Discerning Structure from Freeform Handwritten Notes

Michael Shilman[*], Zile Wei[*], Sashi Raghupathy, Patrice Simard, David Jones
Microsoft Corporation
{michaels, zile}@eecs.berkeley.edu, {sashir, patrice, davejo}@microsoft.com

Abstract

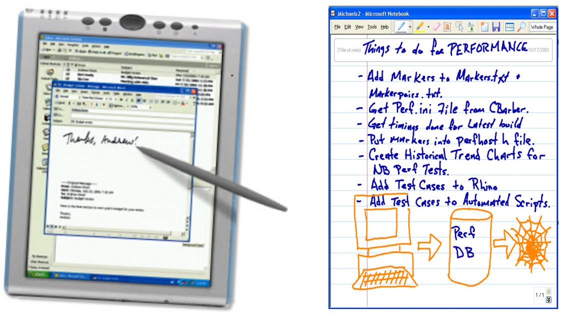
This paper presents an integrated approach to parsing textual structure in freeform handwritten notes. Text-graphics classification and text layout analysis are classical problems in printed document analysis, but the irregularity in handwriting and content in freeform notes reveals limitations in existing approaches. We advocate an integrated technique that solves the layout analysis and classification problems simultaneously: the problems are so tightly coupled that it is not possible to solve one without the other for real user notes. We tune and evaluate our approach on a large corpus of unscripted user files and reflect on the difficult recognition scenarios that we have encountered in practice.

1. Introduction

While computers have all but replaced pen and paper for the majority of document creation tasks, note taking and annotation are areas for which pen and paper are still the preferred medium [1]. Yet there are numerous advantages to note-taking and document annotation in the digital domain, including, but not limited to, better search, archival, editing, and information sharing [2].

There are many approaches to digital note taking, ranging from typing on laptop to scribbling on a handheld personal digital assistant or writing on paper with a wireless or optical capture device. Our platform is the Tablet PC, a new slate-like computer using the Windows XP operating system and driven by a pen. In the context of note-taking, the Microsoft Windows Journal application focuses on immediate thought capture in rich digital ink by emulating a sheet of paper. However, our experience with Journal has been that nearly all of the downstream added value of digital ink requires some degree of ink understanding. This poses interesting technical challenges.

We have isolated two essential problems: (1) distinguishing handwritten text from ink markup and drawings, and (2) understanding the structure of writing, including words, lines, and paragraphs. Solving these problems enables value-added features such as smart facilities for insertion, deletion, formatting (e.g. reflow), search, and repurposing ink into existing applications.



Things to do for PERFORMANCE

- Add Markers to Markers.txt + Markerpairs.txt.
- Get Perf.ini File from CBarber.
- Get timings done for Latest build
- Put markers into perfhost h file.
- Create Historical Trend Charts for NB Perf Tests.
- Add Test Cases to Rhino
- Add Test Cases to Automated Scripts.

Figure 1. A Tablet PC and a page of real digital handwritten notes taken on the device.

We have considered many alternative solutions to these basic problems, employing one or more of the following approaches:

1. Users can switch modes (drawing, new paragraph, etc.) dynamically as they write (multiple pens, buttons on pens, application modes, etc.)
2. Users can attribute ink after the fact to tell the application how to interpret it.
3. Different regions on the page interpret ink in different ways.
4. The application can parse the ink automatically to infer its attributes.

After extensive field trials and user studies conducted at Microsoft, we have concluded that for note-taking, **real users generally do not want to be told where to write, do not want to switch modes, and do not want to be distracted from their thought capture by any additional cognitive load.** For example, even though a "perfect" eraser tool is one of the nice features of digital ink, many users will scribble out their mistakes rather than go to the trouble of changing modes. Yet users also use and appreciate the value-added features of digital ink for note-taking such as search and repurposing. Thus we have chosen an automatic parsing approach (4) to ink understanding in freeform notes.

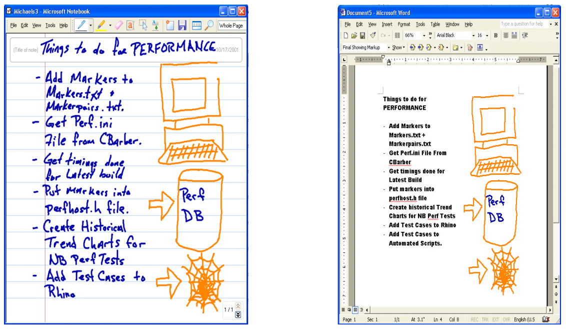
Writing-drawing classification and handwriting layout analysis are not new; they are classical problems for printed document analysis.

[*] Currently at UC Berkeley Department of EECS

Writing-Drawing Classification. Given a page of strokes, [1] label each stroke as writing or drawing. This is analogous to text-graphics classification on connected components in printed document analysis [3, 4].

Handwriting Layout Analysis. Given a page of writing strokes, correctly group the strokes into words, lines, paragraphs, and regions. This is analogous to text layout analysis in printed document analysis [3, 4].

Below we see the results of classification and layout analysis performed on the document from Figure 1 to provide intelligent reflow (left) and smart repurposing of ink into a text document (right):


Things to do for PERFORMANCE

- Add Markers to Markers.txt + Markerpairs.txt
- Get Perf.ini File From CBarber
- Get timings done for Latest Build
- Put markers into perfhost.h file
- Create historical Trend Charts for NB Perf Tests
- Add Test Cases to Rhino
- Add Test Cases to Automated Scripts.

Figure 2. The note from figure 1 reflowed (left) and then repurposed into a text editor (right).

Related work. For online handwritten documents, a much smaller body of work exists. Several attempts have been made at identifying lines of handwriting using bottom-up [5] and Hough transform methods. Namboodiri et al have built a system for text, graphics, and table identification in online documents that most closely resembles this work [6]. Their system first partitions the strokes into text and graphics using the features of stroke length and curvature. It then groups writing strokes into lines assuming a horizontal baseline and groups drawing strokes using a minimum spanning tree-based clustering method.

In our experience it is relatively easy to build a system that will do a flawless job on a selected few “demo” documents or perform well with unrealistic constraints, such as restricting writing to cursive between lines on the page or restricting drawings to rectilinear flow-charts. Real-world notes are unwieldy: most of our unscripted samples contain mixed cursive and printed writing, often written at angles, in various sizes, and at various locations all over the page. Drawings in notes can range from tables and standard diagram types to drawings of horses.

[1] We define a stroke as a sequence of strokes between pen down and pen up.

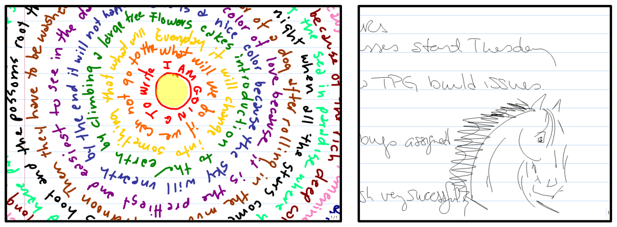


Figure 3. Freeform notes can be unwieldy.

These constraints do not change the problem statement but require a flexible solution that makes few assumptions about the content of the notes. In this paper, we describe a full-system approach to parsing freeform handwritten notes. We begin with a detailed description of our ink parsing approach, which primarily consists of a layout analysis and classification steps. We follow this with a quantitative evaluation and tuning methodology. Finally, we summarize our conclusions and list future work.

2. Parsing Algorithm

Although writing-drawing classification and handwriting layout analysis are posed as two distinct problems, we believe in an integrated solution. The writing-drawing classification problem is fundamentally ambiguous: is a lone circle on the page writing or drawing? **Real notes consist of numerous cases that can only be disambiguated with intelligent use of context.** Because we do not wish to place any constraints on the type of drawings that a note might contain, the only context we can utilize is the structure inherent in handwritten text. Thus, under these assumptions, writing-drawing classification is only possible with sophisticated handwriting layout analysis.

Our parsing approach consists of a layout analysis step and a classification step. Our layout analysis assumes that all strokes are text, combining strokes of similar size and orientation to form words, lines, and blocks (paragraphs), in a bottom-up order. Once a tentative layout has been computed, we then classify the individual strokes into text and drawing, based on local and global attributes of the strokes, words, lines, and blocks. In this approach we make the implicit assumption that writing and drawing strokes will not be combined by the layout analysis step. Our initial results show that this is a reasonable assumption, with a few common exceptions such as arrowheads that are of similar size and in close proximity to the text that they point to.

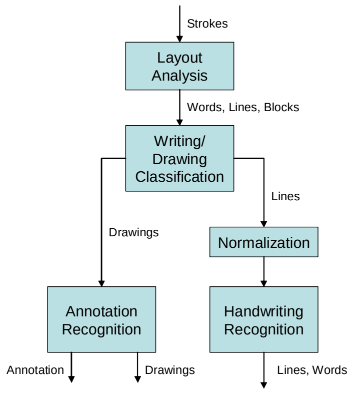


Figure 4. Full system architecture.

The figure above shows our full algorithm flow. In addition to layout analysis and classification, we also detect a few types of annotation, as well as normalize the ink for our handwriting recognizer by rotating angled lines of text to horizontal and retiming “late” strokes, i.e. those that are added to words out of time order. In this section we focus on layout analysis and classification, which are the most interesting and crucial components of the system.

2.1. Layout Analysis

The core of our approach is a multi-pass, bottom-up layout analysis of the ink on the page. The layout analysis algorithm consists of a succession of decisions (which strokes belong to the same word, which words belong to the same line, etc.) based on layout and robust statistics. These decisions yield a hierarchical clustering of the ink strokes on the page, which allows us to calculate global statistics over each cluster. The first decisions are conservative and are based on good local layout relationships when the clusters are small. The later decisions can be more aggressive thanks to global robust statistics (median sizes, spacing, angles, etc.) collected over larger clusters. Multiple passes enable increasingly aggressive decision making.

Our first three passes incorporate temporal information and the later passes are purely spatial. The temporal grouping only tries to merge regions that are ordered consecutively in time, where we define a region’s timestamp to be the minimum timestamp of the strokes that it contains. Although users do not write their notes in perfect time order, we have found time order is a good bootstrapping heuristic that gives us enough information to do an accurate spatial grouping in subsequent phases. Our initial attempts at a purely spatial solution were unsuccessful because in handwriting the pair-wise noise of stroke size and inter-stroke distance is too great for a greedy approach and we were unable to find a non-greedy approach that would operate in real-time to keep up with the user.

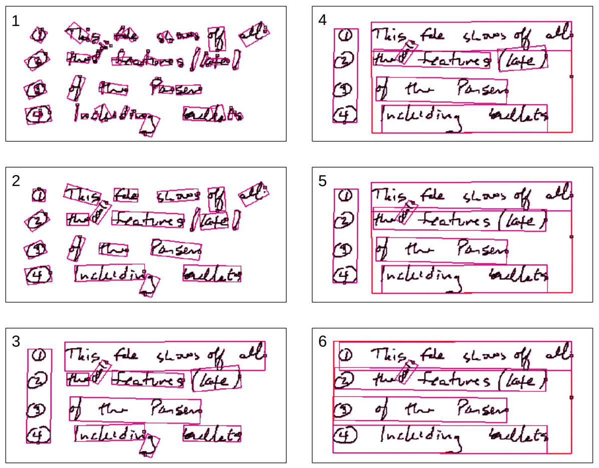


Figure 5. Layout analysis is a bottom-up algorithm that uses robust statistics.

For simplicity’s sake, the input of layout analysis (LA) is a tree of regions in which every stroke is a word, every word is a line, and every line is a paragraph. LA operates greedily, so during each pass merge operations occur, but splits do not. The output of LA is another tree, which corresponds to the perceived layout of the page. Figure 5 shows lines and paragraphs at different stages of the algorithm visualized as bounding boxes.

Stroke Fragments. To regularize cursive and printed writing, we break strokes into fragments, which are determined with hysteresis at local maxima and minima according to the current baseline hypothesis for the line containing the stroke. Useful features such as fragment width and height, as well as the collinearity of fragment centroids with their line hypothesis are useful in both layout analysis and classification. One disadvantage to this scheme is that stroke fragments must be updated as the baseline hypothesis changes.

2.1.1. Temporal Grouping

DP Line Grouping (DPLG). The first phase of LA groups lines in temporal order using a dynamic programming optimization (Figure 5.2). The cost function in dynamic programming reflects the confidence that a given set of strokes is in one word. We take the regression error of fragments in strokes and the ratio of maximum fragment distance in this set (inner distance) and minimum fragment distance to adjacent fragments (outer distance). The first feature is to make sure that strokes actually share a common baseline. The second focuses on grouping strokes in one word, since inner distance in one word should be relatively smaller than outer distance.

The cost for a grouping of strokes from i to j is written:

$$d(i, j) = \max(w(i, j), \max_k(d(i,k) + d(k+1, j)))$$

where $w(i, j)$ describes the confidence that strokes from i to j are in one word.

DPLG is a conservative process. Even if a word is written exactly in temporal order, crossed “t’s” and dotted “i’s” could break the word into two pieces, for inner distance in this word could become larger than the outer. But this is necessary in our greedy LA framework so that there is no over-merging in any step.

Temporal Line Grouping (TLG). After dynamic programming, we run two passes of a line clustering heuristic (Figure 5.3). The first pass is extremely conservative, approximating a word grouping algorithm at the line level. This bootstrapping step gives us enough information about the orientation of lines to do a better job in the second pass, which is more aggressive and tries to get full lines.

```
void TemporalLineClustering(ParseTree pt) {
  //sort lines in time and by length
  Line[] linesByLen = pt.LinesByLength();
  Line[] linesByTime = pt.LinesByTime();

  // iterate through lines by
  // decreasing confidence
  foreach(Line line in linesByLen) {
  // foreach line, scan forward in time
    for(int j = linesByTime.IndexOf(line);
        j < linesByTime.Size(); j++) {

      if(SameLine(line, linesByTime[j]))
        pt.Merge(line, linesByTime[j]);
    else //stop if we can’t merge
        break;
    }
    //do the same backwards in time
  }
}
```

The merge tests in this heuristic are pair-wise tests between lines. The exact test varies depending on our baseline confidence for the lines passed into it. Baseline confidence is a function of the number of fragments in the line and the regression error of the fragment centroids to the estimated baseline. For higher confidence lines (“long”), the statistics are more robust, so we can be more aggressive. For lower confidence lines (“short”), we use a conservative test based on the convex distance between the two lines. The features we use in these tests are summarized here:

	Short	Long
Short	Convex Distance Width Ratio Height Ratio	Width, Height Ratio Horiz, Vert Gap Change in Regression Error
Long	Same as LS	LS + Angle Diff

Because the algorithm proceeds from long to short, we find that the SS test is not very common in the second pass; the long lines tend to “swallow” the shorter ones in LS tests.

2.1.2. Spatial Grouping

Spatial Block Grouping. Given an initial temporal line grouping, we proceed to cluster lines into blocks (Figure 5.4). Block grouping is purely spatial and groups blocks whose lines are of roughly the same orientation and fragment size. The tests that we employ in block grouping are vertical gap and horizontal overlap, in addition to the “font size” features defined in the previous section.

As in TLG, we start with the longest lines first, for more robust size and orientation statistics, and progress down to shorter lines. For each line on the page, we scan all the other lines in the page. Although we could implement a more optimal spatial ordering, we find the $O(n^2)$ algorithm satisfactory for the number of lines on our pages (< 200).

Spatial Line Grouping (SLG). Spatial line grouping operates in much the same way as TLG, but time order is not leveraged (Figure 5.5). As in TLG, we work from high-to-low confidence. As in SBG, we scan over all the lines in the page. SLG’s merge tests are mostly the same, only the criteria are a bit tighter, since we cannot rely on time order. In some cases, we can utilize the block information to be more aggressive. For instance, if two lines are in the same block, only the vertical separation is important for determining whether they are the same line. We can also use robust statistics over the whole block instead of just the line.

List Correction. As in [3], we see many cases in practice in which lists are misclassified using a pure line-finding approach. We correct this with a simple heuristic. This is the only non-greedy part of our algorithm.

2.2. Classification

After Layout Analysis, we classify strokes on the page into writing and drawing based on local and global features. Ideally, a handwriting recognizer would be able to deal with a mixture of text and graphics, providing a “writing/drawing confidence” measure based on its character and language models. Unfortunately, most handwriting recognizers assume their input is text and only provide confidences of one text hypothesis relative to another.

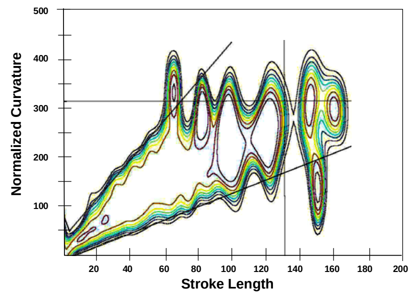


Figure 6. Local features for training set.

Local Features. There is plenty of information available in the local features or a stroke itself to determine whether a stroke is writing of drawing. For instance, even the simple features of stroke length and curvature are enough to reliably separate cursive writing from most block diagram elements [6, 7]. However, real writing isn't always cursive, and real drawings aren't always block diagrams. Therefore, our parser relies heavily on the context provided by Layout Analysis to classify ink strokes as either writing or drawings.

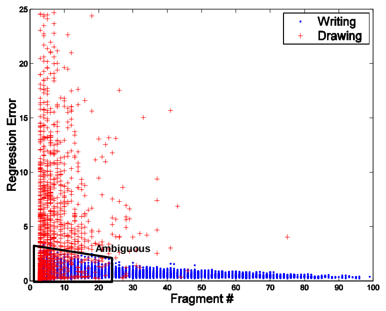


Figure 7. Global features for training set.

Global Features. The intuition behind our classification algorithm is that if you look at a page of ink with squinted eyes or from a distance, you can distinguish writing from drawing by its regular, linear structure. This intuition turns into the assumption that ink with a regular, linear structure reminiscent of text is text. So if the user draws a set of scribbles that might be perceived as text if you squinted your eyes, but is actually nonsense if you try to read it, chances are good that our parser will make the same mistake. This limitation is also an advantage—preliminary experiments suggest that our parser works even on documents containing a mixture of English and Far East text.

We classify each line as either writing or drawing based on fragment count and regression error over the centroids of all fragments in strokes. For writing strokes, fragment number could range up to 100 and regression errors below 3. But most drawing strokes yield fewer fragments and much higher regression error. We employ local features to differentiate strokes in the ambiguous region near the origin of the global features plot.

Classification. Given these strong features, we believe that most standard classification algorithms will suffice. For instance, we initially used a support vector machine (SVM) with radial basis function to perform the classification. Then, as a performance optimization, we constructed a linear decision tree to perform the same task. Although the SVM is a more powerful classification mechanism than the decision tree, we obtained similar results due to the quality of the features.

3. Tuning and Evaluation

Ink parsing for the task of note-taking is only useful if it can work well on typical documents in most note-taking scenarios. Thus data collection must be realistic: algorithms must be tuned and evaluated on a wide range of unscripted files. We have gathered and labeled hundreds of pages of unscripted notes from end users on Tablet PC's and used these pages to evaluate our algorithm's accuracy both quantitatively and qualitatively.

3.1. Quantitative Metrics

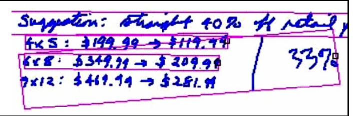
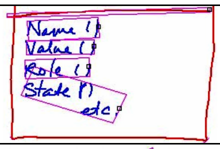
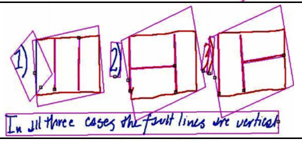
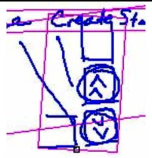
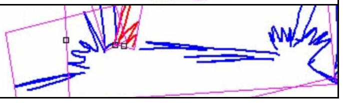
We evaluate our approach according to two metrics. To measure the accuracy of writing-drawing classification, a simple error percentage suffices. The output of layout analysis is a tree, so we measure the similarity of the output tree to the correct tree with the number of splits and merges at each level of hierarchy to transform the result tree into the truth tree.

The following results were obtained by running the two metrics described above over a collection of 520 labeled files from our sample data. The files were chosen to represent key user scenarios (biased towards text) and contain writing in multiple languages and orientations, assorted block diagrams and flow charts, and free-hand sketches. Our classification accuracy was 94.1% over 513,592 total strokes. Our layout analysis achieved the following results:

	Word	**Line**	**Block**
Total #	112,460	33,384	19,236
Split	2811 (2.5%)	267 (.8%)	173 (.9%)
Merge	4498 (4.0%)	67 (.2%)	77 (.4%)

3.2. Qualitative Analysis

The quantitative results provide little insight into the actual algorithm behavior or its impact on the end user. Some common errors include the following:

Error case	**Example**
Text grouped across perceptual boundaries in the page.	
Consecutive short lines over-grouped	
Short drawing lines grouped with text (1) or short text lines classified as graphics (3)	
Regularly spaced and sized graphics strokes grouped and misclassified as text.	
Jaggy drawing stroke misclassified as writing.	

We believe that many of these errors can be improved with a more sophisticated feature set than the one we use, perhaps incorporating features at the line and block level as well as texture features on regions of the page. We are continuing to experiment on our document corpus.

4. Conclusion

In this paper we provide a problem statement for sketch understanding of freeform hand-sketched notes based on previous work in Document Image Analysis. Our integrated ink parsing approach makes very few assumptions about the contents of notes and is thus robust to a wide variety of note-taking styles, scenarios, and even languages. Next, we advocate a particular evaluation methodology and report our initial results. Finally, we describe the user interface and architectural considerations in building a note-taking system that successfully incorporates this parsing functionality.

Future Work. Moving forward, we hope to improve our algorithm with even more aggressive use of context, by recognizing and leveraging perceptual and semantic structures in the document. For example, users often draw lines of ink to separate regions of text, but these cues are not utilized by our current layout analysis algorithm. We are also early in the process of doing user studies on the system and hope to correlate our usability findings with the quantitative results to determine a unified quality metric through controlled experiments. Finally, we would like to extend our system to parse common structures in notes, such as tables and flow charts.

Acknowledgments

We gratefully acknowledge Jian Wang, Zou Yu, Herry Sutanto, Jamie Wakeam, and the Tablet PC team for their contributions. We also thank Bert Keely, Chuck Thacker, and Greg Hullender for stimulating discussions along the way.

References

[1] A. Sellen and R. Harper. Paper as an Analytic Resource for the Design of New Technologies. CHI'97 Conference Proceedings, Atlanta, GA, 1997.

[2] R. Davis, J. Landay, V. Chen, J. Huang, R. Lee, F. Li, J. Lin, C. Morrey III, B. Schleimer, M. Price, and B. Schilit, NotePals: Lightweight Note Sharing by the Group, for the Group. CHI 99, Pittsburgh, PA, 1999.

[3] L. Fletcher and R. Kasturi. A Robust Algorithm for Text String Separation from Mixed Text/Graphics Images. IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol. 10, #6, November, 1988.

[4] H. S. Baird, Anatomy of a Versatile Page Reader, *IEEE Proceedings*, July, 1992.

[5] E. Ratzlaff. Inter-line Distance Estimation and Text Line Extraction for Unconstrained Online Handwriting. *Workshop on Frontiers in Handwriting Recognition*, September 11-13-2000, pp. 33-42.

[6] A. Jain, A. Namboodiri. and J. Subrahmonia. Structure in On-line Documents. In *Proceedings of the 6th Internatoinal Conference on Document Analysis and Recognition (ICDAR '01).*

[7] K. Machii, H. Fukushima, M. Nakagawa. On-line text/drawings segmentation of handwritten patterns. *Document Analysis and Recognition*, 1993.