Whiteboards that Compute: A Workload Analysis

Ryan Dixon and Timothy Sherwood
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93117-5110
Email: {rsd,sherwood}@cs.ucsb.edu

***Abstract*—A whiteboard that automatically identifies drawn strokes, interprets them in context, and augments drawn images with computational results, such as solutions to mathematical equations or results of circuit simulations, is a surprisingly realistic goal for systems architects. In this paper we describe the state of this emerging domain and argue that technical trends will make this a particularly attractive workload in the future. We provide a preliminary characterization of the critical loops that exist within one state of the art system, currently undergoing development, and we attempt to quantify the workloads that whiteboard-sized devices are likely to face in the future. While this work is by no means a typical workload characterization paper, given the shift in programming models that we are about to endure, it is now more important than ever before to identify and understand those applications that have the potential to drive our industry forward.**

I. Introduction

Even after more than thirty years of desktop computing, many (if not most) engineers facing complex problems instinctively run right to the whiteboard. The whiteboard offers an incredibly natural way by which ideas can be remembered, it provides a means of describing and communicating the relationships between complex interacting parts, and can be used to specify rough schematics of final designs. This interface of drawn strokes on a large surface is so natural that even 35,000 years ago our ancestors were doing something very similar on walls of their caves, yet at the same time it is powerful enough to be the chosen tool of expression for many of our greatest modern minds.

While the utility of these simple smooth white slabs is clear, there is an opportunity here to create a computing device that can provide all of these abilities and more–a computer that is as simple as sketching but as powerful as a modern desktop. Written equations can be captured and solved, sketches of state machines can be executed, and previous drawings can be queried, recalled, shared, and transformed.

These ideas are not just science fiction, in fact the HCI community has made significant inroads in this direction by carefully hand crafting solutions to very specific instances of these problems for everything from equation editors, to circuit diagrams, to simple mechanical systems. While these one-off solutions give us a glimpse of the future possibilities, a general purpose framework will be needed to allow the simple creation of new sketch based tools. In this paper we characterize the sketch workload that will be faced by such a framework, and we explore the the inner workings of a state of the art general purpose sketch recognition engine.

With the radical shifts in design philosophy taking place in computer architecture right now, it is now more important than ever that we try and identify, and even shepherd forward, the mass market workloads of the future so that we can continue to reap the benefits provided by broad adoption. Computer architects are forced to consider device and technology changes 5 to 15 years into the future, yet more often than not the unstated assumption is that the applications that will drive those markets will be essentially unchanged from today.

While sketch recognition is certainly not a fully mature field, it is now to the point that performance is one of the most significant bottlenecks facing the area, and in fact represents one of many computationally-intensive interfaces that has the potential to drive the economics of our field forward. Make no mistake, the projects that we have attempted to characterize are still not mature enough to be "product-ready" and performing detailed cache performance analysis or other traditional workload characterization techniques would be very premature. We instead present an argument that technological trends will make this an increasingly attractive workload as the field matures and characterize some of the long-term performance challenges that must be overcome for this workload to become a reality.

Specifically, we characterize the state of the art in the increasingly mature area of sketch recognition, we argue that market forces make this an increasingly attractive area to pursue, and we identify those areas where performance issues are the greatest detriment to the advancement of the field.

We begin with a look into the economics of display technology and uncover recent trends in display growth. In Section III we provide an overview of related works, describing recent research in the field of sketch recognition as well as computing environments designed for free-form input. In Section IV we attempt to characterize the input traffic generated in whiteboard-based environments. We end in Section V with an investigation into a new sketch recognition system for understanding circuit diagrams. We conclude with our predictions for the future demands of whiteboard computing.

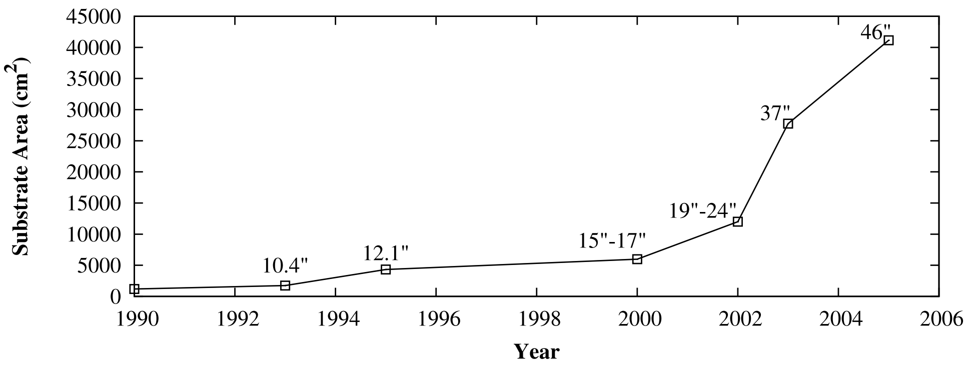

Fig. 1. This plot shows the growth in size of the most affordable LCD displays per square inch taken from published white papers [1] versus time. These devices are fabricated lithographically, and as processing technology has matured we have seen a drastic increase in both the size and affordability of large displays.

II. Motivation for Whiteboard Computers

Even in our modern computer-saturated lives, simple pen and paper sketching plays an important role in the complex tasks that we undertake. For example, it has been observed that sketches are a natural first step in the design of everything from circuits to mechanical devices [2], and that drawing in tandem with discussion greatly increases our ability to communicate complex subject matter [3]. It is perhaps curious then, that even with all of the computation power at our disposal, our first inclination when we need to design something complicated is to stand up and walk away from the computer.

A. The Economics of Display Technology

From laptops to cell phones, mobile devices have been pushing the limits of small-scale technology. So much so, in fact, that it might be easy to overlook a separate and very different trend that is occurring: the rapid increase in the size and affordability of large displays. As illustrated in Figure 1, these displays, constructed lithographically and carved from large pieces of "mother glass," are doubling in area every 1.5 years while prices have been simultaneously dropping. These exponential gains, coupled with the fact that flat panel displays larger than standard whiteboards are already in production, imply that it is only a matter of time before displays become economical enough to find their way into our offices, homes, and classrooms. Even today, DLP projectors are approaching the $500 mark, less than the cost of a 15-inch LCD monitor just 5 years ago. As the price of "human-scale" displays drop, coupled with affordable pen tracking hardware already available on the market, it will become economically viable to replace a traditional whiteboard with a digital screen. As these screens become common, radically different means of interaction become increasingly attractive.

B. Domain Specific Sketch Recognition

Of course large, cheap displays are in fact only part of the challenge. To successfully replace a whiteboard, the display must have an incredibly unobtrusive user interface–always ready for simple sketching, capable of making inferences about sketches, and able to act upon these inferences in real time. Over the last 10 years, hand built prototype systems for tablet PCs have been created for a variety of domain specific problems, and this first generation of prototypes is just now reaching a product-ready level of maturity.

C. Frameworks for Sketch Recognition

One of the biggest problems facing this line of research is the jump from a series of hand built application specific techniques to a general purpose framework that supports the creation and extension of novel graphical domain specific schemes. The context in which a sketch is created is a very large part of understanding how the sketch is to be viewed, yet all sketches are constructed in fundamentally the same way.

More so than in existing computing paradigms, users will not tolerate fine-grained commands or complicated interfaces [4]. Instead, the burden of disambiguating a user's intentions must be left to the machine. Each stroke generated by the user will likely contain tens to hundreds of points alone (Section IV quantifies whiteboard data collected from our lab). With display resolutions containing millions of pixels, a board full of drawings can result in a significant amount of data. The storage, query, and manipulation of these sketches (both the actual points and their associated meanings) quickly becomes difficult to reason about and manage. A scalable shared software architecture that can unify these different one-off projects is badly required.

III. Related Works

Whiteboard computing spans a wide variety of domains. As such, the works referenced in this section are by no means exhaustive. We have focused our selection to a concise set of recent developments pertaining to sketch recognition and large surface displays.

A. Tablet-Based Applications

Sketch based interaction presents a number of challenges to application developers. Interface questions that were once addressed by the now pervasive windows, mouse, and pointer model break down as sketching becomes the main mode of interaction [5]. Application designers must implement an

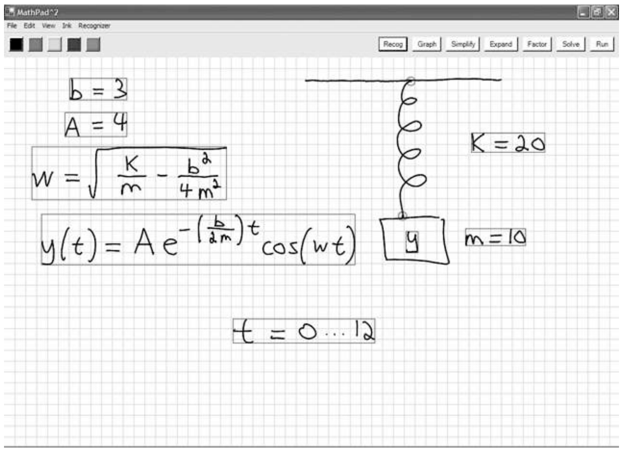

Fig. 2. MathPad^2 interface.

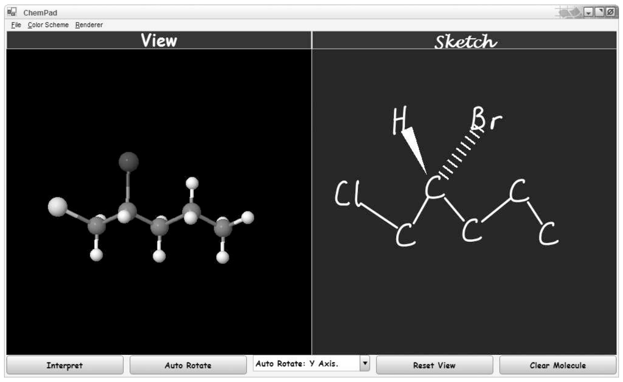

Fig. 3. ChemPad interface.

interface that lies between two extremes: basing input solely on the interpretation of free-form strokes, or restricting the user to only a limited set of predefined gestures. Here we describe a set of existing sketch based applications that each address the input problem with varying degrees of input freedom.

One recurring goal of the sketch community has been robust recognition of mathematical expressions [6], [7]. Recent research has resulted in MathPad^2, a tablet-based tool for combining mathematical sketches and simple, dynamic illustrations [8]. The MathPad^2 interface is predominantly modeless, using a combination of free-form input and high-level gestures to determine the flow of control. Expressions, variables, and numeric values are inferred from a user's sketch, while actions over the sketch data, such as recognition, deletion, and grouping are invoked by gestures. The high-level gestures delineate the actions to be taken as well as the region over which an action is to be performed. A typical usage scenario involves inputting a formula, where the user must first draw an equation and then encircle and tap the region that is to be recognized. An example of the interface can be seen in Figure 2.

In MathPad^2, the full recognition of user input happens over multiple stages [8]. The first stage involves feature extraction, where strokes are normalized and statistical features are extracted. The second and third stages employ statistical analysis, refining the classification of each stroke into its appropriate mathematical symbol. Once classification is complete, a final recognition stage is required to interpret the geometric arrangement of the symbols in order to discover superscripts, fractions, and similar features.

A slightly more restricted form of sketch based interaction has recently found applications in Organic Chemistry in the form of ChemPad, a tablet-based tool for converting 2D molecular sketches into their corresponding 3D representations. ChemPad presents the user with a split view as shown in Figure 3, displaying both the 3D and 2D molecular models [9]. The user is required to generate a 2D sketch through a sequence of single-stroke gestures. Included in the set of gestures is the ability to delete existing input, permitting the user to change only a portion of a given sketch. Once the sketch is complete, the user can invoke recognition of the 2D model and view its 3D counterpart.

ChemPad interprets free-form input on a per stroke basis. Recognition is achieved through a modification of [10], where a highly specific set of rules are applied to differentiate between gestures. One result of this approach is that gestures may require context; therefore, operations such as delete, represented by a scribble) may be applied over entire regions of existing user input.

Both MathPad^2 and ChemPad serve as a good guide and warning for future sketch application systems. Within this field, there is significant freedom to design novel methods of interaction, but this freedom comes at a cost. Designers of both MathPad^2 and ChemPad had to spend significant time and effort building each domain specific recognizer.

SketchREAD is perhaps the most notable attempt at performing recognition across multiple domains. Developed prior to both MathPad^2 and ChemPad, SketchREAD is a framework for diagrammatic sketch recognition. The framework utilizes dynamically constructed Bayesian networks to interpret unconstrained drawings, permitting users to draw objects without respect to a specific stroke ordering. While SketchREAD provides promising system performance results, accuracy remains a limiting factor [11].

B. Larger Surfaces

A new market for devices slightly larger than traditional computer displays appears to be emerging, as recent advances in multi-touch technology is becoming increasingly popular [12], [13]. As these devices are capable of detecting multiple points of contact at any given time, they can be viewed as a generalization of the existing pen and tablet interfaces.

The appeal of a larger interactive surface has attracted research in the past. Xerox's LiveBoard targeted the office setting where workgroup meetings and presentations are commonplace. Using a pen-based interface and specialized applications, the LiveBoard set an early precedent for pen-based

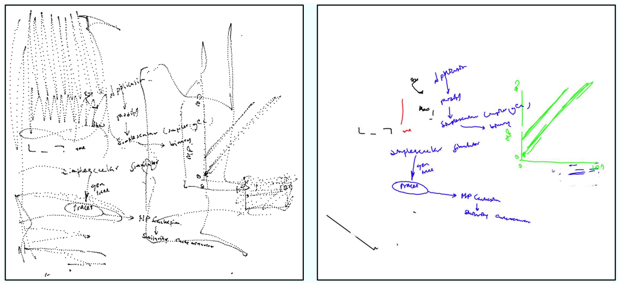

Fig. 4. A sample collected from an eBeam board capture device during the week of January 20, 2008. The left and right images reflect identical data. The image on the left displays only the raw point values, while the image on the right uses available metadata to mimic the current state of the board.

user interface design. Produced in 1994, with a sixty-seven-inch display and retail value between $40,000 to $50,000, the success of the LiveBoard ultimately succumbed to financial loss due to cost, supply, and quality issues [14]. Some of the features that made the LiveBoard so attractive also received the harshest critiques. While the display had a relatively large resolution for its time (1120 x 720 pixels), users felt the image quality needed to be better. While the optical pen, capable of capturing 90 x-y coordinate pairs per second, was a novel feature, the feel and accuracy of the pen were disliked. Lastly, earlier models of LiveBoard used monochrome displays, yet there was a demand for color [4].

By today's standards, the user demands for the LiveBoard seem underwhelming. High-resolution, board-sized color displays are readily available, and high resolution tracking devices are available through a number of vendors. Research has led to results far beyond typical board sizes, including an 18 x 8 foot wall-sized display with a native resolution of 6144 x 3072 pixels [15].

IV. Characterization of Whiteboard Traffic

The applications we have discussed thus far have all been primarily aimed at tablet devices. Considering the recent advancements in display technology and the increased interest in multi-touch interfaces, it's not hard to imagine these applications being used in a broader context within larger, pen-based or touch-based environments. However, to our knowledge there has not been any significant research aimed at understanding how well these algorithms will scale and behave across different mediums. Factors such as display resolution, input resolution, and the kinematics involved with the input process each have the potential to affect the accuracy of the recognition process. As we are interested in targeting whiteboard-sized devices, we have begun observing how free-form input arrives on the whiteboard.

The mention of free-form input generally implies data in the form of two-dimensional points in time. However, a significant amount of metadata can accompany each stroke and possibly each point. Physical attributes such as pressure, size, and color, as well as functional attributes, such as erasure, are just some of the metadata properties that could be included with raw stroke data. To date, very few of these attributes are considered by even the most advanced recognition techniques; many solutions are only concerned with the spatial and temporal data.

To understand more fully the order of magnitude of data and timing involved in such a whiteboard system, using an eBeam capture device [16], we have collected roughly four months worth of input on our laboratory whiteboard. Our experiment does no recognition, instead it passively[1] records statistics from the data points that are naturally written by our lab for computer architecture research so that we can understand the distribution of inputs that might be given to recognition systems. Beyond spatial coordinates, the raw data captured includes timing information, marker color, and marker size. The input radius of each drawing device was used to accurately account for the intersection and erasure of existing strokes. Figure 4 presents an example of the data captured and generated by the whiteboard system.

Over the survey period, approximately 10-15 lab members had access to the whiteboard. The sample period, spanning from November 19, 2007 to March 17, 2008, includes portions of Winter and Spring quarter, including a significant holiday break in between. Of the 120 days of the survey, 49 days (approximately 41%) contained board activity.

The combined days of board activity produced over 300,000 input strokes. An input stroke is generated every time an input pen touches the whiteboard surface. A stroke accumulates all point data captured until the pen is lifted from the surface of the whiteboard. It is assumed that only one pen is drawing on the whiteboard surface throughout the duration of any given stroke. Our entire data collection consists of strokes composed of up to hundreds of points each.

As a means of deciphering and reviewing past board data, we created a variety of tools for automatically segmenting and replaying stroke data. As highlighted in Figure 4, erasure metadata is useful in depicting the visual state of the board as perceived by the user. A review of the data shows a wide variety of drawings. Since the lab board is a communal writing place, there are many active ideas on the board at any given point in time and they are spatially partitioned across the board. In the same way that the inputs are clustered in space, the strokes are clustered into bursts of time as well. Research described in [17] has shown that temporal and spatial clues are often not enough to segregate groups of strokes into their relevant symbols (e.g., circuits and gates); however, our data shows promising results for accurately segmenting strokes at a larger granularity.[2]

To understand and quantify the nature of the board traffic, we have dissected the input into drawing and erasure strokes.

[1] Users continued to use traditional dry erase markers. Electronic pen holders were placed around the original markers to record pen movement. An electronic eraser, capable of removing the dry erase markings, was used to record all erasures.

[2] Although we do not discuss all of our methods for stroke segmentation in detail, it is worth noting that segmentation based on inter-stroke timing delays and segmentation based strictly on batched erasures both provided visually effective results.

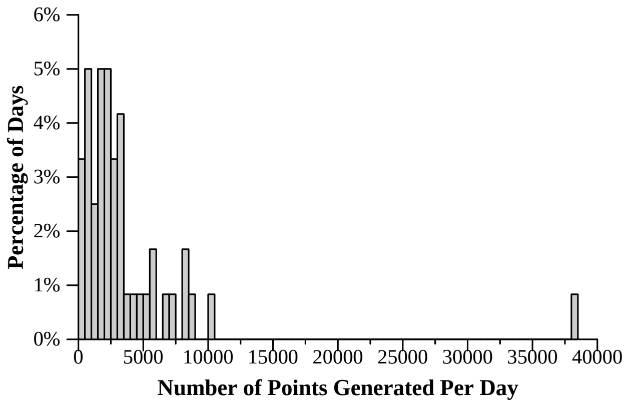

Fig. 5. Number of points generated on a daily basis during four months of whiteboard usage. Over this collection period, over 300,000 points were generated. Days without activity, approximately 59% of the total collection, are not shown.

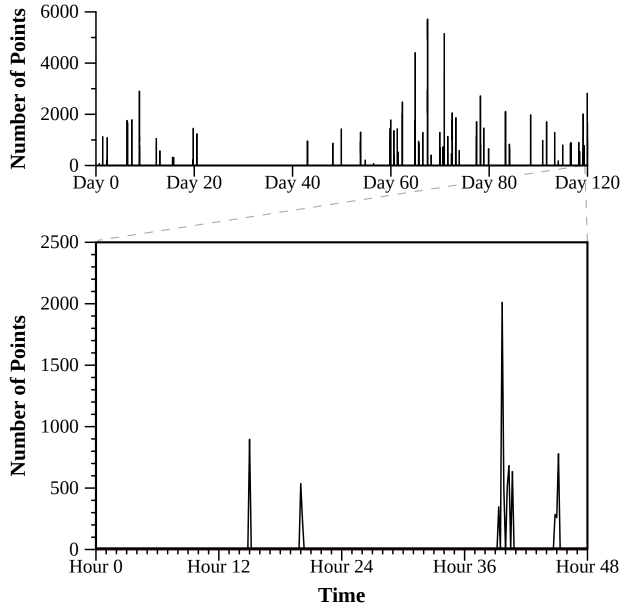

Fig. 6. Number of points generated over ten-minute intervals on a daily basis during four months of whiteboard usage. Over the 120 days of data collection, bursts of board traffic were observed. A review of daily board traffic reveals sessions of board activity are frequently followed by hours of inactivity.

As shown in Figure 5, with only one exception, all days in which drawing input occurred generated less than 11,000 points. The largest percentage of daily input generated less than 5,500 points. Investigating the creation of points over time shows an extremely bursty traffic pattern. Figure 6 shows the number of points generated over ten minute intervals throughout the lifetime of the survey. After reviewing the individual days of collection, it was easy to segment the drawings into board sessions, or periods of time when users were at the board working on a specific problem. While it remains unclear how useful the session metric will be to system design, it's conceivable that sessions could help identify a lower and upper performance bound. For example,

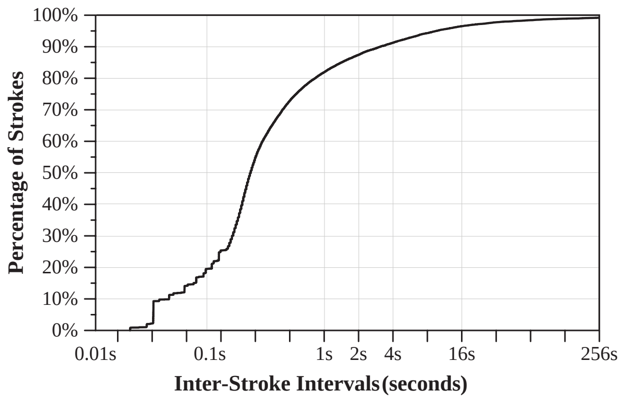

Fig. 7. Cumulative distribution of the inter-stroke delay between drawing (non-erasure) strokes across all sessions. Approximately 20% of drawing strokes occur within one-tenth of a second of the preceding drawing stroke.

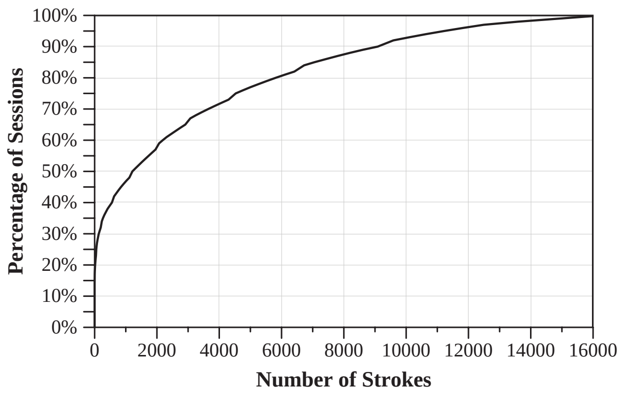

Fig. 8. Cumulative distribution of drawing (non-erasure) strokes per session. Strokes are segmented into sessions based on timing data; inter-stroke delays larger than 30 minutes define a new session.

a session could indicate how much working memory will be required to solve the typical problems faced when interacting with the whiteboard.

A detailed look into inter-stroke arrival times is provided in Figure 7. Since most drawing strokes occur in rapid succession, partitioning the recorded data into sessions in which the inter-stroke delay between two consecutive strokes exceeded 30 minutes, produced an accurate history of board snapshots. We captured approximately 90 unique periods of user activity. Figure 8 describes the number of drawing strokes generated throughout all individual sessions. The trend in the graph indicates that a nontrivial percentage the sessions do not contribute new drawing data and only a small percentage of the sessions are responsible for large drawing contributions. Perhaps not surprisingly, the results shown in Figure 9 indicate that the total area of the board utilized by a session behaves similarly to the number of strokes per session. The relationship between the two figures would suggest that new strokes often do not overlap with existing strokes, or they at least do so in a manner that expands the bounding box housing the current set of strokes.

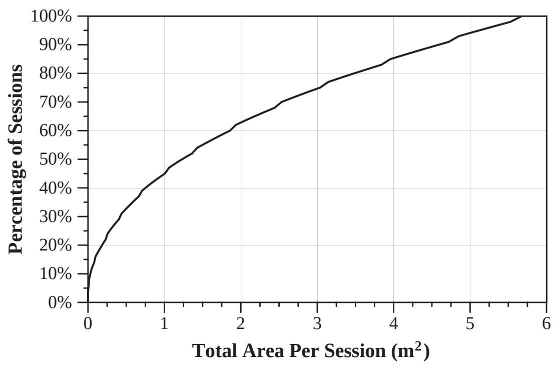

Fig. 9. Cumulative distribution of the square area of drawing (non-erasure) strokes per session. Strokes are segmented into sessions based on timing data; inter-stroke delays larger than 30 minutes define a new session. The area of a session is computed by finding the minimum bounding box containing all strokes within the session. The total size of the whiteboard is approximately 6 meters.

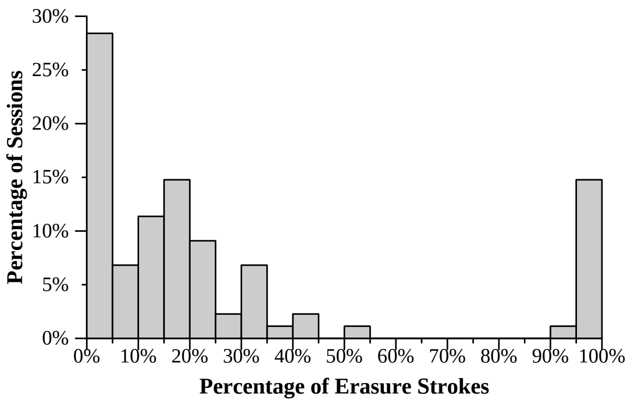

Fig. 11. Percentage of erasure (non-drawing) strokes across sessions. Strokes are segmented into sessions based on timing data; inter-stroke delays larger than 30 minutes define a new session. While most sessions are dominated by drawing strokes, roughly 15% of all sessions are exclusively composed of erasures.

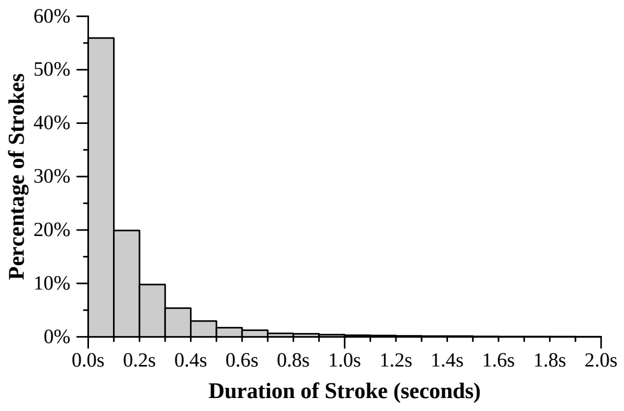

Fig. 10. Duration of drawing (non-erasure) strokes across all sessions. While a small percentage of strokes exceeded two seconds, nearly all detected strokes lasted less than one second. The longest duration discovered reached 8.8 seconds.

TABLE I
NUMBER OF CORNERS PRESENT IN FRAGMENTED DRAWING STROKES
STROKE FRAGMENTATION AND CORNER DETECTION PERFORMED USING SPEED CALCULATIONS AS PRESENTED IN [18]

# Corners	**0**	**1**	**2+**
# Segments (%)	15291 (93.9%)	579 (3.6%)	407 (2.5%)

The data collection process was designed to be as non-intrusive as possible, with no attempts at making whiteboard interaction more convenient for the user. For example, if a user needed to erase the entire board surface, he or she had to do so manually. Therefore the trends recorded are true to the normal trends of our whiteboard.

One trend that stands out is how quickly strokes are drawn. Figure 10 provides a histogram outlining the duration of drawing strokes. Over 55% of the drawing strokes were drawn in under one-tenth of a second. In fact, almost all drawing strokes last less than a full second.

A second important trend to follow is erasure behavior. Existing sketch recognition algorithms tend to acknowledge and then ignore the potential for erasures, so our ability to capture erasure data was of particular interest. In the end, we observed that a large amount of time was spent erasing data. Figure 11 highlights the percentage of erasure strokes that occurred per board session. An important observation is that while some sessions were entirely dedicated to erasing the board–usually when clearing the entire board surface–many drawing sessions are interspersed with erasures. Given the complexity of sketch recognition, it's understandable why the issue of erasing is avoided, yet our data indicates that it's such a frequent task that the role erasing will play in any sketch system is significant. We believe this feature should not be neglected as it has the potential to add immeasurable value to the system.

One final observed trend, that is also highly relevant to modern sketch recognition systems, is the evaluation of the complexity of each individual stroke. While there are a number of applicable metrics, a simple estimate of complexity can be obtained by fragmenting the stroke into approximate pieces, such as lines and curves. Neglecting curves, past work has demonstrated how the speed of a drawing can be used to quickly detect a stroke's corners [18]. Using this method, it is possible to get a feel for the complexity of the board data. Table I provides a breakdown of the number of corners detected within all collected drawing strokes. The results indicate that over 90% of strokes do not contain corners. This is likely an indication of two separate issues. First, users are mostly likely lifting the pen and creating new straight-lined strokes in places where corners exist (e.g., drawing a box using four unique strokes in place of one continuous motion). Second, many strokes that have potential corners are likely gentle curves. The first case is ideal, since the user is essentially performing the fragmentation by hand. The second case, however, is more problematic since it can be difficult to place a threshold to specify which curves represent corners. Upon visual inspection of the board sessions, it is apparent that text likely accounts for a large percentage of the strokes without corners. While many pieces of input data clearly contain corners, most of the input that is not text is frequently accompanied by textual labels.

These data above are not meant to capture the workload of a specific application, and indeed if the whiteboard was an interactive device it is likely that some of these distributions will change. However, we can still take a great deal away from this characterization: the workload is exceedingly bursty, strokes are usually fairly simple yet they come in rapid succession, and an average session will deal with hundreds or thousands of strokes spread over several square meters. With these observations in mind, we shift our focus to a modern tablet-based sketch recognition system that is currently undergoing development.

V. Characterization of Recognition Engine

The best-known algorithms for parsing free-form drawings rely on a tight hypothesize-model-measure loop that places an extraordinarily heavy demand on the system. Each reasonable hypothesis must be tested in isolation so that the most likely interpretation of the input can be discovered. This tight loop is the critical bottleneck. To understand exactly what it does and how it might map onto the machines of the future, in this section we describe these steps in more detail.

There are many levels on which we could characterize the workload of sketch based systems. Irrespective of the ability of the underlying system to quickly recognize and understand free-form input, from the user's perspective, the interface contributes significantly to the perceived responsiveness of the system. The designers of sketch systems must carefully choose when and how recognition is triggered. Furthermore, even after recognition takes place, the system may or may not want to provide immediate feedback to the user. Especially when free-form input is the main mode of interaction, there appears to be a delicate balance between assisting the user with too much or too little feedback. While it may seem counterintuitive, less responsive feedback may result in an overall more pleasing form of interaction [19], [20].

Giving the user the ability to write freely lies at the heart of these tablet-based applications. Ultimately, the interface should disappear and allow the user to freely express his or her ideas without concern for the underlying tool's ability to comprehend the input. However, there are a number of factors that prevent this goal from becoming a reality. Questions about user interface designs aside, giving the user complete freedom to draw in any manner that he or she chooses presents an extraordinary computational challenge. In this section we explain why this problem is computationally challenging and discuss the best known approaches for interpreting free-form input.

Like much other software still under research and development, it is impossible to find solid and openly available codes to characterize and run. Indeed the work we study here is right out of the research lab and is far from ready for distribution. However, working with experts in the area we have been trying to understand the most performance critical sections in this research code. In talking with many of these experts it became clear to us that this is fundamentally a performance-limited problem, and in this section we describe the software architecture and the opportunities for performance improvements to increase recognition power and accuracy.

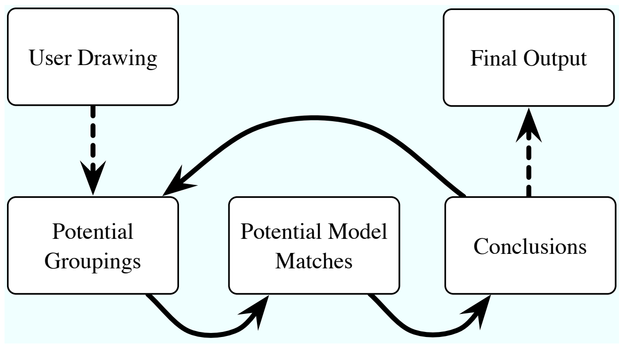

Fig. 12. The abstract sketch recognition loop. The dashed lines represent input and output to and from the system, forming an implicit read, eval, display cycle.

To better understand the performance demands of the sketch recognition process, we have evaluated a system, currently under development at Harvey Mudd, capable of interpreting circuit diagrams [21]. We believe this application represents the state of the art in sketch recognition techniques and the lessons we can take away from an analysis of this approach will generalize when developing similar and larger scale systems.

At a high level, sketch recognition involves multiple stages that must refine raw stroke data into manageable groupings that ultimately match, or fail to match, a set of templates defined by the application. Figure 12 depicts the general sketch recognition loop. It is important to note the dependency each stage has on the previous stages. Errors that occur early in the recognition process propagate and can easily become detrimental to the recognizer's ability to find suitable matches. While it may seem obvious for systems that receive continuous updates, such as new strokes drawn by the user, to consist of an input loop, that is not the purpose of the loop expressed in the figure; the input recognition and user feedback loop is implicit and not shown. The loop outlined in Figure 12 exists to combat the propagation of errors. Since it quickly becomes infeasible to test every possible grouping of every stroke generated by the user, sophisticated techniques must be used to guess which strokes should be combined into formal groups. It is possible that a small set of groupings work well within a local setting, but fail in a global context. It is therefore crucial that the conclusions being drawn actively inform the grouper so that mistakes can be detected and corrected as early as possible.

The developers of the circuit recognizer have created a testing framework dubbed "TestRig" that gives the tester the freedom to define and insert any number of stages in the recognition pipeline (See: Figure 13). For our purposes, we

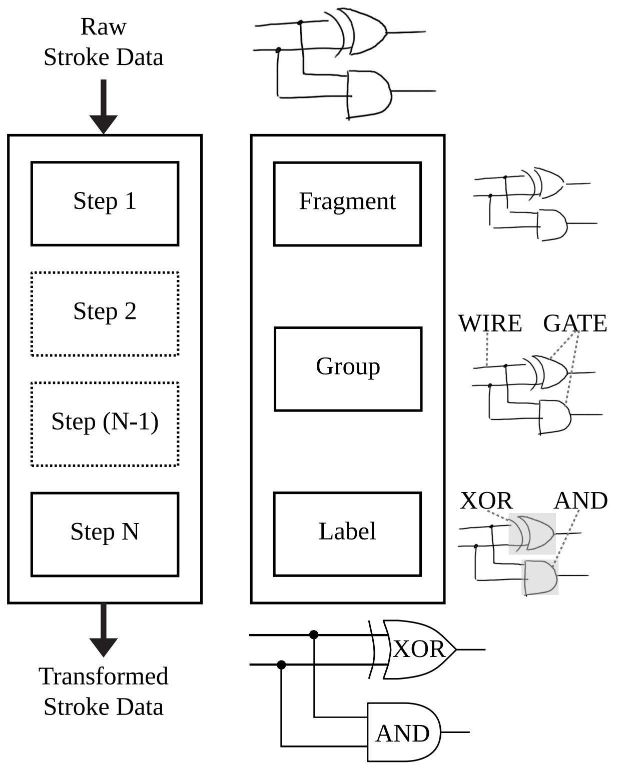

Fig. 13. Harvey Mudd **TestRig** design (Left) and performance evaluation configuration (Right). With TestRig's pipelined design, any number of stages can be connected in order to view and modify the raw stroke data. To model the current state of the circuit recognizer, three stages are required: Fragmentation, Grouping, and Labeling.

have focused on the three significant phases outlined in [21].

The initial input to TestRig consists of raw stroke data. In this recognition sequence, we are only concerned with the two-dimensional points and timing data. The first stage evaluates each stroke and attempts to fragment complex strokes into multiple strokes. Fragmentation is accomplished by analyzing the speed and curvature of a given stroke and locating points with both high curvature and low speed. Former work has shown this approach to be effective at discovering corners [18]. Because users draw gates and other symbols with a variable number of strokes, it is important to brake these "compound" lines apart into their fundamental units. Consider, for example, how one might draw a square–one continuous stroke, two L-shaped strokes, one line and one concave segment, etc. Once fragmentation is complete, the transformed data is sent to a preclassification stage.

The second stage attempts to label the individual strokes as either wires or gates. Classification decisions are based on a pre-trained, conditional random field (CRF) [22]. Similar to a hidden Markov model, the CRF is a probabilistic, graphical model that is effective at labeling and segmenting structured data. Details on the application of conditional random fields to circuit diagrams can be found in [21].

The third and final stage performs a significant amount of computation. At this point, the stroke data has been preclassified and must be grouped; ideally, strokes belonging to the same gate or wire are collected into one shape. There are many potential strategies for performing grouping at this level. A naive approach could try to group every possible combination of every stroke; however, this quickly becomes impractical for even small circuit diagrams. One of the benefits of the preclassification stage is that it can significantly reduce the search space of this final grouping stage. The grouper uses this information to match labeled data that is close in both space and time. A support vector machine (SVM) [23] is used to analyze the grouped gates and determine their final classification.

As this project is still in development, conclusive performance benchmarks are difficult to obtain, but our preliminary results indicate that the majority of the processing overhead occurs in the initial recognition stage. The current performance breakdown, shown in Figure 14, indicates that the total processing time for one pass over relatively basic circuit diagrams–composed of fewer than 100 strokes–requires seconds to complete. The timing data indicates that the labeling stage introduces significantly more overhead than the grouping stage. This result seems reasonable since the grouping stage should only be analyzing a subset of the original data while grouping fragments. A key, and possibly unsettling, observation is that the number of fragments present in a drawing does not appear to consistently affect the the recognizer's performance. In order to understand why this is the case, we must understand what occurs throughout the labeling stage.

The labeling stage is essentially a generalization of the methods presented in [24]; a CRF is constructed and used to classify the fragmented stroke data as either a wire or a gate. The number of nodes created for the CRF is set to the number of fragments in the sketch. Edges are created between nodes that are close in both spatial and temporal proximity. Next, a number of attributes, such as geometric distance, timing data, and speed, must be calculated for both nodes and edges[3]. Once all attributes have been calculated, classification can begin. The graphs created for circuit recognition are dense and therefore employ a loopy belief propagation algorithm to perform the inference.

Profiling TestRig with the data presented in Figure 14 clearly flags the loopy belief inference step as the main performance bottleneck. The relative uncertainty of how fast the algorithm will converge is conveyed by the timing data. The CRF's complexity is a function of not only the number of fragments, but also the temporal ordering and placement of strokes.

As previously noted, successive stages are dependent on the stages that precede them. The performance of the grouping stage is therefore largely dependent on the success of the labeler. The current grouping implementation evaluates every

[3]In total, 17 attributes are used in the current implementation; 9 attributes are used for the nodes, while 8 are used for the edges. Attributes described in [24] have been observed to work empirically; however, choosing the best set of attributes remains an open problem.

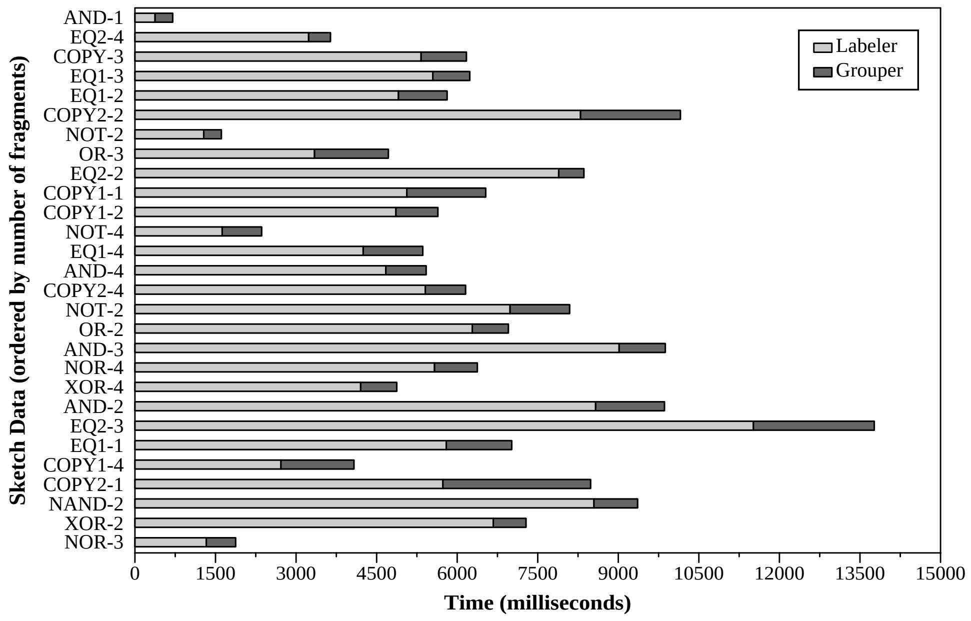

Fig. 14. Performance of circuit recognition phases based on time to perform each task. Data along the y-axis has been sorted based on the number of fragments (ranging from 25 to 60) within each drawing. Data in this evaluation was previously fragmented, so the fragmentation stage has been omitted. Tests performed over similar raw data indicate the fragmentation stage operates an order of magnitude faster than the labeling and grouping stages.

combination of the set of strokes it is passed, yet profiling results seem promising. The time spent by the grouper is largely dominated by the recombining of strokes; the support vector machine proves to be very efficient and does not currently appear to be a source of overhead.

In reviewing TestRig's performance, one question of particular interest stands out: how much of a tradeoff can be made between performance and accuracy? While the complexity of loopy belief propagation might at first seem unacceptable, the use of conditional random fields may actually provide a means for trading performance for accuracy and vice versa. Since the graphs of the CRF are composed at runtime and the attributes defined over the nodes and edges can be arbitrarily complex, it may be possible to tweak these values in order to achieve the desired balance between performance and accuracy dynamically. The ability to make this tradeoff could be invaluable when dealing with data covering surfaces as large as a whiteboard. One could, for example, target specific regions of a sketch for higher accuracy while targeting other regions for high performance.

At first glance, the results presented may appear to be too slow to function in real-time–even more so when you consider that in a fully developed system certain stages will be repeated multiple times before converging on a final output. Even though the processing time appears high, it is important to remember that we are interpreting batched data. Working with dynamic content, the application has the ability to identify individual strokes as they are drawn. Furthermore, it seems likely that users may not necessarily require immediate feedback [19]. Exploiting both of these properties could easily help amortize the cost of the recognition processing overhead.

Still today, handling sketch input poses a significant challenge to many areas of research, including HCI, recognition, and performance. It seems the interesting solutions lie at the intersection of all three goals, and there is a need for improvement in each before free-form sketch recognition becomes a reality.

VI. Conclusions

A computational whiteboard, one that can automatically identify user input and interpret free-form strokes in real time, is a challenging problem, but one that is within our grasp as computer systems developers. Waiting passively for new workloads to find their way into the mainstream and only then characterizing and developing architectures and systems software around them is not something that we have the luxury of doing in these turbulent times. In this paper we have provided a preliminary look into the data and application challenges facing whiteboard computing.

Of course this is not a problem for systems architects alone, but it is one where significant systems expertise is sorely required today. More so than in existing computing paradigms, users will not tolerate fine-grained commands or complicated interfaces. Instead, the burden of disambiguating a user's intentions must be left to the machine. Systems must be designed with imprecision as a first class citizen as the errors inherent to free-form sketch input are enormous. The best-known algorithms for parsing free-form drawings rely on a tight hypothesize-model-measure loop that places an extraordinarily heavy demand on the system. Each reasonable hypothesis must be tested in isolation so that the most likely interpretation of the input can be discovered. New error-conscious parallel primitives could potentially make these steps operational in real time. Novel tradeoffs that play between system perceptiveness, human participation, and computational efficiency may be possible in this new domain [25].

Following this line of reasoning, the interface itself presents some interesting system research opportunities. Managing all of the figures and drawings on the board in both time and space is a very tricky problem. A quick look at a typical white board from our laboratory shows that it is covered with everything from lists and announcements to precise diagrams, sketches used during conversations, even amusing doodles. Appropriately grouping these sketches, moving them around the board as new and old sketches become active, and searching through prior sketches are effectively open problems to different extents in the HCI community. Each stroke generated by the user will likely contain tens to hundreds of points alone. With display resolutions reaching into the millions of pixels, a board full of drawings can result in a significant amount of data. The storage, query, and manipulation of these sketches (both the actual points and their associated meanings) quickly becomes difficult to reason about and manage. A scalable shared software architecture that can unify these different one-off projects is badly required. Through inference or explicit user action, techniques for sketching across domain barriers must be developed. Programming languages have the potential to play a double role in this space. Not only will new tools be required to construct digital whiteboard systems, a whiteboard environment could facilitate completely new approaches to visual language design. It remains unclear what such a whiteboard language would look like, or even if one extensible enough to allow the creation of new whiteboard tools, yet practical enough for everyday use, is even possible.

While there are many challenges to realizing such a vision, ultimately we desire a system that can compute upon free-form input from any number of domains without sacrificing the freedom provided by a conventional whiteboard. We believe this presents a number of new challenges to architect, system, and language designers alike and that we need to take an active role is defining new ways for this work to become a reality.

Acknowledgments

The authors would like to thank Luis Ceze (UW), Art Muir (Luidia), and the anonymous reviewers for their extremely helpful comments regarding this work. We would also like to thank Christine Alvarado (Harvey Mudd) for providing access to her group's code and for the numerous helpful discussions we have had. This work was funded in part by NSF Career Grant CCF-0448654 with hardware donations from Luidia.

References

[1] AKT, "AKT Large Area PECVD Capability," 2005. [Online]. Available: http://www.appliedmaterials.com

[2] C. Alvarado, "A natural sketching environment: Bringing the computer into early stages of mechanical design," Master's thesis, MIT, 2000.

[3] A. Adler and R. Davis, "Speech and sketching for multimodal design," in *IUI '04: Proceedings of the 9th international conference on Intelligent user interfaces*. New York, NY, USA: ACM, 2004, pp. 214–216.

[4] S. Elrod, R. Bruce, R. Gold, D. Goldberg, F. Halasz, W. Janssen, D. Lee, K. McCall, E. Pedersen, K. Pier, J. Tang, , and B. Welch, "Liveboard: A large interactive display supporting group meetings, presentations and remote collaboration," in *Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems*, 1992, pp. 599–607.

[5] G. Apitz and F. Guimbretière, "CrossY: a crossing-based drawing application," vol. 24, no. 3, pp. 930–930, July 2005.

[6] R. Helm, K. Marruitt, and M. Odersky, "Building visual language parsers," in *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM Press, 1991, pp. 105–112.

[7] S. Smithies, K. Novins, and J. Arvo, "A handwriting-based equation editor," in *Graphics Interface*, 1999, pp. 84–91. [Online]. Available: citeseer.ist.psu.edu/smithies99handwritingbased.html

[8] J. J. Laviola and R. C. Zeleznik, "Mathpad2: a system for the creation and exploration of mathematical sketches," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 432–440, August 2004. [Online]. Available: http://portal.acm.org/citation.cfm?id=1015741

[9] D. Tenneson and S. Becker, "Chempad: generating 3d molecules from 2d sketches," in *SIGGRAPH '05: ACM SIGGRAPH 2005 Posters*. New York, NY, USA: ACM, 2005, p. 87.

[10] R. Zeleznik and T. Miller, "Fluid inking: augmenting the medium of free-form inking with gestures," in *GI '06: Proceedings of Graphics Interface 2006*. Toronto, Ont., Canada, Canada: Canadian Information Processing Society, 2006, pp. 155–162.

[11] C. Alvarado and R. Davis, "Sketchread: A multi-domain sketch recognition engine," in *Proceedings of UIST 2004*. New York, New York: ACM Press, October 24-27 2004, pp. 23–32.

[12] J. Y. Han, "Low-cost multi-touch sensing through frustrated total internal reflection," in *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*. New York, NY, USA: ACM, 2005, pp. 115–118.

[13] Microsoft, "Microsoft Surface - Surface Computing," 2008. [Online]. Available: http://www.microsoft.com/surface

[14] H. Chesbrough, "Graceful Exits and Missed Opportunities: Xerox's Management of its Technology Spin-off Organizations," vol. 76, no. 4, pp. 803–837, 2002.

[15] G. Wallace, O. J. Anshus, P. Bi, H. Chen, Y. Chen, D. Clark, P. Cook, A. Finkelstein, T. Funkhouser, A. Gupta, M. Hibbs, K. Li, Z. Liu, R. Samanta, R. Sukthankar, and O. Troyanskaya, "Tools and applications for large-scale display walls," *IEEE Comput. Graph. Appl.*, vol. 25, no. 4, pp. 24–33, 2005.

[16] Luidia, "eBeam - Interactive Whiteboard Technology," 2008. [Online]. Available: http://www.e-beam.com

[17] C. Alvarado and M. Lazzareschi, "Properties of real-world digital logic diagrams," in *Proceedings of 1st International Workshop on Pen-based Learning Technologies*, 2007.

[18] T. M. Sezgin, T. Stahovich, and R. Davis, "Sketch based interfaces: Early processing for sketch understanding," in *Workshop on Perceptive User Interfaces, Orlando FL*, 2001.

[19] J. Hong, J. Landay, A. Long, and J. Mankoff, "Sketch recognizers from the enduser 's, the designer's, and the programmer's perspective," 2002. [Online]. Available: citeseer.ist.psu.edu/article/hong02sketch.html

[20] P. Wais, A. Wolin, and C. Alvarado, "Designing a sketch recognition front-end: User perception of interface elements," in *EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling*, 2007.

[21] C. Alvarado, "Sketch recognition for digital circuit design in the classroom," in *2007 Invited Workshop on Pen-Centric Computing Research*, March 2007.

[22] J. Lafferty, A. McCallum, and F. Pereira, "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data," in *Proc. 18th International Conf. on Machine Learning*. Morgan Kaufmann, San Francisco, CA, 2001, pp. 282–289. [Online]. Available: citeseer.ist.psu.edu/lafferty01conditional.html

[23] C. Chang and C. Lin, "Libsvm: a library for support vector machines," 2001. [Online]. Available: citeseer.ist.psu.edu/chang01libsvm.html

[24] M. Szummer and Y. Qi, "Contextual recognition of hand-drawn diagrams with conditional random fields," in *IWFHR '04: Proceedings of the Ninth International Workshop on Frontiers in Handwriting Recognition*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 32–37.

[25] P. A. Dinda, G. Memik, R. P. Dick, B. Lin, A. Mallik, A. Gupta, and S. Rossoff, "The user in experimental computer systems research," in *ExpCS '07: Proceedings of the 2007 workshop on Experimental computer science*. New York, NY, USA: ACM, 2007, p. 10.