Recognizing Freeform Digital Ink Annotations

Michael Shilman[1], Zile Wei[2]

[1] Microsoft Research, One Microsoft Way, Redmond, WA 98102 USA
shilman@microsoft.com
[2] EECS Department, UC Berkeley, Berkeley, CA 94720 USA
zile@eecs.berkeley.edu

Abstract. Annotations on digital documents have clear advantages over annotations on paper. They can be archived, shared, searched, and easily manipulated. Freeform digital ink annotations add the flexibility and natural expressiveness of pen and paper, but sacrifice some of the structure inherent to annotations created with mouse and keyboard. For instance, current ink annotation systems do not anchor the ink so that it can be logically reflowed as the document is resized or edited. If digital ink annotations do not reflow to keep up with the portions of the document they are annotating, the ink can become meaningless or even misleading. In this paper, we describe an approach to recognizing digital ink annotations to infer this structure, restoring the strengths of more structured digital annotations to a preferable freeform medium. Our solution is easily extensible to support new annotation types and allows us to efficiently resolve ambiguities between different annotation elements in real-time.

Introduction

While the vision of the paperless office remains a distant hope, many technologies including high-resolution displays, advances in digital typography, and the rapid proliferation of networked information systems are contributing to a better electronic reading experience for users. One important area of enabling research is digital document annotation. Digital annotations persist across document versions and can be easily searched, shared, and analyzed in ways that paper annotations cannot.

Many digital annotation systems employ a user interface in which the user selects a portion of the document and a post-it-like annotation object is anchored at that point, as shown in Fig 1(a). The user enters text into the post-it by typing on the keyboard. Later, as the document is edited, the post-it reflows with the anchor. While this method is widely used among commercial applications, it is a cumbersome user interface. Consequently, many users choose to print out their documents and mark them up with a pen on paper, losing the benefits of digital annotations in the process.

A user interface in which users sketch their annotations in freeform digital ink (Fig 1(b)) on a tablet-like reading appliance (Fig 1(c)) overcomes some of these limitations. By mimicking the form and feel of paper on a computer, this method streamlines the user interface and allows the user to focus on the reading task. For instance, in describing their xLibris system, Schilit *et al* introduce the term active

reading, a form of reading in which critical thinking, learning, and synthesis of the material results in document annotation and note-taking. By allowing users to mark directly on the page they add "convenience, immersion in the document context, and visual search." [1]

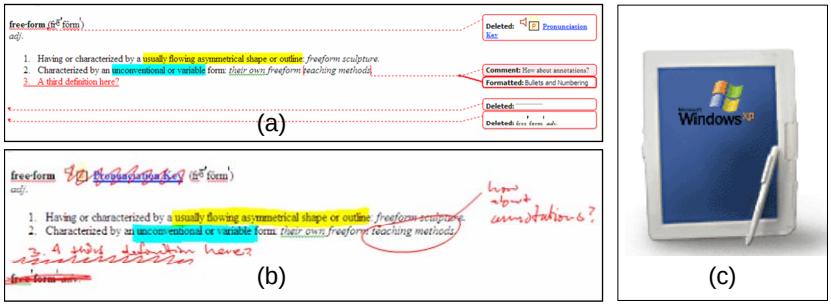

Fig. 1. (a) Digital text annotated and edited with "formal" annotations, and (b) equivalently with informal, freeform annotations. (c) A tablet-like computer with pen to annotate documents with digital ink.

Ink Annotation Recognition

In this paper, we describe a technique for recognizing freeform digital ink annotations created using a paper-like annotation interface on a Tablet PC. Annotation recognition includes grouping digital ink strokes into annotations, classifying annotations into one of a number of types, and anchoring those annotations to an appropriate portion of the underlying document. For example, a line drawn under several words of text might be classified as an underline and anchored to the words it is underlining. We describe the full set of annotation types and anchoring relationships we wish to support in the System Design section.

There are several reasons why we wish to recognize digital ink annotations, including annotation reflow, automatic beautification, and attributing the ink with actionable editing behaviors.

Our primary goal is to reflow digital ink, as shown in Fig 2(a) and (b). Unlike their physical counterparts, digital documents are editable and viewable on different devices. Consequently the document layout may change. If digital ink annotations do not reflow to keep up with the portions of the document they are annotating, the ink can become meaningless or even misleading. Recognizing, anchoring, and reflowing digital ink annotations can avoid this disastrous outcome. Golovchinsky and Denoue first observed this problem in [2], but we have observed that the simple heuristics they report are not robust to a large number of real-world annotations and they do not propose a framework in which to incorporate new types of annotations.

A second goal of recognition is to automatically beautify the annotations, as shown in Fig 2(c). While freeform inking is a convenient input medium, Bargeron reports

that document authors prefer a stylized annotation when reading through comments made by others [3].

A third goal for recognizing digital ink annotations is to make the annotations actionable. Many annotations convey desired changes to the document, such as "delete these words" or "insert this text here." The Chicago Manual of Style [4] defines a standard set of editing symbols. By automatically recognizing annotations, we can add these behaviors to the ink to further streamline the editing process.

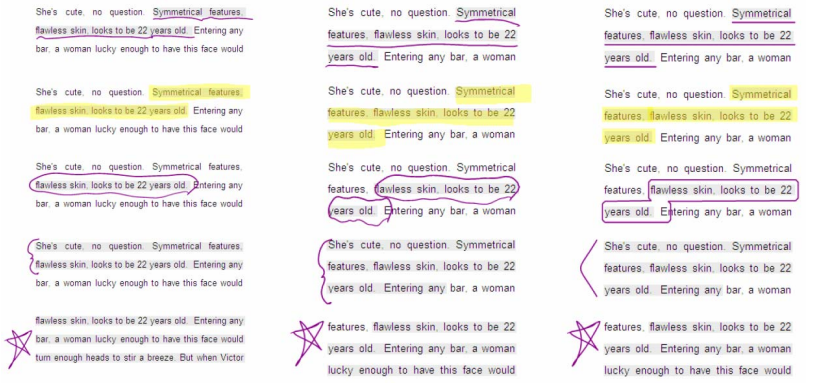

Fig. 2. Annotations reflow and cleaning. (a) Original user annotations (b) are properly reflowed as the document is edited and then (c) cleaned by the system based on its automatic interpretation.

Fulfilling these goals in a system is a broad task that incorporates many facets other than recognition. There are user interface issues such as when and how to show the recognition results and how to correct those results. There are software architecture issues such as how to properly integrate such functionality into a real text editor. There are other algorithmic issues such as how to reflow the ink strokes. However, after building a full working system we have found it useful to separate the annotation recognition as a well-encapsulated software component. In this paper we describe that component in detail, including its architecture, algorithm, and implementation.

Paper Organization

The paper is organized as follows. We begin with a problem statement: the realm of possible annotation types and document contexts is extremely large, and in the System Design section we first describe and justify the subset we have chosen to recognize. We then give an overview of the system architecture to introduce our approach. We have chosen a recognition approach in which multiple detectors offer competing hypotheses, and we resolve those hypotheses efficiently via a dynamic

programming optimization. Next, we evaluate the approach both quantitatively and qualitatively on a set of files. Finally we conclude and outline future work.

System Design

In order to support the application features described in the previous section, including reflow, beautification, and actioning, we have designed a software component to segment, classify, and anchor annotations within a document context. In this section we describe the design of that component. We have scaled back our problem to handle a fixed vocabulary of annotation types: horizontal range, vertical range, container, connector, symbol, writing, and drawing. In this section, we define each of these annotation types, define the document context that is required to perform recognition, and justify this restricted approach.

Annotation Types

While the set of all possible annotations is no doubt unbounded, certain common annotations such as underlines and highlights immediately come to mind. To define a basic set of annotations, we refer to the work of Brush and Marshall [5], which indicates that in addition to margin notes, a small set of annotations (underline / highlight / container) are predominantly used in practice. We have found that it is useful to further divide the category of margin notes into writing and drawings for the purposes of text search and reflow behavior. Thus we pose the problem of annotation recognition as the classification and anchoring of horizontal range, vertical range, container, callout connector, symbol, writing, and drawing annotations. We illustrate examples of these types with a sample annotated document shown in Fig 3.

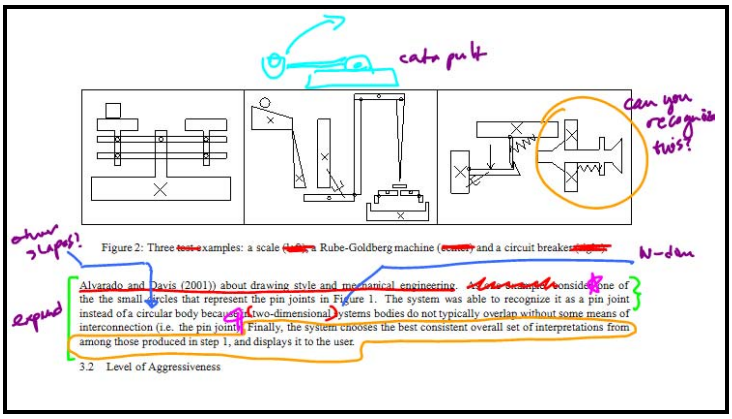

Fig. 3. Common annotation types. Horizontal range (red), vertical range (green), container (orange), callout connector (blue), symbol (magenta), and writing (purple) and drawing (cyan) marginalia.

Document Context

Annotation is a common activity across a wide variety of documents including text documents, presentation slides, spreadsheets, maps, floor plans, and even video (e.g., weathermen and sports commentators). While it is impossible to build an annotation recognizer that spans every possible document, it is desirable to abstract away the problem so that its solution can be applied to a number of common document types.

Defining this appropriate abstraction for document context is difficult: it is unlikely that any simple definition will satisfy all application needs. Nevertheless, we define a document context as a tree structure that starts at the page. The page contains zero or more text blocks and zero or more graphics objects. Text blocks contain one or more paragraphs, which contain one or more lines, which contain one or more words. Each of these regions is abstracted by its bounding box (Fig 4). At this point we do not analyze the underlying text of the document: this has not been necessary and makes our solution language-independent.

This definition of context is rich enough to support a wide variety of documents, including but not limited to, word processing documents, slide presentations, spreadsheets, and web pages.

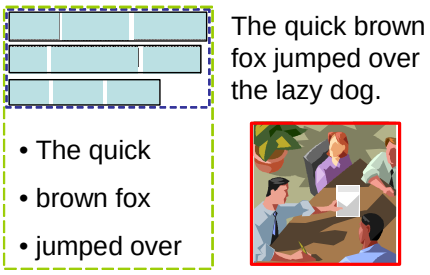

Fig. 4. Simple document context. A basic document context contains words (light blue fill), lines of text (solid black border), paragraphs (dashed blue border), blocks (dashed green border), and images/pictures/charts (solid red border).

Recognition Architecture

Given this design, we have implemented an encapsulated software component for annotation recognition. The component receives strokes and document context as its input and produces a parse tree with anchors into the document context as its output, as shown in Fig 5. This abstraction it is easy to incorporate the recognition component into different applications. So far, the annotation recognizer has been deployed in the Callisto plug-in to the web browser Microsoft Internet Explorer [3].

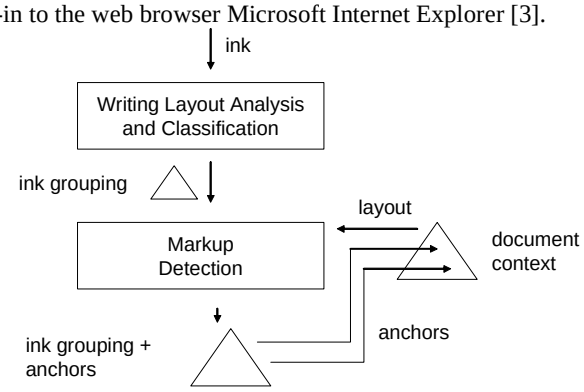

Fig. 5. High-level annotations recognition architecture. A first step separates writing and drawing strokes and groups writing into words, lines, and paragraphs. A second step analyzes ink relative to a document context, classifies markup elements, and anchors the annotations to the document context.

The recognition component itself consists of several stages, as shown in Fig 5. Initially, strokes are run through a component for handwriting layout analysis and classification that groups separates writing strokes and drawing strokes and groups writing strokes into words, lines, and paragraphs, as described in [6]. This stage

produces an initial structural interpretation of the ink without considering the underlying document context.

Once the strokes have been divided into writing and drawing, a markup detection stage looks for common annotation markup (horizontal range, vertical range, container, connector, and symbol) relative to the abstraction of the document context, it produces a revised structural interpretation of the ink, and links the structures to elements in the document context abstraction. We describe the markup detection in the following section.

Annotation Recognition

Markup detection segments and classifies ink into a set of annotation types including horizontal range, vertical range, container, and connector.

One possible approach to markup detection would be to generate all possible combinations of strokes and classify each with respect to the different classes, maximizing some utility or likelihood over all hypotheses. This approach suffers from several practical problems. First, it is combinatorial–even generic spatial pruning heuristics may not be enough to make the system run in real-time. Second, it relies on enough data to train a reasonable classifier and garbage model.

Since we wanted to generate an efficient system that could keep up with user annotation in real-time and did not have large quantities of training data available, we opted for a more flexible solution. Our markup detection is implemented as a set of detectors. Each detector is responsible for identifying and anchoring a particular annotation type among the ink strokes on the page, and uses a technique specific to its annotation type in order to prune the search space over possible groups. When a detector identifies a candidate for a particular annotation type, it adds the resulting hypotheses with an associated confidence to a hypothesis map, as shown in Fig 7. For example, in Fig 7(c), a connector detector hypothesizes that strokes could be connectors on their own (both are relatively straight and have plausible anchors at each of their endpoints, or that they could together form a single connector. We say that a pair of hypotheses *conflict* if they share any of the same strokes.

Detectors

Each annotation type has a set of characteristic features that allow it to be distinguished from other annotations and from random strokes on the page. These features can be divided into two categories: stroke features and context features.

Stroke features capture the similarity between a set of ink strokes and an idealized version of an annotation. For example, the idealized version of an underline is a straight line, so the stroke features measure the distance between a set of strokes that might be an underline and the best straight line that approximates those strokes, i.e. the total regression error on the points in those strokes.

Context features capture the similarity of the best idealized version of a set of strokes and a true annotation on the document context. For example, a stroke might be

a perfect straight line, but it is not an underline unless that line falls beneath a set of words in the document.

Thus the procedure for each detector is to ascertain a best idealized version of the strokes according to its type using stroke features, and then see how well that idealized version fits with the document context using context features. The stroke and context features for each of the annotation types we support are described in Fig 6.

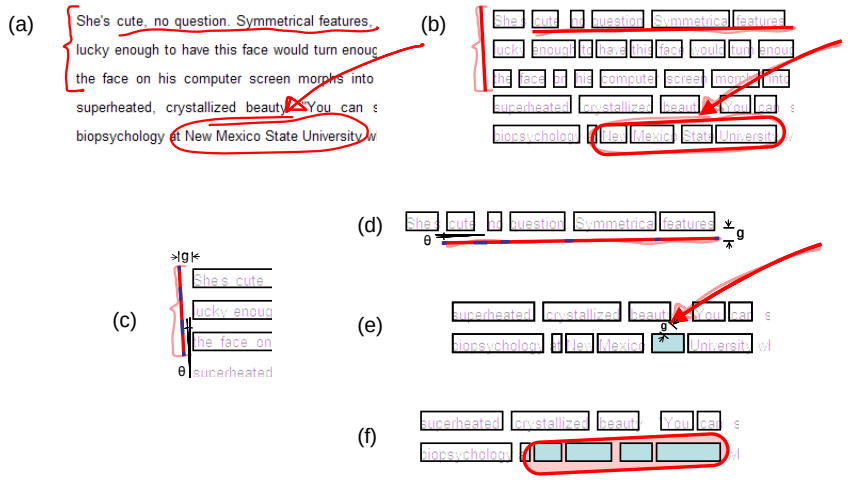

Fig. 6. Detector features. (a) The original ink annotations on the document. (b) The idealized annotations overlayed on the ink annotations, and the document context bounding boxes. (c) Vertical range context features include θ, the angle between the ideal and the lines of text, g, the gap between the ideal and the lines, as well as the sum of the lengths of the overlapping portions of the ideal (in red) and sum of the lengths of the non-overlapping regions (in blue). (d) Horizontal range context features include θ, the angle between the ideal and the lines of text, g, the gap between the ideal and the lines, as well as the sum of the lengths of the overlapping portions of the ideal (in red) and sum of the lengths of the non-overlapping regions (in blue). (e) Callout context features include g, the distance of the arrowhead to a context word along the tangent of the tip of the arrow. (f) Container context features include the area overlapping with the context words (light blue) and the non-overlapping area with the context words (pink).

Resolution

Once all of the detectors have executed, the most likely annotations are extracted from the map through a resolution process and the result is committed to the output, as shown in Fig 7(d). The resolution is designed to pick the best candidates when there are conflicting hypotheses. It is a unifying framework by which detectors can be added modularly to support new annotation types.

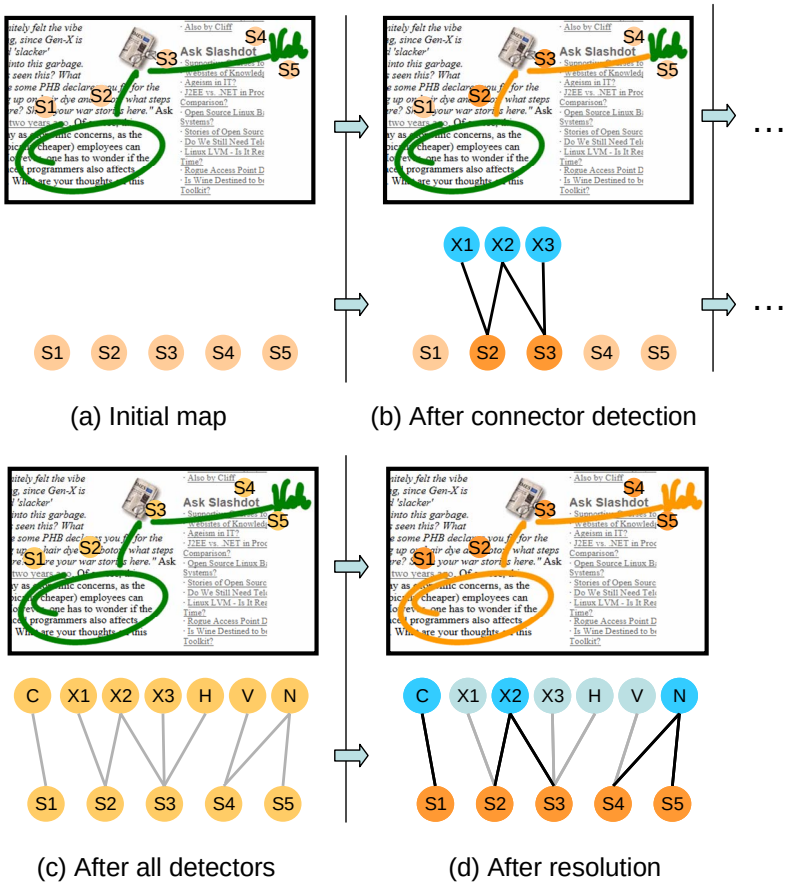

Fig. 7. Hypothesis framework. (a) Initially map is empty. (b) Connector detection inputs three conflicting hypotheses (X1, X2, X3). (c) The rest of the detectors execute, adding container (C), horizontal range (H), vertical range (V), and margin notes (N) to the map. (d) Resolution selects the most likely hypotheses (C, X2, and N).

Resolution is designed to maximize number of explained strokes, maximize the overall confidence, and minimize the number of hypotheses. This can be expressed as the maximization of an energy function:

$$E = \sum_i \text{confidence}_i + \alpha \left|\text{explained strokes}\right| - \beta \left|\text{hypotheses}\right| \quad \textbf{(1)}$$

In Equation 1, α and β are empirically-determined weights. We maximize this function exactly using dynamic programming. Since there is no special ordering of the strokes, we can impose one arbitrarily and solve using the following recurrence relation:

$$E(S) = \begin{cases} 0 & \text{if } S \text{ is empty} \\ \max_{S'}(C(S') + E(S - S') - \beta) & \text{otherwise} \end{cases} \quad (2)$$

In Equation 2, S represents a subset of strokes on the page, S' is a hypothesis containing the stroke in S with *minimum* ID, or no explanation for that stroke, and C is the confidence of that explanation plus α times the strokes it explains, or 0 if the minimum stroke is left unexplained.

Evaluation

Our evaluation goals were two-fold. First, we wanted to understand the accuracy of the complete system. Second, we wanted to understand the effectiveness of the resolution process. We thus measured the accuracy of each of the detectors, and compared those numbers with the final system accuracy.

Our test set consisted of ~100 heavily annotated web pages containing 229 underlines, 250 strikethroughs, 422 containers, 255 callouts and 36 vertical ranges. To simplify accounting, we unify grouping errors and labeling errors into one unit. In other words, an annotation is correct if it is grouped and labeled properly, otherwise it results in a false negative and possibly multiple false positives.

Table 1. Results from running the individual detectors prior to resolution.

	Correct	False positive	False negative
Underline	219	183	10
Strikethrough	244	99	6
Blob	400	6	22
Callout	206	529	49
Margin bar	35	219	1

Table 2. System results after resolution incluing percentage changes from the data in Table 1. Percentages are obtained by $N_{final} - N_{inital} / N_{true}$.

	Correct	False positive	False negative
Underline	206 (-5.7%)	24 (-69.4%)	16(+2.6%)
Strikethrough	229 (-6%)	35 (-25.6%)	9(+1.2%)
Blob	396 (-0.9%)	6 (0%)	25(+0.7%)
Callout	177 (-11.3%)	31 (-195%)	77(+11%)
Margin bar	35 (0%)	140 (-225%)	1(0%)

These results show that the system has reasonably high accuracy despite the inherent ambiguity in the problem, our small quantities of training data, and the compromises we have made in choosing our techniques such that the system could operate in real-time. Pre-resolution our detectors performed adequately for most classes except for callout – we have not been able to identify good features for callouts since most

strokes on the page have objects at one endpoint or the other (callouts with arrowheads are substantially easier to detect). We hope to learn useful features once we have collected a larger data set. The results further show that resolution significantly decreases the number of false positives without substantial change to the false negatives. This indicates that it is a reasonable strategy for this problem.

Conclusions and Future Work

We have presented an approach to recognizing freeform digital ink annotations on electronic documents, along with a practical implementation. The resulting recognizer facilitates all of the operations common to traditional digital annotations, but through the natural and transparent medium of digital ink.

Rather than constraining the user, we employ an extensible framework for annotation recognition which achieves high accuracy even for complex documents. Our work approximates an exhaustive search of possible segmentations and classifications. This makes it possible to analyze a full page of ink in real-time, and can be applied to many other ink recognition problems.

We have implemented our approach in a reusable software component, have integrated the component into a full system for annotating web pages within Microsoft Internet Explorer, and have evaluated its accuracy over a collection of annotated web pages.

However, there are many ways we hope to extend this work both from an analysis standpoint and from a system standpoint. From an analysis standpoint, we have made numerous compromises both for efficiency and due to sparse amounts of labeled data. We are currently collecting and labeling annotation data and hope to explore fast data-driven alternatives our current heuristics for detection and resolution.

From a system standpoint, we have yet to corroborate our intuitions with user studies of the full system. In addition, many of the structures that we recognize, such as boxes and connectors, are also common to other types of sketching such as flow charts and engineering diagrams. Our efficient inference algorithm should also extend to these domains. Furthermore, it should be possible for users to customize the system with their own annotation styles if they are not supported by our basic set. Finally, we are interested in examining other creative ways in which annotations can be used once they are robustly recognized.

Acknowledgments

We gratefully acknowledge David Bargeron, Bert Keely, Paul Viola, Patrice Simard, Sashi Raghupathy, and David Jones for brainstorming and collaboration on this work.

References

1. W. Schilit, G. Golovchinsky, and M. Price. “Beyond Paper: Supporting Active Reading with Free Form Digital Ink Annotations.” Proc. of ACM CHI 1998. ACM Press. pp. 249-256.
2. G. Golovchinsky, L. Denoue, “Moving Markup: Repositioning Freeform Annotations,” Proc. of ACM UIST 2002. ACM Press, pp. 21-30.
3. D. Bargeron, T. Moscovich. “Reflowing Digital Ink Annotations.” Proc. of CHI 2003, ACM Press, pp. 385-393.
4. University of Chicago Press. The Chicago Manual of Style. The University of Chicago Press, Chicago, IL, USA, 13th edition, 1982.
5. C. Marshall and A. Brush. “From Personal to Shared Annotations.” In Proc. of CHI 2002, ACM Press, pp. 812-813.
6. M. Shilman, Z. Wei, S. Raghupathy, P. Simard, D. Jones. “Discerning Structure from Freeform Handwritten Notes.” Proc of ICDAR 2003, pp. 60-65.