., MIT, 2007.
12. O. Veselova and R. Davis, "Perceptually Based Learning of Shape Descriptions," *Proc AAAI*, AAAI Press, 2004, pp. 482-487.
13. T. Hammond and R. Davis, "Interactive Learning of Structural Shape Descriptions from Automatically Generated Near-Miss Examples," *Proc. Intelligent User Interfaces*, ACM Press, 2006, pp. 37-40.
14. E. Kaiser et al., "Demo: A Multimodal Learning Interface for Sketch, Speak and Point Creation of a Schedule Chart," *Proc. Int'l Conf. Multimodal Interfaces* (ICMI), ACM Press, 2004, pp. 329-330.
15. A. Adler and R. Davis, "Speech and Sketching for Multimodal Design," *Proc. 9th Int'l Conf. Intelligent User Interfaces*, ACM Press, 2004, pp. 214-216.

***Randall Davis** is a professor in the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology. His research interests include sketch understanding and multimodal interaction. Davis received a PhD in artificial intelligence from Stanford University. He is a Founding Fellow of the Association for Artificial Intelligence and served terms as both councilor and president. Contact him at davis@csail.mit.edu.*